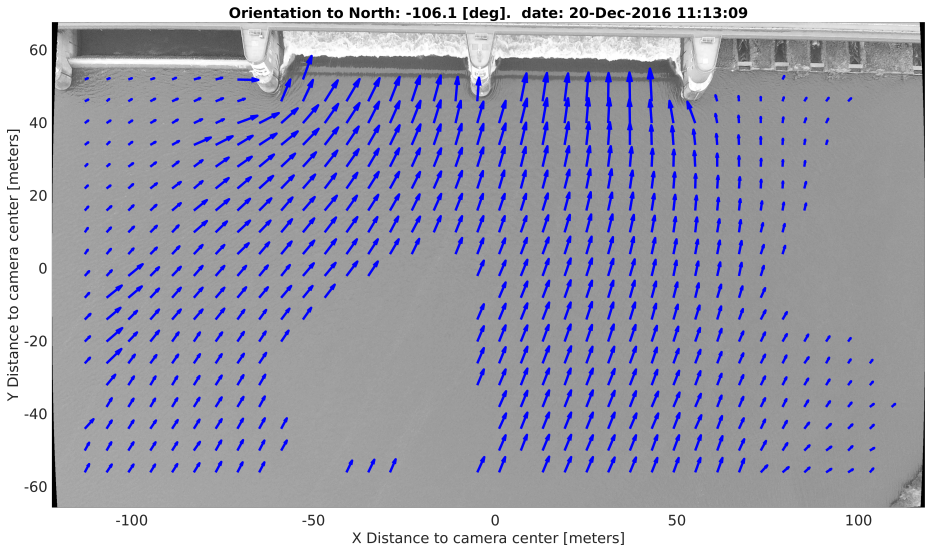
CopterCurrents

V.1.0.04



Helmholtz-Zentrum Geesthacht
Zentrum für Material- und Küstenforschung

Ruben Carrasco

1-07-2019

Contents

Chapter 1

Introduction

The CopterCurrents tool is designed to extract water surface current fields (flow speeds) from Unmanned Aerial Vehicles (UAV) high definition videos. The CopterCurrents measurements are based in the waves dispersion relation properties, and the physical phenomena involves in the measurement process are described in the following publication:

M. Streßer, R. Carrasco and J. Horstmann, "Video-Based Estimation of Surface Currents Using a Low-Cost Quadcopter," in IEEE Geoscience and Remote Sensing Letters, vol. 14, no. 11, pp. 2027-2031, Nov. 2017. doi: 10.1109/LGRS.2017.2749120 URL: `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8049342&isnumber=8082817`

Before reading CopterCurrents user manual, please the read this previous work, where all the concepts and physical phenomena are described. CopterCurrents user manual is focused in the installation and software functionality, not in the physical process involved in the current measurements.

From the signal processing point of view CopterCurrents is divided in 5 steps:

1. Record a video of the water surface by a UAV camera. The AUV camera should be provided of a stabilization unit (gimbal) to prevent jittering effects on the video. The camera must point perpendicular to the water (nadir) and the Drone must be static during the video recording. The superficial waves must be clearly visible on the video, otherwise CopterCurrents will not be able to fit the waves dispersion relation in further steps. The Drone altitude must stored in the video meta-data (DJI drones do this step automatically). Check figure 1.1.
2. Video rectification: The images from the video are rectified to real world coordinates.
3. Slice the rectified video in several squared regions, and set the fit parameters.
4. Fit the most probable current, applying the wave dispersion relation (a spectral energy-based maximization technique) in every squared region.

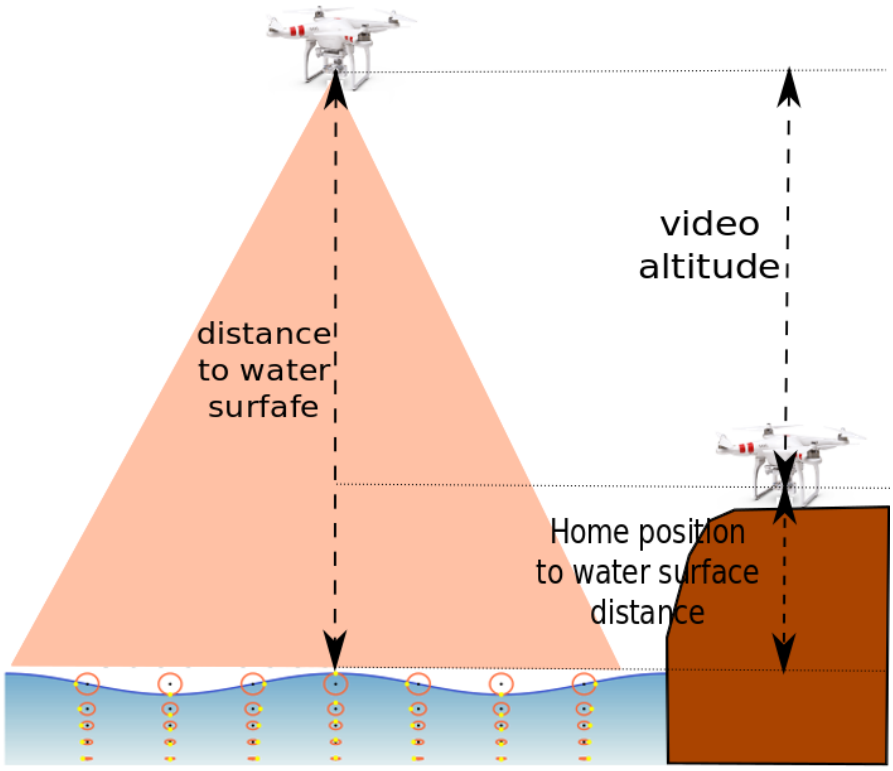


Figure 1.1: Drone recording video, the distance to the water surface is the sum off the video altitude plus the distance from the drone home position to the water surface. The picture was generating merging [1] and [2] .

5. Filter obtained currents with Signal to noise Ratio, and plot currents in Camera reference grid or UTM coordinates.

CopterCurrents has been developed in Department of Radar Hydrography, Helmholtz-Zentrum Geesthacht Centre for Materials and Coastal Research.

Chapter 2

Installation

2.1 Requirements

2.1.1 MATLAB® (mandatory)

CopterCurrents is a toolbox designed under MATLAB® environment, and requires a MATLAB® software license, which can be obtained by purchasing MATLAB® product or downloading a product trial. For more information:

`https://www.mathworks.com/help/install/index.html`

CopterCurrents does not require any extra MATLAB® toolbox, but the 'Parallel Computing Toolbox' speeds up the current fit calculations. The 'Parallel Computing Toolbox' license is optional, not mandatory.

2.1.2 Drone video Codecs (mandatory)

CopterCurrents reads the drone video frames using the MATLAB 'videoreader', which sometimes requires specific codecs. For more information about the compatible codecs:

`https://www.mathworks.com/help/matlab/import_export/supported-video-file-formats.html`

2.1.3 Mediainfo (mandatory)

CopterCurrents needs mediainfo application to read video metadata information. The video metadata information contains the the drone position (longitude, latitude and altitude), camera heading, pitch and roll angles, which are necessary to rectify the video and obtain the real world coordinates. CopterCurrents needs the command 'mediainfo' accessible from operating system terminal. MediaInfo is available in the following URLs:

MediaInfoLib - v0.7.82
`https://mediaarea.net/en/MediaInfo`

`https://mediaarea.net/en/MediaInfo/Download/Ubuntu`
`https://mediaarea.net/download/binary/mediainfo/18.12/MediaInfo_CLI_18.12_Windows_x64.zip`

Note: In windows operating system, install the MediaInfo CLI version (Command Line Interface) and add the MediaInfo folder to system path.

2.1.4 External MATLAB® library 'deg2utm' (optional)

The MATLAB® library 'deg2utm' is an external library used to reference the CopterCurrents current maps to the Universal Transverse Mercator coordinate system (UTM). The library is not mandatory to run CopterCurrents. 'deg2utm' is available in the mathworks filexchange repository:

`https://www.mathworks.com/matlabcentral/fileexchange/10915-deg2utm`

Author: Rafael Palacios, Universidad Pontificia Comillas, Madrid, Spain.

After download the file, unzip the content in the following folder:

../CopterCurrents/external_libraries/deg2utm/

2.1.5 External MATLAB® library 'Camera Calibration Toolbox' (optional)

The Caltech 'Camera Calibration Toolbox' for MATLAB® is an external library which allows to calibrate cameras. Caltech library is also implemented in OpenCV, so adding the Caltech library to the MATLAB path, let CopterCurrents to use OpenCV code for camera calibration. The 'Camera Calibration Toolbox' is optional, CopterCurrents provide his own camera calibration method (Field Of View calibration).

Download url: `https://doi.org/10.22002/D1.20164`

Jean-Yves Bouguet (2022). Camera Calibration Toolbox for Matlab (1.0). CaltechDATA.

