book

b

Rigid Track

Generated by Doxygen 1.8.13

Table of Contents

Chapter 1

Rigid Track Doxygen Documentation

1.1 Introduction

Rigid Track is a software that provides, combined with an OptiTrack camera, the pose estimation of one object in three dimensional space. This is achieved with only one camera in combination with reflective markers. Those are attached to the object ought to be tracked. The accuracy in the range of millimeters and the high update rate of 100 Hz enable use cases for fast and agile objects. The main application is navigation for drones that rely on high precision position data. Where GPS is not available, e.g. indoors or due to a lacking GPS receiver, this setup substitutes for it. Another use case is the pure pose logging when the drone does not depend on the position, e.g. when it is remote piloted by hand. While this setup contains one OptiTrack Flex 3 camera, every other model of OptiTrack should work, despite not tested. With better camera models, e.g. the Prime Series, even outdoor usage is possible. When the capabilities are not sufficient please refer to OptiTracks Software Motive. But keep in mind that this solution needs at least 3 cameras as Rigid Track works with only one.

1.2 Rigid Track Installation

Start the RigidTrack_setup.exe from the enclosed SD card and follow the instructions given in the installation assistant. Default parameters like installation directory or shortcuts to be created can be chosen. But normally clicking Next and keeping the default values should be sufficient. When the installation is completed a shortcut in the start menu and the desktop can be used to start Rigid Track. The program is then successfully installed in C:/Program Files (x86)/TU Munich FSD/Rigid Track.

1.3 Source Code

The most interesting file for you is main.cpp. It contains the relevant functions for pose estimation. Camera calibration and other functional aspects are also implemented there. The GUI program code is found in RigidTrack.cpp. communication.cpp deals only with communication from main.cpp to the GUI.

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

File Documentation

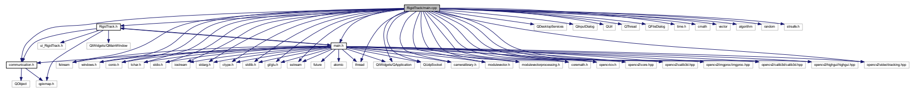
3.1 RigidTrack/main.cpp File Reference

Rigid Track main file that contains most functionallity.

```
#include "RigidTrack.h"
#include "main.h"
#include "communication.h"
#include "cameralibrary.h"
#include "modulevector.h"
#include "modulevectorprocessing.h"
#include "coremath.h"
#include <QtWidgets/QApplication>
#include <QDesktopServices>
#include <QInputDialog>
#include <QUrl>
#include <QThread>
#include <QUdpSocket>
#include <QFileDialog>
#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 dependency graph for main.cpp:



Functions

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

  main initialises the GUI and values for the marker position etc
- QPixmap Mat2QPixmap (cv::Mat src)
- void calcBoardCornerPositions (Size boardSize, float squareSize, std::vector< Point3f > &corners)
- void getEulerAngles (Mat &rotCamerMatrix, Vec3d &eulerAngles)
- int startTracking ()
- void startStopCamera ()

  Start or stop the tracking depending on if the camera is currently running or not.
- int setReference ()
- int calibrateCamera ()

  Start the camera calibration routine that computes the camera matrix and distortion coefficients.
- void loadCalibration (int method)
- void testAlgorithms ()
- void projectCoordinateFrame (Mat pictureFrame)
- void setUpUDP ()

  Open the UDP ports for communication.
- void setHeadingOffset (double d)
- void sendDataUDP (cv::Vec3d &Position, cv::Vec3d &Euler)
- void closeUDP ()
- void loadMarkerConfig (int method)
- void drawPositionText (cv::Mat &Picture, cv::Vec3d &Position, cv::Vec3d &Euler, double error)
- void loadCameraPosition ()
- int determineExposure ()
- void determineOrder ()
- int calibrateGround ()

Variables

- commObject commObj

  class that handles the communication from main.cpp to the GUI
- bool safetyEnable = false

  is the safety feature enabled
- bool safety2Enable = false

  is the second receiver enabled
- double safetyBoxLength = 1.5

  length of the safety area cube in meters
- int safetyAngle = 30

  bank and pitch angle protection in degrees

- bool exitRequested = true

  variable if tracking loop should be exited
- int invertZ = 1

  dummy variable to invert Z direction on request
- double frameTime = 0.01

  100 Hz CoSy rate, is later on replaced with the hardware timestamp delivered by the camera
- double timeOld = 0.0

  old time for finite differences velocity calculation. Is later on replaced with the hardware timestamp delivered by the camera
- double timeFirstFrame = 0

  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 position = Vec3d()

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

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

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

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

  initial position of object in camera CoSy
- Vec3d eulerRef = Vec3d()

  initial euler angle of object respectivley to camera CoSy
- double headingOffset = 0

  heading offset variable for aligning INS heading with tracking heading
- int intIntensity = 15

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

  max is 480 increase if markers are badly visible but should be determined automatically during setReference()
- int intFrameRate = 100

  CoSy rate of camera, maximum is 100 fps.
- int intThreshold = 200

  threshold value for marker detection. If markers are badly visible lower this value but should not be necessary
- Mat Rmat = (cv::Mat_<double>(3, 1) << 0.0, 0.0, 0.0)

  Rotation, translation etc. matrix for PnP results.
- Mat RmatRef = (cv::Mat_<double>(3, 3) << 1., 0., 0., 0., 1., 0., 0., 0., 1.)

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

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

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

  rotation vector (axis-angle notation) from camera CoSy to marker CoSy
- Mat Tvec = (cv::Mat_<double>(3, 1) << 0.0, 0.0, 0.0)

  translation vector from camera CoSy to marker CoSy in camera CoSy
- Mat RvecOriginal

  initial values as start values for algorithms and algorithm tests
- Mat TvecOriginal

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

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

- int methodPNP = 0

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

  number of markers. Is loaded during start up from the marker configuration file
- std::vector< Point3d > list_points3d

  marker positions in marker CoSy
- std::vector< Point2d > list_points2d

  marker positions projected in 2D in camera image CoSy
- std::vector< Point2d > list_points2dOld

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

  difference of the old and new 2D marker position to determine the order of the points
- std::vector< Point2d > list_points2dProjected

  3D marker points projected to 2D in camera image CoSy with the algorithm projectPoints
- std::vector< Point2d > list_points2dUnsorted

  marker points in 2D camera image CoSy, sorted with increasing x (camera image CoSy) but not sorted to correspond with list_points3d
- std::vector< Point3d > coordinateFrame

  coordinate visualisazion of marker CoSy
- std::vector< Point2d > coordinateFrameProjected

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

  old correspondence from list_points3d and list_points_2d
- int pointOrderIndicesNew [ ] = { 0, 1, 2, 3 }

  new correspondence from list_points3d and list_points_2d
- double currentPointDistance = 5000

  distance from the projected 3D points (hence in 2d) to the real 2d marker positions in camera image CoSy
- double minPointDistance = 5000

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

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

  order of the list_points3d and list_points3d already tetermined or not, has to be done once
- bool camera_started = false

  variable thats needed to exit the main while loop
- Mat cameraMatrix

  camera matrix of the camera
- Mat distCoeffs

  distortion coefficients of the camera
- Core::DistortionModel distModel

  distortion model of the camera
- QUdpSocket ∗ udpSocketObject

  socket for the communication with receiver 1
- QUdpSocket ∗ udpSocketSafety

  socket for the communication with safety receiver
- QUdpSocket ∗ udpSocketSafety2

  socket for the communication with receiver 3
- QHostAddress IPAdressObject = QHostAddress("127.0.0.1")

  IPv4 adress of receiver 1.
- QHostAddress IPAdressSafety = QHostAddress("192.168.4.1")

IPv4 adress of safety receiver.
- QHostAddress IPAdressSafety2 = QHostAddress("192.168.4.4")

  IPv4 adress of receiver 2.
- int portObject = 9155

  Port of receiver 1.
- int portSafety = 9155

  Port of the safety receiver.
- int portSafety2 = 9155

  Port of receiver 2.
- QByteArray datagram

  data package that is sent to receiver 1 and 2
- QByteArray data

  data package that's sent to the safety receiver
- const int BACKBUFFER_BITSPERPIXEL = 8

  8 bit per pixel and greyscale image from camera
- std::string strBuf

  buffer that holds the strings that are sent to the Qt GUI
- std::stringstream ss

  stream that sends the strBuf buffer to the Qt GUI
- QString logFileName

  Filename for the logfiles.
- std::string logName

  Filename for the logfiles as standard string.
- SYSTEMTIME logDate

  Systemtime struct that saves the current date and time thats needed for the log file name creation.
- std::ofstream logfile

  file handler for writing the log file

3.1.1 Detailed Description

Rigid Track main file that contains most functionallity.

This file contains allmost all functional code for pose estimation, calibration and so on. The GUI related part is in RigidTrack.cpp and the communication from main.cpp to GUI is done with the commObj class from communication.cpp.

Author

Florian J.T. Wachter

Version

1.0

Date

April, 8th 2017

3.1.2 Function Documentation

3.1.2.1 calcBoardCornerPositions()

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

Calculate the chess board corner positions, used for the camera calibration.

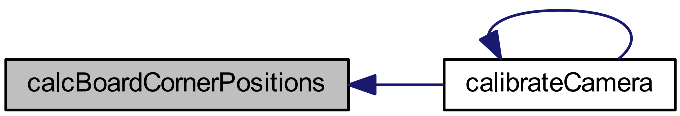
Parameters

in	*boardSize*	denotes how many squares are in each direction.
in	*squareSize*	is the square length in millimeters.
out	*corners*	returns the square corners in millimeters.

Definition at line 238 of file main.cpp.

```
{
    corners.clear();

    for (int i = 0; i < boardSize.height; ++i)
        for (int j = 0; j < boardSize.width; ++j)
            corners.push_back(Point3f(float(j*squareSize), float(i*squareSize), 0));
}
```

Here is the caller graph for this function:



3.1.2.2 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 setReference.

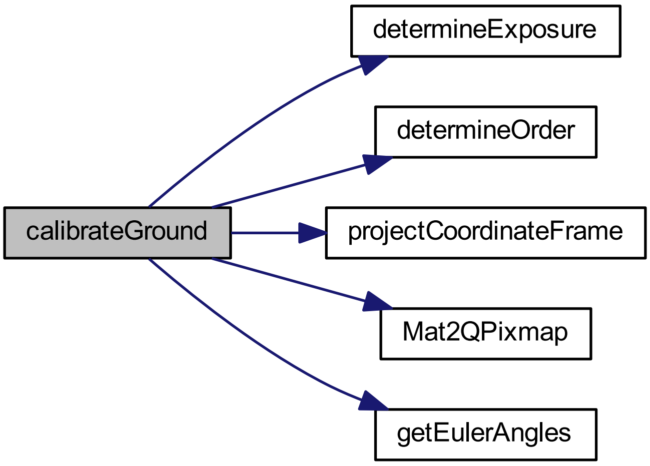
Definition at line 1581 of file main.cpp.

```
{
    //! initialize the variables with starting values
    gotOrder = false;
    posRef = 0;
    eulerRef = 0;
    RmatRef = 0;
    Rvec = RvecOriginal;
    Tvec = TvecOriginal;

    determineExposure();

    ss.str("");
    commObj.addLog("Started ground calibration");

    CameraLibrary_EnableDevelopment();
    //! Initialize Camera SDK ==--
    CameraLibrary::CameraManager::X();

    //! 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 ================----
    CameraManager::X().WaitForInitialization();
    Camera *camera = CameraManager::X().GetCamera();

    //! If no device connected, pop a message box and exit ==--
    if (camera == 0)
    {
        commObj.addLog("No camera found!");
        return 1;
    }

    //! Determine camera resolution to size application window ==----
    int cameraWidth = camera->Width();
    int cameraHeight = camera->Height();
    camera->GetDistortionModel(distModel);
    cv::Mat matFrame(cv::Size(cameraWidth, cameraHeight), CV_8UC1);

    //! Set camera mode to precision mode, it directly provides marker coordinates
    camera->SetVideoType(Core::PrecisionMode);

    //! Start camera output ==--
    camera->Start();

    //! 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
    camera->SetTextOverlay(true);
    camera->SetFrameRate(intFrameRate);
    camera->SetIntensity(intIntensity);
    camera->SetIRFilter(true);
    camera->SetContinuousIR(false);
    camera->SetHighPowerMode(false);

    //! sample some frames and calculate the position and attitude. then average those values and use that as zero position
    int numberSamples = 0;
    int numberToSample = 200;
    double projectionError = 0;

    while (numberSamples < numberToSample)
    {
        //! Fetch a new frame from the camera ===---
        Frame *frame = camera->GetFrame();

        if (frame)
        {
            //! Ok, we've received a new frame, lets do something
            //! with it.
            if (frame->ObjectCount() == numberMarkers)
            {
                //!for(int i=0; i<frame->ObjectCount(); i++)
                for (int i = 0; i < numberMarkers; i++)
                {
                    cObject *obj = frame->Object(i);
                    list_points2dUnsorted[i] = cv::Point2d(obj->X(), obj->Y());
                }

                if (gotOrder == false)
                {
                    determineOrder();
                }

                //! sort the 2d points with the correct indices as found in the preceeding order determination algorithm
                for (int w = 0; w < numberMarkers; w++)
                {
```

```
                    list_points2d[w] = list_points2dUnsorted[pointOrderIndices[w]];
                }
                list_points2dOld = list_points2dUnsorted;

                //!Compute the pose from the 3D-2D corresponses
                solvePnP(list_points3d, list_points2d, cameraMatrix, distCoeffs, Rvec, Tvec, useGuess, methodPNP);

                //! project the marker 3d points with the solution into the camera image CoSy and calculate difference to true camera image
                projectPoints(list_points3d, Rvec, Tvec, cameraMatrix, distCoeffs, list_points2dProjected);
                projectionError = norm(list_points2dProjected, list_points2d);

                if (projectionError > 3)
                {
                    commObj.addLog("Reprojection error is bigger than 3 pixel. Correct marker configuration loaded?\nMarker position measured precisely?");
                    frame->Release();
                    return 1;
                }

                double maxValue = 0;
                double minValue = 0;
                minMaxLoc(Tvec.at<double>(2), &minValue, &maxValue);

                if (maxValue > 10000 || minValue < 0)
                {


                    commObj.addLog("Negative z distance, thats not possible. Start the set zero routine again and check marker configurations.");
                    frame->Release();
                    return 1;
                }

                if (norm(positionOld) - norm(Tvec) < 0.05)   //!<Iterative Method needs time to converge to solution
                {
                    add(posRef, Tvec, posRef);
                    add(eulerRef, Rvec, eulerRef); //!< That are not the values of yaw, roll and pitch yet! Rodriguez has to be called first.
                    numberSamples++;    //!<-- one sample more :D
                    commObj.progressUpdate(numberSamples * 100 / numberToSample);
                }
                positionOld = Tvec;

                Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
                for (int i = 0; i < numberMarkers; i++)
                {
                    circle(cFrame, Point(list_points2d[i].x, list_points2d[i].y), 6, Scalar(0, 225, 0), 3);
                }
                projectCoordinateFrame(cFrame);
                projectPoints(list_points3d, Rvec, Tvec, cameraMatrix, distCoeffs, list_points2d);
                for (int i = 0; i < numberMarkers; i++)
                {
                    circle(cFrame, Point(list_points2d[i].x, list_points2d[i].y), 3, Scalar(225, 0, 0), 3);
                }

                QPixmap QPFrame;
                QPFrame = Mat2QPixmap(cFrame);
                commObj.changeImage(QPFrame);
                QCoreApplication::processEvents();

            }
            frame->Release();
        }
    }
    //! Release camera ==--
    camera->Release();

    //!Divide by the number of samples to get the mean of the reference position
    divide(posRef, numberToSample, posRef);
    divide(eulerRef, numberToSample, eulerRef); //!< eulerRef is here in Axis Angle notation

    Rodrigues(eulerRef, RmatRef);                //!< axis angle to rotation matrix

    getEulerAngles(RmatRef, eulerRef); //!<  rotation matrix to euler
    ss.str("");
    ss << "RmatRef is:\n";
```

```
    ss << RmatRef << "\n";
    ss << "Reference Position is:\n";
    ss << posRef << "[mm] \n";
    ss << "Reference Euler angles are:\n";
    ss << eulerRef << "[deg] \n";

    //! Save the obtained calibration coefficients in a file for later use
    QString fileName = QFileDialog::getSaveFileName(nullptr, "Save ground calibration file", "
referenceData.xml", "Calibration File (*.xml);;All Files (*)");
    FileStorage fs(fileName.toUtf8().constData(), FileStorage::WRITE);
    fs << "M_NC" << RmatRef;
    fs << "eulerRef" << eulerRef;
    strBuf = fs.releaseAndGetString();
    commObj.changeStatus(QString::fromStdString(strBuf));
    commObj.addLog("Saved ground calibration!");
    commObj.progressUpdate(0);
    return 0;
}
```

Here is the call graph for this function:



3.1.2.3 closeUDP()

```
void closeUDP ( )
```

Close the UDP ports again to release network interfaces etc. If this is not done the network resources are still occupied and the program can't exit properly.

Definition at line 1191 of file main.cpp.

