Ship Wake Data Analyzer

Kevin Bélanger
engineering student, Université de Sherbooke

Vivian Issa
DRDC – Valcartier Research Centre

Defence Research and Development Canada
Reference Document
DRDC-RDDC-2018-D141
February 2019

IMPORTANT INFORMATIVE STATEMENTS

Abstract

In support of the Project 01ec a tool is required to facilitate trials data analysis. Wake measurement trials collect raw infrared data images, environmental and climatic data, sensor heading and specifications, naval platform and helicopter GPS coordinates, sea temperature and sea stratification and time. These data collected with multiple sensors has wide range of time resolution. The objective of the Wake Analyzer software is to convert all raw data to the format selected by the user, synchronize all the data collected from different instrument in order to allow automatic processing. The tool, which is a MATLAB graphic user interface, allows the user to open each image and access to all associated Meta data. It allows also selecting one or multiple parts of the image and applying specific algorithms to them. This reference document shows the user guide of the Wake Analyzer tool.

Résumé

En support du projet 01ec, un outil est requis pour faciliter l'analyse des données des essais expérimentaux. Les essais de mesures de sillages collectent des données brutes infrarouges, des données environnementales et climatiques, des données GPS du Platform navale et de l'hélicoptère, de l'orientation du capteur et ses spécifications, de la température de la mer et sa stratification ainsi que l'heure. Ces données sont collectées par des différents capteurs et ne sont pas synchronisées temporellement. L'objectif du logiciel Wake Analyzer est de convertir les données brutes au format choisie par l'utilisateur, synchroniser les données collectées pour permettre un traitement automatique. L'outil, qui est une interface graphique MATLAB, permet à l'utilisateur d'ouvrir les images individuellement et d'accéder à toutes ses métadonnées. Il permet aussi de sélectionner une ou plusieurs parties de l'image pour y appliquer un algorithme spécifique. Ce document de référence montre le guide d'utilisateur de l'outil Wake Analyser.

This page intentionally left blank.

1 Software Architecture

1.1 User Interface

1.1.1 Matlab's GUIDE

The software's user interface was developed with the help of Matlab's GUIDE. This editor,

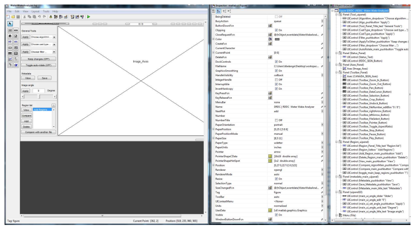


Figure 1: *Matlab's GUIDE editor.*

which picture example can be seen in Figure 1 on the left, is based on user interface components of the Java.swing library. It stands out for its ease of use, drag and drop objects, that are fully customizable by adjusting fixed properties or using the underlying Java.swing properties. This way, most of the basic user interface features can be developed quite fast and modified for more advanced purposes if needed.

1.1.2 Source Folder Package Structure

Even though Matlab is based on scripts and functions that doesn't require further complexity, the structure of the source folder has been organized following most of the JAVA object-oriented and packages fundamentals. This brings clarity, code efficiency and it respects a good programming practice. The main reason behind the structural necessity was the software's composition of over seven thousand lines of code in about 150 different files. This organization was also made to simplify comprehensiveness of the code logic. Figure D.1 on Page 27 describes the high level structure of the source folder. The decoupling of the package hierarchy is based on their utility. The 'customAlgorithms' package contains all the default user generated functions and those that will be defined in the future. The 'externalLibs' package contains all the non-proprietary scripts and executable that were used to accelerate the software development. The 'tools' package is subdivided in five other packages containing all the non user interface functions. The subdivisions are made based on common characteristics of the underlying functionality of the scripts.

The last high level package is the UI one. This package is subdivided based on the software user interfaces since each one has their own graphical components and functionality. Figure D.2 on Page 28 describes the most complex structure that can be found in one of these lower level package by defining the structure of the main UI package. These packages are subdivided themselves based on graphical component categories. Each of the packages contains callback functions that are known by the figure file corresponding to the user interface of their package. These callback functions respect the MVC principle and contains the actual actions linked to a user event of a component. In addition to those functions, the UI sub-packages also contain every file linked to their existence like images or sub-functions.

1.1.3 Main UI Data Structure

Following up on MVC principle, the software's data structure can be explained by the event callback bus system and global parameter that is passed on, in this case the object of data type 'struct' named 'handles'. The figure D.3 on Page 29 shows the attributes data type structure of the 'handles' object for development references. This data is a singleton object by its global definition and can be accessed from any callback functions or UI sub-functions at any time. Part of its structure is passed on sub-UIs and is redefined as such in the latter. It is also sent to the custom algorithms to give as much flexibility as possible to the user for future development.

1.2 Software embedded Tools

Four tools have also been developed to help the user with common tasks related to image processing. These tools are mostly covered in the user guide found on Page 3 but two of these were made specifically because of the short development time frame. The first one is the Sfmov conversion tool. Sfmov extension files would have been loaded directly into the application for analysis if it were not for the metadata origin. Without embedded systems, data collection devices produced desynchronized data that could not be loaded in memory at once. The solution was to convert the data into a more accessible and faster type of data that could contain flexible metadata header at the cost of preprocessing and synchronizing. The Matlab mat format was used for this purpose. This format is easy to use and provides a simple way to access it via the Matlab software workspace. A compromise was also made for the second tool related to time frame, which is the custom algorithm system. A complete embedded code editor could have been developed, but this was a project in itself. Instead, a mildly-complex system was developed to palliate to the necessity of having development perennity. A complete guide can be found on Page 13. Even though it could seem more complex than it really is, every aspect of the necessary flexibility of this system is present and will provide the necessary requirements.

Annex A User Guide

Global view

This guide is an overview of the Ship Wake Analyzer software features and their respective functionality. Figure A.1 below enumerates their position and serves as a visual table of content of the application's sections.

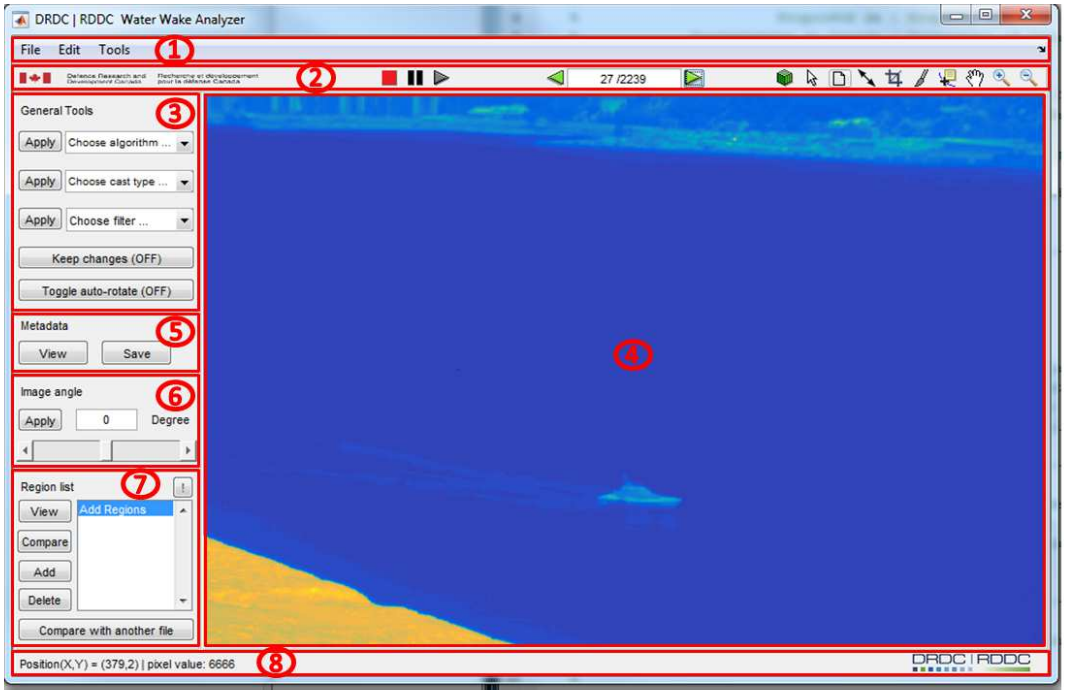


Figure A.1: *Software global view.*

Here is the definition of each section number and their references into this document.

