FastTrack

5.0.0

Generated by Doxygen 1.8.17

Chapter 1

Fast Track developer guide

Introduction

FastTrack is developed to be embedded in existing C++/Qt projects. It can also be adapted to every project by re-implementing the existing software. **Note:** more recent information are available at 1 and 2 for Windows systems.

Installation of the development environment

Windows

Qt installation

1. Go to https://download/qt.io/archive/qt and choose the last version of Qt (this example is made with the 5.12 version). Download the Window installer.
2. Follow the installation steps and select **C:\Qt\Qt5.12.0** for the installation folder. In the Select Components page, select MinGW 7.3.0 64 bit and Qt creator 4.8.
3. Add MinGW to the path:
    - Open the **Settings** dialogue.
    - Open the **Edit the system environment variables** and click on the **Environment Variables** button.
    - Double click on **Path** and add the MinGW path: **C:\Qt\Qt5.12.0\Tools\mingw730_64\bin**.

OpenCV installation

- Download the last version of OpenCV at https://sourceforge.net/projects/opencvlibrary/files/ (in this example 4.0.1). Select the Windows file **opencv-4.0.1-vc14_vc15.exe**.
- Extract OpenCV in the **C:\ ** folder.
- Download and install CMake https://cmake.org/download/.
- Open CMake an set **Where is the source code** to **C:/opencv/source** ** and **Where to build the binaries** to **C:/opencv/source/build **.
- Click on the Configure button and select **MinGW Makefiles** and tick **Specify native compiler**.

- Set the C and C++ path compiler to **C:/Qt/Qt5.12.0/Tools/mingw730_64/bin/gcc.exe** and **C:/Qt/Qt5.12.0/↩ Tools/mingw730_64/bin/g++.exe**.
- Untick **WITH_OPENCL** and tick **WITH_OPENMP** and click on the Configure button. Then click the Generate button.
- Open the **Command Prompt** application.
- Type (replace 8 by your the number of processors on your computer) the command

```
cd C:\opencv\sources\build & mingw32-make -j 8 & mingw32-make install
```

to compile and install OpenCV.

- Add ∗∗C:\opencv\sources\build\install\x64\mingw\bin∗∗ to the Path.

Configure FastTrack

Replace **FastTrack.pro** by:

```
QT       += core gui
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
TARGET = FastTrack
TEMPLATE = app
QMAKE_LFLAGS_RELEASE += -O3
# The following define makes your compiler emit warnings if you use
# any feature of Qt which has been marked as deprecated (the exact warnings
# depend on your compiler). Please consult the documentation of the
# deprecated API in order to know how to port your code away from it.
DEFINES += QT_DEPRECATED_WARNINGS
# You can also make your code fail to compile if you use deprecated APIs.
# In order to do so, uncomment the following line.
# You can also select to disable deprecated APIs only up to a certain version of Qt.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs deprecated before Qt 6.0.0
SOURCES += \
        main.cpp \
        mainwindow.cpp \
        tracking.cpp \
        setupwindow.cpp \
        Hungarian.cpp
QMAKE_CXXFLAGS += -std=c++11 -O3 -fopenmp
INCLUDEPATH += C:/opencv/build/include
LIBS += C:/opencv/sources/build/install/x64/mingw/bin/libopencv_core401.dll
LIBS += C:/opencv/sources/build/install/x64/mingw/bin/libopencv_highgui401.dll
LIBS += C:/opencv/sources/build/install/x64/mingw/bin/libopencv_imgcodecs401.dll
LIBS += C:/opencv/sources/build/install/x64/mingw/bin/libopencv_imgproc401.dll
LIBS += C:/opencv/sources/build/install/x64/mingw/bin/libopencv_videoio401.dll
HEADERS += \
        mainwindow.h\
        tracking.h \
        setupwindow.h \
        Hungarian.h
FORMS += mainwindow.ui \
        setupwindow.ui
RESOURCES += resources.qrc
```

Linux / MacOs

- Download OpenCV.

```
git clone https://github.com/opencv/opencv
```

- Compile OpenCV (can need additional dependencies like **build-essential** and **libgl1-mesa-dev**).

```
cd opencv
mkdir build
cd build
cmake -D WITH_TBB=ON -D WITH_OPENMP=ON -D WITH_IPP=ON -D CMAKE_BUILD_TYPE=RELEASE -D BUILD_EXAMPLES=OFF -D
       WITH_NVCUVID=ON -D WITH_CUDA=ON -D BUILD_DOCS=OFF -D BUILD_PERF_TESTS=OFF -D BUILD_TESTS=OFF -D
       WITH_CSTRIPES=ON -D WITH_OPENCL=OFF CMAKE_INSTALL_PREFIX=/usr/local/ ..
make -j8
sudo make install
```

- Compile FastTrack

```
cd FastTrack
./run full                // Compiles from scratch
./run partial             // Compiles changes
./run debug               // Compiles for debugging
./run profile            // Creates profiling.txt
```

Adapt FastTrack for our project

To adapt FastTrack for our project, you must re-implement the **startProcess()** and **imageAnalysis()** method from the Tracking class with our own image analysis workflow.

startProcess() method

The **startProcess()** method initializes the tracking process by taking the first image of the sequence, detects its format and all the objects in the image.

By default the image analysis workflow is the following:

- Read image
- Binarize (optional)
- Thresholding
- Dilate (optional)
- The region of interest selection (optional)
- Object detection (detection + parameters extraction)
- Parameters saving.

The **startProcess()** method will emit a signal with the images to display and a signal to trigger the analysis of the rest of the image sequence.

imageAnalysis() method

The **imageAnalysis()** method detects objects, extracts its parameters and associates objects to keep track of individual identity.

By default the image analysis workflow is the following:

- Read image
- Binarize (optional)
- Thresholding
- Dilate (optional)
- The region of interest selection (optional)
- Object detection (detection + parameters extraction)
- Objects association.
- Parameters saving.

The **imageAnalysis()** method will emit a signal with images to display and triggered via a timer the analysis of the next image of the image sequence.

Embedded Fast Track in our project

Video tracking

To embedded Fast Track in an existing project, you must first create a thread where the Tracking class will live.

```
 {C++}
thread = new QThread; // Creates a new QThread
tracking = new Tracking("path/to/folder/where/is/stored/the/image/sequence"); // Instantiates Tracking class
tracking -> moveToThread(thread); // Moves the Tracking instance in the new QThread
connect(thread, &QThread::started, tracking, &Tracking::startProcess); // Starts the tracking analysis when
       the thread start
// Do here all useful connect like updating parameters, display images etc...
connect(tracking, &Tracking::finished, thread, &QThread::quit); // Shut down the thread when the tracking
       analysis is finished
connect(tracking, &Tracking::finished, tracking, &Tracking::deleteLater); // Deletes the Tracking instance
       when the tracking analysis is finished
connect(thread, &QThread::finished, thread, &QThread::deleteLater); // Thread will be deleted only after it
       has fully shut down

thread->start(); // Starts the tracking analysis
```

The tracking analysis will be running in the thread and destroy itself at the end.

Real-time tracking

FastTrack supports live tracking analysis. Be sure to test the program on a video before to see if the analysis frame rate is lower or equal to the tracking analysis frame rate.

Create an acquisition image thread with an object Camera that sends a signal newImage(UMat) when a new image is available.

```
 {c++}
thread = new QThread; // Creates a new QThread
tracking = new Tracking(""); // Instantiates Tracking class
tracking -> moveToThread(thread); // Moves the Tracking instance in the new QThread
connect(thread, &QThread::started, tracking, &Tracking::startProcess); // Starts the tracking analysis when
       the thread start
connect(camera, &Camera::newImage, tracking, &Tracking::receivedFrame);
// Do here all usefull connect like updating parameters, display images etc...
connect(tracking, &Tracking::finished, thread, &QThread::quit); // Shut down the thread when the tracking
       analysis is finished
connect(tracking, &Tracking::finished, tracking, &Tracking::deleteLater); // Deletes the Tracking instance
       when the tracking analysis is finished
connect(thread, &QThread::finished, thread, &QThread::deleteLater); // Thread will be deleted only after it
       has fully shut down

thread->start(); // Starts the tracking analysis
```

Generate documentation

The documentation can be generated in HTML and PDF format with Doxygen. Install Doxygen and run

```
./generateDocumentation.sh
```

Test

FastTrack have a test script that allows to test the code after changes. To use the test script, install gcode.

```
./test
```

Revised 2020/03/19

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

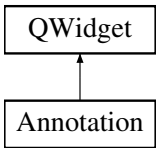
Chapter 4

Class Documentation

4.1 Annotation Class Reference

This class allows to load tracking annotation file.