```
{
    //! check if the socket is open and if yes close it
    if (udpSocketObject->isOpen())
    {
        udpSocketObject->close();
    }

```

```
1199     if (udpSocketSafety->isOpen())
1200     {
1201         udpSocketSafety->close();
1202     }
1203
1204     if (udpSocketSafety2->isOpen())
1205     {
1206         udpSocketSafety2->close();
1207     }
1208     commObj.addLog("Closed all UDP ports.");
1209 }
```

Here is the caller graph for this function:

3.1.2.4 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. Then the mean of those two values is used as exposure.

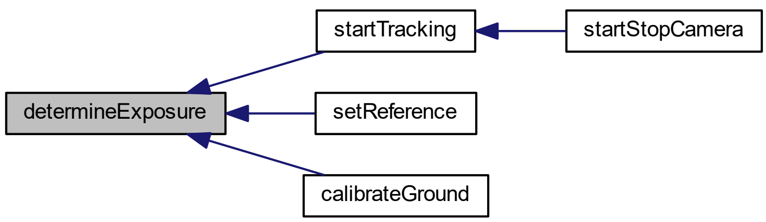
Definition at line 1380 of file main.cpp.

```
1381 {
1382     //! For OptiTrack Ethernet cameras, it's important to enable development mode if you
1383     //! want to stop execution for an extended time while debugging without disconnecting
1384     //! the Ethernet devices.  Lets do that now:
1385
1386     CameraLibrary_EnableDevelopment();
1387
1388     //! Initialize Camera SDK ==--
1389     CameraLibrary::CameraManager::X();
1390
1391     //! At this point the Camera SDK is actively looking for all connected cameras and will initialize
1392     //! them on it's own.
1393
1394     //! Get a connected camera ===================----
1395     CameraManager::X().WaitForInitialization();
1396     Camera *camera = CameraManager::X().GetCamera();
1397
1398     //! If no device connected, pop a message box and exit ==--
1399     if (camera == 0)
1400     {
1401         commObj.addLog("No camera found!");
1402         return 1;
1403     }
1404
1405     //! Determine camera resolution to size application window ==----
1406     int cameraWidth = camera->Width();
1407     int cameraHeight = camera->Height();
1408
1409     camera->SetVideoType(Core::PrecisionMode);  //! set the camera mode to precision mode, it used
       greyscale information for marker property calculations
1410
1411                                                 //! Start camera output ==--
1412     camera->Start();
1413
```

```
    //! Turn on some overlay text so it's clear things are     ===---
    //! working even if there is nothing in the camera's view. ===---
    camera->SetTextOverlay(true);
    camera->SetExposure(intExposure);    //! set the camera exposure
    camera->SetIntensity(intIntensity); //! set the camera infrared LED intensity
    camera->SetFrameRate(intFrameRate); //! set the camera framerate to 100 Hz
    camera->SetIRFilter(true);  //! enable the filter that blocks visible light and only passes infrared light
    camera->SetHighPowerMode(true); //! enable high power mode of the leds
    camera->SetContinuousIR(false); //! enable continuous LED light
    camera->SetThreshold(intThreshold); //! set threshold for marker detection

    //!set exposure such that num markers are visible
    int numberObjects = 0;  //! Number of objects (markers) found in the current picture with the given exposure
    int minExposure = 1;    //! exposure when objects detected the first time is numberMarkers
    int maxExposure = 480;  //! exposure when objects detected is first time numberMarkers+1
    intExposure = minExposure;   //! set the exposure to the smallest value possible
    int numberTries = 0;    //! 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
    camera->SetExposure(intExposure);
    while (numberObjects != numberMarkers && numberTries < 48)
    {
        //! get a new camera frame
        Frame *frame = camera->GetFrame();
        if (frame) //! frame received
        {
            numberObjects = frame->ObjectCount();    //! how many objects are detected in the image
            if (numberObjects == numberMarkers) { minExposure = intExposure; frame->Release(); break; } //! 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
            numberTries++;
            intExposure += 10;
            camera->SetExposure(intExposure);
            ss.str("");
            ss << "Exposure: " << intExposure << "\t";
            ss << "Objects found: " << numberObjects;
            commObj.addLog(QString::fromStdString(ss.str()));
            frame->Release();
        }
    }

    //! Now determine maximum exposure, hence when are numberMarkers+1 objects detected
    numberTries = 0;    //! if the markers arent found after numberTries then there might be no markers at all in the real world
    intExposure = maxExposure;
    camera->SetExposure(intExposure);
    numberObjects = 0;
    while (numberObjects != numberMarkers && numberTries < 48)
    {
        Frame *frame = camera->GetFrame();
        if (frame)
        {
            numberObjects = frame->ObjectCount(); //! how many objects are detected in the image
            if (numberObjects == numberMarkers) { maxExposure = intExposure; frame->Release(); break; } //! 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
            intExposure -= 10;
            numberTries++;
            camera->SetExposure(intExposure);
            ss.str("");
            ss << "Exposure: " << intExposure << "\t";
            ss << "Objects found: " << numberObjects;
            commObj.addLog(QString::fromStdString(ss.str()));
            frame->Release();
        }
    }

    //! set the exposure to the mean of min and max exposure determined
    camera->SetExposure((minExposure + maxExposure) / 2.0);

    //! and now check if the correct amount of markers is detected with that new value
    while (1)
    {
        Frame *frame = camera->GetFrame();
        if (frame)
        {
            numberObjects = frame->ObjectCount(); //! how many objects are detected in the image
            if (numberObjects != numberMarkers) //! are all markers and not more or less detected in the image
            {
                frame->Release();
```

```
                commObj.addLog("Was not able to detect the right amount of markers.");
                //! Release camera ==--
                camera->Release();
                return 1;
            }
            else  //! all markers and not more or less are found
            {
                frame->Release();
                intExposure = (minExposure + maxExposure) / 2.0;
                commObj.addLog("Found the correct number of markers.");
                commObj.addLog("Exposure set to:");
                commObj.addLog(QString::number(intExposure));
                break;
            }
        }
    }

    camera->Release();
    return 0;

}
```

Here is the caller graph for this function:



3.1.2.5 determineOrder()

```
void determineOrder ( )
```

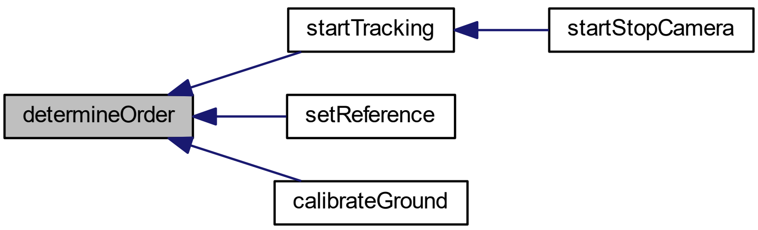
Compute the order of the marker points in 2D so they are the same as in the 3D array. Hence marker 1 must be in first place for both, list_points2d and list_points3d.

Definition at line 1516 of file main.cpp.

```
{
    //! 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
    minPointDistance = 5000;
    std::sort(pointOrderIndices, pointOrderIndices + 4);

    //! now try every possible permutation of correspondence
    do {
        //! reset the starting values for solvePnP
        Rvec = RvecOriginal;
        Tvec = TvecOriginal;

        //! sort the 2d points with the current permutation
```

```
        for (int m = 0; m < numberMarkers; m++)
        {
            list_points2d[m] = list_points2dUnsorted[
pointOrderIndices[m]];
        }

        //! Call solve PNP with P3P since its more robust and sufficient for start value determination
        solvePnP(list_points3d, list_points2d,
cameraMatrix, distCoeffs, Rvec, Tvec, useGuess, SOLVEPNP_P3P);

        //! set the current difference of all point correspondences to zero
        currentPointDistance = 0;

        //! project the 3D points with the solvePnP solution onto 2D
        projectPoints(list_points3d, Rvec, Tvec,
cameraMatrix, distCoeffs, list_points2dProjected);

        //! now compute the absolute difference (error)
        for (int n = 0; n < numberMarkers; n++)
        {
            currentPointDistance += norm(list_points2d[n] -
list_points2dProjected[n]);
        }

        //! if the difference with the current permutation is smaller than the smallest value till now
        //! it is probably the more correct permutation
        if (currentPointDistance < minPointDistance)
        {
            minPointDistance = currentPointDistance;    //!< set the
 smallest value of difference to the current one
            for (int b = 0; b < numberMarkers; b++)    //!< now safe the better permutation
            {
                pointOrderIndicesNew[b] = pointOrderIndices[b];
            }
        }


    }
    //! try every permutation
    while (std::next_permutation(pointOrderIndices,
pointOrderIndices + 4));

    //! now that the correct order is found assign it to the indices array
    for (int w = 0; w < numberMarkers; w++)
    {
        pointOrderIndices[w] = pointOrderIndicesNew[w];
    }
    gotOrder = true;
}
```

Here is the caller graph for this function:



3.1.2.6 drawPositionText()

```
void drawPositionText (
            cv::Mat & Picture,
```

```
cv::Vec3d & Position,
cv::Vec3d & Euler,
double error )
```

Draw the position, attitude and reprojection error in the picture.

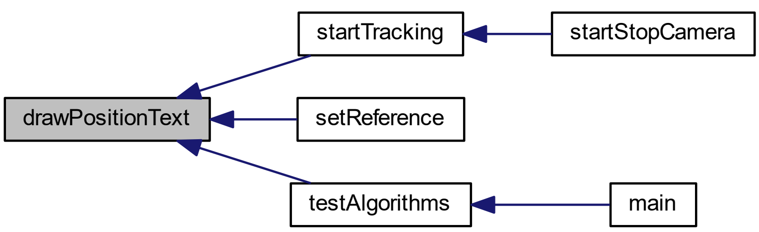
Parameters

in	*Picture*	is the camera image in OpenCV matrix format.
in	*Position*	is the position of the tracked object in navigation CoSy.
in	*Euler*	are the Euler angles with respect to the navigation frame.
in	*error*	is the reprojection error of the pose estimation.

Definition at line 1333 of file main.cpp.

```
{
    ss.str("");
    ss << "X: " << Position[0] << " m";
    putText(Picture, ss.str(), cv::Point(200, 440), 1, 1, cv::Scalar(255, 255, 255));

    ss.str("");
    ss << "Y: " << Position[1] << " m";
    putText(Picture, ss.str(), cv::Point(200, 455), 1, 1, cv::Scalar(255, 255, 255));

    ss.str("");
    ss << "Z: " << Position[2] << " m";
    putText(Picture, ss.str(), cv::Point(200, 470), 1, 1, cv::Scalar(255, 255, 255));

    ss.str("");
    ss << "Heading: " << Euler[2]*180/3.1415 << " deg";
    putText(Picture, ss.str(), cv::Point(350, 440), 1, 1, cv::Scalar(255, 255, 255));

    ss.str("");
    ss << "Pitch: " << Euler[1] * 180 / 3.1415 << " deg";
    putText(Picture, ss.str(), cv::Point(350, 455), 1, 1, cv::Scalar(255, 255, 255));

    ss.str("");
    ss << "Roll: " << Euler[0] * 180 / 3.1415 << " deg";
    putText(Picture, ss.str(), cv::Point(350, 470), 1, 1, cv::Scalar(255, 255, 255));

    ss.str("");
    ss << "Error: " << error << " px";
    putText(Picture, ss.str(), cv::Point(10, 470), 1, 1, cv::Scalar(255, 255, 255));
}
```

Here is the caller graph for this function:



3.1.2.7 getEulerAngles()

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

As explained Euler angles can be extracted from an ordinary rotation matrix. As the OpenCV documentation states the embedded function decomposeProjectionMatrix decomposes a projection matrix into a rotation matrix, camera matrix and Euler angles. But in Rigid Track rotCamerMatrix is always only a pure rotation matrix and the camera matrix equals the unit matrix. Get the euler angles from a rotation matrix

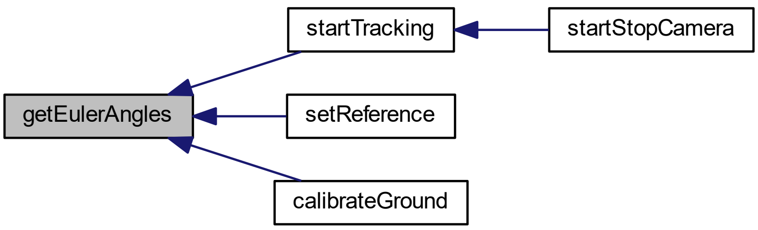
Parameters

in	*rotCamerMatrix*	is a projection matrix, here normally only the extrinsic values.
out	*eulerAngles*	contains the Euler angles that result in the same rotation matrix as rotCamerMatrix.

Definition at line 256 of file main.cpp.

```
256                                                             {
257
258     Mat cameraMatrix, rotMatrix, transVect, rotMatrixX, rotMatrixY, rotMatrixZ;
259     double* _r = rotCamerMatrix.ptr<double>();
260     double projMatrix[12] = { _r[0],_r[1],_r[2],0,
261         _r[3],_r[4],_r[5],0,
262         _r[6],_r[7],_r[8],0 };
263
264     decomposeProjectionMatrix(Mat(3, 4, CV_64FC1, projMatrix),
265         cameraMatrix,
266         rotMatrix,
267         transVect,
268         rotMatrixX,
269         rotMatrixY,
270         rotMatrixZ,
271         eulerAngles);
272 }
```

Here is the caller graph for this function:



3.1.2.8 loadCalibration()

```
void loadCalibration (
            int method )
```

Load a previously saved camera calibration from a file.

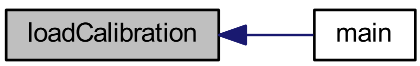
Parameters

in	*method*	whether or not load the camera calibration from calibration.xml. If ==0 then yes, if != 0 then let the user select a different file.

Definition at line 941 of file main.cpp.

```
941                                 {
942 
943     QString fileName;
944     if (method == 0)
945     {
946         fileName = "calibration.xml";
947     }
948     else
949     {
950         fileName = QFileDialog::getOpenFileName(nullptr, "Choose a previous saved calibration file", "", "Calibration Files (*.xml);;All Files (*)");
951         if (fileName.length() == 0)
952         {
953             fileName = "calibration.xml";
954         }
955     }
956     FileStorage fs;
957     fs.open(fileName.toUtf8().constData(), FileStorage::READ);
958     fs["CameraMatrix"] >> cameraMatrix;
959     fs["DistCoeff"] >> distCoeffs;
960     commObj.addLog("Loaded calibration from file:");
961     commObj.addLog(fileName);
962     ss.str("");
963     ss << "\nCamera Matrix is" << "\n" << cameraMatrix << "\n";
964     ss << "\nDistortion Coefficients are" << "\n" << distCoeffs << "\n";
965     commObj.addLog(QString::fromStdString(ss.str()));
966 }
```

Here is the caller graph for this function:



3.1.2.9 loadCameraPosition()

```
void loadCameraPosition ( )
```

Load the rotation matrix from camera CoSy to ground CoSy It is determined during calibrateGround() and stays the same once the camera is mounted and fixed.

Definition at line 1366 of file main.cpp.

```
{
    //! Open the referenceData.xml that contains the rotation from camera CoSy to ground CoSy
    FileStorage fs;
    fs.open("referenceData.xml", FileStorage::READ);
    fs["M_NC"] >> M_CN;
    fs["M_NC"] >> RmatRef;
    fs["posRef"] >> posRef;
    fs["eulerRef"] >> eulerRef;
    commObj.addLog("Loaded reference pose.");
}
```

Here is the caller graph for this function:

3.1.2.10 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.

Parameters

in	*method*	whether or not load the configuration from the markerStandard.xml. If ==0 load it, if != 0 let the user select a different file.

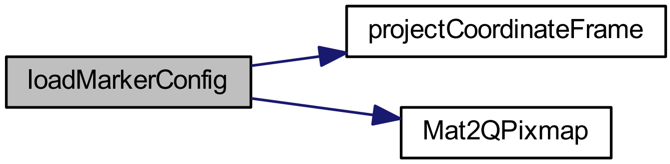
Definition at line 1213 of file main.cpp.

