Rigid Track

Generated by Doxygen 1.8.13

Contents

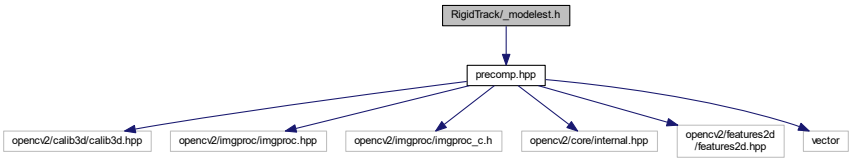
Chapter 1

File Documentation

1.1 RigidTrack/_modelest.h File Reference

```
#include "precomp.hpp"
```

Include dependency graph for _modelest.h:



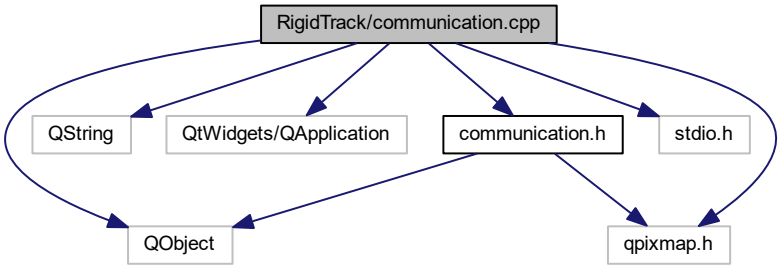
Classes

- class **CvModelEstimator2**

1.2 RigidTrack/communication.cpp File Reference

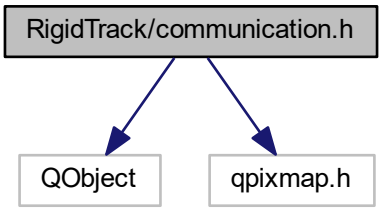
```
#include <QObject>
#include <QString>
#include <QtWidgets/QApplication>
#include <qpixmap.h>
#include <stdio.h>
```

```
#include "communication.h"
```
Include dependency graph for communication.cpp:

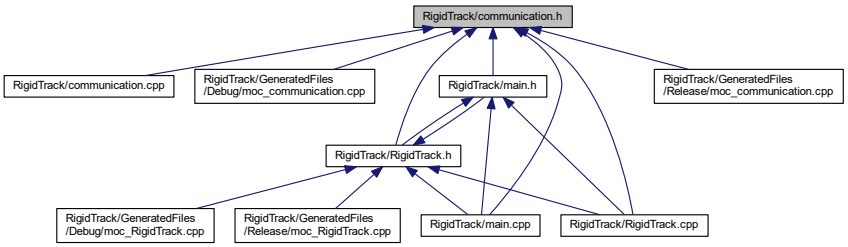


1.3 RigidTrack/communication.h File Reference

```
#include <QObject>
#include <qpixmap.h>
```
Include dependency graph for communication.h:



This graph shows which files directly or indirectly include this file:



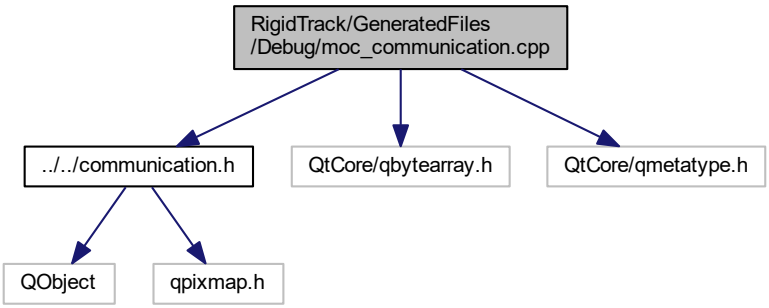
Classes

- class **commObject**

1.4 RigidTrack/GeneratedFiles/Debug/moc_communication.cpp File Reference

```
#include "../../communication.h"
#include <QtCore/qbytearray.h>
#include <QtCore/qmetatype.h>
```

Include dependency graph for moc_communication.cpp:



Classes

- struct **qt_meta_stringdata_commObject_t**

Macros

- #define **QT_MOC_LITERAL**(idx, ofs, len)

1.4.1 Macro Definition Documentation

1.4.1.1 QT_MOC_LITERAL

```
#define QT_MOC_LITERAL(
              idx,
              ofs,
              len )
```

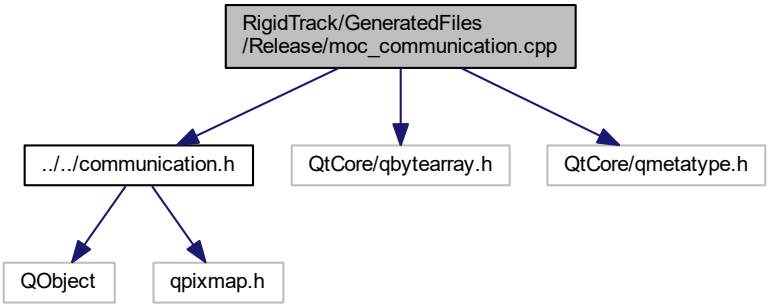
Value:

```
Q_STATIC_BYTE_ARRAY_DATA_HEADER_INITIALIZER_WITH_OFFSET(len, \
    qptrdiff(offsetof(qt_meta_stringdata_commObject_t, stringdata0) + ofs \
        - idx * sizeof(QByteArrayData)) \
    )
```

1.5 RigidTrack/GeneratedFiles/Release/moc_communication.cpp File Reference

```
#include "../../communication.h"
#include <QtCore/qbytearray.h>
#include <QtCore/qmetatype.h>
```

Include dependency graph for moc_communication.cpp:



Classes

- struct **qt_meta_stringdata_commObject_t**

Macros

- #define **QT_MOC_LITERAL**(idx, ofs, len)

1.5.1 Macro Definition Documentation

1.5.1.1 QT_MOC_LITERAL

```
#define QT_MOC_LITERAL(
              idx,
              ofs,
              len )
```

Value:

```
Q_STATIC_BYTE_ARRAY_DATA_HEADER_INITIALIZER_WITH_OFFSET(len, \
    qptrdiff(offsetof(qt_meta_stringdata_commObject_t, stringdata0) + ofs \
        - idx * sizeof(QByteArrayData)) \
    )
```

1.6 RigidTrack/GeneratedFiles/Debug/moc_RigidTrack.cpp File Reference

```
#include "../../RigidTrack.h"
#include <QtCore/qbytearray.h>
#include <QtCore/qmetatype.h>
```

Include dependency graph for moc_RigidTrack.cpp:

Classes

- struct **qt_meta_stringdata_RigidTrack_t**

Macros

- #define **QT_MOC_LITERAL**(idx, ofs, len)

1.6.1 Macro Definition Documentation

1.6.1.1 QT_MOC_LITERAL

```
#define QT_MOC_LITERAL(
              idx,
              ofs,
              len )
```

Value:

```
Q_STATIC_BYTE_ARRAY_DATA_HEADER_INITIALIZER_WITH_OFFSET(len, \
    qptrdiff(offsetof(qt_meta_stringdata_RigidTrack_t, stringdata0) + ofs \
        - idx * sizeof(QByteArrayData)) \
    )
```

1.7 RigidTrack/GeneratedFiles/Release/moc_RigidTrack.cpp File Reference

```
#include "../../RigidTrack.h"
#include <QtCore/qbytearray.h>
#include <QtCore/qmetatype.h>
```

Include dependency graph for moc_RigidTrack.cpp:

Classes

- struct **qt_meta_stringdata_RigidTrack_t**

Macros

- #define **QT_MOC_LITERAL**(idx, ofs, len)

1.7.1 Macro Definition Documentation

1.7.1.1 QT_MOC_LITERAL

```
#define QT_MOC_LITERAL(
              idx,
              ofs,
              len )
```

Value:

```
Q_STATIC_BYTE_ARRAY_DATA_HEADER_INITIALIZER_WITH_OFFSET(len, \
    qptrdiff(offsetof(qt_meta_stringdata_RigidTrack_t, stringdata0) + ofs \
        - idx * sizeof(QByteArrayData)) \
    )
```

1.8 RigidTrack/GeneratedFiles/qrc_RigidTrack.cpp File Reference

Macros

- #define **QT_RCC_PREPEND_NAMESPACE**(name) name
- #define **QT_RCC_MANGLE_NAMESPACE**(name) name

Functions

- int **QT_RCC_MANGLE_NAMESPACE**() **qInitResources_RigidTrack** ()
- int **QT_RCC_MANGLE_NAMESPACE**() **qCleanupResources_RigidTrack** ()

1.8.1 Macro Definition Documentation

1.8.1.1 QT_RCC_MANGLE_NAMESPACE

```
#define QT_RCC_MANGLE_NAMESPACE(
              name ) name
```

1.8.1.2 QT_RCC_PREPEND_NAMESPACE

```
#define QT_RCC_PREPEND_NAMESPACE(
              name ) name
```

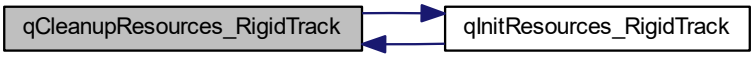
1.8.2 Function Documentation

1.8.2.1 qCleanupResources_RigidTrack()

```
int  QT_RCC_MANGLE_NAMESPACE() qCleanupResources_RigidTrack ( )
```

Here is the call graph for this function:

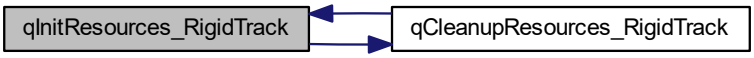
Here is the caller graph for this function:



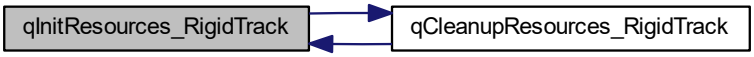
1.8.2.2 qInitResources_RigidTrack()

```
int  QT_RCC_MANGLE_NAMESPACE() qInitResources_RigidTrack ( )
```

Here is the call graph for this function:



Here is the caller graph for this function:

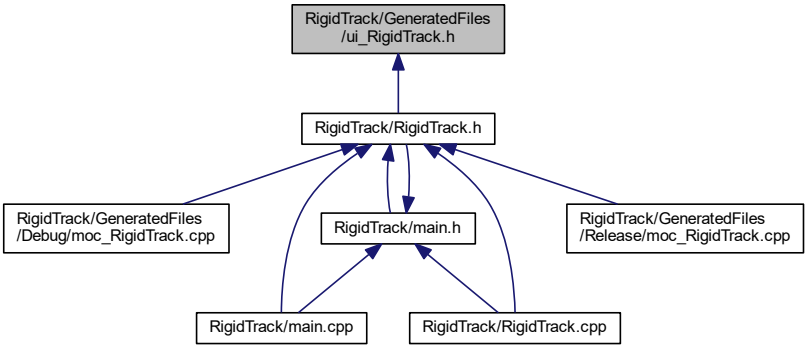


1.9 RigidTrack/GeneratedFiles/ui_RigidTrack.h File Reference

```
#include <QtCore/QVariant>
#include <QtWidgets/QAction>
#include <QtWidgets/QApplication>
#include <QtWidgets/QButtonGroup>
#include <QtWidgets/QCheckBox>
#include <QtWidgets/QDoubleSpinBox>
#include <QtWidgets/QGroupBox>
#include <QtWidgets/QHeaderView>
#include <QtWidgets/QLabel>
#include <QtWidgets/QLineEdit>
#include <QtWidgets/QListWidget>
#include <QtWidgets/QMainWindow>
#include <QtWidgets/QMenu>
#include <QtWidgets/QMenuBar>
#include <QtWidgets/QProgressBar>
#include <QtWidgets/QPushButton>
#include <QtWidgets/QRadioButton>
#include <QtWidgets/QSpinBox>
#include <QtWidgets/QStatusBar>
#include <QtWidgets/QToolBar>
#include <QtWidgets/QWidget>
```

Include dependency graph for ui_RigidTrack.h:

This graph shows which files directly or indirectly include this file:



Classes

- class **Ui_RigidTrackClass**
- class **Ui::RigidTrackClass**

Namespaces

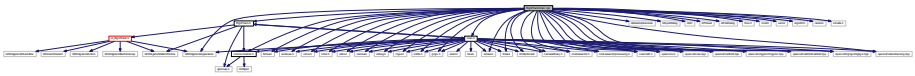
- **Ui**

1.10 RigidTrack/main.cpp File Reference

```
#include "RigidTrack.h"
#include <QtWidgets/QApplication>
#include <QDesktopServices>
#include <QInputDialog>
#include <QUrl>
#include <QThread>
#include <QUdpSocket>
#include <QFileDialog>
#include "cameralibrary.h"
#include "modulevector.h"
#include "modulevectorprocessing.h"
#include "coremath.h"
#include <opencv\cv.h>
#include "opencv2\core.hpp"
#include "opencv2\calib3d.hpp"
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/calib3d/calib3d.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2\video\tracking.hpp>
#include <fstream>
#include <windows.h>
#include <conio.h>
#include <tchar.h>
```

```
#include <stdio.h>
#include <iostream>
#include <stdarg.h>
#include <ctype.h>
#include <stdlib.h>
#include <gl/glu.h>
#include <sstream>
#include <time.h>
#include <cmath>
#include <vector>
#include <algorithm>
#include <random>
#include <thread>
#include <strsafe.h>
#include "main.h"
#include "communication.h"
```

Include dependency graph for main.cpp:



Functions

- int **main** (int argc, char ∗argv[ ])

  file handler for writing the log file
- QPixmap **Mat2QPixmap** (cv::Mat src)

  convert a opencv matrix that represents a picture to a Qt Pixmap object
- void **calcBoardCornerPositions** (Size boardSize, float squareSize, std::vector< Point3f > &corners)

  calculate the chess board corner positions, used for the camera calibration
- void **getEulerAngles** (Mat &rotCamerMatrix, Vec3d & **eulerAngles**)

  get the euler angles from a rotation matrix
- int **start_camera** ()

  start the loop that fetches frames, computes the position etc and sends it to other computers
- void **start_stopCamera** ()

  Start or stop the camera depending on if the camera is currently running or not.
- int **setZero** ()

  determine the initial position of the object that serves as reference point or as ground frame origin
- int **calibrate_camera** ()

  start the camera calibration routine that computes the camera matrix and distortion coefficients
- void **load_calibration** (int method)

