

Image Analysis

Matlab® Routine (v1.0 – November 2018)

Giuseppe Scarlatella

```

% Matlab® Image Analysis Routine (vl.0 November 2018)
%
%% Created by: Giuseppe Scarlatella
%% Date: 13/11/2018
%% Version: 1.0
%% Edited by: ...
%
%% Software based on Matlab® 2017a (Image Processing Toolbox™ required)
%% Software system based on Windows 10 OS (Apple and Linux compatible)
%
%% Terms of Use: all intellectual rights reserved to Lehrstuhl für
%% Turbomaschinen und Flugantriebe - Fakultät für Maschinenwesen - TUM.
%% Any resource relative to the Matlab® Image Analysis Routine must be
%% shared under specific authorization by Lehrstuhl für Turbomaschinen
%% und Flugantriebe - Fakultät für Maschinenwesen - TUM and for specific
%% research purposes only. Any other purpose will violate this Terms of
%% Use and TUM policies. Any software material, including developer tools
%% and sample code (collectively "Software"), are subject to this specific
%% Terms of Use, unless TUM has provided those items to final user under
%% more specific terms, in which case, those more specific terms will
%% apply to the relevant item.
%
%% INPUT %%%%%% COMMON INPUT %%%%%%
%
%% TestList --> Select image FOLDER
clc; clearvars; close all

TestList = ...
{...
'C:\MATLABImageAnalysisROUTINEv1.0\samples\open burner [schlieren test]'
};

%% IMAGE IMPORT %%%%%%
for i = 1:length(TestList)
    clc ; close all;
    disp('-----');
    disp('Test Directory:'); disp(TestList{i});
    disp('----- IMAGE IMPORT -----');
    Images_Import( TestList{i} );
    disp('-----');
    disp('Press a button to continue...');

    pause
    % clc; clearvars -except TestList; close all;
end
clear i

%% SCHLIEREN ANALYSIS %%%%%%
for i = 1:length(TestList)
    clc ; close all;
    disp('-----');
    disp('Test Directory:'); disp(TestList{i});
    disp('----- IMAGE PROCESSING -----');
    [ img_raw,img_sub,img_ref ] = Images_Processing( TestList{i} );
    disp('----- SCHLIEREN TECHNIQUE -----');
    disp('----- SCHLIEREN ANALYSIS -----');
    [ drho_dy,contrast,density,temperature,x_axis,y_axis ] = ...
        Schlieren_Technique( TestList{i},img_raw,img_sub,img_ref );
    disp('-----');
    disp('Press a button to continue...');

    pause
    clc; clearvars -except TestList; close all;
end
clear i

%% CHEMILUMINESCENCE ANALYSIS %%%%%%
for i = 1:length(TestList)
    clc ; close all;
    disp('-----');
    disp('Test Directory:'); disp(TestList{i});
    disp('----- IMAGE PROCESSING -----');
    [ img_raw,img_sub ] = Images_Processing( TestList{i} );
    disp('----- CHEMILUMINESCENCE TECHNIQUE -----');
    disp('----- CHEMILUMINESCENCE ANALYSIS -----');
    Chemiluminescence_Technique( TestList{i},img_raw );

    disp('-----');
    disp('Press a button to continue...');

    pause
    clc; clearvars -except TestList; close all;
end
clear i

```

Import Images

Matlab® Sub-routine (v1.0 – November 2018)

Scarlattella, Difficile, Laera

```

function Images_Import( ImageDir )
% IMPORT IMAGES (Scarlatella Difficile_Laera)
% IMPORT IMAGES This Matlab Script is meant to collect all data from
% a set of images previously taken and collected in a specific folder.
%% Prints RAW IMAGES, ROTATED IMAGES and WINDOWED&ROTATED IMAGES on screen
%% and video file.
%% INPUT:
%%   - Images FOLDER
%%   - Images CUTTING BOUNDARIES
%%   - ANGLE OF CORRECTION
%%   - Video FRAMES PER SECOND
%% SECTIONS:
%%   - PRE-ROUTINE
%%   - IMAGE IMPORT ROUTINE
%%   - OUTPUT ROUTINE
%% OUTPUT:
%%   - 'Images_Windowed.mat' --> WINDOWED&ROTATED IMAGES on a 3D matrix;
%%   - 'map.mat' --> COLOR PROFILE on a 2D matrix;
tic

% PRE-ROUTINE #####
% PRE-ROUTINE This section reads data from files and finds all '.bmp'
% images in a specific folder.
newFile = fullfile( ImageDir,'test_data.m'); % Subroutine to import test data
run(newFile); clear newFile

FileSystemEntries = dir(fullfile(ImageDir,'*.bmp'));

[tmp ind]=sort({FileSystemEntries.date});
FileSystemEntries=FileSystemEntries(ind);

Window.TopLeft = TopLeft;
Window.BottomRight = BottomRight;

% IMAGE IMPORT ROUTINE #####
% IMAGE IMPORT ROUTINE This Section is meant to collect all data from
% specific input previously setted and print them on screen and on video.
%% SUBSECTIONS:
%%   - RAW IMAGE IMPORT ROUTINE
%%   - ANGLE CORRECTION FOR IMPORTED IMAGE
%%   - IMAGE INTEREST AREA ZOOM
%%   - DATA COLLECTION & PLOT
outputVideo = VideoWriter(fullfile(ImageDir,'test.avi'));

% Creates TEST video file (*.avi)
%% Methods:
%%   - open --> Open file for writing video data.
%%   - close --> Close file after writing video data.
%%   - writeVideo --> Write video data to file.
%% Properties:
%%   - 'FrameRate' --> Rate of playback for the video in frames per second. After

you call open, you cannot change the FrameRate value
outputVideo.FrameRate = FPS;
open(outputVideo);
% Sets VIDEO Framerate (STANDARD fps)?
% Opens file for writing VIDEO

k = 0;
for n = 1:size(FileSystemEntries,1)
    %% RAW IMAGE IMPORT ROUTINE #####
    s(1)= subplot(3, 1, 1); % Subplot for 1st image
    k = k+1;
    File = FileSystemEntries(n);
    FilePath = fullfile(ImageDir, File.name);
    [img,map] = imread(FilePath);
    %(or) img = imread(FilePath);

    image(img);

    colormap(map);
    %(or) colormap('gray');
    title(s(1), sprintf("Raw image: %s", File.name));
    %% ANGLE CORRECTION FOR IMPORTED IMAGE #####
    img = imrotate(img,Angle); % CORRECTION ANGLE for images
    s(2)=subplot(3, 1, 2); % Subplot for 2nd image
    image(img);
    title(s(2), sprintf("Rotated image: %s", File.name));
    %% IMAGE INTEREST AREA ZOOM #####
    XL = Window.TopLeft(1); % Sets bounder points [xL,yL] and [xR,yR]
    xR = Window.BottomRight(1); % that have been previously selected (see lines 15-16)
    yL = Window.TopLeft(2);
    yR = Window.BottomRight(2);

    if XL < 1 % 1st verification on TopLeft input
        error(['WRONG INPUT! ...
        'TopLeft is exceeding image dimension!',...
        'Image dimension is: ',num2str(size(img,2)),',',...
        num2str(size(img,1))])
    elseif yL < 1 % 2nd verification on TopLeft input
        error(['WRONG INPUT! ...
        'TopLeft is exceeding image dimension!',...
        'Image dimension is: ',num2str(size(img,1)),',',...
        num2str(size(img,1))]);
    elseif xR > size(img,2) % 1st verification on BottomRight input
        error(['WRONG INPUT! ...
        'BottomRight is exceeding image dimension!',...
        'Image dimension is: ',num2str(size(img,2)),',',...
        num2str(size(img,1))]);
    elseif yR > size(img,1) % 2nd verification on BottomRight input
        error(['WRONG INPUT! ...
        'BottomRight is exceeding image dimension!',...
        'Image dimension is: ',num2str(size(img,2)),',',...
        num2str(size(img,1))]);
    end
    img_Window = img(yL:yR,xL:xR); % Cuts 'img' into 'img_Window', cutting from [xL,yL] to [xR,yR]
    s(3)=subplot(3,1,3); % Subplot for 3rd image
    image(img_Window);
    title(s(3), sprintf("Windowed image: %s", File.name)); % Displays data in array 'img' (WINDOWED&ROTATED VALUES) as an image
    %% DATA COLLECTION & PLOT #####
    images(k,:,:)= double(img_Window); % Saves 'img_Window' into a 3D matrix
    % (for each 'k' we have a WINDOWED&ROTATED IMAGE)
    drawnow; % Plots on screen the ko image
    writeVideo(outputVideo,img_Window); % Writes on video the ko image (as ko frame)
end

```

```

close(outputVideo)                                % Closes file for writing VIDEO
disp(['N° of images: ',num2str(k)]);             % Display on screen N° of images collected
disp(['Raw images size: ',num2str(size(img,2)),', x ',...    % Display on screen (last) Raw image size
      num2str(size(img,1))]);
disp(['Windowed images size: ',num2str(size(img_Window,2)),', x ',...    % Display on screen (last) Windowed image size
      num2str(size(img_Window,1))]);
info = imfinfo(fullfile(ImageDir,File.name));
if info.BitDepth > 8
    warning(['Images may have not B/W color profile. This could affect',...
        'program results. Consider to use GRAY colormap or to',...
        'convert original raw images.']);
end

%% OUTPUT ROUTINE %%%%%%%%%%%%%%
% OUTPUT ROUTINE This Section is meant to save WINDOWED&ROTATED IMAGES data
% in the main folder.
matName = fullfile(ImageDir, 'Images_Windowed.mat');
matName2 = fullfile(ImageDir, 'map.mat');
matName3 = fullfile(ImageDir, 'FileSystemEntries.mat');

save(matName,'images','nocompression','-v7.3');          % 1ST FILE to be read in Processing Routine
save(matName2,'map');                                     % 2ND FILE to be read in Processing Routine
save(matName3,'FileSystemEntries');                      % 3RD FILE to be read in Processing Routine

% Saves 1ST FILE in memory as 'images'
% and on disk as 'Images_Windowed.mat'
% Saves 2ND FILE in memory as 'map'
% and on disk as 'map.mat'
% Saves 3RD FILE in memory as 'FileSystemEntries'
% and on disk as 'FileSystemEntries.mat'

toc
end

```

Process Images

Matlab® Sub-routine (v1.0 – November 2018)

