

PROJECT DOCUMENTATION

DETECT, V_{3.1}

JON JUARISTI CAMPILLO,
MARC DELCROIX,
RICARDO HUESO

*Planetary Sciences Group,
UPV/EHU*

SUPERVISED BY
RICARDO HUESO

13 DE JUNIO DE 2018

Copyright © 2018 Jon Juaristi Campillo, Marc Delcroix, Ricardo Hueso

Note: DeTeCt3.1 has been developed from DeTeCt2.0 (originally written by M. Delcroix and partially based on algorithms developed by L. Calderon and R. Hueso at UPV/EHU). This version of the software was developed by Jon Juaristi and Marc Delcroix as part of the Planetary Space Weather Services (PSWS) on Europlanet-2020 RI. Europlanet 2020 RI has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 654208. The software is released as open software and can be modified by third parties.

Science is what we understand well enough to explain to a computer. Art is everything else we do.

— DONALD E. KNUTH

ÍNDICE GENERAL

Índice general	v
Índice de cuadros	vii
Índice de figuras	viii
1 Introduction	1
1.1. Changelog	1
2 Installation	3
2.1. Requirements	3
2.2. Initial setup	3
2.3. Start the project	3
2.4. First run	4
3 Internals	5
3.1. Major differences from previous versions	5
3.2. Interface: MFC	5
3.2.1. Main window	5
3.2.2. Preferences window	6
3.2.3. About window	7
3.2.4. Notification window	7
3.3. The new impact detection algorithm	8
3.3.1. Definition	8
3.3.2. Results	10
3.4. New functionalities and how to edit them	11
3.4.1. GUI	11
3.4.2. Algorithm	12
3.5. Thorough explanation of the source code	14
3.5.1. Table of files and purpose	14
3.5.2. auxfunc.h: auxiliary functions	14
3.5.3. Common: helper functions	16
3.5.4. Datation: datation functions	16
3.5.5. datation2: extension of datation	17
3.5.6. DeTeCt-MFC: main files	18
3.5.7. DeTeCt-MFCDlg: window/dialog functionality	18
3.5.8. DetectThread: native detection thread	18
3.5.9. Dircnt: third party native windows directory functions	19
3.5.10. dtc: legacy detection algorithm functions [unused]	19
3.5.11. dtcas3: Autostakkert!3 detection functions	19
3.5.12. dtcgui: Main detection functions	19
3.5.13. filefmt: legacy file management functions [un- sed]	20
3.5.14. filefmt2: file management functions	20

ÍNDICE GENERAL

3.5.15. fitsfmt: FITS management functions	22
3.5.16. img: Opencv Matrix operation functions (C API)	22
3.5.17. img2: OpenCV detection functions (C++ API) .	22
3.5.18. max: impact detection helper structures and functions	24
3.5.19. resource.h: MFC resource file	25
3.5.20. serfmt: SER file management	25
3.5.21. wrapper: File capture wrapper	28
3.5.22. wrapper2: extension of wrapper	28

ÍNDICE DE CUADROS

3.1. Pairs of source/header files and their purpose	14
3.2. Filter structure definition	14
3.3. Options structure definition	16
3.4. LogInfo structure definition	17
3.5. ImpactDetectParams structure definition	19
3.6. FileCapture structure definition	21
3.7. Image structure definition	22
3.8. DtcImageVals structure definition	22
3.9. Point structure definition	24
3.10. List structure definition	24
3.11. Item structure definition	24
3.12. dtcImpact structure definition	25
3.13. SerHeader structure definition	26
3.14. SerCapture structure definition	27
3.15. DtcCapture structure definition	28

ÍNDICE DE FIGURAS

ÍNDICE DE FIGURAS

INTRODUCTION

This is the documentation for the current 3.1 version of the DeTeCt software. This is the next big version in the 3.0 branch, started in late 2016 – early 2017 but discontinued due to participants not being able to further develop that branch.

This version tackles the problems and issues from the previous version, unstable and experimental, which didn't yield the results the stable 2.0 branch, did.

This document will serve as both the documentation and a guide for future developers and users to improve the software by the classic method of trial and error.

1.1. CHANGELOG

2018-01-12: Document creation, brief introduction.

2018-02-06: Initial file, similar to v3.0 documentation

2018-02-19: More information now that the software is to be released

2018-02-21: Simple explanation of most files done; needs corrections by Marc Delcroix to be thorough.

2018-06-13: First L^AT_EX version. Some corrections.

INSTALLATION

2.1. REQUIREMENTS

1. Windows XP or later
2. OpenCV (2.4.13 is used and distributed with the software)
3. Visual Studio versions compatible with vc12/2013 (required for OpenCV 2.x). If OpenCV 3.x is used, later versions of Visual Studio are recommended (2015 and 2017 at the time of the writing of this document).
 - a) Community Edition: 30 day trial with a full unlock by using a Microsoft Account
 - b) Atomineer Pro (optional): addon which has been used to create the documentation. Other alternatives include Doxygen.

2.2. INITIAL SETUP

1. Download OpenCV from the official site (<http://opencv.org>). The downloadable file is an executable which can be extracted. When extracted, a folder called "opencv" is created in the same directory.
2. The folder created in the previous step should be moved to C: (So the full path would be C:\opencv2) as this is the current configuration. This can be configured later depending on the developer's needs.
3. In cases where ffmpeg is not available (necessary to read some of the videos) it should be installed in C, too. Bear in mind that both x86 and x64 versions should be downloaded to ensure compilation for both architectures. The Dev version located here (<https://ffmpeg.zeranoe.com/builds/>), and moved to C:
4. Install Visual Studio Community edition, which is the tool used for development. The existing codebase was developed with the 2013 version (supported by OpenCV 2) so a version compatible with that one is recommended. Luckily, as 2013 is not available for free, the 2015 and 2017 Community Editions have backwards compatibility. Furthermore, to ensure all tools for GUI development are available out of the box two items need to be checked in the installation:
 - a) Visual C++
 - b) Microsoft Foundation Classes (MFC) for C++

INSTALLATION

2.3. START THE PROJECT

The easiest way to launch the project is opening the .sln solution file. An instance of Visual Studio will be launched. There, click the build button. In case it doesn't work open Tools >Options. There, go to Debugging >Symbols and in the Symbol file (.pdb) locations, check "Microsoft Symbol Server". After you have closed the software, in the second run the central panel should show a section called "Recent Project" where DeTeCt-MFC should be the only item on the list. Click it to open the project again.

