FMDT Documentation

Release v1.0

FMDT team

Apr 17, 2023

USER MANUAL

CHAPTER

ONE

INTRODUCTION

1.1 Purpose

FMDT (Fast Meteor Detection Toolbox) is derived from a software which was **designed to detect meteors** on board ISS (International Space Station) or a CubeSat (A class of miniaturized satellite based around a form factor consisting of 10 cm (3.9 in) cubes.). FMDT is foreseen to be applied to airborne camera systems, e.g. in atmospheric balloons or aircraft. **It is robust to camera movements by a motion compensation algorithm.**

FMDT is ready for real-time processing on small boards like Raspberry Pi 4 or Nvidia Jetson Nano for embedded systems. For instance, on the Raspberry Pi 4 (@ 1.5 GHz), FMDT is able to compute **30 frames per second** on a HD (High Definition, 1920x1080 resolution) video sequence while the instant power is only **around 4 Watts**.

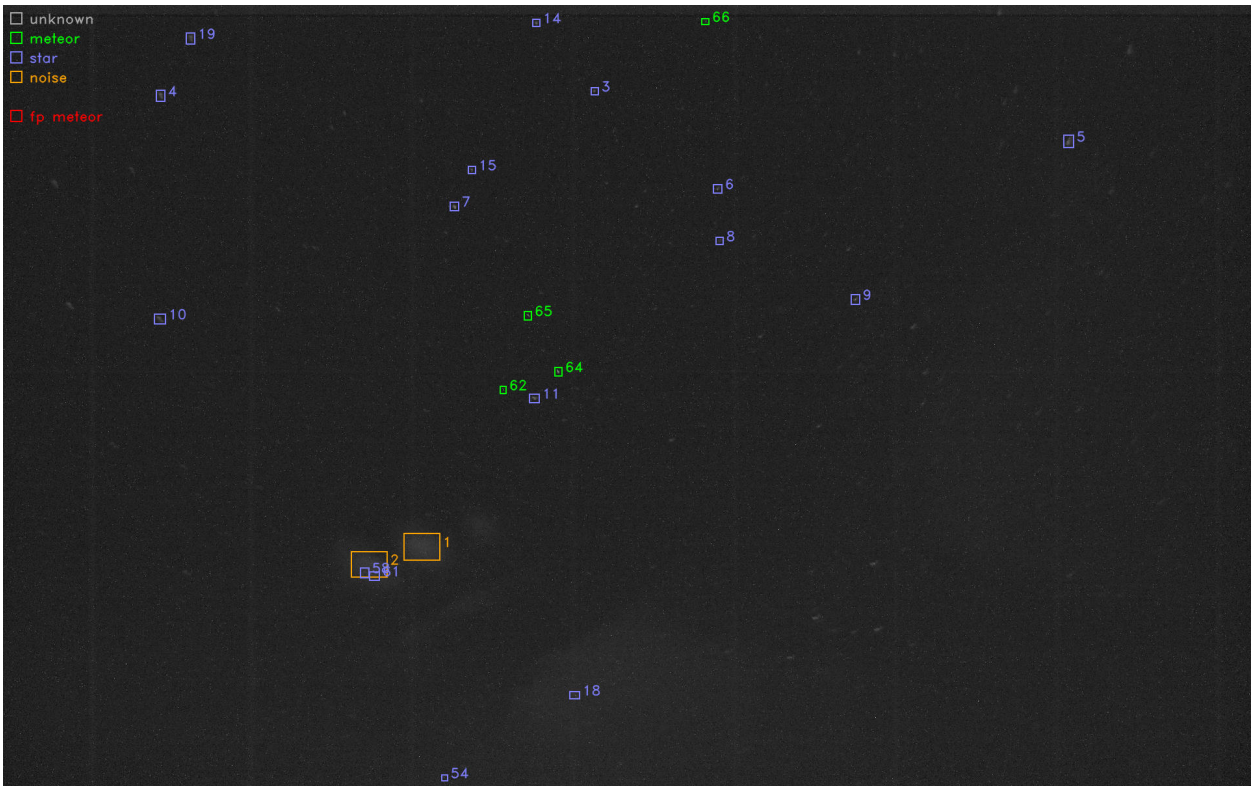

Fig. 1.1: Exemple of meteors detection and visualization.

Fig. 1.2 shows an example of detection on one frame. Green BBs (Bounding Boxes) represent detected *meteors*, purple BBs represent detected *stars* and orange BBs represent detected *noise* (= something which is not a *meteor* and not a

star).

1.2 Scientific Background

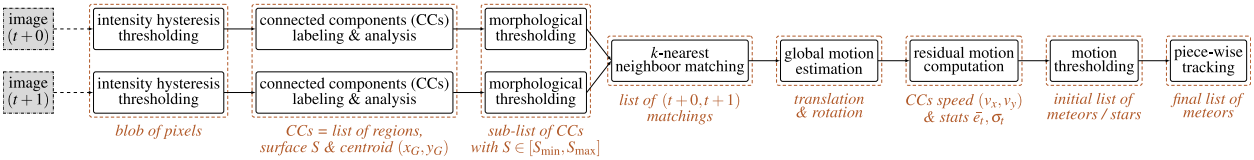

Fig. 1.2: The detection chain.

Fig. 1.2 presents the whole FMDT's detection chain. For each pair of images, **an intensity hysteresis threshold**, **a connected component labeling and an analysis algorithm** are applied to get a list of CCs (Connected-Components) with their bounding boxes and surface. Moreover, it also provides the first raw moments to compute the centroid $(x_G, y_G) = (S_x/S, S_y/S)$ of each blob of pixels. **A morphological threshold** is then done on the surface S to reject small and big CCs. **A** κ-NN (k-Nearest Neighbor) **matching** is applied to extract pairs of CCs from image I_{t+0} and I_{t+1} with t the image number in the video sequence. These matches are used to perform **a first global motion estimation** (rigid registration). Note that CCs are sometimes refered as RoIs (Regions of Interest) in this documentation.

This motion estimation is used to classify the CCs into two classes - still stars or moving meteors according to the following criterion: $|e_k - \bar{e_t}| > \sigma_t$ with e_k the compensation error of the CC (Connected-Component) number k, $\bar{e_t}$ the average error of compensation of all CCs of image I_t and σ_t the standard deviation of the error. **A second motion estimation** is done with only star CCs, to get a more accurate motion estimation and a more robust classification. Finally **a piece-wise tracking** is done by extending the $(t+0, t+1)$ matching with $(t+1, t+2)$ matching (and so on) to reduce the amount of false positive detection.

1.3 Scientific Results

IMCCE (Institut de Mécanique Céleste et de Calcul des Éphémérides, or Institute for Celestial Mechanics and Computation of Ephemerides in English) astronomers (from Paris's Observatory) led an airborne observation campaign of the 2022 τ-Herculids. The 2022 τ-Herculids mission is detailed here. The data collected by the mission have been processed with FMDT. The detection results helped the astronomers to see more meteors than their first "manual" detection (by human eyes). From 28 to 34 meteors thanks to FMDT automated detection. Detailed results are available in an article published in the *Astronomy & Astrophysics* journal [VLC+23].

Some preliminary results about the parallel implementation of the detection chain (see Fig. 1.2) have been presented in a poster [KCM+22] of the workshop AFF3CT (A Fast Forward Error Correction Toolbox). The poster shows results in terms of throughput (FPS (Frames Per Second)), latency and energy consumption. The selected hardware targets match embedded systems constraints (e.g. $\mathcal{T} \geq 30$ FPS and $\mathcal{P} \leq 10$ Watts).

1.4 References

CHAPTER

TWO

INSTALLATION GUIDE

2.1 Dependencies

This project uses `ffmpeg-io`, `nrc2`, `c-vector` and `aff3ct-core` projects as Git submodules, **you need to download them with the following command**:

```
git submodule update --init --recursive
```

Note: `ffmpeg-io` requires the `ffmpeg` executable: **you need to install ffmpeg on your system** if you want to be able to read video files. In addition, if you want to enable text indications in generated videos/images, the `OpenCV` library is required.

On Debian like systems you can easily install these packages with the `apt` package manager:

```
sudo apt install ffmpeg libopencv-dev
```

On macOS, we recommend you to use the `homebrew` package manager:

```
brew install ffmpeg opencv
```

2.2 Compilation with CMake

This project uses CMake in order to generate any type of projects (Makefile, Visual Studio, Eclipse, CLion, XCode, etc.). The code can easily be compiled with the following command lines:

```
mkdir build
cd build
cmake ..
make -j4
```

Note: The previous CMake command (`cmake ..`) will generate a Makefile without any compiler flag.

If you are using a GNU or Clang compiler like, **it is advised to use the following CMake command line** instead:

```
cmake .. -DFMDT_OPENCV_LINK=ON -DFMDT_AFF3CT_RUNTIME=ON -DCMAKE_BUILD_
↪TYPE=RelWithDebInfo -DCMAKE_CXX_FLAGS_RELWITHDEBINFO="-O3 -g" -DCMAKE_C_FLAGS_
↪RELWITHDEBINFO="-O3 -g" -DCMAKE_CXX_FLAGS="-Wall -funroll-loops -fstrict-aliasing -
↪march=native" -DCMAKE_C_FLAGS="-funroll-loops -fstrict-aliasing -march=native"
```

(continues on next page)

(continued from previous page)

Note: On Apple Silicon M1 CPUs and with Apple Clang, use `-mcpu=apple-m1` instead of `-march=native`.

The previous command line generates a Makefile in **release mode** (with debug information `-g`). It will produce optimized and ready for debug binaries. Moreover, OpenCV and AFF3CT libraries will be used during the compilation. It enables advanced features (see the following *CMake Options* section for more details about it).

2.2.1 CMake Options

Here is the list of the CMake available options:

- `FMDT_DETECT_EXE`

  Type BOOLEAN

  Default ON

  Example `cmake .. -DFMDT_DETECT_EXE=OFF`

  Compile the detection chain executable

- `FMDT_VISU_EXE`

  Type BOOLEAN

  Default ON

  Example `cmake .. -DFMDT_VISU_EXE=OFF`

  Compile the tracking visualization executable.

- `FMDT_CHECK_EXE`

  Type BOOLEAN

  Default ON

  Example `cmake .. -DFMDT_CHECK_EXE=OFF`

  Compile the check executable.

- `FMDT_MAXRED_EXE`

  Type BOOLEAN

  Default ON

  Example `cmake .. -DFMDT_MAXRED_EXE=OFF`

  Compile the max reduction executable.

- `FMDT_DEBUG`

  Type BOOLEAN

  Default OFF

  Example `cmake .. -DFMDT_DEBUG=ON`

  Build the project using debugging prints: these additional prints will be output on `stderr` and prefixed by `(DBG)`.

- `FMDT_OPENCV_LINK`

Type BOOLEAN

Default OFF

Example `cmake .. -DFMDT_OPENCV_LINK=ON`

Link with OpenCV library (required to enable some options for improved visualization in `fmdt-xxx` executables).

- `FMDT_AFF3CT_RUNTIME`

  Type BOOLEAN

  Default OFF

  Example `cmake .. -DFMDT_AFF3CT_RUNTIME=ON`

  Link with AFF3CT runtime and produce multi-threaded detection executable (`fmdt-detect-rt`).

CHAPTER

THREE

EXECUTABLES USAGE

This project generates the following **command line** executables:

- `fmdt-detect`,
- `fmdt-visu`,
- `fmdt-check`,
- `fmdt-maxred`.

`fmdt-detect` is an optimized and efficient C/C++ code for meteors detection. It produces only text outputs. The main results are the detected tracks and they can be read on the standard output (in the terminal). If the CMake `-DFMDT_AFF3CT_RUNTIME=ON` option is used to compile the project, then additional detection binaries are produced:

- `fmdt-detect-rt-seq`: this version comes with new performance measurement tools. However, this is a sequential version and the efficiency should be similar with the standard `fmdt-detect` executable,
- `fmdt-detect-rt-pip`: this version is multi-threaded. Thus, the throughput in term of FPS is much higher than the standard `fmdt-detect` executable (depending on the CPU target).

Both `fmdt-detect-rt-seq` and `fmdt-detect-rt-pip` have the same level of features than the standard `fmdt-detect` executable.

`fmdt-visu` mainly uses the `fmdt-detect` text outputs to generate highlighted video sequences. It can be combined with ground truth to distinguish good detected tracks (*true positive*) and bad detected tracks (*false positive*).

`fmdt-check` compares detected tracks (`fmdt-detect`) with a given ground truth. The results are shown on the standard output.

`fmdt-maxred` performs a max-reduction from a video sequence into an image. The produced image is in grayscale mode.

The next sections describe the command line parameters of these tools.

3.1 Detection Parameters

The meteors detection chain is located here: `./bin/fmdt-detect`.

The following table summarizes the available parameters:

Argument	Type	Details
--vid-in-path	STRING	See Section 3.1.2.
--vid-in-start	INTEGER	See Section 3.1.3.
--vid-in-stop	INTEGER	See Section 3.1.4.
--vid-in-skip	INTEGER	See Section 3.1.5.
--vid-in-buff	BOOLEAN	See Section 3.1.6.
--vid-in-loop	INTEGER	See Section 3.1.7.
--vid-in-threads	INTEGER	See Section 3.1.8.
--ccl-hyst-lo	INTEGER	See Section 3.1.9.
--ccl-hyst-hi	INTEGER	See Section 3.1.10.
--ccl-fra-path	STRING	See Section 3.1.11.
--ccl-fra-id	BOOLEAN	See Section 3.1.12.
--mrp-s-min	INTEGER	See Section 3.1.13.
--mrp-s-max	INTEGER	See Section 3.1.14.
--knn-k	INTEGER	See Section 3.1.15.
--knn-d	INTEGER	See Section 3.1.16.
--knn-s	FLOAT	See Section 3.1.17.
--trk-ext-d	INTEGER	See Section 3.1.18.
--trk-ext-o	INTEGER	See Section 3.1.19.
--trk-angle	FLOAT	See Section 3.1.20.
--trk-star-min	INTEGER	See Section 3.1.21.
--trk-meteor-min	INTEGER	See Section 3.1.22.
--trk-meteor-max	INTEGER	See Section 3.1.23.
--trk-ddev	FLOAT	See Section 3.1.24.
--trk-all	BOOLEAN	See Section 3.1.25.
--trk-bb-path	STRING	See Section 3.1.26.
--trk-mag-path	STRING	See Section 3.1.27.
--log-path	STRING	See Section 3.1.28.

3.1.1 Standard Output

`fmdt-detect` outputs a list of tracks. The tracks represent the detected objects in the video sequence. Here is the template of the output text:

```
# -------||---------------------------||---------------------------||---------
#  Track ||           Begin           ||            End            ||  Object
# -------||---------------------------||---------------------------||---------
# -------||---------|--------|--------||---------|--------|--------||---------
#     Id || Frame # |      x |      y || Frame # |      x |      y ||    Type
# -------||---------|--------|--------||---------|--------|--------||---------
   {tid} ||  {fbeg} | {xbeg} | {ybeg} ||  {fend} | {xend} | {yend} || {otype}
```

- `{tid}`: a positive integer (start from 1) value representing a unique track identifier,
- `{fbeg}`: a positive integer value representing the first frame in the video sequence when the track is detected,
- `{xbeg}`: a positive real value of the x-axis coordinate (beginning of the track),
- `{ybeg}`: a positive real value of the y-axis coordinate (beginning of the track),
- `{fend}`: a positive integer value representing the last frame in the video sequence when the track is detected,
- `{xend}`: a positive real value of the x-axis coordinate (end of the track),

- `{yend}`: a positive real value of the y-axis coordinate (end of the track),
- `{otype}`: a string of the object type, can be: `meteor`, `star` or `noise`.

3.1.2 --vid-in-path

Deprecated `--in-video`

Type STRING

Default [empty]

Example `--vid-in-path ~/Videos/meteors.mp4`

Input video path (supports also a path to a sequence of images `path/basename_%05d.jpg`).

3.1.3 --vid-in-start

Deprecated `--fra-start`

Type INTEGER

Default `0`

Example `--vid-in-start 12`

First frame id (included) to start the detection in the video sequence.

3.1.4 --vid-in-stop

Deprecated `--fra-end`

Type INTEGER

Default `0`

Example `--vid-in-stop 42`

Last frame id (included) to stop the detection in the video sequence. If set to `0`, read entire video.

3.1.5 --vid-in-skip

Deprecated `--fra-skip`

Type INTEGER

Default `0`

Example `--vid-in-skip 1`

Number of frames to skip.

3.1.6 --vid-in-buff

Deprecated `--video-buff`

Type BOOLEAN

Example `--vid-in-buff`

Bufferize all the video in global memory before executing the chain.

3.1.7 --vid-in-loop

Deprecated `--video-loop`

Type INTEGER

Default `1`

Example `--vid-in-loop 10`

Number of times the video is read in loop.

3.1.8 --vid-in-threads

Deprecated `--ffmpeg-threads`

Type INTEGER

Default `0`

Example `--vid-in-threads 1`

Select the number of threads to use to decode video input (in `ffmpeg`). If set to `0`, `ffmpeg` chooses the number of threads automatically.

3.1.9 --ccl-hyst-lo

Deprecated `--light-min`

Type INTEGER

Default `55`

Example `--ccl-hyst-lo 100`

Minimum light intensity for hysteresis threshold (grayscale $[0; 255]$).

3.1.10 --ccl-hyst-hi

Deprecated `--light-max`

Type INTEGER

Default `80`

Example `--ccl-hyst-hi 140`

Maximum light intensity for hysteresis threshold (grayscale $[0; 255]$).

3.1.11 --ccl-fra-path

Deprecated `--out-frames`

Type STRING

Default [empty]

Example `--ccl-fra-path ccl_fra/%05d.png`

Path of the files for CC debug (`path/cc_%05d.png`).

3.1.12 --ccl-fra-id

Deprecated `--show-id`

Type BOOLEAN

Example `--ccl-fra-id`

Show the RoI (Region of Interest)/CC ids on the output frames (to combine with `--ccl-fra-path` parameter). Requires to link with OpenCV library (`-DFMDT_OPENCV_LINK` CMake option, see Section 2.2.1).

3.1.13 --mrp-s-min

Deprecated `--surface-min`

Type INTEGER

Default `3`

Example `--mrp-s-min 5`

Minimum surface of the CCs in pixels.

3.1.14 --mrp-s-max

Deprecated `--surface-max`

Type INTEGER

Default `1000`

Example `--mrp-s-max 50`

Maximum surface of the CCs in pixels.

3.1.15 --knn-k

Deprecated `-k`

Type INTEGER

Default `3`

Example `--knn-k 5`

Maximum number of neighbors considered in the κ-NN algorithm.

3.1.16 --knn-d

Deprecated `--max-dist`

Type INTEGER

Default `10`

Example `--knn-d 25`

Maximum distance in pixels between two images (κ-NN algorithm).

3.1.17 --knn-s

Deprecated `--min-ratio-s`

Type FLOAT

Default `0.125`

Example `--knn-s 0.0`

Minimum surface ratio to match two CCs in κ-NN (`0` matches alls, `1` matches nothing). This parameter is also used for extrapolation in the tracking.

3.1.18 --trk-ext-d

Deprecated `--r-extrapol`

Type INTEGER

Default `10`

Example `--trk-ext-d 25`

Search radius in pixels for CC extrapolation (piece-wise tracking).

3.1.19 --trk-ext-o

Deprecated `--extrapol-orde`

Type INTEGER

Default `3`

Example `--trk-ext-o 1`

Maximum number of frames to extrapolate for lost objects (linear extrapolation).

3.1.20 --trk-angle

Deprecated `--angle-max`

Type FLOAT

Default `20.0`

Example `--trk-angle 35.0`

Tracking max angle between two meteors at $t-1$ and t (in degree).

3.1.21 --trk-star-min

Deprecated `--fra-star-min`

Type INTEGER

Default `15`

Example `--trk-star-min 5`

Minimum number of frames required to track a star.

3.1.22 --trk-meteor-min

Deprecated `--fra-meteor-min`

Type INTEGER

Default `3`

Example `--trk-meteor-min 5`

Minimum number of frames required to track a meteor.

3.1.23 --trk-meteor-max

Deprecated `--fra-meteor-max`

Type INTEGER

Default `100`

Example `--trk-meteor-max 50`

Maximum number of frames required to track a meteor.

3.1.24 --trk-ddev

Deprecated `--diff-dev`

Type FLOAT

Default `4.0`

Example `--trk-ddev 5.5`

Multiplication factor of the standard deviation (CC error has to be higher than $ddev \times stddev$ to be considered in movement).

3.1.25 --trk-all

Deprecated `--track-all`

Type BOOLEAN

Example `--trk-all`

By default the program only tracks `meteor` object type. If `--trk-all` is set, all object types are tracked (`meteor`, `star` or `noise`).

This parameter is used in the *_tracking_perform()* function.