  Load a previously saved camera calibration from a file.
- void **test_Algorithm** ()

  project some points from 3D to 2D and then check the accuracy of the algorithms
- void **projectCoordinateFrame** (Mat pictureFrame)

  project a coordinate CoSy with the rotation and translation of the object for visualization
- void **setUpUDP** ()

  open the UDP ports for communication
- void **setHeadingOffset** (double d)

  Add a heading offset to the attitude for the case it is necessary.
- void **sendDataUDP** (cv::Vec3d &Position, cv::Vec3d &Euler)

send the position and attitude over UDP to every receiver, the safety receiver is handled on its own in the start_camera function

- void **closeUDP** ()

  close the UDP ports again to release network interfaces etc.
- void **loadMarkerConfig** (int method)

  load a marker configuration from file. This file has to be created by hand, use the standard marker configuration file as template
- void **drawPositionText** (cv::Mat &Picture, cv::Vec3d &Position, cv::Vec3d &Euler, double **error**)

  draw the position, attitude and reprojection error in the picture
- void **loadCameraPosition** ()
- int **determineExposure** ()

  get the optimal exposure for the camera. For that find the minimum and maximum exposure were the right number of markers are detected
- void **determineOrder** ()
- int **calibrateGround** ()

Variables

- **commObject commObj**
- bool **safetyEnable** = false
- bool **safety2Enable** = false

  is the safety feature enabled
- double **safetyBoxLength** = 1.5

  is the second receiver enabled
- int **safetyAngle** = 30

  length of the safety area cube in meters
- bool **exitRequested** = true

  bank and pitch angle protection in degrees
- int **invertZ** = 1

  variable if tracking loop should be exited
- double **frameTime** = 0.01

  dummy variable to invert Z direction on request
- double **timeOld** = 0.0

  100 Hz CoSy rate, is later on replaced with the hardware timestamp delivered by the camera
- double **timeFirstFrame** = 0

  old time for finite differences velocity calculation. Is later on replaced with the hardware timestamp delivered by the camera
- Vec3d **position** = Vec3d()

  Time stamp of the first frame. This value is then subtracted for every other frame so the time in the log start at zero.
- Vec3d **eulerAngles** = Vec3d()

  position vector x,y,z for object position in O-CoSy, unit is meter
- Vec3d **positionOld** = Vec3d()

  Roll Pitch Heading in this order, units in degrees.
- Vec3d **velocity** = Vec3d()

  old position in O-CoSy for finite differences velocity calculation
- Vec3d **posRef** = Vec3d()

  velocity vector of object in o-CoSy in respect to o-CoSy
- Vec3d **eulerRef** = Vec3d()

  initial position of object in camera CoSy
- double **headingOffset** = 0

  initial euler angle of object respectivley to camera CoSy

- int **intIntensity** = 15

  heading offset variable for aligning INS heading with tracking heading
- int **intExposure** = 1

  max infrared spot light intensity is 15 1-6 is strobe 7-15 is continuous 13 and 14 are meaningless
- int **intFrameRate** = 100

  *max is 480 increase if markers are badly visible but should be determined automatically during **setZero()** (p. 28)*
- int **intThreshold** = 200

  CoSy rate of camera, maximum is 100 fps.
- Mat **Rmat** = (cv::Mat_<double>(3, 1) << 0.0, 0.0, 0.0)

  threshold value for marker detection. If markers are badly visible lower this value but should not be necessary
- Mat **RmatRef** = (cv::Mat_<double>(3, 3) << 1., 0., 0., 0., 1., 0., 0., 0., 1.)

  rotation matrix from camera CoSy to marker CoSy
- Mat **M_CN** = cv::Mat_<double>(3, 3)

  reference rotation matrix from camera CoSy to marker CoSy
- Mat **M_HeadingOffset** = cv::Mat_<double>(3, 3)

  rotation matrix from camera to ground, fixed for given camera position
- Mat **Rvec** = (cv::Mat_<double>(3, 1) << 0.0, 0.0, 0.0)

  rotation matrix that turns the ground system to the INS magnetic heading for alignment
- Mat **Tvec** = (cv::Mat_<double>(3, 1) << 0.0, 0.0, 0.0)

  rotation vector (axis-angle notation) from camera CoSy to marker CoSy
- Mat **RvecOriginal**

  translation vector from camera CoSy to marker CoSy in camera CoSy
- Mat **TvecOriginal**

  initial values as start values for algorithms and algorithm tests
- bool **useGuess** = true

  initial values as start values for algorithms and algorithm tests
- int **methodPNP** = 0

  set to true and the algorithm uses the last result as starting value
- int **numberMarkers** = 4

  solvePNP algorithm 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP //! 4 and 1 are the same and not implemented correctly by OpenCV
- std::vector< Point3d > **list_points3d**

  number of markers. Is loaded during start up from the marker configuration file
- std::vector< Point2d > **list_points2d**

  marker positions in marker CoSy
- std::vector< Point2d > **list_points2dOld**

  marker positions projected in 2D in camera image CoSy
- std::vector< double > **list_points2dDifference**

  marker positions in previous picture in 2D in camera image CoSy
- std::vector< Point2d > **list_points2dProjected**

  difference of the old and new 2D marker position to determine the order of the points
- std::vector< Point2d > **list_points2dUnsorted**

  3D marker points projected to 2D in camera image CoSy with the algorithm projectPoints
- std::vector< Point3d > **coordinateFrame**

  marker points in 2D camera image CoSy, sorted with increasing x (camera image CoSy) but not sorted to correspond with list_points3d
- std::vector< Point2d > **coordinateFrameProjected**

  coordinate visualisazion of marker CoSy
- int **pointOrderIndices** [ ] = { 0, 1, 2, 3 }

  marker CoSy projected from 3D to 2D camera image CoSy
- int **pointOrderIndicesNew** [ ] = { 0, 1, 2, 3 }

old correspondence from list_points3d and list_points_2d

- double **currentPointDistance** = 5000

  new correspondence from list_points3d and list_points_2d
- double **minPointDistance** = 5000

  distance from the projected 3D points (hence in 2d) to the real 2d marker positions in camera image CoSy
- int **currentMinIndex** = 0

  minimum distance from the projected 3D points (hence in 2d) to the real 2d marker positions in camera image CoSy
- bool **gotOrder** = false

  helper variable set to the point order that holds the current minimum point distance
- bool **camera_started** = false

  order of the list_points3d and list_points3d already tetermined or not, has to be done once
- Mat **cameraMatrix**

  variable thats needed to exit the main while loop
- Mat **distCoeffs**

  camera matrix of the camera
- Core::DistortionModel **distModel**

  distortion coefficients of the camera
- QUdpSocket * **udpSocketObject**

  distortion model of the camera
- QUdpSocket * **udpSocketSafety**

  socket for the communication with the object
- QUdpSocket * **udpSocketSafety2**

  socket for the communication with the circuit breaker
- QHostAddress **IPAdressObject** = QHostAddress("127.0.0.1")

  socket for the communication with the rope winch
- QHostAddress **IPAdressSafety** = QHostAddress("192.168.4.1")

  IPv4 adress of the object wifi telemetry chip, can change to 192.168.4.x. This is where the position etc is sent to.
- QHostAddress **IPAdressSafety2** = QHostAddress("192.168.4.4")

  IPv4 adress of the circuit breaker, stays the same.
- int **portObject** = 9155

  IPv4 adress of the rope winch,.
- int **portSafety** = 9155

  Port of the object.
- int **portSafety2** = 9155

  Port of the safety switch.
- QByteArray **datagram**

  Port of the second receiver.
- QByteArray **data**

  data package that is sent to the object
- const int **BACKBUFFER_BITSPERPIXEL** = 8

  data package that's sent to the circuit breaker
- std::string **strBuf**

  8 bit per pixel and greyscale image from camera
- std::stringstream **ss**

  buffer that holds the strings that are sent to the Qt GUI
- QString **logFileName**

  stream that sends the strBuf buffer to the Qt GUI
- std::string **logName**

  Filename for the logfiles.
- SYSTEMTIME **logDate**

  Filename for the logfiles as standard string.
- std::ofstream **logfile**

  Systemtime struct that saves the current date and time thats needed for the log file name creation.

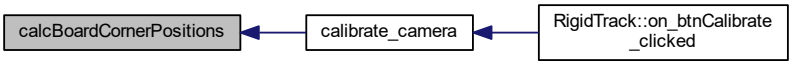
1.10.1 Function Documentation

1.10.1.1 calcBoardCornerPositions()

```
void calcBoardCornerPositions (
            Size boardSize,
            float squareSize,
            std::vector< Point3f > & corners )
```

calculate the chess board corner positions, used for the camera calibration

Here is the caller graph for this function:



1.10.1.2 calibrate_camera()

```
int calibrate_camera ( )
```

start the camera calibration routine that computes the camera matrix and distortion coefficients

== Initialize Camera SDK ==–

== At this point the Camera SDK is actively looking for all connected cameras and will initialize == them on it's own.

== Get a connected camera ==========-—

== Determine camera resolution

== Set Video Mode ==–

== We set the camera to Segment Mode here. This mode is support by all of our products. == Depending on what device you have connected you might want to consider a different == video mode to achieve the best possible tracking quality. All devices that support a == mode that will achieve a better quality output with a mode other than Segment Mode are == listed here along with what mode you should use if you're looking for the best head

== tracking:

== V100:R1/R2 Precision Mode == TrackIR 5 Bit-Packed Precision Mode == V120 Precision Mode == TBar Precision Mode

== S250e Precision Mode

== If you have questions about a new device that might be conspicuously missing here or == have any questions about head tracking, email support or participate in our forums.

== Start camera output ==–

== Camera Matrix creation ==–

== Ok, start main loop. This loop fetches and displays ===— == camera frames. ===— But first set some camera parameters

the user has to provide the size of one square in mm

== Fetch a new frame from the camera ===—

which is why we also set this constant to 8

later on, when we get the frame as usual:

== Lets have the Camera Library raster the camera's == image into our texture.

imwrite("test.jpg" + , matFrame);

If done with success,

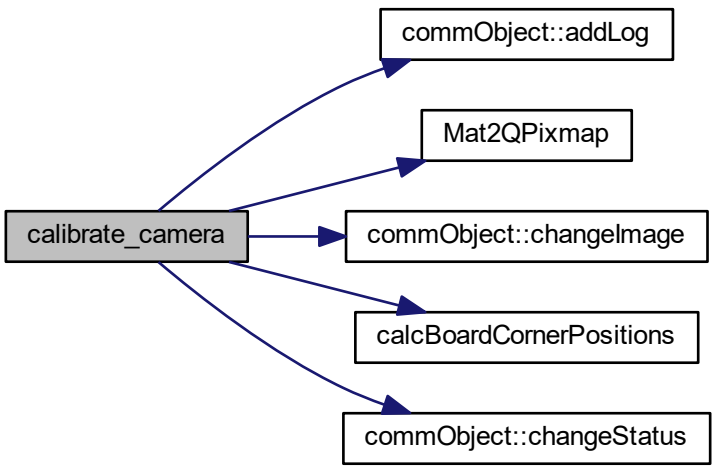
improve the found corners' coordinate accuracy for chessboard

== Release camera ==–

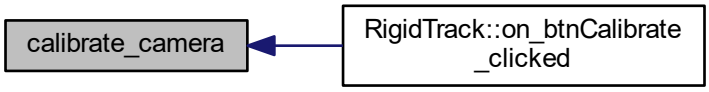
Save the obtained calibration coefficients in a file for later use

- FileStorage::MEMORY);

Here is the call graph for this function:



Here is the caller graph for this function:



1.10.1.3 calibrateGround()

```
int calibrateGround ( )
```

Get the pose of the camera w.r.t the ground calibration frame. This frame sets the navigation frame for later results. The pose is averaged over 200 samples and then saved in the file referenceData.xml. This routine is basically the same as setZero. initialize the variables with starting values

== Initialize Camera SDK ==–

== At this point the Camera SDK is actively looking for all connected cameras and will initialize == them on it's own.

== Get a connected camera ==========-—

== If no device connected, pop a message box and exit ==–

== Determine camera resolution to size application window ==-—

Set camera mode to precision mode, it directly provides marker coordinates

== Start camera output ==–

== Turn on some overlay text so it's clear things are ===— == working even if there is nothing in the camera's view. ===— Set some other parameters as well of the camera

sample some frames and calculate the position and attitude. then average those values and use that as zero position

== Fetch a new frame from the camera ===—

== Ok, we've received a new frame, lets do something == with it.

==for(int i=0; i<frame->ObjectCount(); i++)

sort the 2d points with the correct indices as found in the preceeding order determination algorithm

Compute the pose from the 3D-2D corresponses

project the marker 3d points with the solution into the camera image CoSy and calculate difference to true camera image

Iterative Method needs time to converge to solution

That are not the values of yaw, roll and pitch yet! Rodriguez has to be called first.