2.4. FIRST RUN

You'll see that in the main control bars, there are different controls and two specifically where the words "Release" and "x86" are present. These controls refer to the configuration of the program compilation, the former being the mode (debug being for development and release being for production), the latter being the version (x86 for 32-bit architecture and x64 for 64-bit architecture).

Click in the play button where "Build" appears and if everything goes right the compilation should be done correctly and the main windows should appear. Check the next chapter for more details.

INTERNALS

3.1. MAJOR DIFFERENCES FROM PREVIOUS VERSIONS

Detect v3.1 is the improvement of the previous faulty v3.0 version which is in turn the evolution of the v2.0 branch; main changes being:

1. . Written mostly in C++, instead of being fully implemented in C as previous versions were.
2. Uses (for the most part) the C++ OpenCV API rather than the C API.
3. Graphical application instead of a console application, with an easier (almost non-existent) learning curve.

Some old code has been reused due to time constraints and respecting the KISS and DRY principles in order to not rewrite the whole application in C++ for the same reason. C++ compilers such as the one used in this project are able to interoperate with C code with external linkage.

3.2. INTERFACE: MFC

The GUI framework used in DeTeCt is MFC, short for Microsoft Foundation Classes. MFC allows to create native Windows applications. Although a bit outdated, it supports major Windows versions starting from XP.

Currently, the GUI consists of four windows:

3.2.1. *Main window*

The main window, which in itself consists of a simple interface that for the time being isn't scalable. It can be divided into three parts.

Part 1: Top (menu bar)

Three menus can be seen here:

1. The "File" item opens another menu which allows three controls:
 - a) Open file: opens a window which allows the user to select a single file out of the allowed ones.
 - b) Open folder: opens a window which allows the user to select a folder which will be recursively traversed for allowed files. Depending on the depth of the directory tree the file list can be bigger than expected.

INTERNALS

- c) Exit: exits the program
- 2. The “Preferences” item opens another menu which allows a control:
 - a) Advanced settings: opens a settings window, which will be detailed in later sections.
- 3. The “help” item opens a menu which allows a control:
 - a) About: opens the about window, with information about the software and authors.

Part 2: Middle (log)

The big white rectangle covering most of the window is the part where the events will be logged. It’s scrollable so the user can read the events even when the algorithm is running (since it’s threaded).

Part 3: Down (progress bar and button)

As can be seen, the progress bar is just below the log. This bar will update for each frame analysed, the step being calculated by MFC itself by setting the interval between 1 and the number of frames of the video to be analysed.

The button below just launches the algorithm (which will be detailed in another section) for each item in the list of files).

3.2.2. Preferences window

This window shows the settings the current algorithm uses, with the default values. Most of the parameters are carried from the older version. Some of them have been discarded (but maintained for legacy reasons) and others have been added. They are divided as such:

Impact parameters

Parameters regarding the actual impact: the mean value factor, minimum time in seconds (which will be translated into frames taking the fps into account), the radius in pixels and the brightness threshold.

There are other parameters regarding the mask, etcetera.

ROI parameters

Parameters regarding the region of interest which will ideally contain Jupiter. The size and security factor (so it isn’t big/small) and the median buffer size to compute it. The main difference with previous versions is that the ROI is only computed in the first frame and then detected by correlation in subsequent frames.

Visualisation parameters

Parameters regarding the visualisation of the algorithm in different forms:

1. ROI shows a cropped image of the frame limited by the ROI coordinates
2. Tracking shows the original frame with the ROI and Centre of Mass (or Centre of Brightness)
3. Difference shows the difference frame
4. Reference shows the reference frame
5. Mask shows the mask used to compute the difference frame
6. Threshold shows the thresholded difference frame
7. Smooth shows an smoothed difference frame
8. Histogram shows the histogram of the image
9. Result shows the result of the algorithm (same as difference image)

Other parameters

Parameters that can't be categorised, so to speak. The most important ones are the number of frames for the reference, which will become a running average one we construct it; the minimum number of frames for the video; the debayering code for videos which use a median filter and the filter to smooth the difference image.

3.2.3. *About window*

This window is straightforward and doesn't have any input controls. General information about the software is given: version, authors and a small disclaimer of Europlanet funds as per the requirement of the project.

There are also two links: one to the original project page run by Mark Delcroix and the other one the UPV/EHU software page run by Ricardo Hueso.

3.2.4. *Notification window*

This window is also straightforward and will appear once the algorithm has run. It's shown as a modal so the user has to close it by hand.