3.1.26 --trk-bb-path

Deprecated `--out-bb`

Type STRING

Default [empty]

Example `--trk-bb-path bb.txt`

Path to the bounding boxes file required by `fmdt-visu` to draw detection rectangles. Each bounding box defines the area of an object, frame by frame.

Here is the corresponding line format:

```
{frame_id} {x_radius} {y_radius} {center_x} {center_y} {track_id} {is_extrapolated}
```

Each line corresponds to a frame and to an object, each value is separated by a space character.

3.1.27 --trk-mag-path

Deprecated `--out-mag`

Type STRING

Default [empty]

Example `--trk-mag-path mag.txt`

Path to the output file containing magnitudes of the tracked objects. Each line corresponds to a track/object and here is the corresponding line format:

```
{tid} {otype} {mag1} {mag2} {...} {magn}
```

`{mag1}` is the first magnitude value of the track/object of `{tid}` id. `{mag2}` is the second magnitude value (in the second frame where the object has been tracked). And so on, until the last magnitude value `{magn}`. Note that sometime the magnitude value can be `0`, it means that the object has been extrapolated on this frame, thus the magnitude cannot be computed.

3.1.28 --log-path

Deprecated `--out-stats`

Type STRING

Default [empty]

Example `--log-path detect_logs/`

Path of the output statistics, only required for debugging purpose.

Warning: This section targets advanced users, some knowledge about the implemented algorithms may be required!! You have been warned ;-).

`fmdt-detect` comes with the `--log-path` option to help to understand what is happening during the execution. This option enables to log internal statistics of the different algorithms used to detect meteors.

The folder contains multiple files, one per frame. For instance, the file name for the frame n°12 is: `00012.txt`. Each file contains 5 different tables:

- Table 1: list of RoIs at $t-1$ (result of the CCL (Connected-Components Labeling)/CCA (Connected-Components Analysis) + hysteresis algorithm at $t-1$),
- Table 2: list of RoIs at t (result of the CCL/CCA + hysteresis algorithm at t),
- Table 3: list of associations between $t-1$ RoIs and t RoIs (result of the κ-NN algorithm) + errors/velocities after motion estimation,
- Table 4: motion estimation statistics between $t-1$ and t frame,
- Table 5: list of tracks since the beginning of the execution (final output of the detection chain).

Note: The first log file (usally named `00000.txt`) only contains the table 2. This is normal because algorithms starting from κ-NN require two consecutive frames to work.

Table 1 and table 2: RoIs

```
# ------||----------------||--------------------------||--------------------------||---
↪----------------||-----------||------------
#   RoI ||      Track     ||        Bounding Box       ||   Surface (S in pixels)   ||  ␣
↪   Center       || Magnitude || Saturation
# ------||----------------||--------------------------||--------------------------||---
↪----------------||-----------||------------
# ------||------|---------||------|------|------|------||-----|----------|----------||---
↪------|---------||-----------||------------
#    ID ||   ID |    Type || xmin | xmax | ymin | ymax ||   S |       Sx |       Sy ||  ␣
↪    x |       y ||        -- ||    Counter
# ------||------|---------||------|------|------|------||-----|----------|----------||---
↪------|---------||-----------||------------
  {rid} || {tid}| {otype} ||{xmin}|{xmax}|{ymin}|{ymax}|| {S} |     {Sx} |     {Sy} ||  ␣
↪ {cx} |    {cy} ||     {mag} ||      {sat}
```

Each line corresponds to one RoI:

- `{rid}`: unique identifier for the current RoI (start from 1),

- {tid}: unique identifier of the corresponding track (start from 1), can be, empty if no track is associated to the current RoI,
- {otype}: type of the track object (meteor, noise or star), only if there is a track corresponding to this RoI,
- {xmin}: minimum x position of the bounding box,
- {xmax}: maximum x position of the bounding box,
- {ymin}: minimum y position of the bounding box,
- {ymax}: maximum y position of the bounding box,
- {S}: surface (area) of the RoI in pixels,
- {Sx}: sum of x properties,
- {Sy}: sum of y properties,
- {cx}: x center of mass,
- {cy}: y center of mass,
- {mag}: magnitude of the current RoI (accumulated brightness of the RoI, see the *_features_compute_magnitude()* function),
- {sat}: number of pixels that are saturated in the current RoI (a pixel p is saturated when its intensity $I_p = 255$, see the *_features_compute_magnitude()* function).

Table 3: List of associations between RoIs

```
# --------------------||--------------||-----------------------||-----------
#         RoI ID      ||    Distance   ||  Error (or velocity)   ||   Motion
# --------------------||--------------||-----------------------||-----------
# ----------|---------||--------|------||-------|-------|--------||-----------
#       t-1 |       t || pixels | rank ||    dx |    dy |      e || is moving
# ----------|---------||--------|------||-------|-------|--------||-----------
  {rid_t-1} | {rid_t} || {dist} |  {k} ||  {dx} |  {dy} |    {e} ||      {mov}
```

Each line corresponds to an association between one RoI at $t-1$ and at t:

- {rid_t-1}: id of the RoI in the table 1 (in the $t-1$ frame),
- {rid_t} : id of the RoI in the table 2 (in the t frame),
- {dist}: distance in pixels between the two RoIs,
- {rank}: rank in the κ-NN algorithm, if 1: it means that this is the closest RoI association, if 2: it means that this is the second closest RoI association, etc.,
- {dx}: x distance between the estimated position (after motion estimation) and the real position (in frame $t-1$),
- {dy}: y distance between the estimated position (after motion estimation) and the real position (in frame $t-1$),
- {e}: euclidean distance between the estimated position and the real position,
- {mov}: yes if the RoI is moving, no otherwise. The criteria to detect the motion of an RoI is: $|e - \bar{e_t^1}| > \sigma_t^1$, with e the error of the current RoI, e_t^1 the mean error after the first motion estimation and σ_t^1 the standard deviation after the first motion estimation.

If {mov} = yes then, {dx}, {dy} is the velocity vector and {e} is the velocity norm in pixel.

Note: `{dx}`, `{dy}`, `{e}` and `{mov}` are computed after the second motion estimation.

Table 4: Motion Estimation Statistics

```
# ------------------------------------------------------||-------------------------------
↪----------------------
#   First motion estimation (with all associated RoIs)  ||    Second motion estimation␣
↪(exclude moving RoIs)
# ------------------------------------------------------||-------------------------------
↪----------------------
# ----------|----------|----------|----------|----------||----------|----------|---------
↪-|----------|----------
#     theta |       tx |       ty | mean err |  std dev ||    theta |       tx |      ␣
↪ty | mean err |  std dev
# ----------|----------|----------|----------|----------||----------|----------|---------
↪-|----------|----------
   {theta1} |    {tx1} |    {ty1} |{mean_er1}|{std_dev1}|| {theta2} |    {tx2} |    {ty2}
↪ |{mean_er2}|{std_dev2}
```

There is only one line in this table. It represents the motion estimation between frame $t-1$ and frame t:

- `{theta}`: the estimated rotation angle between frame t and frame $t-1$,
- `{tx}` and `{ty}`: the estimated translation vector from frame t to frame $t-1$,
- `{mean_er}`: the mean error of the associated RoIs,
- `{std_dev}`: the standard deviation of the associated RoI errors.

The first estimation considers all the associated RoIs while the second estimation excludes the associated RoIs in movement. To be considered in movement, an RoI has to verify the following condition: $|e - \bar{e_t^1}| > \sigma_t^1$, with e the error of the current RoI, $\bar{e_t^1}$ the mean error after the first motion estimation and σ_t^1 the standard deviation after the first motion estimation.

Table 5: List of Tracks

```
# -------||--------------------------||--------------------------||---------||---------
↪----------
#  Track ||           Begin          ||            End           ||  Object || Reason␣
↪of changed
# -------||--------------------------||--------------------------||---------||   ␣
↪state (from
# -------||---------|--------|--------||---------|--------|--------||---------||  meteor␣
↪to noise
#     Id || Frame # |      x |      y || Frame # |      x |      y ||    Type ||   ␣
↪object only)
# -------||---------|--------|--------||---------|--------|--------||---------||---------
↪----------
   {tid} ||  {fbeg} | {xbeg} | {ybeg} ||  {fend} | {xend} | {yend} || {otype} ||        ␣
↪ {reason}
```

Most of the columns of this table have been described in the *Standard Output* section, here we focus only on extra columns:

- `{reason}`: reason of the classification from `meteor` to `noise`.

3.2 Visualization Parameters

The meteors visualization program is located here: `./bin/fmdt-visu`.

The following table summarizes the available parameters:

Argument	Type	Details
`--vid-in-path`	STRING	See Section 3.2.1.
`--vid-in-start`	INTEGER	See Section 3.2.2.
`--vid-in-stop`	INTEGER	See Section 3.2.3.
`--vid-in-threads`	INTEGER	See Section 3.2.4.
`--trk-path`	STRING	See Section 3.2.5.
`--trk-bb-path`	STRING	See Section 3.2.6.
`--trk-id`	BOOLEAN	See Section 3.2.7.
`--trk-nat-num`	BOOLEAN	See Section 3.2.8.
`--trk-only-meteor`	BOOLEAN	See Section 3.2.9.
`--gt-path`	STRING	See Section 3.2.10.
`--vid-out-path`	STRING	See Section 3.2.11.

3.2.1 `--vid-in-path`

Deprecated `--in-video`

Type STRING

Default [empty]

Example `--vid-in-path ~/Videos/meteors.mp4`

Input video path (supports also a path to a sequence of images `path/basename_%05d.jpg`).

3.2.2 `--vid-in-start`

Deprecated `--fra-start`

Type INTEGER

Default `0`

Example `--vid-in-start 12`

First frame id (included) to start the detection in the video sequence.

3.2.3 --vid-in-stop

Deprecated `--fra-end`

Type INTEGER

Default `0`

Example `--vid-in-stop 42`

Last frame id (included) to stop the detection in the video sequence. If set to `0`, read entire video.

3.2.4 --vid-in-threads

Deprecated `--ffmpeg-threads`

Type INTEGER

Default `0`

Example `--vid-in-threads 1`

Select the number of threads to use to decode video input (in `ffmpeg`). If set to `0`, `ffmpeg` chooses the number of threads automatically.

3.2.5 --trk-path

Deprecated `--in-tracks`

Type STRING

Default [empty]

Example `--trk-path tracks.txt`

The tracks file corresponding to the input video (generated from `fmdt-detect`). See Section 3.1.1 for the description of the expected text input format.

3.2.6 --trk-bb-path

Deprecated `--in-bb`

Type STRING

Default [empty]

Example `--trk-bb-path bb.txt`

The bounding boxes file corresponding to the input video (generated from `fmdt-detect`). See Section 3.1.26 for the description of the expected text output format.

3.2.7 --trk-id

Deprecated `--show-id`

Type BOOLEAN

Example `--trk-id`

Show the object ids on the output video and frames. Requires to link with OpenCV library (`-DFMDT_OPENCV_LINK` CMake option, see Section 2.2.1).

3.2.8 --trk-nat-num

Deprecated `--show-id`

Type BOOLEAN

Example `--trk-nat-num`

Natural numbering of the object ids, work only if `--trk-id` is set.

3.2.9 --trk-only-meteor

Deprecated `--only-meteor`

Type BOOLEAN

Example `--trk-only-meteor`

Show only meteors.

3.2.10 --gt-path

Deprecated `--in-gt`

Type STRING

Default [empty]

Example `--gt-path gt.txt`

File containing the ground truth. Ground truth file gives objects positions over time. Here is the expected text format of a line:

```
{otype} {fbeg} {xbeg} {ybeg} {fend} {xend} {yend}
```

`{otype}` can be `meteor`, `star` or `noise`. `{fbeg}` and `{fend}` stand for *frame begin* and *frame end*. `{xbeg}` and `{ybeg}` stand for x and y coordinates of the *frame begin*. `{xend}` and `{yend}` stand for x and y coordinates of the *frame end*. `{fbeg}`, `{xbeg}`, `{ybeg}`, `{fend}`, `{xend}`, `{yend}` are positive integers. Each line corresponds to an object and each value is separated by a space character.

3.2.11 `--vid-out-path`

Deprecated `--out-video`

Type STRING

Default [empty]

Example `--vid-out-path sky.mp4`

Path of the output video (supports also a path to a sequence of images `path/basename_%05d.jpg`) with meteor tracking colored rectangles (BBs). If `--gt-path` is set then the bounding rectangles are red if *false positive* meteor and green if *true positive* meteor.

3.3 Check Parameters

The meteors checking program is located here: `./bin/fmdt-check`.

The following table summarizes the available parameters:

Argument	Type	Details
`--trk-path`	STRING	See Section 3.3.2.
`--gt-path`	STRING	See Section 3.3.3.

3.3.1 Standard Output

The first part of `fmdt-check` *stdout* is a table where each entry corresponds to an object of the GT (Ground Truth):

```
# ---------------||---------------||-----------------||--------
#    GT Object   ||      Hits     ||    GT Frames    || Tracks
# ---------------||---------------||-----------------||--------
# -----|---------||--------|------||--------|--------||--------
#   Id |    Type || Detect |  GT  ||  Start |  Stop  ||      #
# -----|---------||--------|------||--------|--------||--------
 {tid} | {otype} ||   {dh} | {gh} || {staf} | {stof} ||   {nt}
```

- `{tid}`: a positive integer value representing a unique identifier of ground truth track/object,
- `{otype}`: a string of the object type, can be: `meteor`, `star` or `noise`,
- `{dh}`: a positive integer value of the number of frames when the object is detected (from the tracks, `--trk-path`),
- `{gh}`: a positive integer value of the number of frame when the object is present (from the ground truth, `--gt-path`),
- `{staf}`: a positive integer value of the frame start (from the ground truth, `--gt-path`),
- `{stof}`: a positive integer value of the frame stop (from the ground truth, `--gt-path`),
- `{nt}`: a positive integer value of the number of tracks that match the ground truth object.

In a second part, `fmdt-check` *stdout* gives some statistics in the following format (`{pi}` stands for *positive integer* and `{pf}` for *positive float*):

```
Statistics:
  - Number of GT objs = ['meteor': {pi}, 'star': {pi}, 'noise': {pi}, 'all': {pi}]
  - Number of tracks  = ['meteor': {pi}, 'star': {pi}, 'noise': {pi}, 'all': {pi}]
  - True positives    = ['meteor': {pi}, 'star': {pi}, 'noise': {pi}, 'all': {pi}]
  - False positives   = ['meteor': {pi}, 'star': {pi}, 'noise': {pi}, 'all': {pi}]
  - True negative     = ['meteor': {pi}, 'star': {pi}, 'noise': {pi}, 'all': {pi}]
  - False negative    = ['meteor': {pi}, 'star': {pi}, 'noise': {pi}, 'all': {pi}]
  - Tracking rate     = ['meteor': {pf}, 'star': {pf}, 'noise': {pf}, 'all': {pf}]
```

- `Number of GT objs`: the number of objects from the ground truth,
- `Number of tracks`: the number of objects from the tracks (`fmdt-detect` output),
- `True positives`: number of detected objects that are in the ground truth (with the same type),
- `False positives`: number of detected objects that are not in the ground truth (or that have a different type).
- `True negative`: number of detected objects that are different from the current type of object. For instance, if we focus on `meteor` object type, the number of false negatives is the sum of all the objects in the tracks that are `star` or `noise`,
- `False negative`: number of non-detected objects (present in the ground truth and not present in the tracks),
- `Tracking rate`: the sum of detected hits on the sum of the ground truth hits. Range is between `1` (perfect tracking) and `0` (nothing is tracked). When there are more hits in a track than in the ground truth, the detected hits are the ground truth hits minus the extra hits of the track.

For each line, the `meteor`, `star` and `noise` object types are considered. `all` stands for all types, sometime `all` can be mean-less.

3.3.2 `--trk-path`

Deprecated `--in-tracks`

Type STRING

Default [empty]

Example `--trk-path tracks.txt`

The tracks file corresponding to the input video (generated from `fmdt-detect`). See Section 3.1.1 for the description of the expected text input format.

3.3.3 `--gt-path`

Deprecated `--in-gt`

Type STRING

Default [empty]

Example `--gt-path gt.txt`

File containing the ground truth. See Section 3.2.10 for the description of the expected text input format.

3.4 Max-reduction Parameters

The max-reduction generation program is located here: `./bin/fmdt-maxred`.

The following table summarizes the available parameters:

Argument	Type	Details
`--vid-in-path`	STRING	See Section 3.4.1.
`--vid-in-start`	INTEGER	See Section 3.4.2.
`--vid-in-stop`	INTEGER	See Section 3.4.3.
`--vid-in-threads`	INTEGER	See Section 3.4.4.
`--trk-path`	STRING	See Section 3.4.5.
`--trk-id`	BOOLEAN	See Section 3.4.6.
`--trk-nat-num`	BOOLEAN	See Section 3.4.7.
`--trk-only-meteor`	BOOLEAN	See Section 3.4.8.
`--gt-path`	STRING	See Section 3.4.9.
`--fra-out-path`	STRING	See Section 3.4.10.

3.4.1 `--vid-in-path`

Deprecated `--in-video`

Type STRING

Default [empty]

Example `--vid-in-path ~/Videos/meteors.mp4`

Input video path (supports also a path to a sequence of images `path/basename_%05d.jpg`).

3.4.2 `--vid-in-start`

Deprecated `--fra-start`

Type INTEGER

Default `0`

Example `--vid-in-start 12`

First frame id (included) to start the detection in the video sequence.

3.4.3 `--vid-in-stop`

Deprecated `--fra-end`

Type INTEGER

Default `0`

Example `--vid-in-stop 42`

Last frame id (included) to stop the detection in the video sequence. If set to `0`, read entire video.

3.4.4 --vid-in-threads

Deprecated `--ffmpeg-threads`

Type INTEGER

Default `0`

Example `--vid-in-threads 1`

Select the number of threads to use to decode video input (in `ffmpeg`). If set to `0`, `ffmpeg` chooses the number of threads automatically.

3.4.5 --trk-path

Deprecated `--in-tracks`

Type STRING

Default [empty]

Example `--trk-path tracks.txt`

The tracks file corresponding to the input video (generated from `fmdt-detect`). See Section 3.1.1 for the description of the expected text input format.

3.4.6 --trk-id

Deprecated `--show-id`

Type BOOLEAN

Example `--trk-id`

Show the object ids on the output video and frames. Requires to link with OpenCV library (`-DFMDT_OPENCV_LINK` CMake option, see Section 2.2.1).

3.4.7 --trk-nat-num

Deprecated `--show-id`

Type BOOLEAN

Example `--trk-nat-num`

Natural numbering of the object ids, work only if `--trk-id` is set.

3.4.8 --trk-only-meteor

Deprecated `--only-meteor`

Type BOOLEAN

Example `--trk-only-meteor`

Show only meteors.

3.4.9 --gt-path

Deprecated `--in-gt`

Type STRING

Default [empty]

Example `--gt-path gt.txt`

File containing the ground truth. Ground truth file gives objects positions over time. Here is the expected text format of a line:

```
{otype} {fbeg} {xbeg} {ybeg} {fend} {xend} {yend}
```

The tracks file corresponding to the input video (generated from `fmdt-detect`). See Section 3.1.1 for the description of the expected text input format.

3.4.10 --fra-out-path

Deprecated `--out-frame`

Type STRING

Default [empty]

Example `--fra-out-path maxred.png`

Path of the output frame.

CHAPTER

FOUR

EXAMPLES OF USE

Download a video sequence containing meteors here. These video sequence comes from IMCCE (*Paris's Observatory*) and is the result of an airborne observation of the 2022 τ-Herculids. More information about the 2022 τ-Herculids is available here.