Scarlattella, Difficile, Laera

```

function [ img_raw,img_sub,img_ref ] = Images_Processing( ImageDir,EvalTimeFactor,EvalWindow,RefImgIndex )
% PROCESS IMAGES (Scarlatella_Difficile_Laera)
% PROCESS IMAGES This Matlab Script is meant to extract raw images in a
% specific temporal interval from 'Images_Windowed.mat' (in test folder)
% and removes video noise by subtracting a REFERENCE IMAGE (wind-off).
%%% Prints REFERENCE IMAGE, RAW IMAGES and DIFFERENCE IMAGES on screen.
%%% INPUT:
%%%   - Images FOLDER
%%%   - EVALUATION TIME FACTOR
%%%   - EVALUATION WINDOW
%%% SECTIONS:
%%%   - PRE-ROUTINE
%%%   - IMAGE PROCESSING ROUTINE
%%% OUTPUT:
%%%   - RAW IMAGES --> unmodified raw images from test session
%%%   - DIFFERENCE IMAGES --> difference between RAW and REFERENCE
%%%   - REFERENCE IMAGE --> "wind-off" test image
tic

% PRE-ROUTINE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PRE-ROUTINE This section reads data from file and selects the temporal
% interval of interest.
newFile = fullfile( ImageDir,'test_data.m'); % Subroutine to import test data
run(newFile); clear newFile

load(fullfile(ImageDir, 'Images_Windowed.mat'));
load(fullfile(ImageDir, 'FileSystemEntries'));
load(fullfile(ImageDir, 'map.mat'));

eval_time = (EvalTimeFactor/1000)*(size(images,1));
eval_time = cell(eval_time);

display(['Evaluation Time (center of interval) index is: ',...
    num2str(eval_time)]);
if eval_time < (size(images,1))/2
    display(['EvalWindow Limit is: ',...
        num2str(eval_time-1)]);
elseif eval_time >= (size(images,1))/2
    display(['EvalWindow Limit is: ',...
        num2str((size(images,1)-eval_time))]);
end

if eval_time < 0
    error(['WRONG INPUT! '...
        'EvalTimeFactor must be an index comprehend in [0,1000]!']);
elseif eval_time > (size(images,1))
    error(['WRONG INPUT! '...
        'EvalTimeFactor must be an index comprehend in [0,1000]!']);
end
if (eval_time-EvalWindow) <= 0
    error(['WRONG INPUT! '...
        'EvalWindow must be an index minor than EvalTime!']);
elseif (eval_time+EvalWindow) > size(images,1)
    error(['WRONG INPUT! '...
        'EvalWindow must not exceed Time Limit!']);
elseif EvalWindow == 0
    error(['WRONG INPUT! '...
        'EvalWindow must be an interval!']);
end

img_eval = images((eval_time-EvalWindow):(eval_time+EvalWindow), :,:);
%(or) img_eval = images(30:146, :,:);

%% IMAGE PROCESSING ROUTINE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% IMAGE PRE-PROCESSING ROUTINE This Section corrects all raw images
% contained in main folder, removing video noise subtracting a REFERENCE
% IMAGE (properly selected).
if RefImgIndex < 1
    error(['WRONG INPUT! '...
        'RefImgIndex must be an index comprehend in [1,',...
        num2str(size(images,1)),']']);
elseif RefImgIndex > size(images,1)
    error(['WRONG INPUT! '...
        'RefImgIndex must be an index comprehend in [1,',...
        num2str(size(images,1)),']']);
end

img_ref = squeeze(images(RefImgIndex, :,:));

figure
img_raw = zeros(size(img_eval,1),size(img_ref,1),size(img_ref,2));
img_sub = zeros(size(img_eval,1),size(img_ref,1),size(img_ref,2));
for k = 1:size(img_eval,1)

    FilePath = FileSystemEntries(k);
    FilePath = fullfile(ImageDir, FilePath);

    s(1)= subplot(2, 2, 1);
    img_raw(k,:,:)= squeeze(img_eval(k,:,:));
    image(squeeze(img_raw(k,:,:)));

    colormap(map);
    title(s(1) , sprintf('Raw image: %s', FilePath));
    s(2)=subplot(2,2,2);
    image(img_ref);
    title(s(2) , 'Reference image');

    s(3)=subplot(2,2,3);
    img_sub(k,:,:)=squeeze(img_raw(k,:,:))-img_ref;
    image(squeeze(img_sub(k,:,:)));
    colormap(map);
    title(s(3) , sprintf('Difference image: %s', FilePath));
    drawnow;

end
toc
end

```

Schlieren Technique

Matlab® Sub-routine (v1.0 – November 2018)

Giuseppe Scarlatella

```

function [drho_dx,contrast,rho_mean,temperature_mean,x_axis,y_axis] = Schlieren_Technique( ImageDir,img_raw,img_sub,img_ref )
% SCHLIEREN TECHNIQUE (ScarlateLLa)
% SCHLIEREN TECHNIQUE This Matlab Script conducts a Schlieren Analysis on
% processed images.
%% Prints IMAGE DIFFERENCE and CONTRAST on screen. Then evaluates density
%% gradient along a specific axis, integrates this gradient through
%% differential equation for density and temperature, prints on screen
%% these results and saves them in the test folder.
%% INPUT:
%%   - Images FOLDER
%%   - RAW IMAGES
%%   - DIFFERENCE IMAGES
%%   - REFERENCE IMAGE
%% SECTIONS:
%%   - PRE-ROUTINE
%%   - SCHLIEREN IMAGE PROCESSING ROUTINE
%%   - SCHLIEREN FLUIDODYNAMIC ANALYSIS ROUTINE
%%   - OUTPUT ROUTINE
%% OUTPUT:
%%   - 'Contrast.mat' --> CONTRAST IMAGES on a 3D matrix;
%%   - 'Density.mat' --> AVERAGE DENSITY IMAGE
%%   - 'Temperature.mat' --> AVERAGE TEMPERATURE IMAGE
%%   - 'X-Axis.mat' --> Current X-AXIS info vector
%%   - 'Y-Axis.mat' --> Current Y-AXIS info vector

% PRE-ROUTINE This section reads data from file.
newFile = fullfile( ImageDir,'test_data.m'); % Subroutine to import test data
run(newFile); clear newFile
newFile = fullfile( ImageDir,'schlieren_data.m'); % Subroutine to import schlieren data
run(newFile); clear newFile

load(fullfile(ImageDir,'FileSystemEntries'));
load(fullfile(ImageDir,'map.mat'));

%% SCHLIEREN IMAGE PROCESSING ROUTINE Description Here
%% SUBSECTIONS:
%%   - MAX & MIN Values
%%   - NORMALIZATION & CONTRAST
%%   - IMAGES PRINTING
disp('SCHLIEREN IMAGE PROCESSING ROUTINE:'); tic

%%%%%%%%%%%%% MAX & MIN Values %%%%%%
MAX_vector = zeros(length(img_raw(:,1,1)),1); % 'MAX & MIN Values' evaluates minimum and maximum
MIN_vector = zeros(length(img_raw(:,1,1)),1); % values of grey graduation on 'img_raw', in order to
for k = 1:length(img_raw(:,1,1)) % scale all the images along the temporal interval
    MAX_raw = maximum(squeeze(img_raw(k,:,:)));
    MAX_sub = maximum(squeeze(img_sub(k,:,:)));
    MAX_vector(k) = MAX_raw;
    if MAX_raw < MAX_sub % to the same levels
        MAX_vector(k) = MAX_sub;
    end
end

MIN_raw = minimum(squeeze(img_raw(k,:,:)));
MIN_sub = minimum(squeeze(img_sub(k,:,:)));
MIN_vector(k) = MIN_sub;
if MIN_raw > MIN_sub
    MIN_vector(k) = MIN_sub;
end
test_MAX = max(MAX_vector);
test_MIN = min(MIN_vector);

%%%%%%%%%%%%% NORMALIZATION & CONTRAST %%%%%%
%% Ref Image NORMALIZATION %%
norm_img_ref = normalize(img_ref(:,:,1),test_MIN,test_MAX,0); % Normalizes data to [-1,+1]
%% Ref Image LIGHTNING %%
[lux_img_ref,MAX] = light_scaling(norm_img_ref,AverageLight); % Converts reference grayscale image in [lux]
% a values matrix, then gives back MAX value for scaling.

%% Ref Images LOW LIGHT LIMIT %%
for m = 1:length(lux_img_ref(:,1))
    for n = 1:length(lux_img_ref(:,1,:))
        if lux_img_ref(m,n) < LowLightLIM % 'LowLightLIM' is also refered to lowest [lux] level
            lux_img_ref(m,n) = LowLightLIM; % sensitivity of CMOS Camera model
        end
    end
end

%% Matrices Pre-Allocation %%
contrast = zeros(length(img_raw(:,1,1)),size(img_ref,1),size(img_ref,2));
lux_img_raw_k = zeros(length(img_raw(:,1,1)),... % (see 'schlieren_data.m')
size(img_ref,1),size(img_ref,2));
lux_img_sub_k = zeros(length(img_raw(:,1,1)),... % see 'schlieren_data.m'
size(img_ref,1),size(img_ref,2));

%% CONTRAST Routine %%
for k=1:length(img_raw(:,1,1))
    %% Raw Images NORMALIZATION %%
    norm_img_raw = normalize(squeeze(img_raw(k,:,:)),test_MIN,test_MAX,0); % Normalizes data to [-1,+1]
    %% Raw Images LIGHTNING %%
    lux_img_raw = norm_img_raw.*MAX; % Converts all grayscale raw images in [lux] values matrices
    %% Raw Images LOW LIGHT LIMIT %%
    for m = 1:length(lux_img_raw(:,1))
        for n = 1:length(lux_img_raw(:,1,:))
            if lux_img_raw(m,n) < LowLightLIM % 'LowLightLIM' is also refered to lowest [lux] level
                lux_img_raw(m,n) = LowLightLIM; % sensitivity of CMOS Camera model
            end
        end
    end
    %% Difference Images LIGHTNING %%
    lux_img_sub = lux_img_raw - lux_img_ref; % Evaluates difference images in [lux] scale
    %% CONTRAST %%
    contrast(k,:,:)= lux_img_sub./lux_img_ref; % Evaluates CONTRAST on each temporal step
    %% Contrast Clean-Up routine %%
    for m = 1:length(img_raw(:,1))
        for n = 1:length(img_raw(:,1,:))
            if lux_img_ref(m,n) <= sensMINValue*LowLightLIM % Cleans up 'contrast' from numerical errors
                contrast(k,m,n) = 0;
            end
        end
    end
    %% Images MEMORIZATION %%
    lux_img_raw_k(:,:,k) = lux_img_raw;
    lux_img_sub_k(:,:,k) = lux_img_sub;
end
clear norm_img_sub; clear norm_img_raw; clear norm_img_ref; % Cleans unnecessary variables from memory
clear lux_img_ref; clear lux_img_raw; clear lux_img_sub; % to increment performances

%%%%%%%%%%%%% IMAGES PRINTING %%%%%%
%% Pre-Routine %%
x_axis = (0:1:size(img_ref,1)-1); % X-axis in [px]
y_axis = (0:1:size(img_ref,2)-1); % Y-axis in [px]