==– one sample more :D

== Release camera ==–

Divide by the number of samples to get the mean of the reference position

eulerRef is here in Axis Angle notation

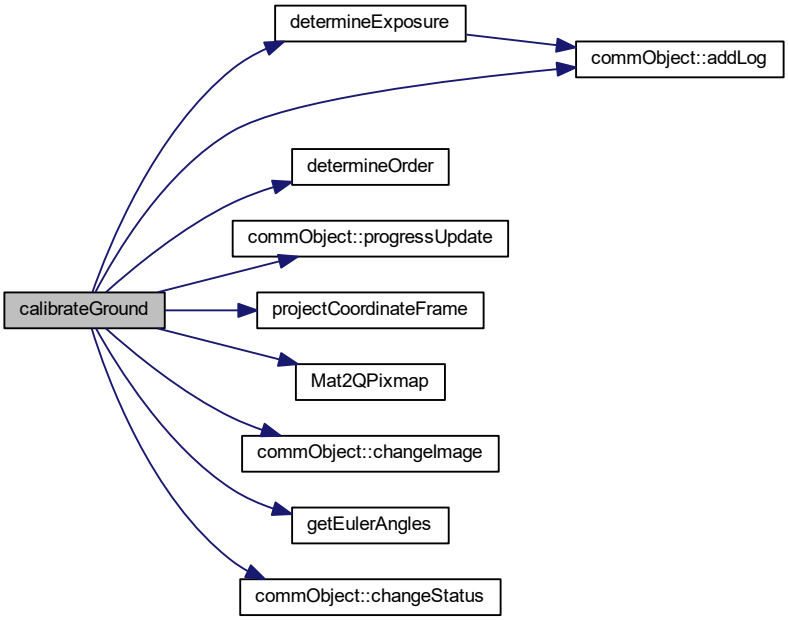
axis angle to rotation matrix ==– Euler Angles, finally

rotation matrix to euler

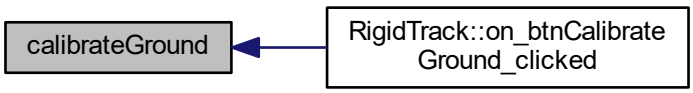
Save the obtained calibration coefficients in a file for later use

- FileStorage::MEMORY);

Here is the call graph for this function:



Here is the caller graph for this function:

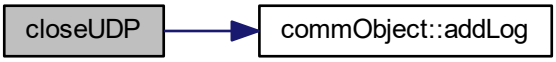


1.10.1.4 closeUDP()

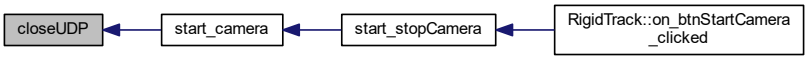
```
void closeUDP ( )
```

close the UDP ports again to release network interfaces etc.

check if the socket is open and if yes close it Here is the call graph for this function:



Here is the caller graph for this function:



1.10.1.5 determineExposure()

```
int determineExposure ( )
```

get the optimal exposure for the camera. For that find the minimum and maximum exposure were the right number of markers are detected

== For OptiTrack Ethernet cameras, it's important to enable development mode if you == want to stop execution for an extended time while debugging without disconnecting == the Ethernet devices. Lets do that now:

== Initialize Camera SDK ==–

== At this point the Camera SDK is actively looking for all connected cameras and will initialize == them on it's own.

== Get a connected camera ==========-—

== If no device connected, pop a message box and exit ==–

== Determine camera resolution to size application window ==-—

set the camera mode to precision mode, it used greyscale imformation for marker property calculations

== Start camera output ==–

== Turn on some overlay text so it's clear things are ===— == working even if there is nothing in the camera's view. ===—

set the camera exposure

set the camera infrared LED intensity

set the camera framerate to 100 Hz

enable the filter that blocks visible light and only passes infrared light

enable high power mode of the leds

enable continuous LED light

set threshold for marker detection

set exposure such that num markers are visible

Number of objects (markers) found in the current picture with the given exposure

exposure when objects detected the first time is numberMarkers

exposure when objects detected is first time numberMarkers+1

set the exposure to the smallest value possible

if the markers arent found after numberTries then there might be no markers at all in the real world

Determine minimum exposure, hence when are numberMarkers objects detected

get a new camera frame

frame received

how many objects are detected in the image

if the right amount if markers is found, exit while loop

not the right amount of markers was found so increase the exposure and try again

Now determine maximum exposure, hence when are numberMarkers+1 objects detected

if the markers arent found after numberTries then there might be no markers at all in the real world

how many objects are detected in the image

if the right amount if markers is found, exit while loop

not the right amount of markers was found so decrease the exposure and try again

set the exposure to the mean of min and max exposure determined

and now check if the correct amount of markers is detected with that new value

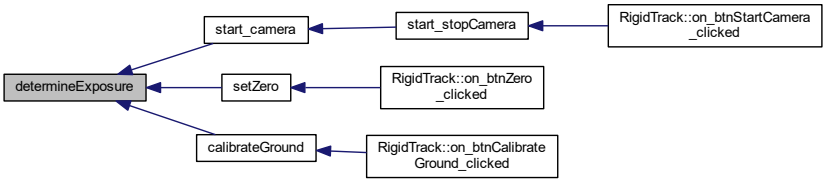
how many objects are detected in the image

are all markers and not more or less detected in the image

== Release camera ==–

all markers and not more or less are found Here is the call graph for this function:

Here is the caller graph for this function:



1.10.1.6 determineOrder()

```
void determineOrder ( )
```

compute the order of the marker points in 2D so they are the same as in the 3D array. Hence marker 1 must be in first place for both, list_points2d and list_points3d determine the 3D-2D correspondences that are crucial for the PnP algorithm Try every possible correspondence and solve PnP Then project the 3D marker points into the 2D camera image and check the difference between projected points and points as seen by the camera the corresponce with the smallest difference is probably the correct one

the difference between true 2D points and projected points is super big

now try every possible permutation of correspondence

reset the starting values for solvePnP

sort the 2d points with the current permutation

Call solve PNP with P3P since its more robust and sufficient for start value determination

set the current difference of all point correspondences to zero

project the 3D points with the solvePnP solution onto 2D

now compute the absolute difference (error)

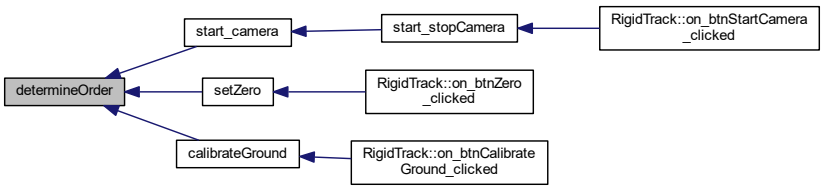
if the difference with the current permutation is smaller than the smallest value till now it is probably the more correct permutation

set the smallest value of difference to the current one

now safe the better permutation

try every permutation

now that the correct order is found assign it to the indices array Here is the caller graph for this function:

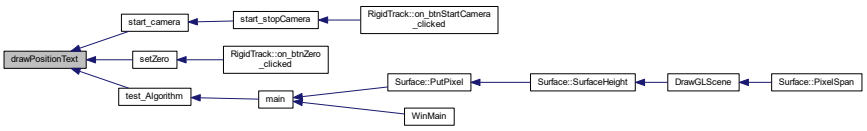


1.10.1.7 drawPositionText()

```
void drawPositionText (
            cv::Mat & Picture,
            cv::Vec3d & Position,
            cv::Vec3d & Euler,
            double error )
```

draw the position, attitude and reprojection error in the picture

Here is the caller graph for this function:

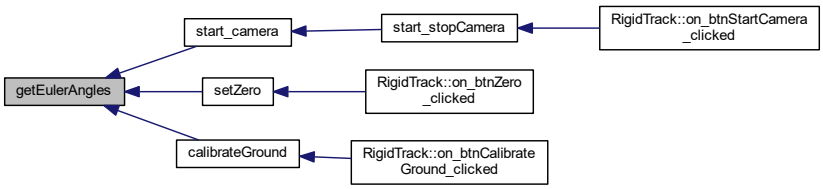


1.10.1.8 getEulerAngles()

```
void getEulerAngles (
            Mat & rotCamerMatrix,
            Vec3d & eulerAngles )
```

get the euler angles from a rotation matrix

Here is the caller graph for this function:

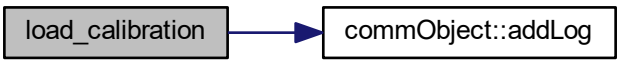


1.10.1.9 load_calibration()

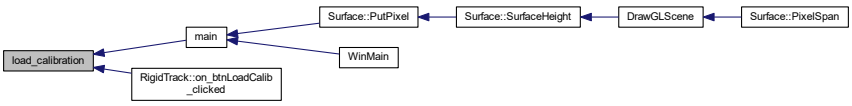
```
void load_calibration (
            int method )
```

Load a previously saved camera calibration from a file.

Here is the call graph for this function:



Here is the caller graph for this function:

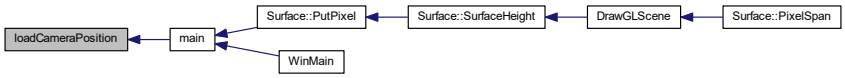


1.10.1.10 loadCameraPosition()

```
void loadCameraPosition ( )
```

load the rotation matrix from camera CoSy to ground CoSy It is determined during ground calibration and once the camera is mounted and fixed stays the same Open the referenceData.xml that contains the rotation from camera CoSy to ground CoSy Here is the call graph for this function:

Here is the caller graph for this function:



1.10.1.11 loadMarkerConfig()

```
void loadMarkerConfig (
            int method )
```

load a marker configuration from file. This file has to be created by hand, use the standard marker configuration file as template

during start up of the programm load the standard marker configuration

open the standard marker configuration file

copy the values to the respective variables

inizialise vectors with correct length depending on the number of markers

save the marker locations in the points3d vector

if the load marker configuration button was clicked show a open file dialog

was cancel or abort clicked

if yes load the standard marker configuration

open the selected marker configuration file

copy the values to the respective variables

inizialise vectors with correct length depending on the number of markers

save the marker locations in the points3d vector

Print out the number of markers and their position to the GUI

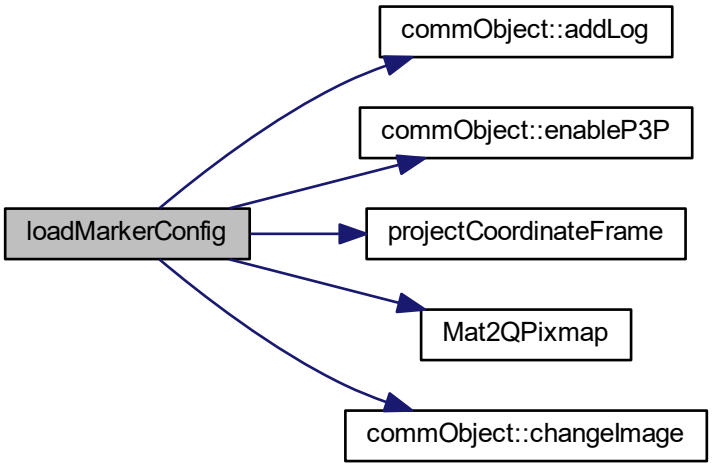
check if P3P algorithm can be enabled, it needs exactly 4 marker points to work

if P3P is possible, let the user choose which algorithm he wants but keep iterative active

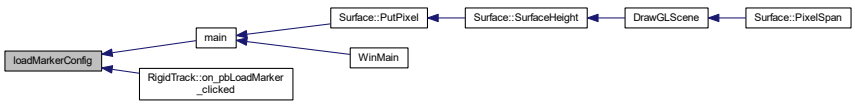
More (or less) marker than 4 loaded, P3P is not possible, hence user cant select P3P in GUI

now display the marker configuration in the camera view

Set the camera pose parallel to the marker coordinate system Here is the call graph for this function:



Here is the caller graph for this function:



1.10.1.12 main()

```
int main (
             int argc,
             char * argv[] )
```

file handler for writing the log file

/<======= Entry Point ======–

main inizialised the GUI and values for the marker position etc show the GUI

connect the Qt slots and signals for event handling

initial guesses for position and rotation, important for Iterative Method!

Points that make up the marker CoSy axis system, hence one line in each axis direction

set position initial values

set position initial values

set position initial values

set velocity initial values

set velocity initial values

set velocity initial values

set initial euler angles to arbitrary values for testing

set initial euler angles to arbitrary values for testing

set initial euler angles to arbitrary values for testing

set the heading offset to 0

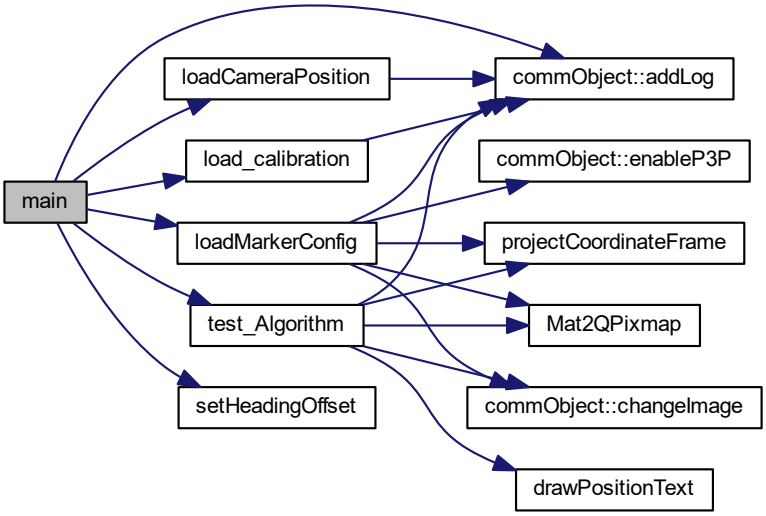
outputs in the log etc are limited to 3 decimal values

load the rotation matrix from camera CoSy to ground CoSy

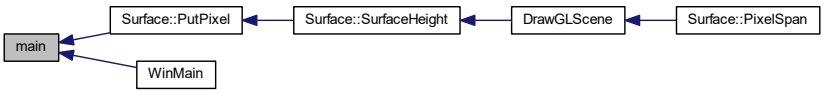
load the calibration file with the camera intrinsics

load the standard marker configuration

test the algorithms and their accuracy Here is the call graph for this function:



Here is the caller graph for this function:

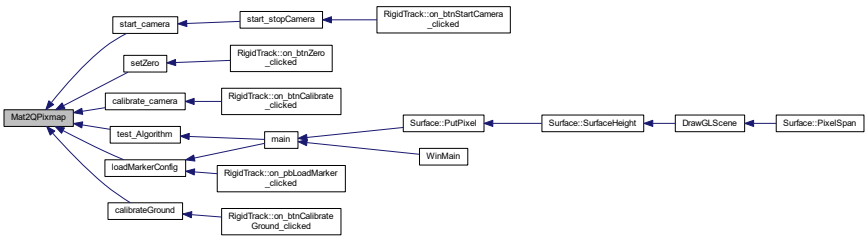


1.10.1.13 Mat2QPixmap()

```
QPixmap Mat2QPixmap (
            cv::Mat src )
```

convert a opencv matrix that represents a picture to a Qt Pixmap object

enforce deep copy, see documentation of QImage::QImage (const uchar ∗ data, int width, int height, Format format) Here is the caller graph for this function:



1.10.1.14 projectCoordinateFrame()

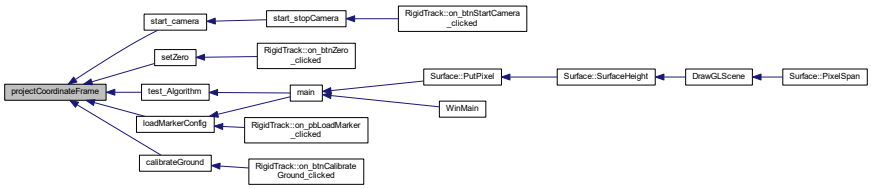
```
void projectCoordinateFrame (
            Mat pictureFrame )
```

project a coordinate CoSy with the rotation and translation of the object for visualization

z-axis

x-axis

y-axis Here is the caller graph for this function:



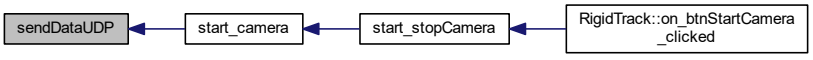
1.10.1.15 sendDataUDP()

```
void sendDataUDP (
            cv::Vec3d & Position,
            cv::Vec3d & Euler )
```

send the position and attitude over UDP to every receiver, the safety receiver is handled on its own in the start_↩ camera function

Roll Pitch Heading

if second receiver is activated send it also the tracking data Here is the caller graph for this function:



1.10.1.16 setHeadingOffset()

```
void setHeadingOffset (
            double d )
```

Add a heading offset to the attitude for the case it is necessary.