1. Menu tabs:	see Section A.1 on Page 4 of this guide.
2. Toolbar:	see Section A.2 on Page 8 of this guide.
3. General tools panel:	see Section A.3 on Page 9 of this guide.
4. Main image view:	see Section A.4 on Page 9 of this guide.
5. Metadata panel:	see Section A.5 on Page 10 of this guide.
6. Rotation panel:	see Section A.6 on Page 10 of this guide.
7. Region list Panel:	see Section A.7 on Page 10 of this guide.
8. Status bar:	see Section A.8 on Page 11 of this guide.

A.1 Menu tabs

A.1.1 File menu

The file tab is the basic menu tab for folders and application's I/O management. Unless there is file conversion to do, it is the starting point of any analysis.

- **Save As Image...** (shortcut-key Ctrl+S) saves the currently viewed image as JPG, PNG or BMP format without additional metadata. For more type and metadata conversion, see conversion tools from tool menu.
- **Save As Raw Data...** (shortcut-key Ctrl+R) saves the currently viewed image as the Raw format in a mat file with the current available metadata.

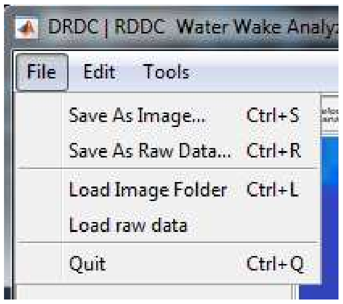


Figure A.2: File menu content.

- **Load Image Folder** (shortcut-key Ctrl+L) Select a folder and every image file in JPG, PNG and BMP format will be loaded into the application from that source.
- **Load raw data** loads any pre-converted data from the sfmov conversion tool of the tool menu. This file should be recognized as AnyName_1.mat, which is the header file in converted Raw data.
- **Quit** will exit the current application and close the necessary components.

A.1.2 Edit menu

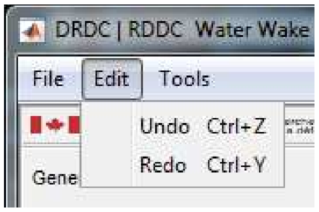


Figure A.3: Edit menu content.

The edit tab is a simple menu tab giving access to the command buffer features. Any changes made to the main image can be undone with the **Undo** option (shortcut-key Ctrl+Z) or redone with the **Redo** option (shortcut-key Ctrl+Y). The actual buffer can contain up to 30 modifications. If more modifications are applied, the first modifications are lost and the keep changes options from the general tool panel will not be able to sustain the consistency of all the changes made. In that case, the buffer will be reset upon changing the image file.

A.1.3 Tool menu

The tool tab is a more complex menu tab. It contains separate application calls and each one will be redefined in its own sub-section of this guide.

- **Image Type Converter** can convert standard image file formats into other types of standard image files. See Section A.1.3.1 on Page 5.
- **SFMov / Mat Converter** can convert an sfmov raw movie format into an easy to work with mat file. See Section A.1.3.2 on Page 6.

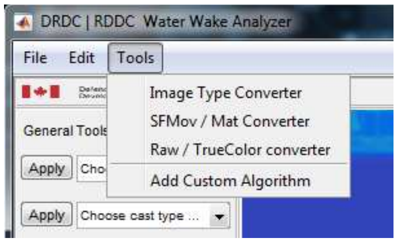


Figure A.4: *Tool menu content.*

- **Raw / Truecolor Converter** can convert the Raw image format from mat file into standard truecolor image file format. See Section A.1.3.3 on Page 7.
- **Add Custom Algorithm** gives access to a function generator for customizable algorithms compatible with the compare tool of the region panel. This tool needs a little more in-depth explanation, see Annex B on Page 13.

A.1.3.1 image type converter tool

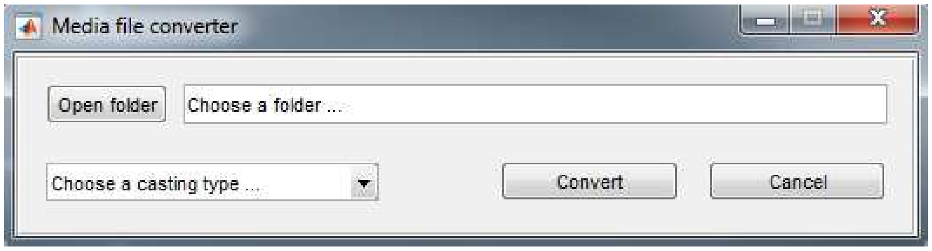


Figure A.5: *Image type converter user interface.*

This tool can convert any of the five following image types into another one:
JPG, PNG, BMP, GIF and/or TIFF.

The user must first click on the open folder button and select a folder containing one or more of the five compatible types. Then, a casting type must be selected from the drop-down menu right under the folder button. The casting type is the type in which all compatible image formats from the selected folder will be converted. Finally, clicking the convert button will create a sub-folder in the selected folder named convertedFiles containing the converted files.

The user can cancel the conversion progress at any time by pressing the X button at the right of the progress bar as seen in the Figure A.6 below.

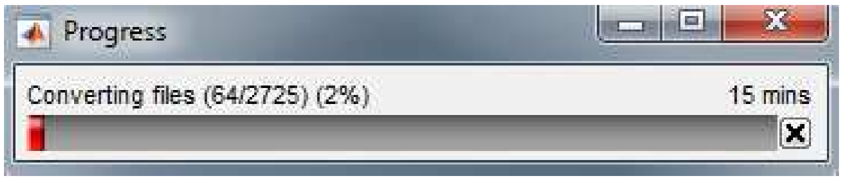


Figure A.6: *Image type converter progress bar.*

A.1.3.2 Sfmov to mat converter tool

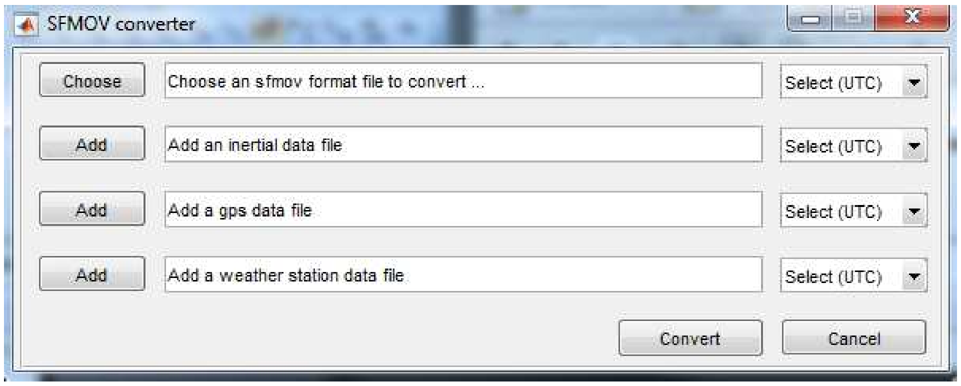
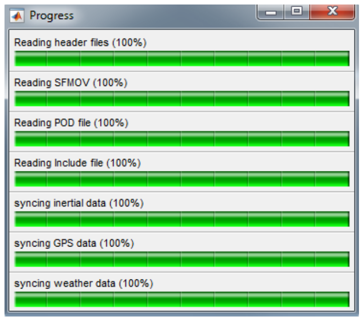


Figure A.7: *SFMov to mat converter user interface.*

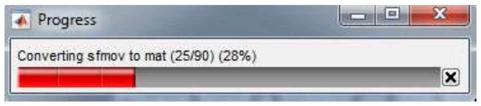
FLIR cameras can produce sfmov movie formats. Analysis of these types of data would necessitate complex algorithms and would be quite slow, especially in terms of image file browsing. To overcome this problem, a converter can transfer sfmov data into Matlab's mat format. The converter user interface can be seen in the Figure A.7 above. This converter gives the opportunity to analyze the data in a more flexible, easy and fast way. Three other **optional** files can be loaded to add metadata to the images. The user only has to click the button on the left of a file selection definition box to actually select one. Each file selected needs an associated UTC time reference selected from their respective drop-down menu on the right. This is needed so the metadata of each image can be synchronized with the data from each file.

Once the user presses the convert button, the progress bar from Figure A.8a below will appear and guide the user on the progress of synchronizing the metadata files. Normally, an include file and a pod file (Parameter Oriented Data) should come with the sfmov movie. They are actually required for the conversion process as they contain crucial information on the data. once these loaded, the progress bars will give an assessment of time needed to synchronize each file if they were defined earlier. Note that this step cannot be canceled to prevent I/O errors since too much disk operations are made at the same time.