Inheritance diagram for Annotation:



Public Slots

- void write (int index, const QString &text)

  Adds an annotation to the annotation QMap.
- void read (int index)

  Reads an annotation from the annotation QMap.
- void find (const QString &expression)

  Finds the index of all the annotation with expression inside their text.
- int next ()

  Returns the next element of the findIndexes list of annotations that contains the expression to find.
- int prev ()

  Returns the previous element of the findIndexes list of annotations that contains the expression to find.

Signals

- void annotationText (const QString &text)

  Emitted when a new annotation is read.

Public Member Functions

- Annotation (const QString &annotationFile)

  Constructs the annotation object from a file path.

Private Member Functions

- void writeToFile ()

  Writes all the annotation to a file.

Private Attributes

- QFile * **annotationFile**
- QMap< int, QString > * **annotations**
- QList< int > **findIndexes**
- int **findIndex**

4.1.1 Detailed Description

This class allows to load tracking annotation file.

Author

Benjamin Gallois

Version

Revision

4.8

Contact: benjamin.gallois@fasttrack.sh

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Annotation()

```
Annotation::Annotation (
            const QString & filePath )  [explicit]
```

Constructs the annotation object from a file path.

Parameters

in	*filePath*	Path to the tracking folder.

4.1.3 Member Function Documentation

4.1.3.1 annotationText

```
void Annotation::annotationText (
            const QString & text )  [signal]
```

Emitted when a new annotation is read.

Parameters

text	Text of the requested annotation.

4.1.3.2 find

```
void Annotation::find (
            const QString & expression )  [slot]
```

Finds the index of all the annotation with expression inside their text.

Parameters

in	*expression*	Expression to find, case sensitive.

4.1.3.3 read

```
void Annotation::read (
            int index )  [slot]
```

Reads an annotation from the annotation QMap.

Parameters

in	*index*	Image index.

4.1.3.4 write

```
void Annotation::write (
            int index,
            const QString & text )  [slot]
```

Adds an annotation to the annotation QMap.

Parameters

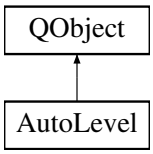
in	*index*	Image index.
in	*text*	Annotation text.

The documentation for this class was generated from the following files:

- src/annotation.h
- src/annotation.cpp

4.2 AutoLevel Class Reference

Inheritance diagram for AutoLevel:



Public Slots

- QMap< QString, double > level ()

  Levels the tracking parameters.

Signals

- void **forceFinished** ()
- void **levelParametersChanged** (QMap< QString, double >)
- void **finished** ()

Public Member Functions

- AutoLevel (const string &path, const UMat &background, const QMap< QString, QString > &parameters)

  Constructs the AutoLevel object.

Static Public Member Functions

- static double stdev (const QVector< double > &vect)

  Compute the std from a vector.

Private Member Functions

- double computeStdAngle (const Data &data)

  Compute the standard deviation of the angle distribution.
- double computeStdDistance (const Data &data)

  Compute the standard deviation of the distance distribution.
- double computeStdArea (const Data &data)

  Compute the standard deviation of the area distribution.
- double computeStdPerimeter (const Data &data)

  Compute the standard deviation of the angle distribution.

Private Attributes

- int m_endImage
- string m_path
- QString m_spotSuffix
- UMat m_background
- QMap< QString, QString > m_parameters

4.2.1 Constructor & Destructor Documentation

4.2.1.1 AutoLevel()

```
AutoLevel::AutoLevel (
            const string & path,
            const UMat & background,
            const QMap< QString, QString > & parameters )
```

Constructs the AutoLevel object.

Parameters

in	*path*	Path to the movie to track.
in	*background*	Background image.

4.2.2 Member Function Documentation

4.2.2.1 computeStdAngle()

```
double AutoLevel::computeStdAngle (
            const Data & data )  [private]
```

Compute the standard deviation of the angle distribution.

Parameters

in	*data*	Tracking data.

Returns

Standard deviation.

4.2.2.2 computeStdArea()

```
double AutoLevel::computeStdArea (
            const Data & data )  [private]
```

Compute the standard deviation of the area distribution.

Parameters

in	*data*	Tracking data.

Returns

Standard deviation.

4.2.2.3 computeStdDistance()

```
double AutoLevel::computeStdDistance (
            const Data & data )  [private]
```

Compute the standard deviation of the distance distribution.

Parameters

in	*data*	Tracking data.

Returns

Standard deviation.

4.2.2.4 computeStdPerimeter()

```
double AutoLevel::computeStdPerimeter (
            const Data & data )  [private]
```

Compute the standard deviation of the angle distribution.

Parameters

in	*data*	Tracking data.

Returns

Standard deviation.

4.2.2.5 level

```
QMap< QString, double > AutoLevel::level ( )  [slot]
```

Levels the tracking parameters.

Returns

Map containing the levelled parameters.

4.2.2.6 stdev()

```
double AutoLevel::stdev (
            const QVector< double > & vect )  [static]
```

Compute the std from a vector.

Parameters

in	*data*	Distribution.

Returns

Std.

4.2.3 Member Data Documentation

4.2.3.1 m_background

```
UMat AutoLevel::m_background  [private]
```

Path to video file/image sequence.

4.2.3.2 m_endImage

```
int AutoLevel::m_endImage  [private]
```

Optimal ending image index.

4.2.3.3 m_parameters

```
QMap<QString, QString> AutoLevel::m_parameters  [private]
```

Optimal ending image index.

4.2.3.4 m_path

```
string AutoLevel::m_path  [private]
```

Path to video file/image sequence.

4.2.3.5 m_spotSuffix

```
QString AutoLevel::m_spotSuffix  [private]
```

Spot to track.

The documentation for this class was generated from the following files:

- src/autolevel.h
- src/autolevel.cpp

4.3 Autolevel Class Reference

This class is entended to level the soft tracking parameters.

4.3.1 Detailed Description

This class is entended to level the soft tracking parameters.

Author

Benjamin Gallois

Version

Revision

5.0

Contact: gallois.benjamin08@gmail.com

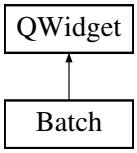
The documentation for this class was generated from the following file:

- src/autolevel.cpp

4.4 Batch Class Reference

The Batch widget provides an convenient way to add multiple files to analyze.

Inheritance diagram for Batch:



Classes

- struct process

Public Slots

- void updateParameters ()

  Updates the parameterList vector with the new parameter when users changes a parameter in the QTableWidget of parameters. Triggered when ui->tableParameters is modified. Emits the updated parameters QMap.
- void startTracking ()

  Starts a new tracking analysis. First, it gets the path to the folder containing the image sequence. It creates a folder named Tracking_Result in this folder and a file parameters.txt containing the parameterList. It creates a new Tracking object that has to be run in a separate thread. When the analysis is finished, the Tracking object is destroyed and a new analysis is started. Triggerred when the start analysis button is clicked or when the signal finishedAnalysis() is emitted.
- void openPathFolder ()

  Opens a dialog window to select folders. Triggered when the openPath button is clicked. If auto-detection mode is enable, it will also selects a background image and/or a parameter file and update the list of file to process. It also possible to add a suffix to the selected path.
- void openPathBackground (int)

  Opens a dialog window to select a background image. Triggered when an open background is clicked in the pathTable.
- void addPath (QString, QString, QString)

  Adds movie to the list of movies to analyze.
- void removePath ()

  Deletes the selected line in the ui->tablePath and the corresponding path in the pathList. Triggered when the ui->removePath button is clicked.
- void removePath (int index)

  Deletes the row at index in the pathPanel.
- void updateParameterTable ()

  Takes the QMap parameterList and updates the parameters panel table..
- bool loadParameterFile (QString path)

  Reads a parameter file, updates parameters.
- void openParameterFile (int)

  Opens a dialog to select a parameter file.
- void errors (int code)

  Displays an error message.

Signals

- void newParameterList (const QMap< QString, QString > &parameterList)

  Emitted when a parameter is changed.
- void next ()

  Emitted when a tracking analysis is finished.
- void **log** (QMap< QString, QString > log)

Public Member Functions

- Batch (QWidget *parent=nullptr)

  Constructs the Batch widget.

Private Member Functions

- void loadSettings ()

  Loads the settings file at the startup of the program and updates the ui->parameterTable with the new parameters.
- void saveSettings ()

  Saves all the parameters in the settings file.

Private Attributes

- Ui::Batch * ui
- QMap< QString, QString > parameterList
- QThread * thread
- Tracking * tracking
- QShortcut * wShortcut
- QShortcut * qShortcut
- QShortcut * aShortcut
- QShortcut * dShortcut
- QVector< process > **processList**
- QSettings * settingsFile
- QString memoryDir
- bool **isEditable**
- int **currentPathCount**

4.4.1 Detailed Description

The Batch widget provides an convenient way to add multiple files to analyze.

Author

Benjamin Gallois

Version

Revision

460

Contact: gallois.benjamin08@gmail.com

4.4.2 Member Function Documentation

4.4.2.1 addPath

```
void Batch::addPath (
            QString pathMovie,
            QString pathBackground,
            QString pathParameter )  [slot]
```

Adds movie to the list of movies to analyze.