Download the file *calib_doc.zip*, then unzip the content of the file *toolbox_calib_2015_10_14.zip* in the following folder:

../CopterCurrents/external_libraries/TOOLBOX_calib/

A 'Camera Calibration Toolbox' example is provided by CopterCurrents:

../Calibration_examples/Matlab_Caltech/Phantom3_3840

Note: be sure that the paths to the external libraries are included in matlab path. The script *add_CopterCurrents_matlab_path.m* add automatically the paths if the libraries are located in the defalults folders locations.

2.1.6 OpenCV (optional)

OpenCV is the most spread 'Computer vision' open-source platform for Photogrammetry applications. Thanks to the 'Camera Calibration Toolbox' CopterCurrents is able to use camera calibrations performed by OpenCV. CopterCurrents provides a python script to perform camera calibration by OpenCV libraries. For more information regarding OpenCV:

`https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_calibration/py_calibration.html`

`https://docs.opencv.org/master/d9/df8/tutorial_root.html`

A camera calibration file, obtained by 'OpenCV' is provided by CopterCurrents:

../Calibration_examples/OpenCV/Phantom3_3840

2.2 CopterCurrents installation

1. Install the requirements described in the 2.1:
   (a) MATLAB® (mandatory)2.1.1
   (b) Drone video Codecs (mandatory) 2.1.2
   (c) Mediainfo (mandatory) 2.1.3
   (d) External MATLAB® library 'deg2utm' (optional)
   (e) External MATLAB® library 'Camera Calibration Toolbox' (optional) 2.1.5
   (f) OpenCV (optional) 2.1.6
2. From the Matlab environment, go to the CopterCurrents folder and run the script *add_CopterCurrents_matlab_path.m* to add the CopterCurrents folders to the MATLAB path. Then, MATLAB displays the following messages, if the external libraries are not available:

```
-------------------------------------------------------------------------------------------------------------
MATLAB Version: 9.6.0.1072779 (R2019a)
MATLAB License Number: 932330
Operating System: Linux 4.15.0-47-generic #50~16.04.1-Ubuntu SMP Fri Mar 15 16:06:21 UTC 2019 x86_64
Java Version: Java 1.8.0_181-b13 with Oracle Corporation Java HotSpot(TM) 64-Bit Server VM mixed mode
-------------------------------------------------------------------------------------------------------------
MATLAB                                                Version 9.6         (R2019a)

mediainfo --Version: Signal 127
MediaInfo Command line,
MediaInfoLib - v0.7.82

nanmean function found.
nansum  function found.

-----------------------------------------------
Checking optional libraries:
-----------------------------------------------
deg2utm: NOT FOUND (Optional)
Camera Calibration Toolbox: NOT FOUND (Optional)
```

If the external libraries are available, the displayed message will be:

```
------------------------------------------------------------------------------------------------
MATLAB Version: 9.6.0.1072779 (R2019a)
MATLAB License Number: 932330
Operating System: Linux 4.15.0-47-generic #50~16.04.1-Ubuntu SMP Fri Mar 15 16:06:21 UTC 2019 x86_64
Java Version: Java 1.8.0_181-b13 with Oracle Corporation Java HotSpot(TM) 64-Bit Server VM mixed mode
------------------------------------------------------------------------------------------------
MATLAB                                                Version 9.6         (R2019a)

mediainfo --Version: Signal 127
MediaInfo Command line,
MediaInfoLib - v0.7.82

nanmean function found.
nansum  function found.

-----------------------------------------------
Checking optional libraries:
-----------------------------------------------
deg2utm: AVAILABLE (Optional)
Camera Calibration Toolbox: AVAILABLE (Optional)
```

3. Final test: check the CopterCurrents toolbox with your own drone video. From MATLAB environment, go to the *test_scripts* folder in */CopterCurrents/test_scripts/* and run the script *read_Quadcopter_video_script.m*. Edit the line 40 *"video_fname = 'path to test video/test_video.MP4';"* with your own video filename. Then run the script; a frame will be displayed during 5 seconds and the video metadata information will be displayed in MATLAB terminal:

```
Metadata info obtained
Longitude :10.5573 deg
Latitude :53.3693 deg
Altitude :203.8 meters
time stamp: 04-Apr-2017 10:02:33
Camera heading:127.4 deg
Camera pitch:-90 deg
Camera roll :0 deg
```

Chapter 3

CopterCurrents

The CopterCurrents Toolbox is available in `https://github.com/RubenCarrascoAlvarez/CopterCurrents`. A drone video over the Elbe river is available in `https://doi.org/10.5281/zenodo.3088437`.

3.1 CopterCurrents folder structure

The figure 3.1 shows the CopterCurrents project content. The folder *Calibration_examples* contains images examples for camera calibration. The folder *CopterCurrents* contains all MATLAB source code files, the content is displayed in the figure 3.2. The code folder structure is:

- *CopterCurrents_CameraCalib*: folder containing the camera calibration files.
- *external_libraries*: folder containing the extrenal MATLAB code libreries, as *deg2utm* and *Camera Calibration Toolbox*.
- *extra*: folder containing auxiliar librareis like *nanmean.m* and *nansum.m*.
- *test_scripts*: Folder containing user scripts. The user should work and edit the scripts contained in this folder.

3.2 Run CopterCurrents

As is explained in the introduction chapter 1, CopterCurrents process the data in 5 steps, performed by the script *Run_CopterCurrents_script.m*. In this section all the steps are described one by one, feel free to edit the script parameters to your own video requirements. All the parameters are briefly commented in the script.

3.2.1 CopterCurrents step 1: video data selection

The video input parameters define which frames will be analyze. Due to RAM memory limitation, not all the frames can be processed. The camera calibration

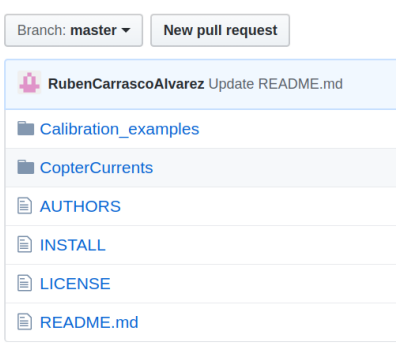


Figure 3.1: CopterCurrents project folder.

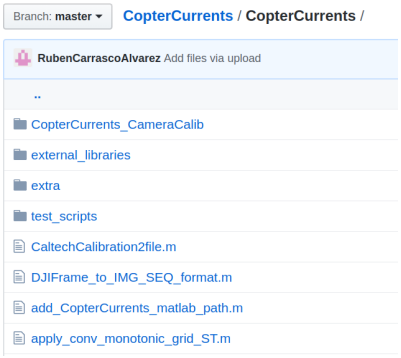


Figure 3.2: CopterCurrents folder distribution.

file, *CopterCurrents_CamCalib*, must be also selected in this step. The parameters to be edited in the script are:

- video_fname: file name corresponding to the Drone video.
- time_limits: time interval to be extracted from the the video, defined in seconds. In this example, the data extracted goes from second 5 to the second 35.
- dt: gap in seconds between extracted frames. Should be a multiple of the video frame rate. If the selected *dt* is not a multiple of the video frame rate, CopterCurrents will adjust the value to the nearest multiple.
- offset_home2water_Z: distance between the home point and the water surface in meters. This parameter is very important to obtain the effective distance from the camera to the water surface. The altitude stored in the video metadata is the altitude from the drone position to the drone home position. As is displayed in the figure 1.1, the distance from the drone camera to the water surface is the results to add the video altitude plus the distance from the drone home position to the water surface. When the home position is over water level the distance is sign is positive.
- CopterCurrents_calibration_filename: the *CopterCurrents_CamCalib* camera calibration file to be used for the video rectification. See section 3.4.

The Georeference Configuration structure contain all the video parameters previously described, which are:

- video_fname: video filename (complete path).
- dt: distance in seconds between frames.
- time_limits: Vector defining the video data to be used.
  [initial_time_video_sec end_time_video_sec].
- CopterCurrents_CamCalib: Camera calibration structure.