Reading header files (100%)
Reading SFMOV (100%)
Reading POD file (100%)
Reading Include file (100%)
syncing inertial data (100%)
syncing GPS data (100%)
syncing weather data (100%)

(a) File headers conversion progress



Converting sfmov to mat (25/90) (28%)

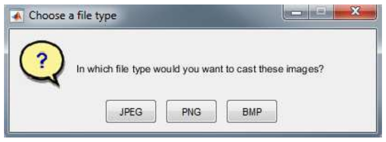
(b) Image conversion progress

Figure A.8: *sfmov to mat conversion tool progress bars.*

Then the final step should start as it can be seen in Figure A.8b above. Each image is embedded with metadata and converted to mat format. The overall operation can be canceled at this step by clicking on the X button on the right of the progress bar. After the conversion process, multiple files with mat extension will be left in the same folder as the sfmov folder. They should all have the same name as the sfmov file with a _# at the end. The sfmovName_1.mat is the header file containing all the structured metadata and reference to the other files. The other sfmovName_#.mat files are the actual images stored in order.

A.1.3.3 Raw to trueColor converter tool

The Raw to Truecolor converter tool takes the converted sfmov to mat header file, a.k.a. the sfmovName_1.mat file and converts the entire movie into one of three types of standard image file formats. Upon clicking on the menu tab, the user will be prompted with a file selection window. After choosing any converted mat format header file, the user will be greeted by a question box as seen in Figure A.9 below.



In which file type would you want to cast these images?

Figure A.9: *Raw to trueColor converter image type selection.*

Once the image conversion type has been chosen, a progress bar will appear. The files are stored in a folder named convertedFiles under the same folder as the mat file. Only the include and POD file data will be copied and BMP format will not include these metadata due to some EXIF and development time limitations.

A.2 Toolbar

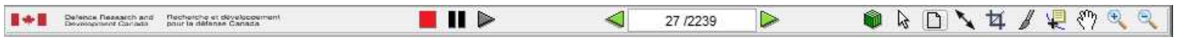

Defence Research and Development Canada
Recherche et développement pour la défense Canada
27 /2239

Figure A.10: *Application main toolbar.*

The Figure A.10 above shows the application main toolbar. It is comprised of standard and specialized interactive tools activated by buttons that are defined below:

These are common toggleable Matlab graph tools. The user can zoom, pan or pin datacursor on the main image with these. See Matlab documentation for more info.

This tool gives the user the possibility to select four points in the image, forming a polygonal region. This region can then be cropped or stored in the region list.

This tool will crop the current selected region, whichever is from the toolbar selection tool or from a selected region of the region list.

This tool toggles the aspect ratio of the current image. It iterates between the original ratio and the full size of the image view box.

This tool will prompt the user with a file selection tool to load any image of the movie without browsing through the entire list.

The pointer tool is only the normal cursor. It resets every toolbar activated action if needed. It can be considered as a panic button.

The undock button will prompt the user with a new figure containing the current image. This figure will contain all the standard Matlab options, menus and toolbar tools.

27 /2239

This tool gives the user information on the current file selected and the total file contained in the movie. The user can browse the images by clicking on the left and right green arrows.

The movie tools give the possibility to watch the movie at the maximum speed at which the computer can load the images. There is no underlying buffering system in place to maximize reading speed. The play button will browse the image chronologically, the pause button will stop on the current played image and the stop button will stop the play and reload the first image of the movie.

A.3 General tools panel

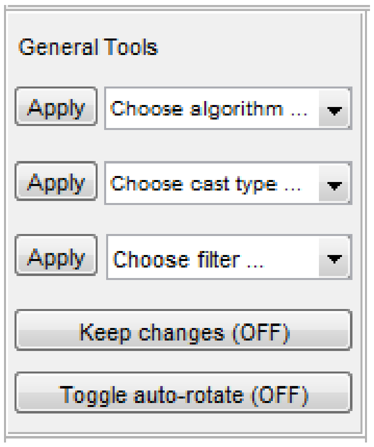


Figure A.11: *General tool panel.*

The Figure A.11 on the left shows the general tool panel. The cast type and the filter drop-down menus are common tools used in image analysis. The user can cast from any standard color types to any other. The supported color types are BGR, HSV, RAW, YCbCr and Grayscale. There are also nine two dimensional filters provided, which are named average, disk, Gaussian, Laplacian, log, motion, Prewitt and Sobel. For more details on these filters, see the Matlab fspecial() function documentation from the image processing toolbox. The two buttons at the base of the panel are toggleable. The keep changes button, once 'ON', keeps track of the application buffer while browsing through images. It means that if any operations have been made to the central image, these changes will be kept and applied to another image while browsing. The auto-rotate button toggles between a beta system. When regions are cropped, it will try to adjust the region's polygon to the horizontal line by using the polygon center lines. It works well, but is bound to the user definition of the polygon so another option has been added in the rotation panel for manual rotations.

The last drop-down menu at the top of the panel contains miscellaneous algorithms. These are consistent with the default customizable algorithms from the region list compare tool. There is the row, column and region cumulative histograms, the Hough and Fourier transform and the wake to background contrast algorithms. There are also three other tools that were used in the development process. The last option in the list is named 'custom scripts'. This will be detailed with the custom algorithms in Annex B on Page 13. It provides a lot of flexibility as the user can elaborate its own algorithm and use it with all the other functionality. The custom script is even stored in the modification buffer for temporal analysis, so the changes made in these algorithm will persist on the next files.

A.4 Main image view

The main view speaks for itself as it's the main viewer of the application where the currently loaded image is displayed. Its only feature will be discussed in the status bar section because when hovering the mouse inside the main view box, it produces positioning data in this latter.

A.5 Metadata panel

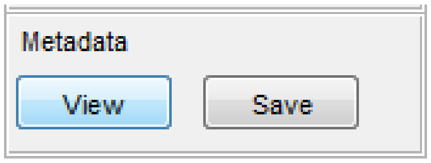


Figure A.12: *Metadata Panel.*

The Figure A.12 on the left shows the Metadata panel. There are only two options and they are linked with the current image displayed. The view button will prompt the user with popup boxes containing all of the available metadata for this current image. The save button will prompt the user to choose a path and a file name. Then the available metadata will be written in a structured manner on disk in a Matlab mat format file that can be loaded anytime in the Matlab workspace environment by using the following command: load('filename.mat'). Note that if the current image is from a converted sfmov movie, all the available metadata will be copied, even the optional file data that have been synchronized. There are no limitations like the case of the Raw to Truecolor converter tool.

A.6 Rotation panel

The Figure A.13 on the right shows the Rotation panel. This is a manual tool to actually rotate the currently loaded image. The user can scroll from -180 degrees to 180 degrees. The degree number will appear in the box on top of the scrollbar and will be applied when the user clicks on the apply button. The changes can be viewed on the current image as the user scrolls.

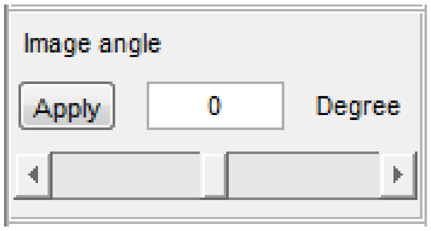


Figure A.13: *Rotation Panel.*

A.7 Region list Panel

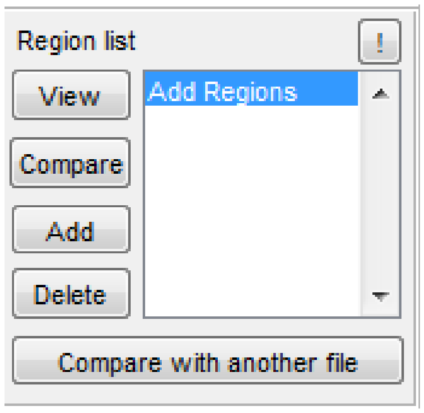
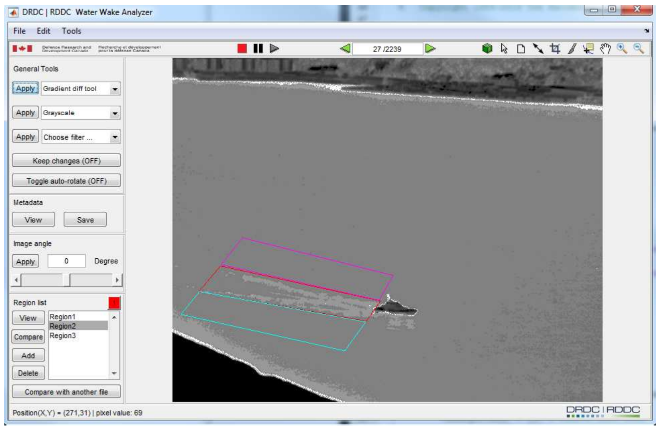


Figure A.14: *Region list Panel.*

The Figure A.14 on the left shows the Region list panel. When the user selects a polygonal region with the toolbar selection tool, if the Add Button from this panel is pressed afterward, the region will be stored in the scrollable area on the right. Clicking on any region stored in this list will make it reappear on the current image. While a region is selected in this manner, the user can crop it, remove it from the list with the delete button or press the view button to prompt a new figure with the cropped region displayed. To help with region selection precision, the little '!' button on the top right corner will display all the region at once on the current.

image when pressed. It is toggleable and it will turn red when it is 'ON'. The feature can be described better by taking a look at Figure A.15 down below. The different regions can be seen at once with different color codes, thus adding precision when selecting regions.