```

```

%%% Lighted Images PRINT %%%
AxisMIN=min(min(min(lux_img_raw_k))); AxisMAX=max(max(max(lux_img_raw_k))); % Prints on screen RAW IMAGES [lux]
for k=1:length(lux_img_raw_k(:,:,1,1))
    figure(4)
    imagesc(x_axis,y_axis,flipud(squeeze(lux_img_raw_k(k,:,:,:))),...
        [AxisMIN AxisMAX]);
    set(gca,'YDir','normal'); xlabel('px'); ylabel('px');
    colorbar; cb = colorbar; ylabel(cb,'lux');
    title('Raw Images');
    drawnow;
end

%%% Difference Images PRINT %%%
AxisMIN=min(min(min(lux_img_sub_k))); AxisMAX=max(max(max(lux_img_sub_k)));
for k=1:length(lux_img_sub_k(:,:,1,1))
    figure(5)
    imagesc(x_axis,y_axis,flipud(squeeze(lux_img_sub_k(k,:,:,:))),...
        [AxisMIN AxisMAX]);
    set(gca,'YDir','normal'); xlabel('px'); ylabel('px');
    colorbar; cb = colorbar; ylabel(cb,'lux');
    title('Difference Images');
    drawnow;
end
clear lux_img_sub_k; % Cleans 'lux_img_sub' from memory to imcrement performances

%%% Contrast Images PRINT & Video OUTPUT %%%
outputVideo = VideoWriter(fullfile(ImageDir,'contrast.avi'));
outputVideo.FrameRate = FPS;
open(outputVideo) % Creates CONTRAST video file (*.avi)
% Sets VIDEO Framerate

AxisMIN=-100.*contrastMAXvalue; AxisMAX=100.*contrastMAXvalue;
for k=1:length(contrast(:,:,1,1))
    figure(6)
    imagesc(x_axis,y_axis,flipud(100.*squeeze(contrast(k,:,:,:))),... % Prints on screen CONTRAST sequence [%]
        [AxisMIN AxisMAX]);
    set(gca,'YDir','normal'); xlabel('px'); ylabel('px');
    colorbar; cb = colorbar; ylabel(cb,'%');
    colormap('bone'); title('Contrast');
    drawnow;
    video_contrast_gray = mat2gray(100.*squeeze(contrast(k,:,:,:)),... % Saves video for CONTRAST sequence
        [AxisMIN AxisMAX]);
    video_contrast_ind = gray2ind(video_contrast_gray);
    video_contrast_RGB = ind2rgb(video_contrast_ind,colormap('bone'));
    writeVideo(outputVideo,video_contrast_RGB);
end

close(outputVideo)
clear video_contrast_gray;clear video_contrast_ind;
clear video_contrast_RGB;
toc

%%%%%%%%%%%%% MEAV VALUES %%%%%%%%%%%%%%
%%% Brightness MEAN VALUES %%%
lux_mean = squeeze(mean(lux_img_raw_k,1)); % Evaluates MEAN BRIGHTNESS matrix [lux]

figure(7)
AxisMIN=min(min(lux_mean)); AxisMAX=max(max(lux_mean));
imagesc(x_axis,y_axis,flipud(lux_mean),[AxisMIN AxisMAX]); % Prints on screen MEAN BRIGHTNESS matrix [lux]
set(gca,'YDir','normal'); xlabel('px'); ylabel('px');
colorbar; cb = colorbar; ylabel(cb,'lux');
title('Average Brightness');drawnow;
saveas.figure(7, [ImageDir '\Average Brightness'],'png') % Saves as file MEAN BRIGHTNESS matrix [lux]
saveas.figure(7, [ImageDir '\Average Brightness'],'fig') % Cleans 'lux_img_raw_k' from memory to imcrement performances

clear lux_img_raw_k;

%%% Contrast MEAN VALUES %%%
contrast_mean = squeeze(mean(contrast,1)); % Evaluates CONTRAST matrix [%]

figure(8)
AxisMIN=-100.*contrastMAXvalue; AxisMAX=100.*contrastMAXvalue;
imagesc(x_axis,y_axis,flipud(100.*contrast_mean),[AxisMIN AxisMAX]); % Prints on screen CONTRAST matrix [%]
set(gca,'YDir','normal'); xlabel('px'); ylabel('px');
colorbar; cb = colorbar; ylabel(cb,'%');
colormap('bone'); title('Average Contrast');drawnow;
saveas.figure(8, [ImageDir '\Average Contrast'],'png') % Saves as file CONTRAST matrix [%]
saveas.figure(8, [ImageDir '\Average Contrast'],'fig')

%% SCHLIEREN FLUIDODYNAMIC ANALYSIS ROUTINE %%%%%%%%%%%%%%
%% SCHLIEREN FLUIDODYNAMICS ANALYSIS ROUTINE Description Here
%%% SUBSECTIONS:
%%% - PRE-ROUTINE
%%% - DENSITY GRADIENT
%%% - DENSITY
%%% - TEMPERATURE
%%% - PRESSURE
disp('SCHLIEREN FLUIDODYNAMIC ANALYSIS ROUTINE:'); tic
%%% PRE-ROUTINE %%%%%%%%%%%%%%
load(fullfile(ImageDir,'FileSystemEntries'));
File = FileSystemEntries(1); % Selects wind-off picture
FilePath = fullfile(ImageDir, File.name); % Writes PATH for K^* file
[wind_off] = imread(FilePath);

[diam] = find_circle_diameter( wind_off );
step = Eval_area_r_d/diam;

L = abs(Extension(1)-Extension(2))*step; % [m] % Depth of the flowfield along the optical axis. No need
x_axis = (0:1:size(contrast_mean,1)-1)*(step*1e3); % [mm] % any corrections for "Uniform" Object
y_axis = (0:1:size(contrast_mean,2)-1)*(step*1e3); % [mm] % X-axis in [mm]
% X-axis in [mm]

%%% Window Detection %%%
if Window_setup == "On"
    disp("Window detected: rays deflection correction for 'n0' values different from 'n_air'");
    disp("for medium with ''rho0'' value different from ''rho_air'''");
    disp("Best flowfield depth option is: ''AllDomain'''");
    disp(['Selected flowfield depth domain: ',Geometry]);
    if Flowfield_depth == 0
        warning("Flowfield depth must be different from ''0'' for closed domains!");
    end
else
    warning("Window setup == "Off");
end
elseif Window_setup == "Off"
    disp("Window undetected: unable to properly correct rays deflection for 'n0' values different from 'n_air'");
    disp("for medium with ''rho0'' value different from ''rho_air'''");
    disp("Best flowfield depth option are: ''Linear/Diamond'' or ''Sinusoidal/Cylindric''.");
    disp(['Selected flowfield depth domain: ',Geometry]);
    if Flowfield_depth ~= 0
        warning("Flowfield depth must be equal to ''0'' for opened domains!");
    end
end
else
    warning("Wrong input on 'Window_setup'!!!");
end

%%%%%%%%%%%%% DENSITY GRADIENT %%%%%%%%%%%%%%
gldstn = ((n0-1)/rho0)*(f2/a_k); % Quantity close to Gladstone-Dale constant