```
{
    QString fileName;
    //! during start up of the programm load the standard marker configuration
    if (method == 0)
    {
        //! open the standard marker configuration file
        FileStorage fs;
        fs.open("markerStandard.xml", FileStorage::READ);

        //! copy the values to the respective variables
        fs["numberMarkers"] >> numberMarkers;

        //! inizialise vectors with correct length depending on the number of markers
        list_points3d = std::vector<Point3d>(numberMarkers);
        list_points2d = std::vector<Point2d>(numberMarkers);
        list_points2dOld = std::vector<Point2d>(numberMarkers);
        list_points2dDifference = std::vector<double>(
  numberMarkers);
        list_points2dProjected = std::vector<Point2d>(
  numberMarkers);
        list_points2dUnsorted = std::vector<Point2d>(
  numberMarkers);

```

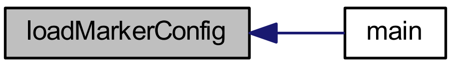
```
        //! save the marker locations in the points3d vector
        fs["list_points3d"] >> list_points3d;
        fs.release();
        commObj.addLog("Loaded marker configuration from file:");
        commObj.addLog(fileName);



    }
    else
    {
        //! if the load marker configuration button was clicked show a open file dialog
        fileName = QFileDialog::getOpenFileName(nullptr, "Choose a previous saved marker configuration file
", "", "marker configuratio files (*.xml);;All Files (*)");

        //! was cancel or abort clicked
        if (fileName.length() == 0)
        {
            //! if yes load the standard marker configuration
            fileName = "markerStandard.xml";
        }

        //! open the selected marker configuration file
        FileStorage fs;
        fs.open(fileName.toUtf8().constData(), FileStorage::READ);

        //! copy the values to the respective variables
        fs["numberMarkers"] >> numberMarkers;

        //! inizialise vectors with correct length depending on the number of markers
        list_points3d = std::vector<Point3d>(numberMarkers);
        list_points2d = std::vector<Point2d>(numberMarkers);
        list_points2dOld = std::vector<Point2d>(numberMarkers);
        list_points2dDifference = std::vector<double>(numberMarkers);
        list_points2dProjected = std::vector<Point2d>(numberMarkers);
        list_points2dUnsorted = std::vector<Point2d>(numberMarkers);

        //! save the marker locations in the points3d vector
        fs["list_points3d"] >> list_points3d;
        fs.release();
        commObj.addLog("Loaded marker configuration from file:");
        commObj.addLog(fileName);

    }

    //! Print out the number of markers and their position to the GUI
    ss.str("");
    ss << "Number of Markers: " << numberMarkers << "\n";
    ss << "Marker 3D Points X,Y and Z [mm]: \n";
    for (int i = 0; i < numberMarkers; i++)
    {
        ss << "Marker " << i + 1 << ":\t" << list_points3d[i].x << "\t" << list_points3d[i].y << "\t" <<
list_points3d[i].z << "\n";
    }
    commObj.addLog(QString::fromStdString(ss.str()));

    //! check if P3P algorithm can be enabled, it needs exactly 4 marker points to work
    if (numberMarkers == 4)
    {
        //! if P3P is possible, let the user choose which algorithm he wants but keep iterative active
        methodPNP = 0;
        commObj.enableP3P(true);
    }
    else
    {
        //! More (or less) marker than 4 loaded, P3P is not possible, hence user cant select P3P in GUI
        methodPNP = 0;
        commObj.enableP3P(false);
        commObj.addLog("P3P algorithm disabled, only works with 4 markers.");
    }

    //! now display the marker configuration in the camera view
    Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));

    //! Set the camera pose parallel to the marker coordinate system
    Tvec.at<double>(0) = 0;
    Tvec.at<double>(1) = 0;
    Tvec.at<double>(2) = 4500;
    Rvec.at<double>(0) = 0 * 3.141592653589 / 180.0;
    Rvec.at<double>(1) = 0 * 3.141592653589 / 180.0;
    Rvec.at<double>(2) = -90. * 3.141592653589 / 180.0;

    projectPoints(list_points3d, Rvec, Tvec, cameraMatrix,
distCoeffs, list_points2dProjected);
    for (int i = 0; i < numberMarkers; i++)
    {
        circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 3, Scalar(255, 0, 0
```

```
        ), 3);
    }

    projectCoordinateFrame(cFrame);
    QPixmap QPFrame;
    QPFrame = Mat2QPixmap(cFrame);
    commObj.changeImage(QPFrame);
    QCoreApplication::processEvents();

}
```

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.2.11 main()

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

main initialises the GUI and values for the marker position etc

Both function arguments are not used. After initializing the QApplication and GUI the Rigid Track version and build date is added to the message log. In the next lines Tvec and Rvec are set to start values and the coordinate frame thats shown during tracking is created.After setting position, velocity and Euler angles to default values a heading offset of zero degrees is set. Then the calibrated camera pose, camera calibration and standard marker configuration are loaded.Finally the solvePnP algorithm is tested with test_Algorithm(). Now the programm is fully loaded and waits for events. First the GUI is set up with Signals and Slots, see Qt docu for how that works. Then some variables are initialized with arbitrary values. At last calibration and marker configuration etc. are loaded from xml files.

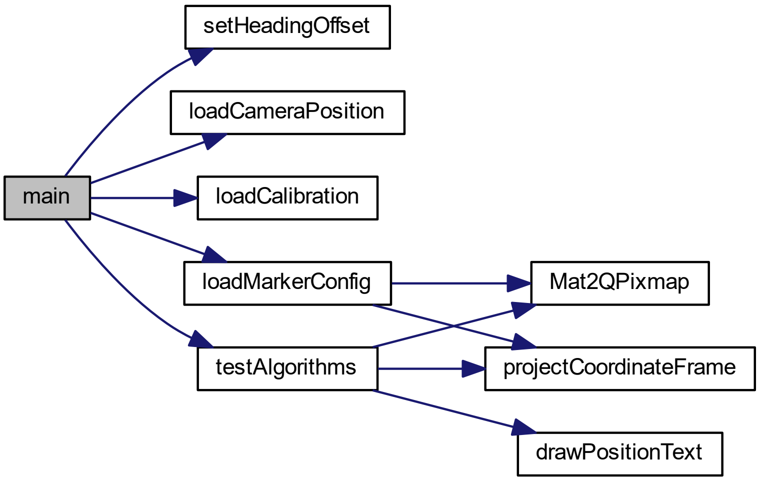
Parameters

in	*argc*	is not used.
in	*argv*	is also not used.

Definition at line 165 of file main.cpp.

```
{
    QApplication a(argc, argv);
    RigidTrack w;
    w.show();   //!< show the GUI
    //! connect the Qt slots and signals for event handling
    QObject::connect(&commObj, SIGNAL(statusChanged(QString)), &w, SLOT(setStatus(QString)),
  Qt::DirectConnection);
    QObject::connect(&commObj, SIGNAL(imageChanged(QPixmap)), &w, SLOT(setImage(QPixmap)),
  Qt::DirectConnection);
    QObject::connect(&commObj, SIGNAL(logAdded(QString)), &w, SLOT(setLog(QString)),
  Qt::DirectConnection);
    QObject::connect(&commObj, SIGNAL(logCleared()), &w, SLOT(clearLog(QString)),
  Qt::DirectConnection);
    QObject::connect(&commObj, SIGNAL(P3Penabled(bool)), &w, SLOT(enableP3P(bool)),
  Qt::DirectConnection);
    QObject::connect(&commObj, SIGNAL(progressUpdated(int)), &w, SLOT(progressUpdate(int)),
  Qt::DirectConnection);

    commObj.addLog("RigidTrack Version:");
    commObj.addLog(QString::number(_MSC_FULL_VER));
    commObj.addLog("Built on:");
    commObj.addLog(QString(__DATE__));

    //! initial guesses for position and rotation, important for Iterative Method!
    Tvec.at<double>(0) = 45;
    Tvec.at<double>(1) = 45;
    Tvec.at<double>(2) = 4500;
    Rvec.at<double>(0) = 0 * 3.141592653589 / 180.0;
    Rvec.at<double>(1) = 0 * 3.141592653589 / 180.0;
    Rvec.at<double>(2) = -45 * 3.141592653589 / 180.0;

    //! Points that make up the marker CoSy axis system, hence one line in each axis direction
    coordinateFrame = std::vector<Point3d>(4);
    coordinateFrameProjected = std::vector<Point2d>(4);
    coordinateFrame[0] = cv::Point3d(0, 0, 0);
    coordinateFrame[1] = cv::Point3d(300, 0, 0);
    coordinateFrame[2] = cv::Point3d(0, 300, 0);
    coordinateFrame[3] = cv::Point3d(0, 0, 300);

    position[0] = 1.1234;     //!< set position initial values
    position[1] = 1.2345;     //!< set position initial values
    position[2] = 1.3456;     //!< set position initial values

    velocity[0] = 0.123;    //!< set velocity initial values
    velocity[1] = 0.234;    //!< set velocity initial values
    velocity[2] = 0.345;    //!< set velocity initial values

    eulerAngles[0] = 1.002;  //!< set initial euler angles to arbitrary values for testing
    eulerAngles[1] = 1.003;  //!< set initial euler angles to arbitrary values for testing
    eulerAngles[2] = 1.004;  //!< set initial euler angles to arbitrary values for testing

    setHeadingOffset(0.0); //!< set the heading offset to 0

    ss.precision(4); //!< outputs in the log etc are limited to 3 decimal values

    loadCameraPosition(); //!< load the rotation matrix from camera CoSy to ground CoSy
    loadCalibration(0); //!< load the calibration file with the camera intrinsics
    loadMarkerConfig(0); //!< load the standard marker configuration
    testAlgorithms(); //!< test the algorithms and their accuracy

    return a.exec();
}
```

Here is the call graph for this function:



3.1.2.12 Mat2QPixmap()

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

Convert an opencv matrix that represents a picture to a Qt Pixmap object for the GUI.

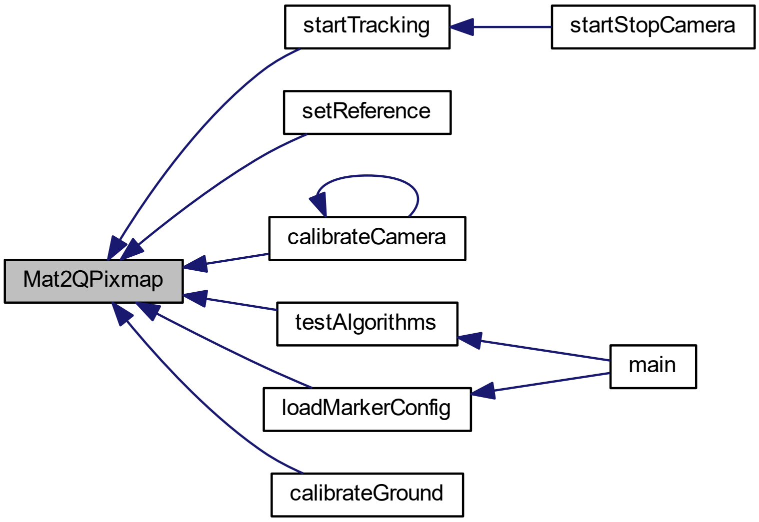
Parameters

in	*src*	is the camera image represented as OpenCV matrix.

Definition at line 225 of file main.cpp.

```
{
    QImage dest((const uchar *)src.data, src.cols, src.rows, src.step, QImage::Format_RGB888);
    dest.bits(); //! enforce deep copy, see documentation
                 //! of QImage::QImage ( const uchar * data, int width, int height, Format format )
    QPixmap pixmapDest = QPixmap::fromImage(dest);
    return pixmapDest;
}
```

Here is the caller graph for this function:



3.1.2.13 projectCoordinateFrame()

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

Project the coordinate CoSy origin and axis direction of the marker CoSy with the rotation and translation of the object for visualization.

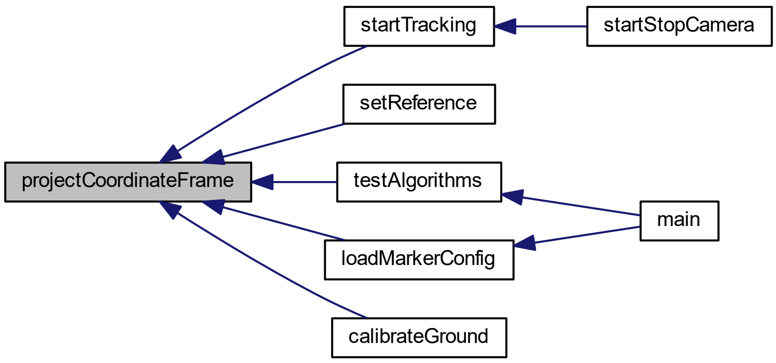
Parameters

in	*pictureFrame*	the image in which the CoSy frame should be pasted.

Definition at line 1099 of file main.cpp.

```
{
    projectPoints(coordinateFrame, Rvec, Tvec,
cameraMatrix, distCoeffs, coordinateFrameProjected);
    line(pictureFrame, coordinateFrameProjected[0],
coordinateFrameProjected[3], Scalar(0, 0, 255), 2); //!<z-axis
    line(pictureFrame, coordinateFrameProjected[0],
coordinateFrameProjected[1], Scalar(255, 0, 0), 2); //!<x-axis
    line(pictureFrame, coordinateFrameProjected[0],
coordinateFrameProjected[2], Scalar(0, 255, 0), 2); //!<y-axis
}
```

Here is the caller graph for this function:



3.1.2.14 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↩ Tracking function because its send rate is less than 100 Hz.

Definition at line 1172 of file main.cpp.

```
{
    datagram.clear();
    QDataStream out(&datagram, QIODevice::WriteOnly);
    out.setVersion(QDataStream::Qt_4_3);
    out << (float)Position[0] << (float)Position[1] << (float)Position[2];
    out << (float)Euler[0] << (float)Euler[1] << (float)Euler[2]; //! Roll Pitch Heading
    udpSocketObject->writeDatagram(datagram, IPAdressObject, portObject);

    //! if second receiver is activated send it also the tracking data
    if (safety2Enable)
    {
        udpSocketSafety2->writeDatagram(datagram, IPAdressSafety2, portSafety2);
    }

}
```

Here is the caller graph for this function:

3.1.2.15 setHeadingOffset()

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

Add a heading offset to the attitude for the case it is wanted by the user.

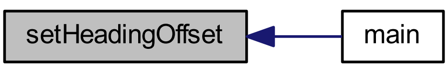
Parameters

in	*d*	denotes heading offset in degrees.

Definition at line 1140 of file main.cpp.

```
1141 {
1142     headingOffset = d;
1143     d = d * 3.141592653589 / 180.0; //! Convert heading offset from degrees to rad
1144 
1145     //! Calculate rotation about x axis
1146     Mat R_x = (Mat_<double>(3, 3) <<
1147         1, 0, 0,
1148         0, 1, 0,
1149         0, 0, 1
1150         );
1151 
1152     //! Calculate rotation about y axis
1153     Mat R_y = (Mat_<double>(3, 3) <<
1154         1, 0, 0,
1155         0, 1, 0,
1156         0, 0, 1
1157         );
1158 
1159     //! Calculate rotation about z axis
1160     Mat R_z = (Mat_<double>(3, 3) <<
1161         cos(d), -sin(d), 0,
1162         sin(d), cos(d), 0,
1163         0, 0, 1);
1164 
1165 
1166     //! Combined rotation matrix
1167     M_HeadingOffset = R_z * R_y * R_x;
1168 }
```

Here is the caller graph for this function:



3.1.2.16 setReference()

```
int setReference ( )
```

Determine the initial position of the object that serves as reference point or as ground frame origin. Computes the pose 200 times and then averages it. The position and attitude are from now on used as navigation CoSy.

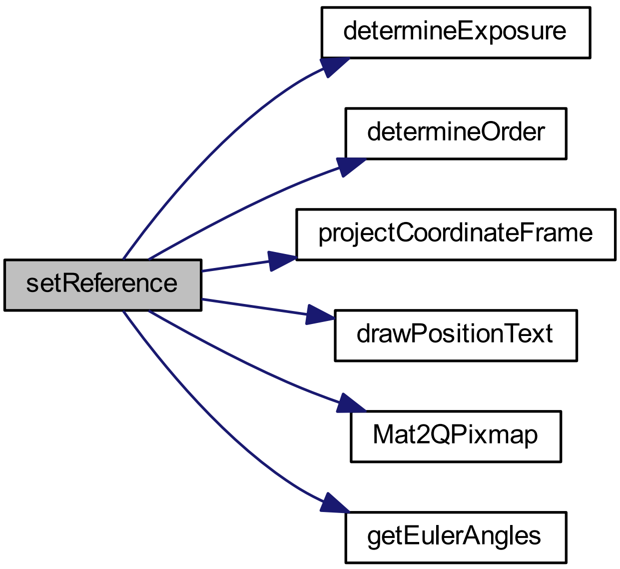
Definition at line 613 of file main.cpp.