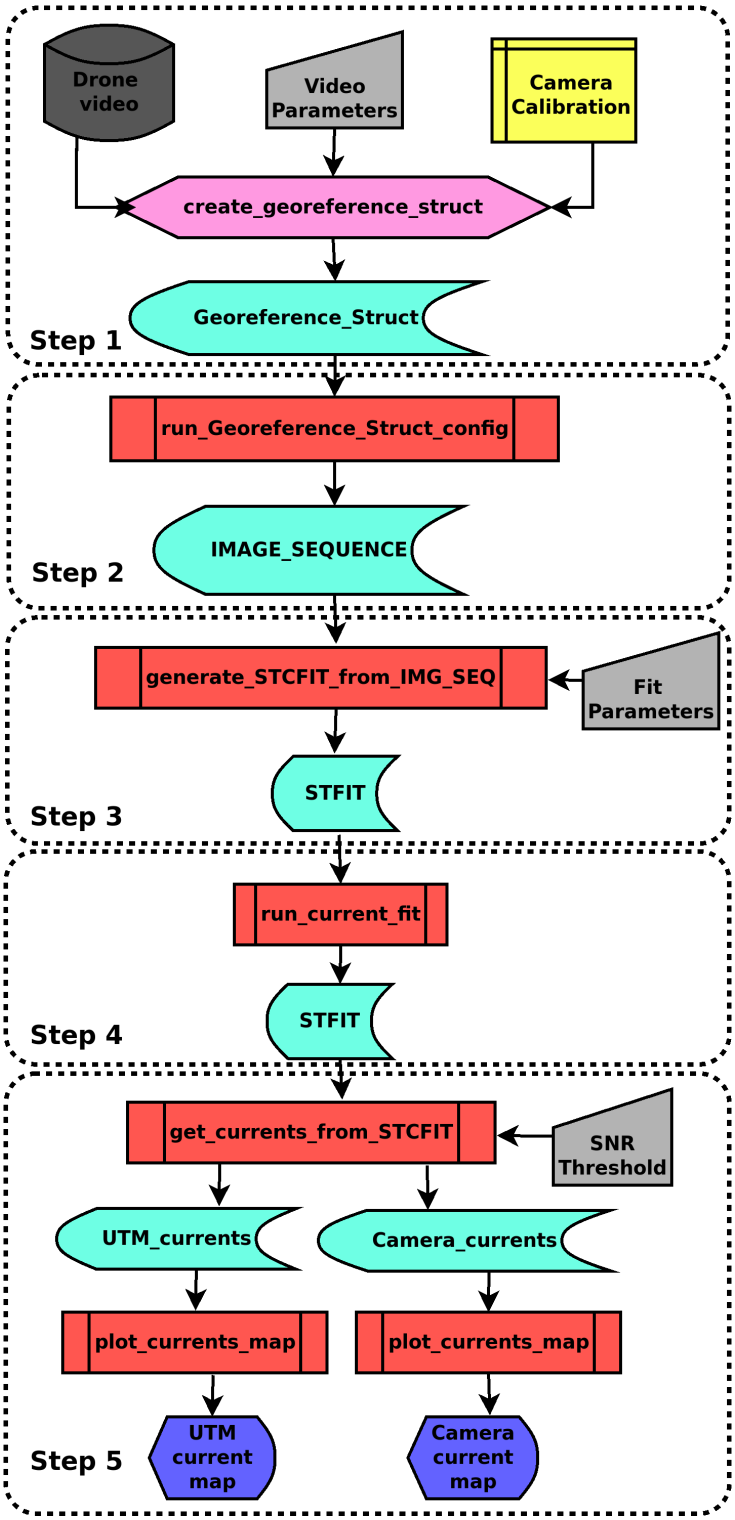


Figure 3.3: CopterCurrents flow diagram.

The code to run the step 1 in the script is:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% read Drone video - step 1
%%%%%%%%%%%%%%%%meters%%%%%%%%%%%%%%%

% video to be analysed
% video_fname = '/path2thevideo/Drone_video.MP4';
video_fname = '/.../Zenodo/20170404_over_Elbe.MP4';

% get video position in DJI drone
[CamPos_ST] = get_Camera_Position_Struct(video_fname);

% % Add data manually if is not a DJI drone
% CamPos_ST = struct('LONGITDE',LON,'LATITUDE',LAT,'Height',Height,...
%                    'timestamp',timestamp,'yaw',yaw,'pitch',pitch,...
%                    'roll',roll,'extra',mediainfo_string);

% time stamps to be used in the video [initial_time  end_time]
time_limits = [5 35];

% time step between frames in seconds
dt = 0.10;

% distance between the home point and the water surface in meters
offset_home2water_Z = 0.8;

% FOV calibration structure to be used
CopterCurrents_calibration_filename = ...
        '/CopterCurrents_CameraCalib/Phantom3_v1_FOV_3840x2160.mat';
% CopterCurrents_calibration_filename = ...
%        '/CopterCurrents_CameraCalib/Phantom3_v1_OpenCV_3840x2160.mat';


% Create a Georeference configuration structure
[Georeference_Struct_config] = create_georeference_struct(...
                video_fname, dt, time_limits, offset_home2water_Z, ...
                CopterCurrents_calibration_filename);
```

3.2.2 CopterCurrents step 2: video rectification

Run video rectification, which generates the image sequence structure. The image sequence structure contains all video frames projected in real size magnitude. The camera location defines the origin axis, so the coordinate (0,0,0) is located in the camera position. All the coordinates are relative to the camera position and the units are meters.

- IMG: 3D Intensty data (Nx,Ny,Nt).
- gridX: 2D grid X data in meters (Nx,Ny) monotonic in ascending order.
- gridY: 2D grid Y data in meters (Nx,Ny) monotonic in ascending order.
- dx: X resolution in meters.
- dy: Y resolution in meters.
- dt: time resolution in seconds.
- ts_video: video time stamps in datenum format, 1xNt vector monotonic in ascending order.

- altitude: altitude to the water surface in meters.
  altitude = video altitude + offset_home2water_Z
- pitch: video pitch (for Nadir pitch = -90).
- roll: video roll (for Nadir roll = 0).
- heading: video heading to North (yaw).
- Longitude: video longitude in degrees.
- mediainfo_string: raw mediainfo string
- Georeference_Struct_config: Georeference_Struct used.

The code to run the step 2 in the script is:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Rectify video and obtain REAL WORLD coordenates - step 2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Apply Georeference_Struct_config and retrieve the Image sequence
% corrected
IMG_SEQ = run_Georeference_Struct_config(Georeference_Struct_config);
```

3.2.3 CopterCurrents step 3: define dispersion relation fit constrains

This step divides the IMG_SEQ structure in several windows, where the current will be fitted in the next step. The fit windows and current fit parameters are setup by the function *generate_STCFIT_from_IMG_SEQ.m*, where the input parameters are:

- IMG_SEQ: Georferenceced Image sequence structure.
- sq_size_m: square size in meters, defining the fit windows area. If the parameter is empty, the function set sq_size_m to a value which fit 10 windows in the width of the image. The figure 3.7 display a example where the image is sliced in 50 fitting squares. Read the section 3.3.1, where the parameter sq_size_m is reviewed in detail.
- sq_dist_m: square distance in meter between current fit windows. If the parameter is empty, the function set sq_dist_m equal to sq_size_m.
- mask_2D: 2d logic mask defining the usable area by the fit. If the parameters is empty, all the area is set to 1 (all the area is usable). *mask_2D* is very helpful to avoid to compute areas where the fit is useless, like land areas.
- nan_percentage_thr: maximum percentage of NaN or not usable data data (defined by mask_2D), to enable fit window. If the parameter is empty, the value is set to 5%, so if in a fitting window the number of NaNs or not usable points is bigger than 5%, the window will not be considered.

- Ux_limits_FG: Ux current interval for the fit first guess in meter per seconds. Define the range of currents in X direction, where the fit algorithm will search for the best fit. If the measured Ux is out of this limits, the algorithm will fail. Set the interval wide enough to contain the measured Ux currents. Increasing the interval also increase the computation time. If the parameter is empty, the value is set to [-1.5 1.5] m/s. Before set this parameter, the user should play with the General User Interface provide by CopterCurrents, with the aim to explore the boundaries of Ux_limits_FG and Uy_limits_FG parameter. The GUI is described in the section 3.3.2.

- Uy_limits_FG: Uy current interval for the fit first guess in meter per seconds. Define the range of currents in X direction, where the fit algorithm will search for the best fit. If the measured Uy is out of this limits, the algorithm will fail. Set the interval wide enough to contain the measured Ux currents. Increasing the interval also increase the computation time. If the parameter is empty, the value is set to [-1.5 1.5] m/s. Before set this parameter, the user should play with the General User Interface provide by CopterCurrents, with the aim to explore the boundaries of Ux_limits_FG and Uy_limits_FG parameter. The GUI is described in the section 3.3.2.

- U_FG_res: Current step in meters per seconds to be used in the first guess. If the parameter is empty, the value is set to 0.1.

- w_width_FG: filter width in radians per second, used first guess fit step. If the parameter is empty, the value is set to 1. The GUI described in section 3.3.2 shows the effect of the w_width in the Signal to Noise Ratio calculation.

- U_SG_res: Current step in meters per seconds to be used in the second guess step. If the parameter is empty, the value is set to 0.025.

- w_width_SG: filter width in radians per second, used second guess fit step. If the parameter is empty, the width is set to half of w_width_FG.

- waveLength_limits_m: wavelength limits in meters, used by the dispersion relation fit. If the parameter is empty, then the minimum wavelength is set to 4 times the pixel size, and the maximum wavelength is set to the window diagonal length divided by 4. *waveLength_limits_m* defines the wavenumber interval to be used in the spectral domain for the dispersion relation fit. Adjust *waveLength_limits_m* to the studied wave field saves computing time, use the GUI 3.3.2 for this purpose. Anyway, the default values are good enough to process the wave field successfully.