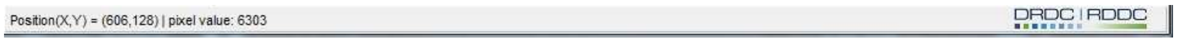


Position(X,Y) = (271,31) | pixel value: 69

Figure A.15: *Mutliple regions selection displayed.*

There are two other buttons to compare the regions. Both will prompt the user with the compare tool figure containing the user customizable algorithms. The difference between those two buttons is that one compares region from the current image displayed and the button at the bottom of the panel lets the user choose another file to compare regions with. The logic behind this and much more is discussed in Annex B on Page 13 since its complexity demands a better and in-depth explanations.

A.8 Status bar



Position(X,Y) = (606,128) | pixel value: 6303

Figure A.16: *Application main status bar.*

The Figure A.16 above shows the application main status bar. Its main purpose is to display important messages. There are two main messages that can be displayed on it. When loading files from any sources, any successful attempt will be documented in this status bar for a limited time. Another interesting use is when an image is displayed. The Windows mouse event is redirected to a custom function which calculates the position in real-time. This position is displayed as X and Y Cartesian coordinates with its origin at the top left corner of the image. When possible, the pixel value in 14 bits radiance counts will be displayed on the right of its position. Grayscale color type is an example of impossible casting due to its lack of data integrity.

This page intentionally left blank.

Annex B Custom algorithm tool guide

The application has two fully customizable systems that gives almost a perfect freedom to the user in term of analysis evolution and research. There is a function generator for the comparison system that comes with the region panel. That system is quite simple, yet requires some basic knowledge of the underlying system to master its features because such freedom comes at a certain cost of complexity. The second system is in the general tool section, it is not designed for comparison, yet still offers it at the cost of coding it yourself. That system is the custom script in the algorithm dropdown menu. It is lighter than the comparison system, although offers much more in terms of common tool readiness. The following guide will go through two use case scenarios for the comparison system. The first case will demonstrate each step on how to create your own algorithms. The second one will emphasize on the compare algorithm's protection logic by analyzing an already developed algorithm. The last section will focus on the custom script system. All this will take into consideration that the user has already been familiar with the Matlab's image processing toolbox.

B.1 Walkthrough of the compare algorithm system

Start by clicking on the 'Add Custom Algorithm'

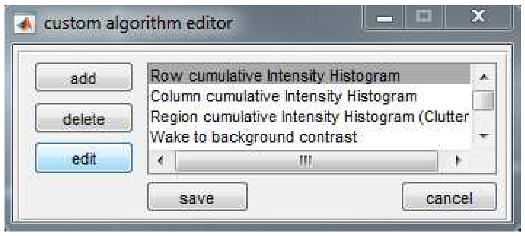
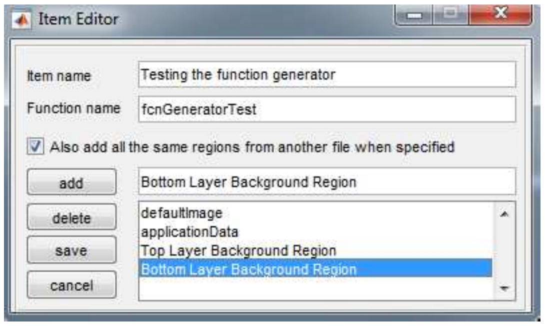

custom algorithm editor
add
delete
edit
Row cumulative Intensity Histogram
Column cumulative Intensity Histogram
Region cumulative Intensity Histogram (Clutter
Wake to background contrast
save
cancel

Figure B.1: *Custom algorithm editor too.*

The option is in the Tools tab of the application main menu and should prompt the user with an interface like the one seen in Figure B.1 on the left. As it can be seen, a list of algorithm examples comes with the application. Any function can be added, removed or edited with this tool. For the benefit of this guide, we will now create a new algorithm.

Clicking on the add button will open an item editor

The Figure B.2 on the right shows the definition of a function's elements. The item name is what will be shown to the user in any list of the application to identify the algorithm. The function name is the file name of the function that will be auto-generated. The check box can be quite confusing, but it means that if the user uses the 'compare with other file' button from the region list panel, additional region will have to be selected for each region of each image. In this example, we will define two specific regions, thus meaning that when the user will use this function, he will have to choose two regions in the current image and two other regions in the compared file image. If the option was unchecked, the same regions would be taken by default on both images. The list at the bottom is the function parameters. The 'defaultImage' and 'applicationData' are there by default but can be deleted. They are the primary region selected and the software data structure. Every other parameter added to the list will be considered as an additional region prompted as a choice to the user while using the compare tool. Keep in mind that the names entered here are exactly those that will be asked to the user upon using the compare tool. To add a parameter, write the name in the box on the right of the add button and press the add button. In this case, we added the 'Top Layer Background Region' and 'Bottom Layer Background Region' parameters. When ready, press the save button and you will be sent back to the custom algorithm editor window.

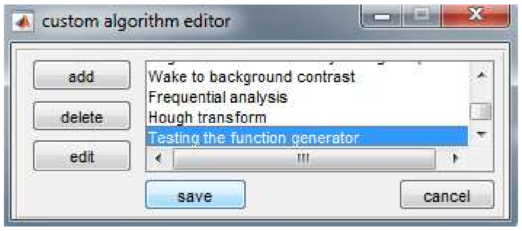


Item name: Testing the function generator
Function name: fcnGeneratorTest
☑ Also add all the same regions from another file when specified
Bottom Layer Background Region

defaultImage
applicationData
Top Layer Background Region
Bottom Layer Background Region

Figure B.2: Custom algorithm item editor function generation example.

Save your new function definition so it can be auto-generated



Wake to background contrast
Frequential analysis
Hough transform
Testing the function generator

Figure B.3: Custom algorithm editor tool function generation example.

You should now see a new entry in the list corresponding to the item name you gave it. In this case we can see the new 'Testing the function generator' entry. in the figure B.3 on the left. Now by clicking on the save button, a new function should be generated under the following folder:

src/bin/customAlgorithms/

The file should have the name given as function name, which is in this case: 'fcnGeneratorTest.m'.

Lets analyze the auto-generated script

Code generated in the src/bin/customAlgorithms/fcnGeneratorTest.m file:

```
 function fcnGeneratorTest(img_in, applicationData, region1, regionFromOtherFile1, ...
                                    region2, regionFromOtherFile2, otherFileImage)
% function argument name can be changed but order must be respected
% to ensure scripts correlation.
```

Here are the parameters explanation:

- img_in: The 'defaultImage' parameter from the item editor, it corresponds to the primary region selection.
- applicationData: The 'applicationData' parameter from the item editor, it corresponds to the software data, containing the current data and metadata on the current displayed image (see Annex C on Page 21 for more details).
- region1: The 'Top Layer Background Region' parameter from the item editor, an additional region selection that will be prompted to the user.
- regionFromOtherFile1: The 'Top Layer Background Region' parameter from the item editor, an additional region selection that will be prompted to the user if he is comparing with another file.
- region2: The 'Bottom Layer Background Region' parameter from the item editor, an additional region selection that will be prompted to the user.
- regionFromOtherFile2: The 'Bottom Layer Background Region' parameter from the item editor, an additional region selection that will be prompted to the user if he is comparing with another file.
- otherFileImage: The region corresponding to the primary selected regions from the compared image file.