4.1 Meteors detection

```
./bin/fmdt-detect --vid-in-path ./2022_05_31_tauh_34_meteors.mp4
```

Write tracks and bounding boxes into text files for `fmdt-visu` and `fmdt-check`:

```
./bin/fmdt-detect --vid-in-path ./2022_05_31_tauh_34_meteors.mp4 --trk-bb-path ./out_
↪detect_bb.txt > ./out_detect_tracks.txt
```

4.2 Visualization

Visualization **WITHOUT** ground truth:

```
./bin/fmdt-visu --vid-in-path ./2022_05_31_tauh_34_meteors.mp4 --trk-path ./out_detect_
↪tracks.txt --trk-bb-path ./out_detect_bb.txt --vid-out-path out_visu.mp4
```

Visualization **WITH** ground truth:

```
./bin/fmdt-visu --vid-in-path ./2022_05_31_tauh_34_meteors.mp4 --trk-path ./out_detect_
↪tracks.txt --trk-bb-path ./out_detect_bb.txt --gt-path ../validation/2022_05_31_tauh_
↪34_meteors.txt --vid-out-path out_visu.mp4
```

4.3 Offline checking

Use `fmdt-check` with the following arguments:

```
./bin/fmdt-check --trk-path ./out_detect_tracks.txt --gt-path ../validation/2022_05_31_
↪tauh_34_meteors.txt
```

```
#  --------------------
# |         ----*      |
# | --* FMDT-CHECK --* |
# |   -------*         |
#  --------------------
#
# Parameters:
# -----------
#  * trk-path = ./out_detect_tracks.txt
#  * gt-path  = ../validation/2022_05_31_tauh_34_meteors.txt
#
# The program is running...
# ---------------||--------------||---------------||--------
#    GT Object   ||     Hits     ||   GT Frames    || Tracks
# ---------------||--------------||---------------||--------
# -----|---------||--------|-----||-------|-------||--------
#   Id |    Type || Detect |  GT || Start |  Stop ||      #
# -----|---------||--------|-----||-------|-------||--------
     1 |  meteor ||      7 |   7 ||   102 |   108 ||      1
     2 |  meteor ||     17 |  16 ||   110 |   125 ||      1
     3 |  meteor ||      8 |   9 ||   111 |   119 ||      1
     4 |  meteor ||      3 |   3 ||   121 |   123 ||      1
     5 |  meteor ||      3 |   3 ||   127 |   129 ||      1
     6 |  meteor ||      3 |   3 ||   129 |   131 ||      1
     7 |  meteor ||      9 |  10 ||   133 |   142 ||      1
     8 |  meteor ||     10 |  10 ||   134 |   143 ||      1
     9 |  meteor ||      4 |   4 ||   134 |   137 ||      1
    10 |  meteor ||      3 |   4 ||   135 |   138 ||      1
    11 |  meteor ||      6 |  10 ||   137 |   146 ||      1
    12 |  meteor ||      4 |   4 ||   139 |   142 ||      1
    13 |  meteor ||     11 |  11 ||   140 |   150 ||      1
    14 |  meteor ||      4 |   4 ||   146 |   149 ||      1
    15 |  meteor ||      3 |   3 ||   156 |   158 ||      1
    16 |  meteor ||     10 |  10 ||   156 |   165 ||      1
    17 |  meteor ||      6 |   6 ||   157 |   162 ||      1
    18 |  meteor ||      4 |   4 ||   160 |   163 ||      1
    19 |  meteor ||      4 |   4 ||   164 |   167 ||      1
    20 |  meteor ||      3 |   3 ||   167 |   169 ||      1
    21 |  meteor ||      5 |   5 ||   171 |   175 ||      1
    22 |  meteor ||      7 |   7 ||   174 |   180 ||      1
    23 |  meteor ||      8 |   8 ||   178 |   185 ||      1
    24 |  meteor ||     11 |  11 ||   179 |   189 ||      1
    25 |  meteor ||      3 |   3 ||   179 |   181 ||      1
    26 |  meteor ||      5 |   5 ||   180 |   184 ||      1
    27 |  meteor ||      7 |   7 ||   183 |   189 ||      1
    28 |  meteor ||      4 |   4 ||   194 |   197 ||      1
    29 |  meteor ||      3 |   4 ||   197 |   200 ||      1
    30 |  meteor ||      6 |   5 ||   199 |   203 ||      2
    31 |  meteor ||      6 |   6 ||   200 |   205 ||      1
    32 |  meteor ||      7 |   7 ||   223 |   229 ||      1
    33 |  meteor ||      5 |   5 ||   224 |   228 ||      1
    34 |  meteor ||      4 |   4 ||   249 |   252 ||      1
Statistics:
```

(continues on next page)

(continued from previous page)

```
  - Number of GT objs = ['meteor':   34, 'star':    0, 'noise':    0, 'all':   34]
  - Number of tracks  = ['meteor':   38, 'star':    0, 'noise':    0, 'all':   38]
  - True positives    = ['meteor':   35, 'star':    0, 'noise':    0, 'all':   35]
  - False positives   = ['meteor':    3, 'star':    0, 'noise':    0, 'all':    3]
  - True negative     = ['meteor':    0, 'star':   38, 'noise':   38, 'all':   76]
  - False negative    = ['meteor':    0, 'star':    0, 'noise':    0, 'all':    0]
  - tracking rate     = ['meteor': 0.95, 'star':  nan, 'noise':  nan, 'all': 0.95]
# End of the program, exiting.
```

4.4 Max-reduction

Use `fmdt-maxred` with the following arguments:

```
./bin/fmdt-maxred --vid-in-path ./2022_05_31_tauh_34_meteors.mp4 --fra-out-path out_
↪maxred.pgm
```

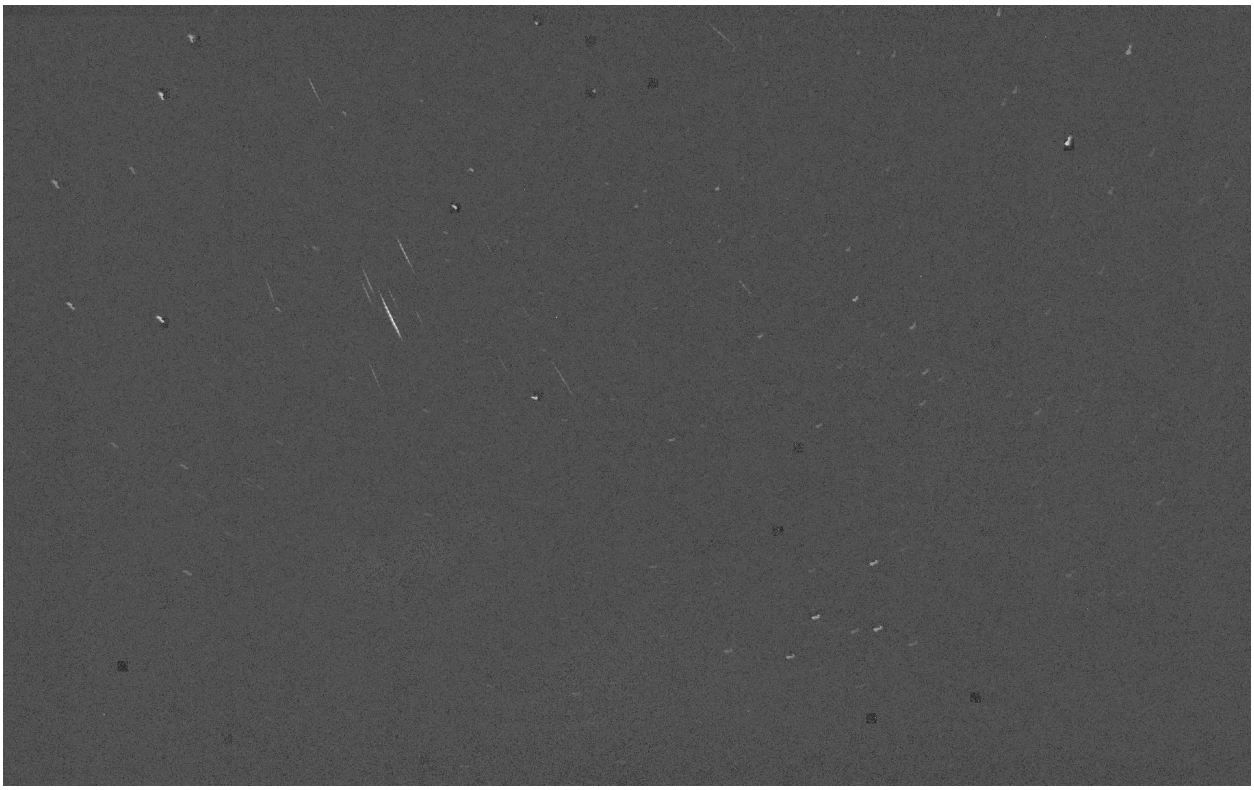

Fig. 4.1: Max-reduction image of the 2022 τ-Herculids video sequence.

CHAPTER

FIVE

PROJECT ARCHITECTURE

First of all, this is mainly a project written in C language. There are some exceptions with some part of the code written in C++ but the C++ code is not mandatory and the project can always compile with a C compiler.

Thus, this projects can be seen as a pool of C structures and C functions. The headers are located in the `./include/c/fmdt` folder (= structures, enumerations, defines and functions declarations). And the implementations of the functions are located in the `./src/common` folder.

5.1 Modules

Headers (`.h` files) and function implementations (`.c` files) are grouped into *modules*. A *module* is a set of headers and implementation files that are working on the same "topic". For instance, a κ-NN module has been implemented is the project. It is composed of the following files:

- `./include/c/fmdt/kNN.h`: this is a proxy header file that includes `kNN_struct.h`, `kNN_compute.h` and `kNN_io.h` headers,
- `./include/c/fmdt/kNN/kNN_struct.h`: contains structure definitions related to κ-NN,
- `./include/c/fmdt/kNN/kNN_compute.h`: declares the functions related to κ-NN computations,
- `./include/c/fmdt/kNN/kNN_io.h`: declares the functions related to κ-NN inputs and outputs, in the case of the κ-NN matching there are only functions to display the output results after the computations,
- `./src/common/kNN/kNN_compute.c`: implementations of the functions declared in the `kNN_compute.h` file, plus additional private functions,
- `./src/common/kNN/kNN_io.c`: implementations of the functions declared in the `kNN_io.h` file, plus additional private functions.

This decomposition in several files is made to have a good separation of concerns. This way developers can easily know what to find in each file.

5.2 Executables

The source code of the final executables is located in `./src/mains/` directory. Each file corresponds to a final executable and thus contains a `main` function.

5.3 Public Interfaces

Generally there are two levels to call a processing function. For instance, in the κ-NN module and in the `kNN_compute.h` header, the two following functions are defined:

```
void _kNN_match(float** data_distances, uint32_t** data_nearest, uint32_t* data_
↪conflicts, const uint32_t* RoIs0_id,
                const uint32_t* RoIs0_S, const float* RoIs0_x, const float* RoIs0_y,␣
↪uint32_t* RoIs0_next_id,
                const size_t n_RoIs0, const uint32_t* RoIs1_id, const uint32_t* RoIs1_S,␣
↪const float* RoIs1_x,
                const float* RoIs1_y, uint32_t* RoIs1_prev_id, const size_t n_RoIs1,␣
↪const int k,
                const uint32_t max_dist, const float min_ratio_S);
```

```
void kNN_match(kNN_data_t* kNN_data, const RoIs_basic_t* RoIs0_basic, const RoIs_basic_
↪t* RoIs1_basic,
               RoIs_asso_t* RoIs0_asso, RoIs_asso_t* RoIs1_asso, const int k, const␣
↪uint32_t max_dist,
               const float min_ratio_S);
```

Both functions compute the κ-NN matching. The function prefixed with an underscore (`_kNN_match`) requires only buffers of native types (`float` and `uint32_t` here) while the other function (`kNN_match`) requires structure types (`kNN_data_t`, `RoIs_basic_t` and `RoIs_asso_t`). In the implementation, the `kNN_match` function simply call the `_kNN_match` function.

Compute functions often use inner data. This data is NOT input or output data. This is data required to store intermediate results during the computation. They are different ways to manage this type of data in C codes. In FMDT the chosen pattern is to allocate this inner data before calling the compute function. And to deallocate this data after. For instance, in the previous `kNN_match` function, the first parameter is a pointer of `kNN_data_t` type. This data can be allocated with the `kNN_alloc_data` function defined in the same `kNN_compute.h` header.

The following lines illustrate how to properly use the κ-NN module:

```
// inner data allocation on the heap
kNN_data_t* kNN_data = kNN_alloc_data(MAX_SIZE);
// initialization of the data with zeros (this is NOT mandatory)
kNN_init_data(kNN_data);
// kNN matching computation (multiple calls of kNN match function with the same `kNN_
↪data`)
kNN_match(kNN_data, /* ... */);
kNN_match(kNN_data, /* ... */);
kNN_match(kNN_data, /* ... */);
kNN_match(kNN_data, /* ... */);
// inner data deallocation
kNN_free_data(kNN_data);
```

5.4 Dependencies

FMDT depends on multiple external libraries to work. The following section details each of these libraries.

5.4.1 ffmpeg-io

ffmpeg-io is a wrapper for the `ffmpeg` executable. In FMDT, this library is used in the `video` module (to read/write videos/images).

Note: ffmpeg-io requires the installation of the `ffmpeg` executable to work. The library mainly exchanges data with `ffmpeg` through system pipes.

5.4.2 NRC (Numerical Recipes in C)

NRC is a library dedicated to 1D and multidimensional efficient memory allocations. This library is used everywhere data allocation are needed.

5.4.3 C Vector

C Vector in a library that implements dynamic arrays like `std::vector` in C++. This is useful when we cannot predict in advance the size of a buffer. For instance, in FMDT, a C Vector is used to store the final tracks.

5.4.4 AFF3CT-core

AFF3CT-core is a library that includes a multi-threaded runtime. In FMDT, this multi-threaded runtime is used to speed the restitution time of the final executables. For instance, the `./src/detect_rt.cpp` is feature compliant with `./src/detect.cpp`. The main difference is that `./src/detect_rt.cpp` is multi-threaded with the AFF3CT-core library.

Note: AFF3CT-core is a C++ library. When FMDT is linked with AFF3CT-core, then the code requires a C++ compiler to be compiled.

5.4.5 OpenCV (Open Computer Vision library)

OpenCV is a famous library dedicated to a large set of computer vision algorithms. In FMDT, OpenCV is mainly used to write text in images.

Note: OpenCV is a C++ library. When FMDT is linked with OpenCV, then the code requires a C++ compiler to be compiled.

CHAPTER

SIX

CONVENTIONS

Start reading our code and you'll get the hang of it. For the readability, we apply some conventions detailed in the following sections.

This is open source software. Consider the people who will read your code, and make it look nice for them. It's sort of like driving a car: Perhaps you love doing donuts when you're alone, but with passengers the goal is to make the ride as smooth as possible.

6.1 Coding Conventions

6.1.1 General

- Indentation is made by using spaces (4 spaces).
- ALWAYS put spaces after list items and method parameters (`[1, 2, 3]`, not `[1,2,3]`), around operators (`x += 1`, not `x+=1`), and around hash arrows.
- The number of characters is limited to 120 per line of code.
- For data buffers, explicitly sized types from `stdint.h` should be preferred (for instance, `int` is NOT good and `int32_t` should be used instead).
- Please use unsigned integers to store data that cannot take negative values.
- Use double precision floating-points numbers ONLY when it is necessary. Most of the time, simple precision floating-points numbers should be enough.

6.1.2 Functions

- First parameters parenthesis is put directly after the function name (`motion_compute(int param)` is valid, while `motion_compute (int param)` is NOT valid).
- Parameters that are only read in the function have to be post-fixed by the `const` qualifier (ex.: `void my_func(const float* read_only_data, float* write_data)`),
- Braces are directly put after the last parameters parenthesis (see the example below).

```
void filename_verb(int param, int long_param_name) {
    for (int i = 0; i < 12; i++) {
        printf("Hello World %d\n", i);
    }
}
```

6.1.3 Structures and Enumerations

Here are some code examples to illustrate the conventions.

```
typedef struct {
    uint32_t attr1_var;
    uint32_t attr2_var;
    uint32_t* attr3_ptr;
} my_struct_t;
```

```
enum color_e { COLOR_MISC = 0,
               COLOR_GRAY,
               COLOR_GREEN,
               COLOR_RED,
               COLOR_PURPLE,
               COLOR_ORANGE,
               COLOR_BLUE,
               COLOR_YELLOW,
               N_COLORS
};
```

6.1.4 Conditional Structures and Loops

Here are some code examples to illustrate the conventions.

```
if (counter < 12 && is_valid) {
    // do something
} else {
    // do something else
}
```

```
switch (value) {
case 1:
    // do something
    break;
case 2:
    // do something
    break;
case 3:
    // do something
    break;
default:
    break;
}
```

```
for (int i = 0; i < 12; i++) {
    // do something
}
```

```
while (i < 100) {
    // do something
```

(continues on next page)

(continued from previous page)

```
    i++;
}
```

6.1.5 Source Code Auto-format

This project mainly follow LLVM coding conventions. For coding conventions (except for the naming) the code formatting can be automatized thanks to the `clang-format` parser. At the root of the project a `clang-format` configuration file is provided (see the `.clang-format` file).

For instance, if you want to auto-format the `src/motion.c` file you can run `clang-format` from the project root as follow:

```
clang-format -i src/motion.c
```

6.2 Naming Conventions

6.2.1 General

- This is an English code (functions/variables/defines/comments/... should be written in English).
- The snake case is used, (`my_variable`, not `myVariable`), classes start with an upper case (`My_class`, not `my_class`) and variables/methods/functions start with a lower case.

6.2.2 Variables

- Global variables are prefixed with `g_`.
- Parameter variables from the command line are prefixed with `p_`.
- If a variable contains more that one element, its name should ends with a "s" (ex.: `int values[100]`).
- Static variables from defines are all uppercase (ex.: `#define MY_STATIC_VAR 12`).
- Defines that come from the compiler should be prefixed with `FMDT_`.

6.2.3 Functions

- Function name starts with the corresponding module name (for instance, if you are in the `motion_compute.c` file and you want to write a function that compute the motion, the function name could be `motion_compute`).
- Function name should always contains a verb.

```
void filename_verb(int param, int long_param_name) {
    for (int i = 0; i < 12; i++) {
        printf("Hello World %d\n", i);
    }
}
```

6.2.4 Structures and Enumerations

- Structure name is always post-fixed with `_t` (ex.: `my_struct_t`).
- Enumeration name is always post-fixed with `_e` (ex.: `my_enum_e`).
- Enumeration values are in uppercase and always start with the name of the enumeration (in the following example `COLOR_`). Except for the last value that can be in the form `N_*s`.

```
enum color_e { COLOR_MISC = 0,
               COLOR_GRAY,
               COLOR_GREEN,
               COLOR_RED,
               COLOR_PURPLE,
               COLOR_ORANGE,
               COLOR_BLUE,
               COLOR_YELLOW,
               N_COLORS
};
```

CHAPTER

SEVEN

CONTRIBUTING GUIDE

FMDT code versioning is achieved thanks to Git. This section details how new contributions are integrated to the repository. There are two possible way to contribute depending on if your are a external contributor or if your are an inner contributor, see the next sections.

Important: The FMDT project exposes two mains protected branches: `master` and `develop`. The merge/pull requests are only accepted in the `develop` branch. In other words, all merge/pull requests targeting the `master` branch will be rejected.

Danger: Please read the coding conventions first in Section 6. Contributions that do not follow the coding and naming conventions will not be accepted!

7.1 Inner Contributions on GitLab

This is the inner workflow for people that have access to the private GitLab repository. In this repository, the `master` and `develop` branches are public because they are automatically mirrored on the public GitHub repository. By definitions, the other branches are private.

The way to contribute is to create a new branch from the `develop` to develop a new feature (lets call this a feature branch). When the feature branch is mature enough (and when it passes the CI (Continuous Integration) pipeline). The developer should send a **merge request** (MR (Merge Request)) from the feature branch into the `develop` branch. To send a MR in GitLab, you need to do it from the GitLab web interface. If you don't know how to do that, you can refer to the official documentation here: https://docs.gitlab.com/ee/user/project/merge_requests/.

Once your MR is submitted, your code will be reviewed and accepted later if it matches the requirements.

7.2 External Contributions on GitHub

External contributions are also more than welcome. Everyone can access and clone the public FMDT repository from GitHub (https://github.com/alsoc/fmdt).

The way to contribute is to submit PR (Pull Request) to the `develop` branch. This can be done from the GitHub web interface. If you don't know how to do that, you can refer to the official documentation here: https://docs.github.com/en/pull-requests/.

Once your PR is submitted, your code will be reviewed and accepted later if it matches the requirements.

7.3 Workflow Git

Every contributions are firstly merged in the `develop` branch. When we consider that the current state of the `develop` branch is stable enough, a versioning tag (for instance `v1.0.0`) is added to a specific commit in the `develop` branch, then the `develop` branch is merge in the `master` branch.

CHAPTER

EIGHT

CONTINUOUS INTEGRATION

A CI pipeline is setup in the private GitLab repository. It is composed of 4 stages:

1. Static analysis: for now there is only one job in the stage that compiles the documentation.
2. Build: this stage compiles FMDT on various compilers and with various compiler definitions.
3. Test: regression tests and memory leaks tests are performed.
4. Coverage: the code coverage of the regression tests is computed.

The CI pipeline is triggered after each `push` on the GitLab repository. The jobs are executed on runners hosted by the LIP6 laboratory. The jobs can easily be deployed thanks to the use of Docker images. The public AFF3CT container registry is used (https://gitlab.com/aff3ct/aff3ct/container_registry).

CHAPTER

NINE

LIBRARY API

9.1 Class Hierarchy

9.2 File Hierarchy

9.3 Full API

9.3.1 Namespaces

Namespace std

STL namespace.