- wavePeriod_limits_sec: wave period limits in seconds, used by the dispersion relation fit. If the parameter is empty, then the minimum wave period is set to the image sequence frame step (dt), and the maximum wave period is set to the image sequence duration time.

Note: all the *limits* vectors are 2 values vector, where the first value is the lower limit and the second value is the upper limit.

The function *generate_STCFIT_from_IMG_SEQ.m* generates an output *STCFIT* structure, with the following fields:

- STCFIT.Generic.gridX: 2D X grid (Horizontal) in meters.
- STCFIT.Generic.gridY: 2D Y grid (Vertical) in meters.
- STCFIT.Generic.image: image example.
- STCFIT.Generic.Longitude: Longitude [deg].
- STCFIT.Generic.Latitude: Latitude [deg].
- STCFIT.Generic.heading: Angle to North [deg].
- STCFIT.Generic.altitude: altitude in meters.
- STCFIT.Generic.time_stamp: time stamps in datenum format.
- STCFIT.Generic.usable_data_mask_2D: usable points for fit.
- STCFIT.Generic.water_depth_mask_2D: water depth.
- STCFIT.Windows.N_fit_windows: number of windows used for the current fit.
- STCFIT.Windows.w_corners_dim1: 3xN_fit_windows.
- STCFIT.Windows.w_corners_dim2: 3xN_fit_windows.
- STCFIT.Windows.sq_size_m: square size in meter for the current fit windows.
- STCFIT.Windows.sq_dist_m: square distance in meter between current fit windows.
- STCFIT.Windows.average_depth_win: average depth in meters.
- STCFIT.fit_param.Ux_FG_2D: MxN Ux first guess matrix.
- STCFIT.fit_param.Uy_FG_2D: MxN Uy first guess matrix.
- STCFIT.fit_param.Ux_SG_2D: MxN Ux second guess matrix (offset matrix).
- STCFIT.fit_param.Uy_SG_2D: MxN Uy second guess matrix (offset matrix).
- STCFIT.fit_param.w_width_FG: first guess filter width in w [rad/s].
- STCFIT.fit_param.w_width_SG: second guess filter width in w [rad/s].
- STCFIT.fit_param.waveLength_limits_m: [min max] wavelength to use [meters].
- STCFIT.fit_param.wavePeriod_limits_sec: [min max] wave Period to use [seconds].
- STCFIT.fit_param.K_limits: [min max] wave number to use [rad/m].
- STCFIT.fit_param.W_limits: [min max] wave frequency to use [rad/sec].

The code to generate the STCFIT structure is:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Slice Rectified sequence and define fit parameters - step 3
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% generate structure with the fit structure
STCFIT = generate_STCFIT_from_IMG_SEQ(IMG_SEQ, sq_size_m,sq_dist_m,mask_2D,...
    nan_percentage_thr,water_depth_mask_2D,Ux_limits_FG,Uy_limits_FG,U_FG_res,...
    w_width_FG,U_SG_res,w_width_SG,waveLength_limits_m, wavePeriod_limits_sec);
```

3.2.4 CopterCurrents step 4: waves dispersion relation fit

This step computes the waves dispersion relation fit. The fit is performed individually in every window defined by the STCFIT structure. The fit algorithm search in the best current (Ux,Uy) solution inside the limits defined by (Ux_limits_FG,Uy_limits_FG). The criteria to choose the best fit is the Signal to Noise Ratio density defined in the equation (2) [3]. The parameters used in the fit are stored in STCFIT.fit_param. The fit is performed in the following order:

1. Select the intensity data according to the window limits, as result a 3D intensity matrix is retrieved.

2. Perform a 3D fft in the window data, the spacial-time information is translated to the wavenumber and frequency domain. The 3D spectra is cut according to STCFIT.fit_param.K_limits and STCFIT.fit_param.W_limits.

3. First guess: apply the wave dispersion relation equation to all the possible current combinations (STCFIT.fit_param.Ux_FG_2D,STCFIT.fit_param.Uy_FG_2D), and retrieve the Signal to Noise Ratio density for every combination. A first guess SNR density map is displayed in the figure 3.9, upper-right plot. Then, the current pair (Ux,Uy) which maximizes the SNR density, is selected as first guess current fit result.

4. Second guess: Search for maximum Signal to Noise ratio around first guess, using a smaller filter width (STCFIT.fit_param.w_width_SG) and current step U_SG_res. The new maximum SNR density corresponding pair (Ux,Uy) retrieves second guess current fit. A second guess SNR density map is displayed in the figure 3.9, lower-right plot.

All the Signal to Noise Ratio calculation are stored in the STCFIT.out_fit structure, for further analysis:

- FG_fit.Ux_2D: 2d matrix with Ux first guess.
- FG_fit.Uy_2D: 2d matrix with Uy first guess.
- FG_fit.signal_2D: Signal in the dispersion relation in every (Ux,Uy) pair.
- FG_fit.noise_2D: Noise out of the dispersion relation in every (Ux,Uy) pair.
- FG_fit.signal_nvalues_2D: n values in the dispersion relation in every (Ux,Uy) pair.

- FG_fit.noise_nvalues_2D: n values out of the dispersion relation in every (Ux,Uy) pair.
- FG_fit.SNR_2D: Signal to Noise ratio in every (Ux,Uy) pair.
- FG_fit.SNR_density_2D: Signal to Noise density ratio in every (Ux,Uy) pair.
- FG_fit.Ux_fit: best first guess fit for Ux.
- FG_fit.Uy_fit: best first guess fit for Uy.
- FG_fit.SNR_max: SNR corresponding to the best first guess.
- FG_fit.SNR_density_max: SNR density corresponding to the best first guess.
- SG_fit.Ux_2D: 2d matrix with Ux second guess.
- SG_fit.Uy_2D: 2d matrix with Uy second guess.
- SG_fit.signal_2D: Signal in the dispersion relation in every (Ux,Uy) pair.
- SG_fit.noise_2D: Noise out of the dispersion relation in every (Ux,Uy) pair.
- SG_fit.signal_nvalues_2D: n values in the dispersion relation in every (Ux,Uy) pair.
- SG_fit.noise_nvalues_2D: n values out of the dispersion relation in every (Ux,Uy) pair.
- SG_fit.SNR_2D: Signal to Noise ratio in every (Ux,Uy) pair.
- SG_fit.SNR_density_2D: Signal to Noise density ratio in every (Ux,Uy) pair
- SG_fit.Ux_fit: best second guess fit for Ux.
- SG_fit.Uy_fit: best second guess fit for Uy.
- SG_fit.SNR_max: SNR corresponding to the best second guess.
- SG_fit.SNR_density_max: SNR density corresponding to the second first guess.

The number of fit performed in the first guess (STCFIT.fit_param.Ux_FG_2D,STCFIT.fit_param.Ux_FG_2D), depends on Ux_limits_FG, Uy_limits_FG and U_FG_res. The number of fits performed on the second guess depend on U_SG_res. Reducing the number of fits is a option to reduce the algorithm computing time. The code to run the current fit is:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Run current fit  - step 4
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% run current fit in every square
STCFIT = run_current_fit(IMG_SEQ,STCFIT);
```

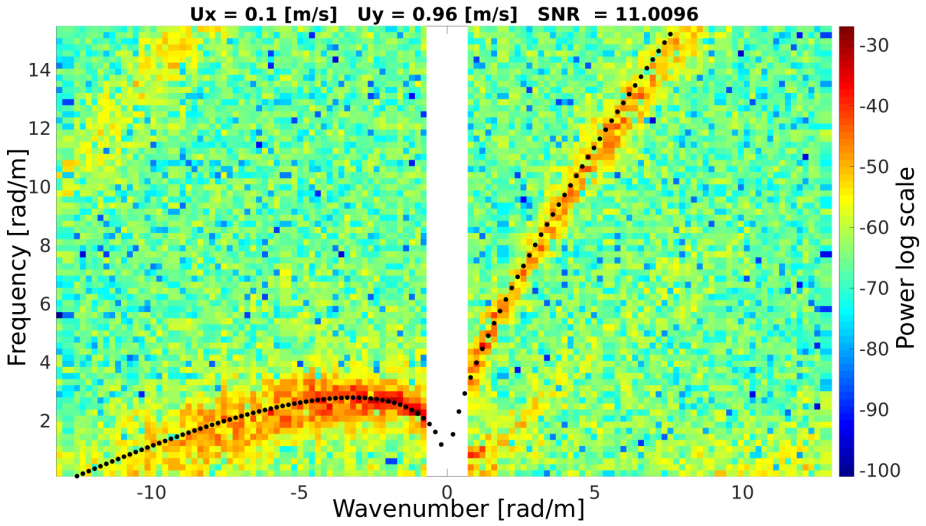


Figure 3.4: Spectra example.