- [in] pathMovie Path to the image sequence to analyze.
- [in] pathBackground Path to the background image.
- [in] pathParameter Path to the parameter file.

4.4.2.2 newParameterList

```
void Batch::newParameterList (
            const QMap< QString, QString > & parameterList )  [signal]
```

Emitted when a parameter is changed.

Parameters

parameterList	All parameters necessary to the tracking analysis.

4.4.2.3 openParameterFile

```
void Batch::openParameterFile (
            int row )  [slot]
```

Opens a dialog to select a parameter file.

- [in] row Index of the row containing the button in the pathPanel.

4.4.2.4 openPathBackground

```
void Batch::openPathBackground (
            int row )  [slot]
```

Opens a dialog window to select a background image. Triggered when an open background is clicked in the path↩ Table.

- [in] row Index of the row containing the button in the pathPanel.

4.4.2.5 removePath

```
void Batch::removePath (
            int row )  [slot]
```

Deletes the row at index in the pathPanel.

- [in] row Index of the row.

4.4.2.6 updateParameters

```
void Batch::updateParameters ( )  [slot]
```

Updates the parameterList vector with the new parameter when users changes a parameter in the QTableWidget of parameters. Triggered when ui->tableParameters is modified. Emits the updated parameters QMap.

Parameters

in	*item*	QTableWidgetItem from a QTableWidget.

4.4.3 Member Data Documentation

4.4.3.1 aShortcut

```
QShortcut* Batch::aShortcut  [private]
```

Keyboard shortcut to previous frame.

4.4.3.2 dShortcut

```
QShortcut* Batch::dShortcut  [private]
```

Keyboard shortcut to next frame.

4.4.3.3 memoryDir

```
QString Batch::memoryDir  [private]
```

Saves the path of the last opened folder.

4.4.3.4 parameterList

```
QMap<QString, QString> Batch::parameterList  [private]
```

All the parameters necessary for the tracking analysis.

4.4.3.5 qShortcut

```
QShortcut* Batch::qShortcut  [private]
```

Keyboard shortcut to previous frame.

4.4.3.6 settingsFile

```
QSettings* Batch::settingsFile  [private]
```

Saves parameters in a settings.ini file.

4.4.3.7 thread

```
QThread* Batch::thread  [private]
```

Thread where lives the Tracking object.

4.4.3.8 tracking

```
Tracking* Batch::tracking  [private]
```

Objects that track images sequence.

4.4.3.9 ui

```
Ui::Batch* Batch::ui  [private]
```

ui file from Qt designer.

4.4.3.10 wShortcut

```
QShortcut* Batch::wShortcut  [private]
```

Keyboard shortcut to next occlusion.

The documentation for this class was generated from the following files:

- src/batch.h
- src/batch.cpp

4.5 Data Class Reference

This class allows to load tracking data produced by the Tracking class.

Public Member Functions

- Data (QString dataPath)

  Constructs the data object from a tracking result file.
- QVector< object > getData (int imageIndex) const

  Gets the tracking data at the selected image number for all the objects.
- QMap< QString, double > getData (int imageIndex, int id) const

  Gets the tracking data at the selected image number for one selected object.
- QMap< QString, QVector< double > > **getDataId** (int id) const
- QList< int > getId (int imageIndex) const

  Gets the ids of all the objects in the frame.
- QList< int > getId (int imageIndexFirst, int imageIndexLast) const

  Gets the ids of all the objects in several frames.
- int getObjectInformation (int objectId) const

  Gets the object's information.
- void swapData (int firstObject, int secondObject, int from)

  In the tracking data, swaps two objects from a selected index to the end.
- void deleteData (int objectId, int from, int to)

  Deletes the tracking data of one object from a selected index to the end.
- void insertData (int objectId, int from, int to)

  Insert the tracking data for one object from a selected index to the end.
- void save ()

  Saves the data in the tracking result file.

Public Attributes

- QMap< int, QVector< object > > data
- int **maxId**
- int **maxFrameIndex**
- QMap< int, QVector< object > > **dataCopy**

Private Attributes

- QString dir

4.5.1 Detailed Description

This class allows to load tracking data produced by the Tracking class.

Author

Benjamin Gallois

Version

Revision

4.0

Contact: gallois.benjamin08@gmail.com

4.5.2 Constructor & Destructor Documentation

4.5.2.1 Data()

```
Data::Data (
            QString dataPath )  [explicit]
```

Constructs the data object from a tracking result file.

Parameters

in	*dataPath*	Path to the tracking data file.

4.5.3 Member Function Documentation

4.5.3.1 deleteData()

```
void Data::deleteData (
            int objectId,
```

```
            int from,
            int to )
```

Deletes the tracking data of one object from a selected index to the end.

- [in] objectId The object id.
- [in] from Start index from which the data will be swapped.
- [in] to End index from which the data will be swapped.

4.5.3.2 getData() [1/2]

```
QVector< object > Data::getData (
            int imageIndex ) const
```

Gets the tracking data at the selected image number for all the objects.

Parameters

in	*imageIndex*	The index of the image where to extracts the data.

Returns

The tracking data in a QVector that contains a structure with the Id of the object and data for this object. The data are stored in a QMap<dataName, value>.

4.5.3.3 getData() [2/2]

```
QMap< QString, double > Data::getData (
            int imageIndex,
            int id ) const
```

Gets the tracking data at the selected image number for one selected object.

Parameters

in	*imageIndex*	The index of the image where to extracts the data.
in	*id*	The id of the object.

Returns

The tracking data for for the selected object at the selected image. The data are stored in a QMap<dataName, value>.

4.5.3.4 getId() [1/2]

```
QList< int > Data::getId (
            int imageIndex ) const
```

Gets the ids of all the objects in the frame.

- [in] imageIndex Index of the frame.

Returns

List of indexes.

4.5.3.5 getId() [2/2]

```
QList< int > Data::getId (
            int imageIndexFirst,
            int imageIndexLast ) const
```

Gets the ids of all the objects in several frames.

- [in] imageIndexFirst Index of the first frame.
- [in] imageIndexLast Index of the last frame.

Returns

List of indexes.

4.5.3.6 getObjectInformation()

```
int Data::getObjectInformation (
            int objectId ) const
```

Gets the object's information.

- [in] objectId Id of the object.

Returns

First appearance image index.

4.5.3.7 insertData()

```
void Data::insertData (
            int objectId,
            int from,
            int to )
```

Insert the tracking data for one object from a selected index to the end.

- [in] objectId The object id.
- [in] from Start index from which the data will be swapped.
- [in] to End index from which the data will be swapped.

4.5.3.8 swapData()

```
void Data::swapData (
            int firstObject,
            int secondObject,
            int from )
```

In the tracking data, swaps two objects from a selected index to the end.

- [in] firstObject The first object id.
- [in] secondObject The second object id.
- [in] from Start index from which the data will be swapped.

4.5.4 Member Data Documentation

4.5.4.1 data

```
QMap<int, QVector<object> > Data::data
```

Tracking data stored in a QMap, the keys are the image index and the value a vector of data stored in a structure with a field containing the object id and a field containing the data stored in a QMap where the keys are the data name and the velue the data value.

4.5.4.2 dir

```
QString Data::dir  [private]
```

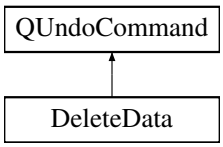
Path to the tracking result file.

The documentation for this class was generated from the following files:

- src/data.h
- src/data.cpp

4.6 DeleteData Class Reference

Inheritance diagram for DeleteData:



Public Member Functions

- **DeleteData** (int object, int from, int to, Data ∗data)
- void **undo** () override
- void **redo** () override

Private Attributes

- int **m_object**
- int **m_from**
- int **m_to**
- Data ∗ **m_data**

The documentation for this class was generated from the following files:

- src/data.h
- src/data.cpp

4.7 HungarianAlgorithm Class Reference

Public Member Functions

- double **Solve** (vector< vector< double > > &DistMatrix, vector< int > &Assignment)

Private Member Functions

- void **assignmentoptimal** (int ∗assignment, double ∗cost, double ∗distMatrix, int nOfRows, int nOfColumns)
- void **buildassignmentvector** (int ∗assignment, bool ∗starMatrix, int nOfRows, int nOfColumns)
- void **computeassignmentcost** (int ∗assignment, double ∗cost, double ∗distMatrix, int nOfRows)
- void **step2a** (int ∗assignment, double ∗distMatrix, bool ∗starMatrix, bool ∗newStarMatrix, bool ∗primeMatrix, bool ∗coveredColumns, bool ∗coveredRows, int nOfRows, int nOfColumns, int minDim)
- void **step2b** (int ∗assignment, double ∗distMatrix, bool ∗starMatrix, bool ∗newStarMatrix, bool ∗primeMatrix, bool ∗coveredColumns, bool ∗coveredRows, int nOfRows, int nOfColumns, int minDim)
- void **step3** (int ∗assignment, double ∗distMatrix, bool ∗starMatrix, bool ∗newStarMatrix, bool ∗primeMatrix, bool ∗coveredColumns, bool ∗coveredRows, int nOfRows, int nOfColumns, int minDim)
- void **step4** (int ∗assignment, double ∗distMatrix, bool ∗starMatrix, bool ∗newStarMatrix, bool ∗primeMatrix, bool ∗coveredColumns, bool ∗coveredRows, int nOfRows, int nOfColumns, int minDim, int row, int col)
- void **step5** (int ∗assignment, double ∗distMatrix, bool ∗starMatrix, bool ∗newStarMatrix, bool ∗primeMatrix, bool ∗coveredColumns, bool ∗coveredRows, int nOfRows, int nOfColumns, int minDim)

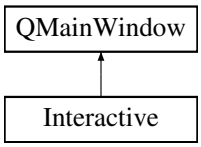
The documentation for this class was generated from the following files:

- src/Hungarian.h
- src/Hungarian.cpp

4.8 Interactive Class Reference

The Interactive widget provides an environment to use the tracking widget in an interactive environment.