9.3.2 Classes and Structs

Struct BB_t

- Defined in file_c_fmdt_tracking_tracking_struct.h

Struct Documentation

struct **BB_t**

Bounding box structure. Used to represent the bounding box around a RoI.

Public Members

uint32_t **frame_id**

Frame id corresponding to the bounding box.

uint32_t **track_id**

Track id corresponding to the bounding box.

uint32_t **bb_x**

Center x of the bounding box.

uint32_t **bb_y**

Center y of the bounding box.

uint32_t **rx**

Radius x of the bounding box.

uint32_t **ry**

Radius y of the bounding box.

int **is_extrapolated**

Boolean that defines if the bounding box is a real bounding box (from a connected-component) or if it has been extrapolated in the tracking.

Struct CCL_data_t

- Defined in file_c_fmdt_CCL_CCL_struct.h

Struct Documentation

struct **CCL_data_t**

Inner CCL data required to perform labeling.

Public Members

int **i0**

First y index in the image (included).

int **i1**

Last y index in the image (included).

int **j0**

First x index in the image (included).

int **j1**

Last x index in the image (included).

uint32_t ****er**

Relative labels.

uint32_t ****era**

Relative <-> absolute labels equivalences.

uint32_t **rlc

Run-length coding.

uint32_t *eq

Table of equivalence.

uint32_t *ner

Number of relative labels.

Struct img_data_t

- Defined in file_c_fmdt_image_image_struct.h

Struct Documentation

struct img_data_t

Image data structure. Used for storing images according to different libraries (OpenCV / NRC). Note that this container can be used for grayscale and color images because it relies on opaque types.

Public Members

size_t height

Image height.

size_t width

Image width.

void *pixels

Opaque type, contains image data (= the pixels).

void *container_2d

Opaque type, contains 2D image container.

Struct kNN_data_t

- Defined in file_c_fmdt_kNN_kNN_struct.h

Struct Documentation

struct **kNN_data_t**

Inner data structure required to compute associations between RoIs.

Public Members

float ****distances**

2D array of euclidean distances ([_max_size][_max_size]). y axis represents RoIs at $t-1$ and x axis represents RoIs at t. For instance, `distances`$[i][j]$ represents the distance between RoI_{t-1}^{i} and RoI_{t}^{j}. Note that sometime, for efficiency reasons, the implementation may choose to store squared euclidean distances instead of euclidean distances.

uint32_t ****nearest**

2D array of ranks ([_max_size][_max_size]). y axis represents RoIs at $t-1$ and x axis represents RoIs at t. For instance, `nearest`$[i][j]$ represents the rank of RoI_{t-1}^{i} and RoI_{t}^{j}. Rank = 1 means that i and j are the closest possible RoIs association, rank = 2 means that i and j are the second closest possible RoIs association, and so on. Rank = 0 means that i and j were not associated together (common reason is that they are too far from each others).

uint32_t ***conflicts**

1D array of conflicts ([_max_size]). A conflict happens when they are more than one RoI_{t-1} that is the closet to RoI_{t}^{j}. `conflicts`$[j]$ contains 0 if there is no conflict. `conflicts`$[j]$ contains more than 0 if there are conflicts. For instance if RoI_{t-1}^{i1}, RoI_{t-1}^{i2} and RoI_{t-1}^{i3} are all the closest to RoI_{t}^{j}, then `conflicts`$[j] = 2$. This buffer is allocated only if the FMDT_ENABLE_DEBUG macro is defined.

size_t **_max_size**

Maximum number of RoIs allocated in the previous fields.

Struct motion_t

- Defined in file_c_fmdt_motion_motion_struct.h

Struct Documentation

struct **motion_t**

Structure that defines the global motion estimation between two consecutive images at $t-1$ and t. These fields define an angle and a translation vector from I_t to I_{t-1}.

Public Members

float **theta**

Rotation angle in radian.

float **tx**

x component of the translation vector.

float **ty**

y component of the translation vector.

float **mean_error**

Mean error of the global motion estimation.

float **std_deviation**

Standard deviation of the global motion estimation.

Struct rgb8_t

- Defined in file_c_fmdt_image_image_struct.h

Struct Documentation

struct **rgb8_t**

Red Green Blue (RGB) structure.

Public Members

uint8_t **r**

Red color component.

uint8_t **g**

Green color component.

uint8_t **b**

Blue color component.

Struct RoI_t

- Defined in file_c_fmdt_tracking_tracking_struct.h

Struct Documentation

struct **RoI_t**

Features required in the tracking.

Public Members

uint32_t **id**

RoI unique identifiers. A RoI identifier should starts from 1 while 0 should be reserved for uninitialized structure.

uint32_t **frame**

Frame number of the RoI.

uint32_t **xmin**

Minimum x coordinates of the bounding box.

uint32_t **xmax**

Maximum x coordinates of the bounding box.

uint32_t **ymin**

Minimum y coordinates of the bounding box.

uint32_t **ymax**

Maximum y coordinates of the bounding box.

uint32_t **S**

Numbers of points/pixels = surfaces of the RoIs.

float **x**

x coordinates of the centroid ($x = S_x/S$).

float **y**

y coordinates of the centroid ($y = S_y/S$).

uint32_t **prev_id**

Previous corresponding RoI identifiers ($RoI_{t-1} \leftrightarrow RoI_t$).

uint32_t **next_id**

Next corresponding RoI identifiers ($RoI_t \leftrightarrow RoI_{t+1}$).

float **dx**

x components of the distance between centroids at $t-1$ and t.

float **dy**

y components of the distance between centroids at $t-1$ and t.

float **error**

Velocity norm / error. $e = \sqrt{dx^2 + dy^2}$.

uint32_t **magnitude**

Magnitudes or brightness of the RoIs. Sums of the pixels intensities.

uint32_t **time**

Number of times the RoI and its predecessors have been associated (non-moving RoI).

uint32_t **time_motion**

Number of times the RoI and its predecessors have been associated (moving RoI).

uint8_t **is_extrapolated**

Boolean that defines if this RoI has been extrapolated. It prevents to associate it to a new track.

Struct RoIs_asso_t

- Defined in file_c_fmdt_features_features_struct.h

Struct Documentation

struct **RoIs_asso_t**

Associations between RoIs. $RoI_{t-1} \leftrightarrow RoI_t$ and $RoI_t \leftrightarrow RoI_{t+1}$. Generally these associations are computed by a k-Nearest Neighbors (k-NN) matching algorithm. The memory layout is a Structure of Arrays (SoA), each field is an array of `_max_size` capacity (except for `_max_size` itself and `_size` fields that are both scalar values).

Public Members

uint32_t ***id**

RoI unique identifiers. A RoI identifier should starts from 1 while 0 should be reserved for uninitialized structure.

uint32_t ***prev_id**

Previous corresponding RoI identifiers ($RoI_{t-1} \leftrightarrow RoI_t$).

uint32_t ***next_id**

Next corresponding RoI identifiers ($RoI_t \leftrightarrow RoI_{t+1}$).

size_t *_size

Current number of RoIs in each *feature* field. Note: this field a pointer but it has to be a scalar value.

size_t *_max_size

Maximum capacity of each *feature* field (= maximum number of elements in the arrays). Note: this field a pointer but it has to be a scalar value.

Struct RoIs_basic_t

- Defined in file_c_fmdt_features_features_struct.h

Struct Documentation

struct **RoIs_basic_t**

Basic features: bounding box, surface & centroid. A bounding box represents a rectangular box around the RoI. The surface is the number of pixels that are in the connected-component (CC). The centroid is the center of mass of the RoI. The memory layout is a Structure of Arrays (SoA), each field is an array of `_max_size` capacity (except for `_max_size` itself and `_size` fields that are both scalar values).

Public Members

uint32_t *id

RoI unique identifiers. A RoI identifier should starts from 1 while 0 should be reserved for uninitialized structure.

uint32_t *xmin

Minimum x coordinates of the bounding box.

uint32_t *xmax

Maximum x coordinates of the bounding box.

uint32_t *ymin

Minimum y coordinates of the bounding box.

uint32_t *ymax

Maximum y coordinates of the bounding box.

uint32_t *S

Numbers of points/pixels = surfaces of the RoIs.

uint32_t *Sx

Sums of x properties.

uint32_t *Sy

Sums of y properties.

float *x

x coordinates of the centroid ($x = S_x/S$).

float *y

y coordinates of the centroid ($y = S_y/S$).

size_t *_size

Current number of RoIs in each *feature* field. Note: this field a pointer but it has to be a scalar value.

size_t *_max_size

Maximum capacity of each *feature* field (= maximum number of elements in the arrays). Note: this field a pointer but it has to be a scalar value.

Struct RoIs_history_t

- Defined in file_c_fmdt_tracking_tracking_struct.h

Struct Documentation

struct **RoIs_history_t**

History of the previous RoI features. This structure allows to access RoI in the past frames. RoIs at t are stored in the first array element while RoIs at $t -$ `_size` are store in the `_size` $- 1$ element. The memory layout is a Structure of Arrays (SoA), each field is an array of `_max_size` capacity (except for `_max_size` itself and `_size` fields that are both scalar values).

Public Members

RoI_t **array

2D array of RoIs, the first dimension is the time and the second dimension is the RoIs at a given time.

motion_t *motion

Array of motion estimations.

uint32_t *n_RoIs

Array of numbers of RoIs.

uint32_t _max_n_RoIs

Maximum number of RoIs.

size_t _size

Current size/utilization of the fields.

size_t **_max_size**

Maximum capacity of data that can be contained in the fields.

Struct RoIs_misc_t

- Defined in file_c_fmdt_features_features_struct.h

Struct Documentation

struct **RoIs_misc_t**

Miscellaneous features. This structure contains features that are considered "less important" than others. The memory layout is a Structure of Arrays (SoA), each field is an array of `_max_size` capacity (except for `_max_size` itself and `_size` fields that are both scalar values).

Public Members

uint32_t ***id**

RoI unique identifiers. A RoI identifier should starts from 1 while 0 should be reserved for uninitialized structure.

uint32_t ***magnitude**

Magnitudes or brightness of the RoIs. Sums of the pixels intensities.

uint32_t ***sat_count**

Number of pixels that are saturated in the CC. A pixel is saturated if its intensity I_p is equal to the maximum value (here it is 255).

size_t ***_size**

Current size/utilization of the fields. Note: it is allocated on the heap but it represents only one value.

size_t ***_max_size**

Maximum capacity of data that can be contained in the fields. Note: it is allocated on the heap but it represents only one value.

Struct RoIs_motion_t

- Defined in file_c_fmdt_features_features_struct.h

Struct Documentation

struct **RoIs_motion_t**

Motion between RoI at $t-1$ and t. The features of this structure are values computed after motion compensation. The memory layout is a Structure of Arrays (SoA), each field is an array of `_max_size` capacity (except for `_max_size` itself and `_size` fields that are both scalar values).

Public Members

uint32_t ***id**

RoI unique identifiers. A RoI identifier should starts from 1 while 0 should be reserved for uninitialized structure.

float ***dx**

x components of the distance between centroids at $t-1$ and t. It can represent either abscissa velocity (if `is_moving` == 1) or abscissa error distance (if `is_moving` == 0).

float ***dy**

y components of the distance between centroids at $t-1$ and t. It can represent either ordinate velocity (if `is_moving` == 1) or ordinate error distance if (`is_moving` == 0).

float ***error**

Velocity norm (if `is_moving` == 1) or error (if `is_moving` == 0). $e = \sqrt{dx^2 + dy^2}$.

uint8_t ***is_moving**

Boolean that defines if the RoI is moving (`is_moving` == 1) or not (`is_moving` == 0).

size_t ***_size**

Current size/utilization of the fields. Note: it is allocated on the heap but it represents only one value.

size_t ***_max_size**

Maximum capacity of data that can be contained in the fields. Note: it is allocated on the heap but it represents only one value.

Struct RoIs_t

- Defined in file_c_fmdt_features_features_struct.h

Struct Documentation

struct **RoIs_t**

Structure of RoI structures. This structure contains all previously defined RoI structures. `id`, `_size` and `_max_size` fields are shared with the sub-structures.

See also:

RoIs_basic_t.

See also:

RoIs_asso_t.

See also:

RoIs_motion_t.

See also:

RoIs_misc_t.

Public Members

uint32_t ***id**

RoI unique identifiers. A RoI identifier should starts from 1 while 0 should be reserved for uninitialized structure.

RoIs_basic_t ***basic**

Basic features.

RoIs_asso_t ***asso**

Association features.

RoIs_motion_t ***motion**

Motion features.

RoIs_misc_t ***misc**

Miscellaneous features.

size_t **_size**

Current size/utilization of the fields.

size_t **_max_size**

Maximum capacity of data that can be contained in the fields.

Struct track_t

- Defined in file_c_fmdt_tracking_tracking_struct.h

Struct Documentation

struct **track_t**

Description of a track.

Public Members

uint32_t **id**

Track unique identifiers. A track identifier should starts from 1 while 0 should be reserved for uninitialized structure.

RoI_t **begin**

First RoI corresponding to this track.

RoI_t **end**

Last RoI corresponding to this track.

float **extrapol_x**

Last x position of the extrapolated track (used only if `state` == `STATE_LOST`).

float **extrapol_y**

Last y position of the extrapolated track (used only if `state` == `STATE_LOST`).

float **extrapol_u**

Velocity x estimation of the track for extrapolation (used only if `state` == `STATE_LOST`).

float **extrapol_v**

Velocity y estimation of the track for extrapolation (used only if `state` == `STATE_LOST`).

uint8_t **extrapol_order**

Number of times this track has been extrapolated (used only if `state` == `STATE_LOST`).

enum *state_e* **state**

State of the track.

enum *obj_e* **obj_type**

Object type (classification).

enum *change_state_reason_e* **change_state_reason**

Reason of the noise type classification.

vec_uint32_t **magnitude**

Vector of the magnitudes history of this track.

Struct tracking_data_t

- Defined in file_c_fmdt_tracking_tracking_struct.h

Struct Documentation

struct **tracking_data_t**

Inner data used by the tracking.

Public Members

vec_track_t **tracks**

Vector of tracks.

RoIs_history_t ***RoIs_history**

RoIs history.

RoI_t ***RoIs_list**

List of RoIs. This is a temporary array used to group all the RoIs belonging to a same track.

Struct validation_obj_t

- Defined in file_c_fmdt_validation_validation_struct.h

Struct Documentation

struct **validation_obj_t**

Data corresponding to a ground truth track.

Public Members

int16_t **t0**

float **x0**

float **y0**

int16_t **t1**

float **x1**

float **y1**

int16_t **t0_min**

int16_t **t1_max**

int **track_t0**

int **track_t1**

float **track_y0**

float **track_x0**

float **track_x1**

float **track_y1**

float **bb_x0**

float **bb_x1**

float **bb_y0**

float **bb_y1**

int16_t **bb_x0_m**

int16_t **bb_x1_m**

int16_t **bb_y0_m**

int16_t **bb_y1_m**

int16_t **bb_x0_p**

int16_t **bb_x1_p**

int16_t **bb_y0_p**

int16_t **bb_y1_p**

float **a**

float **b**

uint8_t **dirX**

uint8_t **dirY**

track_t ***track**

unsigned **track_id**

float **xt**

float **yt**

uint16_t **nb_tracks**

uint16_t **hits**

uint16_t **is_valid**

uint16_t **is_valid_last**

enum *obj_e* **obj_type**

Struct video_reader_t

- Defined in file_c_fmdt_video_video_struct.h

Struct Documentation

struct **video_reader_t**

Video reader structure.

Public Members

ffmpeg_options **ffmpeg_opts**

FFMPEG options.

ffmpeg_handle **ffmpeg**

FFMPEG handle.

size_t **frame_start**

Start frame number (first frame is frame 0).

size_t **frame_end**

Last frame number.

size_t **frame_skip**

Number of frames to skip between two frames (0 means no frame is skipped).

size_t **frame_current**

Current frame number (always starts to 0, even if `frame_start` > 0).

char **path**[2048]

Path to the video or images.

uint8_t ***fra_buffer**

Buffer containing the all frames in memory (may be allocated or not depending on the implementation).

size_t **fra_count**

Number of frames in `fra_buffer` array.

size_t **loop_size**

Number of times the video sequence should be played in loop (1 means that the video sequence is played once).

size_t **cur_loop**

Current loop.

Struct video_writer_t

- Defined in file_c_fmdt_video_video_struct.h

Struct Documentation

struct **video_writer_t**

Video writer structure.

Public Members

ffmpeg_options **ffmpeg_opts**

FFMPEG options.

ffmpeg_handle **ffmpeg**

FFMPEG handle.

char **path**[2048]

Path to the video or images.

9.3.3 Enums

Enum change_state_reason_e

- Defined in file_c_fmdt_tracking_tracking_struct.h

Enum Documentation

enum **change_state_reason_e**

Enumeration of the possible reasons why an `OBJ_METEOR` has been finally classified as an `OBJ_NOISE`.

Values:

enumerator **REASON_UNKNOWN**

Unknown (= uninitialized).

enumerator **REASON_TOO_BIG_ANGLE**

Angle made by the 3 last positions is to big.

enumerator **REASON_WRONG_DIRECTION**

Track radically changed its direction.

enumerator **REASON_TOO_LONG_DURATION**

Track lived a too long time to be a meteor.

enumerator **N_REASONS**

Number of reasons in the enumeration.

Enum color_e

- Defined in file_c_fmdt_image_image_struct.h

Enum Documentation

enum **color_e**

Enumeration for colors.

Values:

enumerator **COLOR_MISC**

Miscellaneous color (= uninitialized).

enumerator **COLOR_GRAY**

Gray color.

enumerator **COLOR_GREEN**

Green color.

enumerator **COLOR_RED**

Red color.

enumerator **COLOR_PURPLE**

Purple color.

enumerator **COLOR_ORANGE**

Orange color.

enumerator **COLOR_BLUE**

Blue color.

enumerator **COLOR_YELLOW**

Yellow color.

enumerator **N_COLORS**

Number of colors in the enumeration.

Enum obj_e

- Defined in file_c_fmdt_tracking_tracking_struct.h

Enum Documentation

enum **obj_e**

Enumeration of the different object types (= object classification).

Values:

enumerator **OBJ_UNKNOWN**

Unknown (= uninitialized).

enumerator **OBJ_METEOR**

Meteor.

enumerator **OBJ_STAR**

Star.

enumerator **OBJ_NOISE**

Noise (generally noise means that it is not a meteor and not a star).

enumerator **N_OBJECTS**

Number of objects in the enumeration.

Enum pixfmt_e

- Defined in file_c_fmdt_video_video_struct.h

Enum Documentation

enum **pixfmt_e**

Pixel formats enumeration.

Values:

enumerator **PIXFMT_RGB24**

24 bits Red-Green-Blue.

enumerator **PIXFMT_GRAY**

8 bits grayscale.

Enum state_e

- Defined in file_c_fmdt_tracking_tracking_struct.h

Enum Documentation

enum **state_e**

Enumeration of the states in the tracking finite-state machine.

Values:

enumerator **STATE_UNKNOWN**

Unknown (= uninitialized).

enumerator **STATE_UPDATED**

Track has been updated (or created).

enumerator **STATE_LOST**

Track has not been updated, it is lost.

enumerator **STATE_FINISHED**

Track is finished.

enumerator **N_STATES**

Number of states in the enumeration.

9.3.4 Functions

Function _CCL_LSL_apply

- Defined in file_c_fmdt_CCL_CCL_compute.h

Function Documentation

uint32_t **_CCL_LSL_apply**(uint32_t **CCL_data_er, uint32_t **CCL_data_era, uint32_t **CCL_data_rlc, uint32_t *CCL_data_eq, uint32_t *CCL_data_ner, const uint8_t **img, uint32_t **labels, const int i0, const int i1, const int j0, const int j1)

Compute the Light Speed Labeling (LSL) algorithm.

Parameters

- **CCL_data_er** – Relative labels (2D array $[i1 - i0 + 1][j1 - j0 + 1]$).
- **CCL_data_era** – Relative <-> absolute labels equivalences (2D array $[i1 - i0 + 1][j1 - j0 + 1]$).
- **CCL_data_rlc** – Run-length coding (2D array $[i1 - i0 + 1][j1 - j0 + 1]$).
- **CCL_data_eq** – Table of equivalence (1D array $[(i1 - i0 + 1) * (j1 - j0 + 1)]$).

- **CCL_data_ner** – Number of relative labels (1D array $[i1 - i0 + 1]$).
- **img** – Input binary image (2D array $[i1 - i0 + 1][j1 - j0 + 1]$, $\{0, 1\}$ has to be coded as $\{0, 255\}$).
- **labels** – Output labels (2D array $[i1 - i0 + 1][j1 - j0 + 1]$. The labels are in $[1; 2^{32} - 1]$ and 0 value means no label).
- **i0** – First y index in the image (included).
- **i1** – Last y index in the image (included).
- **j0** – First x index in the image (included).
- **j1** – Last x index in the image (included).

Returns Number of labels.

