Rigid Track

Generated by Doxygen 1.8.13

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

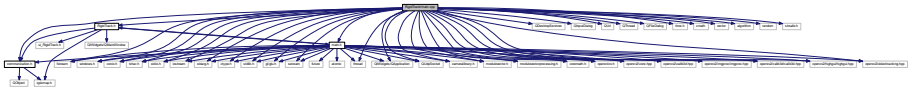
3.1 RigidTrack/main.cpp File Reference

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.

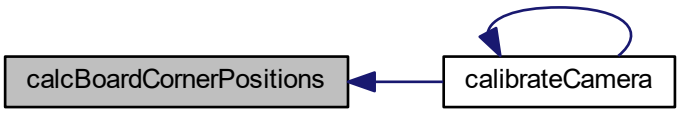
Parameters

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 229 of file main.cpp.

```
230 {
231     corners.clear();
232
233     for (int i = 0; i < boardSize.height; ++i)
234         for (int j = 0; j < boardSize.width; ++j)
235             corners.push_back(Point3f(float(j*squareSize), float(i*squareSize), 0));
236 }
```

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.

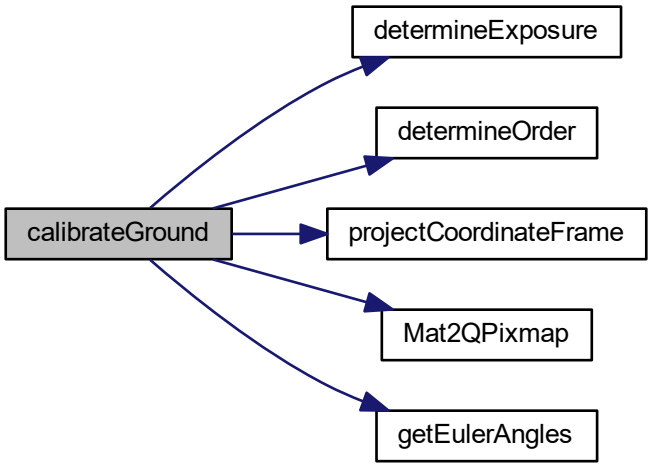
Definition at line 1563 of file main.cpp.

```
1564 {
1565     //! initialize the variables with starting values
1566     gotOrder = false;
1567     posRef = 0;
1568     eulerRef = 0;
1569     RmatRef = 0;
1570     Rvec = RvecOriginal;
1571     Tvec = TvecOriginal;
1572 
1573     determineExposure();
1574 
1575     ss.str("");
1576     commObj.addLog("Started ground calibration");
1577 
1578     CameraLibrary_EnableDevelopment();
1579     //! Initialize Camera SDK ===-
1580     CameraLibrary::CameraManager::X();
1581 
1582     //! At this point the Camera SDK is actively looking for all connected cameras and will initialize
1583     //! them on it's own.
1584 
1585     //! Get a connected camera ==================---
1586     CameraManager::X().WaitForInitialization();
1587     Camera *camera = CameraManager::X().GetCamera();
1588 
1589     //! If no device connected, pop a message box and exit ==--
1590     if (camera == 0)
1591     {
1592         commObj.addLog("No camera found!");
1593         return 1;
1594     }
1595 
1596     //! Determine camera resolution to size application window ==----
1597     int cameraWidth = camera->Width();
1598     int cameraHeight = camera->Height();
1599     camera->GetDistortionModel(distModel);
1600     cv::Mat matFrame(cv::Size(cameraWidth, cameraHeight), CV_8UC1);
1601 
1602     //! Set camera mode to precision mode, it directly provides marker coordinates
1603     camera->SetVideoType(Core::PrecisionMode);
1604 
1605     //! Start camera output ==--
1606     camera->Start();
1607 
1608     //! Turn on some overlay text so it's clear things are     ======-
1609     //! working even if there is nothing in the camera's view. ======-
1610     //! Set some other parameters as well of the camera
1611     camera->SetTextOverlay(true);
1612     camera->SetFrameRate(intFrameRate);
1613     camera->SetIntensity(intIntensity);
1614     camera->SetIRFilter(true);
1615     camera->SetContinuousIR(false);
1616     camera->SetHighPowerMode(false);
1617 
1618     //! sample some frames and calculate the position and attitude. then average those values and use that
      as zero position
1619     int numberSamples = 0;
1620     int numberToSample = 200;
1621     double projectionError = 0;
1622 
1623     while (numberSamples < numberToSample)
1624     {
1625         //! Fetch a new frame from the camera ===---
1626         Frame *frame = camera->GetFrame();
1627 
1628         if (frame)
1629         {
1630             //! Ok, we've received a new frame, lets do something
1631             //! with it.
1632             if (frame->ObjectCount() == numberMarkers)
1633             {
1634                 //!for(int i=0; i<frame->ObjectCount(); i++)
1635                 for (int i = 0; i < numberMarkers; i++)
1636                 {
1637                     cObject *obj = frame->Object(i);
1638                     list_points2dUnsorted[i] = cv::Point2d(obj->X(), obj->Y());
1639                 }
1640 
1641                 if (gotOrder == false)
1642                 {
1643                     determineOrder();
1644                 }
1645 
1646                 //! sort the 2d points with the correct indices as found in the preceeding order
      determination algorithm
1647                 for (int w = 0; w < numberMarkers; w++)
1648                 {
```

```
1649                    list_points2d[w] = list_points2dUnsorted[
      pointOrderIndices[w]];
1650                }
1651                list_points2dOld = list_points2dUnsorted;
1652
1653                //!Compute the pose from the 3D-2D corresponses
1654                solvePnP(list_points3d, list_points2d,
      cameraMatrix, distCoeffs, Rvec, Tvec, useGuess,
      methodPNP);
1655
1656                //! project the marker 3d points with the solution into the camera image CoSy and calculate
       difference to true camera image
1657                projectPoints(list_points3d, Rvec, Tvec,
      cameraMatrix, distCoeffs, list_points2dProjected);
1658                projectionError = norm(list_points2dProjected,
      list_points2d);
1659
1660                if (projectionError > 3)
1661                {
1662                    commObj.addLog("Reprojection error is bigger than 3 pixel. Correct marker
       configuration loaded?\nMarker position measured precisely?");
1663                    frame->Release();
1664                    return 1;
1665                }
1666
1667                double maxValue = 0;
1668                double minValue = 0;
1669                minMaxLoc(Tvec.at<double>(2), &minValue, &maxValue);
1670
1671                if (maxValue > 10000 || minValue < 0)
1672                {
1673
1674
1675                    commObj.addLog("Negative z distance, thats not possible. Start the set zero
       routine again and check marker configurations.");
1676                    frame->Release();
1677                    return 1;
1678                }
1679
1680                if (norm(positionOld) - norm(Tvec) < 0.05)   //!<Iterative Method needs time
       to converge to solution
1681                {
1682                    add(posRef, Tvec, posRef);
1683                    add(eulerRef, Rvec, eulerRef); //!< That are not the values of yaw,
       roll and pitch yet! Rodriguez has to be called first.
1684                    numberSamples++;    //!<-- one sample more :D
1685                    commObj.progressUpdate(numberSamples * 100 / numberToSample);
1686                }
1687                positionOld = Tvec;
1688
1689                Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
1690                for (int i = 0; i < numberMarkers; i++)
1691                {
1692                    circle(cFrame, Point(list_points2d[i].x,
      list_points2d[i].y), 6, Scalar(0, 225, 0), 3);
1693                }
1694                projectCoordinateFrame(cFrame);
1695                projectPoints(list_points3d, Rvec, Tvec,
      cameraMatrix, distCoeffs, list_points2d);
1696                for (int i = 0; i < numberMarkers; i++)
1697                {
1698                    circle(cFrame, Point(list_points2d[i].x,
      list_points2d[i].y), 3, Scalar(225, 0, 0), 3);
1699                }
1700
1701                QPixmap QPFrame;
1702                QPFrame = Mat2QPixmap(cFrame);
1703                commObj.changeImage(QPFrame);
1704                QCoreApplication::processEvents();
1705
1706            }
1707            frame->Release();
1708        }
1709    }
1710    //! Release camera ==--
1711    camera->Release();
1712
1713    //!Divide by the number of samples to get the mean of the reference position
1714    divide(posRef, numberToSample, posRef);
1715    divide(eulerRef, numberToSample, eulerRef); //!< eulerRef is here in Axis Angle
       notation
1716
1717    Rodrigues(eulerRef, RmatRef);               //!< axis angle to rotation matrix
1718
1719    getEulerAngles(RmatRef, eulerRef); //!<  rotation matrix to euler
1720    ss.str("");
1721    ss << "RmatRef is:\n";
```

```
1722     ss << RmatRef << "\n";
1723     ss << "Reference Position is:\n";
1724     ss << posRef << "[mm] \n";
1725     ss << "Reference Euler angles are:\n";
1726     ss << eulerRef << "[deg] \n";
1727
1728     //! Save the obtained calibration coefficients in a file for later use
1729     QString fileName = QFileDialog::getSaveFileName(nullptr, "Save ground calibration file", "
     referenceData.xml", "Calibration File (*.xml);;All Files (*)");
1730     FileStorage fs(fileName.toUtf8().constData(), FileStorage::WRITE);
1731     fs << "M_NC" << RmatRef;
1732     fs << "eulerRef" << eulerRef;
1733     strBuf = fs.releaseAndGetString();
1734     commObj.changeStatus(QString::fromStdString(strBuf));
1735     commObj.addLog("Saved ground calibration!");
1736     commObj.progressUpdate(0);
1737     return 0;
1738 }
```

Here is the call graph for this function:



3.1.2.3 closeUDP()

```
void closeUDP ( )
```

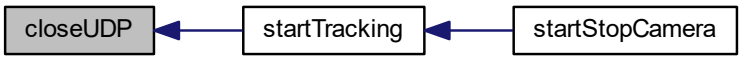
Close the UDP ports again to release network interfaces etc. If this is not done the network resources are still occupied and the program can't exit properly.

Definition at line 1173 of file main.cpp.

```
1174 {
1175     //! check if the socket is open and if yes close it
1176     if (udpSocketObject->isOpen())
1177     {
1178         udpSocketObject->close();
1179     }
1180
```

```
1181     if (udpSocketSafety->isOpen())
1182     {
1183         udpSocketSafety->close();
1184     }
1185
1186     if (udpSocketSafety2->isOpen())
1187     {
1188         udpSocketSafety2->close();
1189     }
1190     commObj.addLog("Closed all UDP ports.");
1191 }
```

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.