3.2.5 CopterCurrents step 5: Current maps generation

The last step consist of the current maps retrieving. The current maps are generated from the STCFIT structure by the function *get_currents_from_STCFIT.m*. Then the current maps structures are plotted by the function *plot_currents_map.m*. These are the parameters to take in account:

- SNR_thr: Signal to noise Ratio threshold. Minimum SNR to be considered as a valid measurement. Normally is not used. Set to 0.
- SNR_density_thr: Signal to noise Ratio density threshold. Minimum SNR density to be considered as a valid measurement. The SNR density provide us a idea about how good is the quality of the fit. A bigger value means a more trustable result.
- currentdir_flag: defines the sense of the current vectors, set to 1 for 'going to' direction.

The current map structures contain the following information:

- UTM_currents Structure: The current maps are referenced to the Universal Transverse Mercator coordinate system (UTM).
  - UTM_currents.x: vector containing the UTM x positions.
  - UTM_currents.y: vector containing the UTM y positions.
  - UTM_currents.Ux: vector containing Ux in meter per second.
  - UTM_currents.Uy: vector containing Uy in meter per second.
  - UTM_currents.SNR: SNR vector
  - UTM_currents.SNR_density: SNR density vector
  - UTM_currents.time_stamp: time stamp in datenum format

  - UTM_currents.utmzone: string defining the utm zone.
  - UTM_currents.uav_x_utm: x drone position in UTM coordinates.
  - UTM_currents.uav_y_utm: y drone position in UTM coordinates.
  - UTM_currents.currentdir_flag: [1] current direction 'going to',[0] current direction 'coming from'.

- Camera_currents Structure: The current maps are referenced to the camera position, so all the current extracted are referenced to the camera axis, not to North, so the currents values are relative to the camera.
  - Camera_currents.x: x position from camera center.
  - Camera_currents.y: y position from camera center.
  - Camera_currents.Ux: Ux [m/s] in the camera direction.
  - Camera_currents.Uy: Uy [m/s] in the camera direction.
  - Camera_currents.SNR: SNR vector
  - Camera_currents.SNR_density: SNR density vector
  - Camera_currents.time_stamp: time stamp in datenum format
  - Camera_currents.currentdir_flag: [1] current direction 'going to', [0] current direction 'coming from'.

In the figures 3.5 and 3.6 are displayed both examples obtained by *plot_currents_map.m*. The figure 3.5 use the Camera_currents Structure, and the figure 3.6 use the UTM_currents Structure.

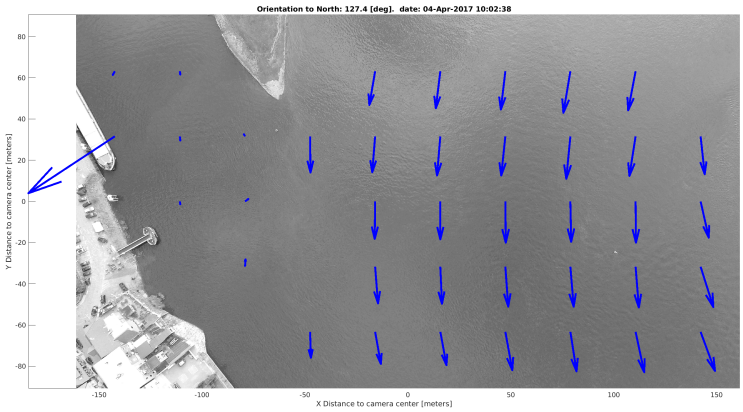


Figure 3.5: Current maps example, plotted with the camera location as reference.

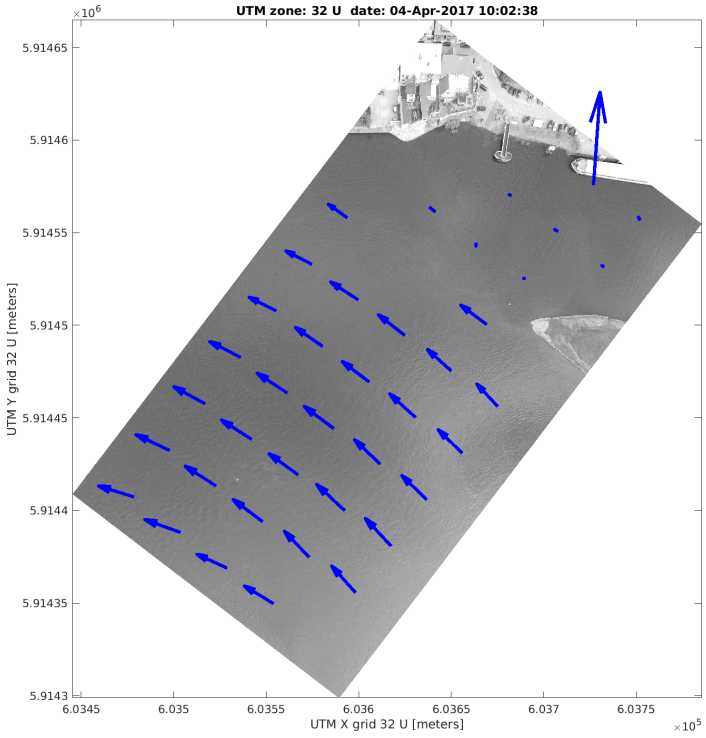


Figure 3.6: Current maps example, plotted in the Universal Transverse Mercator coordinate system (UTM).

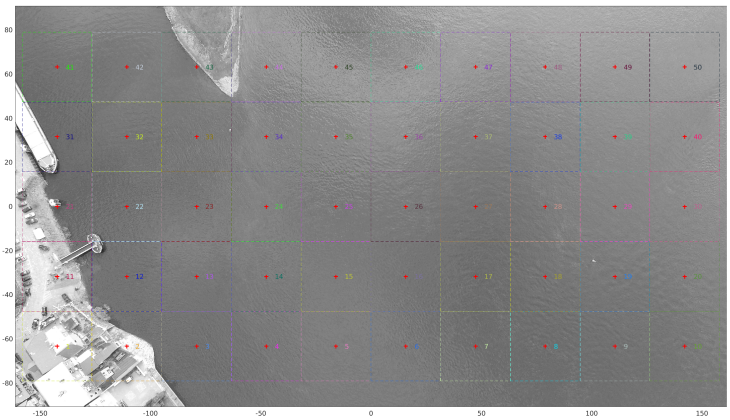


Figure 3.7: Fitting squares distribution. The video is sliced in 50 regions, where the dispersion relation fit will be executed in every window. The windows are processed independently, the results in one windows will not affect to the adjacent windows.

3.3 Parameter selection

CopterCurrents is designed to process an Image sequence just using the default parameters. A default parameter is set every time that a parameters is set to the empty value, so, a unskilled user is able to obtain current field in a straight forward way. Most of the parameters defined in *generate_STCFIT_from_IMG_SEQ.m* can be set to a empty value, and the function will set up a default value. Nevertheless, the secondary purpose of CopterCurrents is displaying how each parameter affect to the dispersions relation fit. Some extra functions are provided to let the user play around the parameters.

3.3.1 Square size

The fit square size (in meters) *sq_size_m* defines the area used in every fit. A bigger area increases the number of waves involved in the dispersion relation fit, increasing the fit robustness. Increasing the area, also reduce the resolution (number of measurements per square meters) of the algorithm, so the user should arrange an agreement between robustness and resolution.

The function *plot_STCFIT.m* display windows distribution in a STCFIT structure, as we can see in the picture 3.7.

```
[h] = plot_STCFIT(STCFIT);
```

The distance between squares, *sq_dist_m*, can be also reduce to increase the density of measurements. I recommend to set *sq_dist_m* to half of *plot_STCFIT.m*, to overlap the windows measurements. For example in [3], the *sq_size_m* is set to 8 meters, but *sq_dist_m* is set to 4 meters, so there is half

square overlap between measurements.

Another way to improve the robustness is expand the video textittime_limits, described in the step 1, 3.2.1. When the time is increased, the number of waves analyzed by the fit algorithm is increased too. As disadvantage, increasing the time, increases the frames to be analyzed, which requires more RAM memory during the processing.

3.3.2 Dispersion relation filtering

CopterCurrents provides two Graphical User Interfaces to test the dispersion relation parameters. The GUIs display the raw spectra, and the dispersion relation shell, according to Ux, Uy and filter width. These three parameters can be modified in real time by three sliding bars.

The dispersion relation is defined in 3 dimensional space (kx,ky,w), but the GUIs only plot 2 dimensional data (k,w). The GUIs plot only a 2 dimesional cut of the 3 dimensional spectra. A fourth sliding bar is added to choose the direction of the K (wavenumber) and W (frequency) cut to be displayed.

