UE Videoverarbeitung Winter Term 2011

Exercise 1

1) General Task

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 on 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 on all images.

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

2) Submission

The deadline for exercise1 is **07.11.2011**

The submission must include:

- Commented matlab source files of all filters.
- At the beginning of each filter file following fields have to be completed:
    - 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.

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

3) General information on the 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 takes automatically 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 also 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 with **video.frame(1).filtered**, the previous one with **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. But EVERY filter MUST save its results in the RGB format (RGB values between 0 and 255). Thus no distinction and multiple implementation for different image and color formats is 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 informations 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 current 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 transposed, 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 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)

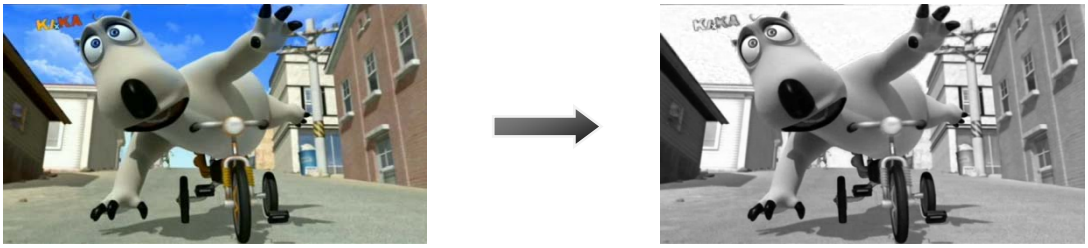
Example 3:
r_info = video.filtered(1:10,1:10,1) → x = 1:10, y = 1:10, c = 1
r_info = 10 x 10 matrix → red color information at the positions (x = 1:10, y = 1:10)

HINT 3:
Try all the filters in an arbitrary order to verify correct functioning.

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 a RGB image and provides a RGB image that contains only shades of gray.



Implementation

For this example the following functions must 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 to 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 to the RGB color space.

Useful MATLAB commands:

rgb2hsv, hsv2rgb

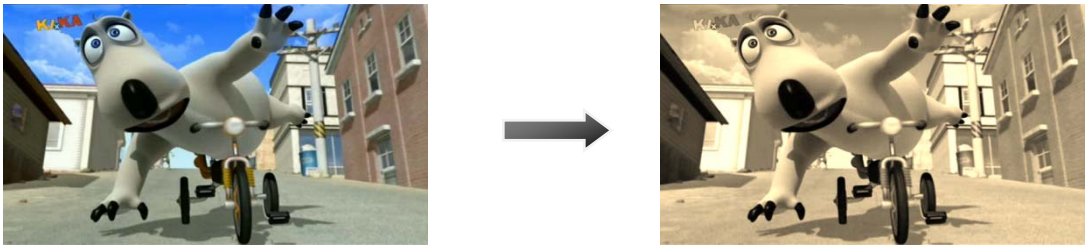
References:

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 example the following functions must 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 to sepia like values. You can use the following formula but also different equations are possible:

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

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

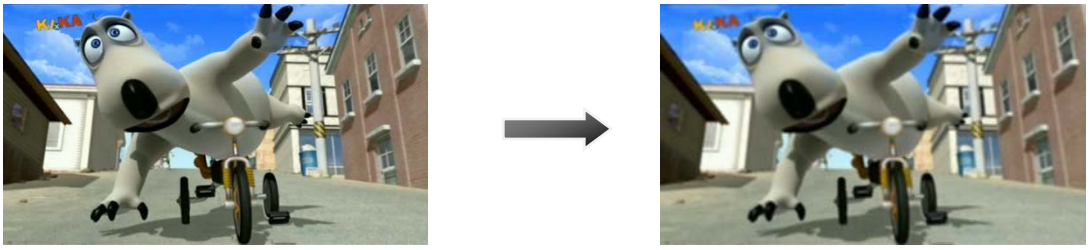
References

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

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 as input the video structure and any parameter that controls the unsharpness of the image. Possible parameters are for example the size or the sigma of the Gaussian kernel. Find useful parameter values for the filter.