```

```

% ( k=(n-1)/rho )

drho_dx = zeros(size(contrast));
for k=1:length(contrast(:,1,1))
    drho_dx(k,:,:) = (1/gldstn)*(1/L)*(n_air/n0).*contrast(k,:,:);
end

%%% Density Gradient Images PRINT & Video OUTPUT
outputVideo = VideoWriter(fullfile(ImageDir,'gradient.avi'));
outputVideo.FrameRate = FPS;
open(outputVideo)

AxisMIN=min(min(min(drho_dx)));AxisMAX=max(max(max(drho_dx)));
for k=1:length(drho_dx(:,:,1,1))
    figure(9)
    imagesc(x_axis,y_axis,flipud(1e-3*squeeze(drho_dx(k,:,:))),...
        1e-3*[AxisMIN/gradientZOOMfactor AxisMAX/gradientZOOMfactor]);
    set(gca,'YDir','normal'); xlabel('mm'); ylabel('mm');
    colorbar; cb = colorbar; ylabel(cb,'kg/m^3*mm^{-1}');
    colormap('jet'); title('Density Gradient'); drawnow;
    video_drho_gray = mat2gray(1e-3*squeeze(drho_dx(k,:,:)),...
        1e-3*[AxisMIN/gradientZOOMfactor AxisMAX/gradientZOOMfactor]);
    video_drho_ind = gray2ind(video_drho_gray);
    video_drho_RGB = ind2rgb(video_drho_ind,colormap('jet'));
    writeVideo(outputVideo,video_drho_RGB);
end

close(outputVideo)
clear video_drho_gray;clear video_drho_ind;
clear video_drho_RGB;

%%% Density Gradient MEAN VALUES %%
drho_dx_mean = squeeze(mean(drho_dx,1));

figure(10)
imagesc(x_axis,y_axis,1e-3*flipud(drho_dx_mean),...
    [AxisMIN/gradientZOOMfactor AxisMAX/gradientZOOMfactor]*1e-3);
set(gca,'YDir','normal'); xlabel('mm'); ylabel('mm');
colorbar; cb = colorbar; ylabel(cb,'kg/m^3*mm^{-1}');
colormap('jet'); title('Average Density Gradient'); drawnow;
saveas(figure(10),[ImageDir '\Average Density Gradient'], 'png')
saveas(figure(10),[ImageDir '\Average Density Gradient'], 'fig')

clear AxesMIN; clear AxesMAX;

%%%%%%%%%%%%% DENSITY %%%%%%%%%%%%%%
disp('Density integration procedure. Please wait...');

%%% Matrix Rotation %%
contrast_mean = flip_matrix( contrast_mean,xVERSE,yVERSE );
% Reads verse of integration and flips 'contrast_mean' matrix

%%% Integration ( by Taylor Expansion to 1st term - alternative method ) \% % Evaluates DENSITY by a Taylor Expansion to 1st term on
% [X0] = x_BCs_reader(x_BCs,drho_dx_mean,rho0);
% [Y0] = y_BCs_reader(y_BCs,drho_dx_mean,rho0);
% [rho_mean] = matrix_integral( drho_dx_mean,step,X0,xVERSE,Y0,yVERSE );
% on uniform domain hypothesis

% Initiates 'dtemperature_dx_mean'
% Evaluates BCs on density
% Initiates 'drho_mean'
% Evaluates BCs on density first derivative

%%% Integration ( by ODE45 ) %%
rho_mean = zeros(size(contrast_mean,1),size(contrast_mean,2));
rho_mean(:,1) = rho0.*ones(size(contrast_mean,1),1);
drho_mean = zeros(size(contrast_mean,1),size(contrast_mean,2));
drho_mean(:,1) = (1/gldstn)*(1/le-3)*contrast_mean(:,1);
% (or)
% drho_mean(:,1) = (1/gldstn)*(1/L)*contrast_mean(:,1);
xspan = [0,step*size(contrast_mean,2)];
xx = linspace(xspan(1),xspan(2),size(contrast_mean,2));
for m = 1:size(contrast_mean,1)
    C = @(x) interpl(xx,contrast_mean(m,:),x);
    y0 = [rho_mean(m,1); drho_mean(m,1)];
    options= odeset('MaxStep',1/(2*size(contrast_mean,2)));
    sol_rho = ode45(@(x) ODE_rho_function(x,y,...%
        gldstn,n0,n_air,xx,C,Geometry,Extension,Flowfield_depth,...%
        flowfield_min_depth,step,xVERSE,yVERSE),xx,y0,options);
    Rho = deval(sol_rho,x0);
    rho_mean(m,:) = Rho(1,:);
    drho_mean(m,:) = Rho(2,:);
end
clear y0; clear Rho; clear sol_rho;

%%% Density Matrices Rotation Correction %%
contrast_mean = unflip_matrix( contrast_mean,xVERSE,yVERSE );
rho_mean = unflip_matrix( rho_mean,xVERSE,yVERSE );
drho_mean = unflip_matrix( drho_mean,xVERSE,yVERSE );

%%% Density PRINT %%
AxisMIN=min(min(rho_mean));AxisMAX=max(max(rho_mean));
if AxisMIN < densitySENSITIVITY(1)
    AxisMIN = densitySENSITIVITY(1);
end
if AxisMAX > densitySENSITIVITY(2)
    AxisMAX = densitySENSITIVITY(2);
end
figure(11)
imagesc(x_axis,y_axis,flipud(rho_mean),[AxisMIN AxisMAX]);
set(gca,'YDir','normal'); xlabel('mm'); ylabel('mm');
colorbar; cb = colorbar; ylabel(cb,'kg/m^3');
colormap('parula'); title('Average Density'); drawnow;
saveas(figure(11),[ImageDir '\Average Density'], 'png')
saveas(figure(11),[ImageDir '\Average Density'], 'fig')

%%% Density Avg. Values along integration axis Evaluation & PRINT %%
figure(12)
plot(1e3*xx,mean(rho_mean),'LineWidth',2);
xlabel('mm')ylabel('kg/m^3');
axis([1e3*xspan(1) 1e3*xspan(2) min(mean(rho_mean))...
    max(mean(rho_mean))]); grid ON;
if yVERSE == "Null"
    title('Average Density along x-axis'); drawnow;
    saveas(figure(12),[ImageDir '\Average Density along x-axis'], 'png')
    saveas(figure(12),[ImageDir '\Average Density along x-axis'], 'fig')
elseif xVERSE == "Null"
    title('Average Density along y-axis'); drawnow;
    saveas(figure(12),[ImageDir '\Average Density along y-axis'], 'png')
    saveas(figure(12),[ImageDir '\Average Density along y-axis'], 'fig')
end

%%% Density Gradient PRINT %%
% AxesMIN=min(min(drho_mean));AxesMAX=max(max(drho_mean));
% figure(18)
% imagesc(x_axis,y_axis,1e-3*flipud(drho_mean),1e-3*[AxesMIN AxesMAX]);
% set(gca,'YDir','normal'); xlabel('mm'); ylabel('mm');
% colorbar; cb = colorbar; ylabel(cb,'kg/m^3*mm^{-1}');
% colormap('parula'); title('Average Density Gradient'); drawnow;
% saveas(figure(18),[ImageDir '\Average Density Gradient'], 'png')

%%% Avg. Density 3-D Model PRINT %%
AxisMIN=min(min(rho_mean));AxisMAX=max(max(rho_mean));
% Redefines Axes for DENSITY 3-D Model print

```

```

if AxisMIN < densitySENSITIVITY(1) % Limits DENSITY 3-D Model axes for avoiding numerical errors
    AxisMIN = densitySENSITIVITY(1);
end
if AxisMAX > densitySENSITIVITY(2)
    AxisMAX = densitySENSITIVITY(2);
end
figure(20)
surf(y_axis,x_axis,flipud(rho_mean),'LineStyle','none');
zlim([AxisMIN AxisMAX]);
xlabel('mm'); ylabel('mm'); view(45,75); grid off; axis off;
colorbar; cb = colorbar; ylabel(cb,'kg/m^3'); caxis([AxisMIN AxisMAX]);
colorbar off;
colormap('parula'); title('Avg. Density 3-D Model'); drawnow;

%%% Avg. Density Contour 2-D PRINT %%%
figure(22)
[Cnt c] = contour(y_axis,x_axis,flipud(rho_mean),15);
c.LineWidth = 2;
xlabel('mm'); ylabel('mm');
colorbar; cb = colorbar; ylabel(cb,'kg/m^3'); caxis([AxisMIN AxisMAX]);
colorbar off;
colormap('parula'); title('Avg. Density Contour 2-D'); drawnow;

%%%%%%%%%%%%% TEMPERATURE %%%%%%%%%%%%%%
disp('Temperature integration procedure. Please wait...');

%%% Matrix Rotation %%%
contrast_mean = flip_matrix( contrast_mean,xVERSE,yVERSE ); % Reads verse of integration and flips 'contrast_mean' matrix

%%% pressure = p0*ones(size(contrast_mean,1),1); % Hp: ISOBARIC PROCESS
% (Valid for Open Burner, Not Valid for Comb.Ch.)

temperature_mean = zeros(size(contrast_mean,1),size(contrast_mean,2));
temperature_mean(:,1) = T0.*ones(size(contrast_mean,1),1);

% / Integration ( by Taylor Expansion to 1st term - alternative method )\% % Evaluates TEMPERATURE by a Taylor Expansion to 1st term on
% for m=1:size(contrast_mean,1) % on uniform domain hypotesis (only for L->R integrations)
% for n=2:size(contrast_mean,2)
%     temperature_mean(m,n) = temperature_mean(m,n-1) + ...
%     (-1/gldstn)*(1/L)*(R/p0)*temperature_mean(m,n-1).^2*contrast_mean(m,n)*step;
% end
% end

%%% Integration ( by ODE15s ) %%%
dtemperature_mean = zeros(size(contrast_mean,1),size(contrast_mean,2)); % Initiates 'dtemperature_dx_mean'
% pressure_mean(:,1) = ... % Evaluates BCs on density first derivative
temperature_mean(:,1).^2;
% (or)
% dtemperature_mean(:,1) = ...
% -(1/gldstn)*(1/le-1)*R./pressure.*contrast_mean(:,1).*...
% temperature_mean(:,1).^2;
% (or)
% dtemperature_mean(:,1) = zeros(size(contrast_mean,1),1); % Defines variable of integration interval extremes
xspan = [0,step*size(contrast_mean,2)]; % Generates a vector of coordiantes for variable of integration
xx = linspace(xspan(1),xspan(2),size(contrast_mean,2));
for m = 1:size(contrast_mean,1)
    p = pressure(m);
    C = @(x) interp1(xx,contrast_mean(m,:),x);
    y0 = [temperature_mean(m,1); dtemperature_mean(m,1)];
    options= odeset('MaxStep',1/(2*size(contrast_mean,2)));
    sol_temp = ode15s(@(x,y) ODE_T_function(x,y,... % Interpolates the data set (Contr_x,Contr) at value x
        p,gldstn,n0,n_air,R,xx,C,Geometry,Extension,Flowfield_depth,... % BCs for ODE15s Routine
        flowfield_min_depth,step,xVERSE,yVERSE),xx,y0,options); % Limit on 'MaxStep' for ODE15s Routine
    T = deval(sol_temp,xx); % Solves differential equation for each row
    temperature_mean(m,:) = T(1,:);
    dtemperature_mean(m,:) = T(2,:);
end
clear y0; clear T; clear sol_temp;

%%% Matrices Rotation Correction %%%
contrast_mean = unflip_matrix( contrast_mean,xVERSE,yVERSE );
temperature_mean = unflip_matrix( temperature_mean,xVERSE,yVERSE );
dtemperature_mean = unflip_matrix( dtemperature_mean,xVERSE,yVERSE );

%%% Temperature PRINT %%
AxisMIN=min(min(temperature_mean));AxisMAX=max(max(temperature_mean));
if AxisMIN < temperatureSENSITIVITY(1) % Redefines Axes for TEMPERATURE print
    AxisMIN = temperatureSENSITIVITY(1);
end
if AxisMAX > temperatureSENSITIVITY(2) % Limits TEMPERATURE axes for avoiding numerical errors
    AxisMAX = temperatureSENSITIVITY(2);
end
figure(13)
imagesc(x_axis,y_axis,flipud(temperature_mean),[AxisMIN AxisMAX]);
set(gca,'YDir','normal'); xlabel('mm'); ylabel('mm');
colorbar; cb = colorbar; ylabel(cb,'K');
colormap('hot'); title('Average Temperature'); drawnow;
saveas(figure(13),[ImageDir 'Average Temperature','png']);
saveas(figure(13),[ImageDir '\Average Temperature','fig'])

%%% Temperature Avg. Values along integration axis Evaluation & PRINT %%
figure(14)
plot(1e3*xx,mean(temperature_mean), 'LineWidth',2);
xlabel('mm'); ylabel('K');
axis([1e3*xspan(1) 1e3*xspan(2) min(mean(temperature_mean))... % max(mean(temperature_mean))]); grid ON;
if yVERSE == "Null"
    title('Average Temperature along x-axis'); drawnow;
    saveas(figure(14),[ImageDir 'Average Temperature along x-axis'], 'png')
    saveas(figure(14),[ImageDir 'Average Temperature along x-axis'], 'fig')
elseif xVERSE == "Null"
    title('Average Temperature along y-axis'); drawnow;
    saveas(figure(14),[ImageDir 'Average Temperature along y-axis'], 'png')
    saveas(figure(14),[ImageDir '\Average Temperature along y-axis'], 'fig')
end

% % Temperature Gradient PRINT %
% figure(19)
% imagesc(x_axis,y_axis,1e-3*flipud(dtemperature_mean),... % 1e-3*[AxesMIN AxesMAX]);
% set(gca,'YDir','normal'); xlabel('mm'); ylabel('mm');
% colorbar; cb = colorbar; ylabel(cb,'K*mm^-1');
% colormap('autumn'); title('Average Temperature Gradient'); drawnow;
% saveas(figure(18),[ImageDir 'Average Temepature Gradient'], 'png')

%%% Thresholding Procedure ( Flame Shape ) %%%
metric = 0; N = 1;
while metric <=0.95 && N < 20
    [thresh,metric] = multithresh(temperature_mean,N);
    N = N+1;
end
seg_temperature_mean = imquantize(temperature_mean,thresh);
figure(15)
imagesc(x_axis,y_axis,flipud(seg_temperature_mean));
set(gca,'YDir','normal'); xlabel('mm'); ylabel('mm');
colorbar; cb = colorbar; ylabel(cb,'K');
colormap('hot'); title('Object Shape'); drawnow;