The figure 3.8 display the results of the GUI function *display_fit_guess.m*, as we can see, the GUI provides:

- plots:
  - Upper-left window: Wave dispersion relation average in frequency axis. A red line the shows direction of the Wavenumber-frequency cut.
  - Upper-Right window: Wavenumber-frequency cut, raw spectra according to 'w_cut_dir' direction. The dispersion relation corresponding to the current, defined by the sliders (Ux,Uy), is plotted in a black dotted line. Move the sliders to check effect in the dispersion relation.
  - Lower-Right window: Wavenumber-frequency cut, filtered spectra. Similar to the upper window, but the results was filtered by filter width parameter. Use the *filter width* sliding bar to play around the desired parameters.
- sliding bars:
  - Ux: Current in X direction to be applied to the dispersion relation.
  - Uy: Current in Y direction to be applied to the dispersion relation.
  - KW cut direction: Defines the direction of the Wavenumber-frequency cut, in degrees. The direction is plotted in the Upper-left window, with a red line.
  - filter width: Filter width in rad/sec used in the filter. Check the effect of the filter width in the Lower-Right window.

Play around with the sliding bars and check how the SNR is increased or decreased. Then, the user is able to obtain manually the current (Ux,Uy),

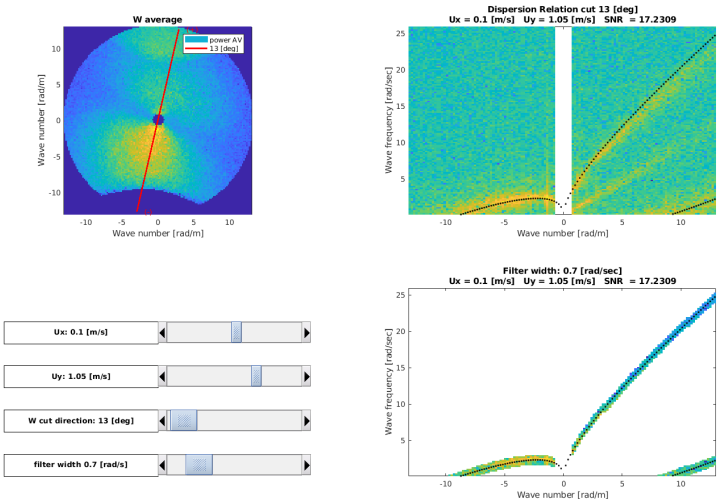


Figure 3.8: Dispersion Relation filter GUI *display_fit_guess.m*

maximizing the SNR densitiy.

The figure 3.9 display the results of the GUI function *display_fit_guess_v2.m*. this GUI is similar to the previous one, but includes the Signal to Noise Ration maps. The GUI provides:

- plots:
  - Upper-left window: Wave dispersion relation average in frequency axis. A red line the shows direction of the Wavenumber-frequency cut.
  - Upper-Center window: Wavenumber-frequency cut, raw spectra according to 'w_cut_dir' direction. The dispersion relation corresponding to the current, defined by the sliders (Ux,Uy), is plotted in a black dotted line. Move the sliders to check effect in the dispersion relation.
  - Lower-Center window: Wavenumber-frequency cut, filtered spectra. Similar to the upper window, but the results was filtered by filter width parameter. Use the *filter width* sliding bar to play around the desired parameters.
  - Upper-Right window: First guess Signal to Noise Ratio density map. Displays the SNR density dependence on Ux and Uy parameters, for the first guess step described in 3.2.4. A clear peak must be observed, otherwise the fit will fail. A cross + shows the current retrieved by the fit function. A cross + shows the current defined by the sliding bars.
  - Lower-Right window: Second guess Signal to Noise Ratio density map. Displays the SNR density dependence on Ux and Uy parameters, for the second guess step described in 3.2.4. The second guess refines

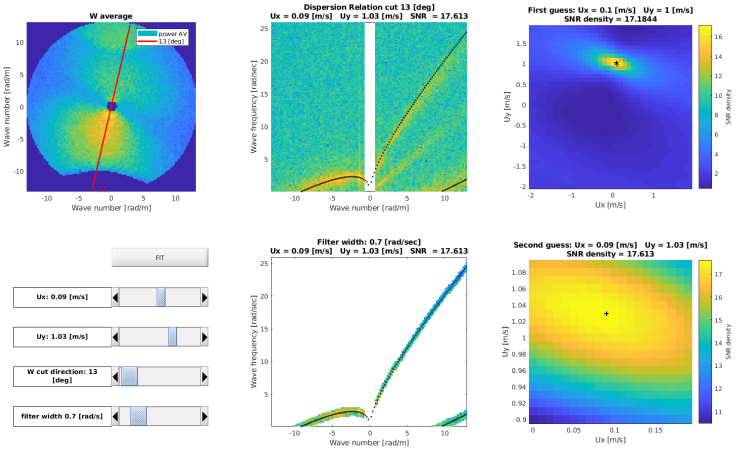


Figure 3.9: Dispersion Relation filter GUI *display_fit_guess_v2.m*

the solutions around of the first guess. A cross + shows the current retrieved by the fit function. A cross + shows the current defined by the sliding bars.

- sliding bars:
  - Ux: Current in X direction to be applied to the dispersion relation filter.
  - Uy: Current in Y direction to be applied to the dispersion relation filter.
  - KW cut direction: Defines the direction of the Wavenumber-frequency cut. The direction is in degrees. The direction is plotted in the Upper-left window, with a red line.
  - filter width: Filter width in rad/sec used in the filter. Check the effect of the filter width in the Lower-Center window.
- bottom:
  - FIT: Recalculate the SNR maps according to the filter width displayed on the slide bar.

The Signal to Noise Ratio maps are very important to determine if the first guess current limits (*Ux_limits_FG* and *Uy_limits_FG*) are set properly. The first guess SNR map must display a clear peak, otherwise there is no wave signal on the analyzed window, or the the current on that window are out the limits. In the figure 3.9 a clear peak is displayed in the Upper-Right window.

3.4 Camera Calibration

Before run CopterCurrents, the UAV camera must be calibrated. A correct calibration allows to obtain the pixels real size, in meters, from the video frames.

This is the most important step for CopterCurrents, otherwise, the dispersion relation fit will fail. CopterCurrents allows 3 different methods for camera calibration: FOV, Caltech and OpenCV, all of them will be explained in the next sections. The user should choose only one of them.

The calibration parameters are stored in a calibration MATLAB structure called *CopterCurrents_CamCalib* and saved in a .mat file. The calibration parameters (intrinsic camera parameters) depends on the camera, and the camera resolution. The camera must be calibrated for every resolution, in other words, the calibration parameters are only valid for the resolution that was used during the adjustment.

All the calibration files should be stored in the CopterCurrents folder *CopterCurrents_CameraCalib*.

The camera calibration is useless if the camera auto-focus is on, read your Drone manual and disable the auto-focus option. The calibrations performed in this chapter were done with a DJI Phantom 3 camera, which has a fix focal length (no auto-focus option).

A general calibration guideline written by Paolo Galeone is available in the following link `https://pgaleone.eu/computer-vision/2018/03/04/camera-calibration-guidelines/#guidelines`. The guideline was not written for CopterCurrents, but defines all the requirements to perform a proper camera calibration.

Finally, all the measurements used during the calibration process must be fulfilled in **meters**. Do not use centimeters, millimeters or inches, otherwise the dispersion relation calculation units will not match with the CopterCurrents code.

3.4.1 Camera Calibration FOV

The Field Of View calibration (FOV) is a simple and straight forward calibration method. It does not take in account the lens distortion, therefore it is not the most accurate, but it is the recommended method for beginners.

FOV calibration consist of record a know size object, a right triangle, for examaple, and use the GUI to calculate correct field of view parameters.

1. Record a known size object , with the Drone camera in nadir position (90 degrees). In the figure 3.10, a 10 meters right triangle is displayed, recorded at 15 meters height.

2. Run the script *example_FOV_script.m*. Edit first the script parameters *video_fname* and *path2CopterCurrents_Calibration_files* with the video file-name and the CopterCurrents path *CopterCurrents_CameraCalib* in your computer.

3. After run the script, the following window will be displayed 3.10. Then, adjust the zoom to enhance the triangle and press the SPACE bar on the keyboard.

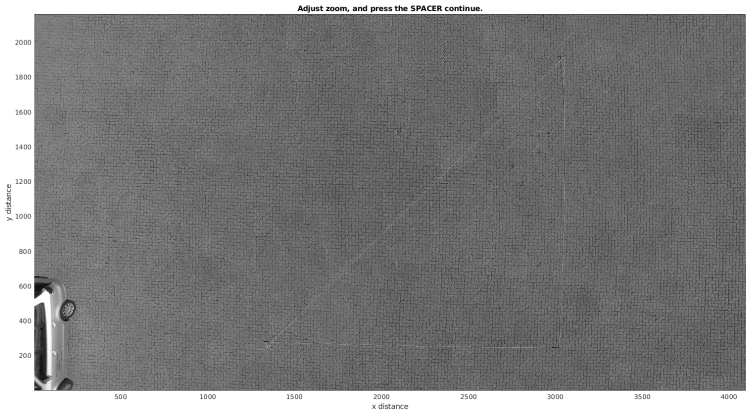


Figure 3.10: Frame from 10 meter right triangle at 15 meters height.