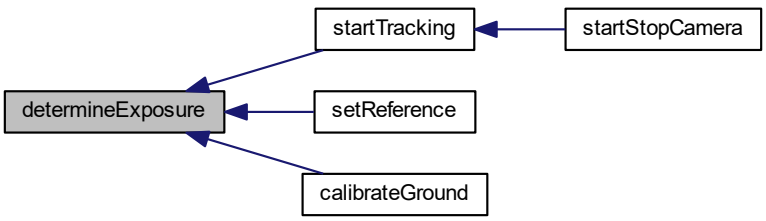
Definition at line 1362 of file main.cpp.

```
1363 {
1364     //! For OptiTrack Ethernet cameras, it's important to enable development mode if you
1365     //! want to stop execution for an extended time while debugging without disconnecting
1366     //! the Ethernet devices.  Lets do that now:
1367
1368     CameraLibrary_EnableDevelopment();
1369
1370     //! Initialize Camera SDK ==--
1371     CameraLibrary::CameraManager::X();
1372
1373     //! At this point the Camera SDK is actively looking for all connected cameras and will initialize
1374     //! them on it's own.
1375
1376     //! Get a connected camera ====================
1377     CameraManager::X().WaitForInitialization();
1378     Camera *camera = CameraManager::X().GetCamera();
1379
1380     //! If no device connected, pop a message box and exit ==--
1381     if (camera == 0)
1382     {
1383         commObj.addLog("No camera found!");
1384         return 1;
1385     }
1386
1387     //! Determine camera resolution to size application window ==----
1388     int cameraWidth = camera->Width();
1389     int cameraHeight = camera->Height();
1390
1391     camera->SetVideoType(Core::PrecisionMode);  //! set the camera mode to precision mode, it used
       greyscale imformation for marker property calculations
1392
1393                                                 //! Start camera output ==--
1394     camera->Start();
1395
```

```
1396     //! Turn on some overlay text so it's clear things are     ======
1397     //! working even if there is nothing in the camera's view. ======
1398     camera->SetTextOverlay(true);
1399     camera->SetExposure(intExposure);    //! set the camera exposure
1400     camera->SetIntensity(intIntensity); //! set the camera infrared LED intensity
1401     camera->SetFrameRate(intFrameRate); //! set the camera framerate to 100 Hz
1402     camera->SetIRFilter(true);  //! enable the filter that blocks visible light and only passes infrared
      light
1403     camera->SetHighPowerMode(true); //! enable high power mode of the leds
1404     camera->SetContinuousIR(false); //! enable continuous LED light
1405     camera->SetThreshold(intThreshold); //! set threshold for marker detection
1406
1407     //!set exposure such that num markers are visible
1408     int numberObjects = 0;  //! Number of objects (markers) found in the current picture with the given
      exposure
1409     int minExposure = 1;    //! exposure when objects detected the first time is numberMarkers
1410     int maxExposure = 480;  //! exposure when objects detected is first time numberMarkers+1
1411     intExposure = minExposure;   //! set the exposure to the smallest value possible
1412     int numberTries = 0;    //! if the markers arent found after numberTries then there might be no markers
      at all in the real world
1413
1414                             //! Determine minimum exposure, hence when are numberMarkers objects detected
1415     camera->SetExposure(intExposure);
1416     while (numberObjects != numberMarkers && numberTries < 48)
1417     {
1418         //! get a new camera frame
1419         Frame *frame = camera->GetFrame();
1420         if (frame) //! frame received
1421         {
1422             numberObjects = frame->ObjectCount();   //! how many objects are detected in the image
1423             if (numberObjects == numberMarkers) { minExposure =
     intExposure; frame->Release(); break; } //! if the right amount if markers is found, exit while
      loop
1424             //! not the right amount of markers was found so increase the exposure and try again
1425             numberTries++;
1426             intExposure += 10;
1427             camera->SetExposure(intExposure);
1428             ss.str("");
1429             ss << "Exposure: " << intExposure << "\t";
1430             ss << "Objects found: " << numberObjects;
1431             commObj.addLog(QString::fromStdString(ss.str()));
1432             frame->Release();
1433         }
1434     }
1435
1436     //! Now determine maximum exposure, hence when are numberMarkers+1 objects detected
1437     numberTries = 0;    //! if the markers arent found after numberTries then there might be no markers at
      all in the real world
1438     intExposure = maxExposure;
1439     camera->SetExposure(intExposure);
1440     numberObjects = 0;
1441     while (numberObjects != numberMarkers && numberTries < 48)
1442     {
1443         Frame *frame = camera->GetFrame();
1444         if (frame)
1445         {
1446             numberObjects = frame->ObjectCount(); //! how many objects are detected in the image
1447             if (numberObjects == numberMarkers) { maxExposure =
     intExposure; frame->Release(); break; } //! if the right amount if markers is found, exit while
      loop
1448
1449             //! not the right amount of markers was found so decrease the exposure and try again
1450             intExposure -= 10;
1451             numberTries++;
1452             camera->SetExposure(intExposure);
1453             ss.str("");
1454             ss << "Exposure: " << intExposure << "\t";
1455             ss << "Objects found: " << numberObjects;
1456             commObj.addLog(QString::fromStdString(ss.str()));
1457             frame->Release();
1458         }
1459     }
1460
1461     //! set the exposure to the mean of min and max exposure determined
1462     camera->SetExposure((minExposure + maxExposure) / 2.0);
1463
1464     //! and now check if the correct amount of markers is detected with that new value
1465     while (1)
1466     {
1467         Frame *frame = camera->GetFrame();
1468         if (frame)
1469         {
1470             numberObjects = frame->ObjectCount(); //! how many objects are detected in the image
1471             if (numberObjects != numberMarkers) //! are all markers and not more or less
      detected in the image
1472             {
1473                 frame->Release();
```

```
1474                 commObj.addLog("Was not able to detect the right amount of markers.");
1475                 //! Release camera ===-
1476                 camera->Release();
1477                 return 1;
1478             }
1479             else  //! all markers and not more or less are found
1480             {
1481                 frame->Release();
1482                 intExposure = (minExposure + maxExposure) / 2.0;
1483                 commObj.addLog("Found the correct number of markers.");
1484                 commObj.addLog("Exposure set to:");
1485                 commObj.addLog(QString::number(intExposure));
1486                 break;
1487             }
1488         }
1489     }
1490
1491     camera->Release();
1492     return 0;
1493
1494 }
```

Here is the caller graph for this function:



3.1.2.5 determineOrder()

```
void determineOrder ( )
```

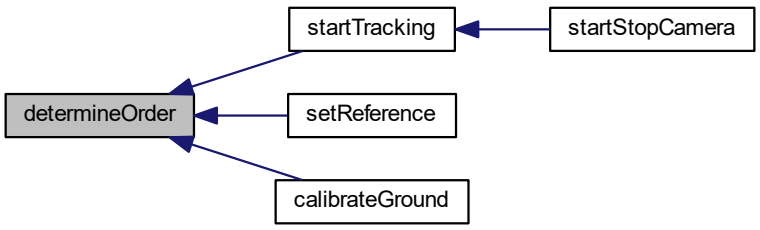
Compute the order of the marker points in 2D so they are the same as in the 3D array. Hence marker 1 must be in first place for both, list_points2d and list_points3d.

Definition at line 1498 of file main.cpp.

```
1499 {
1500     //! determine the 3D-2D correspondences that are crucial for the PnP algorithm
1501     //! Try every possible correspondence and solve PnP
1502     //! Then project the 3D marker points into the 2D camera image and check the difference
1503     //! between projected points and points as seen by the camera
1504     //! the corresponce with the smallest difference is probably the correct one
1505
1506         //! the difference between true 2D points and projected points is super big
1507     minPointDistance = 5000;
1508     std::sort(pointOrderIndices, pointOrderIndices + 4);
1509
1510     //! now try every possible permutation of correspondence
1511     do {
1512         //! reset the starting values for solvePnP
1513         Rvec = RvecOriginal;
1514         Tvec = TvecOriginal;
1515
1516         //! sort the 2d points with the current permutation
```

```
1517         for (int m = 0; m < numberMarkers; m++)
1518         {
1519             list_points2d[m] = list_points2dUnsorted[
     pointOrderIndices[m]];
1520         }
1521 
1522         //! Call solve PNP with P3P since its more robust and sufficient for start value determination
1523         solvePnP(list_points3d, list_points2d,
     cameraMatrix, distCoeffs, Rvec, Tvec, useGuess, SOLVEPNP_P3P);
1524 
1525         //! set the current difference of all point correspondences to zero
1526         currentPointDistance = 0;
1527 
1528         //! project the 3D points with the solvePnP solution onto 2D
1529         projectPoints(list_points3d, Rvec, Tvec,
     cameraMatrix, distCoeffs, list_points2dProjected);
1530 
1531         //! now compute the absolute difference (error)
1532         for (int n = 0; n < numberMarkers; n++)
1533         {
1534             currentPointDistance += norm(list_points2d[n] -
     list_points2dProjected[n]);
1535         }
1536 
1537         //! if the difference with the current permutation is smaller than the smallest value till now
1538         //! it is probably the more correct permutation
1539         if (currentPointDistance < minPointDistance)
1540         {
1541             minPointDistance = currentPointDistance;    //!< set the
      smallest value of difference to the current one
1542             for (int b = 0; b < numberMarkers; b++)    //!< now safe the better permutation
1543             {
1544                 pointOrderIndicesNew[b] = pointOrderIndices[b];
1545             }
1546         }
1547 
1548 
1549     }
1550     //! try every permutation
1551     while (std::next_permutation(pointOrderIndices,
     pointOrderIndices + 4));
1552 
1553     //! now that the correct order is found assign it to the indices array
1554     for (int w = 0; w < numberMarkers; w++)
1555     {
1556         pointOrderIndices[w] = pointOrderIndicesNew[w];
1557     }
1558     gotOrder = true;
1559 }
```

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.

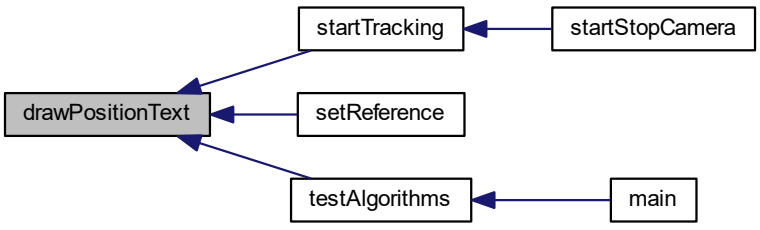
Parameters

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 1315 of file main.cpp.

```
1316 {
1317     ss.str("");
1318     ss << "X: " << Position[0] << " m";
1319     putText(Picture, ss.str(), cv::Point(200, 440), 1, 1, cv::Scalar(255, 255, 255));
1320
1321     ss.str("");
1322     ss << "Y: " << Position[1] << " m";
1323     putText(Picture, ss.str(), cv::Point(200, 455), 1, 1, cv::Scalar(255, 255, 255));
1324
1325     ss.str("");
1326     ss << "Z: " << Position[2] << " m";
1327     putText(Picture, ss.str(), cv::Point(200, 470), 1, 1, cv::Scalar(255, 255, 255));
1328
1329     ss.str("");
1330     ss << "Heading: " << Euler[2] << " deg";
1331     putText(Picture, ss.str(), cv::Point(350, 440), 1, 1, cv::Scalar(255, 255, 255));
1332
1333     ss.str("");
1334     ss << "Pitch: " << Euler[1] << " deg";
1335     putText(Picture, ss.str(), cv::Point(350, 455), 1, 1, cv::Scalar(255, 255, 255));
1336
1337     ss.str("");
1338     ss << "Roll: " << Euler[0] << " deg";
1339     putText(Picture, ss.str(), cv::Point(350, 470), 1, 1, cv::Scalar(255, 255, 255));
1340
1341     ss.str("");
1342     ss << "Error: " << error << " px";
1343     putText(Picture, ss.str(), cv::Point(10, 470), 1, 1, cv::Scalar(255, 255, 255));
1344 }
```

Here is the caller graph for this function:



3.1.2.7 getEulerAngles()

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

Get the euler angles from a rotation matrix

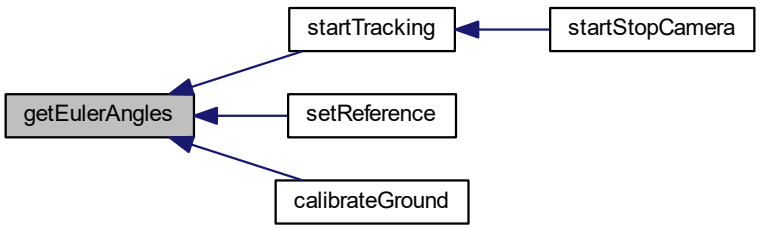
Parameters

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 241 of file main.cpp.

```
241                                                                 {
242
243     Mat cameraMatrix, rotMatrix, transVect, rotMatrixX, rotMatrixY, rotMatrixZ;
244     double* _r = rotCamerMatrix.ptr<double>();
245     double projMatrix[12] = { _r[0],_r[1],_r[2],0,
246         _r[3],_r[4],_r[5],0,
247         _r[6],_r[7],_r[8],0 };
248
249     decomposeProjectionMatrix(Mat(3, 4, CV_64FC1, projMatrix),
250         cameraMatrix,
251         rotMatrix,
252         transVect,
253         rotMatrixX,
254         rotMatrixY,
255         rotMatrixZ,
256         eulerAngles);
257 }
```

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.