Inheritance diagram for Interactive:



Signals

- void **message** (QString message)
- void **log** (QMap< QString, QString > log)

Public Member Functions

- Interactive (QWidget *parent=nullptr)

  Constructs the interactive object derived from a QMainWindow object.
- ~Interactive ()

  Destructors.

Private Slots

- void openFolder ()

  Asks the path to a folder where an image sequence is stored. Setups the ui and resets the class attributs for a new analysis. Triggered when the open button from the menu bar is clicked.
- void display (int index, int scale=0)

  Displays the image at index in the image sequence in the ui.
- void display (QImage image)

  This is an overloaded function to display a QImage in the display.
- void display (UMat image)

  This is an overloaded function to display a UMat in the display.
- void zoomIn ()

  Zooms in the display.
- void zoomOut ()

  Zooms out the display.
- void getParameters ()

  Gets all the tracking parameters from the ui and updates the parameter map that will be passed to the tracking object.
- void previewTracking ()

  Does a tracing analysis on a sub-part of the image sequence defined by the user. Triggered when previewButton is clicked.
- void track ()

  Does a tracking analysis. Triggered when the trackButton is clicked.
- void computeBackground ()

  Computes and displays the background image in the display. Triggered when the backgroundComputeButton is clicked.

- void selectBackground ()

  Opens a dialogue to select a background image. Triggered when ui->backgroundSelectButton is pressed.
- bool eventFilter (QObject *target, QEvent *event)

  Manages all the mouse inputs in the display.
- void crop ()

  Crops the image from a rectangle drawed by the user with the mouse on the display. Triggered when the QPushButton ui->crop is clicked.
- void reset ()

  Resets the region of interest. Triggered by the reset button.
- void loadSettings ()

  Loads the settings.
- void saveSettings ()

  Saves the settings.
- void loadParameters (QString path)

  Reads a parameter file, updates parameters.
- void level ()

  Level the parameters.

Private Attributes

- Ui::Interactive * **ui**
- QSettings * **settingsFile**
- int **currentLayout**
- QMap< QString, QString > **settings**
- QLabel * **counterLabel**
- QAction * **replayAction**
- QString memoryDir
- QSize resizedFrame
- QSize originalImageSize
- QSize cropedImageSize
- QMap< QString, QString > parameters
- QString **path**
- QString backgroundPath
- QString dir
- Tracking * tracking
- UMat background
- bool isBackground
- QPair< QPoint, QPoint > **clicks**
- QPointF **panReferenceClick**
- QPointF **zoomReferencePosition**
- Rect **roi**
- QPixmap **resizedPix**
- vector< Point3i > **colorMap**
- double **currentZoom**
- Replay * **replay**
- VideoReader * **video**
- bool **videoStatus**

4.8.1 Detailed Description

The Interactive widget provides an environment to use the tracking widget in an interactive environment.

Author

Benjamin Gallois

Version

Revision

480

Contact: benjamin.gallois@fasttrack.sh

4.8.2 Member Function Documentation

4.8.2.1 display

```
void Interactive::display (
            int index,
            int scale = 0 )  [private], [slot]
```

Displays the image at index in the image sequence in the ui.

Parameters

in	*index*	Index of the image to display in the image sequence.
in	*scale*	Optional scale to display.

4.8.2.2 eventFilter

```
bool Interactive::eventFilter (
            QObject * target,
            QEvent * event )  [private], [slot]
```

Manages all the mouse inputs in the display.

Parameters

in	*target*	Widget to apply the filter.
in	*event*	Describes the mouse event.

4.8.3 Member Data Documentation

4.8.3.1 background

```
UMat Interactive::background  [private]
```

Background image.

4.8.3.2 backgroundPath

```
QString Interactive::backgroundPath  [private]
```

Path to the background image.

4.8.3.3 cropedImageSize

```
QSize Interactive::cropedImageSize  [private]
```

Size of the croped image.

4.8.3.4 dir

```
QString Interactive::dir  [private]
```

Path to the folder where the image sequence to display is stored.

4.8.3.5 isBackground

```
bool Interactive::isBackground  [private]
```

Is the background computed.

4.8.3.6 memoryDir

```
QString Interactive::memoryDir  [private]
```

Saves the path to the last opened folder in dialog.

4.8.3.7 originalImageSize

```
QSize Interactive::originalImageSize  [private]
```

Size of the original image.

4.8.3.8 parameters

```
QMap<QString, QString> Interactive::parameters  [private]
```

Tracking parameters.

4.8.3.9 resizedFrame

```
QSize Interactive::resizedFrame  [private]
```

Size of the resized image in the display QWidget.

4.8.3.10 tracking

```
Tracking* Interactive::tracking  [private]
```

Tracking object.

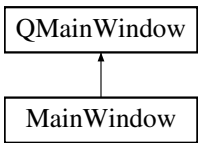
The documentation for this class was generated from the following files:

- src/interactive.h
- src/interactive.cpp

4.9 MainWindow Class Reference

The MainWindow class is derived from a QMainWindow widget. It displays the main window of the program.

Inheritance diagram for MainWindow:



Public Member Functions

- MainWindow (QWidget *parent=0)

  Constructs the MainWindow QObject and initializes the UI.
- ~MainWindow ()

  Destructs the MainWindow object and saves the previous set of parameters.

Private Member Functions

- void closeEvent (QCloseEvent *event)

  Close event reimplemented to ask confirmation before closing.

Private Attributes

- Ui::MainWindow * ui
- Interactive * **interactive**
- Batch * **batch**
- Replay * **replay**
- TrackingManager * **trackingManager**

4.9.1 Detailed Description

The MainWindow class is derived from a QMainWindow widget. It displays the main window of the program.

Author

Benjamin Gallois

Version

Revision

4.0

Contact: gallois.benjamin08@gmail.com

4.9.2 Member Data Documentation

4.9.2.1 ui

```
Ui::MainWindow* MainWindow::ui  [private]
```

ui file from Qt designer.

The documentation for this class was generated from the following files:

- src/mainwindow.h
- src/mainwindow.cpp

4.10 object Struct Reference

Public Attributes

- int **id**
- QMap< QString, double > **data**

The documentation for this struct was generated from the following file:

- src/data.h

4.11 Batch::process Struct Reference

Public Attributes

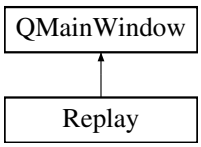
- QString **path**
- QString **backgroundPath**
- QMap< QString, QString > **trackingParameters**

The documentation for this struct was generated from the following file:

- src/batch.h

4.12 Replay Class Reference

Inheritance diagram for Replay:



Public Slots

- void openReplayFolder ()

  Opens a dialogue to select a folder.
- void loadReplayFolder (QString dir)

  Loads a folder containing an image sequence and the tracking data if it exists. Triggerred when ui->pathButton is pressed.
- void loadFrame (int frameIndex)

  Displays the image and the tracking data in the ui->displayReplay. Triggered when the ui->replaySlider value is changed.
- void zoomIn ()

  Zooms in the display.
- void zoomOut ()

Zooms out the display.

- bool eventFilter (QObject *target, QEvent *event)

  Manages all the mouse input in the display.

- void updateInformation (int objectId, int imageIndex, QTableWidget *table)

  Update the information of an object inside a table widget.

- void correctTracking ()

  Gets the index of the two selected objects, the start index, swaps the data from the start index to the end, and saves the new tracking data. Triggered when ui->swapButton is pressed or a right-click event is registered inside the replayDisplay.

- void nextOcclusionEvent ()

  Finds and displays the next occlusion event on the ui->replayDisplay. Triggered when ui->nextReplay is pressed.