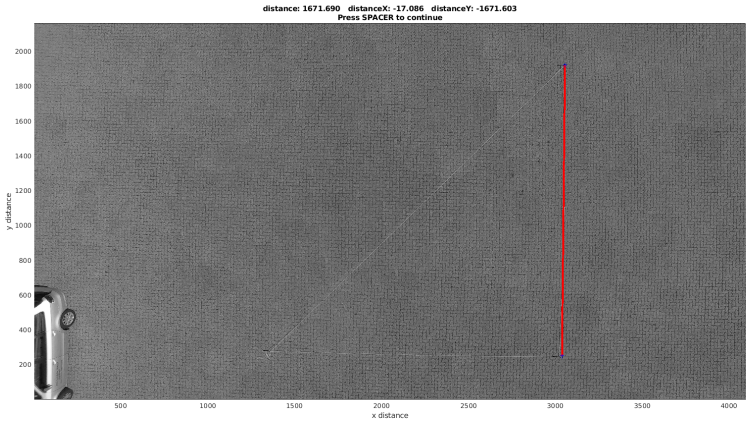


Figure 3.11: Rigth click in the starting and end 10 meters line.

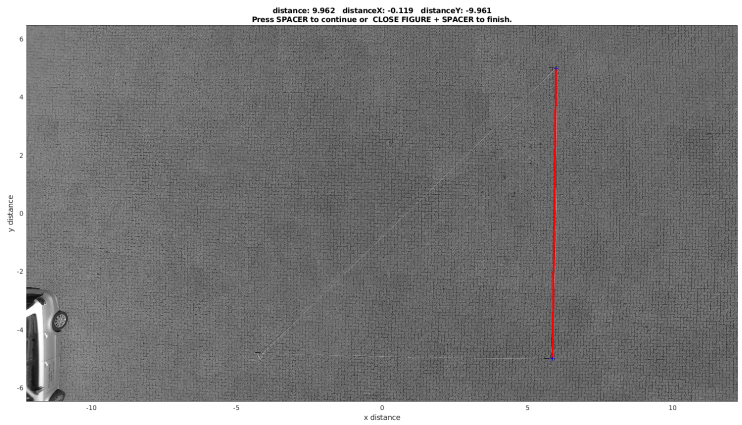


Figure 3.12: Check calibration real size.

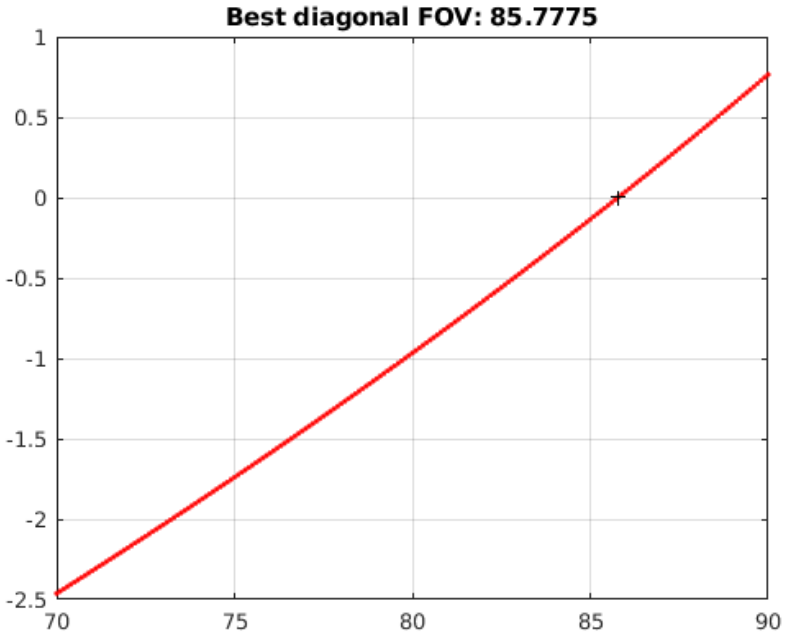


Figure 3.13: FOV calculation.

4. Define a known distance: Make a mouse Right click in the first corner, and then Right click again in the end corner. The following figure 3.11 will be displayed.

5. Press the keyboard SPACER to continue. Then *"Write distance in meters:"* will be displayed in the MATLAB terminal. Write down the real distance in meters, in this example *10*, and press ENTER.

6. The script calculates the FOV parameters to be adjusted to the pixels size. The values will be interpolate between 70 to 90 degrees. These values cover most of the cameras on the market. Finally the interpolated value is displayed 3.13.

7. Check pattern real size. The figure 3.12 will be displayed, use this GUI again, to double check the real distances obtained by the new camera calibration. Try as many times as you want. Close the figure and press the SPACER to exit the GUI.

8. Finally the parameters are saved and the *CamCalib* filename is displayed in the MATLAB terminal.

The output of the script on the terminal should be:

```
    Measure a known distance
    Write distance in meters: 10
    distance used: 10 [meters]
    testing FOV from 70 to 90
    FOV: 70
    FOV: 71
    FOV: 72
    FOV: 73
    FOV: 74
    FOV: 75
    FOV: 76
    FOV: 77
    FOV: 78
    FOV: 79
    FOV: 80
    FOV: 81
    FOV: 82
    FOV: 83
    FOV: 84
    FOV: 85
    FOV: 86
    FOV: 87
    FOV: 88
    FOV: 89
    FOV: 90

    ../CopterCurrents_Calibration_files/FOV_manual_4096x2160.mat

  fov_diag: 85.7775
  fov_x: 78.8161
  fov_y: 46.8531
  size_X: 4096
  size_Y: 2160
  camera_offset_Z: 0
  source: 'Manual_FOV_calibration'
```

The calibration file generated is now ready to be used in a CopterCurrents script.The **CopterCurrents_CamCalib** structure contains the following parameters:

- fov_diag: Diagonal field of view.
- fov_x: X direction field of view (width).
- fov_y: Y direction field of view (height).
- size_X: number of pixels in width direction.
- size_Y: number of pixels in height direction.
- camera_offset_Z: Extra camera offset in meters. Normally set to 0.
- source: Manual_FOV_calibration.

3.4.2 Caltech Camera Calibration

The Caltech camera calibration method takes in account the lens correction, but requires a chessboard grid pattern to be performed. The MATLAB *'Camera Calibration Toolbox'* must be installed, check the chapter 2.1.5.

The camera calibration will be performed using a chessboard grid pattern. A video will be recorded by the Drone camera, while the chessboard grid is rotated in several angles. At the end of the calibration process, a *Copter-Currents_CamCalib* file will be generated. The *CopterCurrents_CamCalib* file contains all the information regarding the intrinsic camera parameters. These are the steps to perform the calibration:

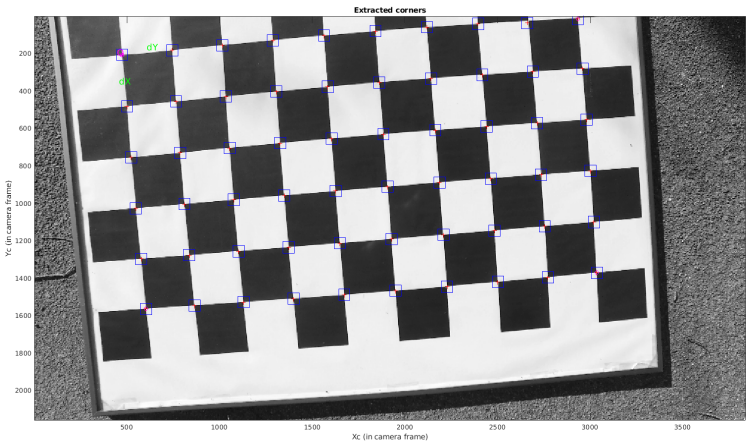


Figure 3.14: Grid corners extration with *'Camera Calibration Toolbox'*.

1. Record a video by the drone camera, while the chessboard is rotating. If the grid is not easy to handle, the best option will be rotate the drone camera manually. The rotation must be performed in the 3 camera angles heading, pitch and roll.

2. Extract several frames from the video and store the frames in the same folder. A script to extract the frames is provided by CopterCurrents *extract_images.m*. Some pictures test are provided with CopterCurrents, which are stored in:
   ../Calibration_examples/Matlab_Caltech/Phantom3_3840/

3. Run *'Camera Calibration Toolbox'* calibration. Write *calib_gui* command in MATLAB terminal. Follow the instructions described in the toolbox manual `http://www.vision.caltech.edu/bouguetj/calib_doc/`. After save the calibration, the result will be stored in the file *Calib_Results.mat*.