```

```

saveas(figure(15),[ImageDir '\Temperature Shape'],'png')

%%% Avg. Temperature 3-D Model PRINT %%%
AxisMIN=min(min(temperature_mean));AxisMAX=max(max(temperature_mean));
if AxisMIN < temperatureSENSITIVITY(1)
    AxisMIN = temperatureSENSITIVITY(1);
end
if AxisMAX > temperatureSENSITIVITY(2)
    AxisMAX = temperatureSENSITIVITY(2);
end
figure(21)
surf(y_axis,x_axis,flipud(temperature_mean),'LineStyle','none');
zlim([AxisMIN AxisMAX]);
xlabel('mm'); ylabel('mm'); view(45,75); grid off; axis off;
colorbar; cb = colorbar; ylabel(cb,'K'); caxis([AxisMIN AxisMAX]);
colorbar off;
colormap('hot'); title('Avg. Temperature 3-D Model'); drawnow;

%%% Avg. Temperature Contour 2-D PRINT %%%
figure(23)
[Cnt c] = contour(y_axis,x_axis,flipud(temperature_mean),15);
c.LineWidth = 2;
xlabel('mm'); ylabel('mm');
colorbar; cb = colorbar; ylabel(cb,'K'); caxis([AxisMIN AxisMAX]);
colormap('hot'); title('Avg. Density Contour 2-D'); drawnow;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% PRESSURE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pressure_mean = R*rho_mean.*temperature_mean; % In case of ISOBARIC PROCESS, this graphic
% has verification purposes only

%%% Pressure PRINT %%%
if Window_setup == "On"
    AxisMIN=min(min(pressure_mean));AxisMAX=max(max(pressure_mean));
elseif Window_setup == "off"
    AxisMIN=0.5*1e5; AxisMAX=1.5*1e5;
end
figure(16)
imagesc(x_axis,y_axis,le-5*flipud(pressure_mean),le-5*[AxisMIN AxisMAX]);
set(gca,'Ydir','normal'); xlabel('mm'); ylabel('mm');
colorbar; cb = colorbar; ylabel(cb,'MPa');
colormap('winter'); title('Average Pressure'); drawnow;
saveas(figure(16),[ImageDir '\Average Pressure'],'png');
saveas(figure(16),[ImageDir '\Average Pressure'],'fig');

%%% Pressure Avg. Values along integration axis Evaluation & PRINT %
if Window_setup == "On"
    AxisMIN=min(mean(pressure_mean));AxisMAX=max(mean(pressure_mean));
elseif Window_setup == "off"
    AxisMIN=0.5*1e5; AxisMAX=1.5*1e5;
end
figure(17)
plot(1e3*x,le-5*mean(pressure_mean),'LineWidth',2);
xlabel('mm'); ylabel('MPa');
axis([1e3*xspan(1) 1e3*xspan(2) le-5*AxisMIN...
    le-5*AxisMAX]); grid On;
if yVERSE == "Null"
    title('Average Pressure along x-axis'); drawnow;
    saveas(figure(17),[ImageDir '\Average Pressure along x-axis'],'png')
    saveas(figure(17),[ImageDir '\Average Pressure along x-axis'],'fig')
elseif xVERSE == "Null"
    title('Average Pressure along y-axis'); drawnow;
    saveas(figure(17),[ImageDir '\Average Pressure along y-axis'],'png')
    saveas(figure(17),[ImageDir '\Average Pressure along y-axis'],'fig')
end

toc
%% OUTPUT ROUTINE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% OUTPUT ROUTINE This Section is meant to save CONTRAST IMAGES, DENSITY and
% TEMPERATURE data and current AXES in the main folder.

matName = fullfile(ImageDir, 'Contrast.mat');
save(matName,'contrast','-nocompression','-v7.3');

matName = fullfile(ImageDir, 'Density.mat');
save(matName,'rho_mean','-nocompression','-v7.3');

matName = fullfile(ImageDir, 'Temperature.mat');
save(matName,'temperature_mean','-nocompression','-v7.3');

matName = fullfile(ImageDir, 'X-Axis.mat');
save(matName,'x_axis','-nocompression','-v7.3');

matName = fullfile(ImageDir, 'Y-Axis.mat');
save(matName,'y_axis','-nocompression','-v7.3');

end

```

Chemiluminescence Technique

Matlab® Sub-routine (v1.0 – November 2018)

Difficile, Scarlatella

```

function [] = Chemiluminescence_technique( ImageDir,img_raw )
%% CHEMILUMINESCENCE technique [Difficile]
%% CHEMILUMINESCENCE technique This Matlab Script conducts Chemiluminescence
%% Analysis on processed images.
%%% INPUT:
%%%   - Images FOLDER
%%%   - RAW IMAGES
%%%   - DIFFERENCE IMAGES
%%% SECTIONS:
%%%   - PRE-ROUTINE
%% CHEMILUMINESCENCE IMAGE PROCESSING ROUTINE
%%% OUTPUTS:
%%%   - ???
tic

%% PRE-ROUTINE %%%%%%%%%%%%%%
newFile = fullfile( ImageDir,'test_data.m'); % Subroutine to import test data
run(newFile); clear newFile

load(fullfile(ImageDir, 'Images_Windowed.mat'));
load(fullfile(ImageDir, 'FileSystemEntries'));
load(fullfile(ImageDir, 'map.mat'));

%% CHEMILUMINESCENCE IMAGE PROCESSING ROUTINE %%%%%%%%%%%%%%
% IMAGE PROCESSING ROUTINE Description Here
%%% SUBSECTIONS:
%%%   - MEAN IMAGE & VALUES
%%%   - ???
%%%%%%%%%%%%% MEAN IMAGE & VALUES %%%%%%%%%%%%%%
%%%%%%%%%%%%% ??? %%%%%%%%%%%%%%
img_mean = squeeze(mean(img_raw));
s(1)=subplot(2,2,1);
image(img_mean);
meanBrightness=mean(mean(img_mean));
title(sprintf('Mean Image, brightness=%4.1f', meanBrightness));

img_var = squeeze(var(img_raw, 0, 1));

s(4)=subplot(2,2,4);
img_var = sqrt(img_var);
contour(img_var);
% contourf(img_var);
% colormap(map);
title('var(raw-mean)');

saveas(gcf,[ImageDir '\plot1'],'fig')

%%%%%%%%%%%%% Plots %%%%%%%%%%%%%%
%%%%%%%%%%%%% Average of 50 pictures at the evaluation time %%%%%%%%%%%%%%
figure(2)

img_mean = squeeze(mean(img_raw));
image(img_mean);

%meanBrightness = mean(mean(img_mean));
%title(sprintf('Mean Image, brightness=%4.1f', meanBrightness));
colorbar;
limits=[0 50];
caxis(limits)
%caxis([10 70]) %to define the limits of the color
title('var(raw-mean)');
% xlim([])
% activate only for paper format!
%   c=colorbar('southoutside');
%   set(c,'XTick',[10 70],'XTicklabel',{'min','max'});
axis off
title ''
box off
% set(gcf,'PaperPositionMode','auto');
% set(gcf, 'Position', [0 0 400 120]);
% set(gca,'position',[0 0 1],'units','normalized');
saveas(figure(2),[ImageDir '\Average'],'fig')
saveas(figure(2),[ImageDir '\Average'],'png')

%%%%%%%%%%%%% Brightness %%%%%%%%%%%%%%
figure(3)

plot(mean(img_mean))
title(sprintf('Brightness'));
%legend(sprintf('BRe%4.1f', meanBrightness))
% set(gcf,'PaperPositionMode','auto');
% set(gcf, 'Position', [0 0 400 120]);
% set(gca,'position',[0 0 1],'units','normalized');
saveas(figure(3),[ImageDir '\Brightness'],'fig')
saveas(figure(3),[ImageDir '\Brightness'],'png')

%%%%%%%%%%%%% Average false color %%%%%%%%%%%%%%
figure(4)

img_mean = squeeze(mean(img_raw));
image(img_mean);

%to divide the image into 5 regions based on intensity and color %
%to generate a false color image, that is 0-50 with black, %
%50-100 with blue,100-150 with green, 150-200 red, and 200-255 white %

%{~, ~, binnumber} = histcounts(GrayArray, [0 50 100 150 255]);
%cmap = [0 0 0; 0 0 1; 0 1 0; 1 0 0; 1 1 1];
%imshow( binnumber, 'ColorMap', cmap)

%X = grayslice(img_mean, 5);
%cmap = [0 0 0; 0 0 1; 0 1 0; 1 0 0; 1 1 1];
%rgb_image = ind2rgb(X, cmap);

colormap(jet(256));
%colormapeditor

colorbar;
%caxis([10 70]) %to define the limits of the color
%title('var(raw-mean)');
% xlim([])
% activate only for paper format!
%   c=colorbar('southoutside');
%   set(c,'XTick',[10 70],'XTicklabel',{'min','max'});
axis off
title ''
box off
% set(gcf,'PaperPositionMode','auto');
% set(gcf, 'Position', [0 0 400 120]);
% set(gca,'position',[0 0 1],'units','normalized');
saveas(figure(4),[ImageDir '\Average false color'],'fig')
saveas(figure(4),[ImageDir '\Average false color'],'png')