- void previousOcclusionEvent ()

  Finds and displays the previous occlusion event on the ui->replayDisplay. Triggered when ui->previousReplay is pressed.

- void saveTrackedMovie ()

  Saves the tracked movie in .avi. Triggered when ui->previousReplay is pressed.

Public Member Functions

- **Replay** (QWidget *parent=nullptr, bool standalone=true)

Private Attributes

- Ui::Replay * **ui**
- QShortcut * deletedFrameFocus
- QUndoStack * **commandStack**
- QAction * **undoAction**
- QAction * **redoAction**
- QComboBox * **object1Replay**
- QComboBox * **object2Replay**
- QSpinBox * **deletedFrameNumber**
- QString memoryDir
- Data * **trackingData**
- Annotation * **annotation**
- vector< Point3i > colorMap
- QVector< int > occlusionEvents
- int replayNumberObject
- bool isReplayable
- int replayFps
- int autoPlayerIndex
- bool object
- QSize resizedFrame
- QSize originalImageSize
- int currentIndex
- double **currentZoom**
- QPointF **panReferenceClick**
- QPointF **zoomReferencePosition**
- QList< int > **ids**
- VideoReader * **video**

4.12.1 Detailed Description

Author

Benjamin Gallois

Version

Revision

4.1

Contact: gallois.benjamin08@gmail.com

4.12.2 Member Function Documentation

4.12.2.1 eventFilter

```
bool Replay::eventFilter (
            QObject * target,
            QEvent * event )  [slot]
```

Manages all the mouse input in the display.

Parameters

in	*target*	Target widget to apply the filter.
in	*event*	Describes the mouse event.

4.12.2.2 loadReplayFolder

```
void Replay::loadReplayFolder (
            QString dir )  [slot]
```

Loads a folder containing an image sequence and the tracking data if it exists. Triggerred when ui->pathButton is pressed.

- [in] dir Path to the folder where the image sequence is stored.

4.12.2.3 updateInformation

```
void Replay::updateInformation (
            int objectId,
            int imageIndex,
            QTableWidget * table )  [slot]
```

Update the information of an object inside a table widget.

Parameters

in	*objectId*	The id of the object to display the data.
in	*imageIndex*	The index of the image where to extracts the data.
in	*table*	Pointer to a QTableWidget where to display the data.

4.12.3 Member Data Documentation

4.12.3.1 autoPlayerIndex

```
int Replay::autoPlayerIndex  [private]
```

Index of the image displayed in autoplay mode in the replay.

4.12.3.2 colorMap

```
vector<Point3i> Replay::colorMap  [private]
```

RGB color map to display each object in one color.

4.12.3.3 currentIndex

```
int Replay::currentIndex  [private]
```

Current image index.

4.12.3.4 deletedFrameFocus

```
QShortcut* Replay::deletedFrameFocus  [private]
```

Keyboard shortcut to next frame.

4.12.3.5 isReplayable

```
bool Replay::isReplayable  [private]
```

True if user input is an images sequences that can be played.

4.12.3.6 memoryDir

```
QString Replay::memoryDir  [private]
```

Saves the path of the last opened folder.

4.12.3.7 object

```
bool Replay::object  [private]
```

Alternatively true or false to associate either object A or object B at each click of the user in the ui->replayDisplay.

4.12.3.8 occlusionEvents

```
QVector<int> Replay::occlusionEvents  [private]
```

Index of each occlusion event in the replayed images sequence.

4.12.3.9 originalImageSize

```
QSize Replay::originalImageSize  [private]
```

Width and height of the original image in the images sequence.

4.12.3.10 replayFps

```
int Replay::replayFps  [private]
```

Frame rate value at which a new image is displayed in autoplay mode in the replay.

4.12.3.11 replayNumberObject

```
int Replay::replayNumberObject  [private]
```

Number of objects tracked in the replayed images sequence.

4.12.3.12 resizedFrame

```
QSize Replay::resizedFrame  [private]
```

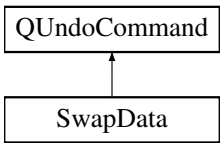
Width and height of displayed QPixmap to accomodate window size changment.

The documentation for this class was generated from the following files:

- src/replay.h
- src/replay.cpp

4.13 SwapData Class Reference

Inheritance diagram for SwapData:



Public Member Functions

- **SwapData** (int firstObject, int secondObject, int from, Data ∗data)
- void **undo** () override
- void **redo** () override

Private Attributes

- int **m_firstObject**
- int **m_secondObject**
- int **m_from**
- Data ∗ **m_data**

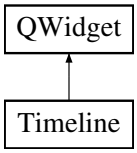
The documentation for this class was generated from the following files:

- src/data.h
- src/data.cpp

4.14 Timeline Class Reference

Draw a time line with cursor, hover and marker set.

Inheritance diagram for Timeline:



Signals

- void **valueChanged** (int value)

Public Member Functions

- **Timeline** (QWidget *parent=nullptr)
- void setValue (const int index)

  Set the left cursor (left click cursor) at a given value.
- void setCursorValue (const int index)

  Set the cursor at a given value.
- void setMaximum (const int max)

  Set the maximum value.
- void setMinimum (const int min)

  Set the minimum value, currently forced to zero.
- int value ()

  Return the last left value.
- int currentValue ()

  Return the current value.
- void togglePlay ()

  Start/Stop the autoplay of the replay.

Private Member Functions

- void setLayout (const int width, const int imageNumber)

  Set the layout of the timeline.
- void resizeEvent (QResizeEvent *event)

  Handle the widget redrawing when resized.
- bool eventFilter (QObject *target, QEvent *event)

  Handle the pointer event, click and hover.
- void drawMarker (const int index)

  Draw a line marker at a given index.
- void clearMarker (const int index)

  Delete a line marker at a given index.
- void update (const int index)

  Redraw the widget keeping markers and cursors.

Private Attributes

- Ui::Timeline * **ui**
- int **m_imageNumber**
- int **m_imageMin**
- int **m_width**
- int **m_offset**
- int **m_currentIndex**
- int **m_currentIndexLeft**
- int **m_scale**
- QTimer * **timer**
- QGraphicsScene * **timelineScene**
- QGraphicsLineItem * **cursor**
- QGraphicsLineItem * **cursorLeft**
- QGraphicsSimpleTextItem * **indexNumber**
- QVector< int > **markers**

4.14.1 Detailed Description

Draw a time line with cursor, hover and marker set.

Author

Benjamin Gallois

Version

Revision

5.0

Contact: benjamin.gallois@fasttrack.sh

4.14.2 Member Function Documentation

4.14.2.1 clearMarker()

```
void Timeline::clearMarker (
            const int index )  [private]
```

Delete a line marker at a given index.

Parameters

in	*index*	Index.

4.14.2.2 drawMarker()

```
void Timeline::drawMarker (
            const int index )  [private]
```

Draw a line marker at a given index.

Parameters

in	*index*	Index.

4.14.2.3 eventFilter()

```
bool Timeline::eventFilter (
            QObject * target,
            QEvent * event )  [private]
```

Handle the pointer event, click and hover.

Parameters

in	*target	Pointer to the target widget.
in	*event	Pointer to the event.

4.14.2.4 resizeEvent()

```
void Timeline::resizeEvent (
            QResizeEvent * event )  [private]
```

Handle the widget redrawing when resized.

Parameters

in	*event	Pointer to the event.

4.14.2.5 setCursorValue()

```
void Timeline::setCursorValue (
            const int index )
```

Set the cursor at a given value.

Parameters

in	*index*	Index.

4.14.2.6 setLayout()

```
void Timeline::setLayout (
            const int width,
            const int imageNumber )  [private]
```

Set the layout of the timeline.

Parameters

in	*width*	The width of the widget.
in	*imageNumber*	The number of images.

4.14.2.7 setMaximum()

```
void Timeline::setMaximum (
            const int max )
```

Set the maximum value.

Parameters

in	*max*	Maximum value.

4.14.2.8 setMinimum()

```
void Timeline::setMinimum (
            const int min )
```

Set the minimum value, currently forced to zero.

Parameters

in	*max*	minimum value.

4.14.2.9 setValue()

```
void Timeline::setValue (
            const int index )
```

Set the left cursor (left click cursor) at a given value.

Parameters

in	*index*	Index.

4.14.2.10 update()

```
void Timeline::update (
            const int index )  [private]
```

Redraw the widget keeping markers and cursors.

Parameters

in	*index*	Index.

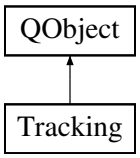
The documentation for this class was generated from the following files:

- src/timeline.h
- src/timeline.cpp

4.15 Tracking Class Reference

This class is intended to execute a tracking analysis on an image sequence. It is initialized with the path to the folder where images are stored. This class can be used inside an application by creating a new thread and calling the method startProcess. The tracking workflow can be changed by reimplementing the method startProcess and imageProcessing. This class can also be used as a library by constructing Tracking tracking("", "") to access the public class members and builds a new workflow.