4. Export the calibration result to a *CamCalib* struct. Use the script *Save_Caltech_Calibration_script.m*. Edit the script, and adapt the file names to your local paths names:

   (a) Set the variable *Caltech_calibration_filename* to the previous *Calib_Results.mat* file name, obtained in the previous step.

   (b) Set the variable *CopterCurrents_calibration_filename* to the desired new *CopterCurrents_CamCalib* file name.

   (c) Run the script. Then, the Calibration file *CopterCurrents_calibration_filename* will be stored in disk and will be ready to be used by CopterCurrents. All the calibration files should be stored in the *CopterCurrents_CameraCalib* folder.

The **CopterCurrents_CamCalib** structure contains the following parameters:

- fc: 2x1 double value with the camera focal length.
- cc: 2x1 double value with the camera principal point coordinates.
- kc: 5×1 double value with the distortion coefficients.
- alpha_c: normally 0.
- nx: number of pixels in width direction.
- ny: number of pixels in height direction.
- camera_offset_Z: Extra camera offset in meters. Normally set to 0.
- source: Caltech_Matlab.

The parameters description is available in `http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/parameters.html`.

3.4.3 OpenCV Camera Calibration

The OpenCV camera calibration is similar to the Caltech calibration described in the previous section 3.4.2. The calibration is performed by the python OpenCV library, which searches automatically for the chessboard grid points. The OpenCV camera model is an implementation of the Caltech model, so all the camera intrinsic parameters are compatible. The MATLAB *'Camera*

Calibration Toolbox' must be installed, otherwise CopterCurrents will not be able to process the *CopterCurrents_CamCalib* generated.

As in the previous section 3.4.2, the camera calibration will be performed using a chessboard grid pattern. A video will be recorded by the Drone camera, while the chessboard grid is rotated in several angles. At the end of the calibration process, a *CopterCurrents_CamCalib* file will be generated. The *CopterCurrents_CamCalib* file contains all the information regarding the intrinsic camera parameters. These are the steps to perform the OpenCV calibration:

1. Record a video by the drone camera, while the chessboard is rotating. If the grid is not easy to handle, the best option will be rotate the drone camera manually. The rotation must be performed in the 3 camera angles heading, pitch and roll.

2. Extract several frames from the video and store the frames in the same folder. A script to extract the frames is provided by CopterCurrents *extract_images.m*. Some pictures test are provided with CopterCurrents, which are stored in:
   ../Calibration_examples/Matlab_Caltech/Phantom3_3840.

3. CopterCurrents provide a python script *camera_calibration.py* to perform the calibration. Edit the script and adjust the following parameters to your grid characteristics:

   ```
   # define grid crorners
   n_corners_vert = 6 # number of corners in vertical direction
   n_corners_horz = 10 # number of corners in Horizontal direction
   square_size = 0.105 # square size in meters
   subpixel_search_winSize = 21 # subpixels resoluion
   path2images = '/path2images/'
   ```

   The chessboard matched pictures will be saved in the active folder, an example is displayed in the figure 3.15. The calibration file will be also stored in the active folder, with the following file name format:

   CopterCurrents_CamCalib_RESOLUTION_HORxRESOLUTION_VERT.mat.

   Where, *RESOLUTION_HOR* is the frame horizontal resolution, and *RESOLUTION_VER* is the vertical resolution.

4. All the calibration files should be stored in the *CopterCurrents_CameraCalib* folder. After run the script, copy the retrieved *CopterCurrents_CamCalib* file to the local *CopterCurrents_CameraCalib* folder.

The **CopterCurrents_CamCalib** structure contains the following parameters:

- fc: 2x1 double value with the camera focal length.
- cc: 2x1 double value with the camera principal point coordinates.
- kc: 5×1 double value with the distortion coefficients.

- alpha_c: normally 0.
- nx: number of pixels in width direction.
- ny: number of pixels in height direction.
- camera_offset_Z: Extra camera offset in meters. Normally set to 0.
- source: OpenCV.

The parameters description is available in `https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html`.

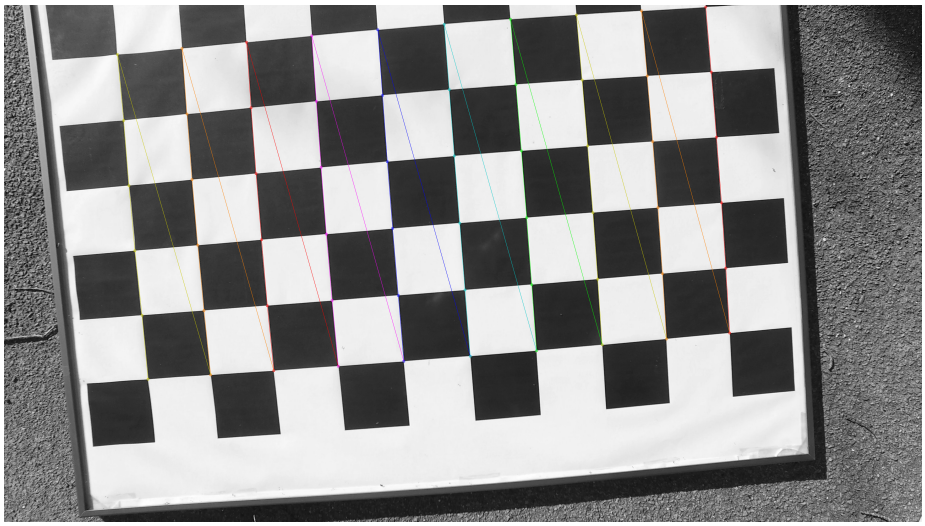


Figure 3.15: Grid corners extration with *'OpenCV'*.

Chapter 4

TroubleShooting

4.1 Video metadata

CopterCurrents has been tested only with metadata from DJI drone videos. If the video metadata format does not follow the DJI format, the function *get_mediainfo_information.m* will fail.

As work around solution, the user can edit manually the function *Run_CopterCurrents_script.m* to add directly the Camera Position:

```
% get video position in DJI drone
% [CamPos_ST] = get_Camera_Position_Struct(video_fname);

% Add data manually if is not a DJI drone
CamPos_ST = struct('LONGITUDE',LON,'LATITUDE',LAT,'Height',Height,...
                   'timestamp',timestamp,'yaw',yaw,'pitch',pitch,...
                   'roll',roll,'extra',mediainfo_string);
```

The **CamPos_ST** structure (Camera position structure), contains the following information:

1. LONGITUDE: Longitude of the camera position in degrees.
2. LATITUDE: Latitude of the camera position in degrees.
3. Height: Camera height from home position in meters.
4. timestamp: time stamp in datenum matlab format.
5. yaw : yaw (heading) camera angle in degrees, according to DJI camera angles definition. In Nadir position the value should be the heading to the north in degrees.
6. pitch: pitch camera angle in degrees, according to DJI camera angles definition. In Nadir position the value should be -90.
7. roll: roll camera angle in degrees, according to DJI camera angles definition. In Nadir position the value should be 0.
8. extra: extra user information (optional).

Please contact to CopterCurrents maintainer (ruben.carrasco@hzg.de) to add your Drone metadata format to CopterCurrents.

4.2 Video Codec

CopterCurrents reads the drone video frames using the MATLAB 'videoreader', which sometimes requires specific codecs. For more information about the compatible codecs:

`https://www.mathworks.com/help/matlab/import_export/supported-video-file-formats.html`

Problems were reported during CopterCurrents development, the DJI video codecs (MPEG-4, H.264) under Linux environment were not easy to install. Mathworks suggested the following work around solution:

`https://www.mathworks.com/matlabcentral/answers/94531-why-do-i-receive-an-error-when-creating-a-videoreader-object-on-linux-in-matlab-r2010b-7-11#answer_103883`

4.3 Matlab videoreader

Matlab videoreader get stuck when is reading the last frames of the video. This behavior was reported using Matlab 2017b. It is solved in Matlab2018. To avoid this issue, do not use the last second of the videos.

4.4 split function

The function *add_CopterCurrents_matlab_path* does not work under matlab 2016 and previous versions. The *split* function used in *add_CopterCurrents_matlab_path* does not exist in these versions. As work around solution add manually the CopterCurrents path to your matlab environment.

4.5 Altimeter accuracy

The video rectification procedure is very sensitive to the altitude. DJI drones use a barometric sensor to determine the altitude from the home position. When the home position is set, the sensor pressure value is set as reference, and all the altitude will be calculated regarding this reference. So any change on the pressure or in the sensor temperature (the barometric sensor drifts with the temperature) will affect the altitude measurement. A healthy practice is set the home position just before to start every flight.

Bibliography

[1] W. Commons, "Dji phantom2 vision plus," 2014.

[2] K. C. B.-S. . (https://creativecommons.org/licenses/by sa/3.0)], "Orbital wave motion," 2008.

[3] M. Streßer, R. Carrasco, and J. Horstmann, "Video-based estimation of surface currents using a low-cost quadcopter," *IEEE Geoscience and Remote Sensing Letters*, vol. 14, pp. 2027–2031, Nov 2017.