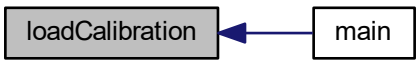
Parameters

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 923 of file main.cpp.

```
923                                 {
924
925     QString fileName;
926     if (method == 0)
927     {
928         fileName = "calibration.xml";
929     }
930     else
931     {
932         fileName = QFileDialog::getOpenFileName(nullptr, "Choose a previous saved calibration file", "", "
    Calibration Files (*.xml);;All Files (*)");
933         if (fileName.length() == 0)
934         {
935             fileName = "calibration.xml";
936         }
937     }
938     FileStorage fs;
939     fs.open(fileName.toUtf8().constData(), FileStorage::READ);
940     fs["CameraMatrix"] >> cameraMatrix;
941     fs["DistCoeff"] >> distCoeffs;
942     commObj.addLog("Loaded calibration from file:");
943     commObj.addLog(fileName);
944     ss.str("");
945     ss << "\nCamera Matrix is" << "\n" << cameraMatrix << "\n";
946     ss << "\nDistortion Coefficients are" << "\n" << distCoeffs << "\n";
947     commObj.addLog(QString::fromStdString(ss.str()));
948 }
```

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 1348 of file main.cpp.

```
1349 {
1350     //! Open the referenceData.xml that contains the rotation from camera CoSy to ground CoSy
1351     FileStorage fs;
1352     fs.open("referenceData.xml", FileStorage::READ);
1353     fs["M_NC"] >> M_CN;
1354     fs["M_NC"] >> RmatRef;
1355     fs["posRef"] >> posRef;
1356     fs["eulerRef"] >> eulerRef;
1357     commObj.addLog("Loaded reference pose.");
1358 }
```

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.

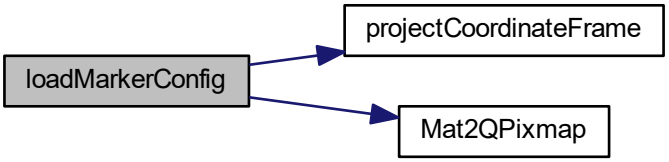
Definition at line 1195 of file main.cpp.

```
1196 {
1197     QString fileName;
1198     //! during start up of the programm load the standard marker configuration
1199     if (method == 0)
1200     {
1201         //! open the standard marker configuration file
1202         FileStorage fs;
1203         fs.open("markerStandard.xml", FileStorage::READ);
1204
1205         //! copy the values to the respective variables
1206         fs["numberMarkers"] >> numberMarkers;
1207
1208         //! inizialise vectors with correct length depending on the number of markers
1209         list_points3d = std::vector<Point3d>(numberMarkers);
1210         list_points2d = std::vector<Point2d>(numberMarkers);
1211         list_points2dOld = std::vector<Point2d>(numberMarkers);
1212         list_points2dDifference = std::vector<double>(
      numberMarkers);
1213         list_points2dProjected = std::vector<Point2d>(
      numberMarkers);
1214         list_points2dUnsorted = std::vector<Point2d>(
      numberMarkers);
1215
```

```
1216         //! save the marker locations in the points3d vector
1217         fs["list_points3d"] >> list_points3d;
1218         fs.release();
1219         commObj.addLog("Loaded marker configuration from file:");
1220         commObj.addLog(fileName);
1221 
1222 
1223 
1224     }
1225     else
1226     {
1227         //! if the load marker configuration button was clicked show a open file dialog
1228         fileName = QFileDialog::getOpenFileName(nullptr, "Choose a previous saved marker configuration file
     ", "", "marker configuratio files (*.xml);;All Files (*)");
1229 
1230         //! was cancel or abort clicked
1231         if (fileName.length() == 0)
1232         {
1233             //! if yes load the standard marker configuration
1234             fileName = "markerStandard.xml";
1235         }
1236 
1237         //! open the selected marker configuration file
1238         FileStorage fs;
1239         fs.open(fileName.toUtf8().constData(), FileStorage::READ);
1240 
1241         //! copy the values to the respective variables
1242         fs["numberMarkers"] >> numberMarkers;
1243 
1244         //! inizialise vectors with correct length depending on the number of markers
1245         list_points3d = std::vector<Point3d>(numberMarkers);
1246         list_points2d = std::vector<Point2d>(numberMarkers);
1247         list_points2dOld = std::vector<Point2d>(numberMarkers);
1248         list_points2dDifference = std::vector<double>(numberMarkers);
1249         list_points2dProjected = std::vector<Point2d>(numberMarkers);
1250         list_points2dUnsorted = std::vector<Point2d>(numberMarkers);
1251 
1252         //! save the marker locations in the points3d vector
1253         fs["list_points3d"] >> list_points3d;
1254         fs.release();
1255         commObj.addLog("Loaded marker configuration from file:");
1256         commObj.addLog(fileName);
1257 
1258     }
1259 
1260     //! Print out the number of markers and their position to the GUI
1261     ss.str("");
1262     ss << "Number of Markers: " << numberMarkers << "\n";
1263     ss << "Marker 3D Points X,Y and Z [mm]: \n";
1264     for (int i = 0; i < numberMarkers; i++)
1265     {
1266         ss << "Marker " << i + 1 << ":\t" << list_points3d[i].x << "\t" << list_points3d[i].y << "\t" <<
     list_points3d[i].z << "\n";
1267     }
1268     commObj.addLog(QString::fromStdString(ss.str()));
1269 
1270     //! check if P3P algorithm can be enabled, it needs exactly 4 marker points to work
1271     if (numberMarkers == 4)
1272     {
1273         //! if P3P is possible, let the user choose which algorithm he wants but keep iterative active
1274         methodPNP = 0;
1275         commObj.enableP3P(true);
1276     }
1277     else
1278     {
1279         //! More (or less) marker than 4 loaded, P3P is not possible, hence user cant select P3P in GUI
1280         methodPNP = 0;
1281         commObj.enableP3P(false);
1282         commObj.addLog("P3P algorithm disabled, only works with 4 markers.");
1283     }
1284 
1285     //! now display the marker configuration in the camera view
1286     Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
1287 
1288     //! Set the camera pose parallel to the marker coordinate system
1289     Tvec.at<double>(0) = 0;
1290     Tvec.at<double>(1) = 0;
1291     Tvec.at<double>(2) = 4500;
1292     Rvec.at<double>(0) = 0 * 3.141592653589 / 180.0;
1293     Rvec.at<double>(1) = 0 * 3.141592653589 / 180.0;
1294     Rvec.at<double>(2) = -90. * 3.141592653589 / 180.0;
1295 
1296     projectPoints(list_points3d, Rvec, Tvec, cameraMatrix,
     distCoeffs, list_points2dProjected);
1297     for (int i = 0; i < numberMarkers; i++)
1298     {
1299         circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 3, Scalar(255, 0, 0
```

```
      ), 3);
1300     }
1301
1302     projectCoordinateFrame(cFrame);
1303     QPixmap QPFrame;
1304     QPFrame = Mat2QPixmap(cFrame);
1305     commObj.changeImage(QPFrame);
1306     QCoreApplication::processEvents();
1307
1308 }
```

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

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.

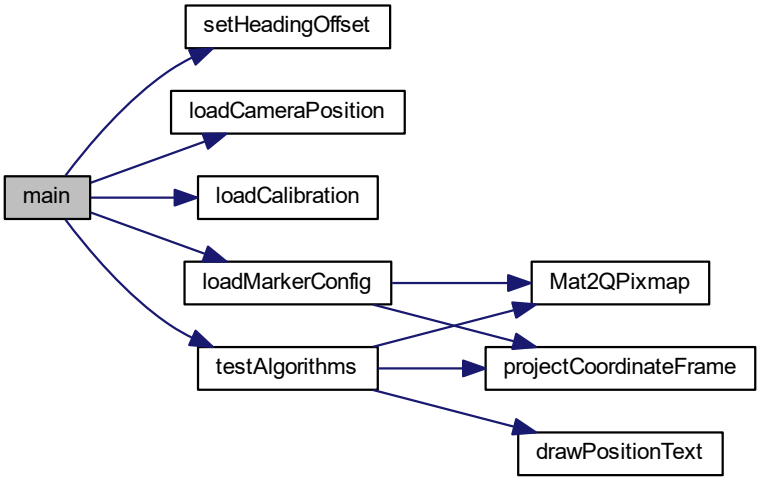
Parameters

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

Definition at line 156 of file main.cpp.

```
157 {
158     QApplication a(argc, argv);
159     RigidTrack w;
160     w.show();   //!< show the GUI
161     //! connect the Qt slots and signals for event handling
162     QObject::connect(&commObj, SIGNAL(statusChanged(QString)), &w, SLOT(setStatus(QString)),
    Qt::DirectConnection);
163     QObject::connect(&commObj, SIGNAL(imageChanged(QPixmap)), &w, SLOT(setImage(QPixmap)),
    Qt::DirectConnection);
164     QObject::connect(&commObj, SIGNAL(logAdded(QString)), &w, SLOT(setLog(QString)),
    Qt::DirectConnection);
165     QObject::connect(&commObj, SIGNAL(logCleared()), &w, SLOT(clearLog(QString)),
    Qt::DirectConnection);
166     QObject::connect(&commObj, SIGNAL(P3Penabled(bool)), &w, SLOT(enableP3P(bool)),
    Qt::DirectConnection);
167     QObject::connect(&commObj, SIGNAL(progressUpdated(int)), &w, SLOT(progressUpdate(int)),
    Qt::DirectConnection);
168
169     commObj.addLog("RigidTrack Version:");
170     commObj.addLog(QString::number(_MSC_FULL_VER));
171     commObj.addLog("Built on:");
172     commObj.addLog(QString(__DATE__));
173
174     //! initial guesses for position and rotation, important for Iterative Method!
175     Tvec.at<double>(0) = 45;
176     Tvec.at<double>(1) = 45;
177     Tvec.at<double>(2) = 4500;
178     Rvec.at<double>(0) = 0 * 3.141592653589 / 180.0;
179     Rvec.at<double>(1) = 0 * 3.141592653589 / 180.0;
180     Rvec.at<double>(2) = -45 * 3.141592653589 / 180.0;
181
182     //! Points that make up the marker CoSy axis system, hence one line in each axis direction
183     coordinateFrame = std::vector<Point3d>(4);
184     coordinateFrameProjected = std::vector<Point2d>(4);
185     coordinateFrame[0] = cv::Point3d(0, 0, 0);
186     coordinateFrame[1] = cv::Point3d(300, 0, 0);
187     coordinateFrame[2] = cv::Point3d(0, 300, 0);
188     coordinateFrame[3] = cv::Point3d(0, 0, 300);
189
190     position[0] = 1.1234;      //!< set position initial values
191     position[1] = 1.2345;      //!< set position initial values
192     position[2] = 1.3456;      //!< set position initial values
193
194     velocity[0] = 0.123;     //!< set velocity initial values
195     velocity[1] = 0.234;     //!< set velocity initial values
196     velocity[2] = 0.345;     //!< set velocity initial values
197
198     eulerAngles[0] = 1.002;  //!< set initial euler angles to arbitrary values for testing
199     eulerAngles[1] = 1.003;  //!< set initial euler angles to arbitrary values for testing
200     eulerAngles[2] = 1.004;  //!< set initial euler angles to arbitrary values for testing
201
202     setHeadingOffset(0.0); //!< set the heading offset to 0
203
204     ss.precision(4); //!< outputs in the log etc are limited to 3 decimal values
205
206     loadCameraPosition(); //!< load the rotation matrix from camera CoSy to ground CoSy
207     loadCalibration(0); //!< load the calibration file with the camera intrinsics
208     loadMarkerConfig(0); //!< load the standard marker configuration
209     testAlgorithms(); //!< test the algorithms and their accuracy
210
211     return a.exec();
212 }
```

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.

