UE VIDEOVERARBEITUNG 2017W

EXERCISE 1

GENERAL

The following filters have to be implemented for the first exercise:

1) **filter_remove_color.m**
    a. **Black/White**
    b. **Sepia**
2) **filter_unsharp.m**
3) **filter_rand_illumination.m**
4) **filter_high_contrast.m**
5) **filter_iris.m**
6) **filter_low_framerate.m**

The filter functions have to be implemented in the above-mentioned files. All filters have a different amount of parameters, which control the properties of the filters. You can expand the number of parameters for every filter but you should be able to justify each new parameter. During the implementation of each filter, meaningful parameter values that result in authentic looking results have to be found.

All the files which are needed for implementing each filter (filter_*.m) are in the /src directory. By calling the function exercise1.m all implemented filters are applied on an image sequence.

Example 1: **exercise1('../images/', '../output/');**
applies all filters to the frames frameXXXX.png in the directory '../images' and stores the resulting images in the directory '../output'.

Example 2: **exercise1('../images/', '../output/', start_frame, end_frame);**
processes only frames 'start_frame' to 'end_frame'.

Example 3: **exercise1({'../images/','../images/'}, '../output/');**
reads all images from 2 input directories '../images1' and '../images2' (in this order) and applies the filters to all images.

Example 4: **help exercise1**
displays help text for each file in the command window (in this example for the file exercise1).

SUBMISSION

The deadline for exercise1 is **06.11.2017**.

The submission must include:

- Commented Matlab source files of all filters.
- Following fields have to be completed at the beginning of each filter:
    - Short description of the implementation.
    - Short summary of the physical background/reason of the effect.
      For example: What causes Sepia discoloration?
    - Meaningful parameter values.
- One picture of every filter that demonstrates the operation of the filter.

You will get points for

- Your implementation
- The description of your filters
- The mention of meaningful parameter values

All files have to be uploaded as a .zip file (UE_GROUPx_EXERCISEy.zip) on TUWEL. Only one submission is needed per group.

NOTE: Check your submission for completeness because points will only be given to the submitted files!

GENERAL INFORMATION

DATA CONVERSION

You can use the function convert_video_to_images to extract frames from a video of your choice and save them as .png files. Be warned that this process might lead to a large consumption of disk space if the movie is very long.

Furthermore, the function convert_images_to_video does the opposite and combines all images from a directory into a movie. This can be used to create the final movie after the filters were applied.

IMPLEMENTATION OF THE FILTERS

All exercises are handed out with a Matlab framework. In this exercise the framework consists of the file exercise1.m. The framework automatically takes care of reading, saving and displaying the images. To each filter in this exercise, the struct video will be passed as the first argument. It has the following structure:

video.original	Original image in RGB
video.filtered	Filtered image in RGB
video.original_frame_nr	Frame number of this frame in the input directory (is not modified by any buffer!)
video.frame_nr	Frame number of this frame (can change if frames are inserted/removed by a filter)

When implementing a filter the following has to be taken into account:

1) Each filter works on the field video.filtered and saves the result there again. In this way, all filters can work sequentially on the same image.
2) To provide access to previous frames a frame buffer is installed. Thus, two previous frames can be addressed. You can access the current frame over video.frame(1).filtered, the previous one over video.frame(2).filtered.

video.frame(1).filtered	current frame i
video.frame(2).filtered	frame i-1
video.frame(3).filtered	frame i-2

3) You can perform any conversion of the input information. However, EVERY filter MUST save its results in the RGB format (RGB values between 0 and 255). Thus, no distinction and multiple implementations for different image and color formats are needed and the filters can be executed in an arbitrary order.

Example:

```
function filter_test(video)
  img = video.frame(1).filtered;
  …
  video.frame(1).filtered = img;
end
```

In order to pass information from one filter to another, the struct video will be expanded in the course of this lecture. Temporary filters, which operate on multiple images, require knowledge about the status of the filtering (What has been done so far? What has to be done in this step?).