```

```

%%%%%%%%%%%%% Variance false color %%%%%%
figure(5)
img_var = squeeze(var(img_raw, 0, 1));
img_var = sqrt(img_var);
image(img_var);

colormap(jet(80));
%colormapeditor

c=colormap;
%caxis([10 70]) %to define the limits of the color
title('var(raw-mean)');
% xlim([])
% activate only for paper format!
% c=colorbar('southoutside');
% set(c,'Xtick',[10 70],'Xticklabel',{'min','max'});
axis off
title ''
box off
% set(gcf,'PaperPositionMode','auto');
% set(gcf,'Position', [0 0 400 120]);
% set(gca,'position',[0 0 1 1],'units','normalized');
saveas.figure(5),[ImageDir '\Variance false color'],'fig')
saveas.figure(5),[ImageDir '\Variance false color'], 'png')

%%%%%%%%%%%%% Average Edge %%%%%%
figure(6)
img_mean = squeeze(mean(img_raw));
image(img_mean);
%BW_log = edge(img_mean, 'log', 0.1);
%BW_sobel = edge(img_mean, 'sobel', 8.9);
%BW_prewitt = edge(img_mean, 'prewitt', 8.9);
%BW_roberts = edge(img_mean, 'roberts', 8.9);
%BW_zerocross = edge(img_mean, 'zerocross', 0.1);
BW_canny = edge(img_mean, 'canny', 0.21); %0.49 %0.21/2.2 - 0.21

%imshowpair(img_mean,BW_log,'montage');
%imshowpair(img_mean,BW_sobel,'montage');
%imshowpair(img_mean,BW_prewitt,'montage');
%imshowpair(img_mean,BW_roberts,'montage');
%imshowpair(img_mean,BW_zerocross,'montage');
BW_canny_inv = imcomplement(BW_canny);
imshow(BW_canny_inv);

%caxis([10 70]) %to define the limits of the color
%title('var(raw-mean)');
% xlim([])
% activate only for paper format!
% c=colorbar('southoutside');
% set(c,'Xtick',[10 70],'Xticklabel',{'min','max'});
axis off
title ''
box off
%set(gcf,'PaperPositionMode','auto');
%set(gcf, 'Position', [10 0 400 120]);
%set(gca,'position',[0 0 1 1],'units','normalized');
saveas.figure(6),[ImageDir '\Average edge detection'],'fig')
saveas.figure(6),[ImageDir '\Average edge detection'], 'png')

%%%%%%%%%%%%% Instantaneous edge %%%%%%
% for n = 1:size(img_raw, 1)
% figure (7)
% img_raw = squeeze(img_raw(n,:,:));
% image(img_raw);
% BW_canny = edge(img_raw, 'canny', 0.21);
% BW_canny_inv = imcomplement(BW_canny);
% imshow(BW_canny_inv);
% axis off
% title ''
% box off
% fname = ['\edge_ ' num2str(n)];
% saveas.figure(7),[ImageDir fname], 'fig')
% saveas.figure(7),[ImageDir fname], 'png')
% end
%
% %initialize 1st picture of masking manually for n=1:
% fname = ['\edge_1'];
% img_masked = imread([ImageDir fname '.png']);
%
% %continue masking with n=2:
% for n = 2:size(img_raw, 1)
% fname = ['\edge_ ' num2str(n)];
%
% img = imread([ImageDir fname '.png']); % use image to overlay
% bw = sum(img,3) > 400; % convert to a binary image
% mask = cast(bw, class(img)); % ensure the types are compatible
% img_masked = img_masked .* repmat(mask, [1 1 3]); % apply the mask
%
% %plot masked image
% figure (8)
% imshow(img_masked);
% axis off
% title ''
% box off
% end
%
% %save masked image
% saveas.figure(8),[ImageDir '\edges_total'],'fig')
% saveas.figure(8),[ImageDir '\edges_total'], 'png')

%%%%%%%%%%%%% Average binalization %%%%%%
%%%%% Need to work on it!!!! %%%%%%
figure(9)
hold on
subplot(2,1,1)
img_mean = squeeze(mean(img_raw));
I = img_mean;
BW = im2bw(I); %Binalisierung function

image(img_mean)
colormap(map);
subplot(2,1,2)
imshow(BW)
% imshowpair(I,BW,'montage'); %shows pictures together for comparison
saveas.imshow(BW),[ImageDir '\Average binalization'],'fig')
saveas.imshow(BW),[ImageDir '\Average binalization'], 'png')

% a='Average - test9 - TCW-1-20-34-10-2 OH';
% b='NL 2.6';
% for i=1:length(a)
% I = imread(a{i}), 'png'); %read image
% level = graythresh(I); %calculates graythresh value
% BW = im2bw(I, 0.05); %Binalisierung function
% imshowpair(I,BW,'montage'); %shows pictures together for comparison
% saveas(gcf,[C:\Users\maiaf\Documents\MATLAB\out\ b{i}], 'png')
% saveas(gcf,[C:\Users\maiaf\Documents\MATLAB\out\ b{i}], 'jpeg')

```

```

% saveas(gcf,['C:\Users\maiaf\Documents\MATLAB\out' b(i)],'fig')
% %close gcf
% end

%%%%%%%%%%%%% Plot in different axial distances %%%%%%
figure (10)
graph1 = plot(img_mean(:,[10,30,50,70,90,110,130,150,170,190,210,230,250,260]));figure(gcf);

set(graph1,'LineWidth',2);
legend('show');

saveas(figure(10),[ImageDir '\Average Axial distances','fig'])
saveas(figure(10),[ImageDir '\Average Axial distances'], 'png')

% figure (11)
% graph1 = plot(img_mean(:,[50,106,171,235,300,366,429,454]));figure(gcf);
% set(graph1,'LineWidth',2);
% legend('show');

% saveas(figure(11),[ImageDir '\Average Axial distances for Plot'],'fig')
% saveas(figure(11),[ImageDir '\Average Axial distances for Plot'], 'png')

%%%%%%%%%%%%% Plot in different radial distances %%%%%%
% %OH EMISSION
% $2.1
% figure (11)
% plot(img_mean(68,:));figure(gcf);
% legend('show');
% saveas(figure(11),[ImageDir '\Average radial distance 68'],'fig')
% saveas(figure(11),[ImageDir '\Average radial distance 68'], 'png')
%
% figure (12)
% plot(img_mean(118,:));figure(gcf);
% legend('show');
% saveas(figure(12),[ImageDir '\Average radial distance 118'],'fig')
% saveas(figure(12),[ImageDir '\Average radial distance 118'], 'png')
%
% $2.6
% figure (11)
% plot(img_mean(54,:));figure(gcf);
% legend('show');
% saveas(figure(11),[ImageDir '\Average radial distance 54'],'fig')
% saveas(figure(11),[ImageDir '\Average radial distance 54'], 'png')
%
% figure (12)
% plot(img_mean(103,:));figure(gcf);
% legend('show');
% saveas(figure(12),[ImageDir '\Average radial distance 103'],'fig')
% saveas(figure(12),[ImageDir '\Average radial distance 103'], 'png')

% $3.0
% figure (11)
% plot(img_mean(57,:));figure(gcf);
% legend('show');
% saveas(figure(11),[ImageDir '\Average radial distance 57'],'fig')
% saveas(figure(11),[ImageDir '\Average radial distance 57'], 'png')
%
% figure (12)
% plot(img_mean(105,:));figure(gcf);
% legend('show');
% saveas(figure(12),[ImageDir '\Average radial distance 105'],'fig')
% saveas(figure(12),[ImageDir '\Average radial distance 105'], 'png')

% $3.4
% figure (11)
% plot(img_mean(61,:));figure(gcf);
% legend('show');
% saveas(figure(11),[ImageDir '\Average radial distance 61'],'fig')
% saveas(figure(11),[ImageDir '\Average radial distance 61'], 'png')
%
% figure (12)
% plot(img_mean(110,:));figure(gcf);
% legend('show');
% saveas(figure(12),[ImageDir '\Average radial distance 110'],'fig')
% saveas(figure(12),[ImageDir '\Average radial distance 110'], 'png')

%%%%%%%%%%%%% Shading correction %%%%%%
%%%%%%%%%%%%% Need to work on it!!!! %%%%%%
%
% img_raw = squeeze(img_raw(5,:,:));
% I_backg = imopen(img_raw,strel('disk',15));
%
% % Display the Background Approximation as a Surface
% figure(13)
% surf(double(I_backg(1:8:end,1:8:end)),zlim([0 255]);
% ax = gca;
% ax.YDir = 'reverse';
%
% figure (14)
% img_corr = img_raw - I_backg;
% imshow(img_corr);

%
% img_mean = squeeze(mean(img_raw));
% I_backg = imopen(img_mean,strel('disk',20));
%
% % Display the Background Approximation as a Surface
% figure(13)
% surf(double(I_backg(1:8:end,1:8:end)),zlim([0 255]);
% ax = gca;
% ax.YDir = 'reverse';
%
% figure (14)
% img_corr = img_mean - I_backg;
% imshow(img_corr);

%
% function [img_norm] = normalizeImage(img)
% img_norm = 255*(img - min(min(img))) / ( max(max(img)) - min(min(img)) );
%
toc
end

```

Image Analysis

Matlab® SUPPORT FUNCTIONS

(v1.0 – November 2018)