3.3. THE NEW IMPACT DETECTION ALGORITHM

3.3.1. Definition

The algorithm is a variation of the original with a few modifications and additions by the UPV/EHU team (Jon Juaristi and Ricardo Hueso). The list below explains how an execution goes for a video:

1. Read video file. SER videos are automatically converted into the original colours, ultimately BGR or RGB.
2. For the video, get the framerate in frames/s and the number of frames. Some videos in wmv format might produce negative or wrong framerates.
3. If $n \leq \min_{frames}$ ignore the file
4. Else:

- a) Read frame
- b) Convert frame to grayscale, taking into account the colour of the original video (RGB, RBG, Bayer filters). For SER files the Bayer conversion is done automatically, manually (an option in the interface) for the rest.
- c) If $n_{frame} = 1$
 - 1) Get the CM (centre of mass, but actually the centre of brightness of the image) and the ROI.
 - 2) Smooth the frame with a filter stipulated by the user
 - 3) Convert the frame to the reference frame
 - 4) Add frame to reference frame queue (size is an option inputted by the user, 50 by default).
- d) If $n_{frame} > 1$
 - 1) We apply a cross correlation algorithm (as the frames could be unaligned) with the original ROI and a provisional ROI slightly bigger than the original; thus obtaining a new ROI with the original dimensions.
 - 2) Normalise the frame with

$$Grey_n = Grey_n \cdot \frac{\overline{Grey_1}}{\overline{Grey_n}}$$

- 3) Create the difference frame by applying differential photometry:

$$Dif_n = Grey_n - Ref_{n-1}$$

- 4) Apply thresholding to possible negative values.
- 5) Smooth with a filter selected by the user (standard, median, Gaussian or none).

3.3 THE NEW IMPACT DETECTION ALGORITHM

- 6) Save the maximum brightness point in with the frame number (minus the faulty frames), x and y coordinates and the value.
- 7) Add this to a list of maximum brightness-es of differential photometry frames of the video.
- 8) Check the size of the reference frame queue in regards to the number of frames needed for the reference frame a' If its lower, update the new reference frame:

$$Ref_n = \frac{n-1}{n} \cdot Ref_{n-1} + \frac{1}{n} \cdot Grey_n$$

- b' If it isn't, pop the first item and update the new reference frame, being the popped item:

$$Ref_n = Ref_{n-1} + \frac{Grey_n}{N} - \frac{Q_0}{n}$$

- 9) Add grayscale frame to the reference frame queue.
- e) Show the desired output frames: difference, ROI, tracking, etc.
- f) Once the file has been fully examined we create the detection images:
 - 1) Mean frame: each pixel is the mean brightness value of the coregistered frames.
 - 2) Max-mean frame (original and brightness-scaled): each pixel is the difference between the maximum brightness and mean brightness for each position in the coregistered sequence of frames.
 - 3) If the detail option has been selected other result frames will be created.

5. 5. After the video has been analysed, the impact algorithm is run:

- a) Check that $B_{max} > \bar{B} \cdot meanValueFactor$
- b) Create an empty queue whose size will between 5 and
- c) For each item on the list;
 - 1) When the queue is full (equal to or bigger than the size)
 - a' Get the mean brightness value
 - b' If $\bar{Q}_B < \bar{B} \cdot (1 + meanValueFactor)$, remove the first item and add item to queue and skip to the next step of the loop
 - c' Else, get the brightest element (first item when sorting by brightness), which will be assumed as the point of impact.
 - d' Check the number of frames in a radius close to the brightest item:

$$r = (x - x_0)^2 + (y - y_0)^2$$

INTERNALS

- e'* If more than 70 % of the frames are in the radius of the impact, it is considered as an impact candidate.
 - f'* Reorder the list in the original form (ascending order of frame number).
 - g'* If the mean is bigger than the previous mean (meaning it's the brightest impact), and impact detection object is created, with the first, last and maximum brightness frames; another item being added to detect if it could be a longer impact.
 - h'* If the mean is lower, the first item is removed and an item is added to the back of the queue, skipping to the next loop.
- 2) When the queue isn't full, the item is pushed to the back of the queue
- d)* If we have an impact, we check if the length is really the stipulated one, just in case; then show its duration (in the debug console, for the time being).
6. The interface will show if an impact has been detected or not: in a positive case, the information (start, finish, maximum brightness frames) will be shown, whereas in a negative case the user will be promptly notified about the lack of impact.
 7. Either way, the detection image will be shown for some seconds (four for the time being).
 8. If the image being processed was the last one of the bunch, a window will be shown where the user is notified of the location of the log which should be sent to Marc Delcroix.

3.3.2. Results

The results depend on the used parameters. As of now, the default parameters are used, with a mean value factor of 0,4.

A small functionality which creates .csv files has been added. This file contains one line per frame with its number, and the information about the maximum of brightness (x and y coordinates, the value of brightness). The files are used to test the algorithm and check if the parameters are the correct ones.

Comparing it to the log files, there are a couple of videos where the impact is really evident.

Those are the videos where there isn't much noise / clouds and the impact is clear; whereas there are others where the impacts are not detected:

- Impact is discernible and detected: Wesley, Go, Aoki, Ichimaru, McKeon, Kernbauer, Thomas, Pedranghelu
- Impact is neither discernible nor detected: Fleckstein

3.4 NEW FUNCTIONALITIES AND HOW TO EDIT THEM

- Impact detection is a bit faulty: Masayuki

We've added a small part to the log which shows the certainty of the impact.

First we get a brightness factor BF operating with the maximum brightness values (B) of the differential images used for the impact detection

$$BF = \frac{B_{max}}{\bar{B}} - 1$$

Then we obtain the certainty with this formula:

$$Certainty = \frac{BF}{MIS} \cdot \log_{10} \left(\frac{N_{real}}{N_{minimum}} \cdot 10 \right)$$

Where MIS is the minimum impact strength explained before, B is the mean of brightness and N is the number of frames of the impact, real being the actual impact and minimum being the parameter established in the option window.

The number will depend in two factors: the brightness factor and the length of the impact itself.

3.4. NEW FUNCTIONALITIES AND HOW TO EDIT THEM

3.4.1. GUI

The main file (called a *resource file* in MFC context) is named DeTeCt-MFC.rc. An area called "*Resource view*" will be seen in the right side of the window when the project is opened.

A list of items will be shown with a non-trivial name; which in turn the ID of the control.