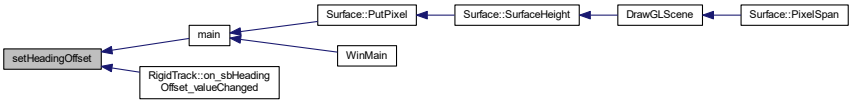
Convert heading offset from degrees to rad

Calculate rotation about x axis

Calculate rotation about y axis

Calculate rotation about z axis

Combined rotation matrix Here is the caller graph for this function:



1.10.1.17 setUpUDP()

```
void setUpUDP ( )
```

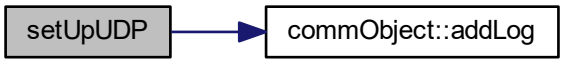
open the UDP ports for communication

Initialise the QDataStream that stores the data to be send

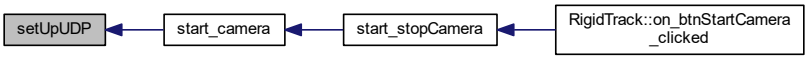
Create UDP slots

if the safety feature is activated open the udp port

if the second receiver feature is activated open the udp port Here is the call graph for this function:



Here is the caller graph for this function:



1.10.1.18 setZero()

```
int setZero ( )
```

determine the initial position of the object that serves as reference point or as ground frame origin

initialize the variables with starting values

== Initialize Camera SDK ==–

== At this point the Camera SDK is actively looking for all connected cameras and will initialize == them on it's own.

== Get a connected camera ==========-—

== If no device connected, pop a message box and exit ==–

== Determine camera resolution to size application window ==-—

Set camera mode to precision mode, it directly provides marker coordinates

== Start camera output ==–

== Turn on some overlay text so it's clear things are ===— == working even if there is nothing in the camera's view. ===— Set some other parameters as well of the camera

sample some frames and calculate the position and attitude. then average those values and use that as zero position

difference between the marker points as seen by the camera and the projected marker points with Rvec and Tvec

== Fetch a new frame from the camera ===—

== Ok, we've received a new frame, lets do something == with it.

==for(int i=0; i<frame->ObjectCount(); i++)

sort the 2d points with the correct indices as found in the preceeding order determination algorithm

Compute the pose from the 3D-2D corresponses

project the marker 3d points with the solution into the camera image CoSy and calculate difference to true camera image

Iterative Method needs time to converge to solution

That are not the values of yaw, roll and pitch yet! Rodriguez has to be called first.

==– one sample more :D

== Release camera ==–

Divide by the number of samples to get the mean of the reference position

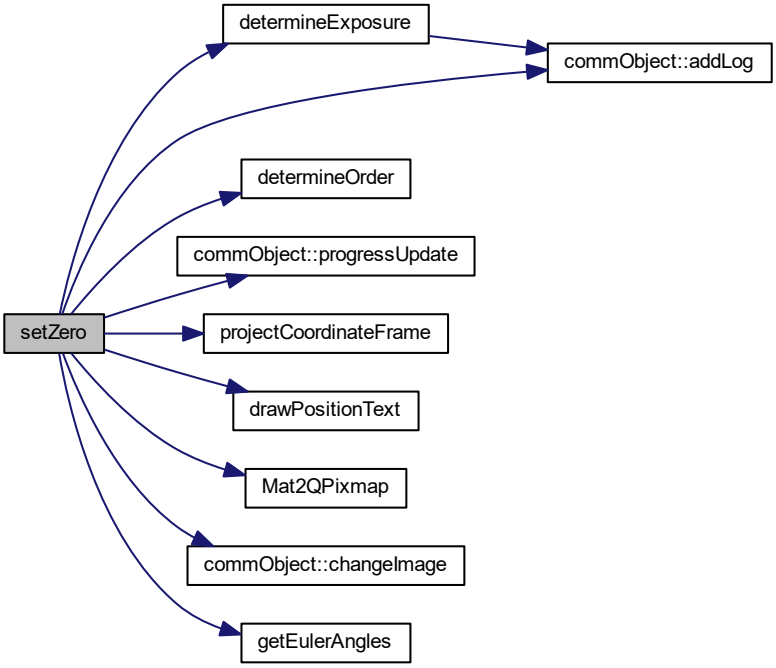
eulerRef is here in Axis Angle notation

axis angle to rotation matrix

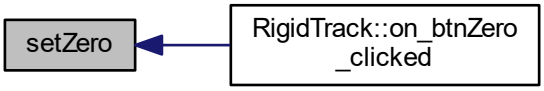
==– Euler Angles, finally

rotation matrix to euler

compute the difference between last obtained TVec and the average Value When it is large the iterative method has not converged properly so it is advised to start the **setZero()** (p. 28) function once again Here is the call graph for this function:



Here is the caller graph for this function:



1.10.1.19 start_camera()

```
int start_camera ( )
```

start the loop that fetches frames, computes the position etc and sends it to other computers

The order of points, hence which entry in list_points3d corresponds to which in list_points2d is not calculated yet

Use the value of Rvec that was set in **main()** (p. 24) as starting value for the solvePnP algorithm

Use the value of Tvec that was set in **main()** (p. 24) as starting value for the solvePnP algorithm

Get the current date and time to name the log file

Concat the log file name as followed. The file is saved in the folder /logs in the Rigid Track installation folder

Convert the QString to a standard string

Get the exposure where the right amount of markers is detected

For OptiTrack Ethernet cameras, it's important to enable development mode if you want to stop execution for an extended time while debugging without disconnecting the Ethernet devices. Lets do that now:

Initialize Camera SDK

At this point the Camera SDK is actively looking for all connected cameras and will initialize them on it's own

Get a connected camera

If no camera can be found, inform user in message log and exit function

Determine camera resolution to size application window

Set the camera mode to precision mode, it used greyscale imformation for marker property calculations

Start camera output

Turn on some overlay text so it's clear things are working even if there is nothing in the camera's view

Set the camera exposure

Set the camera infrared LED intensity

Set the camera framerate to 100 Hz

Enable the filter that blocks visible light and only passes infrared light

Enable high power mode of the LEDs

Disable continuous LED light

Set threshold for marker detection

Create a new matrix that stores the grayscale picture from the camera

QPixmap is the corresponding Qt class that saves images

Matrix that stores the colored picture, hence marker points, coordinate frame and reprojected points

Helper variable used to kick safety switch

Variables for the min and max values that are needed for sanity checks

Ff a marker is not visible or accuracy is bad increase this counter

Equals the quality of the tracking

Open sockets and ports for UDP communication

If the safety feature is enabled send the starting message

Send enable message, hence send a 9 and then a 1

Fetch a new frame from the camera

Get the timestamp of the first frame. This time is subtracted from every subseeding frame so the time starts at 0 in the logs

While no new frame is received loop

Get a new camera frame

There is actually a new frame

Get the time stamp for the first frame. It is subtracted for the following frames

Release the frame so the camera can continue

Exit the while loop

Now enter the main loop that processes each frame and computes the pose, sends it and logs stuff

Check if the user has not pressed "Stop Tracking" yet

Fetch a new frame from the camera

Did we got a new frame or does the camera still need more time

Increase by one, if everything is okay it is decreased at the end of the loop again

Only use this frame it the right number of markers is found in the picture

Get the marker points in 2D in the camera image frame and store them in the list_points2dUnsorted vector The order of points that come from the camera corresponds to the Y coordinate

Was the order already determined? This is false for the first frame and from then on true

Now compute the order

Sort the 2d points with the correct indices as found in the preceeding order determination algorithm

pointOrderIndices was calculated in **determineOrder()** (p. 20)

The first time the 2D-3D corresspondence was determined with gotOrder was okay. But this order can change as the object moves and the marker objects appear in a different order in the frame->Object() array. The solution is that: When a marker point (in the camera image, hence in 2D) was at a position then it wont move that much from one frame to the other. So for the new frame we take a marker object and check which marker was closest this point in the old image frame? This is probably the same (true) marker. And we do that for every other marker as well. When tracking is good and no frames are dropped because of missing markers this should work every frame.

The sum of point distances is set to something unrealistic large

Calculate N_2 norm of unsorted points minus old points

If the norm is smaller than minPointDistance the correspondence is more likely to be correct

Update the array that saves the new point order

Now the new order is found, set the point order to the new value

Save the unsorted position of the marker points for the next loop

Compute the object pose from the 3D-2D corresponses

Project the marker 3d points with the solution into the camera image CoSy and calculate difference to true camera image

Difference of true pose and found pose

Increase the framesDropped variable if accuracy of tracking is too bad

Set number of subsequent frames dropped to zero because error is small enough and no marker was missing

Get the min and max values from TVec for sanity check

Sanity check of values. negative z means the marker CoSy is behind the camera, that's not possible.

Release the frame so the camera can move on

Release the camera

Close all UDP connections so the programm can be closed later on and no resources are locked

Exit the function

Next step is the transformation from camera CoSy to navigation CoSy Compute the relative object position from the reference position to the current one given in the camera CoSy: T_C^{NM} = Tvec - Tvec_{Ref}

Transform the position from the camera CoSy to the navigation CoSy with INS alligned heading and convert from [mm] to [m] T_N^{NM} = M_{NC} T_C^{NM}

Position is the result of the preceeding calculation

Invert Z if check box in GUI is activated, hence height above ground is considered

Realtive angle between reference orientation and current orientation

Convert axis angle respresentation to ordinary rotation matrix

The difference of the reference rotation and the current rotation R_{ NM } = M_{ NC } R_{ CM }

Euler Angles, finally

Get the euler angles from the rotation matrix

Add the heading offset to the heading angle

Compute the velocity with finite differences. Only use is the log file. It is done here because the more precise time stamp can be used

Time between the old frame and the current frame

Set the old frame time to the current one

Calculate the x velocity with finite differences

Calculate the y velocity with finite differences

Calculate the z velocity with finite differences

Set the old position to the current one for next frame velocity calcuation

Send position and Euler angles over WiFi with 100 Hz

Save the values in a log file, values are: Time sinc tracking started Position Euler Angles Velocity

Open the log file, the folder is RigidTrackInstallationFolder/logs

Close the file to save values

Check if the position and euler angles are below the allowed value, if yes send OKAY signal (1), if not send shutdown signal (0) Absolute x, y and z position in navigation CoSy must be smaller than the allowed distance

Absolute Euler angles must be smaller than allowed value. Heading is not considered

Send the OKAY signal to the desired computer every 5th time

Send the 1

reset the counter that is needed for decimation to every 5th time step

The euler angles of the object exceeded the allowed euler angles, send the shutdown signal (0)

Send the shutdown signal, a 0

Inform the user

The position of the object exceeded the allowed position, shut the object down

Send the shutdown signal, a 0

Inform the user

Inform the user if tracking system is disturbed (marker lost or so) or error was too big

Also send the shutdown signal

Send the shutdown signal, a 0

Inform the user

Rasterize the frame so it can be shown in the GUI

Convert the frame from greyscale as it comes from the camera to rgb color

Project (draw) the marker CoSy origin into 2D and save it in the cFrame image

Project the marker points from 3D to the camera image frame (2d) with the computed pose

Draw a circle around the projected points so the result can be better compared to the real marker position In the resulting picture those are the red dots

Write the current position, attitude and error values as text in the frame

Send the new camera picture to the GUI and call the GUI processing routine

Update the picture in the GUI

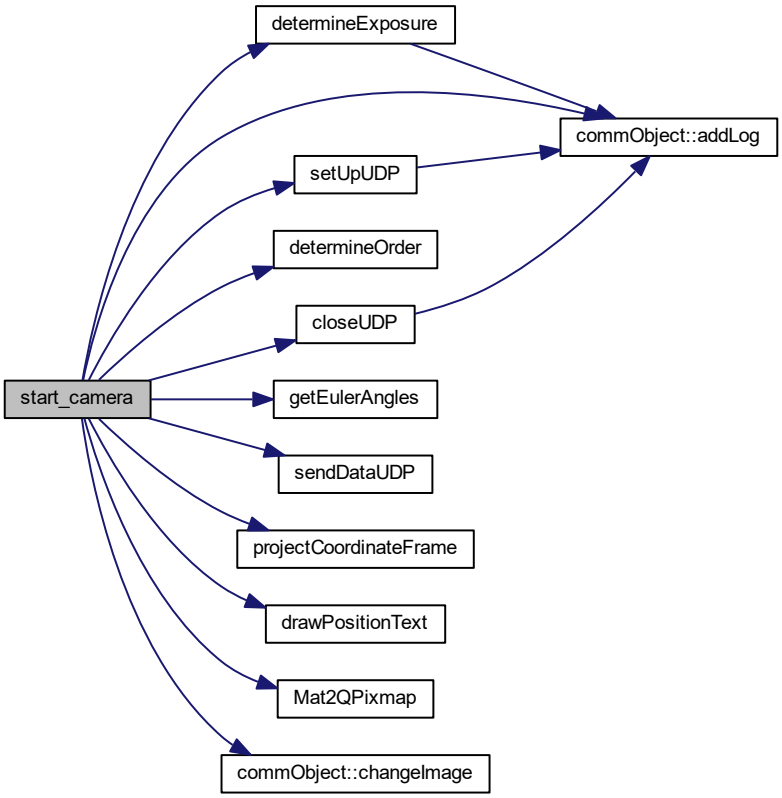
Give Qt time to handle everything

Release the camera frame to fetch the new one

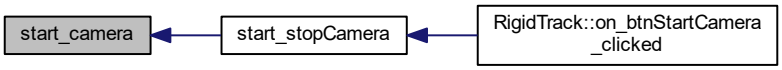
User choose to stop the tracking, clean things up