```
{
    //! initialize the variables with starting values
    gotOrder = false;
    posRef = 0;
    eulerRef = 0;
    RmatRef = 0;
    Rvec = RvecOriginal;
    Tvec = TvecOriginal;

    determineExposure();

    ss.str("");
    commObj.addLog("Started reference coordinate determination.");

    CameraLibrary_EnableDevelopment();
    //! Initialize Camera SDK ==--
    CameraLibrary::CameraManager::X();

    //! 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 ================----
    CameraManager::X().WaitForInitialization();
    Camera *camera = CameraManager::X().GetCamera();

    //! If no device connected, pop a message box and exit ==--
    if (camera == 0)
    {
        commObj.addLog("No camera found!");
        return 1;
    }

    //! Determine camera resolution to size application window ==----
    int cameraWidth = camera->Width();
    int cameraHeight = camera->Height();
    camera->GetDistortionModel(distModel);
    cv::Mat matFrame(cv::Size(cameraWidth, cameraHeight), CV_8UC1);

    //! Set camera mode to precision mode, it directly provides marker coordinates
    camera->SetVideoType(Core::PrecisionMode);

    //! Start camera output ==--
    camera->Start();

    //! 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
    camera->SetTextOverlay(true);
    camera->SetFrameRate(intFrameRate);
    camera->SetIntensity(intIntensity);
    camera->SetIRFilter(true);
    camera->SetContinuousIR(false);
    camera->SetHighPowerMode(false);

    //! sample some frames and calculate the position and attitude. then average those values and use that
   as zero position
    int numberSamples = 0;
    int numberToSample = 200;
    double projectionError = 0; //!< difference between the marker points as seen by the camera and the
   projected marker points with Rvec and Tvec

    while (numberSamples < numberToSample)
    {
        //! Fetch a new frame from the camera ===---
        Frame *frame = camera->GetFrame();

        if (frame)
        {
            //! Ok, we've received a new frame, lets do something
            //! with it.
            if (frame->ObjectCount() == numberMarkers)
            {
                //!for(int i=0; i<frame->ObjectCount(); i++)
                for (int i = 0; i < numberMarkers; i++)
                {
                    cObject *obj = frame->Object(i);
                    list_points2dUnsorted[i] = cv::Point2d(obj->X(), obj->Y());
                }

                if (gotOrder == false)
                {
                    determineOrder();
                }

                //! sort the 2d points with the correct indices as found in the preceeding order
   determination algorithm
                for (int w = 0; w < numberMarkers; w++)
```

```
                {
                    list_points2d[w] = list_points2dUnsorted[pointOrderIndices[w]];
                }
                list_points2dOld = list_points2dUnsorted;

                //!Compute the pose from the 3D-2D corresponses
                solvePnP(list_points3d, list_points2d, cameraMatrix, distCoeffs, Rvec, Tvec, useGuess, methodPNP);

                //! project the marker 3d points with the solution into the camera image CoSy and calculate difference to true camera image
                projectPoints(list_points3d, Rvec, Tvec, cameraMatrix, distCoeffs, list_points2dProjected);
                projectionError = norm(list_points2dProjected, list_points2d);

                double maxValue = 0;
                double minValue = 0;
                minMaxLoc(Tvec.at<double>(2), &minValue, &maxValue);

                if (maxValue > 10000 || minValue < 0)
                {
                    ss.str("");
                    ss << "Negative z distance, thats not possible. Start the set zero routine again or restart Programm.";
                    commObj.addLog(QString::fromStdString(ss.str()));
                    frame->Release();
                    return 1;
                }

                if (projectionError > 5)
                {
                    commObj.addLog("Reprojection error is bigger than 5 pixel. Correct marker configuration loaded?\nMarker position measured precisely?");
                    frame->Release();
                    return 1;
                }

                if (norm(positionOld) - norm(Tvec) < 0.05)   //!<Iterative Method needs time to converge to solution
                {
                    add(posRef, Tvec, posRef);
                    add(eulerRef, Rvec, eulerRef); //!< That are not the values of yaw, roll and pitch yet! Rodriguez has to be called first.
                    numberSamples++;    //!<  one sample more :D
                    commObj.progressUpdate(numberSamples * 100 / numberToSample);
                }
                positionOld = Tvec;

                Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
                for (int i = 0; i < numberMarkers; i++)
                {
                    circle(cFrame, Point(list_points2d[i].x, list_points2d[i].y), 6, Scalar(0, 225, 0), 3);
                }
                projectCoordinateFrame(cFrame);
                projectPoints(list_points3d, Rvec, Tvec, cameraMatrix, distCoeffs, list_points2d);
                for (int i = 0; i < numberMarkers; i++)
                {
                    circle(cFrame, Point(list_points2d[i].x, list_points2d[i].y), 3, Scalar(225, 0, 0), 3);
                }
                drawPositionText(cFrame, position, eulerAngles, projectionError);

                QPixmap QPFrame;
                QPFrame = Mat2QPixmap(cFrame);
                commObj.changeImage(QPFrame);
                QCoreApplication::processEvents();

            }
            frame->Release();
        }
    }
    //! Release camera ==--
    camera->Release();

    //!Divide by the number of samples to get the mean of the reference position
    divide(posRef, numberToSample, posRef);
    divide(eulerRef, numberToSample, eulerRef); //!< eulerRef is here in Axis Angle notation

    Rodrigues(eulerRef, RmatRef);                //!< axis angle to rotation matrix
    //!-- Euler Angles, finally
```

```
    getEulerAngles(RmatRef, eulerRef); //!<  rotation matrix to euler
    ss.str("");
    ss << "RmatRef is:\n";
    ss << RmatRef << "\n";
    ss << "Reference Position is:\n";
    ss << posRef << "[mm] \n";
    ss << "Reference Euler Angles are:\n";
    ss << eulerRef << "[deg] \n";

    //! 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 setReference() function once again
    double error = norm(posRef) - norm(Tvec);
    if (error > 5.0)
    {
        ss << "Caution, distance between reference position and last position is: " << error << "\n Start the set zero routine once again.";
    }
    commObj.addLog(QString::fromStdString(ss.str()));
    commObj.progressUpdate(0);
    return 0;
}
```

Here is the call graph for this function:



3.1.2.17 startTracking()

```
int startTracking ( )
```

Start the loop that fetches frames, computes the position etc and sends it to other computers. This function is the core of this program, hence the pose estimation is done here.

Definition at line 276 of file main.cpp.

```
                    {


    gotOrder = false; //! The order of points, hence which entry in list_points3d corresponds to which in list_points2d is not calculated yet
    Rvec = RvecOriginal; //! Use the value of Rvec that was set in main() as starting value for the solvePnP algorithm
    Tvec = TvecOriginal; //! Use the value of Tvec that was set in main() as starting value for the solvePnP algorithm
    GetLocalTime(&logDate);  //! 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
    logFileName = "./logs/positionLog_" + QString::number(logDate.wDay) + "_" + QString::number(logDate.wMonth) + "_" + QString::number(logDate.wYear);
    logFileName += "_" + QString::number(logDate.wHour) + "_" + QString::number(logDate.wMinute) + "_" + QString::number(logDate.wSecond) + ".txt";
    logName = logFileName.toStdString(); //! Convert the QString to a standard string

    determineExposure(); //! 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:

    CameraLibrary_EnableDevelopment();
    CameraLibrary::CameraManager::X(); //! 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
    CameraManager::X().WaitForInitialization();
    Camera *camera = CameraManager::X().GetCamera();

    //! If no camera can be found, inform user in message log and exit function
    if (camera == 0)
    {
        commObj.addLog("No camera found!");
        return 1;
    }

    //! Determine camera resolution to size application window
    int cameraWidth = camera->Width();
    int cameraHeight = camera->Height();

    camera->SetVideoType(Core::PrecisionMode);  //! Set the camera mode to precision mode, it used greyscale information for marker property calculations

    camera->Start(); //! 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
    camera->SetTextOverlay(true);
    camera->SetExposure(intExposure);     //! Set the camera exposure
    camera->SetIntensity(intIntensity); //! Set the camera infrared LED intensity
    camera->SetFrameRate(intFrameRate); //! Set the camera framerate to 100 Hz
    camera->SetIRFilter(true);  //! Enable the filter that blocks visible light and only passes infrared light
    camera->SetHighPowerMode(true); //! Enable high power mode of the LEDs
    camera->SetContinuousIR(false); //! Disable continuous LED light
    camera->SetThreshold(intThreshold); //! Set threshold for marker detection

    //! Create a new matrix that stores the grayscale picture from the camera
    Mat matFrame = Mat::zeros(cv::Size(cameraWidth, cameraHeight), CV_8UC1);
    QPixmap QPFrame; //! QPixmap is the corresponding Qt class that saves images
    //! Matrix that stores the colored picture, hence marker points, coordinate frame and reprojected points
    Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));

    int v = 0;  //! Helper variable used to kick safety switch
    //! Variables for the min and max values that are needed for sanity checks
    double maxValue = 0;
    double minValue = 0;
    int framesDropped = 0; //! Ff a marker is not visible or accuracy is bad increase this counter
    double projectionError = 0; //! Equals the quality of the tracking

    setUpUDP(); //! Open sockets and ports for UDP communication

    if (safetyEnable) //! If the safety feature is enabled send the starting message
    {
        //! Send enable message, hence send a 9 and then a 1
        data.setNum((int)(9));
        udpSocketSafety->write(data);
        data.setNum((int)(1));
        udpSocketSafety->write(data);
```

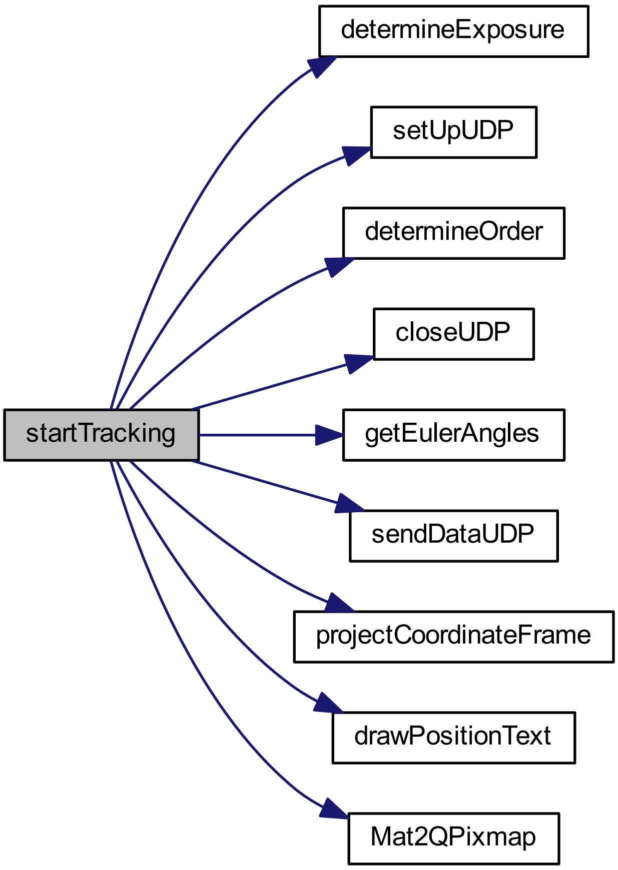
```
    }

    //! Fetch a new frame from the camera
    bool gotTime = false; //! 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 (!gotTime) //! While no new frame is received loop
    {
        Frame *frame = camera->GetFrame(); //! Get a new camera frame
        if (frame)  //! There is actually a new frame
        {
            timeFirstFrame = frame->TimeStamp(); //! Get the time stamp for the first frame. It is subtracted for the following frames
            frame->Release();   //! Release the frame so the camera can continue
            gotTime = true; //! Exit the while loop
        }
    }

    //! Now enter the main loop that processes each frame and computes the pose, sends it and logs stuff
    while (!exitRequested) //! Check if the user has not pressed "Stop Tracking" yet
    {

        Frame *frame = camera->GetFrame(); //! Fetch a new frame from the camera

        if (frame) //! Did we got a new frame or does the camera still need more time
        {
            framesDropped++; //! 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
            if (frame->ObjectCount() == numberMarkers)
            {
                //! 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
                for (int i = 0; i < numberMarkers; i++)
                {
                    cObject *obj = frame->Object(i);
                    list_points2dUnsorted[i] = cv::Point2d(obj->X(), obj->Y());
                }

                if (gotOrder == false) //! Was the order already determined? This is false for the first frame and from then on true
                {
                    determineOrder(); //! Now compute the order
                }

                //! Sort the 2d points with the correct indices as found in the preceeding order determination algorithm
                for (int w = 0; w < numberMarkers; w++)
                {
                    list_points2d[w] = list_points2dUnsorted[pointOrderIndices[w]]; //! pointOrderIndices was calculated in determineOrder()
                }

                //! 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.
                for (int j = 0; j < numberMarkers; j++)
                {
                    minPointDistance = 5000; //! The sum of point distances is set to something unrealistic large
                    for (int k = 0; k < numberMarkers; k++)
                    {
                        //! Calculate N_2 norm of unsorted points minus old points
                        currentPointDistance = norm(list_points2dUnsorted[pointOrderIndices[j]] - list_points2dOld[k]);
                        //! If the norm is smaller than minPointDistance the correspondence is more likely to be correct
                        if (currentPointDistance < minPointDistance)
                        {
                            //! Update the array that saves the new point order
                            minPointDistance = currentPointDistance;
                            pointOrderIndicesNew[j] = k;
                        }
                    }
                }
```

```

                //! Now the new order is found, set the point order to the new value
                for (int k = 0; k < numberMarkers; k++)
                {
                    pointOrderIndices[k] = pointOrderIndicesNew[k];
                    list_points2d[k] = list_points2dUnsorted[pointOrderIndices[k]];
                }

                //! Save the unsorted position of the marker points for the next loop
                list_points2dOld = list_points2dUnsorted;

                //!Compute the object pose from the 3D-2D corresponses
                solvePnP(list_points3d, list_points2d, cameraMatrix, distCoeffs, Rvec, Tvec, useGuess, methodPNP);

                //! Project the marker 3d points with the solution into the camera image CoSy and calculate difference to true camera image
                projectPoints(list_points3d, Rvec, Tvec, cameraMatrix, distCoeffs, list_points2dProjected);
                projectionError = norm(list_points2dProjected, list_points2d); //! Difference of true pose and found pose

                //! Increase the framesDropped variable if accuracy of tracking is too bad
                if (projectionError > 5)
                {
                    framesDropped++;
                }
                else
                {
                    framesDropped = 0;  //! 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
                minMaxLoc(Tvec.at<double>(2), &minValue, &maxValue);

                //! Sanity check of values. negative z means the marker CoSy is behind the camera, that's not possible.
                if (minValue < 0)
                {
                    commObj.addLog("Negative z distance, that is not possible. Start the set zero routine again or restart Program.");
                    frame->Release(); //! Release the frame so the camera can move on
                    camera->Release(); //! Release the camera
                    closeUDP(); //! Close all UDP connections so the programm can be closed later on and no resources are locked
                    return 1; //! 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: \f$ T_C^{NM} = Tvec - Tvec_{Ref} \f$
                subtract(Tvec, posRef, position);

                //! Transform the position from the camera CoSy to the navigation CoSy with INS alligned heading and convert from [mm] to [m]
                //! \f$ T_N^{NM} = M_{NC} \times T_C^{NM} \f$
                Mat V = 0.001 * M_HeadingOffset * M_CN.t() * (Mat)position;
                position = V;   //! Position is the result of the preceeding calculation
                position[2] *= invertZ;  //! Invert Z if check box in GUI is activated, hence height above ground is considered

                //! Realtive angle between reference orientation and current orientation
                Rodrigues(Rvec, Rmat);  //! Convert axis angle respresentation to ordinary rotation matrix

                //! The difference of the reference rotation and the current rotation
                //! \f$ R_{ NM } = M_{ NC } \times R_{ CM } \f$
                Rmat = RmatRef.t() *Rmat;

                //! Euler Angles, finally
                getEulerAngles(Rmat, eulerAngles); //! Get the euler angles from the rotation matrix
                eulerAngles[2] += headingOffset; //! 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
                frameTime = frame->TimeStamp() - timeOld;   //! Time between the old frame and the current frame
                timeOld = frame->TimeStamp();    //! Set the old frame time to the current  one
                velocity[0] = (position[0] - positionOld[0]) / frameTime; //! Calculate the x velocity with finite differences
                velocity[1] = (position[1] - positionOld[1]) /
```

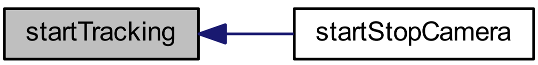
```
frameTime; //! Calculate the y velocity with finite differences
                velocity[2] = (position[2] - positionOld[2]) /
frameTime; //! Calculate the z velocity with finite differences
                positionOld = position;  //! Set the old position to the current one for
 next frame velocity calcuation

                eulerAngles[0] = eulerAngles[0] * -3.141592653589 / 180.0; //!
 Convert the Euler angles from degrees to rad
                eulerAngles[1] = eulerAngles[1] * -3.141592653589 / 180.0;
                eulerAngles[2] = eulerAngles[2] * 3.141592653589 / 180.0;

                //! Send position and Euler angles over WiFi with 100 Hz
                sendDataUDP(position, eulerAngles);

                //! Save the values in a log file, values are:
                //! Time sinc tracking started  Position    Euler Angles    Velocity
                logfile.open(logName, std::ios::app); //! Open the log file, the folder is
 RigidTrackInstallationFolder/logs
                logfile << frame->TimeStamp() - timeFirstFrame << ";" <<
position[0] << ";" << position[1] << ";" << position[2] << ";";
                logfile << eulerAngles[0] << ";" <<
eulerAngles[1] << ";" << eulerAngles[2] << ";";
                logfile << velocity[0] << ";" << velocity[1] << ";" <<
velocity[2] << "\n";
                logfile.close(); //! 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
            if (safetyEnable)
            {
                if ((abs(position[0]) < safetyBoxLength && abs(position[1]) <
safetyBoxLength && abs(position[2]) < safetyBoxLength))
                {
                    //! Absolute Euler angles must be smaller than allowed value. Heading is not considered

                    if ((abs(eulerAngles[0]) < safetyAngle && abs(eulerAngles[1]) <
safetyAngle))
                    {
                        //! Send the OKAY signal to the desired computer every 5th time
                        if (v == 5) {
                            data.setNum((int)(1));
                            udpSocketSafety->write(data); //! Send the 1
                            v = 0; //! 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)
                    else
                    {
                        data.setNum((int)(0));  //! Send the shutdown signal, a 0
                        udpSocketSafety->write(data);
                        commObj.addLog("Object exceeded allowed Euler angles, shutdown signal sent."
); //! Inform the user

                    }
                }
                //! The position of the object exceeded the allowed position, shut the object down
                else
                {
                    data.setNum((int)(0));  //! Send the shutdown signal, a 0
                    udpSocketSafety->write(data);
                    commObj.addLog("Object left allowed area, shutdown signal sent."); //! Inform
 the user

                }
            }

            //! Inform the user if tracking system is disturbed (marker lost or so) or error was too big
            if (framesDropped > 10)
            {
                if (safetyEnable) //! Also send the shutdown signal
                {
                    data.setNum((int)(0));  //! Send the shutdown signal, a 0
                    udpSocketSafety->write(data);
                }
                commObj.addLog("Lost marker points or precision was bad!"); //! Inform the user
                framesDropped = 0;
            }

            //! Rasterize the frame so it can be shown in the GUI
            frame->Rasterize(cameraWidth, cameraHeight, matFrame.step,
BACKBUFFER_BITSPERPIXEL, matFrame.data);