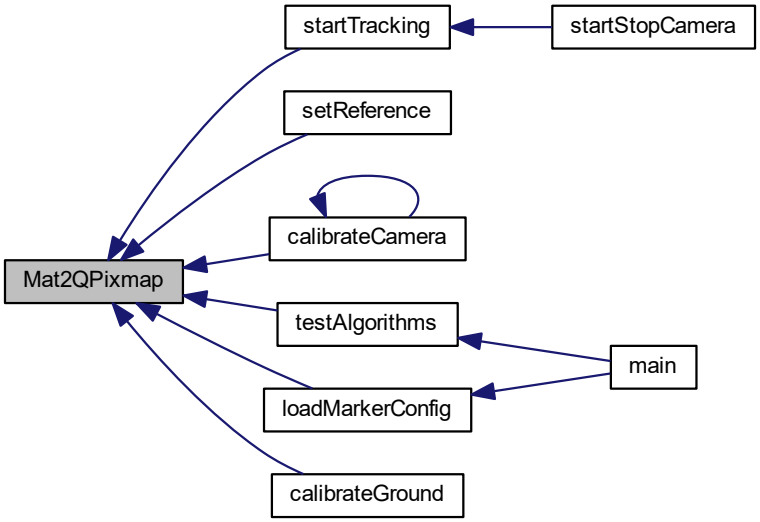
Parameters

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

Definition at line 216 of file main.cpp.

```
217 {
218     QImage dest((const uchar *)src.data, src.cols, src.rows, src.step, QImage::Format_RGB888);
219     dest.bits(); //! enforce deep copy, see documentation
220                  //! of QImage::QImage ( const uchar * data, int width, int height, Format format )
221     QPixmap pixmapDest = QPixmap::fromImage(dest);
222     return pixmapDest;
223 }
```

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.

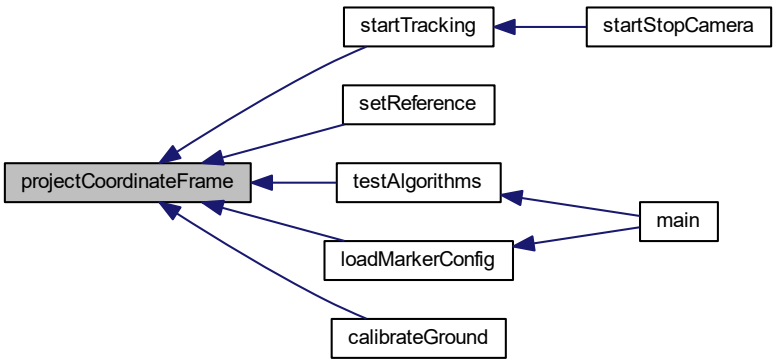
Parameters

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

Definition at line 1081 of file main.cpp.

```
1082 {
1083     projectPoints(coordinateFrame, Rvec, Tvec,
     cameraMatrix, distCoeffs, coordinateFrameProjected);
1084     line(pictureFrame, coordinateFrameProjected[0],
     coordinateFrameProjected[3], Scalar(0, 0, 255), 2); //!<z-axis
1085     line(pictureFrame, coordinateFrameProjected[0],
     coordinateFrameProjected[1], Scalar(255, 0, 0), 2); //!<x-axis
1086     line(pictureFrame, coordinateFrameProjected[0],
     coordinateFrameProjected[2], Scalar(0, 255, 0), 2); //!<y-axis
1087 }
```

Here is the caller graph for this function:



3.1.2.14 sendDataUDP()

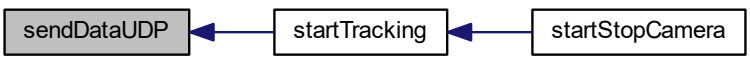
```
void sendDataUDP (
            cv::Vec3d & Position,
            cv::Vec3d & Euler )
```

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 1154 of file main.cpp.

```
1155 {
1156     datagram.clear();
1157     QDataStream out(&datagram, QIODevice::WriteOnly);
1158     out.setVersion(QDataStream::Qt_4_3);
1159     out << (float)Position[0] << (float)Position[1] << (float)Position[2];
1160     out << (float)Euler[0] << (float)Euler[1] << (float)Euler[2]; //! Roll Pitch Heading
1161     udpSocketObject->writeDatagram(datagram,
      IPAdressObject, portObject);
1162
1163     //! if second receiver is activated send it also the tracking data
1164     if (safety2Enable)
1165     {
1166         udpSocketSafety2->writeDatagram(datagram,
      IPAdressSafety2, portSafety2);
1167     }
1168
1169 }
```

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 1122 of file main.cpp.

```
1123 {
1124     headingOffset = d;
1125     d = d * 3.141592653589 / 180.0; //! Convert heading offset from degrees to rad
1126
1127     //! Calculate rotation about x axis
1128     Mat R_x = (Mat_<double>(3, 3) <<
1129         1, 0, 0,
1130         0, 1, 0,
1131         0, 0, 1
1132         );
1133
1134     //! Calculate rotation about y axis
1135     Mat R_y = (Mat_<double>(3, 3) <<
1136         1, 0, 0,
1137         0, 1, 0,
1138         0, 0, 1
1139         );
1140
1141     //! Calculate rotation about z axis
1142     Mat R_z = (Mat_<double>(3, 3) <<
1143         cos(d), -sin(d), 0,
1144         sin(d), cos(d), 0,
1145         0, 0, 1);
1146
1147
1148     //! Combined rotation matrix
1149     M_HeadingOffset = R_z * R_y * R_x;
1150 }
```

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.

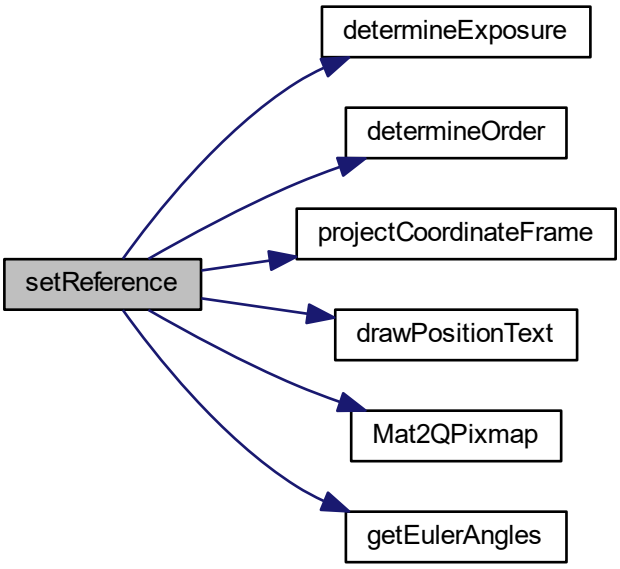
Definition at line 595 of file main.cpp.

```
596 {
597     //! initialize the variables with starting values
598     gotOrder = false;
599     posRef = 0;
600     eulerRef = 0;
601     RmatRef = 0;
602     Rvec = RvecOriginal;
603     Tvec = TvecOriginal;
604 
605     determineExposure();
606 
607     ss.str("");
608     commObj.addLog("Started reference coordinate determination.");
609 
610     CameraLibrary_EnableDevelopment();
611     //! Initialize Camera SDK ==--
612     CameraLibrary::CameraManager::X();
613 
614     //! At this point the Camera SDK is actively looking for all connected cameras and will initialize
615     //! them on it's own.
616 
617     //! Get a connected camera ==================----
618     CameraManager::X().WaitForInitialization();
619     Camera *camera = CameraManager::X().GetCamera();
620 
621     //! If no device connected, pop a message box and exit ==--
622     if (camera == 0)
623     {
624         commObj.addLog("No camera found!");
625         return 1;
626     }
627 
628     //! Determine camera resolution to size application window ==----
629     int cameraWidth = camera->Width();
630     int cameraHeight = camera->Height();
631     camera->GetDistortionModel(distModel);
632     cv::Mat matFrame(cv::Size(cameraWidth, cameraHeight), CV_8UC1);
633 
634     //! Set camera mode to precision mode, it directly provides marker coordinates
635     camera->SetVideoType(Core::PrecisionMode);
636 
637     //! Start camera output ==--
638     camera->Start();
639 
640     //! Turn on some overlay text so it's clear things are     ===---
641     //! working even if there is nothing in the camera's view. ===---
642     //! Set some other parameters as well of the camera
643     camera->SetTextOverlay(true);
644     camera->SetFrameRate(intFrameRate);
645     camera->SetIntensity(intIntensity);
646     camera->SetIRFilter(true);
647     camera->SetContinuousIR(false);
648     camera->SetHighPowerMode(false);
649 
650     //! sample some frames and calculate the position and attitude. then average those values and use that
    as zero position
651     int numberSamples = 0;
652     int numberToSample = 200;
653     double projectionError = 0; //!< difference between the marker points as seen by the camera and the
    projected marker points with Rvec and Tvec
654 
655     while (numberSamples < numberToSample)
656     {
657         //! Fetch a new frame from the camera ===---
658         Frame *frame = camera->GetFrame();
659 
660         if (frame)
661         {
662             //! Ok, we've received a new frame, lets do something
663             //! with it.
664             if (frame->ObjectCount() == numberMarkers)
665             {
666                 //!for(int i=0; i<frame->ObjectCount(); i++)
667                 for (int i = 0; i < numberMarkers; i++)
668                 {
669                     cObject *obj = frame->Object(i);
670                     list_points2dUnsorted[i] = cv::Point2d(obj->X(), obj->Y());
671                 }
672 
673                 if (gotOrder == false)
674                 {
675                     determineOrder();
676                 }
677 
678                 //! sort the 2d points with the correct indices as found in the preceeding order
    determination algorithm
679                 for (int w = 0; w < numberMarkers; w++)
```

```
680                 {
681                     list_points2d[w] = list_points2dUnsorted[
    pointOrderIndices[w]];
682                 }
683                 list_points2dOld = list_points2dUnsorted;
684 
685                 //!Compute the pose from the 3D-2D corresponses
686                 solvePnP(list_points3d, list_points2d,
    cameraMatrix, distCoeffs, Rvec, Tvec, useGuess,
    methodPNP);
687 
688                 //! project the marker 3d points with the solution into the camera image CoSy and calculate
     difference to true camera image
689                 projectPoints(list_points3d, Rvec, Tvec,
    cameraMatrix, distCoeffs, list_points2dProjected);
690                 projectionError = norm(list_points2dProjected,
    list_points2d);
691 
692                 double maxValue = 0;
693                 double minValue = 0;
694                 minMaxLoc(Tvec.at<double>(2), &minValue, &maxValue);
695 
696                 if (maxValue > 10000 || minValue < 0)
697                 {
698                     ss.str("");
699                     ss << "Negative z distance, thats not possible. Start the set zero routine again or
     restart Programm.";
700                     commObj.addLog(QString::fromStdString(ss.str()));
701                     frame->Release();
702                     return 1;
703                 }
704 
705                 if (projectionError > 3)
706                 {
707                     commObj.addLog("Reprojection error is bigger than 3 pixel. Correct marker
     configuration loaded?\nMarker position measured precisely?");
708                     frame->Release();
709                     return 1;
710                 }
711 
712                 if (norm(positionOld) - norm(Tvec) < 0.05)   //!<Iterative Method needs time
     to converge to solution
713                 {
714                     add(posRef, Tvec, posRef);
715                     add(eulerRef, Rvec, eulerRef); //!< That are not the values of yaw,
     roll and pitch yet! Rodriguez has to be called first.
716                     numberSamples++;    //!<  one sample more :D
717                     commObj.progressUpdate(numberSamples * 100 / numberToSample);
718                 }
719                 positionOld = Tvec;
720 
721                 Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
722                 for (int i = 0; i < numberMarkers; i++)
723                 {
724                     circle(cFrame, Point(list_points2d[i].x,
    list_points2d[i].y), 6, Scalar(0, 225, 0), 3);
725                 }
726                 projectCoordinateFrame(cFrame);
727                 projectPoints(list_points3d, Rvec, Tvec,
    cameraMatrix, distCoeffs, list_points2d);
728                 for (int i = 0; i < numberMarkers; i++)
729                 {
730                     circle(cFrame, Point(list_points2d[i].x,
    list_points2d[i].y), 3, Scalar(225, 0, 0), 3);
731                 }
732                 drawPositionText(cFrame, position,
    eulerAngles, projectionError);
733 
734                 QPixmap QPFrame;
735                 QPFrame = Mat2QPixmap(cFrame);
736                 commObj.changeImage(QPFrame);
737                 QCoreApplication::processEvents();
738 
739             }
740             frame->Release();
741         }
742     }
743     //! Release camera ==--
744     camera->Release();
745 
746     //!Divide by the number of samples to get the mean of the reference position
747     divide(posRef, numberToSample, posRef);
748     divide(eulerRef, numberToSample, eulerRef); //!< eulerRef is here in Axis Angle
     notation
749 
750     Rodrigues(eulerRef, RmatRef);               //!< axis angle to rotation matrix
751     //!-- Euler Angles, finally
```

```
752     getEulerAngles(RmatRef, eulerRef); //!<  rotation matrix to euler
753     ss.str("");
754     ss << "RmatRef is:\n";
755     ss << RmatRef << "\n";
756     ss << "Reference Position is:\n";
757     ss << posRef << "[mm] \n";
758     ss << "Reference Euler Angles are:\n";
759     ss << eulerRef << "[deg] \n";
760
761     //! compute the difference between last obtained TVec and the average Value
762     //! When it is large the iterative method has not converged properly so it is advised to start the
        setReference() function once again
763     double error = norm(posRef) - norm(Tvec);
764     if (error > 5.0)
765     {
766         ss << "Caution, distance between reference position and last position is: " << error << "\n Start
        the set zero routine once again.";
767     }
768     commObj.addLog(QString::fromStdString(ss.str()));
769     commObj.progressUpdate(0);
770     return 0;
771 }
```