Inheritance diagram for Tracking:



Public Slots

- void startProcess ()

  Initializes a tracking analysis and triggers its execution. Constructs from the path to a folder where the image sequence is stored, detects the image format and processes the first image to detect objects. First, it computes the background by averaging images from the sequence, then it subtracts the background from the first image and then binarizes the resulting image. It detects the objects by contour analysis and extracts features by computing the object moments. It triggers the analysis of the second image of the sequence.
- void updatingParameters (const QMap< QString, QString > &)

  Updates the private members from the external parameters. This function links the tracking logic with the graphical user interface.
- void imageProcessing ()

  Processes an image from an images sequence and tracks and matchs objects according to the previous image in the sequence. Takes a new image from the image sequence, substracts the background, binarises the image and crops according to the defined region of interest. Detects all the objects in the image and extracts the object features. Then matches detected objects with objects from the previous frame. This function emits a signal to display the images in the user interface.

Signals

- void progress (int)

  Emitted when an image is processed.
- void backgroundProgress (int)

  Emitted when an image to compute the background is processed.
- void finishedProcessFrame ()

  Emitted when the first image has been processed to trigger the starting of the analysis.
- void finished ()

  Emitted when all images have been processed.
- void forceFinished ()

  Emitted when a crash occurs during the analysis.
- void error (int code)

  Emitted when an error occurs.
- void statistic (long long int time)

  Emitted at the end of the analysis.

Public Member Functions

- Tracking (string path, string background, int startImage=0, int stopImage=-1)

  Constructs the tracking object from a path to an image sequence and an optional path to a background image.
- Tracking (string path, UMat background, int startImage=0, int stopImage=-1)

  Constructs the tracking object from a list of path, a background image and a range of image.
- ~Tracking ()

  Destructs the tracking object.
- Point2d curvatureCenter (const Point3d &tail, const Point3d &head)

  Computes the center of the curvature, defined as the intersection of the minor axis of the head ellipse with the minor axis of the tail ellipse of the object.
- double curvature (Point2d center, const Mat &image)

  Computes the radius of curvature of the object defined as the inverse of the mean distance between each pixel of the object, and the center of the curvature. The center of curvature is defined as the intersection of the two minor axes of the head and tail ellipse.
- double divide (double a, double b)

  Computes the float division and handle the division by 0 by returning 0.
- bool objectDirection (const UMat &image, vector< double > &information)

  Computes the direction of the object from the object parameter (coordinate of the center of mass and orientation). To use this function, the object major axis has to be the horizontal axis of the image. Therefore, it is necessary to rotate the image before calling objectDirection.
- vector< double > objectInformation (const UMat &image)

  Computes the equivalent ellipse of an object by computing the moments of the image. If the image is a circle, return nan as the orientation.
- vector< Point3d > reassignment (const vector< Point3d > &past, const vector< Point3d > &input, const vector< int > &assignment)

  Sorts a vector accordingly to a new set of indexes. The sorted vector at index i is the input at index assignment[i].
- UMat backgroundExtraction (VideoReader &video, int n, const int method, const int registrationMethod)

  Computes the background of an image sequence by averaging n images.
- void registration (UMat imageReference, UMat &frame, int method)

  Register two images. To speed-up, the registration is made in a pyramidal way: the images are downsampled then registered to have a an approximate transformation then upslampled to have the precise transformation.
- void binarisation (UMat &frame, char backgroundColor, int value)

  Binarizes the image by thresholding.
- vector< vector< Point3d > > objectPosition (const UMat &frame, int minSize, int maxSize)

Computes the positions of the objects and extracts the object's features.

- vector< int > costFunc (const vector< vector< Point3d >> &prevPos, const vector< vector< Point3d >> &pos, double LENGHT, double ANGLE, double LO, double AREA, double PERIMETER)

  Computes a cost function and use a global optimization association to associate targets between images. Method adapted from: "An effective and robust method for Tracking multiple fish in video image based on fish head detection" YQ Chen et al. Uses the Hungarian method implemented by Cong Ma, 2016 "https://github.com/mcximing/hungarian-algorithm-cpp" adapted from the Matlab implementation by Markus Buehren "https://fr.mathworks.com/matlabcentral/fileexchange/6543-functions-for-the-rectangular-assignment-problem".
- void cleaning (const vector< int > &occluded, vector< int > &lostCounter, vector< int > &id, vector< vector< Point3d >> &input, double param_maximalTime)

  Cleans the data if an object is lost more than a certain time.
- vector< Point3d > prevision (vector< Point3d > past, vector< Point3d > present)

  Predicts the next position of an object from the previous position.
- vector< Point3i > color (int number)

  Computes a random set of colors.
- vector< int > findOcclusion (vector< int > assignment)

  Finds the objects that are occluded during the tracking.

Static Public Member Functions

- static double modul (double angle)

  Computes the usual mathematical modulo 2∗PI of an angle.
- static double angleDifference (double alpha, double beta)

  Computes the least difference between two angles, alpha - beta. The difference is oriented in the trigonometric convention.

Public Attributes

- UMat m_binaryFrame
- UMat m_visuFrame
- vector< vector< Point3d > > m_out
- vector< vector< Point3d > > m_outPrev

Private Attributes

- QElapsedTimer ∗ timer
- UMat m_background
- bool statusBinarisation
- VideoReader ∗ **video**
- int m_im
- int m_startImage
- int m_stopImage
- Rect m_ROI
- QTextStream m_savefile
- QFile m_outputFile
- vector< cv::String > m_files
- vector< Point3i > m_colorMap
- vector< vector< Point > > m_memory
- vector< int > m_id
- vector< int > m_lost
- int **m_idMax**

- string m_path
- string m_backgroundPath
- int m_displayTime
- int param_n
- int param_maxArea
- int param_minArea
- int param_spot
- double param_len
- double param_angle
- double param_area
- double param_perimeter
- double param_lo
- double param_to
- int param_thresh
- double param_nBackground
- int param_methodBackground
- int param_methodRegistrationBackground
- int param_registration
- int param_x1
- int param_y1
- int param_x2
- int param_y2
- int param_kernelSize
- int param_kernelType
- int param_morphOperation
- QMap< QString, QString > parameters

4.15.1 Detailed Description

This class is intended to execute a tracking analysis on an image sequence. It is initialized with the path to the folder where images are stored. This class can be used inside an application by creating a new thread and calling the method startProcess. The tracking workflow can be changed by reimplementing the method startProcess and imageProcessing. This class can also be used as a library by constructing Tracking tracking("", "") to access the public class members and builds a new workflow.

Author

Benjamin Gallois

Version

Revision

4.0

Contact: gallois.benjamin08@gmail.com

4.15.2 Constructor & Destructor Documentation

4.15.2.1 Tracking() [1/2]

```
Tracking::Tracking (
            string path,
            string backgroundPath,
            int startImage = 0,
            int stopImage = -1 )
```

Constructs the tracking object from a path to an image sequence and an optional path to a background image.

Parameters

in	*path*	The path to a folder where images are stocked.
in	*backgroundPath*	The path to a background image.
in	*startImage*	Index of the beginning image.
in	*stopImage*	Index of the ending image.

4.15.2.2 Tracking() [2/2]

```
Tracking::Tracking (
            string path,
            UMat background,
            int startImage = 0,
            int stopImage = -1 )
```

Constructs the tracking object from a list of path, a background image and a range of image.

Parameters

in	*imagePath*	List of path to the images.
in	*background*	A background image.
in	*startImage*	Index of the beginning image.
in	*stopImage*	Index of the ending image.

4.15.3 Member Function Documentation

4.15.3.1 angleDifference()

```
double Tracking::angleDifference (
            double alpha,
            double beta )  [static]
```

Computes the least difference between two angles, alpha - beta. The difference is oriented in the trigonometric convention.

Parameters

in	*alpha*	Input angle.
in	*beta*	Input angle.

Returns

Least difference.

4.15.3.2 backgroundExtraction()

```
UMat Tracking::backgroundExtraction (
            VideoReader & video,
            int n,
            const int method,
            const int registrationMethod )
```

Computes the background of an image sequence by averaging n images.

Parameters

in	*files*	List of paths to each image in the images sequence.
in	*n*	The number of images to average to computes the background.
in	*Method*	0: minimal projection, 1: maximal projection, 2: average projection.

Returns

The background image. TO DO: currently opening all the frames, to speed-up the process and if step is large can skip frames by replacing nextImage by getImage.

4.15.3.3 backgroundProgress

```
void Tracking::backgroundProgress (
            int  )  [signal]
```

Emitted when an image to compute the background is processed.

Parameters

int	The number of processed image.

4.15.3.4 binarisation()

```
void Tracking::binarisation (
            UMat & frame,
            char backgroundColor,
            int value )
```

Binarizes the image by thresholding.