HINT 1:

You can insert any additional fields in the struct video if you need them for the proper working of the filters.

HINT 2:

Matlab manages the images in a transposed way, which means that the image point (x,y) corresponds to the value (y,x) in the image structure. Each image in video.filtered and video.original has the format [H x W x C], where H represents the image height, W the image width and C the color depth. For the RGB color space C = 3 (R/B/G).

Here are some examples showing how to access the color information of the images:

Example 1:

```
r_info = video.filtered(4,3,1);        % x = 3, y = 4, c = 1
r_info = 34                            % red color information at the position (3,4)
```

Example 2:

```
rgb_info = video.filtered(4,3,1:3)     % x = 3, y = 3, c = 1:3
rgb_info = [34 12 120]                 % RGB values at the position (3,4)
```

Example 3:

```
r_info = video.filtered(1:10,1:10,1)   % x = 1:10, y = 1:10, c = 1
r_info = 10 x 10 matrix                % red color information at the positions (x = 1:10, y
                                                                                   1:10)
```

HINT 3:

Try all the filters in an arbitrary order to verify correct functioning.

Useful MATLAB links:

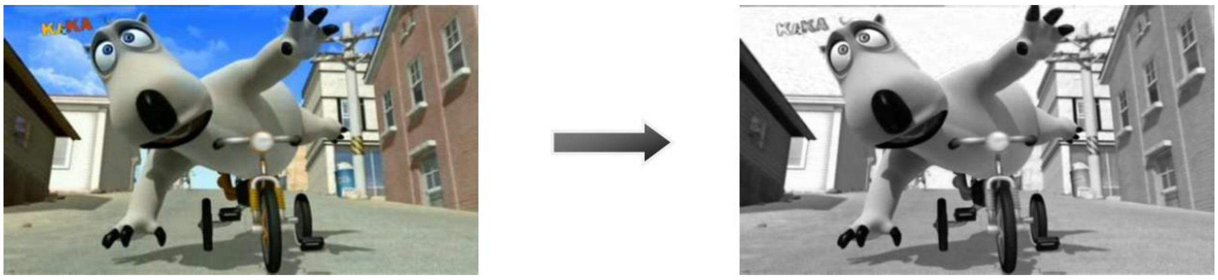
https://de.mathworks.com/help/matlab/ref/videoreader.html

https://de.mathworks.com/help/matlab/ref/videowriter.html

FILTER 1A: FILTER_REMOVE_COLOR

GENERAL INFORMATION

This filter has to be implemented in the function filter_remove_color(video, ‘bw’). It removes the color information from an RGB image and provides an RGB image that contains only shades of gray.



IMPLEMENTATION

For this task, the following functions have to be implemented:

Function:	black_white in filter_remove_color
Input:	RGB frame
Output:	grayscale image stored as RGB frames
File:	filter_remove_color.m

1) Convert the image into the HSV color space.
2) Modify the H/S/V information. What needs to be removed in this color space to eliminate the color information?
3) Convert the image back into the RGB color space.