Double clicking in one of the items will open an interface editing view. These are the most important ones:

1. Dialog

- a) IDD_ABOUTBOX: ID for the about dialog/window
- b) IDD_DETECTMFCDIALOG: ID for the about dialog/window
- c) IDD_PREFERENCES: ID for the settings dialog/window
- d) IDD_SENDDIALOG: ID for the result dialog/window

2. Menu

- a) IDR_MENU1: ID for the main menu bar

All dialogs have their own classes which have been defined in DeTeCt-MFC.h and DeTeCt-MFC.cpp header and source file respectively.

The correspondence between the Ids and the classes is the next one:

1. IDD_ABOUTBOX: CAboutDlg

INTERNALS

2. `IDD_DETECTMFC_DIALOG`: `CDeTeCtMFCDlg`
3. `IDD_PREFERENCES`: `PrefDlg`
4. `IDD_SENDDIALOG`: `SendEmailDlg`

To add items to the code, after right-clicking the “*Add member*” variable item of the menu has to be used. It will be added to the respective class, being declared in the header file. Double clicking will create events for the item in question.

In both cases a dialog will be opened, which is easy to navigate, in order to add the item and/or the event.

Checking the MFC documentation is suggested.

3.4.2. *Algorithm*

Most of the functionality has been converted into the C++ API. Thus, the functions are nearly the same with a few additions. Instead of the `IplImage` object, `Mat` objects are being used. This has the drawback since no OpenCV 2 structure has a ROI field, so a struct `Image` has been developed which in turn is made of the frame and a ROI.

The `filefmt2.h/cpp` files contain nearly the same code as `filefmt.h/c`. The main differences are in the functions that end in 2, which are the OpenCV 2.x versions of the old functions.

Changes

SER file management (`serfmt`):

1. `Serfmt2.cpp/h` were created but are currently unused
2. When creating the structure, depending on the header values enough memory is allocated to create the `Mat` from the file data.
3. A function named `serFrameRead` was created which reads the file data into the allocated memory. Depending on the size and channels it's read differently, taking into account how it's organised in the first place. For more information check the spec page:
<http://www.grischa-hahn.homepage.t-online.de/astro/ser/SER>
4. The data read by the aforementioned function can be retrieved in pointer format (`void*`) by another function named `serQueryFrameData`.

Wrapper files (`wrapper`)

1. A `dttCapture2` item has been created, where the videocapture has been migrated from the deprecated `CvCapture` class to the current `cv::VideoCapture` class. At the time being, it is unused.
2. The function `dttCaptureFromFile2` has been created as a new implementation using the new API.

3.4 NEW FUNCTIONALITIES AND HOW TO EDIT THEM

3. `dtcQueryFrame2` has been adapted to the new SER v3 specification. Raw data from the file is read normally, but depending on the header values the OpenCV Mat containing the video frame is created differently. Depending on the byte depth the Mat contains is one or three channels of 16-bit or 8-bit data. If the video has been captured using Bayer filters, a colour conversion will be applied.

Image manipulation files (`img2`):

1. Some of the functions which operate with the ROI, as stated above, instead of single `IplImage` instance we have to use an Image structure which consists in a Mat and Rect instances; differences being trivial.
2. There's a cross-correlation function, `correlateROI`, in which the current frame and the original frame, both being cut with their respective ROI (a bigger cut in the former case), by using OpenCV functions, allows us to get the cross-correlation matrix and the resulting aligned frame. It also returns an accurate value of the displacement of the Centre of Brightness
3. The histogram function is also very different due to the API changes being heavy in this regard.
4. There are a couple of functions which draw the location of the Centre of Mass and the location of the impact as a crosshair, as well as a rectangle delimiting the Region of Interest.

INTERNALS

3.5. THOROUGH EXPLANATION OF THE SOURCE CODE

3.5.1. Table of files and purpose

Filename	Purpose
<code>auxfunc.h</code>	Auxiliar functions
<code>cmdline</code>	Command line functions
<code>common</code>	Common functions
<code>datation</code>	Datation functions (legacy)
<code>datation2</code>	Datation functions (additions)
<code>DeTeCt-MFC</code>	Main program file
<code>DeTeCt-MFCDlg</code>	Dialog/window classes & functions
<code>DetectThread</code>	Impact detection Windows native thread
<code>dtc</code>	Impact detection algorithm for CLI (legacy)
<code>dtcas3</code>	Impact detection algorithm for AS!3
<code>dtcgui</code>	Impact detection algorithm for GUI
<code>filefmt</code>	File management functions
<code>fitsfmt</code>	Fits file management functions
<code>img</code>	OpenCV IplImage treatment functions (legacy)
<code>img2</code>	OpenCV Mat treatment functions (adaptations)
<code>max</code>	Functions for impact detection by max. brightness
<code>resource</code>	Visual Studio autogenerated file
<code>serfmt</code>	SER file management functions
<code>stdafx</code>	Visual Studio autogenerated file wrapper.

Cuadro 3.1: Pairs of source/header files and their purpose

3.5.2. `auxfunc.h`: auxiliary functions

This file only contains only a function:

- `DBOUT(S)`: returns the introduced string stream as an output string stream (`std::wostringstream`). Used mainly for debugging, the name stands for DeBug OUT.