Giuseppe Scarlatella

```
%%%%% Matlab® Image Analysis Routine (v1.0 November 2018) %%%%
%%%%% Support Functions (v1.0 November 2018) %%%%
%%%%% Created by: Giuseppe Scarlatella
%%% Date: 05/11/2018
%%% Version: 1.0
%%% Edited by: ...
%%% Software based on Matlab® 2017a (Image Processing Toolbox™ required)
%%% Software system based on Windows 10 OS (Apple and Linux compatible)
%%% Terms of Use: all intellectual rights reserved to Lehrstuhl für
%%% Turbomaschinen und Flugantriebe - Fakultät für Maschinenwesen - TUM.
%%% Any resource relative to the Matlab® Image Analysis Routine must be
%%% shared under specific authorization by Lehrstuhl für Turbomaschinen
%%% und Flugantriebe - Fakultät für Maschinenwesen - TUM and for specific
%%% research purposes only. Any other purpose will violate this Terms of
%%% Use and TUM policies. Any software material, including developer tools
%%% and sample code (collectively "Software"), are subject to this specific
%%% Terms of Use, unless TUM has provided those items to final user under
%%% more specific terms, in which case, those more specific terms will
%%% apply to the relevant item.
```

```

function [ diam ] = find_circle_diameter( M )
% FIND CIRCLE DIAMETER (Scarlatella)
% FIND CIRCLE DIAMETER This Matlab script finds the diameter [px] of a
% circular image in a matrix.
%%% This routine determines the diameter on a well exposed image that shows
%%% values of brightness different from 0 only in the circular shaped
%%% region of interest.

MEAN = mean_centre( M );
lowlight = 0.5*MEAN;

p=zeros(size(M,1),1);
for m = 1:size(M,1)
    for n = 1:size(M,2)
        if M(m,n) > lowlight
            p(m) = p(m)+1;
        end
    end
end

diam = max(p);

if diam < size(M,1)/2
    warning('Image Exposure is too low to evaluate pixel scale!');
end

end

function [ i,j ] = find_matrix_element( X,value )
% FIND MATRIX ELEMENT (Scarlatella)
% FIND MATRIX ELEMENT This Matlab script finds a specific element of a
% matrix.
flag = 0;
for i=1:length(X(:,1))
    for j=1:length(X(1,:))
        if X(i,j) == value
            flag = flag+1;
            display(['Element N',num2str(flag), ' indexes: ', ...
                num2str(i), ', ',num2str(j)]);
        end
    end
end

if flag == 0
    disp('No element found.');
else
    display(['N' ' of elements found: ',num2str(flag)]);
end

end

function M = flip_matrix( M,xVERSE,yVERSE )
% FLIP MATRIX (Scarlatella)
% FLIP MATRIX This Matlab script flips a matrix in a specific direction.
%%% Direction is specified by 'xVERSE' and 'yVERSE'. In case that 'yVERSE'
%%% is different from "Null", all flips along x-axis will be lost.

if xVERSE == "Left"
    disp('Integration along x-axis (L->R).');
elseif xVERSE == "Right"
    M = fliplr(M);
    disp('Integration along x-axis (R->L).');
else
    warning('Wrong xVERSE input!');
end

if yVERSE == "Up"
    M = rot90(M);
    disp('Integration along y-axis (U->B).');
    warning('Every integration info along x-axis will be lost.');
elseif yVERSE == "Bottom"
    M = rot90(M,-1);
    disp('Integration along y-axis (B->U).');
    warning('Every integration info along x-axis will be lost.');
elseif xVERSE == "Null" && yVERSE == "Null"
    disp('NO integration along specific axes.');
elseif yVERSE ~= "Up" && yVERSE ~= "Bottom" && yVERSE ~= "Null"
    warning('Wrong yVERSE input!');
end

end

function L = flowfield( x,Extension,xVERSE,yVERSE,step,Geometry,Flowfield_depth,flowfield_min_depth,xx )
% FLOWFIELD (Scarlatella)
% FLOWFIELD This Matlab script evaluates flowfield depth L(x) along
% the optical axis (z-axis).
%%% Flowfield depth along optical axis L(x) is generally a function of
%%% coordinate along which the integration process is conducted.
%%% Casistic:
%%%     - "AllDomain" Object: domain is a parallelepiped which depth is a
%%%         constant flowfield depth;
%%%     - "Uniform" Object: restricts "AllDomain" only to the region of
%%%         interest;
%%%     - "Linear" Object: depth from "Uniform" is no more constant, but
%%%         follows a linear law;
%%%     - "Diamond" Object: same as "Linear", but follows an increasing
%%%         linear law on first half of interval, maximum value at centre of
%%%         interval and then decreasing linear law;
%%%     - "Sinusoidal" Object: depth from "Uniform" is no more constant, but
%%%         follows a sinusoidal law;
%%%     - "Cylindric" Object: same as "Sinusoidal", but follows an
%%%         increasing sinusoidal law on first half of interval, maximum value
%%%         at centre of interval and then decreasing sinusoidal law;
%%% (NOTE: sin([0,90])_meanvalue = cos([0,90])_meanvalue = 0.6366)
if yVERSE == "Bottom"
    Xstart = Extension(1);
    Xend = Extension(2);
elseif yVERSE == "Up"
    Xstart = xx(end)/step-Extension(2);
    Xend = xx(end)/step-Extension(1);
elseif yVERSE == "Null"
    if xVERSE == "Left"
        Xstart = Extension(1);
        Xend = Extension(2);
    elseif xVERSE == "Right"

```

```

Xstart = xx(end)/step-Extension(2);
Xend = xx(end)/step-Extension(1);
end

if Flowfield_depth == 0
    L_x = abs(Xstart-Xend); %[px]
else
    L_x = Flowfield_depth; %[m]
    L_x = L_x/step; %[px]
end

cost_thick = flowfield_min_depth; % Extension of the flowfield along the integration axis
% (evaluates the interest zone extension)

if Geometry == "AllDomain"
    L = L_x*step;
elseif Geometry == "Uniform"
    L = interp1([xx(1) Xstart*step,... % Costant flowfield thickness for avoid numerical errors
    (Xstart + 0.1*abs(Xstart-Xend))*step,... % ('cost_thick' as to be as low as possible to reach
    (Xend - 0.1*abs(Xstart-Xend))*step,... % realistic numerical values).
    Xend*step xx(end)]...,[cost_thick cost_thick L_x*step L_x*step cost_thick cost_thick],x);
elseif Geometry == "Linear"
    L = interp1([xx(1) Xstart*step Xend*step xx(end)],...,[cost_thick 2*cost_thick L_x*step L_x*step],x);
elseif Geometry == "Diamond"
    L = interp1([xx(1) Xstart*step abs(Xend-Xstart)/2 Xend*step xx(end)],...,[cost_thick 2*cost_thick L_x*step 2*cost_thick cost_thick],x);
elseif Geometry == "Sinusoidal"
    if x <= Xstart*step
        L = cost_thick;
    elseif x > Xstart*step && x <= Xend*step
        L = sin(deg2rad((x/Xstart)/(abs(Xstart-Xend)))*90)) * ...
            L_x*step + cost_thick;
    elseif x > Xend*step
        L = L_x*step + cost_thick;
    end
elseif Geometry == "Cylindric"
    if x <= Xstart*step
        L = cost_thick;
    elseif x >= Xstart*step && x <= Xend*step
        L = sin(deg2rad((x/Xstart)/(abs(Xstart-Xend)/2))*90)) * ...
            L_x*step + cost_thick;
    elseif x > Xend*step
        L = cost_thick;
    else
        L = cost_thick;
    end
else
    warning('Geometry input value not allowed!');
end

```

```

function [lux_M,MAX] = light_scaling( M,MEAN )
%% LIGHT SCALING (Scarlatella)
% LIGHT SCALING This Matlab scripts scales a [0,1] matrix in a [lux] values
% matrix.
%%% This routine extracts a mean value from a centred squared region of a
%%% reference image. Updates the 'HighLightLIM' to a plausible value that
%%% matches a mean value corresponding to the 'AverageLight' value referred
%%% to the CMOS Camera model (see 'schlieren_data.m').

%%% Exposure Evaluation %%
MEAN = (mean_centre(M) / (1/2)) *MEAN;

%%% Light Scaling %%
MAX = 5*MEAN;
meanVALUE = mean_centre(M) *MAX;

while abs(meanVALUE-MEAN) > 1e-6
    MAX = MAX*(MEAN/meanVALUE);
    meanVALUE = mean_centre(M)*MAX;
end

lux_M = M*MAX;

```

```

function [lux_M,lux_Mref,MAX] = light_scaling_alternative_slow_version( M,Mref,MIN,MEAN )
%% LIGHT SCALING (Scarlatella)
% LIGHT SCALING This Matlab scripts scales a [0,1] vector in a [lux] values
% vector.
%%% This routine extracts a mean value from a centred squared region of a
%%% reference image. Updates the 'HighLightLIM' to a plausible value that
%%% matches a mean value corresponding to the 'AverageLight' value referred
%%% to the CMOS Camera model (see 'schlieren_data.m'). 'LowlightLIM' is
%%% also referred to lowest [lux] level sensitivity of CMOS Camera model
%%% (see 'schlieren_data.m').

MAX = 5*MEAN;
X = linspace(MIN,MAX,1e3);
meanVALUE = lighting(mean_centre(Mref),X,MAX);

while abs(meanVALUE-MEAN) > 1e-6
    MAX = MAX*(MEAN/meanVALUE);
    X = linspace(MIN,MAX,1e3);
    meanVALUE = lighting(mean_centre(Mref),X,MAX);
end

lux_M = lighting_alternative_slow_version(M,X,MAX);
lux_Mref = lighting_alternative_slow_version(Mref,X,MAX);

end

```

```

function [lux_M] = lighting_alternative_slow_version( M,X,MAX )
%% LIGHTING (Scarlatella)
% LIGHTING This Matlab script specifically scales a [0,1] vector in a
% [lux] values vector, without any assumptions on 'HighLightLIM' from CMOS
% Camera model.
%%% This routine finds a correspondance between [0,1] values from a matrix
%%% and X=[LowLightLIM,HighLightLIM] from a vector.

for m = 1:length(M(:,1))
    for n = 1:length(M(1,:))
        k=1;
        if M(m,n) >= 0
            while (X(k)/MAX) < M(m,n)

```

```

        k=k+1;
    end
    lux_M(m,n) = X(k);
else
    while (X(k)/MAX) < abs(M(m,n))
        k=k+1;
    end
    lux_M(m,n) = -X(k);
end
end
end