Implementation

For this example the following functions must 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

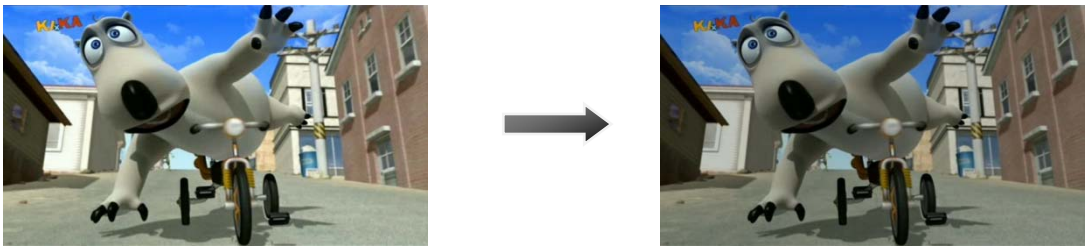
References

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 weaken by a random factor *luma_factor*. This *luma_factor* should lie between a minimum and maximum value.



Implementation

For this example the following functions must 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, in which 0 <= min_brightness <= max_brightness <= 1.
2) Multiply each color channel with this *luma_factor*.

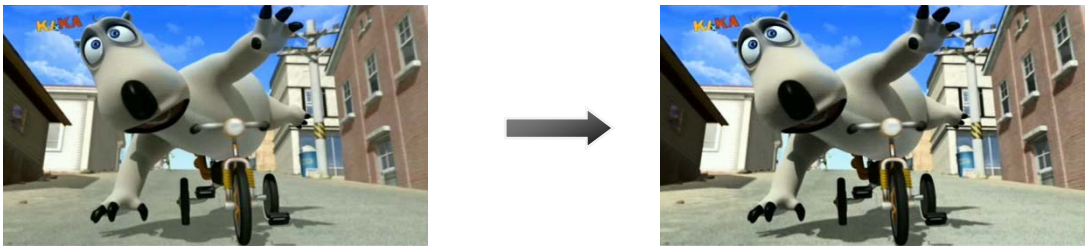
Useful MATLAB commands:

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 get mapped closer to the minimal and maximal values.

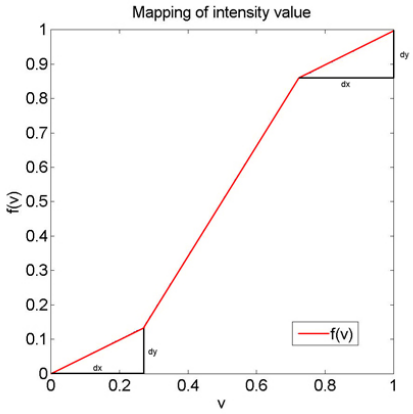


Implementation

For this example the following functions must 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 get 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 to the HSV color space. Map the intensity values to the range 0-255.
3) Map each intensity value with f(v).
4) Convert the image back to the RGB color space.

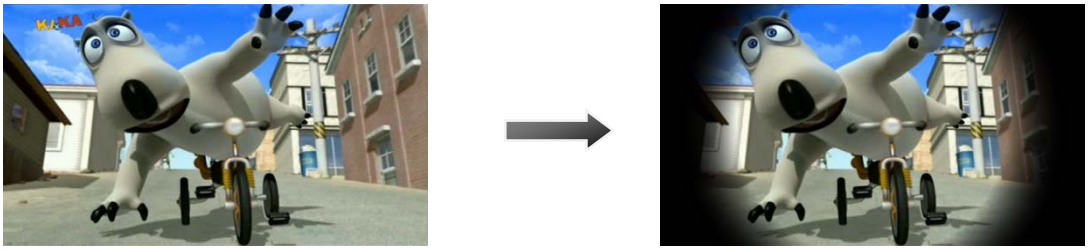
Useful MATLAB commands:

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 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.

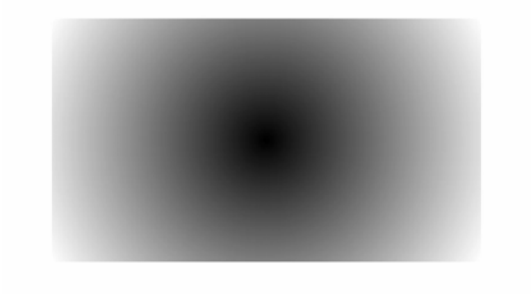


Implementation

For this example the following functions must 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 is calculated. The parameters dist_x and dist_y indicate the horizontal/vertical distance between the iris center 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 to 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 example the following functions must 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 must be doubled in the correct ratio.

1) The first time the filter is called create 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