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 261 of file main.cpp.

```
261                     {
262
263
264     gotOrder = false; //! The order of points, hence which entry in list_points3d corresponds to
        which in list_points2d is not calculated yet
265     Rvec = RvecOriginal; //! Use the value of Rvec that was set in main() as starting value
        for the solvePnP algorithm
266     Tvec = TvecOriginal; //! Use the value of Tvec that was set in main() as starting value
        for the solvePnP algorithm
267     GetLocalTime(&logDate);  //! Get the current date and time to name the log file
268
269     //! Concat the log file name as followed. The file is saved in the folder /logs in the Rigid Track
        installation folder
270     logFileName = "./logs/positionLog_" + QString::number(logDate.wDay) + "_" +
        QString::number(logDate.wMonth) + "_" + QString::number(logDate.wYear);
271     logFileName += "_" + QString::number(logDate.wHour) + "_" + QString::number(
        logDate.wMinute) + "_" + QString::number(logDate.wSecond) + ".txt";
272     logName = logFileName.toStdString(); //! Convert the QString to a standard string
273
274     determineExposure(); //! Get the exposure where the right amount of markers is
        detected
275
276     //! For OptiTrack Ethernet cameras, it's important to enable development mode if you
277     //! want to stop execution for an extended time while debugging without disconnecting
278     //! the Ethernet devices.  Lets do that now:
279
280     CameraLibrary_EnableDevelopment();
281     CameraLibrary::CameraManager::X(); //! Initialize Camera SDK
282
283     //! At this point the Camera SDK is actively looking for all connected cameras and will initialize
284     //! them on it's own
285
286     //! Get a connected camera
287     CameraManager::X().WaitForInitialization();
288     Camera *camera = CameraManager::X().GetCamera();
289
290     //! If no camera can be found, inform user in message log and exit function
291     if (camera == 0)
292     {
293         commObj.addLog("No camera found!");
294         return 1;
295     }
296
297     //! Determine camera resolution to size application window
298     int cameraWidth = camera->Width();
299     int cameraHeight = camera->Height();
300
301     camera->SetVideoType(Core::PrecisionMode);  //! Set the camera mode to precision mode, it used
        greyscale imformation for marker property calculations
302
303     camera->Start(); //! Start camera output
304
305     //! Turn on some overlay text so it's clear things are
306     //! working even if there is nothing in the camera's view
307     camera->SetTextOverlay(true);
308     camera->SetExposure(intExposure);    //! Set the camera exposure
309     camera->SetIntensity(intIntensity); //! Set the camera infrared LED intensity
310     camera->SetFrameRate(intFrameRate); //! Set the camera framerate to 100 Hz
311     camera->SetIRFilter(true);  //! Enable the filter that blocks visible light and only passes infrared
        light
312     camera->SetHighPowerMode(true); //! Enable high power mode of the LEDs
313     camera->SetContinuousIR(false); //! Disable continuous LED light
314     camera->SetThreshold(intThreshold); //! Set threshold for marker detection
315
316     //! Create a new matrix that stores the grayscale picture from the camera
317     Mat matFrame = Mat::zeros(cv::Size(cameraWidth, cameraHeight), CV_8UC1);
318     QPixmap QPFrame; //! QPixmap is the corresponding Qt class that saves images
319     //! Matrix that stores the colored picture, hence marker points, coordinate frame and reprojected
        points
320     Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
321
322     int v = 0;  //! Helper variable used to kick safety switch
323     //! Variables for the min and max values that are needed for sanity checks
324     double maxValue = 0;
325     double minValue = 0;
326     int framesDropped = 0; //! Ff a marker is not visible or accuracy is bad increase this counter
327     double projectionError = 0; //! Equals the quality of the tracking
328
329     setUpUDP(); //! Open sockets and ports for UDP communication
330
331     if (safetyEnable) //! If the safety feature is enabled send the starting message
332     {
333         //! Send enable message, hence send a 9 and then a 1
334         data.setNum((int)(9));
335         udpSocketSafety->write(data);
336         data.setNum((int)(1));
337         udpSocketSafety->write(data);
```

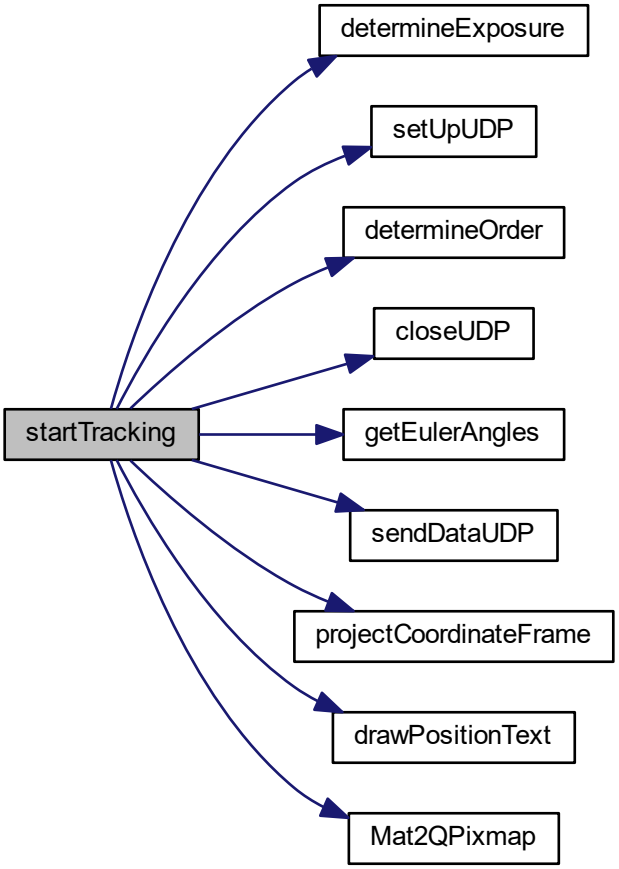
```
338     }
339
340     //! Fetch a new frame from the camera
341     bool gotTime = false; //! Get the timestamp of the first frame. This time is subtracted from every
      subseeding frame so the time starts at 0 in the logs
342     while (!gotTime) //! While no new frame is received loop
343     {
344         Frame *frame = camera->GetFrame(); //! Get a new camera frame
345         if (frame)  //! There is actually a new frame
346         {
347             timeFirstFrame = frame->TimeStamp(); //! Get the time stamp for the first frame.
      It is subtracted for the following frames
348             frame->Release();   //! Release the frame so the camera can continue
349             gotTime = true; //! Exit the while loop
350         }
351     }
352
353     //! Now enter the main loop that processes each frame and computes the pose, sends it and logs stuff
354     while (!exitRequested) //! Check if the user has not pressed "Stop Tracking" yet
355     {
356
357         Frame *frame = camera->GetFrame(); //! Fetch a new frame from the camera
358
359         if (frame) //! Did we got a new frame or does the camera still need more time
360         {
361             framesDropped++; //! Increase by one, if everything is okay it is decreased at the end of the
      loop again
362
363             //! Only use this frame it the right number of markers is found in the picture
364             if (frame->ObjectCount() == numberMarkers)
365             {
366                 //! Get the marker points in 2D in the camera image frame and store them in the
      list_points2dUnsorted vector
367                 //! The order of points that come from the camera corresponds to the Y coordinate
368                 for (int i = 0; i < numberMarkers; i++)
369                 {
370                     cObject *obj = frame->Object(i);
371                     list_points2dUnsorted[i] = cv::Point2d(obj->X(), obj->Y());
372                 }
373
374                 if (gotOrder == false) //! Was the order already determined? This is false for the
      first frame and from then on true
375                 {
376                     determineOrder(); //! Now compute the order
377                 }
378
379                 //! Sort the 2d points with the correct indices as found in the preceeding order
      determination algorithm
380                 for (int w = 0; w < numberMarkers; w++)
381                 {
382                     list_points2d[w] = list_points2dUnsorted[
    pointOrderIndices[w]]; //! pointOrderIndices was calculated in determineOrder()
383                 }
384                 list_points2dOld = list_points2dUnsorted;
385
386                 //! The first time the 2D-3D corresspondence was determined with gotOrder was okay.
387                 //! But this order can change as the object moves and the marker objects appear in a
388                 //! different order in the frame->Object() array.
389                 //! The solution is that: When a marker point (in the camera image, hence in 2D) was at
390                 //! a position then it wont move that much from one frame to the other.
391                 //! So for the new frame we take a marker object and check which marker was closest this
      point
392                 //! in the old image frame? This is probably the same (true) marker. And we do that for
      every other marker as well.
393                 //! When tracking is good and no frames are dropped because of missing markers this should
      work every frame.
394                 for (int j = 0; j < numberMarkers; j++)
395                 {
396                     minPointDistance = 5000; //! The sum of point distances is set to
      something unrealistic large
397                     for (int k = 0; k < numberMarkers; k++)
398                     {
399                         //! Calculate N_2 norm of unsorted points minus old points
400                         currentPointDistance = norm(
    list_points2dUnsorted[pointOrderIndices[j]] -
    list_points2dOld[k]);
401                         //! If the norm is smaller than minPointDistance the correspondence is more likely
      to be correct
402                         if (currentPointDistance <
    minPointDistance)
403                         {
404                             //! Update the array that saves the new point order
405                             minPointDistance =
    currentPointDistance;
406                             pointOrderIndicesNew[j] = k;
407                         }
408                     }
```

```
409                 }
410
411                 //! Now the new order is found, set the point order to the new value
412                 for (int k = 0; k < numberMarkers; k++)
413                 {
414                     pointOrderIndices[k] = pointOrderIndicesNew[k];
415                     list_points2d[k] = list_points2dUnsorted[
    pointOrderIndices[k]];
416                 }
417
418                 //! Save the unsorted position of the marker points for the next loop
419                 list_points2dOld = list_points2dUnsorted;
420
421                 //!Compute the object pose from the 3D-2D corresponses
422                 solvePnP(list_points3d, list_points2d,
    cameraMatrix, distCoeffs, Rvec, Tvec, useGuess,
    methodPNP);
423
424                 //! Project the marker 3d points with the solution into the camera image CoSy and calculate
     difference to true camera image
425                 projectPoints(list_points3d, Rvec, Tvec,
    cameraMatrix, distCoeffs, list_points2dProjected);
426                 projectionError = norm(list_points2dProjected,
    list_points2d); //! Difference of true pose and found pose
427
428                 //! Increase the framesDropped variable if accuracy of tracking is too bad
429                 if (projectionError > 5)
430                 {
431                     framesDropped++;
432                 }
433                 else
434                 {
435                     framesDropped = 0;  //! Set number of subsequent frames dropped to zero because error
     is small enough and no marker was missing
436                 }
437
438                 //! Get the min and max values from TVec for sanity check
439                 minMaxLoc(Tvec.at<double>(2), &minValue, &maxValue);
440
441                 //! Sanity check of values. negative z means the marker CoSy is behind the camera, that's
     not possible.
442                 if (minValue < 0)
443                 {
444                     commObj.addLog("Negative z distance, that is not possible. Start the set zero
     routine again or restart Program.");
445                     frame->Release(); //! Release the frame so the camera can move on
446                     camera->Release(); //! Release the camera
447                     closeUDP(); //! Close all UDP connections so the programm can be closed later
     on and no resources are locked
448                     return 1; //! Exit the function
449                 }
450
451                 //! Next step is the transformation from camera CoSy to navigation CoSy
452                 //! Compute the relative object position from the reference position to the current one
453                 //! given in the camera CoSy: \f$ T_C^{NM} = Tvec - Tvec_{Ref} \f$
454                 subtract(Tvec, posRef, position);
455
456                 //! Transform the position from the camera CoSy to the navigation CoSy with INS alligned
     heading and convert from [mm] to [m]
457                 //! \f$ T_N^{NM} = M_{NC} \times T_C^{NM} \f$
458                 Mat V = 0.001 * M_HeadingOffset * M_CN.t() * (Mat)
    position;
459                 position = V;   //! Position is the result of the preceeding calculation
460                 position[2] *= invertZ;  //! Invert Z if check box in GUI is activated,
     hence height above ground is considered
461
462                 //! Realtive angle between reference orientation and current orientation
463                 Rodrigues(Rvec, Rmat);  //! Convert axis angle respresentation to ordinary rotation
     matrix
464
465                 //! The difference of the reference rotation and the current rotation
466                 //! \f$ R_{ NM } = M_{ NC } \times R_{ CM } \f$
467                 Rmat = RmatRef.t() *Rmat;
468
469                 //! Euler Angles, finally
470                 getEulerAngles(Rmat, eulerAngles); //! Get the euler angles
     from the rotation matrix
471                 eulerAngles[2] += headingOffset; //! Add the heading offset to the
     heading angle
472
473                 //! 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
474                 frameTime = frame->TimeStamp() - timeOld;   //! Time between the old frame
     and the current frame
475                 timeOld = frame->TimeStamp();    //! Set the old frame time to the current  one
476                 velocity[0] = (position[0] - positionOld[0]) /
    frameTime; //! Calculate the x velocity with finite differences
```