Close the UDP connections so resources are deallocated

Release camera Here is the call graph for this function:



Here is the caller graph for this function:



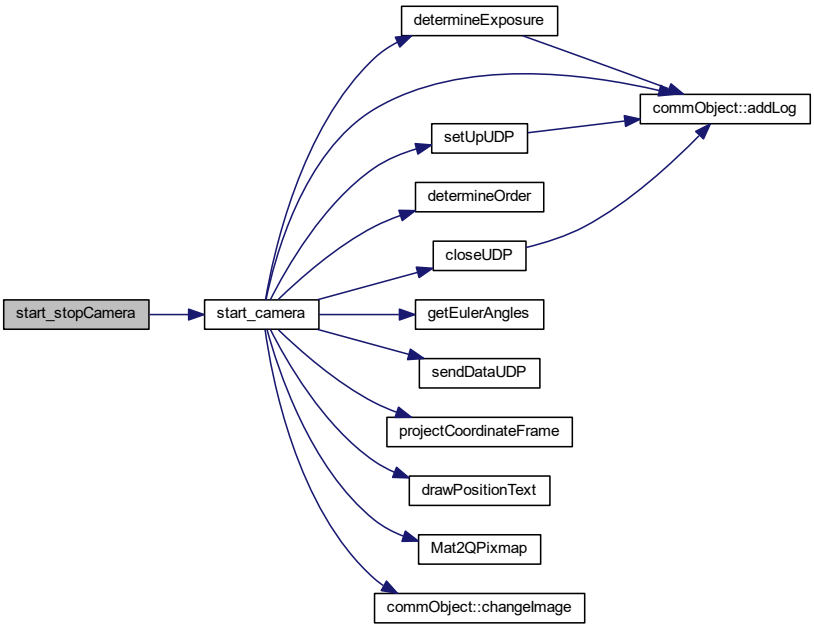
1.10.1.20 start_stopCamera()

```
void start_stopCamera ( )
```

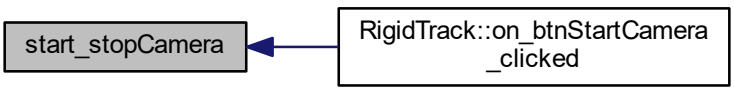
Start or stop the camera depending on if the camera is currently running or not.

tracking is not running so start it

tracking is currently running, set exitRequest to true so the while loop in **start_camera()** (p. 30) exits Here is the call graph for this function:



Here is the caller graph for this function:



1.10.1.21 test_Algorithm()

```
void test_Algorithm ( )
```

project some points from 3D to 2D and then check the accuracy of the algorithms

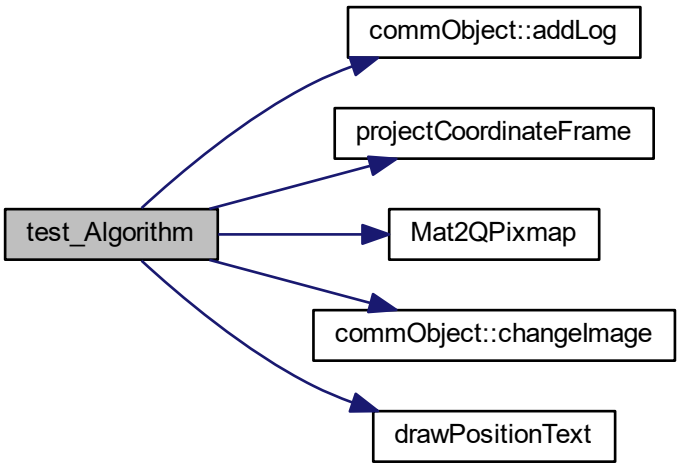
in mm

0 = iterative 1 = EPNP 2 = P3P 4 = UPNP //! not used

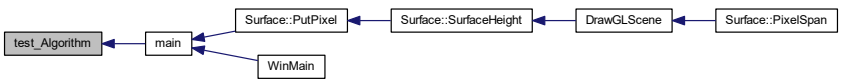
0 = iterative 1 = EPNP 2 = P3P 4 = UPNP //! not used

0 = iterative 1 = EPNP 2 = P3P 4 = UPNP //! not used

0 = iterative 1 = EPNP 2 = P3P 4 = UPNP //! not used Here is the call graph for this function:



Here is the caller graph for this function:



1.10.2 Variable Documentation

1.10.2.1 BACKBUFFER_BITSPERPIXEL

```
const int BACKBUFFER_BITSPERPIXEL = 8
```

data package that's sent to the circuit breaker

1.10.2.2 camera_started

```
bool camera_started = false
```

order of the list_points3d and list_points3d already tetermined or not, has to be done once

1.10.2.3 cameraMatrix

```
Mat cameraMatrix
```

variable thats needed to exit the main while loop

1.10.2.4 commObj

```
commObject commObj
```

1.10.2.5 coordinateFrame

```
std::vector<Point3d> coordinateFrame
```

marker points in 2D camera image CoSy, sorted with increasing x (camera image CoSy) but not sorted to correspond with list_points3d

1.10.2.6 coordinateFrameProjected

```
std::vector<Point2d> coordinateFrameProjected
```

coordinate visualisazion of marker CoSy

1.10.2.7 currentMinIndex

```
int currentMinIndex = 0
```

minimum distance from the projected 3D points (hence in 2d) to the real 2d marker positions in camera image CoSy

1.10.2.8 currentPointDistance

```
double currentPointDistance = 5000
```

new correspondence from list_points3d and list_points_2d

1.10.2.9 data

```
QByteArray data
```

data package that is sent to the object

1.10.2.10 datagram

```
QByteArray datagram
```

Port of the second receiver.

1.10.2.11 distCoeffs

```
Mat distCoeffs
```

camera matrix of the camera

1.10.2.12 distModel

```
Core::DistortionModel distModel
```

distortion coefficients of the camera

1.10.2.13 eulerAngles

```
Vec3d eulerAngles = Vec3d()
```

position vector x,y,z for object position in O-CoSy, unit is meter

1.10.2.14 eulerRef

```
Vec3d eulerRef = Vec3d()
```

initial position of object in camera CoSy

1.10.2.15 exitRequested

```
bool exitRequested = true
```

bank and pitch angle protection in degrees

1.10.2.16 frameTime

```
double frameTime = 0.01
```

dummy variable to invert Z direction on request

1.10.2.17 gotOrder

```
bool gotOrder = false
```

helper variable set to the point order that holds the current minimum point distance

1.10.2.18 headingOffset

```
double headingOffset = 0
```

initial euler angle of object respectivley to camera CoSy

1.10.2.19 intExposure

```
int intExposure = 1
```

max infrared spot light intensity is 15 1-6 is strobe 7-15 is continuous 13 and 14 are meaningless

1.10.2.20 intFrameRate

```
int intFrameRate = 100
```

max is 480 increase if markers are badly visible but should be determined automatically during **setZero()** (p. 28)

1.10.2.21 intIntensity

```
int intIntensity = 15
```

heading offset variable for aligning INS heading with tracking heading

1.10.2.22 intThreshold

```
int intThreshold = 200
```

CoSy rate of camera, maximum is 100 fps.

1.10.2.23 invertZ

```
int invertZ = 1
```

variable if tracking loop should be exited

1.10.2.24 IPAdressObject

```
QHostAddress IPAdressObject = QHostAddress("127.0.0.1")
```

socket for the communication with the rope winch

1.10.2.25 IPAdressSafety

```
QHostAddress IPAdressSafety = QHostAddress("192.168.4.1")
```

IPv4 adress of the object wifi telemetry chip, can change to 192.168.4.x. This is where the position etc is sent to.

1.10.2.26 IPAdressSafety2

```
QHostAddress IPAdressSafety2 = QHostAddress("192.168.4.4")
```

IPv4 adress of the circuit breaker, stays the same.

1.10.2.27 list_points2d

```
std::vector<Point2d> list_points2d
```

marker positions in marker CoSy

1.10.2.28 list_points2dDifference

```
std::vector<double> list_points2dDifference
```

marker positions in previous picture in 2D in camera image CoSy

1.10.2.29 list_points2dOld

```
std::vector<Point2d> list_points2dOld
```

marker positions projected in 2D in camera image CoSy

1.10.2.30 list_points2dProjected

```
std::vector<Point2d> list_points2dProjected
```

difference of the old and new 2D marker position to determine the order of the points

1.10.2.31 list_points2dUnsorted

```
std::vector<Point2d> list_points2dUnsorted
```

3D marker points projected to 2D in camera image CoSy with the algorithm projectPoints

1.10.2.32 list_points3d

```
std::vector<Point3d> list_points3d
```

number of markers. Is loaded during start up from the marker configuration file

1.10.2.33 logDate

```
SYSTEMTIME logDate
```

Filename for the logfiles as standard string.

1.10.2.34 logfile

```
std::ofstream logfile
```

Systemtime struct that saves the current date and time thats needed for the log file name creation.

1.10.2.35 logFileName

```
QString logFileName
```

stream that sends the strBuf buffer to the Qt GUI

1.10.2.36 logName

```
std::string logName
```

Filename for the logfiles.

1.10.2.37 M_CN

```
Mat M_CN = cv::Mat_<double>(3, 3)
```

reference rotation matrix from camera CoSy to marker CoSy

1.10.2.38 M_HeadingOffset

```
Mat M_HeadingOffset = cv::Mat_<double>(3, 3)
```

rotation matrix from camera to ground, fixed for given camera position

1.10.2.39 methodPNP

```
int methodPNP = 0
```

set to true and the algorithm uses the last result as starting value

1.10.2.40 minPointDistance

```
double minPointDistance = 5000
```

distance from the projected 3D points (hence in 2d) to the real 2d marker positions in camera image CoSy

1.10.2.41 numberMarkers

```
int numberMarkers = 4
```

solvePNP algorithm 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP //! 4 and 1 are the same and not implemented correctly by OpenCV

1.10.2.42 pointOrderIndices

```
int pointOrderIndices[] = { 0, 1, 2, 3 }
```

marker CoSy projected from 3D to 2D camera image CoSy

1.10.2.43 pointOrderIndicesNew

```
int pointOrderIndicesNew[] = { 0, 1, 2, 3 }
```

old correspondence from list_points3d and list_points_2d

1.10.2.44 portObject

```
int portObject = 9155
```

IPv4 adress of the rope winch,.

1.10.2.45 portSafety

```
int portSafety = 9155
```

Port of the object.

1.10.2.46 portSafety2

```
int portSafety2 = 9155
```

Port of the safety switch.

1.10.2.47 position

```
Vec3d position = Vec3d()
```

Time stamp of the first frame. This value is then subtracted for every other frame so the time in the log start at zero.

1.10.2.48 positionOld

```
Vec3d positionOld = Vec3d()
```

Roll Pitch Heading in this order, units in degrees.

1.10.2.49 posRef

```
Vec3d posRef = Vec3d()
```

velocity vector of object in o-CoSy in respect to o-CoSy

1.10.2.50 Rmat

```
Mat Rmat = (cv::Mat_<double>(3, 1) << 0.0, 0.0, 0.0)
```

threshold value for marker detection. If markers are badly visible lower this value but should not be necessary

== Rotation, translation etc. matrix for PnP results

1.10.2.51 RmatRef

```
Mat RmatRef = (cv::Mat_<double>(3, 3) << 1., 0., 0., 0., 1., 0., 0., 0., 1.)
```

rotation matrix from camera CoSy to marker CoSy

1.10.2.52 Rvec

```
Mat Rvec = (cv::Mat_<double>(3, 1) << 0.0, 0.0, 0.0)
```

rotation matrix that turns the ground system to the INS magnetic heading for alignment

1.10.2.53 RvecOriginal

```
Mat RvecOriginal
```

translation vector from camera CoSy to marker CoSy in camera CoSy

1.10.2.54 safety2Enable

```
bool safety2Enable = false
```

is the safety feature enabled

1.10.2.55 safetyAngle

```
int safetyAngle = 30
```

length of the safety area cube in meters

1.10.2.56 safetyBoxLength

```
double safetyBoxLength = 1.5
```

is the second receiver enabled

1.10.2.57 safetyEnable

```
bool safetyEnable = false
```

1.10.2.58 ss

```
std::stringstream ss
```

buffer that holds the strings that are sent to the Qt GUI

1.10.2.59 strBuf

```
std::string strBuf
```

8 bit per pixel and greyscale image from camera

1.10.2.60 timeFirstFrame

```
double timeFirstFrame = 0
```

old time for finite differences velocity calculation. Is later on replaced with the hardware timestamp delivered by the camera

1.10.2.61 timeOld

```
double timeOld = 0.0
```

100 Hz CoSy rate, is later on replaced with the hardware timestamp delivered by the camera

1.10.2.62 Tvec

```
Mat Tvec = (cv::Mat_<double>(3, 1) << 0.0, 0.0, 0.0)
```

rotation vector (axis-angle notation) from camera CoSy to marker CoSy

1.10.2.63 TvecOriginal

```
Mat TvecOriginal
```

initial values as start values for algorithms and algorithm tests

1.10.2.64 udpSocketObject

```
QUdpSocket* udpSocketObject
```

distortion model of the camera

IP adress of the circuit breaker that disables the object if a specified region is exited.