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

```
            cvtColor(matFrame, cFrame, COLOR_GRAY2RGB);

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

            //! Project the marker points from 3D to the camera image frame (2d) with the computed pose
            projectPoints(list_points3d, Rvec, Tvec,
    cameraMatrix, distCoeffs, list_points2d);
            for (int i = 0; i < numberMarkers; i++)
            {
                //! 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
                circle(cFrame, Point(list_points2d[i].x,
    list_points2d[i].y), 3, Scalar(225, 0, 0), 3);
            }

            //! Write the current position, attitude and error values as text in the frame
            drawPositionText(cFrame, position, eulerAngles, projectionError);

            //! Send the new camera picture to the GUI and call the GUI processing routine
            QPixmap QPFrame;
            QPFrame = Mat2QPixmap(cFrame);
            commObj.changeImage(QPFrame); //! Update the picture in the GUI
            QCoreApplication::processEvents(); //! Give Qt time to handle everything

            //! Release the camera frame to fetch the new one
            frame->Release();
        }
    }

    //! User choose to stop the tracking, clean things up
    closeUDP(); //! Close the UDP connections so resources are deallocated
    camera->Release();  //! Release camera
    return 0;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.2.18 testAlgorithms()

```
void testAlgorithms ( )
```

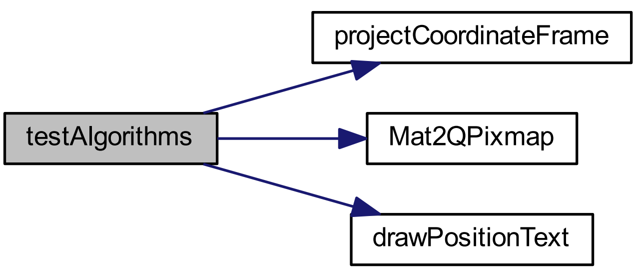
Project some points from 3D to 2D and then check the accuracy of the algorithms. Mainly to generate something that can be shown in the camera view so the user knows everything loaded correctly.

Definition at line 970 of file main.cpp.

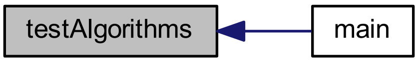
```
{

  int _methodPNP;

  std::vector<Point2d> noise(numberMarkers);

  RvecOriginal = Rvec;
  TvecOriginal = Tvec;

  projectPoints(list_points3d, Rvec, Tvec, cameraMatrix, distCoeffs, list_points2dProjected);

  ss.str("");
  ss << "Unsorted Points 2D Projected \n";
  ss << list_points2dProjected << "\n";
  commObj.addLog(QString::fromStdString(ss.str()));

  Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
  for (int i = 0; i < numberMarkers; i++)
  {
      circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 6, Scalar(0, 255, 0), 3);
  }

  projectCoordinateFrame(cFrame);

  ss.str("");
  ss << "=====================================================\n";
  ss << "================= Projected Points =================\n";
  ss << list_points2dProjected << "\n";

  randn(noise, 0, 0.5);
  add(list_points2dProjected, noise, list_points2dProjected);

  ss << "================ With Noise Points =================\n";
  ss << list_points2dProjected << "\n";
  commObj.addLog(QString::fromStdString(ss.str()));


  bool useGuess = true;
  _methodPNP = 0; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP  //!< not used

  solvePnP(list_points3d, list_points2dProjected, cameraMatrix, distCoeffs, Rvec, Tvec, useGuess, _methodPNP);

  ss.str("");
  ss << "=====================================================\n";
  ss << "==================== Iterative =====================\n";
  ss << "rvec: " << "\n";
  ss << Rvec << "\n";
  ss << "tvec: " << "\n";
  ss << Tvec << "\n";

  commObj.addLog(QString::fromStdString(ss.str()));

  _methodPNP = 1; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP UPnP not used
  Rvec = cv::Mat::zeros(3, 1, CV_64F);
  Tvec = cv::Mat::zeros(3, 1, CV_64F);
  solvePnP(list_points3d, list_points2dProjected, cameraMatrix, distCoeffs, Rvec, Tvec, useGuess, _methodPNP);

  ss.str("");
  ss << "=====================================================\n";
  ss << "====================    EPNP    ====================\n";
  ss << "rvec: " << "\n";
  ss << Rvec << "\n";
  ss << "tvec: " << "\n";
  ss << Tvec << "\n";

  projectPoints(list_points3d, Rvec, Tvec, cameraMatrix, distCoeffs, list_points2dProjected);
  for (int i = 0; i < numberMarkers; i++)
  {
      circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 3, Scalar(255, 0, 0), 3);
  }
  QPixmap QPFrame;
  QPFrame = Mat2QPixmap(cFrame);
  commObj.changeImage(QPFrame);
```

```
    QCoreApplication::processEvents();
    commObj.addLog(QString::fromStdString(ss.str()));
    if (numberMarkers == 4)
    {
        _methodPNP = 2; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP  //!< not used
        Rvec = cv::Mat::zeros(3, 1, CV_64F);
        Tvec = cv::Mat::zeros(3, 1, CV_64F);
        solvePnP(list_points3d, list_points2dProjected, cameraMatrix, distCoeffs, Rvec, Tvec, useGuess, _methodPNP);

        ss.str("");
        ss << "====================================================\n";
        ss << "====================     P3P    =====================\n";
        ss << "rvec: " << "\n";
        ss << Rvec << "\n";
        ss << "tvec: " << "\n";
        ss << Tvec << "\n";

        projectPoints(list_points3d, Rvec, Tvec, cameraMatrix, distCoeffs, list_points2dProjected);
        for (int i = 0; i < numberMarkers; i++)
        {
            circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 3, Scalar(255, 0, 0), 3);
        }
        double projectionError = norm(list_points2dProjected, list_points2d);
        putText(cFrame, "Testing Algorithms Finished", cv::Point(5, 420), 1, 1, cv::Scalar(255, 255, 255));
        drawPositionText(cFrame, position, eulerAngles, projectionError);

        QPixmap QPFrame;
        QPFrame = Mat2QPixmap(cFrame);
        commObj.changeImage(QPFrame);
        QCoreApplication::processEvents();
        commObj.addLog(QString::fromStdString(ss.str()));
    }

    _methodPNP = 4; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP  //!< not used
    Rvec = cv::Mat::zeros(3, 1, CV_64F);
    Tvec = cv::Mat::zeros(3, 1, CV_64F);
    solvePnP(list_points3d, list_points2dProjected, cameraMatrix, distCoeffs, Rvec, Tvec, useGuess, _methodPNP);

    ss.str("");
    ss << "====================================================\n";
    ss << "====================    UPNP    =====================\n";
    ss << "rvec: " << "\n";
    ss << Rvec << "\n";
    ss << "tvec: " << "\n";
    ss << Tvec << "\n";

    commObj.addLog(QString::fromStdString(ss.str()));

    Rvec = RvecOriginal;
    Tvec = TvecOriginal;

}
```

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.3 Variable Documentation

3.1.3.1 commObj

```
commObject commObj
```

class that handles the communication from main.cpp to the GUI

Now declare variables that are used across the main.cpp file. Basically almost every variable used is declared here.

Definition at line 68 of file main.cpp.

3.1.3.2 Rmat

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

Rotation, translation etc. matrix for PnP results.

rotation matrix from camera CoSy to marker CoSy

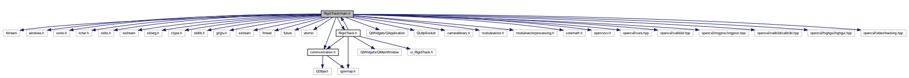
Definition at line 95 of file main.cpp.

3.2 RigidTrack/main.h File Reference

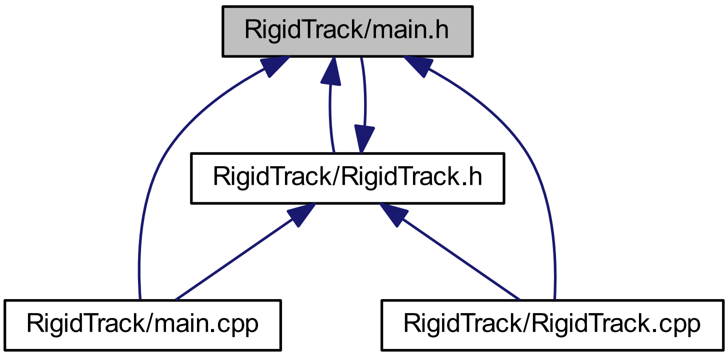
Header file for main.cpp.

```
#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 startTracking ()
- void startStopCamera ()

  Start or stop the tracking depending on if the camera is currently running or not.
- int setReference ()
- int calibrateCamera ()

  Start the camera calibration routine that computes the camera matrix and distortion coefficients.
- void loadCalibration (int method)
- void testAlgorithms ()
- void projectCoordinateFrame (Mat pictureFrame)
- void setUpUDP ()

  Open the UDP ports for communication.
- void setHeadingOffset (double d)
- void sendDataUDP (cv::Vec3d &Position, cv::Vec3d &Euler)
- void closeUDP ()
- void loadMarkerConfig (int method)
- void drawPositionText (cv::Mat &Picture, cv::Vec3d &Position, cv::Vec3d &Euler, double error)
- void loadCameraPosition ()
- int determineExposure ()
- void determineOrder ()
- int calibrateGround ()

Variables

- int methodPNP

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

  is the safety feature enabled
- bool safety2Enable

is the second receiver enabled
- double safetyBoxLength

  length of the safety area cube in meters
- int safetyAngle

  bank and pitch angle protection in degrees
- QHostAddress IPAdressObject

  IPv4 adress of receiver 1.
- QHostAddress IPAdressSafety

  IPv4 adress of safety receiver.
- QHostAddress IPAdressSafety2

  IPv4 adress of receiver 2.
- int portObject

  Port of receiver 1.
- int portSafety

  Port of the safety receiver.
- int portSafety2

  Port of receiver 2.
- int invertZ

  dummy variable to invert Z direction on request
- commObject commObj

  class that handles the communication from main.cpp to the GUI

3.2.1 Detailed Description

Header file for main.cpp.

Author

Florian J.T. Wachter

Version

1.0

Date

April, 8th 2017

3.2.2 Function Documentation

3.2.2.1 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 setReference.

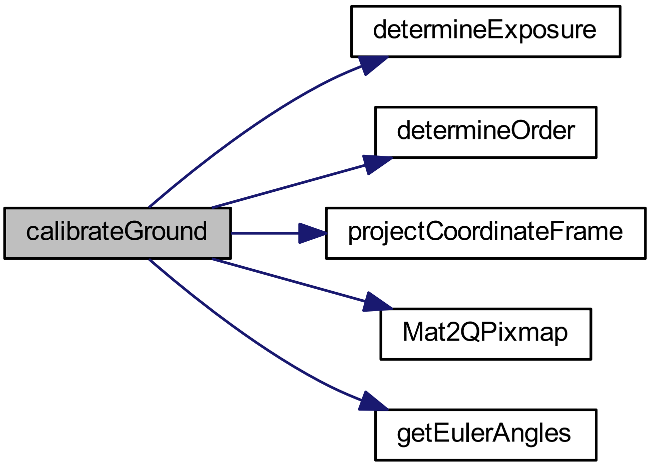
Definition at line 1581 of file main.cpp.

```
{
    //! initialize the variables with starting values
    gotOrder = false;
    posRef = 0;
    eulerRef = 0;
    RmatRef = 0;
    Rvec = RvecOriginal;
    Tvec = TvecOriginal;

    determineExposure();

    ss.str("");
    commObj.addLog("Started ground calibration");

    CameraLibrary_EnableDevelopment();
    //! Initialize Camera SDK ==--
    CameraLibrary::CameraManager::X();

    //! 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 ================----
    CameraManager::X().WaitForInitialization();
    Camera *camera = CameraManager::X().GetCamera();

    //! If no device connected, pop a message box and exit ==--
    if (camera == 0)
    {
        commObj.addLog("No camera found!");
        return 1;
    }

    //! Determine camera resolution to size application window ==----
    int cameraWidth = camera->Width();
    int cameraHeight = camera->Height();
    camera->GetDistortionModel(distModel);
    cv::Mat matFrame(cv::Size(cameraWidth, cameraHeight), CV_8UC1);

    //! Set camera mode to precision mode, it directly provides marker coordinates
    camera->SetVideoType(Core::PrecisionMode);

    //! Start camera output ==--
    camera->Start();

    //! 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
    camera->SetTextOverlay(true);
    camera->SetFrameRate(intFrameRate);
    camera->SetIntensity(intIntensity);
    camera->SetIRFilter(true);
    camera->SetContinuousIR(false);
    camera->SetHighPowerMode(false);

    //! sample some frames and calculate the position and attitude. then average those values and use that as zero position
    int numberSamples = 0;
    int numberToSample = 200;
    double projectionError = 0;

    while (numberSamples < numberToSample)
    {
        //! Fetch a new frame from the camera ===---
        Frame *frame = camera->GetFrame();

        if (frame)
        {
            //! Ok, we've received a new frame, lets do something
            //! with it.
            if (frame->ObjectCount() == numberMarkers)
            {
```

```
                //!for(int i=0; i<frame->ObjectCount(); i++)
                for (int i = 0; i < numberMarkers; i++)
                {
                    cObject *obj = frame->Object(i);
                    list_points2dUnsorted[i] = cv::Point2d(obj->X(), obj->Y());
                }

                if (gotOrder == false)
                {
                    determineOrder();
                }

                //! sort the 2d points with the correct indices as found in the preceeding order determination algorithm
                for (int w = 0; w < numberMarkers; w++)
                {
                    list_points2d[w] = list_points2dUnsorted[pointOrderIndices[w]];
                }
                list_points2dOld = list_points2dUnsorted;

                //!Compute the pose from the 3D-2D corresponses
                solvePnP(list_points3d, list_points2d, cameraMatrix, distCoeffs, Rvec, Tvec, useGuess, methodPNP);

                //! project the marker 3d points with the solution into the camera image CoSy and calculate difference to true camera image
                projectPoints(list_points3d, Rvec, Tvec, cameraMatrix, distCoeffs, list_points2dProjected);
                projectionError = norm(list_points2dProjected, list_points2d);

                if (projectionError > 3)
                {
                    commObj.addLog("Reprojection error is bigger than 3 pixel. Correct marker configuration loaded?\nMarker position measured precisely?");
                    frame->Release();
                    return 1;
                }

                double maxValue = 0;
                double minValue = 0;
                minMaxLoc(Tvec.at<double>(2), &minValue, &maxValue);

                if (maxValue > 10000 || minValue < 0)
                {


                    commObj.addLog("Negative z distance, thats not possible. Start the set zero routine again and check marker configurations.");
                    frame->Release();
                    return 1;
                }

                if (norm(positionOld) - norm(Tvec) < 0.05)   //!<Iterative Method needs time to converge to solution
                {
                    add(posRef, Tvec, posRef);
                    add(eulerRef, Rvec, eulerRef); //!< That are not the values of yaw, roll and pitch yet! Rodriguez has to be called first.
                    numberSamples++;    //!<-- one sample more :D
                    commObj.progressUpdate(numberSamples * 100 / numberToSample);
                }
                positionOld = Tvec;

                Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
                for (int i = 0; i < numberMarkers; i++)
                {
                    circle(cFrame, Point(list_points2d[i].x, list_points2d[i].y), 6, Scalar(0, 225, 0), 3);
                }
                projectCoordinateFrame(cFrame);
                projectPoints(list_points3d, Rvec, Tvec, cameraMatrix, distCoeffs, list_points2d);
                for (int i = 0; i < numberMarkers; i++)
                {
                    circle(cFrame, Point(list_points2d[i].x, list_points2d[i].y), 3, Scalar(225, 0, 0), 3);
                }

                QPixmap QPFrame;
                QPFrame = Mat2QPixmap(cFrame);
                commObj.changeImage(QPFrame);
                QCoreApplication::processEvents();

            }
```

```
                frame->Release();
            }
        }
        //! Release camera ==--
        camera->Release();

        //!Divide by the number of samples to get the mean of the reference position
        divide(posRef, numberToSample, posRef);
        divide(eulerRef, numberToSample, eulerRef); //!< eulerRef is here in Axis Angle notation

        Rodrigues(eulerRef, RmatRef);                   //!< axis angle to rotation matrix

        getEulerAngles(RmatRef, eulerRef); //!<  rotation matrix to euler
        ss.str("");
        ss << "RmatRef is:\n";
        ss << RmatRef << "\n";
        ss << "Reference Position is:\n";
        ss << posRef << "[mm] \n";
        ss << "Reference Euler angles are:\n";
        ss << eulerRef << "[deg] \n";

        //! Save the obtained calibration coefficients in a file for later use
        QString fileName = QFileDialog::getSaveFileName(nullptr, "Save ground calibration file", "referenceData.xml", "Calibration File (*.xml);;All Files (*)");
        FileStorage fs(fileName.toUtf8().constData(), FileStorage::WRITE);
        fs << "M_NC" << RmatRef;
        fs << "eulerRef" << eulerRef;
        strBuf = fs.releaseAndGetString();
        commObj.changeStatus(QString::fromStdString(strBuf));
        commObj.addLog("Saved ground calibration!");
        commObj.progressUpdate(0);
        return 0;
}
```

Here is the call graph for this function:



3.2.2.2 closeUDP()

```
void closeUDP ( )
```

Close the UDP ports again to release network interfaces etc. If this is not done the network resources are still occupied and the program can't exit properly.