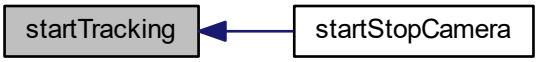
```
477                 velocity[1] = (position[1] - positionOld[1]) /
    frameTime; //! Calculate the y velocity with finite differences
478                 velocity[2] = (position[2] - positionOld[2]) /
    frameTime; //! Calculate the z velocity with finite differences
479                 positionOld = position;  //! Set the old position to the current one for
     next frame velocity calcuation
480
481                 //! Send position and Euler angles over WiFi with 100 Hz
482                 sendDataUDP(position, eulerAngles);
483
484                 //! Save the values in a log file, values are:
485                 //! Time sinc tracking started  Position    Euler Angles    Velocity
486                 logfile.open(logName, std::ios::app); //! Open the log file, the folder is
     RigidTrackInstallationFolder/logs
487                 logfile << frame->TimeStamp() - timeFirstFrame << ";" <<
    position[0] << ";" << position[1] << ";" << position[2] << ";";
488                 logfile << eulerAngles[0] << ";" <<
    eulerAngles[1] << ";" << eulerAngles[2] << ";";
489                 logfile << velocity[0] << ";" << velocity[1] << ";" <<
    velocity[2] << "\n";
490                 logfile.close(); //! Close the file to save values
491             }
492
493             //! Check if the position and euler angles are below the allowed value, if yes send OKAY signal
     (1), if not send shutdown signal (0)
494             //! Absolute x, y and z position in navigation CoSy must be smaller than the allowed distance
495             if (safetyEnable)
496             {
497                 if ((abs(position[0]) < safetyBoxLength && abs(position[1]) <
    safetyBoxLength && abs(position[2]) < safetyBoxLength))
498                 {
499                     //! Absolute Euler angles must be smaller than allowed value. Heading is not considered

500                     if ((abs(eulerAngles[0]) < safetyAngle && abs(eulerAngles[1]) <
    safetyAngle))
501                     {
502                         //! Send the OKAY signal to the desired computer every 5th time
503                         if (v == 5) {
504                             data.setNum((int)(1));
505                             udpSocketSafety->write(data); //! Send the 1
506                             v = 0; //! reset the counter that is needed for decimation to every 5th time
     step
507                         }
508                     }
509                     //! The euler angles of the object exceeded the allowed euler angles, send the shutdown
     signal (0)
510                     else
511                     {
512                         data.setNum((int)(0));  //! Send the shutdown signal, a 0
513                         udpSocketSafety->write(data);
514                         commObj.addLog("Object exceeded allowed Euler angles, shutdown signal sent."
    ); //! Inform the user
515
516                     }
517                 }
518                 //! The position of the object exceeded the allowed position, shut the object down
519                 else
520                 {
521                     data.setNum((int)(0));  //! Send the shutdown signal, a 0
522                     udpSocketSafety->write(data);
523                     commObj.addLog("Object left allowed area, shutdown signal sent."); //! Inform
     the user
524
525                 }
526             }
527
528             //! Inform the user if tracking system is disturbed (marker lost or so) or error was too big
529             if (framesDropped > 10)
530             {
531                 if (safetyEnable) //! Also send the shutdown signal
532                 {
533                     data.setNum((int)(0));  //! Send the shutdown signal, a 0
534                     udpSocketSafety->write(data);
535                 }
536                 commObj.addLog("Lost marker points or precision was bad!"); //! Inform the user
537                 framesDropped = 0;
538             }
539
540             //! Rasterize the frame so it can be shown in the GUI
541             frame->Rasterize(cameraWidth, cameraHeight, matFrame.step,
    BACKBUFFER_BITSPERPIXEL, matFrame.data);
542
543             //! Convert the frame from greyscale as it comes from the camera to rgb color
544             cvtColor(matFrame, cFrame, COLOR_GRAY2RGB);
545
546             //! Project (draw) the marker CoSy origin into 2D and save it in the cFrame image
547             projectCoordinateFrame(cFrame);
```

```
548
549             //! Project the marker points from 3D to the camera image frame (2d) with the computed pose
550             projectPoints(list_points3d, Rvec, Tvec,
    cameraMatrix, distCoeffs, list_points2d);
551             for (int i = 0; i < numberMarkers; i++)
552             {
553                 //! Draw a circle around the projected points so the result can be better compared to the
     real marker position
554                 //! In the resulting picture those are the red dots
555                 circle(cFrame, Point(list_points2d[i].x,
    list_points2d[i].y), 3, Scalar(225, 0, 0), 3);
556             }
557
558             //! Write the current position, attitude and error values as text in the frame
559             drawPositionText(cFrame, position, eulerAngles, projectionError);
560
561             //! Send the new camera picture to the GUI and call the GUI processing routine
562             QPixmap QPFrame;
563             QPFrame = Mat2QPixmap(cFrame);
564             commObj.changeImage(QPFrame); //! Update the picture in the GUI
565             QCoreApplication::processEvents(); //! Give Qt time to handle everything
566
567             //! Release the camera frame to fetch the new one
568             frame->Release();
569         }
570     }
571
572     //! User choose to stop the tracking, clean things up
573     closeUDP(); //! Close the UDP connections so resources are deallocated
574     camera->Release();  //! Release camera
575     return 0;
576 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



3.1.2.18 testAlgorithms()

```
void testAlgorithms ( )
```

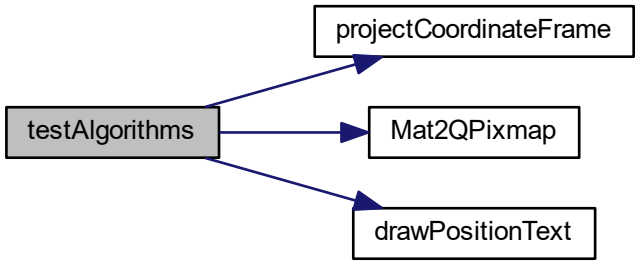
Project some points from 3D to 2D and then check the accuracy of the algorithms. Mainly to generate something that can be shown in the camera view so the user knows everything loaded correctly.

Definition at line 952 of file main.cpp.

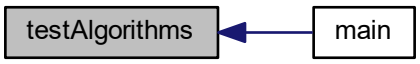
```
953 {
954
955     int _methodPNP;
956
957     std::vector<Point2d> noise(numberMarkers);
958
959     RvecOriginal = Rvec;
960     TvecOriginal = Tvec;
961
962     projectPoints(list_points3d, Rvec, Tvec, cameraMatrix,
    distCoeffs, list_points2dProjected);
963
964     ss.str("");
965     ss << "Unsorted Points 2D Projected \n";
966     ss << list_points2dProjected << "\n";
967     commObj.addLog(QString::fromStdString(ss.str()));
968
969     Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
970     for (int i = 0; i < numberMarkers; i++)
971     {
972         circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 6, Scalar(0, 255, 0
    ), 3);
973     }
974
975     projectCoordinateFrame(cFrame);
976
977     ss.str("");
978     ss << "=====================================================\n";
979     ss << "================= Projected Points ==================\n";
980     ss << list_points2dProjected << "\n";
981
982     randn(noise, 0, 0.5);
983     add(list_points2dProjected, noise, list_points2dProjected);
984
985     ss << "================ With Noise Points ==================\n";
986     ss << list_points2dProjected << "\n";
987     commObj.addLog(QString::fromStdString(ss.str()));
988
989
990     bool useGuess = true;
991     _methodPNP = 0; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP  //!< not used
992
993     solvePnP(list_points3d, list_points2dProjected, cameraMatrix,
    distCoeffs, Rvec, Tvec, useGuess, _methodPNP);
994
995     ss.str("");
996     ss << "=====================================================\n";
997     ss << "==================== Iterative ======================\n";
998     ss << "rvec: " << "\n";
999     ss << Rvec << "\n";
1000     ss << "tvec: " << "\n";
1001     ss << Tvec << "\n";
1002
1003     commObj.addLog(QString::fromStdString(ss.str()));
1004
1005     _methodPNP = 1; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP UPnP not used
1006     Rvec = cv::Mat::zeros(3, 1, CV_64F);
1007     Tvec = cv::Mat::zeros(3, 1, CV_64F);
1008     solvePnP(list_points3d, list_points2dProjected, cameraMatrix,
    distCoeffs, Rvec, Tvec, useGuess, _methodPNP);
1009
1010     ss.str("");
1011     ss << "=====================================================\n";
1012     ss << "====================    EPNP    ====================\n";
1013     ss << "rvec: " << "\n";
1014     ss << Rvec << "\n";
1015     ss << "tvec: " << "\n";
1016     ss << Tvec << "\n";
1017
1018     projectPoints(list_points3d, Rvec, Tvec, cameraMatrix,
    distCoeffs, list_points2dProjected);
1019     for (int i = 0; i < numberMarkers; i++)
1020     {
1021         circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 3, Scalar(255, 0, 0
    ), 3);
1022     }
1023     QPixmap QPFrame;
1024     QPFrame = Mat2QPixmap(cFrame);
1025     commObj.changeImage(QPFrame);
```