For filter, these are the parameters:

Field	Type	Purpose
<code>type</code>	<code>integer</code>	Type of filter: blur, median, Gaussian. functions
<code>param</code>	<code>integer[4]</code>	Parameters: radius, height, width and sigma

Cuadro 3.2: Filter structure definition

And for options:

3.5 THOROUGH EXPLANATION OF THE SOURCE CODE

Field	Type	Purpose	
filename	char*	Input filename	
ofilename		Output filename	
darkfilename		Dark frame filename	
ovfname		Output video filename	
sfname		<unknown>filename	
dirname		directory name	
nSaveframe		<unknown>filename	
ostype	integer	Source video type	
ovtype		Output video type	
timeImpact	double	Time of impact in seconds	
incrLumImpact		Minimum impact strength	
incrFrameImpact	integer	Number of frames for impact	
radius	double	Radius of impact in pixels	
nFramesROI	unsigned long	Number of frames for ROI (<i>deprecated</i>)	
nFramesRef		Number of frames for reference frame	
wROI		ROI width (<i>deprecated</i>)	
hROI		ROI height (<i>deprecated</i>)	
bayer	integer	Debayering code	
medSize	double	Median filter buffer size for ROI calculation	
facSize		Size factor for ROI	
secSize		Security factor for ROI	
threshold		Time of impact in seconds	
learningRate		Learning rate (<i>deprecated</i>)	
histScale		Histogram scale	
wait	boolean	Time to wait in milliseconds	
thrWithMask		Use mask for reference frame	
viewROI		View ROI frame	
viewTrk		View tracking frame	
viewDif		View original differential frame	
viewRef		View reference frame	
viewMsk		View masked differential frame	
viewThr		View thresholded differential frame	
viewSmo		View smoothed differential frame	
viewRes		View resulting differential frame	
verbose		Verbose output (<i>deprecated</i>)	
debug		Debug output (<i>deprecated</i>)	
videotest		Test input video file (<i>deprecated</i>)	
ADUdtconly		Use Adudtc algorithm only (max, max-mean images)	
detail		Show more frames in output	
allframes		Save all intermediate frames	
minframes		Minimum frames to process video	
filter		Filter	Filter defined in the table above

Continues in next page

Continues from previous page

Field	Type	Purpose
dateonly	integer	Display date information and stop processing
ignore	integer	Ignore incorrect frames

Cuadro 3.3: Options structure definition

3.5.3. *Common: helper functions*

These files define multiple common or helping functions:

- Mid: copy a substring.
- Left: copy a left substring???
- Right: copy a substring to the right???
- Replace_str: replace string
- inStr: Check if substring is inside string
- strrstr: check location of string in substring
- InRstr:???
- Lcase: convert string to lowercase
- Ucase: convert string to uppercase

3.5.4. *Datation: datation functions*

These files define functions regarding image datation, i.e., obtaining when the video was recorded. Operates with Julian dates, UTC dates, etc. For this, we use the following functions:

- dtcGetDatation: Gets datation from file capture
- dtcGetCorrectDatation: Fixes the datation if computed incorrectly
- dtcWriteLog: writes DeTeCt.log, the contents of the file are explained in a later appendix.
- dtcGetDatationFromFileInfo: Gets datation from file system information
- dtcGetDatationFromLogFile: Gets datation from different software log files
- gregorian_calendar_to_jd: Converts dates to Julian day

3.5 THOROUGH EXPLANATION OF THE SOURCE CODE

- `jd_to_date`: Converts Julian days into dates
- `fprint_jd`: Prints Julian day
- `fprint_jd_wj`: Prints Julian day and calendar date
- `fprint_timetype`: Prints type of date, local, universal or unknown
- `JD_from_time_t`: Obtains Julian day from C++ `time_t` instance
- `month_nb`: Get month from String

3.5.5. *datation2*: extension of *datation*

This is an extension of the *datation* functions. Since in this version we support analysing multiple files at once, writing an unique log containing a line for each video analysed.

We define an struct here, named `LogInfo`:

Field	Type	Purpose
<code>filename</code>	<code>char*</code>	Name of analysed file
<code>start_time</code>	double	Start time of the video (calendar date)
<code>end_date</code>		End time of the video (calendar date)
<code>duration</code>		Duration of video in seconds
<code>fps</code>		Frames per second
<code>timetype</code>	<code>TIME_TYPE</code>	Type of time: UT, LT or unknown
<code>comment</code>	<code>char*</code>	Comment: software info & functions
<code>nb_impact</code>	integer	Number of impacts found
<code>certainty</code>	double	Impact confidence

Cuadro 3.4: `LogInfo` structure definition

With this, we can now write a log file with the next functions:

- `createLogInfo`: creates an instance of `LogInfo`
- `fprint_jd_wj`: prints the julian date converted into a calendar date into the file stream i.e., the log file.
- `dtcWriteWholeLog`: legacy function, unused
- `dtcWriteWholeLog2`: writes the information into the log file. It traverses a list of `LogInfo` items and prints them line by line. The contents will be explained in an appendix.
- `getDateTime`: helper function to print the date and time of the messages both in the output `.log` file and GUI

INTERNALS

3.5.6. *DeTeCt-MFC: main files*

The main file of the software, this is the starting point from where runs and operates as the user wants. The application is defined there, which is the instance of a class named `CDeTeCtMFCAApp`.

Due to an option to develop some functionalities compatible with AutoStakkert!3 there are two different forms of starting the program

1. The usual way: double clicking the icon and running the software in GUI mode normally.
2. AS!3 compatibility mode: opening a console and typing Should open a new terminal where the same algorithm is run (with similar output) but the image coregistration data is already known. This feature is not completed yet.

```
DeTeCt3.1.win32.exe -autostakkert filename
```

There aren't any functions except the `InitInstance` which works in the aforementioned ways. In the first case, components from the next file will be the ones spawned.

3.5.7. *DeTeCt-MFCDlg: window/dialog functionality*

As said before, this file pair is the one which has the whole window or dialog (Dlg stands for it) functionality, the equivalence is:

1. Main dialog: `CDeTeCtMFCDlg` (ID: `IDD_DETECTMFC_DIALOG`)
2. Preferences Dialog: `CDeTeCtMFCDlg` (ID: `IDD_ABOUTBOX`)
3. About dialog: `PrefDlg` (ID: `IDD_PREFERENCES`)
4. Results / log send dialog: `: SendEmailDlg` (ID: `IDD_SENDDIALOGDIALOG`)

3.5.8. *DetectThread: native detection thread*

This class is a bit redundant since the standard C++ libraries already allow for threading but was created in a time where the execution was synchronous and the interface couldn't be updated because of blocking calls.