Definition at line 1191 of file main.cpp.

```
{
    //! check if the socket is open and if yes close it
    if (udpSocketObject->isOpen())
    {
        udpSocketObject->close();
    }

    if (udpSocketSafety->isOpen())
    {
        udpSocketSafety->close();
    }

    if (udpSocketSafety2->isOpen())
    {
        udpSocketSafety2->close();
    }
    commObj.addLog("Closed all UDP ports.");
}
```

Here is the caller graph for this function:

3.2.2.3 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. Then the mean of those two values is used as exposure.

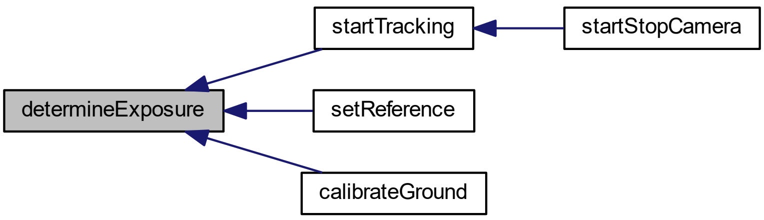
Definition at line 1380 of file main.cpp.

```
{
    //! 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:

    CameraLibrary_EnableDevelopment();

    //! Initialize Camera SDK ==--
    CameraLibrary::CameraManager::X();

    //! 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 =================----
    CameraManager::X().WaitForInitialization();
    Camera *camera = CameraManager::X().GetCamera();

    //! If no device connected, pop a message box and exit ==--
    if (camera == 0)
```

```
    {
        commObj.addLog("No camera found!");
        return 1;
    }

    //! Determine camera resolution to size application window ==----
    int cameraWidth = camera->Width();
    int cameraHeight = camera->Height();

    camera->SetVideoType(Core::PrecisionMode);  //! set the camera mode to precision mode, it used greyscale information for marker property calculations

                                            //! Start camera output ==--
    camera->Start();

    //! Turn on some overlay text so it's clear things are     ===---
    //! working even if there is nothing in the camera's view. ===---
    camera->SetTextOverlay(true);
    camera->SetExposure(intExposure);     //! set the camera exposure
    camera->SetIntensity(intIntensity); //! set the camera infrared LED intensity
    camera->SetFrameRate(intFrameRate); //! set the camera framerate to 100 Hz
    camera->SetIRFilter(true);  //! enable the filter that blocks visible light and only passes infrared light
    camera->SetHighPowerMode(true); //! enable high power mode of the leds
    camera->SetContinuousIR(false); //! enable continuous LED light
    camera->SetThreshold(intThreshold); //! set threshold for marker detection

    //!set exposure such that num markers are visible
    int numberObjects = 0;  //! Number of objects (markers) found in the current picture with the given exposure
    int minExposure = 1;    //! exposure when objects detected the first time is numberMarkers
    int maxExposure = 480;  //! exposure when objects detected is first time numberMarkers+1
    intExposure = minExposure;   //! set the exposure to the smallest value possible
    int numberTries = 0;    //! 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
    camera->SetExposure(intExposure);
    while (numberObjects != numberMarkers && numberTries < 48)
    {
        //! get a new camera frame
        Frame *frame = camera->GetFrame();
        if (frame) //! frame received
        {
            numberObjects = frame->ObjectCount();   //! how many objects are detected in the image
            if (numberObjects == numberMarkers) { minExposure = intExposure; frame->Release(); break; } //! 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
            numberTries++;
            intExposure += 10;
            camera->SetExposure(intExposure);
            ss.str("");
            ss << "Exposure: " << intExposure << "\t";
            ss << "Objects found: " << numberObjects;
            commObj.addLog(QString::fromStdString(ss.str()));
            frame->Release();
        }
    }

    //! Now determine maximum exposure, hence when are numberMarkers+1 objects detected
    numberTries = 0;    //! if the markers arent found after numberTries then there might be no markers at all in the real world
    intExposure = maxExposure;
    camera->SetExposure(intExposure);
    numberObjects = 0;
    while (numberObjects != numberMarkers && numberTries < 48)
    {
        Frame *frame = camera->GetFrame();
        if (frame)
        {
            numberObjects = frame->ObjectCount(); //! how many objects are detected in the image
            if (numberObjects == numberMarkers) { maxExposure = intExposure; frame->Release(); break; } //! 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
            intExposure -= 10;
            numberTries++;
            camera->SetExposure(intExposure);
            ss.str("");
            ss << "Exposure: " << intExposure << "\t";
            ss << "Objects found: " << numberObjects;
            commObj.addLog(QString::fromStdString(ss.str()));
            frame->Release();
        }
    }
```

```

    //! set the exposure to the mean of min and max exposure determined
    camera->SetExposure((minExposure + maxExposure) / 2.0);

    //! and now check if the correct amount of markers is detected with that new value
    while (1)
    {
        Frame *frame = camera->GetFrame();
        if (frame)
        {
            numberObjects = frame->ObjectCount(); //! how many objects are detected in the image
            if (numberObjects != numberMarkers) //! are all markers and not more or less detected in the image
            {
                frame->Release();
                commObj.addLog("Was not able to detect the right amount of markers.");
                //! Release camera ==--
                camera->Release();
                return 1;
            }
            else  //! all markers and not more or less are found
            {
                frame->Release();
                intExposure = (minExposure + maxExposure) / 2.0;
                commObj.addLog("Found the correct number of markers.");
                commObj.addLog("Exposure set to:");
                commObj.addLog(QString::number(intExposure));
                break;
            }
        }
    }

    camera->Release();
    return 0;

}
```

Here is the caller graph for this function:



3.2.2.4 determineOrder()

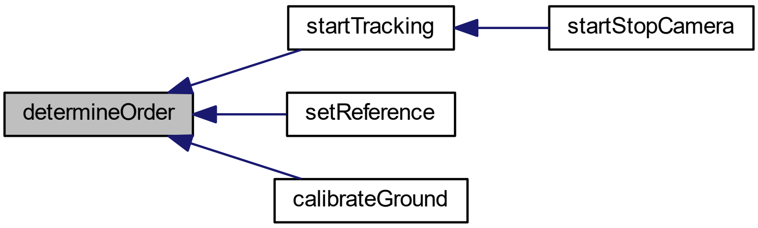
```
void determineOrder ( )
```

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.

Definition at line 1516 of file main.cpp.

```
{
    //! 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
    minPointDistance = 5000;
    std::sort(pointOrderIndices, pointOrderIndices + 4);

    //! now try every possible permutation of correspondence
    do {
        //! reset the starting values for solvePnP
        Rvec = RvecOriginal;
        Tvec = TvecOriginal;

        //! sort the 2d points with the current permutation
        for (int m = 0; m < numberMarkers; m++)
        {
            list_points2d[m] = list_points2dUnsorted[pointOrderIndices[m]];
        }

        //! Call solve PNP with P3P since its more robust and sufficient for start value determination
        solvePnP(list_points3d, list_points2d, cameraMatrix, distCoeffs, Rvec, Tvec, useGuess, SOLVEPNP_P3P);

        //! set the current difference of all point correspondences to zero
        currentPointDistance = 0;

        //! project the 3D points with the solvePnP solution onto 2D
        projectPoints(list_points3d, Rvec, Tvec, cameraMatrix, distCoeffs, list_points2dProjected);

        //! now compute the absolute difference (error)
        for (int n = 0; n < numberMarkers; n++)
        {
            currentPointDistance += norm(list_points2d[n] - list_points2dProjected[n]);
        }

        //! if the difference with the current permutation is smaller than the smallest value till now
        //! it is probably the more correct permutation
        if (currentPointDistance < minPointDistance)
        {
            minPointDistance = currentPointDistance;    //!< set the smallest value of difference to the current one
            for (int b = 0; b < numberMarkers; b++)    //!< now safe the better permutation
            {
                pointOrderIndicesNew[b] = pointOrderIndices[b];
            }
        }


    }
    //! try every permutation
    while (std::next_permutation(pointOrderIndices, pointOrderIndices + 4));

    //! now that the correct order is found assign it to the indices array
    for (int w = 0; w < numberMarkers; w++)
    {
        pointOrderIndices[w] = pointOrderIndicesNew[w];
    }
    gotOrder = true;
}
```

Here is the caller graph for this function:



3.2.2.5 drawPositionText()

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

Draw the position, attitude and reprojection error in the picture.

Parameters

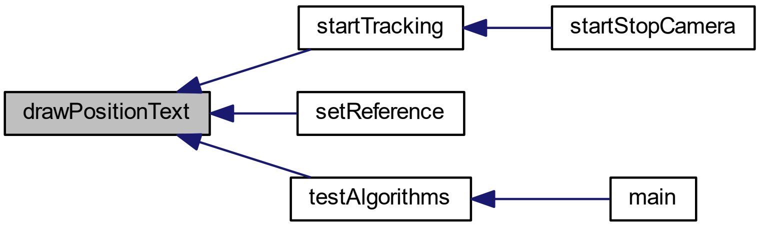
in	*Picture*	is the camera image in OpenCV matrix format.
in	*Position*	is the position of the tracked object in navigation CoSy.
in	*Euler*	are the Euler angles with respect to the navigation frame.
in	*error*	is the reprojection error of the pose estimation.

Definition at line 1333 of file main.cpp.

```
{
    ss.str("");
    ss << "X: " << Position[0] << " m";
    putText(Picture, ss.str(), cv::Point(200, 440), 1, 1, cv::Scalar(255, 255, 255));

    ss.str("");
    ss << "Y: " << Position[1] << " m";
    putText(Picture, ss.str(), cv::Point(200, 455), 1, 1, cv::Scalar(255, 255, 255));

    ss.str("");
    ss << "Z: " << Position[2] << " m";
    putText(Picture, ss.str(), cv::Point(200, 470), 1, 1, cv::Scalar(255, 255, 255));

    ss.str("");
    ss << "Heading: " << Euler[2]*180/3.1415 << " deg";
    putText(Picture, ss.str(), cv::Point(350, 440), 1, 1, cv::Scalar(255, 255, 255));

    ss.str("");
    ss << "Pitch: " << Euler[1] * 180 / 3.1415 << " deg";
    putText(Picture, ss.str(), cv::Point(350, 455), 1, 1, cv::Scalar(255, 255, 255));

    ss.str("");
```

```
    ss << "Roll: " << Euler[0] * 180 / 3.1415 << " deg";
    putText(Picture, ss.str(), cv::Point(350, 470), 1, 1, cv::Scalar(255, 255, 255));

    ss.str("");
    ss << "Error: " << error << " px";
    putText(Picture, ss.str(), cv::Point(10, 470), 1, 1, cv::Scalar(255, 255, 255));
}
```

Here is the caller graph for this function:



3.2.2.6 loadCalibration()

```
void loadCalibration (
              int method )
```

Load a previously saved camera calibration from a file.

Parameters

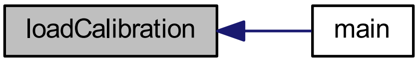
in	*method*	whether or not load the camera calibration from calibration.xml. If ==0 then yes, if != 0 then let the user select a different file.

Definition at line 941 of file main.cpp.

```
941                                  {
942 
943     QString fileName;
944     if (method == 0)
945     {
946         fileName = "calibration.xml";
947     }
948     else
949     {
950         fileName = QFileDialog::getOpenFileName(nullptr, "Choose a previous saved calibration file", "", "
      Calibration Files (*.xml);;All Files (*)");
951         if (fileName.length() == 0)
952         {
953             fileName = "calibration.xml";
954         }
955     }
956     FileStorage fs;
957     fs.open(fileName.toUtf8().constData(), FileStorage::READ);
958     fs["CameraMatrix"] >> cameraMatrix;
959     fs["DistCoeff"] >> distCoeffs;
960     commObj.addLog("Loaded calibration from file:");
```

```
    commObj.addLog(fileName);
    ss.str("");
    ss << "\nCamera Matrix is" << "\n" << cameraMatrix << "\n";
    ss << "\nDistortion Coefficients are" << "\n" << distCoeffs << "\n";
    commObj.addLog(QString::fromStdString(ss.str()));
}
```

Here is the caller graph for this function:



3.2.2.7 loadCameraPosition()

`void loadCameraPosition ( )`

Load the rotation matrix from camera CoSy to ground CoSy It is determined during calibrateGround() and stays the same once the camera is mounted and fixed.

Definition at line 1366 of file main.cpp.

```
{
    //! Open the referenceData.xml that contains the rotation from camera CoSy to ground CoSy
    FileStorage fs;
    fs.open("referenceData.xml", FileStorage::READ);
    fs["M_NC"] >> M_CN;
    fs["M_NC"] >> RmatRef;
    fs["posRef"] >> posRef;
    fs["eulerRef"] >> eulerRef;
    commObj.addLog("Loaded reference pose.");
}
```

Here is the caller graph for this function:

3.2.2.8 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.

Parameters

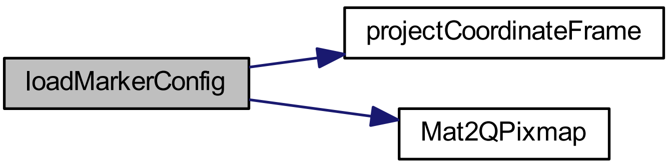
in	*method*	whether or not load the configuration from the markerStandard.xml. If ==0 load it, if != 0 let the user select a different file.

Definition at line 1213 of file main.cpp.

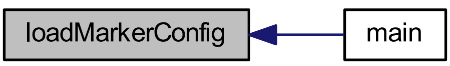
```
{
    QString fileName;
    //! during start up of the programm load the standard marker configuration
    if (method == 0)
    {
        //! open the standard marker configuration file
        FileStorage fs;
        fs.open("markerStandard.xml", FileStorage::READ);

        //! copy the values to the respective variables
        fs["numberMarkers"] >> numberMarkers;

        //! inizialise vectors with correct length depending on the number of markers
        list_points3d = std::vector<Point3d>(numberMarkers);
        list_points2d = std::vector<Point2d>(numberMarkers);
        list_points2dOld = std::vector<Point2d>(numberMarkers);
        list_points2dDifference = std::vector<double>(numberMarkers);
        list_points2dProjected = std::vector<Point2d>(numberMarkers);
        list_points2dUnsorted = std::vector<Point2d>(numberMarkers);

        //! save the marker locations in the points3d vector
        fs["list_points3d"] >> list_points3d;
        fs.release();
        commObj.addLog("Loaded marker configuration from file:");
        commObj.addLog(fileName);



    }
    else
    {
        //! if the load marker configuration button was clicked show a open file dialog
        fileName = QFileDialog::getOpenFileName(nullptr, "Choose a previous saved marker configuration file", "", "marker configuratio files (*.xml);;All Files (*)");

        //! was cancel or abort clicked
        if (fileName.length() == 0)
        {
            //! if yes load the standard marker configuration
            fileName = "markerStandard.xml";
        }

        //! open the selected marker configuration file
        FileStorage fs;
        fs.open(fileName.toUtf8().constData(), FileStorage::READ);

        //! copy the values to the respective variables
        fs["numberMarkers"] >> numberMarkers;

        //! inizialise vectors with correct length depending on the number of markers
        list_points3d = std::vector<Point3d>(numberMarkers);
        list_points2d = std::vector<Point2d>(numberMarkers);
        list_points2dOld = std::vector<Point2d>(numberMarkers);
        list_points2dDifference = std::vector<double>(numberMarkers);
        list_points2dProjected = std::vector<Point2d>(numberMarkers);
        list_points2dUnsorted = std::vector<Point2d>(numberMarkers);

        //! save the marker locations in the points3d vector
        fs["list_points3d"] >> list_points3d;
        fs.release();
        commObj.addLog("Loaded marker configuration from file:");
        commObj.addLog(fileName);

    }

    //! Print out the number of markers and their position to the GUI
    ss.str("");
    ss << "Number of Markers: " << numberMarkers << "\n";
    ss << "Marker 3D Points X,Y and Z [mm]: \n";
    for (int i = 0; i < numberMarkers; i++)
    {
```

```
        ss << "Marker " << i + 1 << ":\t" << list_points3d[i].x << "\t" << list_points3d[i].y << "\t" << list_points3d[i].z << "\n";
    }
    commObj.addLog(QString::fromStdString(ss.str()));

    //! check if P3P algorithm can be enabled, it needs exactly 4 marker points to work
    if (numberMarkers == 4)
    {
        //! if P3P is possible, let the user choose which algorithm he wants but keep iterative active
        methodPNP = 0;
        commObj.enableP3P(true);
    }
    else
    {
        //! More (or less) marker than 4 loaded, P3P is not possible, hence user cant select P3P in GUI
        methodPNP = 0;
        commObj.enableP3P(false);
        commObj.addLog("P3P algorithm disabled, only works with 4 markers.");
    }

    //! now display the marker configuration in the camera view
    Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));

    //! Set the camera pose parallel to the marker coordinate system
    Tvec.at<double>(0) = 0;
    Tvec.at<double>(1) = 0;
    Tvec.at<double>(2) = 4500;
    Rvec.at<double>(0) = 0 * 3.141592653589 / 180.0;
    Rvec.at<double>(1) = 0 * 3.141592653589 / 180.0;
    Rvec.at<double>(2) = -90. * 3.141592653589 / 180.0;

    projectPoints(list_points3d, Rvec, Tvec, cameraMatrix, distCoeffs, list_points2dProjected);
    for (int i = 0; i < numberMarkers; i++)
    {
        circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 3, Scalar(255, 0, 0), 3);
    }

    projectCoordinateFrame(cFrame);
    QPixmap QPFrame;
    QPFrame = Mat2QPixmap(cFrame);
    commObj.changeImage(QPFrame);
    QCoreApplication::processEvents();

}
```

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.2.9 projectCoordinateFrame()

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

Project the coordinate CoSy origin and axis direction of the marker CoSy with the rotation and translation of the object for visualization.