```
1026     QCoreApplication::processEvents();
1027     commObj.addLog(QString::fromStdString(ss.str()));
1028     if (numberMarkers == 4)
1029     {
1030         _methodPNP = 2; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP  //!< not used
1031         Rvec = cv::Mat::zeros(3, 1, CV_64F);
1032         Tvec = cv::Mat::zeros(3, 1, CV_64F);
1033         solvePnP(list_points3d, list_points2dProjected,
     cameraMatrix, distCoeffs, Rvec, Tvec, useGuess, _methodPNP);
1034
1035         ss.str("");
1036         ss << "=====================================================\n";
1037         ss << "====================     P3P    ====================\n";
1038         ss << "rvec: " << "\n";
1039         ss << Rvec << "\n";
1040         ss << "tvec: " << "\n";
1041         ss << Tvec << "\n";
1042
1043         projectPoints(list_points3d, Rvec, Tvec, cameraMatrix,
     distCoeffs, list_points2dProjected);
1044         for (int i = 0; i < numberMarkers; i++)
1045         {
1046             circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 3, Scalar(255,
     0, 0), 3);
1047         }
1048         double projectionError = norm(list_points2dProjected, list_points2d);
1049         putText(cFrame, "Testing Algorithms Finished", cv::Point(5, 420), 1, 1, cv::Scalar(255, 255, 255));
1050         drawPositionText(cFrame, position, eulerAngles, projectionError)
     ;
1051
1052         QPixmap QPFrame;
1053         QPFrame = Mat2QPixmap(cFrame);
1054         commObj.changeImage(QPFrame);
1055         QCoreApplication::processEvents();
1056         commObj.addLog(QString::fromStdString(ss.str()));
1057     }
1058
1059     _methodPNP = 4; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP  //!< not used
1060     Rvec = cv::Mat::zeros(3, 1, CV_64F);
1061     Tvec = cv::Mat::zeros(3, 1, CV_64F);
1062     solvePnP(list_points3d, list_points2dProjected, cameraMatrix,
     distCoeffs, Rvec, Tvec, useGuess, _methodPNP);
1063
1064     ss.str("");
1065     ss << "=====================================================\n";
1066     ss << "====================    UPNP    ====================\n";
1067     ss << "rvec: " << "\n";
1068     ss << Rvec << "\n";
1069     ss << "tvec: " << "\n";
1070     ss << Tvec << "\n";
1071
1072     commObj.addLog(QString::fromStdString(ss.str()));
1073
1074     Rvec = RvecOriginal;
1075     Tvec = TvecOriginal;
1076
1077 }
```

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.

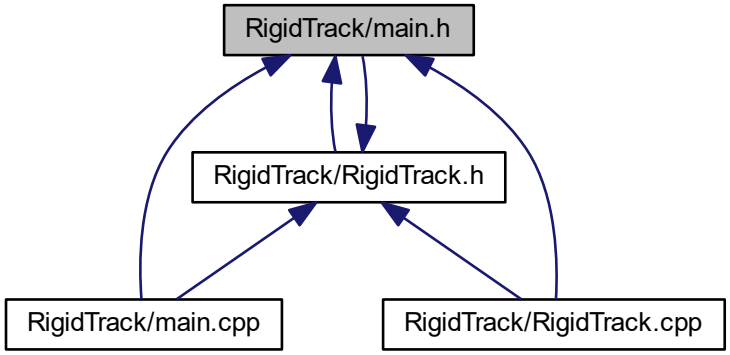
3.2 RigidTrack/main.h File Reference

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.

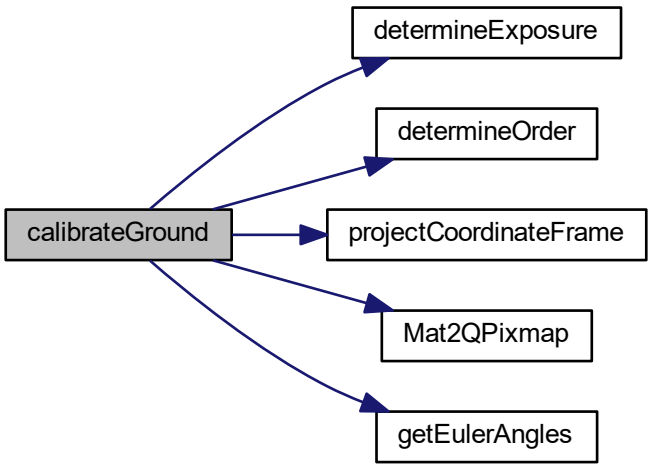
Definition at line 1563 of file main.cpp.

```
1564 {
1565     //! initialize the variables with starting values
1566     gotOrder = false;
1567     posRef = 0;
1568     eulerRef = 0;
1569     RmatRef = 0;
1570     Rvec = RvecOriginal;
1571     Tvec = TvecOriginal;
1572
1573     determineExposure();
1574
1575     ss.str("");
1576     commObj.addLog("Started ground calibration");
1577
1578     CameraLibrary_EnableDevelopment();
1579     //! Initialize Camera SDK ===-
1580     CameraLibrary::CameraManager::X();
1581
1582     //! At this point the Camera SDK is actively looking for all connected cameras and will initialize
1583     //! them on it's own.
1584
1585     //! Get a connected camera ====================
1586     CameraManager::X().WaitForInitialization();
1587     Camera *camera = CameraManager::X().GetCamera();
1588
1589     //! If no device connected, pop a message box and exit ===-
1590     if (camera == 0)
1591     {
1592         commObj.addLog("No camera found!");
1593         return 1;
1594     }
1595
1596     //! Determine camera resolution to size application window ==----
1597     int cameraWidth = camera->Width();
1598     int cameraHeight = camera->Height();
1599     camera->GetDistortionModel(distModel);
1600     cv::Mat matFrame(cv::Size(cameraWidth, cameraHeight), CV_8UC1);
1601
1602     //! Set camera mode to precision mode, it directly provides marker coordinates
1603     camera->SetVideoType(Core::PrecisionMode);
1604
1605     //! Start camera output ==--
1606     camera->Start();
1607
1608     //! Turn on some overlay text so it's clear things are     ===---
1609     //! working even if there is nothing in the camera's view. ======-
1610     //! Set some other parameters as well of the camera
1611     camera->SetTextOverlay(true);
1612     camera->SetFrameRate(intFrameRate);
1613     camera->SetIntensity(intIntensity);
1614     camera->SetIRFilter(true);
1615     camera->SetContinuousIR(false);
1616     camera->SetHighPowerMode(false);
1617
1618     //! sample some frames and calculate the position and attitude. then average those values and use that
      as zero position
1619     int numberSamples = 0;
1620     int numberToSample = 200;
1621     double projectionError = 0;
1622
1623     while (numberSamples < numberToSample)
1624     {
1625         //! Fetch a new frame from the camera ===---
1626         Frame *frame = camera->GetFrame();
1627
1628         if (frame)
1629         {
1630             //! Ok, we've received a new frame, lets do something
1631             //! with it.
1632             if (frame->ObjectCount() == numberMarkers)
1633             {
```

```
1634                //!for(int i=0; i<frame->ObjectCount(); i++)
1635                for (int i = 0; i < numberMarkers; i++)
1636                {
1637                    cObject *obj = frame->Object(i);
1638                    list_points2dUnsorted[i] = cv::Point2d(obj->X(), obj->Y());
1639                }
1640
1641                if (gotOrder == false)
1642                {
1643                    determineOrder();
1644                }
1645
1646                //! sort the 2d points with the correct indices as found in the preceeding order
     determination algorithm
1647                for (int w = 0; w < numberMarkers; w++)
1648                {
1649                    list_points2d[w] = list_points2dUnsorted[
    pointOrderIndices[w]];
1650                }
1651                list_points2dOld = list_points2dUnsorted;
1652
1653                //!Compute the pose from the 3D-2D corresponses
1654                solvePnP(list_points3d, list_points2d,
    cameraMatrix, distCoeffs, Rvec, Tvec, useGuess,
    methodPNP);
1655
1656                //! project the marker 3d points with the solution into the camera image CoSy and calculate
     difference to true camera image
1657                projectPoints(list_points3d, Rvec, Tvec,
    cameraMatrix, distCoeffs, list_points2dProjected);
1658                projectionError = norm(list_points2dProjected,
    list_points2d);
1659
1660                if (projectionError > 3)
1661                {
1662                    commObj.addLog("Reprojection error is bigger than 3 pixel. Correct marker
     configuration loaded?\nMarker position measured precisely?");
1663                    frame->Release();
1664                    return 1;
1665                }
1666
1667                double maxValue = 0;
1668                double minValue = 0;
1669                minMaxLoc(Tvec.at<double>(2), &minValue, &maxValue);
1670
1671                if (maxValue > 10000 || minValue < 0)
1672                {
1673
1674
1675                    commObj.addLog("Negative z distance, thats not possible. Start the set zero
     routine again and check marker configurations.");
1676                    frame->Release();
1677                    return 1;
1678                }
1679
1680                if (norm(positionOld) - norm(Tvec) < 0.05)   //!<Iterative Method needs time
     to converge to solution
1681                {
1682                    add(posRef, Tvec, posRef);
1683                    add(eulerRef, Rvec, eulerRef); //!< That are not the values of yaw,
     roll and pitch yet! Rodriguez has to be called first.
1684                    numberSamples++;    //!<-- one sample more :D
1685                    commObj.progressUpdate(numberSamples * 100 / numberToSample);
1686                }
1687                positionOld = Tvec;
1688
1689                Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
1690                for (int i = 0; i < numberMarkers; i++)
1691                {
1692                    circle(cFrame, Point(list_points2d[i].x,
    list_points2d[i].y), 6, Scalar(0, 225, 0), 3);
1693                }
1694                projectCoordinateFrame(cFrame);
1695                projectPoints(list_points3d, Rvec, Tvec,
    cameraMatrix, distCoeffs, list_points2d);
1696                for (int i = 0; i < numberMarkers; i++)
1697                {
1698                    circle(cFrame, Point(list_points2d[i].x,
    list_points2d[i].y), 3, Scalar(225, 0, 0), 3);
1699                }
1700
1701                QPixmap QPFrame;
1702                QPFrame = Mat2QPixmap(cFrame);
1703                commObj.changeImage(QPFrame);
1704                QCoreApplication::processEvents();
1705
1706            }
```