Until recently a standard thread function was called but as Windows has its API in this regard those functions have been favoured.

The documentation requires the user to spawn a thread with `AfxBeginThread`, which in our case is defined as

```
AfxBeginThread(impactDetection, (LPVOID) params);
```

The function requires a special signature, which in our case ends up being the next one:

3.5 THOROUGH EXPLANATION OF THE SOURCE CODE

```
UINT __cdecl impactDetection(LPVOID pParam);
```

LPVOID is an alias for a pointer to void (void*). Since it's a pointer, we need to create a pointer to a structure, which is aptly named ImpactDetectParams, which is defined as

Field	Type	Purpose
file_list	vector<string>	List of files to be analysed
opts	Options	Options for the algorithm
scan_folder_path	string	Topmost folder where the output is going to be stored

Cuadro 3.5: ImpactDetectParams structure definition

Initially a ImpactDetectParams instance is created and casted to spawn the thread. In the thread function, the LPVOID will be reversed to obtain the original memory structure.

With this the thread will be spawned when the “Detect impact” button is clicked and the functions in dtcgui.h/cpp will be called.

3.5.9. *Dirent: third party native windows directory functions*

These functions are windows directory functions made by Toni Ronkko and available at

<https://github.com/tronkko/dirent/blob/master/include/dirent.h>

3.5.10. *dtc: legacy detection algorithm functions [unused]*

This file is used as the main function of the older, console based, version of DeTeCt, where the detection algorithm, the part where operations with IplImage type objects are made. The detection itself is run with functions present in max.

3.5.11. *dtcas3: Autostakkert!3 detection functions*

As stated before, AS!3 support is in a halt so these functions are not totally implemented yet. The function should be the same that it's called in DetectThread, but with some differences (namely the ROI identification which should be already included in the AS!3 file).

3.5.12. *dtcgui: Main detection functions*

This is, alongside the img2.h/.cpp files, the core of the software, so to speak. It's the adaptation of the dtc files from the previous version. The main functions of the file are:

INTERNALS

- `read_files`: with a folder, it obtains a vector of files in a directory tree reading it recursively. The supported formats are: `m4v`, `avi`, `ser`, `wmv`, `bmp`, `jpg`, `jpeg`, `jp2`, `dib`, `png`, `p?m`, `sr`, `ras`, `tif`, `tiff`, `fit` and `fits`. These files will be the ones which the algorithm will analyse.
- `Itemcmp`: compares maximum brightness items by their brightness value
- `Framecmp`: compares maximum brightness items by their frame number
- `Detect`: impact detection algorithm explained above.

It also includes a pair of helper functions which show which system is the user is running, for internal purposes, these are:

- `StreamDeTeCtOSversions`: shows software (both Windows and DeTeCt) version which the user is running as a text stream.
- `GetOSversion`: Gets Windows operative system version

3.5.13. *filefmt: legacy file management functions [unused]*

This file is the one which will operate with the files. This is the old version used by the 2.0 branch of the software with is present for legacy reasons and should be removed.

3.5.14. *filefmt2: file management functions*

This is the new version of the previous file, the main difference being that it the OpenCV C++ API is used instead of the C one.

In order to operate with the files, an struct named `FileCapture` has been created, with the next fields:

Field	Type	Purpose
<code>FileType</code>	<code>int</code>	Type of file
<code>fh</code>	<code>FILE*</code>	Associated file pointer
<code>frame</code>	<code>int</code>	Number of frame
<code>Image</code>	<code>Mat</code>	OpenCV matrix containing image pixel values
<code>Image</code>	<code>size_t</code>	Bytes per frame
<code>BytesPerPixel</code>		Bytes per pixel (1 or 2)
<code>ImageWidth</code>	<code>int</code>	Width of the image
<code>ImageHeight</code>		Height of he image (1 or 2)
<code>BytesPerPixel</code>		Pixel Depth in bits of the image (which will affect the <code>BytesPerPixel</code> value)

Continues in next page

3.5 THOROUGH EXPLANATION OF THE SOURCE CODE

Continues from previous page

Field	Type	Purpose
ColorID	unsigned int	Colour ID of the image (BRG, RGB, Bayer...)
header_size	size_t	Size of the header data
FrameCount		Number of frames
ValidFrameCount		Number of frames
StartTime_JD	size_t	Start time in Julian day format
StartTimeUTC_JD		Start UTC time in Julian day format
EndTime_JD		End time in Julian day format
EndtTimeUTC_JD		End UTC time in Julian day format
NumberPos	int	???
LeadingZeros		???
FirstFileIdx		???
LastValidFileIdx		???
LastFileIdx		???
Filename_rac		char*
Filename_head	???	
Filename_trail	???	
Filename_ext	???	
Filename_folder	???	

Cuadro 3.6: FileCapture structure definition

This structure will be operated by the next functions:

- FileCaptureFromFile: Opens the file and obtains the information, operating differently if the file is a .fits or a different format not supported by OpenCV.
- fileReinitCaptureRead: Restarts the file capture
- fileQueryFrame: queries a frame, i.e., reads the data into an IplImage.
- fileQueryFrame2: same as before but a Mat instead of IplImage.
- fileGet_info: gets information of the file.
- fileReleaseCapture: Cleans capture.
- fileGenerate_filename: Generates the filename
- fileGet_filename: Obtains the filename
- fileGenerate_number: Generates the file number based on the leading zeros of the original filename [??]