1.10.2.65 udpSocketSafety

```
QUdpSocket* udpSocketSafety
```

socket for the communication with the object

1.10.2.66 udpSocketSafety2

```
QUdpSocket* udpSocketSafety2
```

socket for the communication with the circuit breaker

1.10.2.67 useGuess

```
bool useGuess = true
```

initial values as start values for algorithms and algorithm tests

1.10.2.68 velocity

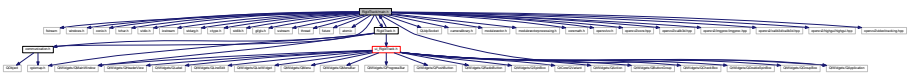
```
Vec3d velocity = Vec3d()
```

old position in O-CoSy for finite differences velocity calculation

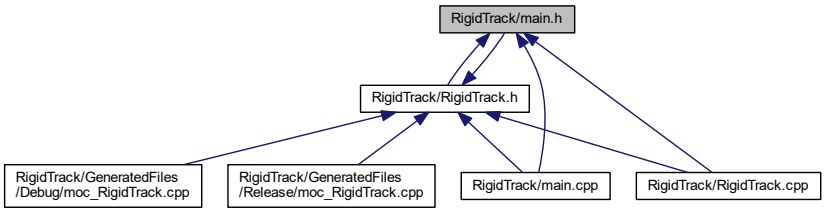
1.11 RigidTrack/main.h File Reference

```
#include <fstream>
#include <windows.h>
#include <conio.h>
#include <tchar.h>
#include <stdio.h>
#include <iostream>
#include <stdarg.h>
#include <ctype.h>
#include <stdlib.h>
#include <gl/glu.h>
#include <sstream>
#include <thread>
#include <future>
#include <atomic>
#include "communication.h"
#include "RigidTrack.h"
#include <QtWidgets/QApplication>
#include <QUdpSocket>
#include "cameralibrary.h"
#include "modulevector.h"
#include "modulevectorprocessing.h"
#include "coremath.h"
#include <opencv\cv.h>
#include "opencv2\core.hpp"
#include "opencv2\calib3d.hpp"
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/calib3d/calib3d.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2\video\tracking.hpp>
```

Include dependency graph for main.h:



This graph shows which files directly or indirectly include this file:



Functions

- int **start_camera** ()

  start the loop that fetches frames, computes the position etc and sends it to other computers
- void **start_stopCamera** ()

  Start or stop the camera depending on if the camera is currently running or not.
- int **setZero** ()

  determine the initial position of the object that serves as reference point or as ground frame origin
- int **calibrate_camera** ()

  start the camera calibration routine that computes the camera matrix and distortion coefficients
- void **load_calibration** (int method)

  Load a previously saved camera calibration from a file.
- void **test_Algorithm** ()

  project some points from 3D to 2D and then check the accuracy of the algorithms
- void **projectCoordinateFrame** (Mat pictureFrame)

  project a coordinate CoSy with the rotation and translation of the object for visualization
- void **setUpUDP** ()

  open the UDP ports for communication
- void **setHeadingOffset** (double d)

  Add a heading offset to the attitude for the case it is necessary.
- void **sendDataUDP** (cv::Vec3d &Position, cv::Vec3d &Euler)

  send the position and attitude over UDP to every receiver, the safety receiver is handled on its own in the start_camera function
- void **closeUDP** ()

  close the UDP ports again to release network interfaces etc.
- void **loadMarkerConfig** (int method)

  load a marker configuration from file. This file has to be created by hand, use the standard marker configuration file as template
- void **drawPositionText** (cv::Mat &Picture, cv::Vec3d &Position, cv::Vec3d &Euler, double **error**)

  draw the position, attitude and reprojection error in the picture
- void **loadCameraPosition** ()
- int **determineExposure** ()

  get the optimal exposure for the camera. For that find the minimum and maximum exposure were the right number of markers are detected
- void **determineOrder** ()
- int **calibrateGround** ()

Variables

- int **methodPNP**

  set to true and the algorithm uses the last result as starting value
- bool **safetyEnable**
- bool **safety2Enable**

  is the safety feature enabled
- double **safetyBoxLength**

  is the second receiver enabled
- int **safetyAngle**

  length of the safety area cube in meters
- QHostAddress **IPAdressObject**

  socket for the communication with the rope winch
- QHostAddress **IPAdressSafety**

  IPv4 adress of the object wifi telemetry chip, can change to 192.168.4.x. This is where the position etc is sent to.
- QHostAddress **IPAdressSafety2**

  IPv4 adress of the circuit breaker, stays the same.
- int **portObject**

  IPv4 adress of the rope winch,.
- int **portSafety**

  Port of the object.
- int **portSafety2**

  Port of the safety switch.
- int **invertZ**

  variable if tracking loop should be exited
- **commObject commObj**

1.11.1 Function Documentation

1.11.1.1 calibrate_camera()

```
int calibrate_camera ( )
```

start the camera calibration routine that computes the camera matrix and distortion coefficients

== Initialize Camera SDK ==–

== At this point the Camera SDK is actively looking for all connected cameras and will initialize == them on it's own.

== Get a connected camera ==========-—

== Determine camera resolution

== Set Video Mode ==–

== We set the camera to Segment Mode here. This mode is support by all of our products. == Depending on what device you have connected you might want to consider a different == video mode to achieve the best possible tracking quality. All devices that support a == mode that will achieve a better quality output with a mode other than Segment Mode are == listed here along with what mode you should use if you're looking for the best head

== tracking:

== V100:R1/R2 Precision Mode == TrackIR 5 Bit-Packed Precision Mode == V120 Precision Mode == TBar Precision Mode

== S250e Precision Mode

== If you have questions about a new device that might be conspicuously missing here or == have any questions about head tracking, email support or participate in our forums.

== Start camera output ==–

== Camera Matrix creation ==–

== Ok, start main loop. This loop fetches and displays ===— == camera frames. ===— But first set some camera parameters

the user has to provide the size of one square in mm

== Fetch a new frame from the camera ===—

which is why we also set this constant to 8

later on, when we get the frame as usual:

== Lets have the Camera Library raster the camera's == image into our texture.

imwrite("test.jpg" + , matFrame);

If done with success,

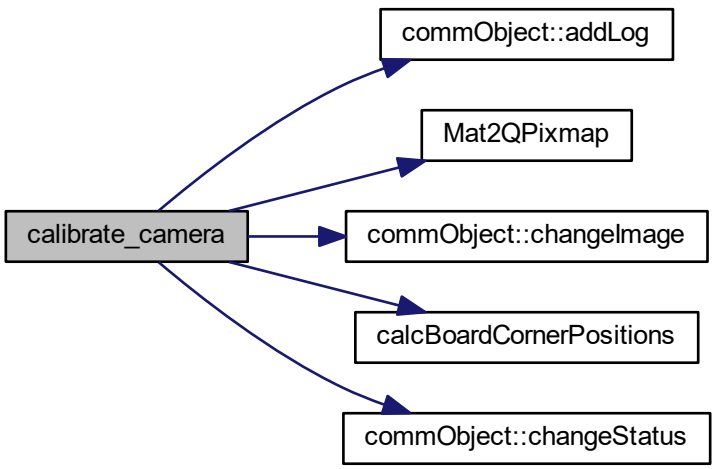
improve the found corners' coordinate accuracy for chessboard

== Release camera ==–

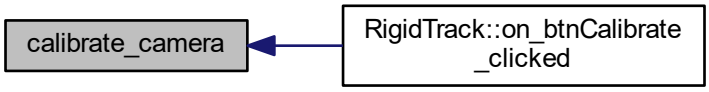
Save the obtained calibration coefficients in a file for later use

- FileStorage::MEMORY);

Here is the call graph for this function:



Here is the caller graph for this function:



1.11.1.2 calibrateGround()

```
int calibrateGround ( )
```

Get the pose of the camera w.r.t the ground calibration frame. This frame sets the navigation frame for later results. The pose is averaged over 200 samples and then saved in the file referenceData.xml. This routine is basically the same as setZero. initialize the variables with starting values

== Initialize Camera SDK ==–

== At this point the Camera SDK is actively looking for all connected cameras and will initialize == them on it's own.

== Get a connected camera ==========-—

== If no device connected, pop a message box and exit ==–

== Determine camera resolution to size application window ==-—

Set camera mode to precision mode, it directly provides marker coordinates

== Start camera output ==–

== Turn on some overlay text so it's clear things are ===— == working even if there is nothing in the camera's view. ===— Set some other parameters as well of the camera

sample some frames and calculate the position and attitude. then average those values and use that as zero position

== Fetch a new frame from the camera ===—

== Ok, we've received a new frame, lets do something == with it.

==for(int i=0; i<frame->ObjectCount(); i++)

sort the 2d points with the correct indices as found in the preceeding order determination algorithm

Compute the pose from the 3D-2D corresponses

project the marker 3d points with the solution into the camera image CoSy and calculate difference to true camera image

Iterative Method needs time to converge to solution

That are not the values of yaw, roll and pitch yet! Rodriguez has to be called first.

==– one sample more :D

== Release camera ==–

Divide by the number of samples to get the mean of the reference position

eulerRef is here in Axis Angle notation

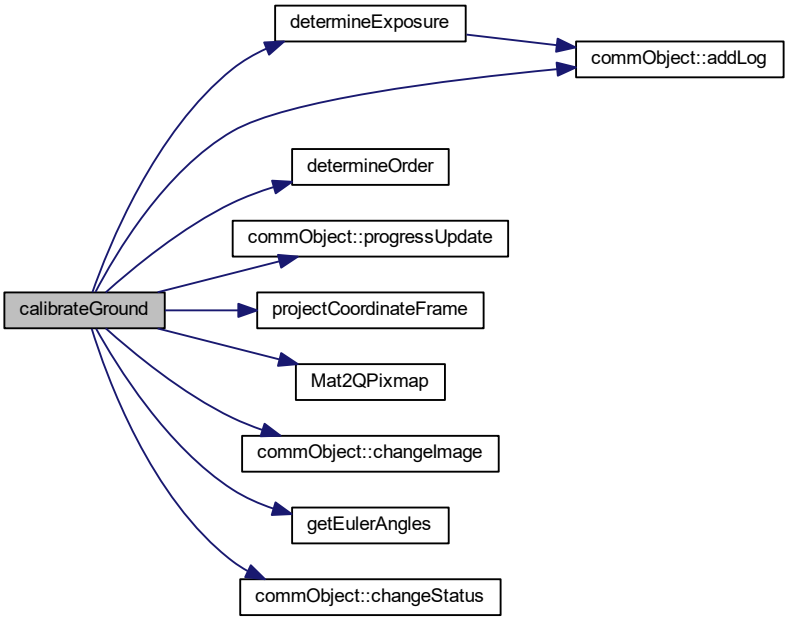
axis angle to rotation matrix ==– Euler Angles, finally

rotation matrix to euler

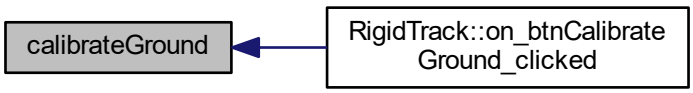
Save the obtained calibration coefficients in a file for later use

- FileStorage::MEMORY);

Here is the call graph for this function:



Here is the caller graph for this function:

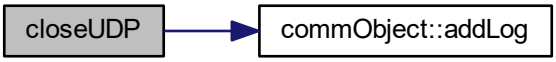


1.11.1.3 closeUDP()

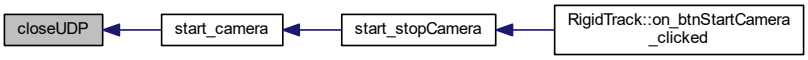
```
void closeUDP ( )
```

close the UDP ports again to release network interfaces etc.

check if the socket is open and if yes close it Here is the call graph for this function:



Here is the caller graph for this function:



1.11.1.4 determineExposure()

```
int determineExposure ( )
```

get the optimal exposure for the camera. For that find the minimum and maximum exposure were the right number of markers are detected

== For OptiTrack Ethernet cameras, it's important to enable development mode if you == want to stop execution for an extended time while debugging without disconnecting == the Ethernet devices. Lets do that now:

== Initialize Camera SDK ==–

== At this point the Camera SDK is actively looking for all connected cameras and will initialize == them on it's own.

== Get a connected camera ==========-—

== If no device connected, pop a message box and exit ==–

== Determine camera resolution to size application window ==-—

set the camera mode to precision mode, it used greyscale imformation for marker property calculations

== Start camera output ==–

== Turn on some overlay text so it's clear things are ===— == working even if there is nothing in the camera's view. ===—

set the camera exposure

set the camera infrared LED intensity

set the camera framerate to 100 Hz

enable the filter that blocks visible light and only passes infrared light

enable high power mode of the leds

enable continuous LED light

set threshold for marker detection

set exposure such that num markers are visible

Number of objects (markers) found in the current picture with the given exposure

exposure when objects detected the first time is numberMarkers

exposure when objects detected is first time numberMarkers+1

set the exposure to the smallest value possible

if the markers arent found after numberTries then there might be no markers at all in the real world

Determine minimum exposure, hence when are numberMarkers objects detected

get a new camera frame

frame received

how many objects are detected in the image

if the right amount if markers is found, exit while loop

not the right amount of markers was found so increase the exposure and try again

Now determine maximum exposure, hence when are numberMarkers+1 objects detected

if the markers arent found after numberTries then there might be no markers at all in the real world

how many objects are detected in the image

if the right amount if markers is found, exit while loop

not the right amount of markers was found so decrease the exposure and try again

set the exposure to the mean of min and max exposure determined

and now check if the correct amount of markers is detected with that new value

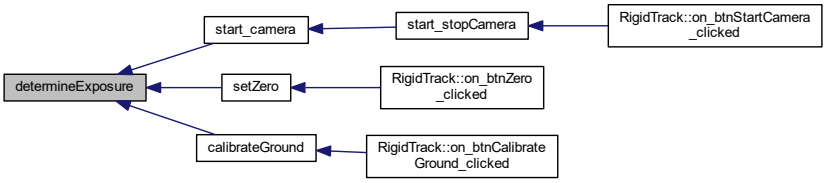
how many objects are detected in the image

are all markers and not more or less detected in the image

== Release camera ==–

all markers and not more or less are found Here is the call graph for this function:

Here is the caller graph for this function:



1.11.1.5 determineOrder()

```
void determineOrder ( )
```

compute the order of the marker points in 2D so they are the same as in the 3D array. Hence marker 1 must be in first place for both, list_points2d and list_points3d determine the 3D-2D correspondences that are crucial for the PnP algorithm Try every possible correspondence and solve PnP Then project the 3D marker points into the 2D camera image and check the difference between projected points and points as seen by the camera the corresponce with the smallest difference is probably the correct one

the difference between true 2D points and projected points is super big

now try every possible permutation of correspondence

reset the starting values for solvePnP

sort the 2d points with the current permutation

Call solve PNP with P3P since its more robust and sufficient for start value determination

set the current difference of all point correspondences to zero

project the 3D points with the solvePnP solution onto 2D

now compute the absolute difference (error)

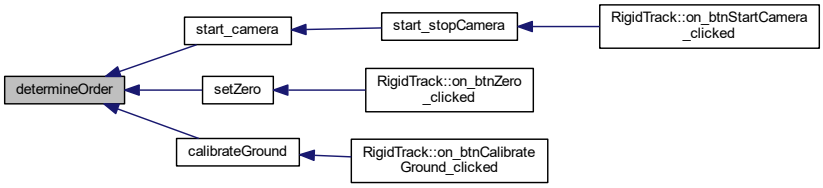
if the difference with the current permutation is smaller than the smallest value till now it is probably the more correct permutation

set the smallest value of difference to the current one

now safe the better permutation

try every permutation

now that the correct order is found assign it to the indices array Here is the caller graph for this function:

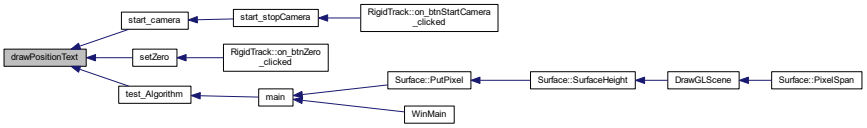


1.11.1.6 drawPositionText()

```
void drawPositionText (
            cv::Mat & Picture,
            cv::Vec3d & Position,
            cv::Vec3d & Euler,
            double error )
```

draw the position, attitude and reprojection error in the picture

Here is the caller graph for this function:

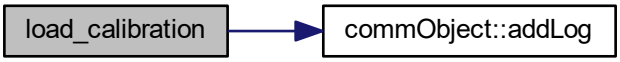


1.11.1.7 load_calibration()

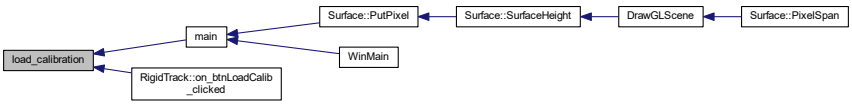
```
void load_calibration (
            int method )
```

Load a previously saved camera calibration from a file.