USEFUL MATLAB COMMANDS

rgb2hsv, hsv2rgb

REFERENCES

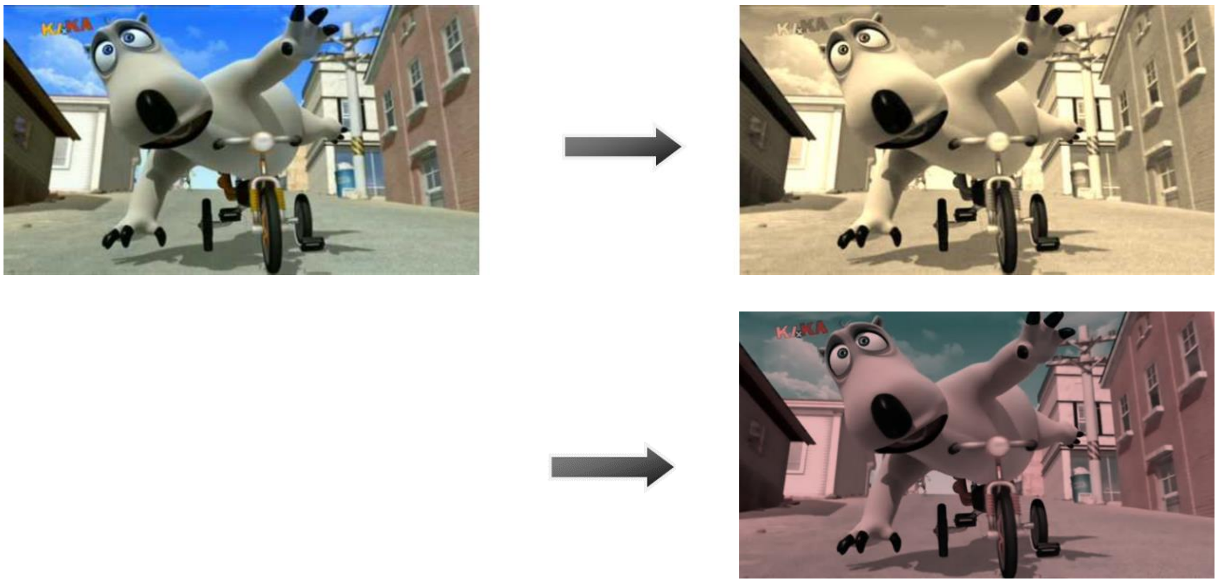
http://en.wikipedia.org/wiki/RGB_color_model

http://en.wikipedia.org/wiki/HSL_and_HSV

FILTER 1B: FILTER_REMOVE_COLOR

GENERAL INFORMATION

This filter has to be implemented in the function filter_remove_color(video, ‘sepia’). It maps each RGB color value to a sepia color value.



IMPLEMENTATION

For this task, the following functions have to be implemented:

Function:	sepia in filter_remove_color
Input:	RGB frame
Output:	converted RGB frame with Sepia color information
File:	filter_remove_color.m

1) Map the RGB values into sepia like values. You can use one of the following formulas buts also different equations are possible:

$$\begin{pmatrix} R_{sepia} \\ G_{sepia} \\ B_{sepia} \end{pmatrix} = \begin{bmatrix} 0.4 & 0.769 & 0.189 \\ 0.349 & 0.686 & 0.168 \\ 0.272 & 0.534 & 0.131 \end{bmatrix} * \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

$$\begin{pmatrix} R_{vintage} \\ G_{vintage} \\ B_{vintage} \end{pmatrix} = \begin{bmatrix} 0.628 & 0.320 & -0.04 \\ 0.026 & 0.644 & 0.033 \\ 0.026 & 0.644 & 0.033 \end{bmatrix} * \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

HINT: The multiplication may cause values to be outside of the definition range. To avoid problems with other filters, these values should be adjusted.

REFERENCES

http://en.wikipedia.org/wiki/RGB_color_model

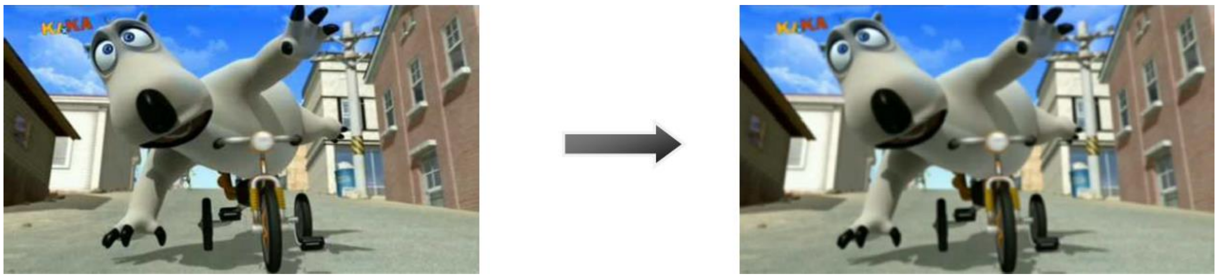
http://en.wikipedia.org/wiki/Photographic_print_toning#Sepia_toning