How to use it:

The Figure B.4 on the right is what will be prompted to the user upon using one of the two compare buttons from the region list panel. The user must make a selection of the primary region to compare from the top left drop-down menu and the algorithm to apply from the bottom left drop-down menu. Then, by clicking on the 'compare with...' button, selecting additional regions will be asked depending on the number of additional parameters. In this case, if the user chose a normal comparison, only two additional image selection will be prompted. In the case of another file comparison, two additional selections for the compared file would be asked.

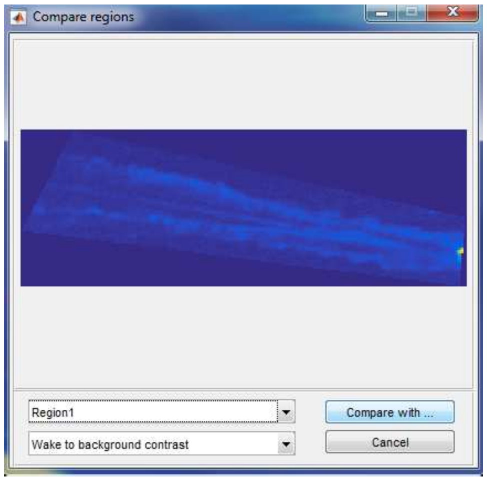
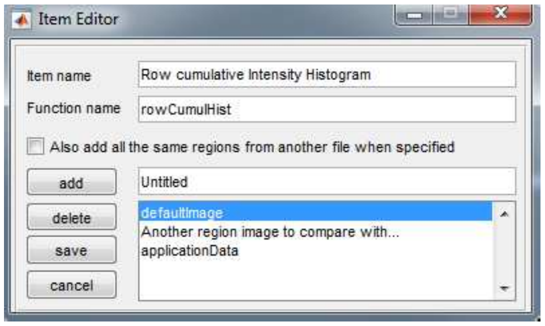


Figure B.4: *Compare tool user interface.*

B.2 In-depth compare algorithm analysis

In this section, we will analyze the case of the row cumulative intensity histogram algorithm, which comes by default with the application, to ensure that the ways to work with the auto-generated functions are clear.


Item Editor

Item name: Row cumulative Intensity Histogram

Function name: rowCumulHist

☐ Also add all the same regions from another file when specified

Untitled

defaultImage
Another region image to compare with...
applicationData

Figure B.5: *Custom algorithm item editor example.*

The Figure B.5 on the left is the item editor for the row cumulative intensity Histogram. This function does a histogram of the cumulative radiance value found for each row in an image region. It does not ask for any other region while comparing with another file and only asks for one additional region to compare with in the case of comparing with itself. Basically, it compares two regions of the same image or the same region from two different files because the option is unchecked.

Lets analyze the script from rowCumulHist.m file by code blocks:

Block 1

```
function rowCumulHist(in_img, applicationData, region1, compareImage)

if isempty(compareImage)
    compareImage = region1;
end

imageColorType = applicationData.imageColorType;
```

In this block we can see that a protection is made to verify if the user is comparing with another file or not. If the compareImage parameter is empty, then the user is comparing with the current image and then to ensure the right comparison with the right image we copy the selected region1 into the compareImage, which will be the variable to compare to in the following code. This type of comparison has to be made in the algorithm to ensure a certain freedom of choice in the application, so it is the algorithm responsibility to be thorough. The second part copies the in_img color type from the application data structure.

Block 2

```
% cast to grayscale if rgb
[M1, ~, P]= size(in_img);
in_image = in_img;
if P > 1
    in_image = toRgb(in_image, imageColorType);
    in_image = rgb2gray(in_image);
end

[M2,  , P] = size(compareImage);
in_compare_image = compareImage;
if P > 1
    in_compare_image = toRgb(compareImage, imageColorType);
    in_compare_image = rgb2gray(in_compare_image);
end
```

In this block, the primary region and the comparing region are copied into new variables, so we can modify them and still keep the original for display purposes at the end of the script. Then they are cast to grayscale if they are a Truecolor. This way we will only analyze one dimensional images, which are raw or grayscale. The 'toRgb' function is a proprietary function of this application, by specifying the current color type it will return a standard RGB output.

Block 3

```
 % get row histograms
histVector1(1:M1) = 0;
for i = 1 : M1
      histVector1(i) = sum(in_image(i, :));
end

histVector2(1:M2) = 0;
for i = 1 : M2
      histVector2(i) = sum(in_compare_image(i, :));
end
```

In this block, we sum each row into an histogram vector, this is actually the computational part. These histograms will then be plotted in the next block section.

Block 4

```
 % set plot text
graphTitles = {'Fit curves of horizontal cumulative intensities', ...
                 'Histograms of horizontal cumulative intensities'};
axLabels = {'Row count (pixel)', 'Cumulative intensities'};

figure
subplot(2,2, 1), plot(histVector1, 'Color', 'b'), hold on,
                  plot(histVector2, 'Color', 'r'), hold off,
                  xlabel(axLabels{1}), ylabel(axLabels{2}), title(graphTitles{1});

subplot(2,2, 2), bar(histVector1, 'b'), hold on,
                  bar(histVector2, 'r'), hold off,
                  xlabel(axLabels{1}), ylabel(axLabels{2}), title(graphTitles{2});

subplot(2,2, 3), imgView(in_img), ...
                  xlabel('x axis (pixel)'), ylabel('y axis (pixel)'),
                  title('1st Region - Blue color');
                  caxis auto, colormap default, daspect([1 1 1]);

subplot(2,2, 4), imgView(compareImage),
                  xlabel('x axis (pixel)'), ylabel('y axis (pixel)'),
                  title('2nd Region - Red color');
                  caxis auto, colormap default, daspect([1 1 1]);
```

In this block, we create labels for the axes and then plot the histograms and original image for the primary region and the region compared to on the same figure.

This annex thus demonstrates so far the actual level of code logic to support for any custom compare algorithms. The input images are not pre-treated in any way and the user has to take care of the way to display or save the data after the computation. The upper layer tool only serves as a customizable entry way to go from the actual displayed image and its region of interest to a fully customizable comparison platform. For in-depth debugging, a pop-up message system has been developed to help and inform the user on the current error(s). Just answer yes for more detailed error messages when it occurs. For more examples on how to work with customizable compare algorithms, browse the 'customAlgorithms' folder for other default script examples.

B.3 Custom Script system

The last customizable system is in the the general tool panel and allows system wide modifications. The user must first create a Matlab script file named 'customScript.m' with the following header:

function [outputImage] = customScript(applicationData)

The output and input names may vary but the input will be of type application data structure which is defined in Annex C on Page 21 and the output must be an image. Also, the file must be put in the 'customAlgorithms' sub-folder and only one script of this type can be present at a time. These are the only restrictions, everything else is possible. If the user does not wish to modify the image, the input image can be returned to prevent modifications and anything else can be done like showing a figure or saving data. As described in the Annex C, the currently displayed image can be accessed in applicationData.currentImageDisplayed.

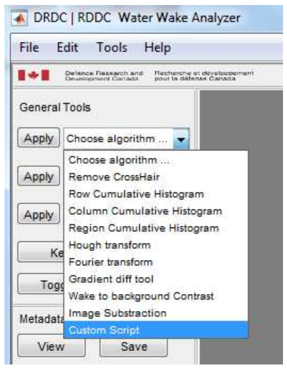


Figure B.6: *Custom script from the general tool panel.*

The Figure B.6 on the left shows how to use the custom script made with the technique shown in this section. This dropdown menu is accessible from the general tool panel on the left of the main display. If the image returned from the script is modified in any way, it will be displayed in the main view. These modifications will be persistent if the user chooses to click on the 'keep Changes' button. they will be kept in the system buffer and allow temporal comparison. In fact, this system is even more flexible than the comparison system as it also has the application data structure and thus the user has access to the region data to do exactly as in the compare system. It requires more work, but with more flexibility comes more complexity.

This page intentionally left blank.

Annex C Software Data Accessing Guide

The Ship Wake Analyzer software is based on the MVC design pattern and thus has a singleton type of data structure object passed over the entire application via a bus. The overall structure of this object is shown in Figure D.3 on Page 29. The object is a 'struct' data type with over 80 fields, but only about 10 of them are explained thoroughly in this document. This is due to the fact that others are GUI related and need more in-depth knowledge on Matlab's GUIDE data structure. It was easier to pass the entire application data to support future development, but the user must then be aware of the consequence of modifying these UI components. Figure, UIControl, Menu, Axes and Panel objects are JAVA UI components that should not be modified in any circumstances, although having access to information from one of the main application component may prove useful in a future development stage and is why it has been left available. The following guide will focus on how to access information from the 'applicationData' parameter passed on to generated functions of the custom algorithm system.