Function _features_compute_magnitude

- Defined in file_c_fmdt_features_features_compute.h

Function Documentation

void **_features_compute_magnitude**(const uint8_t **img, const uint32_t img_width, const uint32_t img_height, const uint32_t **labels, const uint32_t *RoIs_xmin, const uint32_t *RoIs_xmax, const uint32_t *RoIs_ymin, const uint32_t *RoIs_ymax, const uint32_t *RoIs_S, uint32_t *RoIs_magnitude, uint32_t *RoIs_sat_count, const size_t n_RoIs)

Compute magnitude features. The magnitude represents the brightness of a RoI. In a first time, the sum of the pixels intensities is performed. In a second time, the noise level around the connected-component is subtracted to give a better estimation of the real brightness. The magnitude can be defined as follow: $M = \sum_{p=0}^{P} i_p - \left((\sum_{n=0}^{N} i_n)/N\right) \times P$, where P is the the number of pixels in the current CC, i_x is the brightness of the pixel x and N is the number of noisy pixels considered. In addition, this function can also compute the saturation counter for each RoI (e. g. the number of pixels that have an intensity $I_p = 255$).

See also:

RoIs_basic_t for more explanations about the basic features.

See also:

RoIs_misc_t for more explanations about the miscellaneous features.

Parameters

- **img** – Image in grayscale ([img_height][img_width], the values of the pixel range are $[0; 255]$).
- **img_width** – Image width.
- **img_height** – Image height.
- **labels** – 2D array of labels ([img_height][img_width]).
- **RoIs_xmin** – Array of minimum x coordinates of the bounding box.

- **RoIs_xmax** – Array of maximum x coordinates of the bounding box.
- **RoIs_ymin** – Array of minimum y coordinates of the bounding box.
- **RoIs_ymax** – Array of maximum y coordinates of the bounding box.
- **RoIs_S** – Array of RoI surfaces.
- **RoIs_magnitude** – Array of RoI magnitudes.
- **RoIs_sat_count** – Array of RoI saturation counters (if NULL, the saturation counter is not computed).
- **n_RoIs** – Number of connected-components (= number of RoIs).

Function _features_extract

- Defined in file_c_fmdt_features_features_compute.h

Function Documentation

void **_features_extract**(const uint32_t **labels, const int i0, const int i1, const int j0, const int j1, uint32_t *RoIs_id, uint32_t *RoIs_xmin, uint32_t *RoIs_xmax, uint32_t *RoIs_ymin, uint32_t *RoIs_ymax, uint32_t *RoIs_S, uint32_t *RoIs_Sx, uint32_t *RoIs_Sy, float *RoIs_x, float *RoIs_y, const size_t n_RoIs)

Basic features extraction from a 2D array of `labels`. In other words, this function converts a (sparse ?) 2-dimensional representation of connected-components (CCs) into a list of CCs.

See also:

RoIs_basic_t for more explanations about the features.

Parameters

- **labels** – 2D array of labels ($[i1 - i0 + 1][j1 - j0 + 1]$).
- **i0** – First y index in the labels (included).
- **i1** – Last y index in the labels (included).
- **j0** – First x index in the labels (included).
- **j1** – Last x index in the labels (included).
- **RoIs_id** – Array of RoI unique identifiers.
- **RoIs_xmin** – Array of minimum x coordinates of the bounding box.
- **RoIs_xmax** – Array of maximum x coordinates of the bounding box.
- **RoIs_ymin** – Array of minimum y coordinates of the bounding box.
- **RoIs_ymax** – Array of maximum y coordinates of the bounding box.
- **RoIs_S** – Array of RoI surfaces.
- **RoIs_Sx** – Array of sums of x properties.
- **RoIs_Sy** – Array of sums of y properties.
- **RoIs_x** – Array of centroids abscissa.
- **RoIs_y** – Array of centroids ordinate.

- **n_RoIs** – Number of connected-components (= number of RoIs) in the 2D array of `labels`.

Function _features_merge_CCL_HI_v2

- Defined in file_c_fmdt_features_features_compute.h

Function Documentation

void **_features_merge_CCL_HI_v2**(const uint32_t **in_labels, const uint8_t **img_HI, uint32_t **out_labels, const int i0, const int i1, const int j0, const int j1, uint32_t *RoIs_id, const uint32_t *RoIs_xmin, const uint32_t *RoIs_xmax, const uint32_t *RoIs_ymin, const uint32_t *RoIs_ymax, const uint32_t *RoIs_S, const size_t n_RoIs, const uint32_t S_min, const uint32_t S_max)

Hysteresis re-labeling and morphological thresholding. From a 2D array of labels (`in_label`) and a binary image (`img_HI`), the function generates a new 2D array of labels (`out_labels`). The newly produced labels (`out_labels`) are a sub-set of the "old" labels (`in_labels`). Labels from `in_labels` are kept in `out_labels` only if at least one pixel of the current connected-component exists in the binary image (`img_HI`). Finally, this function performs a morphological thresholding as follow: if $S_{min} > S$ or $S > S_{max}$ then the corresponding `RoIs_id` is set to 0.

See also:

RoIs_basic_t for more explanations about the features.

Parameters

- **in_labels** – Input 2D array of labels ($[i1 - i0 + 1][j1 - j0 + 1]$).
- **img_HI** – Binary image (2D array $[i1 - i0 + 1][j1 - j0 + 1]$, $\{0, 1\}$ has to be coded as $\{0, 255\}$). This image results from a threshold filter on the original image. This threshold filter should be higher than the first one used to compute the initial labels (`in_labels`).
- **out_labels** – Output 2D array of labels ($[i1 - i0 + 1][j1 - j0 + 1]$). `out_labels` can be NULL, this way only the features will be updated. `out_labels` can also be the same pointer as `in_labels`, this way the output labels will be computed in place.
- **i0** – First y index in the labels (included).
- **i1** – Last y index in the labels (included).
- **j0** – First x index in the labels (included).
- **j1** – Last x index in the labels (included).
- **RoIs_id** – Array of RoI unique identifiers.
- **RoIs_xmin** – Array of minimum x coordinates of the bounding box.
- **RoIs_xmax** – Array of maximum x coordinates of the bounding box.
- **RoIs_ymin** – Array of minimum y coordinates of the bounding box.
- **RoIs_ymax** – Array of maximum y coordinates of the bounding box.
- **RoIs_S** – Array of RoI surfaces.
- **n_RoIs** – Number of connected-components (= number of RoIs) in the 2D array of `in_labels`.
- **S_min** – Minimum morphological threshold.

- **S_max** – Maximum morphological threshold.

Function _features_RoIs0_RoIs1_write

- Defined in file_c_fmdt_features_features_io.h

Function Documentation

void **_features_RoIs0_RoIs1_write**(FILE *f, const int prev_frame, const int cur_frame, const uint32_t *RoIs0_id, const uint32_t *RoIs0_xmin, const uint32_t *RoIs0_xmax, const uint32_t *RoIs0_ymin, const uint32_t *RoIs0_ymax, const uint32_t *RoIs0_S, const uint32_t *RoIs0_Sx, const uint32_t *RoIs0_Sy, const float *RoIs0_x, const float *RoIs0_y, const uint32_t *RoIs0_magnitude, const uint32_t *RoIs0_sat_count, const size_t n_RoIs0, const uint32_t *RoIs1_id, const uint32_t *RoIs1_xmin, const uint32_t *RoIs1_xmax, const uint32_t *RoIs1_ymin, const uint32_t *RoIs1_ymax, const uint32_t *RoIs1_S, const uint32_t *RoIs1_Sx, const uint32_t *RoIs1_Sy, const float *RoIs1_x, const float *RoIs1_y, const uint32_t *RoIs1_magnitude, const uint32_t *RoIs1_sat_count, const size_t n_RoIs1, const *vec_track_t* tracks)

Print two tables of RoIs, one at $t-1$ and one at t.

See also:

RoIs_basic_t for more explanations about the features.

See also:

RoIs_misc_t for more explanations about the features.

Parameters

- **f** – File descriptor (in write mode).
- **prev_frame** – Frame id corresponding to the RoIs at $t-1$.
- **cur_frame** – Frame id corresponding to the RoIs at t.
- **RoIs0_id** – Array of RoI unique identifiers (at $t-1$).
- **RoIs0_xmin** – Array of minimum x coordinates of the bounding box (at $t-1$).
- **RoIs0_xmax** – Array of maximum x coordinates of the bounding box (at $t-1$).
- **RoIs0_ymin** – Array of minimum y coordinates of the bounding box (at $t-1$).
- **RoIs0_ymax** – Array of maximum y coordinates of the bounding box (at $t-1$).
- **RoIs0_S** – Array of RoI surfaces (at $t-1$).
- **RoIs0_Sx** – Array of sums of x properties (at $t-1$).
- **RoIs0_Sy** – Array of sums of y properties (at $t-1$).
- **RoIs0_x** – Array of centroids abscissa (at $t-1$).
- **RoIs0_y** – Array of centroids ordinate (at $t-1$).
- **RoIs0_magnitude** – Array of RoI magnitudes (at $t-1$) (if NULL, the magnitudes are not shown).

- **RoIs0_sat_count** – Array of RoI saturation counters (at $t-1$) (if NULL, the saturation counters are not shown).
- **n_RoIs0** – Number of connected-components (= number of RoIs) in the 2D array of `labels` (at $t-1$).
- **RoIs1_id** – Array of RoI unique identifiers (at t).
- **RoIs1_xmin** – Array of minimum x coordinates of the bounding box (at t).
- **RoIs1_xmax** – Array of maximum x coordinates of the bounding box (at t).
- **RoIs1_ymin** – Array of minimum y coordinates of the bounding box (at t).
- **RoIs1_ymax** – Array of maximum y coordinates of the bounding box (at t).
- **RoIs1_S** – Array of RoI surfaces (at t).
- **RoIs1_Sx** – Array of sums of x properties (at t).
- **RoIs1_Sy** – Array of sums of y properties (at t).
- **RoIs1_x** – Array of centroids abscissa (at t).
- **RoIs1_y** – Array of centroids ordinate (at t).
- **RoIs1_magnitude** – Array of RoI magnitudes (at t) (if NULL, the magnitudes are not shown).
- **RoIs1_sat_count** – Array of RoI saturation counters (at t) (if NULL, the saturation counters are not shown).
- **n_RoIs1** – Number of connected-components (= number of RoIs) in the 2D array of `labels` (at t).
- **tracks** – Vector of tracks. It enables to match RoIs with corresponding track in the table of RoIs.

Function _features_RoIs_write

- Defined in file_c_fmdt_features_features_io.h

Function Documentation

void **_features_RoIs_write**(FILE *f, const int frame, const uint32_t *RoIs_id, const uint32_t *RoIs_xmin, const uint32_t *RoIs_xmax, const uint32_t *RoIs_ymin, const uint32_t *RoIs_ymax, const uint32_t *RoIs_S, const uint32_t *RoIs_Sx, const uint32_t *RoIs_Sy, const float *RoIs_x, const float *RoIs_y, const uint32_t *RoIs_magnitude, const uint32_t *RoIs_sat_count, const size_t n_RoIs, const *vec_track_t* tracks, const unsigned age)

Print a table of RoIs.

See also:

RoIs_basic_t for more explanations about the features.

See also:

RoIs_misc_t for more explanations about the features.

Parameters

- **f** – File descriptor (in write mode).

- **frame** – Frame id corresponding to the RoIs.
- **RoIs_id** – Array of RoI unique identifiers.
- **RoIs_xmin** – Array of minimum x coordinates of the bounding box.
- **RoIs_xmax** – Array of maximum x coordinates of the bounding box.
- **RoIs_ymin** – Array of minimum y coordinates of the bounding box.
- **RoIs_ymax** – Array of maximum y coordinates of the bounding box.
- **RoIs_S** – Array of RoI surfaces.
- **RoIs_Sx** – Array of sums of x properties.
- **RoIs_Sy** – Array of sums of y properties.
- **RoIs_x** – Array of centroids abscissa.
- **RoIs_y** – Array of centroids ordinate.
- **RoIs_magnitude** – Array of RoI magnitudes (if NULL, the magnitudes are not shown).
- **RoIs_sat_count** – Array of RoI saturation counters (if NULL, the saturation counters are not shown).
- **n_RoIs** – Number of connected-components (= number of RoIs) in the 2D array of `labels`.
- **tracks** – Vector of tracks. It enables to match RoIs with corresponding track in the table of RoIs.
- **age** – 0 if `frame` is the current frame, 1 if `frame` is the $t-1$ frame. This is mandatory to find the corresponding track (if any).

Function _features_shrink

- Defined in file_c_fmdt_features_features_compute.h

Function Documentation

size_t **_features_shrink**(const uint32_t *RoIs_src_id, const uint32_t *RoIs_src_xmin, const uint32_t *RoIs_src_xmax, const uint32_t *RoIs_src_ymin, const uint32_t *RoIs_src_ymax, const uint32_t *RoIs_src_S, const uint32_t *RoIs_src_Sx, const uint32_t *RoIs_src_Sy, const float *RoIs_src_x, const float *RoIs_src_y, const size_t n_RoIs_src, uint32_t *RoIs_dst_id, uint32_t *RoIs_dst_xmin, uint32_t *RoIs_dst_xmax, uint32_t *RoIs_dst_ymin, uint32_t *RoIs_dst_ymax, uint32_t *RoIs_dst_S, uint32_t *RoIs_dst_Sx, uint32_t *RoIs_dst_Sy, float *RoIs_dst_x, float *RoIs_dst_y)

Shrink features. Remove features when feature identifier value is 0. Source features (`RoIs_src_X`) are copied into destination features (`RoIs_dst_X`) if `RoIs_src_id` > 0.

See also:

features_merge_CCL_HI_v2 for more explanations about why some identifiers can be set to 0.

See also:

RoIs_basic_t for more explanations about the features.

Parameters

- **RoIs_src_id** – Source array of RoI unique identifiers.

- **RoIs_src_xmin** – Source array of minimum x coordinates of the bounding box.
- **RoIs_src_xmax** – Source array of maximum x coordinates of the bounding box.
- **RoIs_src_ymin** – Source array of minimum y coordinates of the bounding box.
- **RoIs_src_ymax** – Source array of maximum y coordinates of the bounding box.
- **RoIs_src_S** – Source array of RoI surfaces.
- **RoIs_src_Sx** – Source array of sums of x properties.
- **RoIs_src_Sy** – Source array of sums of y properties.
- **RoIs_src_x** – Source array of centroids abscissas.
- **RoIs_src_y** – Source array of centroids ordinates.
- **n_RoIs_src** – Number of RoIs in the previous arrays.
- **RoIs_dst_id** – Destination array of RoI unique identifiers.
- **RoIs_dst_xmin** – Destination array of minimum x coordinates of the bounding box.
- **RoIs_dst_xmax** – Destination array of maximum x coordinates of the bounding box.
- **RoIs_dst_ymin** – Destination array of minimum y coordinates of the bounding box.
- **RoIs_dst_ymax** – Destination array of maximum y coordinates of the bounding box.
- **RoIs_dst_S** – Destination array of RoI surfaces.
- **RoIs_dst_Sx** – Destination array of sums of x properties.
- **RoIs_dst_Sy** – Destination array of sums of y properties.
- **RoIs_dst_x** – Destination array of centroids abscissas.
- **RoIs_dst_y** – Destination array of centroids ordinates.

Returns Number of regions of interest (RoIs) after the data shrink.

Function _image_gs_draw_labels

- Defined in file_c_fmdt_image_image_compute.h

Function Documentation

void **_image_gs_draw_labels**(*img_data_t* *img_data, const uint32_t **labels, const uint32_t *RoIs_id, const uint32_t *RoIs_xmax, const uint32_t *RoIs_ymin, const uint32_t *RoIs_ymax, const size_t n_RoIs, const uint8_t show_id)

Convert labels into a black & white image. If the program is linked with the OpenCV library, then the `show_id` boolean can be used to draw the label number on the black & white image.

Parameters

- **img_data** – Image data.
- **labels** – Labels (2D array of size [`img_data->height`][`img_data->width`]).
- **RoIs_id** – Array of RoI unique identifiers (useful only if `show_id == 1`).
- **RoIs_xmax** – Array of maximum x coordinates of the bounding box (useful only if `show_id == 1`).

- **RoIs_ymin** – Array of minimum y coordinates of the bounding box (useful only if `show_id == 1`).
- **RoIs_ymax** – Array of maximum y coordinates of the bounding box (useful only if `show_id == 1`).
- **n_RoIs** – Number of connected-components (= number of RoIs) (useful only if `show_id == 1`).
- **show_id** – Boolean to enable display of the label numbers (has no effect if the program has not be linked with the OpenCV library).

Function _kNN_asso_conflicts_write

- Defined in file_c_fmdt_kNN_kNN_io.h

Function Documentation

void **_kNN_asso_conflicts_write**(FILE *f, const float **kNN_data_distances, const uint32_t **kNN_data_nearest, const uint32_t *kNN_data_conflicts, const uint32_t *RoIs0_id, const uint32_t *RoIs0_next_id, const size_t n_RoIs0, const float *RoIs1_dx, const float *RoIs1_dy, const float *RoIs1_error, const uint8_t *RoIs1_is_moving, const size_t n_RoIs1)

Print a table of RoIs association features plus the corresponding RoIs motion features.

Parameters

- **f** – File descriptor (in write mode).
- **kNN_data_distances** – 2D array of euclidean distances.
- **kNN_data_nearest** – 2D array of ranks.
- **kNN_data_conflicts** – 1D array of conflicts. The conflicts are printed only if the FMDT_ENABLE_DEBUG macro is defined.
- **RoIs0_id** – Array of RoI unique identifiers (at $t-1$).
- **RoIs0_next_id** – Array of RoI identifiers at $t-1+1=t$.
- **n_RoIs0** – Number of connected-components (= number of RoIs) (at $t-1$).
- **RoIs1_dx** – Array of x components of the distance between centroids at $t-1$ and t.
- **RoIs1_dy** – Array of y components of the distance between centroids at $t-1$ and t.
- **RoIs1_error** – Array of velocity norms (if `is_moving == 1`) or error (if `is_moving == 0`).
- **RoIs1_is_moving** – Array of booleans that define if the RoI is moving.
- **n_RoIs1** – Number of connected-components (= number of RoIs) (at t).

Function _kNN_match

- Defined in file_c_fmdt_kNN_kNN_compute.h

Function Documentation

void **_kNN_match**(float **data_distances, uint32_t **data_nearest, uint32_t *data_conflicts, const uint32_t *RoIs0_id, const uint32_t *RoIs0_S, const float *RoIs0_x, const float *RoIs0_y, uint32_t *RoIs0_next_id, const size_t n_RoIs0, const uint32_t *RoIs1_id, const uint32_t *RoIs1_S, const float *RoIs1_x, const float *RoIs1_y, uint32_t *RoIs1_prev_id, const size_t n_RoIs1, const int k, const uint32_t max_dist, const float min_ratio_S)

Compute associations between RoIs at $t-1$ and RoIs at t.

Parameters

- **data_distances** – 2D array of euclidean distances.
- **data_nearest** – 2D array of ranks.
- **data_conflicts** – 1D array of conflicts. The conflicts are filled only if the FMDT_ENABLE_DEBUG macro is defined.
- **RoIs0_id** – Array of RoI unique identifiers (at $t-1$).
- **RoIs0_S** – Array of RoI surfaces (at $t-1$).
- **RoIs0_x** – Array of centroids abscissa (at $t-1$).
- **RoIs0_y** – Array of centroids ordinate (at $t-1$).
- **RoIs0_next_id** – Array of RoI identifiers at $t-1+1=t$.
- **n_RoIs0** – Number of connected-components (= number of RoIs) (at $t-1$).
- **RoIs1_id** – Array of RoI unique identifiers (at t).
- **RoIs1_S** – Array of RoI surfaces (at t).
- **RoIs1_x** – Array of centroids abscissa (at t).
- **RoIs1_y** – Array of centroids ordinate (at t).
- **RoIs1_prev_id** – Array of RoI identifiers at $t-1$.
- **n_RoIs1** – Number of connected-components (= number of RoIs) (at t).
- **k** – Number of ranks considered for RoI associations.
- **max_dist** – Maximum distance between 2 RoIs to make the association.
- **min_ratio_S** – Minimum ratio between two RoIs. $r_S = RoI_S^j / RoI_S^i$, if $r_S < r_S^{min}$ then the association is not made.