https://github.com/phoboslab/WebGLImageFilter/blob/master/webgl-image-filter.js

FILTER 2: FILTER_UNSHARP

GENERAL INFORMATION

This filter has to be implemented in the function filter_unsharp(video, param1, …). It applies a Gaussian low-pass filter on each color channel of the RGB image. The filter takes the video structure and any parameter that controls the unsharpness of the image as input. Possible parameters are, for example, the size or the sigma of the Gaussian kernel. Find useful parameter values for the filter.



IMPLEMENTATION

For this task, the following functions have to be implemented:

Function:	filter_unsharp
Input:	RGB frame, filter parameters
Output:	filtered RGB frame
File:	filter_unsharp.m

1) Create a suitable Gaussian low-pass filter.
2) Apply this filter on the image.
3) Find suitable parameters for controlling the filter.

USEFUL MATLAB COMMANDS

fspecial, imfilter

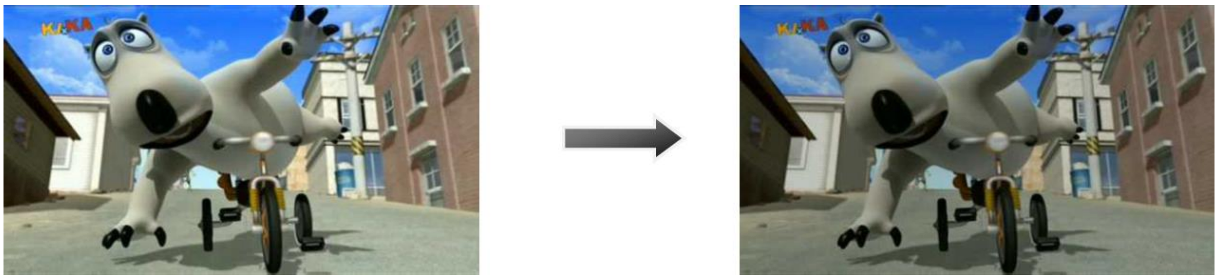
REFERENCES

http://en.wikipedia.org/wiki/Gaussian_interpolation

FILTER 3: FILTER_RAND_ILLUMINATION

GENERAL INFORMATION

This filter has to be implemented in the function filter_rand_illumination(video, min_brightness, max_brightness). The brightness of each pixel should be weakened by a random factor *luma_factor*. This *luma_factor* should lie between a minimum and maximum value.



IMPLEMENTATION

For this task, the following functions have to be implemented:

Function: filter_rand_illumination
Input: RGB frame, min_brightness, max_brightness
Output: filtered RGB frame
File: filter_rand_illumination.m

1) Generate a random value *luma_factor* between [min_brightness, max_brightness] for each frame, where 0 <= min_brightness <= max_brightness <= 1.
2) Multiply each color channel with this *luma_factor*.