C.1 UI component accessing

Let's say the user would want to access the image data property of the main image axis. He would have to use Matlab's GUIDE editor property inspector to find the component name and property name to be able to access it. The information and way to access it would be as follow:

- Axes name: Image_Axes
- Image data parent property: Children
- Image data property: CData

Accessing it from the 'applicationData' parameter:

```
image = applicationData.Image_Axes.Children.CData;
```

Now, the fact is that the most important information is already at hand. It can be accessed with the fields that will be shown in the next section. For example, the image the user just accessed could have been accessed without doing an application wide search with this simple call:

```
image = applicationData.currentImageDisplayed;
```

C.2 Image related data

The following section will focus on the elements shown in Figure D.3, how to access them and their sub-layers and their respective information.

- applicationData.currentImageDisplayed: This is actually the currently displayed image in the main application as it is. Its data type is a 2D array of double.
- applicationData.currentImageRaw: This is the Raw sampled version of the currently displayed image in the main application or a copy of it if it is already a Raw format. Its data type is a 2D array of double.
- applicationData.imageColorType: This is the currently displayed image in the main application color type as a char array data type. It can be one of the following 'raw', 'rgb', 'hsv', 'ycbcr', 'grayscale'.
- applicationData.polyRotate: This boolean value indicates if the beta polygon center-line rotation algorithm is currently active.

Most other elements should not be accessed as they are mostly graphical elements and are more relevant to the software operation logic. However, the last three elements, which are the WorkDir, buffer and regions 'struct' data types will be defined in the following sub-sections since they have themselves sub-layers of complexity.

C.2.1 WorkDir struct

The WorDir struct is composed of 7 describable fields defined as:

- applicationData.WorkDir.name: char array containing the absolute folder path of the work directory
- applicationData.WorkDir.fileIndex.current: The current file index of the currently displayed image in the main application. Its data type is an integer.
- applicationData.WorkDir.fileIndex.length: The number of images contained in the work directory. Its data type is an integer.
- applicationData.WorkDir.FileList.name: A cell array of file names as char array data type for every image in the work directory. If the images are Raw, it contains the header file names for the images.
- applicationData.WorkDir.CurrentImageSize: The size of the currently displayed image in the main application. It contains an array of two integers representing the coordinates as follows [x, y].
- applicationData.WorkDir.headerFileName: The filename of the currently displayed image in the main application or the header file name if the image is Raw. Its data type is a char array.

- applicationData.WorkDir.isRawData: This boolean value indicates if the data is originally Raw sampled.

Loading current image original state code example:

```
 if applicationData.WorkDir.isRawData
        image = loadRawImage(applicationData, applicationData.WorkDir.fileIndex.current);
else
        pathName = strcat(applicationData.WorkDir.name,
                                      applicationData.WorkDir.FileList.name{
                                      applicationData.WorkDir.fileIndex.current})
        image = load(pathName);
end
```

C.2.2 buffer struct

The buffer struct is composed of 1 useful field, although there is 7 in total. The other fields are mostly used for operating the buffer logic throughout the application. The application.elements field is more interesting as it contains a list in the form of cell array data type of all the changes that were made to the original image. An element is composed of 3 fields itself:

- applicationData.buffer.elements{elementNb}.image: The image after the modification as a 2D array of double.
- applicationData.buffer.elements{elementNb}.colorType: The image color type after the modification as a char array.
- applicationData.buffer.elements{elementNb}.operation: The actual modification as a struct.

The operation struct can vary depending on which modification was applied. The struct always has 2 fields itself. Those are the ‘name’ and the ‘details’. The name is always a char array but the details vary by their name type. Here are the possibilities:

for applicationData.buffer.elements{elementNb}.operation.name
and applicationData.buffer.elements{elementNb}.operation.details

- name = ‘init’: No details
- name = ‘cast’: details = color type as a char array data type
- name = ‘crop’: details is a struct with four fields
  - imageColorType: The image color type before the crop as a char array
  - cropZone: The region crop zone in an array of 8 integer
  - cropRect: The region crop rectangle in an array of 4 integer
  - polyRotate: Boolean if the option was on when cropping
- name = ‘rotate’: details = angle value as an integer data type
- name = ‘crossRemoval’: No details
- name = ‘grayDiff’: details is a struct with two fields
  - threshold: The gray diff tool threshold value used as a double
  - cutoff: The gray diff tool cutoff value used as a double
- name = ‘filtering’: The type of filter as a char array data type

C.2.3 regions struct

The regions struct is the information with which is defined the region panel of the main application. It is composed of 4 describable fields as follows:

- applicationData.regions.data: A cell array of crop rectangles which are arrays of 4 integers. These can be linked by indexes to there respective names in the list
- applicationData.regions.length: The number of regions in the list as an integer
- applicationData.regions.list: A cell array of char array containing the list of region names
- applicationData.regions.color: The application color list as a cell array of chars

The ‘keptLines’ and ‘keepRegions’ fields are more related to the application operation logic. They are related to the multi-regions display system.

C.3 How to access metadata

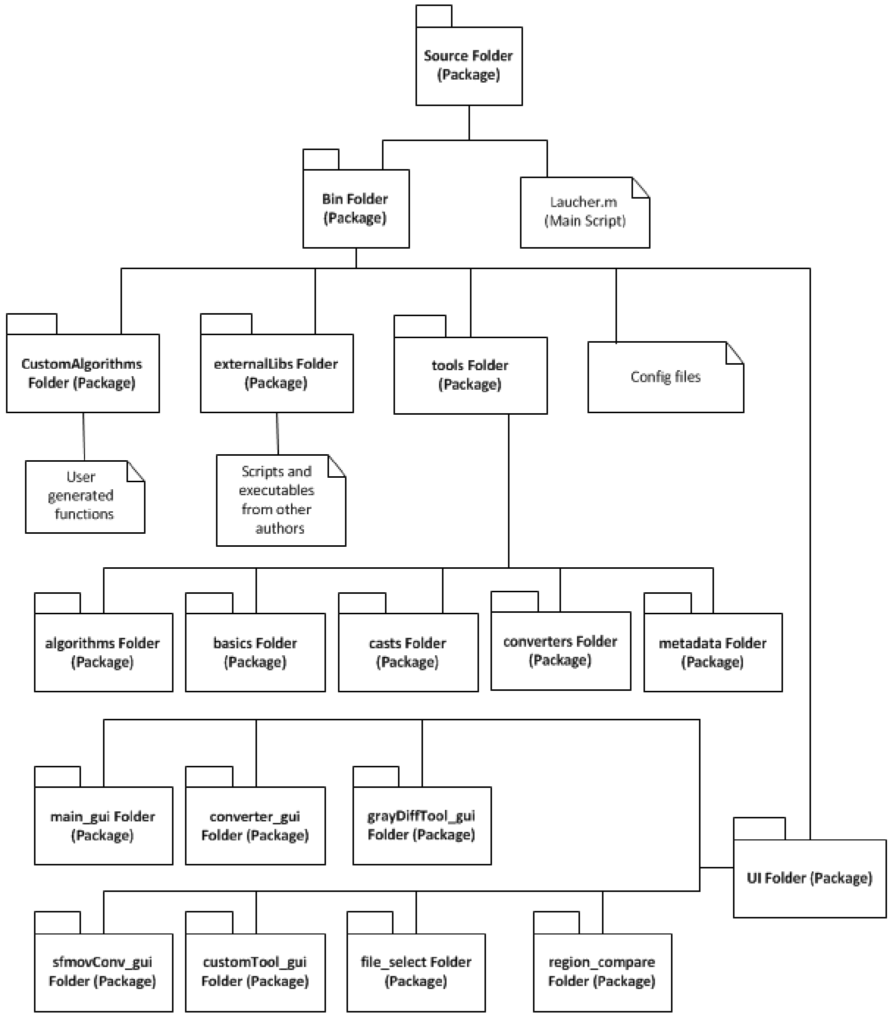
Metadata is not loaded in memory as it would take too much space in the case of raw data and also because there is too many variants of metadata. To cope with this problem, a function was developed to make this as simple as possible.