Function _motion_compute

- Defined in file_c_fmdt_motion_motion_compute.h

Function Documentation

void **_motion_compute**(const float *RoIs0_x, const float *RoIs0_y, const float *RoIs1_x, const float *RoIs1_y, float *RoIs1_dx, float *RoIs1_dy, float *RoIs1_error, const uint32_t *RoIs1_prev_id, uint8_t *RoIs1_is_moving, const size_t n_RoIs1, *motion_t* *motion_est1, *motion_t* *motion_est2)

Compute the global motion estimation and, after global motion compensation, compute the movement of each RoI. In order to compute the motion estimation, the translation vector (Tx, Ty) and the angle of rotation θ must be calculated as follows:

$$\theta = \tan^{-1}\left(\frac{\sum_{i=1}^{N}[(y_i' - \bar{y})(x_i - \bar{x}) - (x_i' - \bar{x})(y_i - \bar{y})]}{\sum_{i=1}^{N}[(x_i' - \bar{x})(x_i - \bar{x}) + (y_i' - \bar{y})(y_i - \bar{y})]}\right),$$

$$T = \begin{bmatrix} T_x \\ T_y \end{bmatrix} = \begin{bmatrix} x' - x \cdot cos(\theta) + y \cdot sin(\theta) \\ y' - x \cdot sin(\theta) - y \cdot cos(\theta) \end{bmatrix},$$

where N is the number of RoIs, (x, y) and (x', y') are the centroids of RoIs at $t-1$ and t, respectively, and

$$\bar{x} = \sum_{i=1}^{N} x_i \quad \bar{y} = \sum_{i=1}^{N} y_i \quad \bar{x}' = \sum_{i=1}^{N} x_i' \quad \bar{y}' = \sum_{i=1}^{N} y_i'.$$

For the first global motion estimation, all the associated RoIs are considered. For the second global motion estimation, only the RoIs considered as "not moving" are considered. To be considered in movement the motion norm of the RoI has to be higher that de motion standard deviation.

Parameters

- **RoIs0_x** – Array of centroids abscissa (at $t-1$).
- **RoIs0_y** – Array of centroids ordinate (at $t-1$).
- **RoIs1_x** – Array of centroids abscissa (at t).
- **RoIs1_y** – Array of centroids abscissa (at t).
- **RoIs1_dx** – Array of x components of the distance between centroids at $t-1$ and t.
- **RoIs1_dy** – Array of y components of the distance between centroids at $t-1$ and t.
- **RoIs1_error** – Array of velocity norms (if `is_moving` == 1) or errors (if `is_moving` == 0).
- **RoIs1_prev_id** – Array of previous corresponding RoI identifiers ($RoI_{t-1} \leftrightarrow RoI_t$).
- **RoIs1_is_moving** – Array of booleans that defines if the RoI is moving (`is_moving` == 1) or not (`is_moving` == 0).
- **n_RoIs1** – Number of connected-components (= number of RoIs) (at t).
- **motion_est1** – First global motion estimation.
- **motion_est2** – Second global motion estimation.

Function _tracking_get_track_time

- Defined in file_c_fmdt_tracking_tracking_struct.h

Function Documentation

size_t **_tracking_get_track_time**(const *RoI_t* track_begin, const *RoI_t* track_end)

Compute the duration of a track.

Parameters

- **track_begin** – First RoI of the track.
- **track_end** – Last RoI of the track.

Returns The elapsed time (in number of frames).

Function _tracking_perform

- Defined in file_c_fmdt_tracking_tracking_compute.h

Function Documentation

void **_tracking_perform**(*tracking_data_t* *tracking_data, const uint32_t *RoIs_id, const uint32_t *RoIs_xmin, const uint32_t *RoIs_xmax, const uint32_t *RoIs_ymin, const uint32_t *RoIs_ymax, const uint32_t *RoIs_S, const float *RoIs_x, const float *RoIs_y, const float *RoIs_error, const uint32_t *RoIs_prev_id, const uint32_t *RoIs_magnitude, const size_t n_RoIs, *vec_BB_t* **BBs, const size_t frame, const *motion_t* *motion_est, const size_t r_extrapol, const float angle_max, const float diff_dev, const int track_all, const size_t fra_star_min, const size_t fra_meteor_min, const size_t fra_meteor_max, const int magnitude, const uint8_t extrapol_order_max, const float min_extrapol_ratio_S)

Create, update and finalize tracks. This function also performs the classification of the tracks.

Parameters

- **tracking_data** – Inner data.
- **RoIs_id** – Array of RoI unique identifiers (at t).
- **RoIs_xmin** – Array of minimum x coordinates of the bounding box (at t).
- **RoIs_xmax** – Array of maximum x coordinates of the bounding box (at t).
- **RoIs_ymin** – Array of minimum y coordinates of the bounding box (at t).
- **RoIs_ymax** – Array of maximum y coordinates of the bounding box (at t).
- **RoIs_S** – Array of RoI surfaces (at t).
- **RoIs_x** – Array of centroids abscissa (at t).
- **RoIs_y** – Array of centroids ordinate (at t).
- **RoIs_error** – Array of velocity norms / errors (at t).
- **RoIs_prev_id** – Array of RoI identifiers at $t - 1$ (at t).
- **RoIs_magnitude** – Array of RoI magnitudes (at t).
- **n_RoIs** – Number of connected-components (= number of RoIs) (at t).

- **BBs** – 2D vector of bounding boxes to be filled. The first dimension represents the frames while the second dimension represents the bounding boxes. BBs can be NULL, if so, the bounding boxes are not saved.
- **frame** – Current frame number.
- **motion_est** – Motion estimation at t.
- **r_extrapol** – Accepted range for extrapolation.
- **angle_max** – Maximum angle that the 3 last positions of a same track can form (if the angle is higher than angle_max then the track is classified as noise).
- **diff_dev** – Multiplication factor in the motion detection criterion. Motion criterion is: $|e_k - \bar{e_t}| > \texttt{diff\_dev} * \sigma_t$, where e_k is the compensation error of the CC/RoI number k, $\bar{e_t}$ the average error of compensation of all CCs of image I_t, and σ_t the standard deviation of the error.
- **track_all** – Boolean that defines if the tracking should track other objects than only meteors.
- **fra_star_min** – Minimum number of CC/RoI associations before creating a star track.
- **fra_meteor_min** – Minimum number of CC/RoI associations before creating a meteor track.
- **fra_meteor_max** – Maximum number of CC/RoI associations after which a meteor track is transformed in a noise track.
- **magnitude** – Boolean that defines if the tracking store the magnitude in its inner data or not.
- **extrapol_order_max** – Maximum number of frames where a lost track is extrapolated (0 means no extrapolation).
- **min_extrapol_ratio_S** – Minimum ratio between two RoIs. $r_S = RoI_S^j / RoI_S^i$, if $r_S < r_S^{min}$ then the association for the extrapolation is not made.

Function args_del

- Defined in file_c_fmdt_args.h

Function Documentation

void **args_del**(int argc, char **argv, int index)

Function args_find

- Defined in file_c_fmdt_args.h

Function Documentation

int **args_find**(int argc, char **argv, const char *arg)

Find if an argument exists in program command line.

Parameters

- **argc** – Number of arguments in `argv` array of arguments.
- **argv** – Array of arguments.
- **arg** – Argument to look for. Note that a list of arguments can be provided: arguments have to be separated by a comma (',') character.

Returns 1 if the argument is found, `0` otherwise.

Function args_find_char

- Defined in file_c_fmdt_args.h

Function Documentation

char ***args_find_char**(int argc, char **argv, const char *arg, char *def)

Find an argument and return its corresponding value as string (array of characters).

Parameters

- **argc** – Number of arguments in `argv` array of arguments.
- **argv** – Array of arguments.
- **arg** – Argument to look for. Note that a list of arguments can be provided: arguments have to be separated by a comma (',') character.
- **def** – Default value if the argument is not found.

Returns Pointer of characters in `argv` corresponding to the argument value if it exists in the command line, `def` pointer otherwise.

Function args_find_float

- Defined in file_c_fmdt_args.h

Function Documentation

float **args_find_float**(int argc, char **argv, const char *arg, float def)

Find an argument and return its corresponding value as a floating-point value.

Parameters

- **argc** – Number of arguments in `argv` array of arguments.
- **argv** – Array of arguments.
- **arg** – Argument to look for. Note that a list of arguments can be provided: arguments have to be separated by a comma (',') character.
- **def** – Default value if the argument is not found.

Returns Value corresponding to the argument if it exists in the command line, `def` value otherwise.

Function args_find_float_max

- Defined in file_c_fmdt_args.h

Function Documentation

float **args_find_float_max**(int argc, char **argv, const char *arg, float def, float max)

Find an argument and return its corresponding value as a floating-point value. This function also tests that the returned value is lower (or equal) than a maximum value. If it is not the case, it prints an error message and exits the program with `-1` value.

Parameters

- **argc** – Number of arguments in `argv` array of arguments.
- **argv** – Array of arguments.
- **arg** – Argument to look for. Note that a list of arguments can be provided: arguments have to be separated by a comma (',') character.
- **def** – Default value if the argument is not found.
- **max** – Maximum accepted value.

Returns Value corresponding to the argument if it exists in the command line, `def` value otherwise.

Function args_find_float_min

- Defined in file_c_fmdt_args.h

Function Documentation

float **args_find_float_min**(int argc, char **argv, const char *arg, float def, float min)

Find an argument and return its corresponding value as a floating-point value. This function also tests that the returned value is higher (or equal) than a minimum value. If it is not the case, it prints an error message and exits the program with `-1` value.

Parameters

- **argc** – Number of arguments in `argv` array of arguments.
- **argv** – Array of arguments.
- **arg** – Argument to look for. Note that a list of arguments can be provided: arguments have to be separated by a comma (',') character.
- **def** – Default value if the argument is not found.
- **min** – Minimum accepted value.

Returns Value corresponding to the argument if it exists in the command line, `def` value otherwise.

Function args_find_float_min_max

- Defined in file_c_fmdt_args.h

Function Documentation

float **args_find_float_min_max**(int argc, char **argv, const char *arg, float def, float min, float max)

Find an argument and return its corresponding value as a floating-point value. This function also tests that the returned value is between the $[min; max]$ range. If it is not the case, it prints an error message and exits the program with `-1` value.

Parameters

- **argc** – Number of arguments in `argv` array of arguments.
- **argv** – Array of arguments.
- **arg** – Argument to look for. Note that a list of arguments can be provided: arguments have to be separated by a comma (',') character.
- **def** – Default value if the argument is not found.
- **min** – Minimum accepted value.
- **max** – Maximum accepted value.

Returns Value corresponding to the argument if it exists in the command line, `def` value otherwise.

Function args_find_int

- Defined in file_c_fmdt_args.h

Function Documentation

int **args_find_int**(int argc, char **argv, const char *arg, int def)

Find an argument and return its corresponding value as an integer value.

Parameters

- **argc** – Number of arguments in `argv` array of arguments.
- **argv** – Array of arguments.
- **arg** – Argument to look for. Note that a list of arguments can be provided: arguments have to be separated by a comma (',') character.
- **def** – Default value if the argument is not found.

Returns Value corresponding to the argument if it exists in the command line, `def` value otherwise.

Function args_find_int_max

- Defined in file_c_fmdt_args.h

Function Documentation

int **args_find_int_max**(int argc, char **argv, const char *arg, int def, int max)

Find an argument and return its corresponding value as an integer value. This function also tests that the returned value is lower (or equal) than a maximum value. If it is not the case, it prints an error message and exits the program with `-1` value.

Parameters

- **argc** – Number of arguments in `argv` array of arguments.
- **argv** – Array of arguments.
- **arg** – Argument to look for. Note that a list of arguments can be provided: arguments have to be separated by a comma (',') character.
- **def** – Default value if the argument is not found.
- **max** – Maximum accepted value.

Returns Value corresponding to the argument if it exists in the command line, `def` value otherwise.

Function args_find_int_min

- Defined in file_c_fmdt_args.h

Function Documentation

int **args_find_int_min**(int argc, char **argv, const char *arg, int def, int min)

Find an argument and return its corresponding value as an integer value. This function also tests that the returned value is higher (or equal) than a minimum value. If it is not the case, it prints an error message and exits the program with `-1` value.

Parameters

- **argc** – Number of arguments in `argv` array of arguments.
- **argv** – Array of arguments.
- **arg** – Argument to look for. Note that a list of arguments can be provided: arguments have to be separated by a comma (',') character.
- **def** – Default value if the argument is not found.
- **min** – Minimum accepted value.

Returns Value corresponding to the argument if it exists in the command line, `def` value otherwise.

Function args_find_int_min_max

- Defined in file_c_fmdt_args.h

Function Documentation

int **args_find_int_min_max**(int argc, char **argv, const char *arg, int def, int min, int max)

Find an argument and return its corresponding value as an integer value. This function also tests that the returned value is between the $[min; max]$ range. If it is not the case, it prints an error message and exits the program with `-1` value.

Parameters

- **argc** – Number of arguments in `argv` array of arguments.
- **argv** – Array of arguments.
- **arg** – Argument to look for. Note that a list of arguments can be provided: arguments have to be separated by a comma (',') character.
- **def** – Default value if the argument is not found.
- **min** – Minimum accepted value.
- **max** – Maximum accepted value.

Returns Value corresponding to the argument if it exists in the command line, `def` value otherwise.

Function CCL_LSL_alloc_data

- Defined in file_c_fmdt_CCL_CCL_compute.h

Function Documentation

CCL_data_t ***CCL_LSL_alloc_data**(int i0, int i1, int j0, int j1)

Allocation of inner data required to perform Light Speed Labeling (LSL).

Parameters

- **i0** – The first y index in the image (included).
- **i1** – The last y index in the image (included).
- **j0** – The first x index in the image (included).
- **j1** – The last x index in the image (included).

Returns The allocated and initialized data.

Function CCL_LSL_apply

- Defined in file_c_fmdt_CCL_CCL_compute.h

Function Documentation

uint32_t **CCL_LSL_apply**(*CCL_data_t* *CCL_data, const uint8_t **img, uint32_t **labels)

Compute the Light Speed Labeling (LSL) algorithm.

Parameters

- **CCL_data** – Inner data required to perform the LSL.
- **img** – Input binary image (2D array $[i1 - i0 + 1][j1 - j0 + 1]$, $\{0, 1\}$ has to be coded as $\{0, 255\}$).
- **labels** – Output labels (2D array $[i1 - i0 + 1][j1 - j0 + 1]$. The labels are in $[1; 2^{32} - 1]$ and 0 value means no label).

Returns Number of labels.

Function CCL_LSL_free_data

- Defined in file_c_fmdt_CCL_CCL_compute.h

Function Documentation

void **CCL_LSL_free_data**(*CCL_data_t* *CCL_data)

Free the inner data.

Parameters **CCL_data** – Inner data.

Function CCL_LSL_init_data

- Defined in file_c_fmdt_CCL_CCL_compute.h

Function Documentation

void **CCL_LSL_init_data**(*CCL_data_t* *CCL_data)

Initialization of the CCL inner data. Set all zeros.

Parameters **CCL_data** – Pointer of inner CCL data.

Function features_alloc_RoIs

- Defined in file_c_fmdt_features_features_compute.h

Function Documentation

RoIs_t *`features_alloc_RoIs`(const uint8_t enable_magnitude, const uint8_t enable_sat_count, const size_t max_size)

Allocation of all the features.

Parameters

- **`enable_magnitude`** – Boolean to allocate the buffer of magnitudes.
- **`enable_sat_count`** – Boolean to allocate the buffer of saturation counters.
- **`max_size`** – Maximum capacity of each *feature* field (= maximum number of elements in the arrays).

Returns Pointer of allocated RoIs.

Function features_alloc_RoIs_asso

- Defined in file_c_fmdt_features_features_compute.h

Function Documentation

RoIs_asso_t *`features_alloc_RoIs_asso`(const size_t max_size, uint32_t *RoIs_id)

Allocation of the association features.

Parameters

- **`max_size`** – Maximum capacity of each *feature* field (= maximum number of elements in the arrays).
- **`RoIs_id`** – Pointer of `max_size` elements to use for the `id` field. If set to NULL, the `id` field is allocated.

Returns Pointer of allocated RoIs.

Function features_alloc_RoIs_basic

- Defined in file_c_fmdt_features_features_compute.h

Function Documentation

RoIs_basic_t *`features_alloc_RoIs_basic`(const size_t max_size, uint32_t *RoIs_id)

Allocation of the basic features.

Parameters

- **`max_size`** – Maximum capacity of each *feature* field (= maximum number of elements in the arrays).

- **RoIs_id** – Pointer of `max_size` elements to use for the `id` field. If set to NULL, the `id` field is allocated.

Returns Pointer of allocated RoIs.

Function features_alloc_RoIs_misc

- Defined in file_c_fmdt_features_features_compute.h

Function Documentation

RoIs_misc_t ***features_alloc_RoIs_misc**(const uint8_t enable_magnitude, const uint8_t enable_sat_count, const size_t max_size, uint32_t *RoIs_id)

Allocation of the miscellaneous features.

Parameters

- **max_size** – Maximum capacity of each *feature* field (= maximum number of elements in the arrays).
- **enable_magnitude** – Boolean to allocate the buffer of magnitudes.
- **enable_sat_count** – Boolean to allocate the buffer of saturation counters.
- **RoIs_id** – Pointer of `max_size` elements to use for the `id` field. If set to NULL, the `id` field is allocated.

Returns Pointer of allocated RoIs.

Function features_alloc_RoIs_motion

- Defined in file_c_fmdt_features_features_compute.h

Function Documentation

RoIs_motion_t ***features_alloc_RoIs_motion**(const size_t max_size, uint32_t *RoIs_id)

Allocation of the motion features.

Parameters

- **max_size** – Maximum capacity of each *feature* field (= maximum number of elements in the arrays).
- **RoIs_id** – Pointer of `max_size` elements to use for the `id` field. If set to NULL, the `id` field is allocated.

Returns Pointer of allocated RoIs.

Function features_compute_magnitude

- Defined in file_c_fmdt_features_features_compute.h

Function Documentation

void **features_compute_magnitude**(const uint8_t **img, const uint32_t img_width, const uint32_t img_height, const uint32_t **labels, const *RoIs_basic_t* *RoIs_basic, *RoIs_misc_t* *RoIs_misc)

See also:

_features_compute_magnitude for the explanations about the nature of the processing.

See also:

RoIs_basic_t for more explanations about the basic features.

See also:

RoIs_misc_t for more explanations about the miscellaneous features.

Parameters

- **img** – Image in grayscale ([img_height][img_width], the values of the pixel range are $[0; 255]$).
- **img_width** – Image width.
- **img_height** – Image height.
- **labels** – 2D array of labels ([img_height][img_width]).
- **RoIs_basic** – Basic features.
- **RoIs_misc** – Miscellaneous features (including the magnitudes).

Function features_extract

- Defined in file_c_fmdt_features_features_compute.h

Function Documentation

void **features_extract**(const uint32_t **labels, const int i0, const int i1, const int j0, const int j1, const size_t n_RoIs, *RoIs_basic_t* *RoIs_basic)

See also:

_features_extract for the explanations about the nature of the processing.

See also:

RoIs_basic_t for more explanations about the features.

Parameters

- **labels** – Input 2D array of labels ($[i1 - i0 + 1][j1 - j0 + 1]$).
- **i0** – First y index in the labels (included).

- **i1** – Last y index in the labels (included).
- **j0** – First x index in the labels (included).
- **j1** – Last x index in the labels (included).
- **n_RoIs** – Number of connected-components (= number of RoIs) in the 2D array of `labels`.
- **RoIs_basic** – Basic features.

Function features_free_RoIs

- Defined in file_c_fmdt_features_features_compute.h

Function Documentation

void **features_free_RoIs**(*RoIs_t* *RoIs)

Free the features.

Parameters **RoIs** – Pointer of RoIs.

Function features_free_RoIs_asso

- Defined in file_c_fmdt_features_features_compute.h

Function Documentation

void **features_free_RoIs_asso**(*RoIs_asso_t* *RoIs_asso, const uint8_t free_id)

Free the features.

Parameters

- **RoIs_asso** – Pointer of RoIs.
- **free_id** – Boolean to free or not the `id` field.

Function features_free_RoIs_basic

- Defined in file_c_fmdt_features_features_compute.h

Function Documentation

void **features_free_RoIs_basic**(*RoIs_basic_t* *RoIs_basic, const uint8_t free_id)

Free the features.

Parameters

- **RoIs_basic** – Pointer of RoIs.
- **free_id** – Boolean to free or not the `id` field.

Function features_free_RoIs_misc

- Defined in file_c_fmdt_features_features_compute.h

Function Documentation

void **features_free_RoIs_misc**(*RoIs_misc_t* *RoIs_misc, const uint8_t free_id)

Free the features.

Parameters

- **RoIs_misc** – Pointer of RoIs.
- **free_id** – Boolean to free or not the `id` field.

Function features_free_RoIs_motion

- Defined in file_c_fmdt_features_features_compute.h

Function Documentation

void **features_free_RoIs_motion**(*RoIs_motion_t* *RoIs_motion, const uint8_t free_id)

Free the features.

Parameters

- **RoIs_motion** – Pointer of RoIs.
- **free_id** – Boolean to free or not the `id` field.