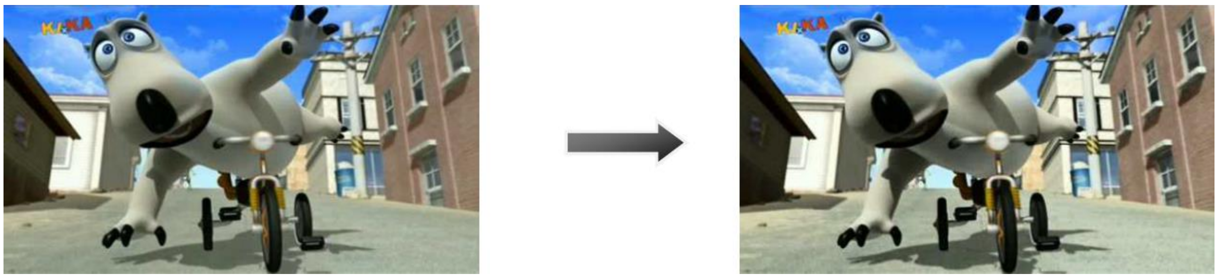
USEFUL MATLAB COMMANDS

rgb2hsv, hsv2rgb, rand

FILTER 4: FILTER_HIGH_CONTRAST

GENERAL INFORMATION

This filter has to be implemented in the function filter_high_contrast(video, dx, dy). The purpose of this filter is to increase the contrast of the image. Therefore, the intensity values should be mapped closer to the minimal and maximal values.

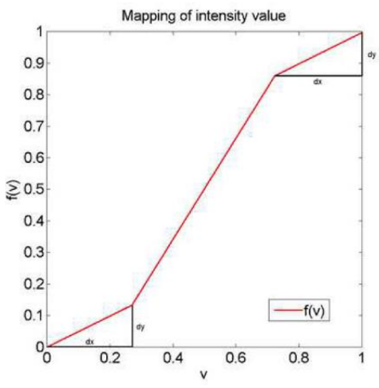


IMPLEMENTATION

For this task, the following functions have to be implemented:

Function:	filter_high_contrast
Input:	RGB frame
Output:	RGB frame with modified histogram
File:	filter_high_contrast.m

1) Create a step function to increase the contrast. The mean gray values have to stay more or less unchanged, therefore, gray values in the upper and lower range should be increased. However, avoid that the image is reduced to a binary representation.



A suitable transfer function can look like this: This function is controlled by the two parameters dx and dy. Create a discrete function f(v) → [0,1] for v between 0-255.

2) Convert the image into the HSV color space. Map the intensity values into the range 0-255.
3) Map each intensity value with f(v).
4) Convert the image back into the RGB color space.

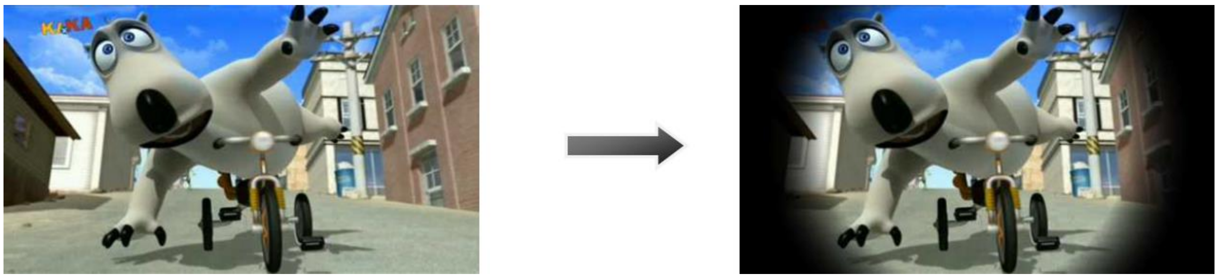
USEFUL MATLAB COMMANDS

rgb2hsv, hsv2rgb, hist

FILTER 5: FILTER_IRIS

GENERAL INFORMATION

This filter has to be implemented in the function filter_iris(video, trans_size, dist_x, dist_y, min_size, max_size). The filter should reproduce the effect of the circular iris that is typical for movies from this era. This should be done by using a brightness mask. A brightness value between 0 and 1 is assigned to each pixel. A value of 1 means that the pixel is not affected by the brightness mask whereas a value of 0 means that the pixel has a brightness value of 0 and is therefore black. The brightness of each pixel is determined by its distance from the center of the camera iris.