Here's an example for getting the current file metadata:

```
metadata = getMetaData(applicationData, applicationData.WorkDir.fileIndex.current, false);
```

This page intentionally left blank.

Annex D Software Architecture Diagrams



Source Folder (Package)
Bin Folder (Package)
Laucher.m (Main Script)
CustomAlgorithms Folder (Package)
externalLibs Folder (Package)
tools Folder (Package)
Config files
User generated functions
Scripts and executables from other authors
algorithms Folder (Package)
basics Folder (Package)
casts Folder (Package)
converters Folder (Package)
metadata Folder (Package)
main_gui Folder (Package)
converter_gui Folder (Package)
grayDiffTool_gui Folder (Package)
UI Folder (Package)
sfmovConv_gui Folder (Package)
customTool_gui Folder (Package)
file_select Folder (Package)
region_compare Folder (Package)

Figure D.1: *Source folder main packaging.*

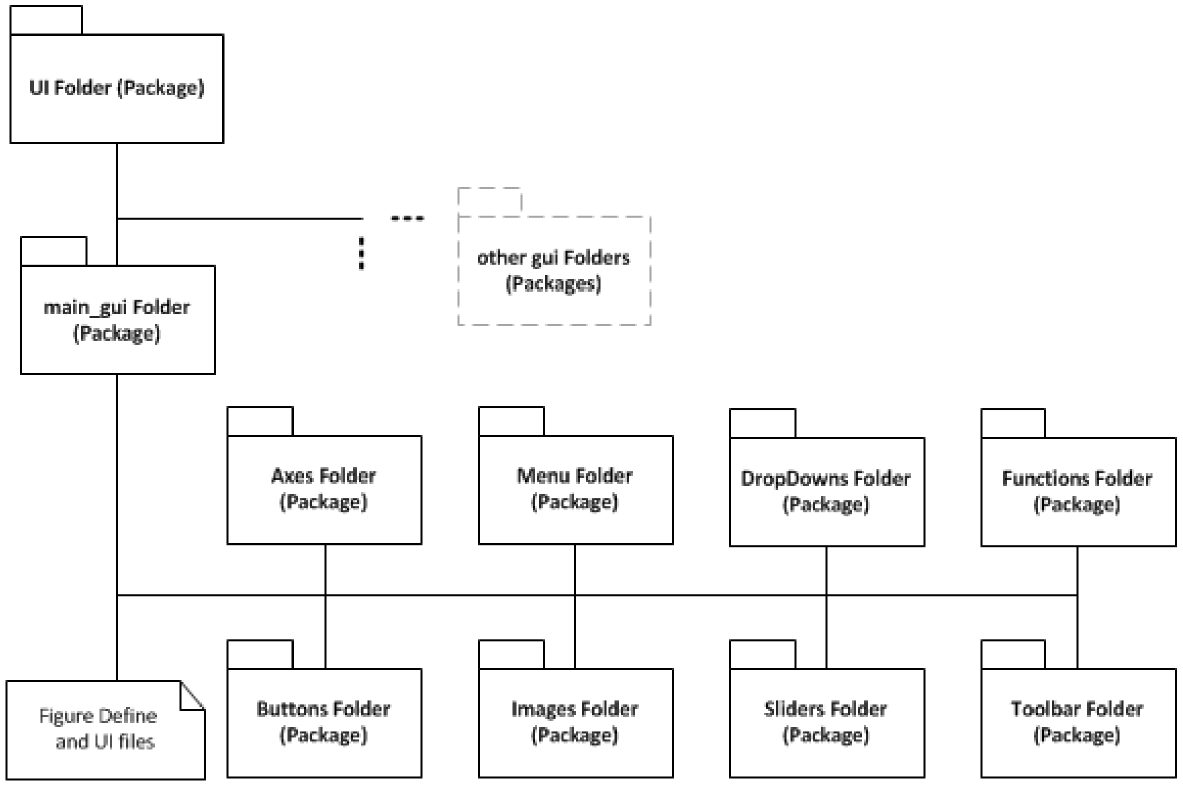
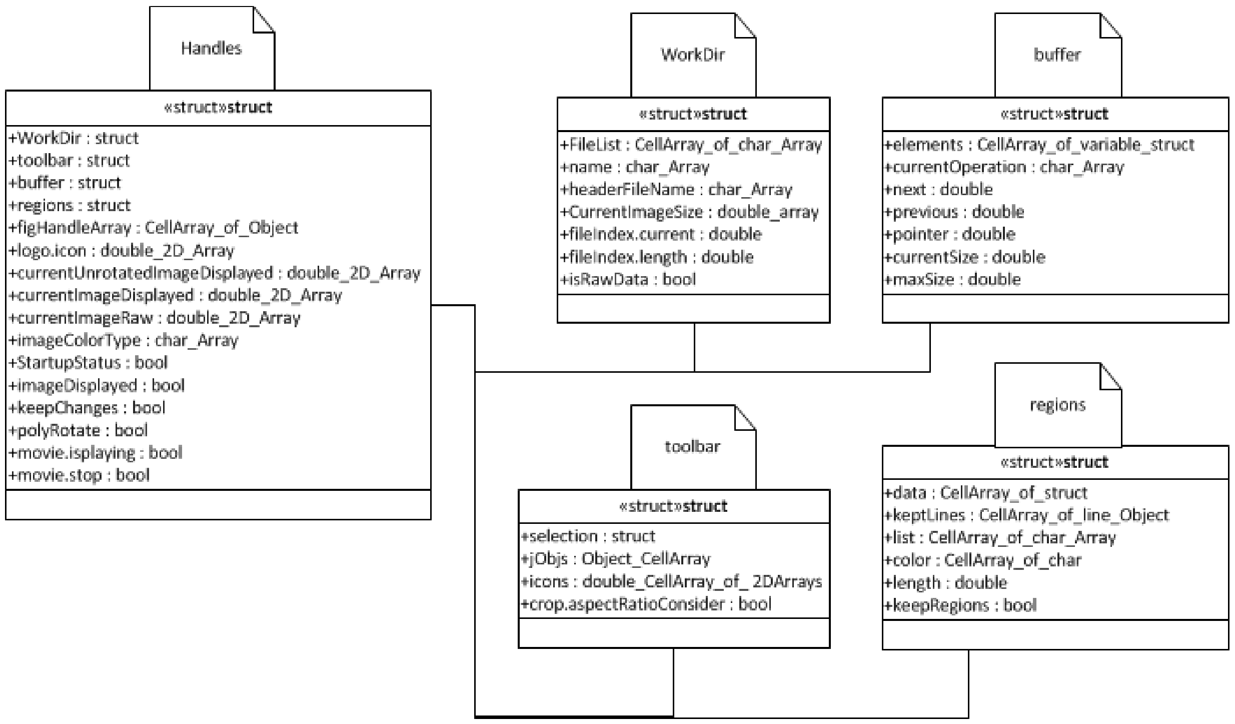


Figure D.2: *Source folder gui packaging.*



Handles
«struct»struct
+WorkDir : struct
+toolbar : struct
+buffer : struct
+regions : struct
+figHandleArray : CellArray_of_Object
+logo.icon : double_2D_Array
+currentUnrotatedImageDisplayed : double_2D_Array
+currentImageDisplayed : double_2D_Array
+currentImageRaw : double_2D_Array
+imageColorType : char_Array
+StartupStatus : bool
+imageDisplayed : bool
+keepChanges : bool
+polyRotate : bool
+movie.isplaying : bool
+movie.stop : bool
WorkDir
«struct»struct
+FileList : CellArray_of_char_Array
+name : char_Array
+headerFileName : char_Array
+CurrentImageSize : double_array
+fileIndex.current : double
+fileIndex.length : double
+isRawData : bool
buffer
«struct»struct
+elements : CellArray_of_variable_struct
+currentOperation : char_Array
+next : double
+previous : double
+pointer : double
+currentSize : double
+maxSize : double
toolbar
«struct»struct
+selection : struct
+jObjs : Object_CellArray
+icons : double_CellArray_of_ 2DArrays
+crop.aspectRatioConsider : bool
regions
«struct»struct
+data : CellArray_of_struct
+keptLines : CellArray_of_line_Object
+list : CellArray_of_char_Array
+color : CellArray_of_char
+length : double
+keepRegions : bool

Figure D.3: *Main UI Data Structure.*

This page intentionally left blank.

Annex E Helpful Function List

This section of the appendix is reserved for the definition of developed functions that are reusable in a custom algorithm.