Here is the call graph for this function:



Here is the caller graph for this function:

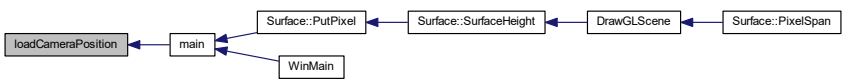


1.11.1.8 loadCameraPosition()

```
void loadCameraPosition ( )
```

load the rotation matrix from camera CoSy to ground CoSy It is determined during ground calibration and once the camera is mounted and fixed stays the same Open the referenceData.xml that contains the rotation from camera CoSy to ground CoSy Here is the call graph for this function:

Here is the caller graph for this function:



1.11.1.9 loadMarkerConfig()

```
void loadMarkerConfig (
            int method )
```

load a marker configuration from file. This file has to be created by hand, use the standard marker configuration file as template

during start up of the programm load the standard marker configuration

open the standard marker configuration file

copy the values to the respective variables

inizialise vectors with correct length depending on the number of markers

save the marker locations in the points3d vector

if the load marker configuration button was clicked show a open file dialog

was cancel or abort clicked

if yes load the standard marker configuration

open the selected marker configuration file

copy the values to the respective variables

inizialise vectors with correct length depending on the number of markers

save the marker locations in the points3d vector

Print out the number of markers and their position to the GUI

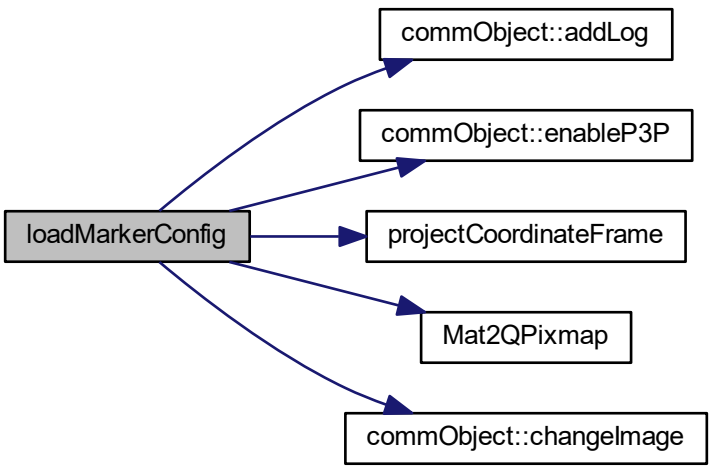
check if P3P algorithm can be enabled, it needs exactly 4 marker points to work

if P3P is possible, let the user choose which algorithm he wants but keep iterative active

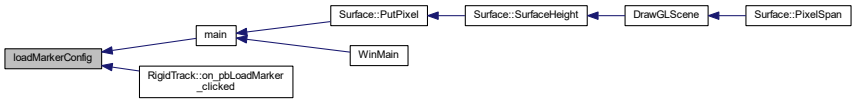
More (or less) marker than 4 loaded, P3P is not possible, hence user cant select P3P in GUI

now display the marker configuration in the camera view

Set the camera pose parallel to the marker coordinate system Here is the call graph for this function:



Here is the caller graph for this function:



1.11.1.10 projectCoordinateFrame()

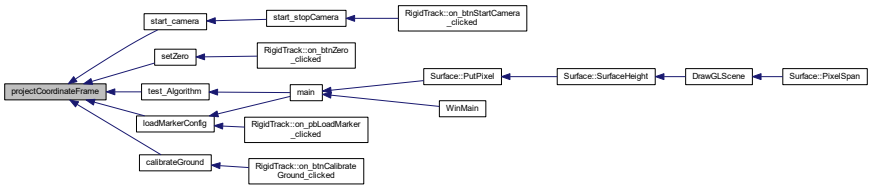
```
void projectCoordinateFrame (
            Mat pictureFrame )
```

project a coordinate CoSy with the rotation and translation of the object for visualization

z-axis

x-axis

y-axis Here is the caller graph for this function:



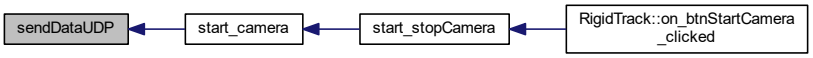
1.11.1.11 sendDataUDP()

```
void sendDataUDP (
            cv::Vec3d & Position,
            cv::Vec3d & Euler )
```

send the position and attitude over UDP to every receiver, the safety receiver is handled on its own in the start_↩ camera function

Roll Pitch Heading

if second receiver is activated send it also the tracking data Here is the caller graph for this function:



1.11.1.12 setHeadingOffset()

```
void setHeadingOffset (
            double d )
```

Add a heading offset to the attitude for the case it is necessary.

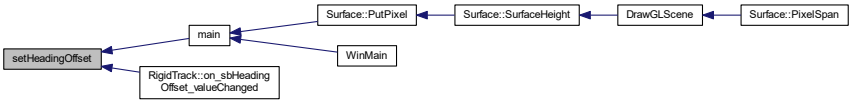
Convert heading offset from degrees to rad

Calculate rotation about x axis

Calculate rotation about y axis

Calculate rotation about z axis

Combined rotation matrix Here is the caller graph for this function:



1.11.1.13 setUpUDP()

```
void setUpUDP ( )
```

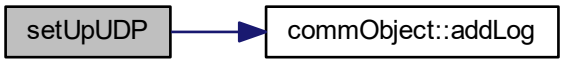
open the UDP ports for communication

Initialise the QDataStream that stores the data to be send

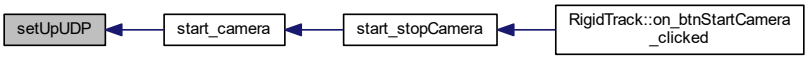
Create UDP slots

if the safety feature is activated open the udp port

if the second receiver feature is activated open the udp port Here is the call graph for this function:



Here is the caller graph for this function:



1.11.1.14 setZero()

```
int setZero ( )
```

determine the initial position of the object that serves as reference point or as ground frame origin

initialize the variables with starting values

== Initialize Camera SDK ==–

== At this point the Camera SDK is actively looking for all connected cameras and will initialize == them on it's own.

== Get a connected camera ==========-—

== If no device connected, pop a message box and exit ==–

== Determine camera resolution to size application window ==-—

Set camera mode to precision mode, it directly provides marker coordinates

== Start camera output ==–

== Turn on some overlay text so it's clear things are ===— == working even if there is nothing in the camera's view. ===— Set some other parameters as well of the camera

sample some frames and calculate the position and attitude. then average those values and use that as zero position

difference between the marker points as seen by the camera and the projected marker points with Rvec and Tvec

== Fetch a new frame from the camera ===—

== Ok, we've received a new frame, lets do something == with it.

==for(int i=0; i<frame->ObjectCount(); i++)

sort the 2d points with the correct indices as found in the preceeding order determination algorithm

Compute the pose from the 3D-2D corresponses

project the marker 3d points with the solution into the camera image CoSy and calculate difference to true camera image

Iterative Method needs time to converge to solution

That are not the values of yaw, roll and pitch yet! Rodriguez has to be called first.

==– one sample more :D

== Release camera ==–

Divide by the number of samples to get the mean of the reference position

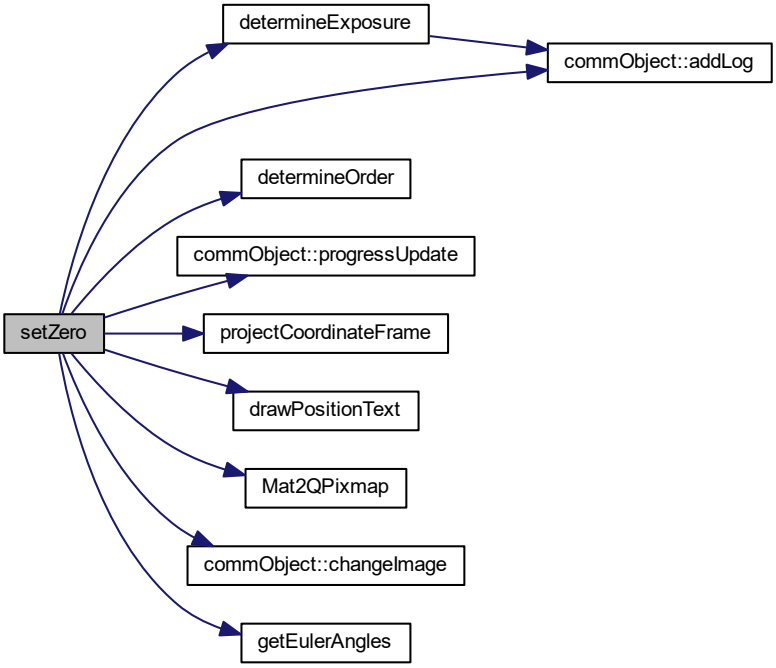
eulerRef is here in Axis Angle notation

axis angle to rotation matrix

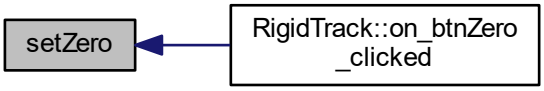
==– Euler Angles, finally

rotation matrix to euler

compute the difference between last obtained TVec and the average Value When it is large the iterative method has not converged properly so it is advised to start the **setZero()** (p. 28) function once again Here is the call graph for this function:



Here is the caller graph for this function:



1.11.1.15 start_camera()

```
int start_camera ( )
```

start the loop that fetches frames, computes the position etc and sends it to other computers

The order of points, hence which entry in list_points3d corresponds to which in list_points2d is not calculated yet

Use the value of Rvec that was set in **main()** (p. 24) as starting value for the solvePnP algorithm

Use the value of Tvec that was set in **main()** (p. 24) as starting value for the solvePnP algorithm

Get the current date and time to name the log file

Concat the log file name as followed. The file is saved in the folder /logs in the Rigid Track installation folder

Convert the QString to a standard string

Get the exposure where the right amount of markers is detected

For OptiTrack Ethernet cameras, it's important to enable development mode if you want to stop execution for an extended time while debugging without disconnecting the Ethernet devices. Lets do that now:

Initialize Camera SDK

At this point the Camera SDK is actively looking for all connected cameras and will initialize them on it's own

Get a connected camera

If no camera can be found, inform user in message log and exit function

Determine camera resolution to size application window

Set the camera mode to precision mode, it used greyscale imformation for marker property calculations

Start camera output

Turn on some overlay text so it's clear things are working even if there is nothing in the camera's view

Set the camera exposure

Set the camera infrared LED intensity

Set the camera framerate to 100 Hz

Enable the filter that blocks visible light and only passes infrared light

Enable high power mode of the LEDs

Disable continuous LED light

Set threshold for marker detection

Create a new matrix that stores the grayscale picture from the camera

QPixmap is the corresponding Qt class that saves images

Matrix that stores the colored picture, hence marker points, coordinate frame and reprojected points

Helper variable used to kick safety switch

Variables for the min and max values that are needed for sanity checks

Ff a marker is not visible or accuracy is bad increase this counter

Equals the quality of the tracking

Open sockets and ports for UDP communication

If the safety feature is enabled send the starting message

Send enable message, hence send a 9 and then a 1

Fetch a new frame from the camera

Get the timestamp of the first frame. This time is subtracted from every subseeding frame so the time starts at 0 in the logs

While no new frame is received loop

Get a new camera frame

There is actually a new frame

Get the time stamp for the first frame. It is subtracted for the following frames

Release the frame so the camera can continue

Exit the while loop

Now enter the main loop that processes each frame and computes the pose, sends it and logs stuff

Check if the user has not pressed "Stop Tracking" yet

Fetch a new frame from the camera

Did we got a new frame or does the camera still need more time

Increase by one, if everything is okay it is decreased at the end of the loop again

Only use this frame it the right number of markers is found in the picture

Get the marker points in 2D in the camera image frame and store them in the list_points2dUnsorted vector The order of points that come from the camera corresponds to the Y coordinate

Was the order already determined? This is false for the first frame and from then on true

Now compute the order

Sort the 2d points with the correct indices as found in the preceeding order determination algorithm

pointOrderIndices was calculated in **determineOrder()** (p. 20)

The first time the 2D-3D corresspondence was determined with gotOrder was okay. But this order can change as the object moves and the marker objects appear in a different order in the frame->Object() array. The solution is that: When a marker point (in the camera image, hence in 2D) was at a position then it wont move that much from one frame to the other. So for the new frame we take a marker object and check which marker was closest this point in the old image frame? This is probably the same (true) marker. And we do that for every other marker as well. When tracking is good and no frames are dropped because of missing markers this should work every frame.