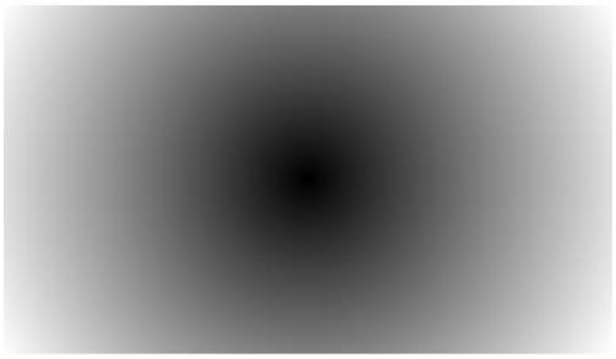


IMPLEMENTATION

For this task, the following functions have to be implemented:

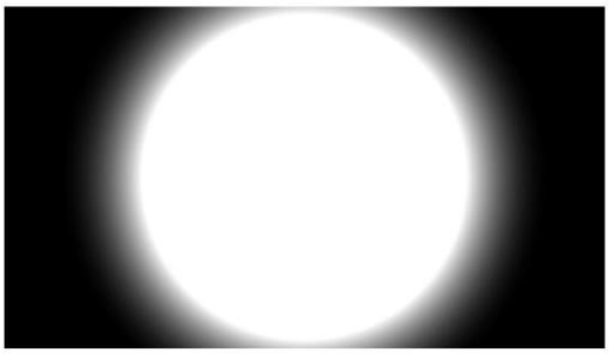
Function:	filter_iris
Input:	RGB frame, dist_x, dist_y, min_size, max_size, trans_size
Output:	filtered RGB frame
File:	filter_iris.m

1) Generate a distance map. The Euclidean distance between the center of the iris and each pixel needs to be calculated. The parameters dist_x and dist_y indicate the horizontal/vertical distance between the center of the iris and the center of the image in pixels. Thus, a shaky motion of the camera can be simulated.



Distance Map

2) Generate a brightness map. The iris has a random size *iris_size* between min_size and max_size (in percent, relative to the image width). Consider d as the distance between a pixel and the center of the iris. Pixels with a smaller or equal distance than iris_size-trans_size stay unchanged. This is the white area in the picture below. For distances between iris_size and iris_size-trans_size a transition from 1 to 0 takes place. You can linearly interpolate or use other transition functions. Pixels with a bigger distance than iris_size get dark.


Brightness Map

3) Multiply the brightness map with each color channel.

REFERENCES

http://en.wikipedia.org/wiki/Alpha_compositing

FILTER 6: FILTER_LOW_FRAMERATE

GENERAL INFORMATION

This filter has to be implemented in the function filter_low_framerate(video, source_fps, dest_fps). The goal is to convert the movie from the original frame rate source_fps into a lower frame rate dest_fps (<source_fps). This filter always runs at the end of the filter chain as the last filter.

IMPLEMENTATION

For this task, the following functions have to be implemented:

Function:	filter_low_framerate
Input:	RGB frame, source_fps, dest_fps
Output:	RGB frame
File:	filter_low_framerate.m

ATTENTION: The video is encoded again with 25 frames per second. In order to achieve the desired effect, images have to be doubled in the correct ratio.

1) The first call of the filter creates a filter-specific array double_frame = (1, 1, 0, 1, 0, 0, 1, …). This array has source_fps entries and indicates whether the current frame (frame(1).filtered) has to be overwritten by the last frame (frame(2).filtered) or not:
    - 0: do not overwrite the frame
    - 1: overwrite the frame, frame(1).filtered = frame(2).filtered
2) Sum(double_frame) = source_fps – dest_fps, since overall dest_fps shall NOT be overwritten. After source_fps the array gets reinitialized for the next source_fps.

USEFUL MATLAB COMMANDS

isfield

REFERENCES

http://en.wikipedia.org/wiki/Frame_rate