```

```

function [ M_int ] = matrix_integral( M,step,x_BCs,xVERSE,y_BCs,yVERSE)
%% MATRIX INTEGRAL (Scarlatella)
% MATRIX INTEGRAL This Matlab script integrates a derivatives matrix along
% a specific axis (and direction) in order to obtain the analytic function
% point-by-point matrix.
%%% This script works only for Taylor 1ST Term Expansion method.

%%% Integration Process along x-axis %%%
if xVERSE == "Left"
    M_int = zeros(size(M,1),size(M,2));
    M_int(:,1) = x_BCs;
    for m=1:size(M,1)
        for n=2:size(M,2)
            M_int(m,n) = M_int(m,n-1) + M(m,n-1)*step;
        end
    end
    if yVERSE == "Null"
        disp('Integration Process along X-Axis');
        disp('([Left->Right])');
    end
elseif xVERSE == "Right"
    M_int = zeros(size(M,1),size(M,2));
    M_int(:,1) = x_BCs;
    [M] = matrix_invert_coloumns(M);
    for m=1:size(M,1)
        for n=2:size(M,2)
            M_int(m,n) = M_int(m,n-1) - M(m,n-1)*step;
        end
    end
    [M_int] = matrix_invert_coloumns( M_int );
    if yVERSE == "Null"
        disp('Integration Process along X-Axis');
        disp('((Right->Left))');
    end
end
elseif xVERSE == "Null"
    if yVERSE == "Null"
        warning('NO INTEGRATION direction selected!');
        M_int = zeros(size(M,1),size(M,2)); M_int(1,1) = Inf;
    end
end

```

```

%%% Integration Process along y-axis %%%
if yVERSE == "Top"
    M_int = zeros(size(M,2),size(M,1));
    M_int(1,:) = y_BCs;
    for n=1:size(M,2)
        for m=2:size(M,1)
            M_int(m,n) = M_int(m-1,n) + M(m-1,n)*step;
        end
    end
    disp('Integration Process along Y-Axis');
    disp('((Top->Bottom))');
    if xVERSE ~= "Null"
        warning('Integration REQUEST along x-axis will be ignored.');
    end
elseif yVERSE == "Bottom"
    M_int = zeros(size(M,2),size(M,1));
    M_int(1,:) = y_BCs;
    [M] = matrix_invert_raws(M);
    for n=1:size(M,2)
        for m=2:size(M,1)
            M_int(m,n) = M_int(m-1,n) - M(m-1,n)*step;
        end
    end
    [M_int] = matrix_invert_raws( M_int );
    disp('Integration Process along Y-Axis');
    disp('((Bottom->Top))');
    if xVERSE ~= "Null"
        warning('Integration REQUEST along x-axis will be ignored.');
    end
end

```

```

function [ M2 ] = matrix_invert_coloumns( M1 )
%% MATRIX INVERT COLOUMNS (ScarlateLLA)
% MATRIX INVERT COLOUMNS This Matlab script inverts coloumns on a matrix.

M2 = zeros(size(M1,1),size(M1,2));
for m = 1:size(M1,1)
    for n = 1:size(M1,2)
        M2(m,end-(n-1)) = M1(m,n);
    end
end

```

```

function [ M2 ] = matrix_invert_raws( M1 )
%% MATRIX INVERT RAWs (ScarlateLLA)
% MATRIX INVERT RAWs This Matlab script inverts raw on a matrix.

M2 = zeros(size(M1,1),size(M1,2));
for m = 1:size(M1,1)
    for n = 1:size(M1,2)
        M2(end-(m-1),n) = M1(m,n);
    end
end

```

```

function [ max ] = maximum( X )
%% MAXIMUM (Scarlatella)
% MAXIMUM This Matlab script evaluates maximum value on row and coloums
% of a matrix.
%%% Works only with matrices!

max = X(1,1);
for p=1:length(X(:,1))
    for q=1:length(X(1,:))
        if max < X(p,q)
            max = X(p,q);
        end
    end
end

```

```

function [ meanVALUE ] = mean_centre( M )
%% MEAN CENTRE (Scarlatella)
% MEAN CENTRE This Matlab Script evaluates mean value of a matrix in a
% squared centred region.
%%% Please select a 'regionSIZE' value between [0.001,0.999].

%%% INPUT %%%
regionSIZE = 0.2;                                % Select a size between [0.001,0.999]

if regionSIZE < 0.001
    warning('Wrong regionSIZE input! Please select a size between [0.005,0.49]');
elseif regionSIZE > 0.999
    warning('Wrong regionSIZE input! Please select a size between [0.005,0.49]');
end

%%% Routine %%%
firstpoint = (1/2) - regionSIZE/2;
lastpoint = (1/2) + regionSIZE/2;

Xinterval = [round(firstpoint*length(M(:,1))):...
    round(lastpoint*length(M(:,1)))];
Yinterval = [round(firstpoint*length(M(1,:))):...
    round(lastpoint*length(M(1,:)))];

meanVALUE = mean(mean(M(Xinterval,Yinterval)));

```

```

function [ min ] = minimum( X )
%% MINIMUM (Scarlatella)
% MINIMUM This Matlab script evaluates minimum value on row and coloums
% of a matrix.
%%% Works only with matrices!

min = X(1,1);
for p=1:length(X(:,1))
    for q=1:length(X(1,:))
        if min > X(p,q)
            min = X(p,q);
        end
    end
end

```

```

function [ normalized_X ] = normalize( X,MIN,MAX,ZERO )
%% NORMALIZE (Scarlatella)
% NORMALIZE This Matlab script normalizes a matrix or a vector in a [-1,1] field
%%% Works with both vectors or matrices. Gets a MIN and a MAX values to
%%% which normalize to and a ZERO value to refers as center of interval.

if MIN == ZERO                               % Verification on elements from X different from 0
    if MAX == ZERO
        normalized_X = X;
        return
    end
end

X0 = ZERO;
if abs(MAX-ZERO) >= abs(ZERO-MIN)
    normalized_X = ((X-X0)/abs(MAX-ZERO));
else
    normalized_X = ((X-X0)/abs(MIN-ZERO));
end

```

```

function density = ODE_rho_function(x,y,gldstn,n0,n_air,xx,C,Geometry,Extension,Flowfield_depth,flowfield_min_depth,step,xVERSE,yVERSE)
%% ODE RHO FUNCTION (Scarlatella)
% ODE RHO FUNCTION This Matlab script solves differential equation for
% density along specified axes (and direction).
%%% This script integrates density along L(x) (flow depth field along
%%% optical z-axis), which generally is a function of the coordinate along
%%% which the integration process is conducted.
L = @(x) flowfield( x,Extension,xVERSE,yVERSE,step,Geometry,Flowfield_depth,flowfield_min_depth,xx );

if yVERSE == "Up"
    density = [ (1/gldstn)*(1/L(x))*(n_air/n0)*C(x); y(2) ];
elseif yVERSE == "Bottom"
    density = [ -(1/gldstn)*(1/L(x))*(n_air/n0)*C(x); y(2) ];
else
    if xVERSE == "Left"
        density = [ (1/gldstn)*(1/L(x))*(n_air/n0)*C(x); y(2) ];
    elseif xVERSE == "Right"
        density = [ -(1/gldstn)*(1/L(x))*(n_air/n0)*C(x); y(2) ];
    else
        warning('Integration Process could not take place.');
    end
end

```

```

function temperature = ODE_T_function(x,y,p,gldstn,n0,n_air,R,xx,C,Geometry,Extension,Flowfield_depth,flowfield_min_depth,step,xVERSE,yVERSE)
% ODE T FUNCTION (Scarlatella)
% ODE T FUNCTION This Matlab script solves differential equation for
% temperature along specified axes (and direction).
%%% This script integrates temperature along L(x) (flow depth field along
%%% optical z-axis), which generally is function of the coordinate along
%%% which the integration process is conducted.
L = @(x) flowfield( x,Extension,xVERSE,step,Geometry,Flowfield_depth,flowfield_min_depth,xx );

```

```

if yVERSE == "Up"
    temperature = [ -(1/gldstn)*(1/L(x))*R/p*(n_air/n0)*C(x).*y(1).^2 ; y(2) ];
elseif yVERSE == "Bottom"
    temperature = [ (1/gldstn)*(1/L(x))*R/p*(n_air/n0)*C(x).*y(1).^2 ; y(2) ];
else
    if xVERSE == "Left"
        temperature = [ -(1/gldstn)*(1/L(x))*R/p*(n_air/n0)*C(x).*y(1).^2 ; y(2) ];
    elseif xVERSE == "Right"
        temperature = [ (1/gldstn)*(1/L(x))*R/p*(n_air/n0)*C(x).*y(1).^2 ; y(2) ];
    else
        warning('Integration Process could not take place.');
    end
end
end

```

```

function [ scaled_X ] = scale( X,MIN,MAX )
% SCALE (Scarlatella)
% SCALE This Matlab script scales a matrix or a vector in a [-1,1] field.
%%% Works with both vectors or matrices. Gets a MIN and a MAX values to
%%% which scale to.
scaled_X = (2.* (X-MIN)./(MAX-MIN))-1;
end

```

```

function M = unflip_matrix( M,xVERSE,yVERSE )
% UNFLIP MATRIX (Scarlatella)
% UNFLIP MATRIX This Matlab script unflips a matrix in specific direction.
%%% Direction is specified by 'xVERSE' and 'yVERSE'. In case that both
%%% 'xVERSE' and 'yVERSE' are different from "Null", program displays a
%%% warning message.
if xVERSE == "Right"
    M = fliplr(M);
elseif xVERSE ~= "Left" && xVERSE ~= "Right" && xVERSE ~= "Null"
    warning('Wrong xVERSE input!');
end
if xVERSE ~= "Null" && yVERSE ~= "Null"
    warning('Attempt to integrate in both directions FAILED.');
end
if yVERSE == "Up"
    M = rot90(M,-1);
elseif yVERSE == "Bottom"
    M = rot90(M);
elseif yVERSE ~= "Up" && yVERSE ~= "Bottom" && yVERSE ~= "Null"
    warning('Wrong yVERSE input!');
end
end

```

```

function [ X0 ] = x_BCs_reader( BCs,IMG,rho0 )
% X BCs READER (Scarlatella)
% X BCs READER This Matlab script imports Boundary Conditions along x-axis.
if BCs == rho0 && size(BCs,2) == 1
    X0 = BCs.*ones(size(IMG,1),1);
elseif size(BCs,2) ~= 1
    X0 = BCs;
elseif size(BCs,1) ~= 1
    warning('On x-axis integration, "BCs" must be a column vector!')
end

```

```

function [ Y0 ] = y_BCs_reader( BCs,IMG,rho0 )
% Y BCs READER (Scarlatella)
% Y BCs READER This Matlab script imports Boundary Conditions along y-axis.
if BCs == rho0 && size(BCs,1) == 1
    Y0 = BCs.*ones(size(IMG,2),1);
elseif size(BCs,1) ~= 1
    Y0 = BCs;
elseif size(BCs,2) ~= 1
    warning('On y-axis integration, "BCs" must be a raw vector!')
end

```