Parameters

in,out	*frame*	The image to binarize.
in	*backgroundColor*	If equals to 'w' the thresholded image will be inverted, if equal to 'b' it will not be inverted.
in	*value*	The value at which to threshold the image.

4.15.3.5 cleaning()

```
void Tracking::cleaning (
            const vector< int > & occluded,
            vector< int > & lostCounter,
            vector< int > & id,
            vector< vector< Point3d >> & input,
            double param_maximalTime )
```

Cleans the data if an object is lost more than a certain time.

Parameters

in	*occluded*	The vector with the index of object missing in the current image.
in	*input*	The vector at current image of size m <= n to be sorted.
in	*lostCounter*	The vector with the number of times each objects are lost consecutively.
in	*id*	The vector with the id of the objects.
in	*param_maximalTime*	

Returns

The sorted vector.

4.15.3.6 color()

```
vector< Point3i > Tracking::color (
            int number )
```

Computes a random set of colors.

Parameters

in	*number*	The number of colors to generate.

Returns

The vector containing the n colors.

4.15.3.7 costFunc()

```
vector< int > Tracking::costFunc (
            const vector< vector< Point3d >> & prevPos,
            const vector< vector< Point3d >> & pos,
            double LENGTH,
            double ANGLE,
            double LO,
            double AREA,
            double PERIMETER )
```

Computes a cost function and use a global optimization association to associate targets between images. Method adapted from: "An effective and robust method for Tracking multiple fish in video image based on fish head detection" YQ Chen et al. Uses the Hungarian method implemented by Cong Ma, 2016 "https://github.↩ com/mcximing/hungarian-algorithm-cpp" adapted from the Matlab implementation by Markus Buehren "https://fr.↩ mathworks.com/matlabcentral/fileexchange/6543-functions-for-the-rectangular-assignment-problem".

Parameters

in	*prevPos*	The vector of objects parameters at the previous image.
in	*pos*	The vector of objects parameters at the current image that we want to sort in order to conserve objects identity.
in	*LENGTH*	The typical displacement of an object in pixels.
in	*ANGLE*	The typical reorientation angle in radians.
in	*LO*	The maximal assignment distance in pixels.

Returns

The assignment vector containing the new index position to sort the pos vector.

4.15.3.8 curvature()

```
double Tracking::curvature (
            Point2d center,
            const Mat & image )
```

Computes the radius of curvature of the object defined as the inverse of the mean distance between each pixel of the object, and the center of the curvature. The center of curvature is defined as the intersection of the two minor axes of the head and tail ellipse.

Parameters

in	*center*	Center of the curvature.
in	*image*	Binary image CV_8U.

Returns

Radius of curvature.

4.15.3.9 curvatureCenter()

```
Point2d Tracking::curvatureCenter (
            const Point3d & tail,
            const Point3d & head )
```

Computes the center of the curvature, defined as the intersection of the minor axis of the head ellipse with the minor axis of the tail ellipse of the object.

Parameters

in	*tail*	The parameters of the tail ellipse: coordinate and direction of the major axis.
in	*head*	The parameters of the head ellipse: coordinate and direction of the major axis.

Returns

Coordinate of the curvature center.

4.15.3.10 divide()

```
double Tracking::divide (
            double a,
            double b )
```

Computes the float division and handle the division by 0 by returning 0.

Parameters

in	*a*	Dividend.
in	*a*	Divisor.

Returns

Division result, 0 if b = 0.

4.15.3.11 findOcclusion()

```
vector< int > Tracking::findOcclusion (
            vector< int > assignment )
```

Finds the objects that are occluded during the tracking.

Parameters

in	*assignment*	The vector with the new indexes that will be used to sort the input vector.

Returns

The vector with the indexes of occluded objects.

4.15.3.12 modul()

```
double Tracking::modul (
            double angle )  [static]
```

Computes the usual mathematical modulo 2∗PI of an angle.

Parameters

in	*angle*	Input angle.

Returns

Output angle.

4.15.3.13 objectDirection()

```
bool Tracking::objectDirection (
            const UMat & image,
            vector< double > & information )
```

Computes the direction of the object from the object parameter (coordinate of the center of mass and orientation). To use this function, the object major axis has to be the horizontal axis of the image. Therefore, it is necessary to rotate the image before calling objectDirection.

Parameters

in	*image*	Binary image CV_8U.
in,out	*information*	The parameters of the object (x coordinate, y coordinate, orientation).

Returns

True if the direction angle is the orientation angle. False if the direction angle is the orientation angle plus pi.

4.15.3.14 objectInformation()

```
vector< double > Tracking::objectInformation (
            const UMat & image )
```

Computes the equivalent ellipse of an object by computing the moments of the image. If the image is a circle, return nan as the orientation.

Parameters

in	*image*	Binary image CV_8U.

Returns

The equivalent ellipse parameters: the object center of mass coordinate and its orientation.

Note

: This function computes the object orientation, not its direction.

4.15.3.15 objectPosition()

```
vector< vector< Point3d > > Tracking::objectPosition (
            const UMat & frame,
            int minSize,
            int maxSize )
```

Computes the positions of the objects and extracts the object's features.

Parameters

in	*frame*	Binary image CV_8U.
in	*minSize*	The minimal size of an object.
in	*maxSize*	The maximal size of an object.

Returns

All the parameters of all the objects formated as follows: one vector, inside of this vector, four vectors for parameters of the head, tail, body and features with number of object size. { { Point(xHead, yHead, theta↩ Head), ...}, Point({xTail, yTail, thetaHead), ...}, {Point(xBody, yBody, thetaBody), ...}, {Point(curvature, 0, 0), ...}}

4.15.3.16 prevision()

```
vector< Point3d > Tracking::prevision (
            vector< Point3d > past,
            vector< Point3d > present )
```

Predicts the next position of an object from the previous position.

Parameters

past	The previous position parameters.
present	The current position parameters.

Returns

The predicted positions.

4.15.3.17 progress

```
void Tracking::progress (
            int  )  [signal]
```

Emitted when an image is processed.

Parameters

int	Index of the processed image.

4.15.3.18 reassignment()

```
vector< Point3d > Tracking::reassignment (
            const vector< Point3d > & past,
            const vector< Point3d > & input,
            const vector< int > & assignment )
```

Sorts a vector accordingly to a new set of indexes. The sorted vector at index i is the input at index assignment[i].

Parameters

in	*past*	The vector at the previous image.
in	*input*	The vector at current image of size m <= n to be sorted.
in	*assignment*	The vector with the new indexes that will be used to sort the input vector.
in	*lostCounter*	The vector with the number of times each objects are lost consecutively.
in	*id*	The vector with the id of the objects.

Returns

The sorted vector.

4.15.3.19 registration()

```
void Tracking::registration (
            UMat imageReference,
            UMat & frame,
            int method )
```

Register two images. To speed-up, the registration is made in a pyramidal way: the images are downsampled then registered to have a an approximate transformation then upslampled to have the precise transformation.

Parameters

in	*imageReference*	The reference image for the registration.
in,out	*frame*	The image to register.
in	*method*	The method of registration: 0 = simple (phase correlation), 1 = ECC, 2 = Features based.

4.15.3.20 updatingParameters

```
void Tracking::updatingParameters (
            const QMap< QString, QString > & parameterList )  [slot]
```

Updates the private members from the external parameters. This function links the tracking logic with the graphical user interface.

Parameters

in	*parameterList*	The list of all the parameters used in the tracking.

4.15.4 Member Data Documentation

4.15.4.1 m_background

```
UMat Tracking::m_background  [private]
```

Background image CV_8U.

4.15.4.2 m_backgroundPath

```
string Tracking::m_backgroundPath  [private]
```

Path to an image background.

4.15.4.3 m_binaryFrame

```
UMat Tracking::m_binaryFrame
```

Binary image CV_8U

4.15.4.4 m_colorMap

```
vector<Point3i> Tracking::m_colorMap  [private]
```

Vector containing RBG color.

4.15.4.5 m_displayTime

```
int Tracking::m_displayTime  [private]
```

Binary image CV_8U.

4.15.4.6 m_files

```
vector<cv::String> Tracking::m_files  [private]
```

Vector containing the path for each image in the images sequence.

4.15.4.7 m_id

```
vector<int> Tracking::m_id  [private]
```

Vector containing the objets Id.

4.15.4.8 m_im

```
int Tracking::m_im  [private]
```

Index of the next image to process in the m_files list.

4.15.4.9 m_lost

```
vector<int> Tracking::m_lost  [private]
```

Vector containing the lost objects.

4.15.4.10 m_memory

```
vector<vector<Point> > Tracking::m_memory  [private]
```

Vector containing the last 50 tracking data.

4.15.4.11 m_out

```
vector<vector<Point3d> > Tracking::m_out
```

Objects information at iteration minus one

4.15.4.12 m_outPrev

```
vector<vector<Point3d> > Tracking::m_outPrev
```

Objects information at current iteration

4.15.4.13 m_outputFile

```
QFile Tracking::m_outputFile  [private]
```

Path to the file where to save tracking data.