- *[img_ out] = affineBkgrndPixels(img_ in , cutOff)*
  This function retrieves the lowest nonzero pixel element value and transforms all the zero pixels into the minimum value minus the defined cutoff value elements. This way the images don't have high contrast and it prevents colormap extreme scale jumps giving unwanted visual results. It actually smoothens the visual effect and afterwards the minimum value in an image is to be considered as its background value.

- *[out_ img] = applyMask(image, imageColorType, mask)*
  This function applies a binary mask on the input image depending on its color type

- *[errStatus] = copyExif(fileName, newFileName)*
  This function copies EXIF metadata from an image file to another. Filenames must be absolute paths. The error status returned is 0 if everything went ok.

- *[RGB_ image]= demosaicing(Image)*
  This function transforms a 14bits Raw image into truecolor rgb type image using rggb inverse bayer filter mosaicing algorithm

- *dispDetailedError(exception)*
  This function displays the basic thrown error and asks the user if it wants more details, if so, all the extended exception is shown with a maximum of 5 errors by pages. Exception input must be an object of type MException received from a try/catch prerogative.

- *[value] = findKeyVal(keys, values, keyName)*
  This function finds a value associated with a keyName and returns its value. keys and values must be cell arrays of char array and keyName also a char array.

- *[colorType] = getColorType(image)*
  This function returns an image color type. It cannot detect YCbCr and HSV formats.

- *[exifData, err] = getexif(fileName)*
  This function recovers EXIF metadata from an image file if error status is 0. fileName must be an absolute path.

- *[metadata] = getMetaData(applicationData, fileIndex, isMessage)*
  This function returns metadata of a chosen file in the form of a struct or a char array with the standard EXIF tagging system.

- *[mask, vertices] = getPolyMask(image, cropZone)*
  This function returns a binary mask of a closed polygon and its vertices formed of corners coordinates of his cropZone in an image.

- *[rotAngle] = getPolyRotateAngle(vertices)*
  This function computes the centerlines angles of a polygon based on the input vertices.

- *[cropRect] = getRectFromRotation(rotatedImage)*
  This function retrieves the minimum size of the bounding box necessary to fit a rotated polygon masked image.

- *[rect] = getRectFromZone(zone)*
  This function retrieves the minimum size of the bounding box necessary to fit the specified zone.

- *[region_image] = imageLoad(data, 'explanations')*
  This UI allows the user to interactively choose a region from the region list for further analysis.
  @Param data = a structure with the following application data reference:
  data.regions = handles.regions.data;
  data.image = handles.currentImageDisplayed;
  data.compareImage = [];
  data.colorType = handles.imageColorType;
  data.polyRotate = handles.polyRotate;
  @Param 'explanations' = A string that will follow the "Please select: " title to explain what kind of region to choose.

- *[h] = imgView(varargin)*
  This function is a copy of the matlab well known imagesc. It overrides the function by doing an affine background check to render a better visual effect by scaling the image colormap appropriately. For mor details on input/output, consult imagesc documentation.

- *[valid] = isValidCropRect(cropRect, imageSize)*
  This function verifies if a crop rectangle is out of image boundaries

- *[Img] = loadRawImage(applicationData, fileIndex)*
  This function loads a raw image from a datafile by specifying the file index.

- *[output] = makeRegularFieldName(fieldName)*
  This function ensure a regular field name is used by removing irregular characters from the input string.

- *[keys, values] = parseKeyVal(inputCellArray)*
  This function parses a cell array from a meta header string into a key and a value list.

- *[out_img, err] = polyCenterLineCrop(in_image, colorType, cropZone, isRotate)*
  This funtion finds a polygon in an image defined by the cropZone, crops it from its cropRect equivalency and then finds the centerline of the polygon, rotates it and crops back the exceeding data. The isRotate argument is part of the beta automatic rotation system.

- *[status] = putRawExif(fileName, includeString, podData)*
  This funtion writes raw metadata onto a target image file if status returns 0. fileName must be an absolute path to the file to copy metadata to. include metadata must be a string formatted and the POD metadata must be struct formatted.

- *[raw] = rawSampling(rgb)*
  This function does a reverse Bayer filtering and a 14Bit sampling to transform a truecolor raster image type into a raw image matrix format. The conversion type used is 'rggb'.

- *[ImageOut] = setImageType(image, newType)*
  This function casts any image type to the desired newType in the list of {'grayscale','hsv','ycbcr'}. The input image must be of type 'rgb'.

- *setScreenPos(figureObject)*
  This function finds the parent figure center and recenters the current figure on screen.

- *[outputImage] = smoothen(Image, boundingRectangle, precision)*
  This function smoothes the pixels around a rectangle in an image. The precision is the number of neighbouring pixel used to smoothen and must be an odd number.

- *[rgb] = toRgb(image, oldColorType)*
  This function casts any image type to truecolor rgb type.

DOCUMENT CONTROL DATA

*Security markings for the title, authors, abstract and keywords must be entered when the document is sensitive

1. ORIGINATOR (Name and address of the organization preparing the document. A DRDC Centre sponsoring a contractor's report, or a tasking agency, is entered in Section 8.) DRDC – Valcartier Research Centre 2459 de la Bravoure Road, Québec QC G3J 1X5, Canada	2a. SECURITY MARKING (Overall security marking of the document, including supplemental markings if applicable.) CAN UNCLASSIFIED 2b. CONTROLLED GOODS NON-CONTROLLED GOODS DMC A

3. TITLE (The document title and sub-title as indicated on the title page.)

Ship Wake Data Analyzer

4. AUTHORS (Last name, followed by initials – ranks, titles, etc. not to be used. Use semi-colon as delimiter)

Bélanger, K.; Issa, V.

5. DATE OF PUBLICATION (Month and year of publication of document.) February 2019	6a. NO. OF PAGES (Total pages, including Annexes, excluding DCD, covering and verso pages.) 35	6b. NO. OF REFS (Total cited in document.) 0

7. DOCUMENT CATEGORY (e.g., Scientific Report, Contract Report, Scientific Letter)

Reference Document

8. SPONSORING CENTRE (The name and address of the department project or laboratory sponsoring the research and development.)

DRDC – Valcartier Research Centre
2459 de la Bravoure Road, Québec QC G3J 1X5, Canada

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.) 01ec	9b. CONTRACT NO. (If appropriate, the applicable contract number under which the document was written.)
10a. DRDC DOCUMENT NUMBER DRDC-RDDC-2018-D141	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)

11a. FUTURE DISTRIBUTION WITHIN CANADA (Approval for further dissemination of the document. Security classification must also be considered.)

Public release

11b. FUTURE DISTRIBUTION OUTSIDE CANADA (Approval for further dissemination of the document. Security classification must also be considered.)

Public release

12. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Use semi-colon as a delimiter.)

ship
wakes
Data

13. ABSTRACT/RÉSUMÉ (When available in the document, the French version of the abstract must be included here.)

In support of the Project 01ec a tool is required to facilitate trials data analysis. Wake measurement trials collect raw infrared data images, environmental and climatic data, sensor heading and specifications, naval platform and helicopter GPS coordinates, sea temperature and sea stratification and time. These data collected with multiple sensors has wide range of time resolution. The objective of the Wake Analyzer software is to convert all raw data to the format selected by the user, synchronize all the data collected from different instrument in order to allow automatic processing. The tool, which is a MATLAB graphic user interface, allows the user to open each image and access to all associated Meta data. It allows also selecting one or multiple parts of the image and applying specific algorithms to them. This reference document shows the user guide of the Wake Analyzer tool.

En support du projet 01ec, un outil est requis pour faciliter l'analyse des données des essais expérimentaux. Les essais de mesures de sillages collectent des données brutes infrarouges, des données environnementales et climatiques, des données GPS du Platform navale et de l'hélicoptère, de l'orientation du capteur et ses spécifications, de la température de la mer et sa stratification ainsi que l'heure. Ces données sont collectées par des différents capteurs et ne sont pas synchronisées temporellement. L'objectif du logiciel Wake Analyzer est de convertir les données brutes au format choisie par l'utilisateur, synchroniser les données collectées pour permettre un traitement automatique. L'outil, qui est une interface graphique MATLAB, permet à l'utilisateur d'ouvrir les images individuellement et d'accéder à toutes ses métadonnées. Il permet aussi de sélectionner une ou plusieurs parties de l'image pour y appliquer un algorithme spécifique. Ce document de référence montre le guide d'utilisateur de l'outil Wake Analyser.