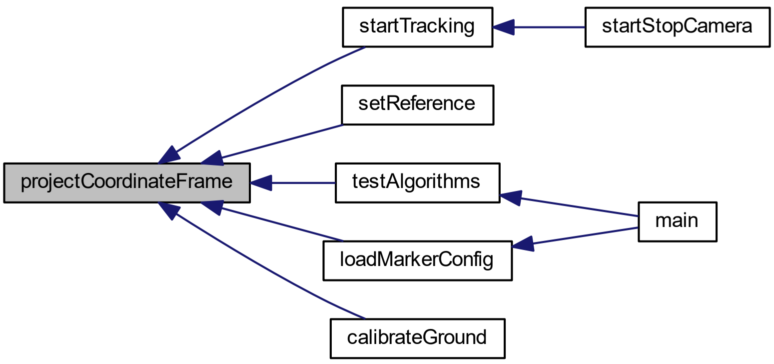
Parameters

in	*pictureFrame*	the image in which the CoSy frame should be pasted.

Definition at line 1099 of file main.cpp.

```
{
    projectPoints(coordinateFrame, Rvec, Tvec,
cameraMatrix, distCoeffs, coordinateFrameProjected);
    line(pictureFrame, coordinateFrameProjected[0],
coordinateFrameProjected[3], Scalar(0, 0, 255), 2); //!<z-axis
    line(pictureFrame, coordinateFrameProjected[0],
coordinateFrameProjected[1], Scalar(255, 0, 0), 2); //!<x-axis
    line(pictureFrame, coordinateFrameProjected[0],
coordinateFrameProjected[2], Scalar(0, 255, 0), 2); //!<y-axis
}
```

Here is the caller graph for this function:



3.2.2.10 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←↩ Tracking function because its send rate is less than 100 Hz.

Definition at line 1172 of file main.cpp.

```
{
    datagram.clear();
    QDataStream out(&datagram, QIODevice::WriteOnly);
    out.setVersion(QDataStream::Qt_4_3);
    out << (float)Position[0] << (float)Position[1] << (float)Position[2];
    out << (float)Euler[0] << (float)Euler[1] << (float)Euler[2]; //! Roll Pitch Heading
    udpSocketObject->writeDatagram(datagram, IPAdressObject, portObject);

    //! if second receiver is activated send it also the tracking data
    if (safety2Enable)
    {
        udpSocketSafety2->writeDatagram(datagram, IPAdressSafety2, portSafety2);
    }

}
```

Here is the caller graph for this function:

3.2.2.11 setHeadingOffset()

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

Add a heading offset to the attitude for the case it is wanted by the user.

Parameters

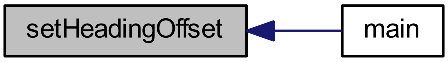
in	*d*	denotes heading offset in degrees.

Definition at line 1140 of file main.cpp.

```
{
    headingOffset = d;
    d = d * 3.141592653589 / 180.0; //! Convert heading offset from degrees to rad

    //! Calculate rotation about x axis
    Mat R_x = (Mat_<double>(3, 3) <<
        1, 0, 0,
        0, 1, 0,
        0, 0, 1
        );

    //! Calculate rotation about y axis
    Mat R_y = (Mat_<double>(3, 3) <<
        1, 0, 0,
        0, 1, 0,
        0, 0, 1
        );

    //! Calculate rotation about z axis
    Mat R_z = (Mat_<double>(3, 3) <<
```

```
        cos(d), -sin(d), 0,
        sin(d), cos(d), 0,
        0, 0, 1);


    //! Combined rotation matrix
    M_HeadingOffset = R_z * R_y * R_x;
}
```

Here is the caller graph for this function:



3.2.2.12 setReference()

```
int setReference ( )
```

Determine the initial position of the object that serves as reference point or as ground frame origin. Computes the pose 200 times and then averages it. The position and attitude are from now on used as navigation CoSy.

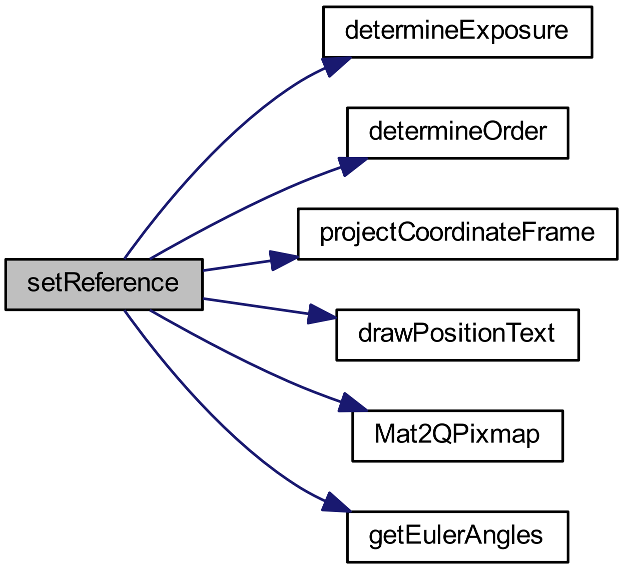
Definition at line 613 of file main.cpp.

```
{
    //! initialize the variables with starting values
    gotOrder = false;
    posRef = 0;
    eulerRef = 0;
    RmatRef = 0;
    Rvec = RvecOriginal;
    Tvec = TvecOriginal;

    determineExposure();

    ss.str("");
    commObj.addLog("Started reference coordinate determination.");

    CameraLibrary_EnableDevelopment();
    //! Initialize Camera SDK ==--
    CameraLibrary::CameraManager::X();

    //! 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 ================----
    CameraManager::X().WaitForInitialization();
    Camera *camera = CameraManager::X().GetCamera();

    //! If no device connected, pop a message box and exit ==--
    if (camera == 0)
    {
        commObj.addLog("No camera found!");
        return 1;
    }

    //! Determine camera resolution to size application window ==----
    int cameraWidth = camera->Width();
    int cameraHeight = camera->Height();
    camera->GetDistortionModel(distModel);
```

```
cv::Mat matFrame(cv::Size(cameraWidth, cameraHeight), CV_8UC1);

//! Set camera mode to precision mode, it directly provides marker coordinates
camera->SetVideoType(Core::PrecisionMode);

//! Start camera output ==--
camera->Start();

//! 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
camera->SetTextOverlay(true);
camera->SetFrameRate(intFrameRate);
camera->SetIntensity(intIntensity);
camera->SetIRFilter(true);
camera->SetContinuousIR(false);
camera->SetHighPowerMode(false);

//! sample some frames and calculate the position and attitude. then average those values and use that as zero position
int numberSamples = 0;
int numberToSample = 200;
double projectionError = 0; //!< difference between the marker points as seen by the camera and the projected marker points with Rvec and Tvec

while (numberSamples < numberToSample)
{
    //! Fetch a new frame from the camera ===---
    Frame *frame = camera->GetFrame();

    if (frame)
    {
        //! Ok, we've received a new frame, lets do something
        //! with it.
        if (frame->ObjectCount() == numberMarkers)
        {
            //!for(int i=0; i<frame->ObjectCount(); i++)
            for (int i = 0; i < numberMarkers; i++)
            {
                cObject *obj = frame->Object(i);
                list_points2dUnsorted[i] = cv::Point2d(obj->X(), obj->Y());
            }

            if (gotOrder == false)
            {
                determineOrder();
            }

            //! sort the 2d points with the correct indices as found in the preceeding order determination algorithm
            for (int w = 0; w < numberMarkers; w++)
            {
                list_points2d[w] = list_points2dUnsorted[pointOrderIndices[w]];
            }
            list_points2dOld = list_points2dUnsorted;

            //!Compute the pose from the 3D-2D corresponses
            solvePnP(list_points3d, list_points2d, cameraMatrix, distCoeffs, Rvec, Tvec, useGuess, methodPNP);

            //! project the marker 3d points with the solution into the camera image CoSy and calculate difference to true camera image
            projectPoints(list_points3d, Rvec, Tvec, cameraMatrix, distCoeffs, list_points2dProjected);
            projectionError = norm(list_points2dProjected, list_points2d);

            double maxValue = 0;
            double minValue = 0;
            minMaxLoc(Tvec.at<double>(2), &minValue, &maxValue);

            if (maxValue > 10000 || minValue < 0)
            {
                ss.str("");
                ss << "Negative z distance, thats not possible. Start the set zero routine again or restart Programm.";
                commObj.addLog(QString::fromStdString(ss.str()));
                frame->Release();
                return 1;
            }

            if (projectionError > 5)
            {
                commObj.addLog("Reprojection error is bigger than 5 pixel. Correct marker configuration loaded?\nMarker position measured precisely?");
```

```
                    frame->Release();
                    return 1;
                }

                if (norm(positionOld) - norm(Tvec) < 0.05)   //!<Iterative Method needs time to converge to solution
                {
                    add(posRef, Tvec, posRef);
                    add(eulerRef, Rvec, eulerRef); //!< That are not the values of yaw, roll and pitch yet! Rodriguez has to be called first.
                    numberSamples++;    //!<  one sample more :D
                    commObj.progressUpdate(numberSamples * 100 / numberToSample);
                }
                positionOld = Tvec;

                Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
                for (int i = 0; i < numberMarkers; i++)
                {
                    circle(cFrame, Point(list_points2d[i].x, list_points2d[i].y), 6, Scalar(0, 225, 0), 3);
                }
                projectCoordinateFrame(cFrame);
                projectPoints(list_points3d, Rvec, Tvec, cameraMatrix, distCoeffs, list_points2d);
                for (int i = 0; i < numberMarkers; i++)
                {
                    circle(cFrame, Point(list_points2d[i].x, list_points2d[i].y), 3, Scalar(225, 0, 0), 3);
                }
                drawPositionText(cFrame, position, eulerAngles, projectionError);

                QPixmap QPFrame;
                QPFrame = Mat2QPixmap(cFrame);
                commObj.changeImage(QPFrame);
                QCoreApplication::processEvents();

            }
            frame->Release();
        }
    }
    //! Release camera ==--
    camera->Release();

    //!Divide by the number of samples to get the mean of the reference position
    divide(posRef, numberToSample, posRef);
    divide(eulerRef, numberToSample, eulerRef); //!< eulerRef is here in Axis Angle notation

    Rodrigues(eulerRef, RmatRef);                //!< axis angle to rotation matrix
    //!-- Euler Angles, finally
    getEulerAngles(RmatRef, eulerRef); //!<  rotation matrix to euler
    ss.str("");
    ss << "RmatRef is:\n";
    ss << RmatRef << "\n";
    ss << "Reference Position is:\n";
    ss << posRef << "[mm] \n";
    ss << "Reference Euler Angles are:\n";
    ss << eulerRef << "[deg] \n";

    //! 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 setReference() function once again
    double error = norm(posRef) - norm(Tvec);
    if (error > 5.0)
    {
        ss << "Caution, distance between reference position and last position is: " << error << "\n Start the set zero routine once again.";
    }
    commObj.addLog(QString::fromStdString(ss.str()));
    commObj.progressUpdate(0);
    return 0;
}
```

Here is the call graph for this function:



3.2.2.13 startTracking()

```
int startTracking ( )
```

Start the loop that fetches frames, computes the position etc and sends it to other computers. This function is the core of this program, hence the pose estimation is done here.

Definition at line 276 of file main.cpp.

```
276                     {
277
278
279     gotOrder = false; //! The order of points, hence which entry in list_points3d corresponds to
       which in list_points2d is not calculated yet
280     Rvec = RvecOriginal; //! Use the value of Rvec that was set in main() as starting value
       for the solvePnP algorithm
281     Tvec = TvecOriginal; //! Use the value of Tvec that was set in main() as starting value
       for the solvePnP algorithm
282     GetLocalTime(&logDate);  //! Get the current date and time to name the log file
283
284     //! Concat the log file name as followed. The file is saved in the folder /logs in the Rigid Track
       installation folder
285     logFileName = "./logs/positionLog_" + QString::number(logDate.wDay) + "_" +
      QString::number(logDate.wMonth) + "_" + QString::number(logDate.wYear);
286     logFileName += "_" + QString::number(logDate.wHour) + "_" + QString::number(
      logDate.wMinute) + "_" + QString::number(logDate.wSecond) + ".txt";
287     logName = logFileName.toStdString(); //! Convert the QString to a standard string
288
289     determineExposure(); //! Get the exposure where the right amount of markers is
       detected
290
291     //! For OptiTrack Ethernet cameras, it's important to enable development mode if you
292     //! want to stop execution for an extended time while debugging without disconnecting
```

```
    //! the Ethernet devices.  Lets do that now:

    CameraLibrary_EnableDevelopment();
    CameraLibrary::CameraManager::X(); //! 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
    CameraManager::X().WaitForInitialization();
    Camera *camera = CameraManager::X().GetCamera();

    //! If no camera can be found, inform user in message log and exit function
    if (camera == 0)
    {
        commObj.addLog("No camera found!");
        return 1;
    }

    //! Determine camera resolution to size application window
    int cameraWidth = camera->Width();
    int cameraHeight = camera->Height();

    camera->SetVideoType(Core::PrecisionMode);  //! Set the camera mode to precision mode, it used greyscale information for marker property calculations

    camera->Start(); //! 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
    camera->SetTextOverlay(true);
    camera->SetExposure(intExposure);    //! Set the camera exposure
    camera->SetIntensity(intIntensity); //! Set the camera infrared LED intensity
    camera->SetFrameRate(intFrameRate); //! Set the camera framerate to 100 Hz
    camera->SetIRFilter(true);  //! Enable the filter that blocks visible light and only passes infrared light
    camera->SetHighPowerMode(true); //! Enable high power mode of the LEDs
    camera->SetContinuousIR(false); //! Disable continuous LED light
    camera->SetThreshold(intThreshold); //! Set threshold for marker detection

    //! Create a new matrix that stores the grayscale picture from the camera
    Mat matFrame = Mat::zeros(cv::Size(cameraWidth, cameraHeight), CV_8UC1);
    QPixmap QPFrame; //! QPixmap is the corresponding Qt class that saves images
    //! Matrix that stores the colored picture, hence marker points, coordinate frame and reprojected points
    Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));

    int v = 0;  //! Helper variable used to kick safety switch
    //! Variables for the min and max values that are needed for sanity checks
    double maxValue = 0;
    double minValue = 0;
    int framesDropped = 0; //! Ff a marker is not visible or accuracy is bad increase this counter
    double projectionError = 0; //! Equals the quality of the tracking

    setUpUDP(); //! Open sockets and ports for UDP communication

    if (safetyEnable) //! If the safety feature is enabled send the starting message
    {
        //! Send enable message, hence send a 9 and then a 1
        data.setNum((int)(9));
        udpSocketSafety->write(data);
        data.setNum((int)(1));
        udpSocketSafety->write(data);
    }

    //! Fetch a new frame from the camera
    bool gotTime = false; //! 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 (!gotTime) //! While no new frame is received loop
    {
        Frame *frame = camera->GetFrame(); //! Get a new camera frame
        if (frame)  //! There is actually a new frame
        {
            timeFirstFrame = frame->TimeStamp(); //! Get the time stamp for the first frame. It is subtracted for the following frames
            frame->Release();   //! Release the frame so the camera can continue
            gotTime = true; //! Exit the while loop
        }
    }

    //! Now enter the main loop that processes each frame and computes the pose, sends it and logs stuff
    while (!exitRequested) //! Check if the user has not pressed "Stop Tracking" yet
    {

        Frame *frame = camera->GetFrame(); //! Fetch a new frame from the camera

        if (frame) //! Did we got a new frame or does the camera still need more time
```