Function features_init_RoIs

- Defined in file_c_fmdt_features_features_compute.h

Function Documentation

void **features_init_RoIs**(*RoIs_t* *RoIs)

Initialization of the features. Set all zeros.

Parameters **RoIs** – Pointer of RoIs.

Function features_init_RoIs_asso

- Defined in file_c_fmdt_features_features_compute.h

Function Documentation

void **features_init_RoIs_asso**(*RoIs_asso_t* *RoIs_asso, const uint8_t init_id)

Initialization of the association features. Set all zeros.

Parameters

- **RoIs_asso** – Pointer of RoIs.
- **init_id** – Boolean to initialize or not the `id` field.

Function features_init_RoIs_basic

- Defined in file_c_fmdt_features_features_compute.h

Function Documentation

void **features_init_RoIs_basic**(*RoIs_basic_t* *RoIs_basic, const uint8_t init_id)

Initialization of the basic features. Set all zeros.

Parameters

- **RoIs_basic** – Pointer of RoIs.
- **init_id** – Boolean to initialize or not the `id` field.

Function features_init_RoIs_misc

- Defined in file_c_fmdt_features_features_compute.h

Function Documentation

void **features_init_RoIs_misc**(*RoIs_misc_t* *RoIs_misc, const uint8_t init_id)

Initialization of the miscellaneous features. Set all zeros.

Parameters

- **RoIs_misc** – Pointer of RoIs.
- **init_id** – Boolean to initialize or not the `id` field.

Function features_init_RoIs_motion

- Defined in file_c_fmdt_features_features_compute.h

Function Documentation

void **features_init_RoIs_motion**(*RoIs_motion_t* *RoIs_motion, const uint8_t init_id)

Initialization of the motion features. Set all zeros.

Parameters

- **RoIs_motion** – Pointer of RoIs.
- **init_id** – Boolean to initialize or not the `id` field.

Function features_merge_CCL_HI_v2

- Defined in file_c_fmdt_features_features_compute.h

Function Documentation

void **features_merge_CCL_HI_v2**(const uint32_t **in_labels, const uint8_t **img_HI, uint32_t **out_labels, const int i0, const int i1, const int j0, const int j1, *RoIs_basic_t* *RoIs_basic, const uint32_t S_min, const uint32_t S_max)

See also:

_features_merge_CCL_HI_v2 for the explanations about the nature of the processing.

See also:

RoIs_basic_t for more explanations about the features.

Parameters

- **in_labels** – Input 2D array of labels ($[i1 - i0 + 1][j1 - j0 + 1]$).
- **img_HI** – Binary image (2D array $[i1 - i0 + 1][j1 - j0 + 1]$ $\{0, 1\}$ has to be coded as $\{0, 255\}$). This image results from a threshold filter on the original image. This threshold filter should be higher than the first one used to compute the initial labels (`in_labels`).
- **out_labels** – Output 2D array of labels ($[i1 - i0 + 1][j1 - j0 + 1]$).
- **i0** – First y index in the labels (included).
- **i1** – Last y index in the labels (included).
- **j0** – First x index in the labels (included).
- **j1** – Last x index in the labels (included).
- **RoIs_basic** – Features.
- **S_min** – Minimum morphological threshold.
- **S_max** – Maximum morphological threshold.

Function features_RoIs0_RoIs1_write

- Defined in file_c_fmdt_features_features_io.h

Function Documentation

void **features_RoIs0_RoIs1_write**(FILE *f, const int prev_frame, const int cur_frame, const *RoIs_basic_t* *RoIs0_basic, const *RoIs_misc_t* *RoIs0_misc, const *RoIs_basic_t* *RoIs1_basic, const *RoIs_misc_t* *RoIs1_misc, const *vec_track_t* tracks)

Print two tables of RoIs, one at $t-1$ and one at t.

See also:

_features_RoIs0_RoIs1_write for the explanations about the nature of the processing.

See also:

RoIs_basic_t for more explanations about the features.

See also:

RoIs_misc_t for more explanations about the features.

Parameters

- **f** – File descriptor (in write mode).
- **prev_frame** – Frame id corresponding to the RoIs at $t-1$.
- **cur_frame** – Frame id corresponding to the RoIs at t.
- **RoIs0_basic** – Basic features (at $t-1$).
- **RoIs0_misc** – Miscellaneous features (at $t-1$).
- **RoIs1_basic** – Basic features (at t).
- **RoIs1_misc** – Miscellaneous features (at t)..
- **tracks** – Vector of tracks. It enables to match RoIs with corresponding track in the table of RoIs.

Function features_RoIs_write

- Defined in file_c_fmdt_features_features_io.h

Function Documentation

void **features_RoIs_write**(FILE *f, const int frame, const *RoIs_basic_t* *RoIs_basic, const *RoIs_misc_t* *RoIs_misc, const *vec_track_t* tracks, const unsigned age)

See also:

_features_RoIs_write for the explanations about the nature of the processing.

See also:

RoIs_basic_t for more explanations about the features.

See also:

RoIs_misc_t for more explanations about the features.

Parameters

- **f** – File descriptor (write mode).
- **frame** – Frame id corresponding to the RoIs.
- **RoIs_basic** – Basic features.
- **RoIs_misc** – Miscellaneous features.
- **tracks** – Vector of tracks. It enables to match RoIs with corresponding track in the table of RoIs.
- **age** – 0 if `frame` is the current frame, 1 if `frame` is the $t-1$ frame. This is mandatory to find the corresponding track (if any).

Function features_shrink

- Defined in file_c_fmdt_features_features_compute.h

Function Documentation

void **features_shrink**(const *RoIs_basic_t* *RoIs_basic_src, *RoIs_basic_t* *RoIs_basic_dst)

See also:

_features_shrink for the explanations about the nature of the processing.

See also:

RoIs_basic_t for more explanations about the features.

Parameters

- **RoIs_basic_src** – Source features.
- **RoIs_basic_dst** – Destination features.

Function image_color_alloc

- Defined in file_c_fmdt_image_image_compute.h

Function Documentation

img_data_t ***image_color_alloc**(const size_t img_width, const size_t img_height)

Allocate color image data.

Parameters

- **img_width** – Image width.
- **img_height** – Image height.

Returns Pointer of image data.

Function image_color_draw_BBs

- Defined in file_c_fmdt_image_image_compute.h

Function Documentation

void **image_color_draw_BBs**(*img_data_t* *img_data, const uint8_t **img, const *BB_t* *BBs, const enum *color_e* *BBs_color, const size_t n_BBs, const uint8_t show_id, const uint8_t is_gt)

Draw bounding boxes (BBs) on a color image. If the program is linked with the OpenCV library, then the `show_id` boolean can be used to draw the ids corresponding to each BB on the color image. Moreover, if the program is linked with OpenCV, this routine add the legend on the top left corner.

Parameters

- **img_data** – Image data.
- **img** – 2D grayscale image (2D array of size [`img_data->height`][`img_data->width`]). This image will be copied in `img_data`.
- **BBs** – List of bounding boxes.
- **BBs_color** – List of colors associated to the bounding boxes.
- **n_BBs** – Number of bounding boxes to draw.
- **show_id** – Boolean to enable display of the BB ids (has no effect if the program has not be linked with the OpenCV).
- **is_gt** – Boolean to draw the ground truth legend (has no effect is the program has not been linked with OpenCV).

Function image_color_free

- Defined in file_c_fmdt_image_image_compute.h

Function Documentation

void **image_color_free**(*img_data_t* *img_data)

Deallocate color image data.

Parameters **img_data** – Image data.

Function image_color_get_pixels

- Defined in file_c_fmdt_image_image_compute.h

Function Documentation

rgb8_t ***image_color_get_pixels**(*img_data_t* *img_data)

Return a pixels array of the color image.

Parameters **img_data** – Image data.

Function image_color_get_pixels_2d

- Defined in file_c_fmdt_image_image_compute.h

Function Documentation

rgb8_t ****image_color_get_pixels_2d**(*img_data_t* *img_data)

Return a 2D pixels array of the color image.

Parameters **img_data** – Image data.

Function image_get_color

- Defined in file_c_fmdt_image_image_compute.h

Function Documentation

rgb8_t **image_get_color**(enum *color_e* color)

From a given color, returns the corresponding RBG representation.

Parameters **color** – Color enum value.

Returns RGB struct.

Function image_gs_alloc

- Defined in file_c_fmdt_image_image_compute.h

Function Documentation

img_data_t ***image_gs_alloc**(const size_t img_width, const size_t img_height)

Allocate grayscale image data.

Parameters

- **img_width** – Image width.
- **img_height** – Image height.

Returns Pointer of image data.

Function image_gs_draw_labels

- Defined in file_c_fmdt_image_image_compute.h

Function Documentation

void **image_gs_draw_labels**(*img_data_t* *img_data, const uint32_t **labels, const *RoIs_basic_t* *RoIs_basic, const uint8_t show_id)

See also:

_image_gs_draw_labels for the explanations about the nature of the processing.

Parameters

- **img_data** – Image data.
- **labels** – Labels (2D array of size [img_data->height][img_data->width]).
- **RoIs_basic** – Basic features (useful only if show_id == 1).
- **show_id** – Boolean to enable display of the label numbers (has no effect if the program has not be linked with the OpenCV library).

Function image_gs_free

- Defined in file_c_fmdt_image_image_compute.h

Function Documentation

void **image_gs_free**(*img_data_t* *img_data)

Deallocate grayscale image data.

Parameters **img_data** – Image data.

Function image_gs_get_pixels

- Defined in file_c_fmdt_image_image_compute.h

Function Documentation

uint8_t ***image_gs_get_pixels**(*img_data_t* *img_data)

Return a pixels array of the grayscale image.

Parameters **img_data** – Image data.

Function image_gs_get_pixels_2d

- Defined in file_c_fmdt_image_image_compute.h

Function Documentation

uint8_t **__image_gs_get_pixels_2d__(*img_data_t* *img_data)

Return a 2D pixels array of the grayscale image.

Parameters **img_data** – Image data.

Function image_save_frame_quad

- Defined in file_c_fmdt_image_image_io.h

Function Documentation

void __image_save_frame_quad__(const char *filename, uint8_t **I0, uint8_t **I1, uint32_t **I2, uint32_t **I3, int nbLabel, *RoIs_t* *stats, int i0, int i1, int j0, int j1)

Function image_save_frame_quad_hysteresis

- Defined in file_c_fmdt_image_image_io.h

Function Documentation

void __image_save_frame_quad_hysteresis__(const char *filename, uint8_t **I0, uint32_t **SH, uint32_t **SB, uint32_t **Y, int i0, int i1, int j0, int j1)

Function image_save_frame_threshold

- Defined in file_c_fmdt_image_image_io.h

Function Documentation

void __image_save_frame_threshold__(const char *filename, uint8_t **I0, uint8_t **I1, int i0, int i1, int j0, int j1)

Function image_write_PNM_row

- Defined in file_c_fmdt_image_image_io.h

Function Documentation

void **image_write_PNM_row**(const uint8_t *line, const int width, FILE *file)

Function kNN_alloc_data

- Defined in file_c_fmdt_kNN_kNN_compute.h

Function Documentation

kNN_data_t ***kNN_alloc_data**(const size_t max_size)

Allocation of inner kNN data. The `conflicts` field is allocated only if the `FMDT_ENABLE_DEBUG` macro is defined.

Parameters **max_size** – Maximum number of RoIs that can considered for associations.

Returns Pointer of kNN data.

Function kNN_asso_conflicts_write

- Defined in file_c_fmdt_kNN_kNN_io.h

Function Documentation

void **kNN_asso_conflicts_write**(FILE *f, const *kNN_data_t* *kNN_data, const *RoIs_asso_t* *RoIs0_asso, const *RoIs_asso_t* *RoIs1_asso, const *RoIs_motion_t* *RoIs1_motion)

See also:

_kNN_asso_conflicts_write for the explanations about the nature of the processing.

Parameters

- **f** – File descriptor (in write mode).
- **kNN_data** – Inner kNN data.
- **RoIs0_asso** – Association features at $t-1$.
- **RoIs1_asso** – Association features at t.
- **RoIs1_motion** – Motion features at t.

Function kNN_free_data

- Defined in file_c_fmdt_kNN_kNN_compute.h

Function Documentation

void **kNN_free_data**(*kNN_data_t* *kNN_data)

Deallocation of inner kNN data.

Parameters **kNN_data** – A pointer of kNN inner data.

Function kNN_init_data

- Defined in file_c_fmdt_kNN_kNN_compute.h

Function Documentation

void **kNN_init_data**(*kNN_data_t* *kNN_data)

Initialization of the kNN inner data. Set all zeros.

Parameters **kNN_data** – Pointer of inner kNN data.

Function kNN_match

- Defined in file_c_fmdt_kNN_kNN_compute.h

Function Documentation

void **kNN_match**(*kNN_data_t* *kNN_data, const *RoIs_basic_t* *RoIs0_basic, const *RoIs_basic_t* *RoIs1_basic, *RoIs_asso_t* *RoIs0_asso, *RoIs_asso_t* *RoIs1_asso, const int k, const uint32_t max_dist, const float min_ratio_S)

See also:

_kNN_match for the explanations about the nature of the processing.

Parameters

- **kNN_data** – Inner kNN data.
- **RoIs0_basic** – Basic features (at $t-1$).
- **RoIs1_basic** – Basic features (at t).
- **RoIs0_asso** – Association features (at $t-1$).
- **RoIs1_asso** – Association features (at t).
- **k** – Number of ranks considered for RoI associations.
- **max_dist** – Maximum distance between 2 RoIs to make the association.
- **min_ratio_S** – Minimum ratio between two RoIs. $r_S = RoI_S^j / RoI_S^i$, if $r_S < r_S^{min}$ then the association is not made.

Function motion_compute

- Defined in file_c_fmdt_motion_motion_compute.h

Function Documentation

void **motion_compute**(const *RoIs_basic_t* *RoIs0_basic, const *RoIs_basic_t* *RoIs1_basic, const *RoIs_asso_t* *RoIs1_asso, *RoIs_motion_t* *RoIs1_motion, *motion_t* *motion_est1, *motion_t* *motion_est2)

See also:

_motion_compute for the explanations about the nature of the processing.

Parameters

- **RoIs0_basic** – Basic features (at $t-1$).
- **RoIs1_basic** – Basic features (at t).
- **RoIs1_asso** – Association features (at t).
- **RoIs1_motion** – Motion features (at t).
- **motion_est1** – First global motion estimation.
- **motion_est2** – Second global motion estimation.

Function motion_write

- Defined in file_c_fmdt_motion_motion_io.h

Function Documentation

void **motion_write**(FILE *f, const *motion_t* *motion_est1, const *motion_t* *motion_est2)

Print a table of global motion estimation.

Parameters

- **f** – File descriptor (in write mode).
- **motion_est1** – First global motion estimation.
- **motion_est2** – Last global motion estimation.

Function threshold

- Defined in file_c_fmdt_threshold_threshold_compute.h

Function Documentation

void **threshold**(const uint8_t **img_in, uint8_t **img_out, const int i0, const int i1, const int j0, const int j1, const uint8_t threshold)

Convert an input image (I_{in}) in grayscale levels into a binary image (I_{out}) depending on a greyscale threshold (T). If $I^i_{in} \geq T$ then $I^i_{out} = 255$, else $I^i_{out} = 0$.

Parameters

- **img_in** – Input greyscale image (2D array $[i1 - i0 + 1][j1 - j0 + 1]$, $\{0, 1\}$).
- **img_out** – Output binary image (2D array $[i1 - i0 + 1][j1 - j0 + 1]$, $\{0, 1\}$, $\{0, 1\}$ is coded as $\{0, 255\}$).
- **i0** – First y index in the image (included).
- **i1** – Last y index in the image (included).
- **j0** – First x index in the image (included).
- **j1** – Last x index in the image (included).
- **threshold** – Value that define if the pixel is kept in the output binary image or not.

Function tools_convert_ui8matrix_ui32matrix

- Defined in file_c_fmdt_tools.h

Function Documentation

void **tools_convert_ui8matrix_ui32matrix**(const uint8_t **X, const int nrl, const int nrh, const int ncl, const int nch, uint32_t **Y)

Convert a 8-bit 2D array in a 32-bit 2D array.

Parameters

- **X** – Input 8-bit matrix (2D array $[i1 - i0 + 1][j1 - j0 + 1]$).
- **nrl** – First y index in the 2D array (included).
- **nrh** – Last y index in the 2D array (included).
- **ncl** – First x index in the 2D array (included).
- **nch** – Last x index in the 2D array (included).
- **Y** – Output 32-bit matrix (2D array $[i1 - i0 + 1][j1 - j0 + 1]$).

Function tools_copy_ui8matrix_ui8matrix

- Defined in file_c_fmdt_tools.h

Function Documentation

void **tools_copy_ui8matrix_ui8matrix**(const uint8_t **X, const int i0, const int i1, const int j0, const int j1, uint8_t **Y)

Copy a 2D array.

Parameters

- **X** – Input matrix (2D array $[i1 - i0 + 1][j1 - j0 + 1]$).
- **i0** – First y index in the 2D array (included).
- **i1** – Last y index in the 2D array (included).
- **j0** – First x index in the 2D array (included).
- **j1** – Last x index in the 2D array (included).
- **Y** – Output matrix (2D array $[i1 - i0 + 1][j1 - j0 + 1]$).

Function tools_create_folder

- Defined in file_c_fmdt_tools.h

Function Documentation

void **tools_create_folder**(const char *folder_path)

System function to create a folder.

Parameters **folder_path** – Path to the folder to create.

Function tools_is_dir

- Defined in file_c_fmdt_tools.h

Function Documentation

int **tools_is_dir**(const char *path)

System function to check if a path is a directory.

Parameters **path** – Path.

Returns 1 if the given path is a folder, 0 otherwise.

Function tools_linear_2d_nrc_f32matrix

- Defined in file_c_fmdt_tools.h

Function Documentation

void **tools_linear_2d_nrc_f32matrix**(const float *X, const int i0, const int i1, const int j0, const int j1, const float **Y)

Convert a 1D (linear) array into a 2D array (32-bit float).

Parameters

- **X** – Input 1D array (1D array $[(i1 - i0 + 1) \times (j1 - j0 + 1)]$).
- **i0** – First y index in the 2D array (included).
- **i1** – Last y index in the 2D array (included).
- **j0** – First x index in the 2D array (included).
- **j1** – Last x index in the 2D array (included).
- **Y** – Output matrix (2D array $[i1 - i0 + 1][j1 - j0 + 1]$).

Function tools_linear_2d_nrc_rgb8matrix

- Defined in file_c_fmdt_tools.h

Function Documentation

void **tools_linear_2d_nrc_rgb8matrix**(const *rgb8_t* *X, const int i0, const int i1, const int j0, const int j1, const *rgb8_t* **Y)

Convert a 1D (linear) array into a 2D array (24-bit RGB).

Parameters

- **X** – Input 1D array (1D array $[(i1 - i0 + 1) \times (j1 - j0 + 1)]$).
- **i0** – First y index in the 2D array (included).
- **i1** – Last y index in the 2D array (included).
- **j0** – First x index in the 2D array (included).
- **j1** – Last x index in the 2D array (included).
- **Y** – Output matrix (2D array $[i1 - i0 + 1][j1 - j0 + 1]$).

Function tools_linear_2d_nrc_ui32matrix

- Defined in file_c_fmdt_tools.h

Function Documentation

void **tools_linear_2d_nrc_ui32matrix**(const uint32_t *X, const int i0, const int i1, const int j0, const int j1, const uint32_t **Y)

Convert a 1D (linear) array into a 2D array (32-bit integers).

Parameters

- **X** – Input 1D array (1D array $[(i1 - i0 + 1) \times (j1 - j0 + 1)]$).
- **i0** – First y index in the 2D array (included).
- **i1** – Last y index in the 2D array (included).
- **j0** – First x index in the 2D array (included).
- **j1** – Last x index in the 2D array (included).
- **Y** – Output matrix (2D array $[i1 - i0 + 1][j1 - j0 + 1]$).

Function tools_linear_2d_nrc_ui8matrix

- Defined in file_c_fmdt_tools.h

Function Documentation

void **tools_linear_2d_nrc_ui8matrix**(const uint8_t *X, const int i0, const int i1, const int j0, const int j1, const uint8_t **Y)

Convert a 1D (linear) array into a 2D array (8-bit integers).

Parameters

- **X** – Input 1D array (1D array $[(i1 - i0 + 1) \times (j1 - j0 + 1)]$).
- **i0** – First y index in the 2D array (included).
- **i1** – Last y index in the 2D array (included).
- **j0** – First x index in the 2D array (included).
- **j1** – Last x index in the 2D array (included).
- **Y** – Output matrix (2D array $[i1 - i0 + 1][j1 - j0 + 1]$).