INTERNALS

3.5.15. *fitsfmt: FITS management functions*

This file uses the fits API by the Smithsonian institute, where this file extension gets its support with the next three functions:

- `fitsImageRead`: reads the bytes of the image.
- `fitsGet_info`: gets file info by reading the header data. For this, it uses the API mentioned before.
- `fitsJD_date`: obtains the Julian Date taking the header data into account.

3.5.16. *img: Opencv Matrix operation functions (C API)*

This file is the file where the deprecated `Ip1Image` objects are going to be operated. This file is deprecated and most functions have been adapted into the `Mat` class of the C++ which are defined in `img2`.

3.5.17. *img2: OpenCV detection functions (C++ API)*

As said before, this file will be the one which will operate the same way (with some addition) using the C++ API instead of the C one. For some instances where will need the ROI, an `Image` struct has been defined:

Field	Type	Purpose
Frame	Mat	Image frame
roi	Rect	Region of interest of said frame

Cuadro 3.7: Image structure definition

Another struct, unused at the moment, called `DtcImageVals` is available to find the luminosity values of the frame:

Field	Type	Purpose
lum	double	Total luminosity value
minlum		Minimum luminosity value
maxlum		Maximum luminosity value

Cuadro 3.8: `DtcImageVals` structure definition

The number of function in this case is pretty high, but most of them are pretty straightforward:

- `dtcGetROI`: returns the ROI of the frame
- `dtcGetCM`: obtains the centre of mass (centre of brightness) of the image.

3.5 THOROUGH EXPLANATION OF THE SOURCE CODE

- `dtcGetGrayCM`: obtains the CM for a grayscale image.
- `dtcGetGrayMatCM`: same as `dtcGetCM`.
- `dtcGetImageROIcCM`: Obtains the ROI by passing the CM.
- `dtcGetGrayImageROIcCM`: Obtains the ROI by passing the CM to a grayscale image
- `dtcGetGrayMat`: there are two versions of this function.
 - Obtains the grey mat but doesn't colour correct the videos which have been captured using Bayer filters [UNUSED]
 - Obtains the grey mat and colour corrects the videos which have been captured using Bayer filters.
- `dtcReduceMatToROI`: Reduces original frame to a rectangle delimited by the ROI.
- `dtcGetFileROIcCM`: Similar to `getImageROIcCM`, unused.
- `dtcGetFrameROIcCM`: Similar to `getImageROIcCM`, unused.
- `dtcMaxRect`: Obtains the bigger rectangle out of two
- `dtcDrawCM`: Draws the centre of mass and the ROI for the tracking frame, i.e., the original video frame.
- `dtcDrawImpact`: Draws the impact in the resulting detection image.
- `applyMaskToFrame`: Applies a mask to the grayscale frame reducing the background noise so the CM and ROI calculations are accurate.
- `correlateROI`: there are two versions of this function
 - Correlates the original ROI with the full frame [UNUSED]
 - Correlates the original ROI with a slightly bigger cut of the current ROI, increasing overall speed.
- `dtcRunningAvg`: computes a running average of two Images. Unused since the OpenCV `accumulateWeighted` function is favoured.
- `dtcWriteVideo`: writes a video with the desired output frames. Not used as of now.
- `dtcLumThreshold_ToZero2`: Applies a threshold to zero in the frame and saves it in the destination [UNUSED]
- `dtcGetImageLum`: Get image luminosity [UNUSED]
- `dtcGetGrayImageVals`: : Get grayscale image luminosity values [UNUSED]
- `dtcGetImageVals`: : Get image luminosity values [UNUSED]

INTERNALS

- `dtcGetHistogramImage`: Show the histogram of the frame in question.
- `dtcShowPhotometry`: Show the photometry data for a given frame.
- `correctBayerFilterImages`: Corrects original images which use bayer filters. Currently unused in favour of
- `dtcWriteFrame`: Writes a frame into a video writer.
- `doublecmp`: Compares two double values, returns true if $a > b$, false if $b < a$.
- `printtbuf`: Prints frame content into console.
- `isEqual`: checks if two frames are equal. This was used to check

3.5.18. *max*: impact detection helper structures and functions

In this file the (old) impact detection algorithm is declared and some structs are defined for that, which are the next ones:

`Point`, which defines the maximum point of brightness for the differential frame taking into account four parameters:

Field	Type	Purpose
<code>frame</code>	<code>long</code>	Number of frame
<code>val</code>	<code>double</code>	Maximum brightness value for given frame
<code>minlum</code>	<code>int</code>	x axis coordinate of <code>val</code>
<code>maxlum</code>		y axis coordinate of <code>val</code>

Cuadro 3.9: Point structure definition

A List

Field	Type	Purpose
<code>size</code>	<code>int</code>	Current list size
<code>maxsize</code>		Maximum possible list size
<code>head</code>	<code>item*</code>	First item
<code>tail</code>		Last item

Cuadro 3.10: List structure definition

Of doubly-linked Items:

Field	Type	Purpose
<code>point</code>	<code>Point*</code>	The point itself
<code>next</code>	<code>Item*</code>	Next item
<code>prev</code>		Previous item

Cuadro 3.11: Item structure definition

3.5 THOROUGH EXPLANATION OF THE SOURCE CODE

From which will we obtain a `dtcImpact` structure, where we will get the next data about the detected impact(s)

Field	Type	Purpose
MaxFrame	long	Frame with maximum brightness value
nMinFrame		Frame where the impact starts
nMaxFrame		Frame where the impact ends

Cuadro 3.12: `dtcImpact` structure definition

Most of the functions in this pair of files manipulate these data structures in one way or another, except the (now unused) impact detection:

- `init_list`: Initiates an empty list list of a given max size.
- `create_point`: creates a point given the parameters states above.
- `create_item`: creates an item with a given point
- `delete_head_item`: Deletes first item of a list
- `add_tail_item`: Adds a item to the end of the list
- `delete_list`: Deletes a list
- `detect_impact`: Detect an impact [unused]
- `get_item_array_mean_value`: Returns mean value of an array of items
- `get_item_list_mean_value`: Returns mean value of a list
- `init_dtc_struct`: Initialices `dtcImpact` structure
- `print_list_item`: Prints a list
- `print_item_array`: Prints an item array

3.5.19. *resource.h*: MFC resource file

This header file is automatically generated by MFC and it shouldn't be edited. It's used by the `.rc` resource file. Some of these definitions are defined in those files and then used in the `Detect-MFCDlg` files, but they aren't to be changed.