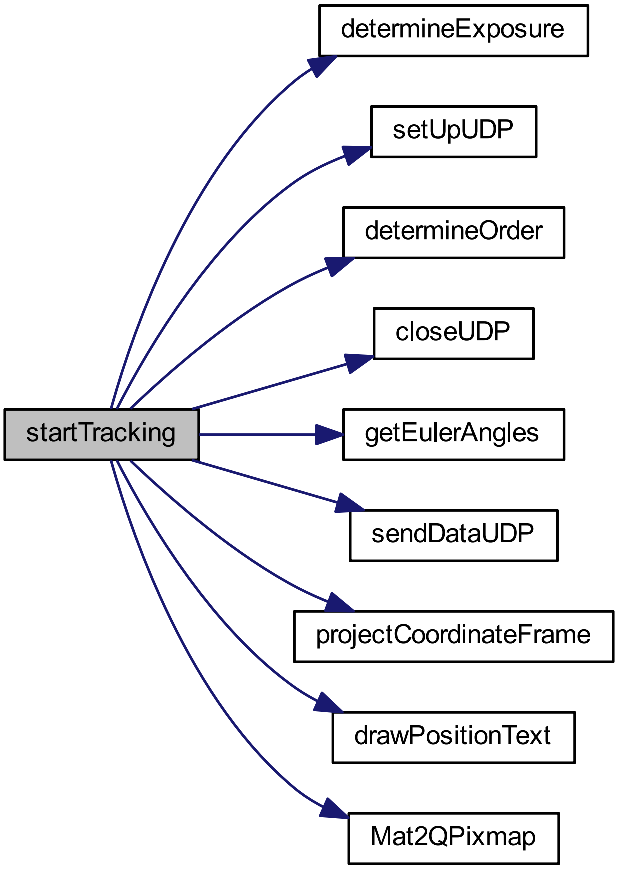
```
    {
        framesDropped++; //! Increase by one, if everything is okay it is decreased at the end of the loop again

        //! Only use this frame it the right number of markers is found in the picture
        if (frame->ObjectCount() == numberMarkers)
        {
            //! 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
            for (int i = 0; i < numberMarkers; i++)
            {
                cObject *obj = frame->Object(i);
                list_points2dUnsorted[i] = cv::Point2d(obj->X(), obj->Y());
            }

            if (gotOrder == false) //! Was the order already determined? This is false for the first frame and from then on true
            {
                determineOrder(); //! Now compute the order
            }

            //! Sort the 2d points with the correct indices as found in the preceeding order determination algorithm
            for (int w = 0; w < numberMarkers; w++)
            {
                list_points2d[w] = list_points2dUnsorted[pointOrderIndices[w]]; //! pointOrderIndices was calculated in determineOrder()
            }

            //! 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.
            for (int j = 0; j < numberMarkers; j++)
            {
                minPointDistance = 5000; //! The sum of point distances is set to something unrealistic large
                for (int k = 0; k < numberMarkers; k++)
                {
                    //! Calculate N_2 norm of unsorted points minus old points
                    currentPointDistance = norm(list_points2dUnsorted[pointOrderIndices[j]] - list_points2dOld[k]);
                    //! If the norm is smaller than minPointDistance the correspondence is more likely to be correct
                    if (currentPointDistance < minPointDistance)
                    {
                        //! Update the array that saves the new point order
                        minPointDistance = currentPointDistance;
                        pointOrderIndicesNew[j] = k;
                    }
                }
            }

            //! Now the new order is found, set the point order to the new value
            for (int k = 0; k < numberMarkers; k++)
            {
                pointOrderIndices[k] = pointOrderIndicesNew[k];
                list_points2d[k] = list_points2dUnsorted[pointOrderIndices[k]];
            }

            //! Save the unsorted position of the marker points for the next loop
            list_points2dOld = list_points2dUnsorted;

            //!Compute the object pose from the 3D-2D corresponses
            solvePnP(list_points3d, list_points2d, cameraMatrix, distCoeffs, Rvec, Tvec, useGuess, methodPNP);

            //! Project the marker 3d points with the solution into the camera image CoSy and calculate difference to true camera image
            projectPoints(list_points3d, Rvec, Tvec, cameraMatrix, distCoeffs, list_points2dProjected);
            projectionError = norm(list_points2dProjected, list_points2d); //! Difference of true pose and found pose

```

```
                //! Increase the framesDropped variable if accuracy of tracking is too bad
                if (projectionError > 5)
                {
                    framesDropped++;
                }
                else
                {
                    framesDropped = 0;  //! 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
                minMaxLoc(Tvec.at<double>(2), &minValue, &maxValue);

                //! Sanity check of values. negative z means the marker CoSy is behind the camera, that's not possible.
                if (minValue < 0)
                {
                    commObj.addLog("Negative z distance, that is not possible. Start the set zero routine again or restart Program.");
                    frame->Release(); //! Release the frame so the camera can move on
                    camera->Release(); //! Release the camera
                    closeUDP(); //! Close all UDP connections so the programm can be closed later on and no resources are locked
                    return 1; //! 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: \f$ T_C^{NM} = Tvec - Tvec_{Ref} \f$
                subtract(Tvec, posRef, position);

                //! Transform the position from the camera CoSy to the navigation CoSy with INS alligned heading and convert from [mm] to [m]
                //! \f$ T_N^{NM} = M_{NC} \times T_C^{NM} \f$
                Mat V = 0.001 * M_HeadingOffset * M_CN.t() * (Mat) position;
                position = V;   //! Position is the result of the preceeding calculation
                position[2] *= invertZ;  //! Invert Z if check box in GUI is activated, hence height above ground is considered

                //! Realtive angle between reference orientation and current orientation
                Rodrigues(Rvec, Rmat);  //! Convert axis angle respresentation to ordinary rotation matrix

                //! The difference of the reference rotation and the current rotation
                //! \f$ R_{ NM } = M_{ NC } \times R_{ CM } \f$
                Rmat = RmatRef.t() *Rmat;

                //! Euler Angles, finally
                getEulerAngles(Rmat, eulerAngles); //! Get the euler angles from the rotation matrix
                eulerAngles[2] += headingOffset; //! 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
                frameTime = frame->TimeStamp() - timeOld;   //! Time between the old frame and the current frame
                timeOld = frame->TimeStamp();    //! Set the old frame time to the current  one
                velocity[0] = (position[0] - positionOld[0]) / frameTime; //! Calculate the x velocity with finite differences
                velocity[1] = (position[1] - positionOld[1]) / frameTime; //! Calculate the y velocity with finite differences
                velocity[2] = (position[2] - positionOld[2]) / frameTime; //! Calculate the z velocity with finite differences
                positionOld = position;  //! Set the old position to the current one for next frame velocity calcuation

                eulerAngles[0] = eulerAngles[0] * -3.141592653589 / 180.0; //! Convert the Euler angles from degrees to rad
                eulerAngles[1] = eulerAngles[1] * -3.141592653589 / 180.0;
                eulerAngles[2] = eulerAngles[2] * 3.141592653589 / 180.0;

                //! Send position and Euler angles over WiFi with 100 Hz
                sendDataUDP(position, eulerAngles);

                //! Save the values in a log file, values are:
                //! Time sinc tracking started  Position    Euler Angles    Velocity
                logfile.open(logName, std::ios::app); //! Open the log file, the folder is RigidTrackInstallationFolder/logs
                logfile << frame->TimeStamp() - timeFirstFrame << ";" << position[0] << ";" << position[1] << ";" << position[2] << ";";
                logfile << eulerAngles[0] << ";" << eulerAngles[1] << ";" << eulerAngles[2] << ";";
                logfile << velocity[0] << ";" << velocity[1] << ";" << velocity[2] << "\n";
```

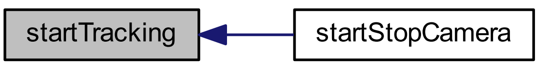
```
                logfile.close(); //! Close the file to save values
            }

            //! 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
            if (safetyEnable)
            {
                if ((abs(position[0]) < safetyBoxLength && abs(position[1]) < safetyBoxLength && abs(position[2]) < safetyBoxLength))
                {
                    //! Absolute Euler angles must be smaller than allowed value. Heading is not considered

                    if ((abs(eulerAngles[0]) < safetyAngle && abs(eulerAngles[1]) < safetyAngle))
                    {
                        //! Send the OKAY signal to the desired computer every 5th time
                        if (v == 5) {
                            data.setNum((int)(1));
                            udpSocketSafety->write(data); //! Send the 1
                            v = 0; //! 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)
                    else
                    {
                        data.setNum((int)(0));  //! Send the shutdown signal, a 0
                        udpSocketSafety->write(data);
                        commObj.addLog("Object exceeded allowed Euler angles, shutdown signal sent."); //! Inform the user

                    }
                }
                //! The position of the object exceeded the allowed position, shut the object down
                else
                {
                    data.setNum((int)(0));  //! Send the shutdown signal, a 0
                    udpSocketSafety->write(data);
                    commObj.addLog("Object left allowed area, shutdown signal sent."); //! Inform the user

                }
            }

            //! Inform the user if tracking system is disturbed (marker lost or so) or error was too big
            if (framesDropped > 10)
            {
                if (safetyEnable) //! Also send the shutdown signal
                {
                    data.setNum((int)(0));  //! Send the shutdown signal, a 0
                    udpSocketSafety->write(data);
                }
                commObj.addLog("Lost marker points or precision was bad!"); //! Inform the user
                framesDropped = 0;
            }

            //! Rasterize the frame so it can be shown in the GUI
            frame->Rasterize(cameraWidth, cameraHeight, matFrame.step, BACKBUFFER_BITSPERPIXEL, matFrame.data);

            //! Convert the frame from greyscale as it comes from the camera to rgb color
            cvtColor(matFrame, cFrame, COLOR_GRAY2RGB);

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

            //! Project the marker points from 3D to the camera image frame (2d) with the computed pose
            projectPoints(list_points3d, Rvec, Tvec, cameraMatrix, distCoeffs, list_points2d);
            for (int i = 0; i < numberMarkers; i++)
            {
                //! 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
                circle(cFrame, Point(list_points2d[i].x, list_points2d[i].y), 3, Scalar(225, 0, 0), 3);
            }

            //! Write the current position, attitude and error values as text in the frame
            drawPositionText(cFrame, position, eulerAngles, projectionError);

            //! Send the new camera picture to the GUI and call the GUI processing routine
            QPixmap QPFrame;
            QPFrame = Mat2QPixmap(cFrame);
            commObj.changeImage(QPFrame); //! Update the picture in the GUI
```

```
            QCoreApplication::processEvents(); //! Give Qt time to handle everything

            //! Release the camera frame to fetch the new one
            frame->Release();
        }
    }

    //! User choose to stop the tracking, clean things up
    closeUDP(); //! Close the UDP connections so resources are deallocated
    camera->Release();  //! Release camera
    return 0;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.2.14 testAlgorithms()

```
void testAlgorithms ( )
```

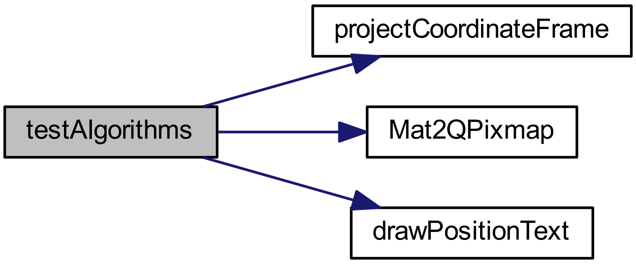
Project some points from 3D to 2D and then check the accuracy of the algorithms. Mainly to generate something that can be shown in the camera view so the user knows everything loaded correctly.

Definition at line 970 of file main.cpp.

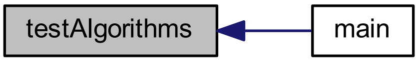
```
971 {
972
973     int _methodPNP;
974
975     std::vector<Point2d> noise(numberMarkers);
976
977     RvecOriginal = Rvec;
978     TvecOriginal = Tvec;
979
980     projectPoints(list_points3d, Rvec, Tvec, cameraMatrix,
      distCoeffs, list_points2dProjected);
981
982     ss.str("");
983     ss << "Unsorted Points 2D Projected \n";
984     ss << list_points2dProjected << "\n";
985     commObj.addLog(QString::fromStdString(ss.str()));
986
987     Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
988     for (int i = 0; i < numberMarkers; i++)
989     {
990         circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 6, Scalar(0, 255, 0
      ), 3);
991     }
992
993     projectCoordinateFrame(cFrame);
994
995     ss.str("");
996     ss << "=====================================================\n";
997     ss << "================= Projected Points =================\n";
998     ss << list_points2dProjected << "\n";
999
1000     randn(noise, 0, 0.5);
1001     add(list_points2dProjected, noise, list_points2dProjected);
1002
1003     ss << "================ With Noise Points =================\n";
1004     ss << list_points2dProjected << "\n";
1005     commObj.addLog(QString::fromStdString(ss.str()));
1006
1007
1008     bool useGuess = true;
1009     _methodPNP = 0; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP  //!< not used
1010
1011     solvePnP(list_points3d, list_points2dProjected, cameraMatrix,
      distCoeffs, Rvec, Tvec, useGuess, _methodPNP);
1012
1013     ss.str("");
1014     ss << "=====================================================\n";
1015     ss << "==================== Iterative =====================\n";
1016     ss << "rvec: " << "\n";
1017     ss << Rvec << "\n";
1018     ss << "tvec: " << "\n";
1019     ss << Tvec << "\n";
1020
1021     commObj.addLog(QString::fromStdString(ss.str()));
1022
1023     _methodPNP = 1; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP UPnP not used
1024     Rvec = cv::Mat::zeros(3, 1, CV_64F);
1025     Tvec = cv::Mat::zeros(3, 1, CV_64F);
1026     solvePnP(list_points3d, list_points2dProjected, cameraMatrix,
      distCoeffs, Rvec, Tvec, useGuess, _methodPNP);
1027
1028     ss.str("");
1029     ss << "=====================================================\n";
1030     ss << "====================    EPNP    ====================\n";
1031     ss << "rvec: " << "\n";
1032     ss << Rvec << "\n";
1033     ss << "tvec: " << "\n";
1034     ss << Tvec << "\n";
```

```

    projectPoints(list_points3d, Rvec, Tvec, cameraMatrix, distCoeffs, list_points2dProjected);
    for (int i = 0; i < numberMarkers; i++)
    {
        circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 3, Scalar(255, 0, 0), 3);
    }
    QPixmap QPFrame;
    QPFrame = Mat2QPixmap(cFrame);
    commObj.changeImage(QPFrame);
    QCoreApplication::processEvents();
    commObj.addLog(QString::fromStdString(ss.str()));
    if (numberMarkers == 4)
    {
        _methodPNP = 2; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP  //!< not used
        Rvec = cv::Mat::zeros(3, 1, CV_64F);
        Tvec = cv::Mat::zeros(3, 1, CV_64F);
        solvePnP(list_points3d, list_points2dProjected, cameraMatrix, distCoeffs, Rvec, Tvec, useGuess, _methodPNP);

        ss.str("");
        ss << "=====================================================\n";
        ss << "====================     P3P     =====================\n";
        ss << "rvec: " << "\n";
        ss << Rvec << "\n";
        ss << "tvec: " << "\n";
        ss << Tvec << "\n";

        projectPoints(list_points3d, Rvec, Tvec, cameraMatrix, distCoeffs, list_points2dProjected);
        for (int i = 0; i < numberMarkers; i++)
        {
            circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 3, Scalar(255, 0, 0), 3);
        }
        double projectionError = norm(list_points2dProjected, list_points2d);
        putText(cFrame, "Testing Algorithms Finished", cv::Point(5, 420), 1, 1, cv::Scalar(255, 255, 255));
        drawPositionText(cFrame, position, eulerAngles, projectionError);

        QPixmap QPFrame;
        QPFrame = Mat2QPixmap(cFrame);
        commObj.changeImage(QPFrame);
        QCoreApplication::processEvents();
        commObj.addLog(QString::fromStdString(ss.str()));
    }

    _methodPNP = 4; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP  //!< not used
    Rvec = cv::Mat::zeros(3, 1, CV_64F);
    Tvec = cv::Mat::zeros(3, 1, CV_64F);
    solvePnP(list_points3d, list_points2dProjected, cameraMatrix, distCoeffs, Rvec, Tvec, useGuess, _methodPNP);

    ss.str("");
    ss << "=====================================================\n";
    ss << "====================    UPNP    =====================\n";
    ss << "rvec: " << "\n";
    ss << Rvec << "\n";
    ss << "tvec: " << "\n";
    ss << Tvec << "\n";

    commObj.addLog(QString::fromStdString(ss.str()));

    Rvec = RvecOriginal;
    Tvec = TvecOriginal;

}
```

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.3 Variable Documentation

3.2.3.1 commObj

```
commObject commObj
```

class that handles the communication from main.cpp to the GUI

Now declare variables that are used across the main.cpp file. Basically almost every variable used is declared here.

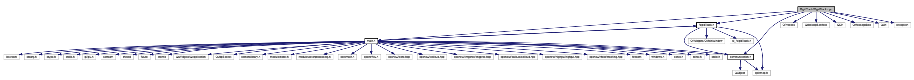
Definition at line 68 of file main.cpp.

3.3 RigidTrack/RigidTrack.cpp File Reference

Rigid Track GUI source that contains functions for GUI events.

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

Include dependency graph for RigidTrack.cpp:



3.3.1 Detailed Description

Rigid Track GUI source that contains functions for GUI events.

Author

Florian J.T. Wachter

Version

1.0

Date

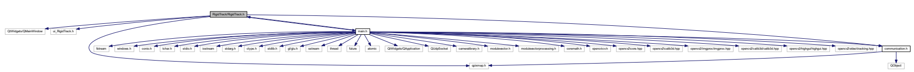
April, 8th 2017

3.4 RigidTrack/RigidTrack.h File Reference

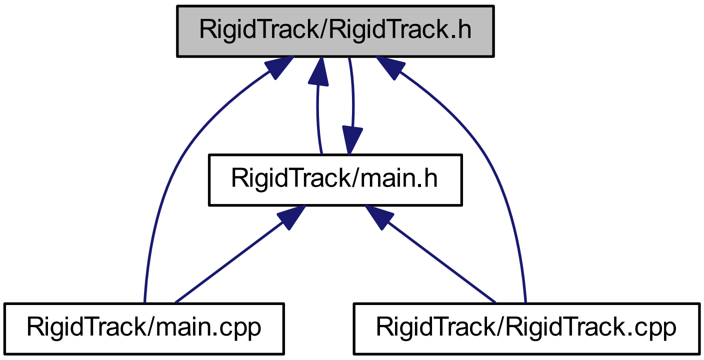
Rigid Track GUI source header with Qt Signals and Slots.

```
#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:



3.4.1 Detailed Description

Rigid Track GUI source header with Qt Signals and Slots.

Author

Florian J.T. Wachter

Version

1.0

Date

April, 8th 2017

Index