Function tracking_alloc_data

- Defined in file_c_fmdt_tracking_tracking_compute.h

Function Documentation

tracking_data_t ***tracking_alloc_data**(const size_t max_history_size, const size_t max_RoIs_size)

Allocation of inner data required to perform the tracking.

Parameters

- **max_history_size** – The maximum size of the history window (number of frames memorized in the history of RoIs).
- **max_RoIs_size** – The maximum number of RoIs per frame.

Returns The allocated data.

Function tracking_BBs_write

- Defined in file_c_fmdt_tracking_tracking_io.h

Function Documentation

void **tracking_BBs_write**(FILE *f, const *vec_BB_t* *BBs, const *vec_track_t* tracks)

Print list of bounding boxes. Each line corresponds to a bounding boxes.

Parameters

- **f** – File descriptor (in write mode).
- **BBs** – A 2D vector of bounding boxes (first dimension is the frames, second dimensions is the bounding boxes).
- **tracks** – A vector of tracks.

Function tracking_count_objects

- Defined in file_c_fmdt_tracking_tracking_struct.h

Function Documentation

size_t **tracking_count_objects**(const *vec_track_t* tracks, unsigned *n_stars, unsigned *n_meteors, unsigned *n_noise)

Counts the number of tracks in a vector of tracks.

Parameters

- **tracks** – A vector of tracks.
- **n_stars** – Write the number of tracks that have been classified as star.
- **n_meteors** – Write the number of tracks that have been classified as meteor.
- **n_noise** – Write the number of tracks that have been classified as noise.

Returns The real number of tracks (may be less than the `tracks` vector size).

Function tracking_free_data

- Defined in file_c_fmdt_tracking_tracking_compute.h

Function Documentation

void **tracking_free_data**(*tracking_data_t* *tracking_data)

Free the tracking inner data.

Parameters **tracking_data** – Pointer of tracking inner data.

Function tracking_get_track_time

- Defined in file_c_fmdt_tracking_tracking_struct.h

Function Documentation

size_t **tracking_get_track_time**(const *vec_track_t* tracks, const size_t t)

Compute the duration of a track.

Parameters

- **tracks** – A vector of tracks.
- **t** – The position of one track in the tracks array.

Returns The elapsed time (in number of frames).

Function tracking_init_data

- Defined in file_c_fmdt_tracking_tracking_compute.h

Function Documentation

void **tracking_init_data**(*tracking_data_t* *tracking_data)

Zero initialization of inner data required to perform the tracking.

Parameters **tracking_data** – Pointer of tracking inner data.

Function tracking_init_global_data

- Defined in file_c_fmdt_tracking_tracking_global.h

Function Documentation

void **tracking_init_global_data**()

Initialize global LUTs (`g_obj_to_color`, `g_obj_to_string`, `g_obj_to_string_with_spaces`, `g_change_state_to_string` and `g_change_state_to_string_with_spaces`).

Function tracking_parse_tracks

- Defined in file_c_fmdt_tracking_tracking_io.h

Function Documentation

void **tracking_parse_tracks**(const char *filename, *vec_track_t* *tracks)

From a given path, parse the corresponding file and fill a vector of tracks.

Parameters

- **filename** – The path of the file to parse.
- **tracks** – A vector of tracks.

Function tracking_perform

- Defined in file_c_fmdt_tracking_tracking_compute.h

Function Documentation

void **tracking_perform**(*tracking_data_t* *tracking_data, const *RoIs_t* *RoIs, *vec_BB_t* **BBs, size_t frame, const *motion_t* *motion_est, const size_t r_extrapol, const float angle_max, const float diff_dev, const int track_all, const size_t fra_star_min, const size_t fra_meteor_min, const size_t fra_meteor_max, const int magnitude, const uint8_t extrapol_order_max, const float min_extrapol_ratio_S)

See also:

_tracking_perform for the explanations about the nature of the processing.

Parameters

- **tracking_data** – Inner data.
- **RoIs** – RoIs features (at t).
- **BBs** – 2D vector of bounding boxes to be filled. The first dimension represents the frames while the second dimension represents the bounding boxes. `BBs` can be NULL, if so, the bounding boxes are not saved.
- **frame** – Current frame number.
- **motion_est** – Motion estimation at t.
- **r_extrapol** – Accepted range for extrapolation.
- **angle_max** – Maximum angle that the 3 last positions of a same track can form (if the angle is higher than `angle_max` then the track is classified as noise).

- **diff_dev** – Multiplication factor in the motion detection criterion. Motion criterion is: $|e_k - \bar{e_t}| > \texttt{diff\_dev} * \sigma_t$, where e_k is the compensation error of the CC/RoI number k, $\bar{e_t}$ the average error of compensation of all CCs of image I_t, and σ_t the standard deviation of the error.
- **track_all** – Boolean that defines if the tracking should track other objects than only meteors.
- **fra_star_min** – Minimum number of CC/RoI associations before creating a star track.
- **fra_meteor_min** – Minimum number of CC/RoI associations before creating a meteor track.
- **fra_meteor_max** – Maximum number of CC/RoI associations after which a meteor track is transformed in a noise track.
- **magnitude** – Boolean that defines if the tracking store the magnitude in its inner data or not.
- **extrapol_order_max** – Maximum number of frames where a lost track is extrapolated (0 means no extrapolation).
- **min_extrapol_ratio_S** – Minimum ratio between two RoIs. $r_S = RoI_S^j / RoI_S^i$, if $r_S < r_S^{min}$ then the association for the extrapolation is not made.

Function tracking_string_to_obj_type

- Defined in file_c_fmdt_tracking_tracking_global.h

Function Documentation

enum *obj_e* **tracking_string_to_obj_type**(const char *string)

Return object type from its corresponding string.

Parameters **string** – A string.

Returns obj_e The right object type.

Function tracking_tracks_magnitudes_write

- Defined in file_c_fmdt_tracking_tracking_io.h

Function Documentation

void **tracking_tracks_magnitudes_write**(FILE *f, const *vec_track_t* tracks)

Print a list of magnitudes per track. Each line corresponds to a track.

Parameters

- **f** – File descriptor (in write mode).
- **tracks** – A vector of tracks.

Function tracking_tracks_write

- Defined in file_c_fmdt_tracking_tracking_io.h

Function Documentation

void **tracking_tracks_write**(FILE *f, const *vec_track_t* tracks)

Print a table of tracks (dedicated to the terminal).

Parameters

- **f** – File descriptor (in write mode).
- **tracks** – A vector of tracks.

Function tracking_tracks_write_full

- Defined in file_c_fmdt_tracking_tracking_io.h

Function Documentation

void **tracking_tracks_write_full**(FILE *f, const *vec_track_t* tracks)

Print a table of tracks (dedicated to the logs).

Parameters

- **f** – File descriptor (in write mode).
- **tracks** – A vector of tracks.

Function validation_count_objects

- Defined in file_c_fmdt_validation_validation_compute.h

Function Documentation

unsigned **validation_count_objects**(const *validation_obj_t* *val_objects, const unsigned n_val_objects, unsigned *n_stars, unsigned *n_meteors, unsigned *n_noise)

Compute the number of objects in a *validation_obj_t* array.

Parameters

- **val_objects** – Array of validation objects.
- **n_val_objects** – Number of validation objects in `val_objects`.
- **n_stars** – Return the number of star objects.
- **n_meteors** – Return the number of meteor objects.
- **n_noise** – Return the number of noise objects.

Returns Total number of objects (stars + meteors + noises).

Function validation_free

- Defined in file_c_fmdt_validation_validation_compute.h

Function Documentation

void **validation_free**(void)

Free the validation global data.

Function validation_init

- Defined in file_c_fmdt_validation_validation_compute.h

Function Documentation

int **validation_init**(const char *val_objects_file)

From a file path, allocate the data required to perform the validation. Note that this function allocates data in global data: allocates the `g_val_objects` buffer and initializes it from the input file + initializes the `g_n_val_objects` global variable.

Parameters **val_objects_file** – Path to an input file of ground truth tracks to parse.

Returns Number of ground truth allocated objects.

Function validation_print

- Defined in file_c_fmdt_validation_validation_io.h

Function Documentation

void **validation_print**(const *vec_track_t* track_array)

Print a validation table into `stdout`. Note that this function uses global data to print the table.

Parameters **track_array** – Vector of tracks.

Function validation_process

- Defined in file_c_fmdt_validation_validation_compute.h

Function Documentation

void **validation_process**(const *vec_track_t* track_array)

From a given vector of tracks, estimates the correctness compared to the ground truth (stored in global data). Read `g_val_objects` and `g_n_val_objects`. Write `g_val_objects`, `g_is_valid_track`, `g_true_positive`, `g_false_positive`, `g_true_negative`, `g_false_negative`.

Parameters **track_array** – Vector of tracks.

Function version_print

- Defined in file_c_fmdt_version.h

Function Documentation

void **version_print**(const char *bin_name)

Print the FMDT version in the standard output.

Parameters **bin_name** – Name of the current executable.

Function video_reader_alloc_init

- Defined in file_c_fmdt_video_video_io.h

Function Documentation

video_reader_t ***video_reader_alloc_init**(const char *path, const size_t start, const size_t end, const size_t skip, const int bufferize, const size_t n_ffmpeg_threads, int *i0, int *i1, int *j0, int *j1)

Allocation and initialization of inner data required for a video reader.

Parameters

- **path** – Path to the video or images.
- **start** – Start frame number (first frame is frame 0).
- **end** – Last frame number (if 0 then the video sequence is entirely read).
- **skip** – Number of frames to skip between two frames (0 means no frame is skipped).
- **bufferize** – Boolean to store the entire video sequence in memory first (this is useful for benchmarks but usually the video sequences are too big to be stored in memory).
- **n_ffmpeg_threads** – Number of threads used in FFMPEG to decode the video sequence (0 means FFMPEG will decide).
- **i0** – Return the first y index in the labels (included).
- **i1** – Return the last y index in the labels (included).
- **j0** – Return the first x index in the labels (included).
- **j1** – Return the last x index in the labels (included).

Returns The allocated data.

Function video_reader_free

- Defined in file_c_fmdt_video_video_io.h

Function Documentation

void **video_reader_free**(*video_reader_t* *video)

Deallocation of inner video reader data.

Parameters **video** – A pointer of video reader inner data.

Function video_reader_get_frame

- Defined in file_c_fmdt_video_video_io.h

Function Documentation

int **video_reader_get_frame**(*video_reader_t* *video, uint8_t **img)

Write grayscale image in a given 2D array.

Parameters

- **video** – A pointer of previously allocated inner video reader data.
- **img** – Output grayscale image (2D array $[i1 - i0 + 1][j1 - j0 + 1]$).

Returns The frame id (positive integer) or -1 if there is no more frame to read.

Function video_writer_alloc_init

- Defined in file_c_fmdt_video_video_io.h

Function Documentation

video_writer_t ***video_writer_alloc_init**(const char *path, const size_t start, const size_t n_ffmpeg_threads, const size_t img_height, const size_t img_width, const enum *pixfmt_e* pixfmt)

Allocation and initialization of inner data required for a video writer.

Parameters

- **path** – Path to the video or images.
- **start** – Start frame number (first frame is frame 0).
- **n_ffmpeg_threads** – Number of threads used in FFMPEG to encode the video sequence (0 means FFMPEG will decide).
- **img_height** – Images height.
- **img_width** – Images width.
- **pixfmt** – Pixels format (grayscale or RGB).

Returns The allocated data.

Function video_writer_free

- Defined in file_c_fmdt_video_video_io.h

Function Documentation

void **video_writer_free**(*video_writer_t* *video)

Deallocation of inner video writer data.

Parameters **video** – A pointer of video writer inner data.

Function video_writer_save_frame

- Defined in file_c_fmdt_video_video_io.h

Function Documentation

void **video_writer_save_frame**(*video_writer_t* *video, const uint8_t **img)

Allocation of inner data required for a video writer.

Parameters

- **video** – A pointer of previously allocated inner video writer data.
- **img** – Input grayscale/RGB image (2D array [img_height][img_width]).

9.3.5 Variables

Variable g_change_state_to_string

- Defined in file_c_fmdt_tracking_tracking_global.h

Variable Documentation

char **g_change_state_to_string**[*N_REASONS*][64]

LUT to find reason string from its reason

Variable g_change_state_to_string_with_spaces

- Defined in file_c_fmdt_tracking_tracking_global.h

Variable Documentation

char **g_change_state_to_string_with_spaces**[*N_REASONS*][64]

LUT to find reason string (with spaces) from its reason

Variable g_false_negative

- Defined in file_c_fmdt_validation_validation_global.h

Variable Documentation

int **g_false_negative**[*N_OBJECTS*]

Counters of false negative tracks depending on the object types.

Variable g_false_positive

- Defined in file_c_fmdt_validation_validation_global.h

Variable Documentation

int **g_false_positive**[*N_OBJECTS*]

Counters of false positive tracks depending on the object types.

Variable g_fmdt_build

- Defined in file_c_fmdt_version.h

Variable Documentation

unsigned **g_fmdt_build**

FMDT build (b)

Variable g_fmdt_sha1

- Defined in file_c_fmdt_version.h

Variable Documentation

char **g_fmdt_sha1**[256]

FMDT full SHA1 hash (from Git)

Variable g_fmdt_version

- Defined in file_c_fmdt_version.h

Variable Documentation

char **g_fmdt_version**[256]

FMDT full version, in the following form: vM.m.p-b-ghash7

Variable g_fmdt_version_major

- Defined in file_c_fmdt_version.h

Variable Documentation

unsigned **g_fmdt_version_major**

FMDT major version (M)

Variable g_fmdt_version_minor

- Defined in file_c_fmdt_version.h

Variable Documentation

unsigned **g_fmdt_version_minor**

FMDT minor version (m)

Variable g_fmdt_version_patch

- Defined in file_c_fmdt_version.h

Variable Documentation

unsigned **g_fmdt_version_patch**

FMDT patch (p)

Variable g_is_valid_track

- Defined in file_c_fmdt_validation_validation_global.h

Variable Documentation

uint8_t **g_is_valid_track**[MAX_TRACKS_SIZE]

Array that contains 1 or 2 value. 1 means that the current track is a true positive, 2 means that the current track is a false positive.

Variable g_n_val_objects

- Defined in file_c_fmdt_validation_validation_global.h

Variable Documentation

unsigned **g_n_val_objects**

Number of tracks from the ground truth.

Variable g_obj_to_color

- Defined in file_c_fmdt_tracking_tracking_global.h

Variable Documentation

enum *color_e* **g_obj_to_color**[*N_OBJECTS*]

LUT to find object color from its type

Variable g_obj_to_string

- Defined in file_c_fmdt_tracking_tracking_global.h

Variable Documentation

char **g_obj_to_string**[*N_OBJECTS*][64]

LUT to find object string from its type

Variable g_obj_to_string_with_spaces

- Defined in file_c_fmdt_tracking_tracking_global.h

Variable Documentation

char **g_obj_to_string_with_spaces**[*N_OBJECTS*][64]

LUT to find object string (with spaces) from its type

Variable g_true_negative

- Defined in file_c_fmdt_validation_validation_global.h

Variable Documentation

int **g_true_negative**[*N_OBJECTS*]

Counters of true negative tracks depending on the object types.

Variable g_true_positive

- Defined in file_c_fmdt_validation_validation_global.h

Variable Documentation

int **g_true_positive**[*N_OBJECTS*]

Counters of true positive tracks depending on the object types.

Variable g_val_objects

- Defined in file_c_fmdt_validation_validation_global.h

Variable Documentation

validation_obj_t ***g_val_objects**

Array of ground truth tracks.

9.3.6 Defines

Define CLAMP

- Defined in file_c_fmdt_macros.h

Define Documentation

CLAMP(x, a, b)

Define CR

- Defined in file_c_fmdt_macros.h

Define Documentation

CR

Define DISP

- Defined in file_c_fmdt_macros.h

Define Documentation

DISP(x)

Define FDISP

- Defined in file_c_fmdt_macros.h

Define Documentation

FDISP(x)

Define IDISP

- Defined in file_c_fmdt_macros.h

Define Documentation

IDISP(x)

Define MAX

- Defined in file_c_fmdt_macros.h

Define Documentation

MAX(a, b)

Define MAX_ROI_SIZE

- Defined in file_c_fmdt_features_features_struct.h

Define Documentation

MAX_ROI_SIZE

Maximum number of RoIs after `features_merge_CCL_HI_v2` selection.

Define MAX_ROI_SIZE_BEFORE_SHRINK

- Defined in file_c_fmdt_features_features_struct.h

Define Documentation

MAX_ROI_SIZE_BEFORE_SHRINK

Maximum number of RoIs before `features_merge_CCL_HI_v2` selection.

Define MAX_TRACKS_SIZE

- Defined in file_c_fmdt_validation_validation_global.h

Define Documentation

MAX_TRACKS_SIZE

Maximum number of tracks to evaluate in the validation process.

Define METEOR_COLOR

- Defined in file_c_fmdt_tracking_tracking_global.h

Define Documentation

METEOR_COLOR

Associate the green color to a meteor

Define METEOR_STR

- Defined in file_c_fmdt_tracking_tracking_global.h

Define Documentation

METEOR_STR

Define “meteor” string

Define MIN

- Defined in file_c_fmdt_macros.h

Define Documentation

MIN(a, b)

Define NOISE_COLOR

- Defined in file_c_fmdt_tracking_tracking_global.h

Define Documentation

NOISE_COLOR

Associate the orange color to noise

Define NOISE_STR

- Defined in file_c_fmdt_tracking_tracking_global.h

Define Documentation

NOISE_STR

Define "noise" string

Define PUTS

- Defined in file_c_fmdt_macros.h

Define Documentation

PUTS(str)

Define SHOWNAME

- Defined in file_c_fmdt_macros.h

Define Documentation

SHOWNAME(X)

Define STAR_COLOR

- Defined in file_c_fmdt_tracking_tracking_global.h

Define Documentation

STAR_COLOR

Associate the purple color to a star

Define STAR_STR

- Defined in file_c_fmdt_tracking_tracking_global.h

Define Documentation

STAR_STR

Define "star" string

Define TOO_BIG_ANGLE_STR

- Defined in file_c_fmdt_tracking_tracking_global.h

Define Documentation

TOO_BIG_ANGLE_STR

Define "too big angle" string

Define TOO_LONG_DURATION_STR

- Defined in file_c_fmdt_tracking_tracking_global.h

Define Documentation

TOO_LONG_DURATION_STR

Define "too long duration" string

Define UNKNOWN_COLOR

- Defined in file_c_fmdt_tracking_tracking_global.h

Define Documentation

UNKNOWN_COLOR

Associate the gray color to unknown object

Define UNKNOWN_STR

- Defined in file_c_fmdt_tracking_tracking_global.h

Define Documentation

UNKNOWN_STR

Define "unknown" string

Define VERBOSE

- Defined in file_c_fmdt_macros.h

Define Documentation

VERBOSE(X)

Define WRONG_DIRECTION_STR

- Defined in file_c_fmdt_tracking_tracking_global.h

Define Documentation

WRONG_DIRECTION_STR

Define "wrong direction" string

9.3.7 Typedefs

Typedef vec_BB_t

- Defined in file_c_fmdt_tracking_tracking_struct.h

Typedef Documentation

typedef *BB_t* ***vec_BB_t**

Vector of *BB_t*, to use with C vector lib.

Typedef vec_color_e

- Defined in file_c_fmdt_image_image_struct.h

Typedef Documentation

typedef enum *color_e* ***vec_color_e**

Vector of colors, to use with C vector lib.

Typedef vec_track_t

- Defined in file_c_fmdt_tracking_tracking_struct.h

Typedef Documentation

typedef *track_t* ***vec_track_t**

Vector of `track_t`, to use with C vector lib.

Typedef vec_uint32_t

- Defined in file_c_fmdt_tracking_tracking_struct.h

Typedef Documentation

typedef uint32_t ***vec_uint32_t**

Vector of `uint32_t`, to use with C vector lib.

BIBLIOGRAPHY

[KCM+22] Mathuran Kandeepan, Clara Ciocan, Maxime Millet, Manuel Bouyer, Adrien Cassagne, and Lionel Lacassagne. Fast Meteor Detection Toolbox. Workshop AFF3CT, November 2022. Poster. doi:10.13140/RG.2.2.12222.36161.

[VLC+23] Jérémie Vaubaillon, Charlotte Loir, Clara Ciocan, Mathuran Kandeepan, Maxime Millet, Adrien Cassagne, Lionel Lacassagne, Pedro da Fonseca, Fabian Zander, David Buttsworth, Stefan Loehle, Juraj Tóth, Scott Gray, Audrey Moingeon, and Nicolas Rambaux. A 2022 tau-Herculid meteor cluster from an airborne experiment: automated detection, characterization, and consequences for meteoroids. *Astronomy and Astrophysics - A&A*, 2023. doi:10.1051/0004-6361/202244993.

INDEX

Symbols

A

B

C

D

F

G

I

K

M

N

O

P

R

S

T

U

V

W