3.5.20. *serfmt*: SER file management

The purpose of this file pair lies in the management of the SER files supported by the newest astronomical capture recorders. It's a distinct file from the ones people are familiarised with and not common for the general users.

INTERNALS

Since its structure is so different, and to follow the specification located here <http://www.grischa-hahn.homepage.t-online.de/astro/ser/SER%20Doc%20V3B> two structures have been created, SerHeader and SerCapture.

For SerHeader, these are the fields:

Field	Type	Purpose
File_ID	char[14]	FileID
LuID	unsigned int	Lumenera camera series id
ColorID		Colour ID: RGB, BGR, MONO, etc.
LittleEndian		Bit order of the pixels, big or little endian
ImageWidth	size_t	Width of the image
ImageHeight		Height of the image
PixelDepth		Pixel depth of the image (in bits)
FrameCount		Number of frames
Observer	char[40]	Information about the observer
Instrument		Information about the instrument
Telescope		Information about the telescope
DateTime	unsigned char[8]	Date and time
DateTimeUTC		Date and UTC time

Cuadro 3.13: SerHeader structure definition

And for SerCapture:

Field	Type	Purpose
fh	FILE*	Associated file pointer
frame	size_t	Frame number
TimeStamp_Frame		Frame timestamp
Image	IplImage*	Bytes per frame
ImageBytes	size_t	Bytes for each image (frame)
BytesPerPixel		Pixel Depth in bits of the image (which will affect the BytesPerPixel value)
TimeStampExists	int	Check if there are timestamps
FrameCount	size_t	Total frame count
ValidFrameCount		Ser file header
StartTime_JD	size_t	Start time in Julian day format
StartTimeUTC_JD		Start UTC time in Julian day format
EndTime_JD		End time in Julian day format
EndtTimeUTC_JD		End UTC time in Julian day format
nChannels	int	Number of channels (1 or 3)
mat_type		Type of matrix: 8/16 bit 1/3 channel

Continues in next page

3.5 THOROUGH EXPLANATION OF THE SOURCE CODE

Continues from previous page

Field	Type	Purpose
byte_depth		Depth in bytes of the image (depends on BytesPerPixel)
current_frame		Frame read at the time
frame_data	void*	Raw frame data
big_endian	boolean	Processor endianness, big (true) or little (false)
data_proc_same_endianness		Frame data and the processor have the same endianness

Cuadro 3.14: SerCapture structure definition

Operated by the next functions:

- `serCaptureFromFile`: starts capture and obtains header data from the SER file.
- `serReinitCaptureRead`: Reinit the file capture (the program should exit if the frame count is too small to find an impact)
- `serReadTimeStamps`: reads the timestamps of the file
- `serQueryFrame`: returns the last read frame as an `IpImage` (deprecated)
- `serQueryTimeStamp`: queries timestamp of the current frame.
- `serReleaseCapture`: releases the data after the SER file has been fully read.
- `serImageRead`: reads a frame and returns the read bytes (deprecated since it uses the SER v2 spec)
- `serTimeStampRead`: reads the current timestamp
- `serDateTime_JD`: transforms the SER date format into Julian day
- `serPrintStr`: prints SER string for a given parameter
- `serPrintHeader`: Prints SER header data:
- `serFrameRead`: reads a frame following SER v3 spec and returns the read bytes.
- `serQueryFrameData`: returns the read frame as a `void*` pointer to the raw data

3.5.21. *wrapper: File capture wrapper*

This file serves as a wrapper for the different files that will be analysed. As stated before, there will be three kinds of files: FITS, SER and the files that can be read with OpenCV.

For that a `DtcCapture` struct will be defined, with the next parameters

Field	Type	Purpose		
type	char [14]	FileID		
framecount	unsigned int	Frame where the impact starts		
u	union	Union of the three available capture methods, in order to read the correct file:		
		Field	Type	Purpose
		capture	cvCapture*	OpenCV
		sercapture	SerCapture*	SER files
		filecapture	FileCapture*	FITS, etc

Cuadro 3.15: `DtcCapture` structure definition

To operate with this struct, the next functions have been defined:

- `dtcCaptureFromFile`: Initialises capture method, depending on the type of the file a different field of the union will be used.
- `dtcReinitCaptureRead`: after reading information for different purposes, the capture will start from the beginning.
- `dtcReleaseCapture`: release the capture object, again depending on the type of file to be analysed.
- `dtcQueryFrame`: Queries a frame of the video.
- `dtcGetCaptureProperty`: gets the property of the video, namely the FPS value.

3.5.22. *wrapper2: extension of wrapper*

This is a extension of the wrapper files, which using the aforementioned struct extends the functionality with a new function which operates with the new SER v3 spec functionality. It was intended to work with the C++ version of `CvCapture` named `VideoCapture`, which is still defined for future usage. It has two functions:

- `dtcCaptureFromFile2`: Same as `dtcCaptureFromFile` but intended to work with the `VideoCapture` class.
- `dtcQueryFrame2`: Same as `dtcQueryFrame` but it operates differently with SER files. The returned raw data must be normalised to a 8bit unsigned character matrix and then colour corrected (if

3.5 THOROUGH EXPLANATION OF THE SOURCE CODE

applicable) to return the original frame. These operations will depend on the header data of the SER file.