4.15.4.14 m_path

```
string Tracking::m_path  [private]
```

Path to an image sequence.

4.15.4.15 m_ROI

```
Rect Tracking::m_ROI  [private]
```

Rectangular region of interest.

4.15.4.16 m_savefile

```
QTextStream Tracking::m_savefile  [private]
```

Stream to output tracking data.

4.15.4.17 m_startImage

```
int Tracking::m_startImage  [private]
```

Index of the next image to process in the m_files list.

4.15.4.18 m_stopImage

```
int Tracking::m_stopImage  [private]
```

Index of the next image to process in the m_files list.

4.15.4.19 m_visuFrame

```
UMat Tracking::m_visuFrame
```

Image 8 bit CV_8U

4.15.4.20 param_angle

```
double Tracking::param_angle  [private]
```

Maximal change in direction of an object between two images.

4.15.4.21 param_area

```
double Tracking::param_area  [private]
```

Normalization area.

4.15.4.22 param_kernelSize

```
int Tracking::param_kernelSize  [private]
```

Size of the kernel of the morphological operation.

4.15.4.23 param_kernelType

```
int Tracking::param_kernelType  [private]
```

Type of the kernel of the morphological operation.

4.15.4.24 param_len

```
double Tracking::param_len  [private]
```

Maximal length travelled by an object between two images.

4.15.4.25 param_lo

```
double Tracking::param_lo  [private]
```

Maximal distance allowed by an object to travel during an occlusion event.

4.15.4.26 param_maxArea

```
int Tracking::param_maxArea  [private]
```

Maximal area of an object.

4.15.4.27 param_methodBackground

```
int Tracking::param_methodBackground  [private]
```

The method used to compute the background.

4.15.4.28 param_methodRegistrationBackground

```
int Tracking::param_methodRegistrationBackground  [private]
```

The method used to register the images for the background.

4.15.4.29 param_minArea

```
int Tracking::param_minArea  [private]
```

Minimal area of an object.

4.15.4.30 param_morphOperation

```
int Tracking::param_morphOperation  [private]
```

Type of the morphological operation.

4.15.4.31 param_n

```
int Tracking::param_n  [private]
```

Number of objects.

4.15.4.32 param_nBackground

```
double Tracking::param_nBackground  [private]
```

Number of images to average to compute the background.

4.15.4.33 param_perimeter

```
double Tracking::param_perimeter  [private]
```

Normalization perimeter.

4.15.4.34 param_registration

```
int Tracking::param_registration  [private]
```

Method of registration.

4.15.4.35 param_spot

```
int Tracking::param_spot  [private]
```

Which spot parameters are used to computes the cost function. 0: head, 1: tail, 2: body.

4.15.4.36 param_thresh

```
int Tracking::param_thresh  [private]
```

Value of the threshold to binarize the image.

4.15.4.37 param_to

```
double Tracking::param_to  [private]
```

Maximal time.

4.15.4.38 param_x1

```
int Tracking::param_x1  [private]
```

Top x corner of the region of interest.

4.15.4.39 param_x2

```
int Tracking::param_x2  [private]
```

Bottom x corner of the region of interest.

4.15.4.40 param_y1

```
int Tracking::param_y1  [private]
```

Top y corner of the region of interest.

4.15.4.41 param_y2

```
int Tracking::param_y2  [private]
```

Bottom y corner of the region of interest.

4.15.4.42 parameters

```
QMap<QString, QString> Tracking::parameters  [private]
```

map of all the parameters for the tracking.

4.15.4.43 statusBinarisation

```
bool Tracking::statusBinarisation  [private]
```

True if wite objects on dark background, flase otherwise.

4.15.4.44 timer

```
QElapsedTimer* Tracking::timer  [private]
```

Timer that measured the time during the analysis execution.

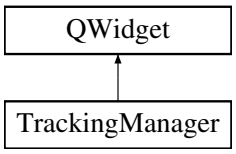
The documentation for this class was generated from the following files:

- src/tracking.h
- src/tracking.cpp

4.16 TrackingManager Class Reference

The TrackingManager widget provides an environment to manage the log of FastTrack tracking analysis.

Inheritance diagram for TrackingManager:



Public Slots

- void addLogEntry (QMap< QString, QString > log)

  Adds a log entry inside the ui table and in a log file.
- void appendToFile (QString path, QMap< QString, QString > line)

  Appends the log entry in a file.
- void writeToFile (QString path, QList< QMap< QString, QString >> lines)

  Writes log entries in a file.
- void readFromFile (QString path)

  Reads a log file.

Public Member Functions

- TrackingManager (QWidget *parent=nullptr)

  Constructs the trackingmanager object derived from a QWidget object.

Private Attributes

- QString **logPath**
- Ui::TrackingManager * **ui**

4.16.1 Detailed Description

The TrackingManager widget provides an environment to manage the log of FastTrack tracking analysis.

Author

Benjamin Gallois

Version

Revision

490

Contact: benjamin.gallois@fasttrack.sh

4.16.2 Member Function Documentation

4.16.2.1 addLogEntry

```
void TrackingManager::addLogEntry (
            QMap< QString, QString > log )  [slot]
```

Adds a log entry inside the ui table and in a log file.

Parameters

in	*log*	QMap that contains the log entry.

4.16.2.2 appendToFile

```
void TrackingManager::appendToFile (
            QString path,
            QMap< QString, QString > line )  [slot]
```

Appends the log entry in a file.

Parameters

in	*path*	QString that contains the path to the output file.
in	*line*	QMap that contains the log entry.

4.16.2.3 readFromFile

```
void TrackingManager::readFromFile (
            QString path )  [slot]
```

Reads a log file.

Parameters

in	*path*	QString that contains the path to the input file.

4.16.2.4 writeToFile

```
void TrackingManager::writeToFile (
            QString path,
            QList< QMap< QString, QString >> lines )  [slot]
```

Writes log entries in a file.

Parameters

in	*path*	QString that contains the path to the output file.
in	*lines*	QList ofQMap that contains log entries.

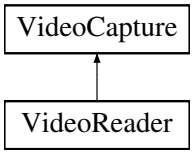
The documentation for this class was generated from the following files:

- src/trackingmanager.h
- src/trackingmanager.cpp

4.17 VideoReader Class Reference

This class is intended to abstract the opening of a video, it can load image sequence and video with the same public API.

Inheritance diagram for VideoReader:



Public Member Functions

- VideoReader (const string &path)

  Construct the VideoReader object from a path to a file that can be either an image from an image sequence or a movie file.
- bool getNext (UMat &destination)

  Get the next image, always one channel.
- bool getNext (Mat &destination)

  Get the next image, always one channel.
- bool getImage (int index, UMat &destination)

  Get the image at selected index, always one channel.
- bool getImage (int index, Mat &destination)

  Get the image at selected index, always one channel.
- unsigned int getImageCount () const

  Get the total number of images in the video.
- bool isSequence ()

  Is the file is an image sequence.

Private Attributes

- bool **m_isSequence**
- int **m_index**

4.17.1 Detailed Description

This class is intended to abstract the opening of a video, it can load image sequence and video with the same public API.

Author

Benjamin Gallois

Version

Revision

5.0

Contact: benjamin.gallois@fasttrack.sh

4.17.2 Constructor & Destructor Documentation

4.17.2.1 VideoReader()

```
VideoReader::VideoReader (
            const string & path )
```

Construct the VideoReader object from a path to a file that can be either an image from an image sequence or a movie file.

Parameters

in	*path*	Path to a video or image file.

4.17.3 Member Function Documentation

4.17.3.1 getImage() [1/2]

```
bool VideoReader::getImage (
            int index,
            Mat & destination )
```

Get the image at selected index, always one channel.

Parameters

in	*index*	Index of the image.
in	*destination*	Mat to store the image.

4.17.3.2 getImage() [2/2]

```
bool VideoReader::getImage (
            int index,
            UMat & destination )
```

Get the image at selected index, always one channel.

Parameters

in	*index*	Index of the image.
in	*destination*	UMat to store the image.

4.17.3.3 getImageCount()

```
unsigned int VideoReader::getImageCount ( ) const
```

Get the total number of images in the video.

Returns

total number of images.

4.17.3.4 getNext() [1/2]

```
bool VideoReader::getNext (
            Mat & destination )
```

Get the next image, always one channel.

Parameters

in	*destination*	UMat to store the image.

4.17.3.5 getNext() [2/2]

```
bool VideoReader::getNext (
            UMat & destination )
```

Get the next image, always one channel.

Parameters

in	*destination*	UMat to store the image.

4.17.3.6 isSequence()

```
bool VideoReader::isSequence ( )
```

Is the file is an image sequence.

Returns

True if an image sequence, false otherwise.

The documentation for this class was generated from the following files:

- src/videoreader.h
- src/videoreader.cpp

Index