The sum of point distances is set to something unrealistic large

Calculate N_2 norm of unsorted points minus old points

If the norm is smaller than minPointDistance the correspondence is more likely to be correct

Update the array that saves the new point order

Now the new order is found, set the point order to the new value

Save the unsorted position of the marker points for the next loop

Compute the object pose from the 3D-2D corresponses

Project the marker 3d points with the solution into the camera image CoSy and calculate difference to true camera image

Difference of true pose and found pose

Increase the framesDropped variable if accuracy of tracking is too bad

Set number of subsequent frames dropped to zero because error is small enough and no marker was missing

Get the min and max values from TVec for sanity check

Sanity check of values. negative z means the marker CoSy is behind the camera, that's not possible.

Release the frame so the camera can move on

Release the camera

Close all UDP connections so the programm can be closed later on and no resources are locked

Exit the function

Next step is the transformation from camera CoSy to navigation CoSy Compute the relative object position from the reference position to the current one given in the camera CoSy: T_C^{NM} = Tvec - Tvec_{Ref}

Transform the position from the camera CoSy to the navigation CoSy with INS alligned heading and convert from [mm] to [m] T_N^{NM} = M_{NC} T_C^{NM}

Position is the result of the preceeding calculation

Invert Z if check box in GUI is activated, hence height above ground is considered

Realtive angle between reference orientation and current orientation

Convert axis angle respresentation to ordinary rotation matrix

The difference of the reference rotation and the current rotation R_{ NM } = M_{ NC } R_{ CM }

Euler Angles, finally

Get the euler angles from the rotation matrix

Add the heading offset to the heading angle

Compute the velocity with finite differences. Only use is the log file. It is done here because the more precise time stamp can be used

Time between the old frame and the current frame

Set the old frame time to the current one

Calculate the x velocity with finite differences

Calculate the y velocity with finite differences

Calculate the z velocity with finite differences

Set the old position to the current one for next frame velocity calcuation

Send position and Euler angles over WiFi with 100 Hz

Save the values in a log file, values are: Time sinc tracking started Position Euler Angles Velocity

Open the log file, the folder is RigidTrackInstallationFolder/logs

Close the file to save values

Check if the position and euler angles are below the allowed value, if yes send OKAY signal (1), if not send shutdown signal (0) Absolute x, y and z position in navigation CoSy must be smaller than the allowed distance

Absolute Euler angles must be smaller than allowed value. Heading is not considered

Send the OKAY signal to the desired computer every 5th time

Send the 1

reset the counter that is needed for decimation to every 5th time step

The euler angles of the object exceeded the allowed euler angles, send the shutdown signal (0)

Send the shutdown signal, a 0

Inform the user

The position of the object exceeded the allowed position, shut the object down

Send the shutdown signal, a 0

Inform the user

Inform the user if tracking system is disturbed (marker lost or so) or error was too big

Also send the shutdown signal

Send the shutdown signal, a 0

Inform the user

Rasterize the frame so it can be shown in the GUI

Convert the frame from greyscale as it comes from the camera to rgb color

Project (draw) the marker CoSy origin into 2D and save it in the cFrame image

Project the marker points from 3D to the camera image frame (2d) with the computed pose

Draw a circle around the projected points so the result can be better compared to the real marker position In the resulting picture those are the red dots

Write the current position, attitude and error values as text in the frame

Send the new camera picture to the GUI and call the GUI processing routine

Update the picture in the GUI

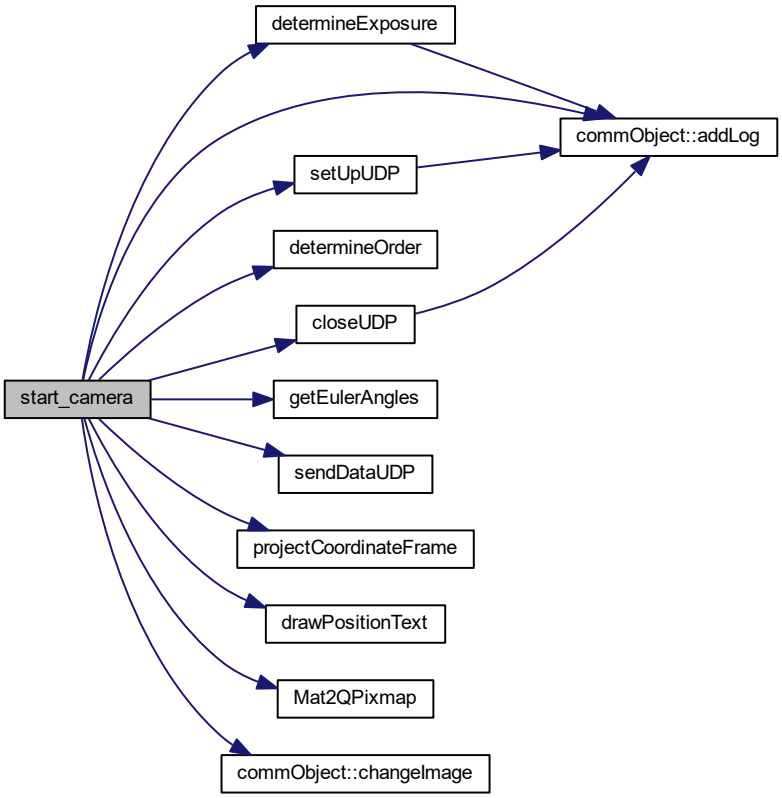
Give Qt time to handle everything

Release the camera frame to fetch the new one

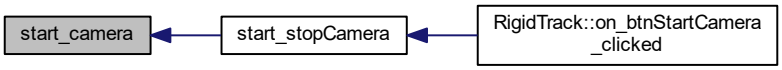
User choose to stop the tracking, clean things up

Close the UDP connections so resources are deallocated

Release camera Here is the call graph for this function:



Here is the caller graph for this function:



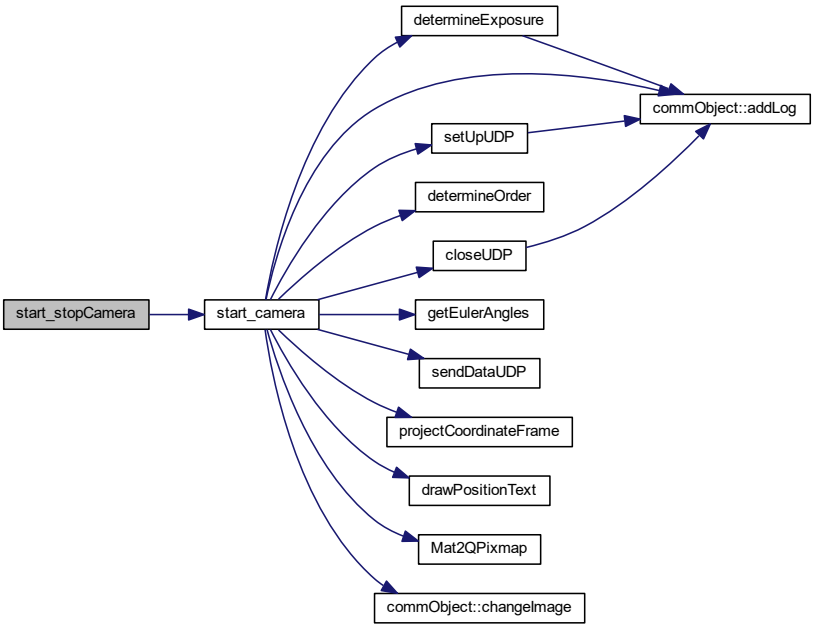
1.11.1.16 start_stopCamera()

```
void start_stopCamera ( )
```

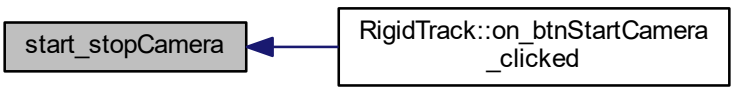
Start or stop the camera depending on if the camera is currently running or not.

tracking is not running so start it

tracking is currently running, set exitRequest to true so the while loop in **start_camera()** (p. 30) exits Here is the call graph for this function:



Here is the caller graph for this function:



1.11.1.17 test_Algorithm()

```
void test_Algorithm ( )
```

project some points from 3D to 2D and then check the accuracy of the algorithms

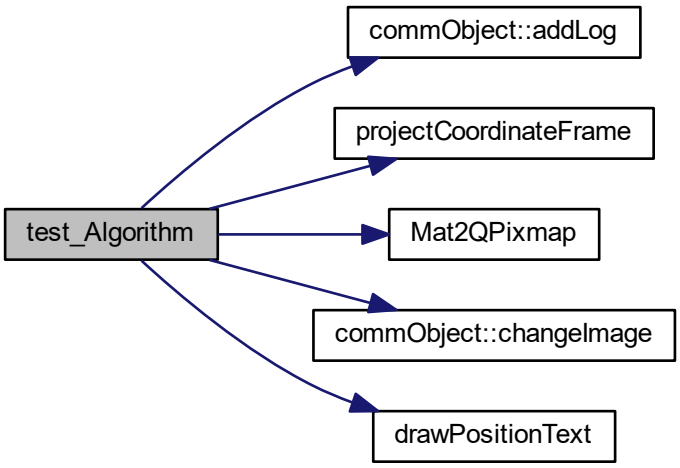
in mm

0 = iterative 1 = EPNP 2 = P3P 4 = UPNP //! not used

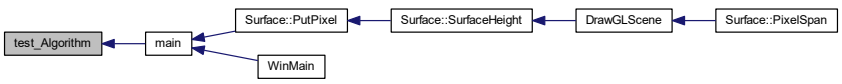
0 = iterative 1 = EPNP 2 = P3P 4 = UPNP //! not used

0 = iterative 1 = EPNP 2 = P3P 4 = UPNP //! not used

0 = iterative 1 = EPNP 2 = P3P 4 = UPNP //! not used Here is the call graph for this function:



Here is the caller graph for this function:



1.11.2 Variable Documentation

1.11.2.1 commObj

```
commObject commObj
```

1.11.2.2 invertZ

```
int invertZ
```

variable if tracking loop should be exited

1.11.2.3 IPAdressObject

```
QHostAddress IPAdressObject
```

socket for the communication with the rope winch

1.11.2.4 IPAdressSafety

```
QHostAddress IPAdressSafety
```

IPv4 adress of the object wifi telemetry chip, can change to 192.168.4.x. This is where the position etc is sent to.

1.11.2.5 IPAdressSafety2

```
QHostAddress IPAdressSafety2
```

IPv4 adress of the circuit breaker, stays the same.

1.11.2.6 methodPNP

```
int methodPNP
```

set to true and the algorithm uses the last result as starting value

1.11.2.7 portObject

```
int portObject
```

IPv4 adress of the rope winch,.

1.11.2.8 portSafety

```
int portSafety
```

Port of the object.

1.11.2.9 portSafety2

```
int portSafety2
```

Port of the safety switch.

1.11.2.10 safety2Enable

```
bool safety2Enable
```

is the safety feature enabled

1.11.2.11 safetyAngle

```
int safetyAngle
```

length of the safety area cube in meters

1.11.2.12 safetyBoxLength

```
double safetyBoxLength
```

is the second receiver enabled

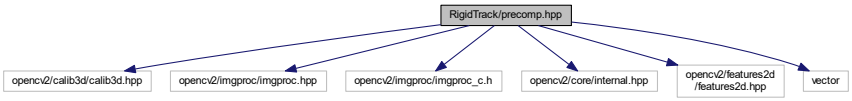
1.11.2.13 safetyEnable

```
bool safetyEnable
```

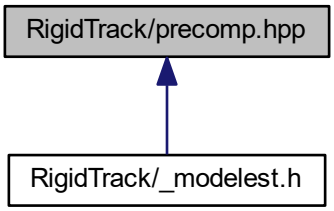
1.12 RigidTrack/precomp.hpp File Reference

```
#include "opencv2/calib3d/calib3d.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/imgproc/imgproc_c.h"
#include "opencv2/core/internal.hpp"
#include "opencv2/features2d/features2d.hpp"
#include <vector>
```

Include dependency graph for precomp.hpp:



This graph shows which files directly or indirectly include this file:



Macros

- #define **GET_OPTIMIZED**(func) (func)

1.12.1 Macro Definition Documentation

1.12.1.1 GET_OPTIMIZED

```
#define GET_OPTIMIZED(
              func ) (func)
```

1.13 RigidTrack/resource.h File Reference

Macros

- #define **IDI_ICON1** 101

  /<{{NO_DEPENDENCIES}} /< Von Microsoft Visual C++ generierte Includedatei. /< Verwendet durch RigidTrack.rc /<

1.13.1 Macro Definition Documentation

1.13.1.1 IDI_ICON1

```
#define IDI_ICON1 101
```

/<{{NO_DEPENDENCIES}} /< Von Microsoft Visual C++ generierte Includedatei. /< Verwendet durch Rigid↩ Track.rc /<

1.14 RigidTrack/RigidTrack.cpp File Reference

```
#include "RigidTrack.h"
#include <QProcess>
#include <QdesktopServices>
#include <QDir>
#include <QUrl>
#include "main.h"
#include "communication.h"
#include <exception>
```

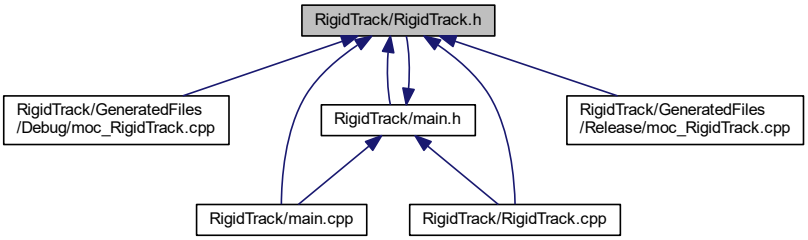
Include dependency graph for RigidTrack.cpp:

1.15 RigidTrack/RigidTrack.h File Reference

```
#include <QtWidgets/QMainWindow>
#include "ui_RigidTrack.h"
#include <qpixmap.h>
#include "main.h"
#include "communication.h"
```

Include dependency graph for RigidTrack.h:

This graph shows which files directly or indirectly include this file:



Classes

- class **RigidTrack**