```
1707             frame->Release();
1708         }
1709     }
1710     //! Release camera ==--
1711     camera->Release();
1712
1713     //!Divide by the number of samples to get the mean of the reference position
1714     divide(posRef, numberToSample, posRef);
1715     divide(eulerRef, numberToSample, eulerRef); //!< eulerRef is here in Axis Angle
       notation
1716
1717     Rodrigues(eulerRef, RmatRef);                //!< axis angle to rotation matrix
1718
1719     getEulerAngles(RmatRef, eulerRef); //!<  rotation matrix to euler
1720     ss.str("");
1721     ss << "RmatRef is:\n";
1722     ss << RmatRef << "\n";
1723     ss << "Reference Position is:\n";
1724     ss << posRef << "[mm] \n";
1725     ss << "Reference Euler angles are:\n";
1726     ss << eulerRef << "[deg] \n";
1727
1728     //! Save the obtained calibration coefficients in a file for later use
1729     QString fileName = QFileDialog::getSaveFileName(nullptr, "Save ground calibration file", "
      referenceData.xml", "Calibration File (*.xml);;All Files (*)");
1730     FileStorage fs(fileName.toUtf8().constData(), FileStorage::WRITE);
1731     fs << "M_NC" << RmatRef;
1732     fs << "eulerRef" << eulerRef;
1733     strBuf = fs.releaseAndGetString();
1734     commObj.changeStatus(QString::fromStdString(strBuf));
1735     commObj.addLog("Saved ground calibration!");
1736     commObj.progressUpdate(0);
1737     return 0;
1738 }
```

Here is the call graph for this function:



3.2.2.2 closeUDP()

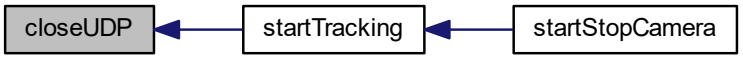
```
void closeUDP ( )
```

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 1173 of file main.cpp.

```
1174 {
1175     //! check if the socket is open and if yes close it
1176     if (udpSocketObject->isOpen())
1177     {
1178         udpSocketObject->close();
1179     }
1180
1181     if (udpSocketSafety->isOpen())
1182     {
1183         udpSocketSafety->close();
1184     }
1185
1186     if (udpSocketSafety2->isOpen())
1187     {
1188         udpSocketSafety2->close();
1189     }
1190     commObj.addLog("Closed all UDP ports.");
1191 }
```

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.

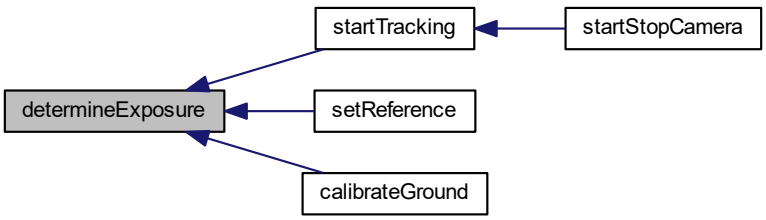
Definition at line 1362 of file main.cpp.

```
1363 {
1364     //! For OptiTrack Ethernet cameras, it's important to enable development mode if you
1365     //! want to stop execution for an extended time while debugging without disconnecting
1366     //! the Ethernet devices.  Lets do that now:
1367
1368     CameraLibrary_EnableDevelopment();
1369
1370     //! Initialize Camera SDK ===-
1371     CameraLibrary::CameraManager::X();
1372
1373     //! At this point the Camera SDK is actively looking for all connected cameras and will initialize
1374     //! them on it's own.
1375
1376     //! Get a connected camera ==================---
1377     CameraManager::X().WaitForInitialization();
1378     Camera *camera = CameraManager::X().GetCamera();
1379
1380     //! If no device connected, pop a message box and exit ==--
1381     if (camera == 0)
```

```
1382     {
1383         commObj.addLog("No camera found!");
1384         return 1;
1385     }
1386 
1387     //! Determine camera resolution to size application window ==----
1388     int cameraWidth = camera->Width();
1389     int cameraHeight = camera->Height();
1390 
1391     camera->SetVideoType(Core::PrecisionMode);  //! set the camera mode to precision mode, it used
     greyscale imformation for marker property calculations
1392 
1393                                                 //! Start camera output ==--
1394     camera->Start();
1395 
1396     //! Turn on some overlay text so it's clear things are     ======
1397     //! working even if there is nothing in the camera's view. ======
1398     camera->SetTextOverlay(true);
1399     camera->SetExposure(intExposure);    //! set the camera exposure
1400     camera->SetIntensity(intIntensity); //! set the camera infrared LED intensity
1401     camera->SetFrameRate(intFrameRate); //! set the camera framerate to 100 Hz
1402     camera->SetIRFilter(true);  //! enable the filter that blocks visible light and only passes infrared
     light
1403     camera->SetHighPowerMode(true); //! enable high power mode of the leds
1404     camera->SetContinuousIR(false); //! enable continuous LED light
1405     camera->SetThreshold(intThreshold); //! set threshold for marker detection
1406 
1407     //!set exposure such that num markers are visible
1408     int numberObjects = 0;  //! Number of objects (markers) found in the current picture with the given
     exposure
1409     int minExposure = 1;    //! exposure when objects detected the first time is numberMarkers
1410     int maxExposure = 480;  //! exposure when objects detected is first time numberMarkers+1
1411     intExposure = minExposure;   //! set the exposure to the smallest value possible
1412     int numberTries = 0;    //! if the markers arent found after numberTries then there might be no markers
     at all in the real world
1413 
1414                             //! Determine minimum exposure, hence when are numberMarkers objects detected
1415     camera->SetExposure(intExposure);
1416     while (numberObjects != numberMarkers && numberTries < 48)
1417     {
1418         //! get a new camera frame
1419         Frame *frame = camera->GetFrame();
1420         if (frame) //! frame received
1421         {
1422             numberObjects = frame->ObjectCount();   //! how many objects are detected in the image
1423             if (numberObjects == numberMarkers) { minExposure =
    intExposure; frame->Release(); break; } //! if the right amount if markers is found, exit while
     loop
1424             //! not the right amount of markers was found so increase the exposure and try again
1425             numberTries++;
1426             intExposure += 10;
1427             camera->SetExposure(intExposure);
1428             ss.str("");
1429             ss << "Exposure: " << intExposure << "\t";
1430             ss << "Objects found: " << numberObjects;
1431             commObj.addLog(QString::fromStdString(ss.str()));
1432             frame->Release();
1433         }
1434     }
1435 
1436     //! Now determine maximum exposure, hence when are numberMarkers+1 objects detected
1437     numberTries = 0;    //! if the markers arent found after numberTries then there might be no markers at
     all in the real world
1438     intExposure = maxExposure;
1439     camera->SetExposure(intExposure);
1440     numberObjects = 0;
1441     while (numberObjects != numberMarkers && numberTries < 48)
1442     {
1443         Frame *frame = camera->GetFrame();
1444         if (frame)
1445         {
1446             numberObjects = frame->ObjectCount(); //! how many objects are detected in the image
1447             if (numberObjects == numberMarkers) { maxExposure =
    intExposure; frame->Release(); break; } //! if the right amount if markers is found, exit while
     loop
1448 
1449             //! not the right amount of markers was found so decrease the exposure and try again
1450             intExposure -= 10;
1451             numberTries++;
1452             camera->SetExposure(intExposure);
1453             ss.str("");
1454             ss << "Exposure: " << intExposure << "\t";
1455             ss << "Objects found: " << numberObjects;
1456             commObj.addLog(QString::fromStdString(ss.str()));
1457             frame->Release();
1458         }
1459     }
```

```
1460
1461     //! set the exposure to the mean of min and max exposure determined
1462     camera->SetExposure((minExposure + maxExposure) / 2.0);
1463
1464     //! and now check if the correct amount of markers is detected with that new value
1465     while (1)
1466     {
1467         Frame *frame = camera->GetFrame();
1468         if (frame)
1469         {
1470             numberObjects = frame->ObjectCount(); //! how many objects are detected in the image
1471             if (numberObjects != numberMarkers) //! are all markers and not more or less
     detected in the image
1472             {
1473                 frame->Release();
1474                 commObj.addLog("Was not able to detect the right amount of markers.");
1475                 //! Release camera ===-
1476                 camera->Release();
1477                 return 1;
1478             }
1479             else  //! all markers and not more or less are found
1480             {
1481                 frame->Release();
1482                 intExposure = (minExposure + maxExposure) / 2.0;
1483                 commObj.addLog("Found the correct number of markers.");
1484                 commObj.addLog("Exposure set to:");
1485                 commObj.addLog(QString::number(intExposure));
1486                 break;
1487             }
1488         }
1489     }
1490
1491     camera->Release();
1492     return 0;
1493
1494 }
```

Here is the caller graph for this function:



3.2.2.4 determineOrder()

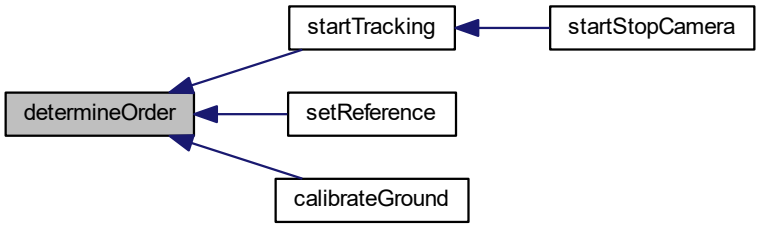
```
void determineOrder ( )
```

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 1498 of file main.cpp.

```
1499 {
1500     //! determine the 3D-2D correspondences that are crucial for the PnP algorithm
1501     //! Try every possible correspondence and solve PnP
1502     //! Then project the 3D marker points into the 2D camera image and check the difference
1503     //! between projected points and points as seen by the camera
1504     //! the corresponce with the smallest difference is probably the correct one
1505 
1506         //! the difference between true 2D points and projected points is super big
1507     minPointDistance = 5000;
1508     std::sort(pointOrderIndices, pointOrderIndices + 4);
1509 
1510     //! now try every possible permutation of correspondence
1511     do {
1512         //! reset the starting values for solvePnP
1513         Rvec = RvecOriginal;
1514         Tvec = TvecOriginal;
1515 
1516         //! sort the 2d points with the current permutation
1517         for (int m = 0; m < numberMarkers; m++)
1518         {
1519             list_points2d[m] = list_points2dUnsorted[
    pointOrderIndices[m]];
1520         }
1521 
1522         //! Call solve PNP with P3P since its more robust and sufficient for start value determination
1523         solvePnP(list_points3d, list_points2d,
    cameraMatrix, distCoeffs, Rvec, Tvec, useGuess, SOLVEPNP_P3P);
1524 
1525         //! set the current difference of all point correspondences to zero
1526         currentPointDistance = 0;
1527 
1528         //! project the 3D points with the solvePnP solution onto 2D
1529         projectPoints(list_points3d, Rvec, Tvec,
    cameraMatrix, distCoeffs, list_points2dProjected);
1530 
1531         //! now compute the absolute difference (error)
1532         for (int n = 0; n < numberMarkers; n++)
1533         {
1534             currentPointDistance += norm(list_points2d[n] -
    list_points2dProjected[n]);
1535         }
1536 
1537         //! if the difference with the current permutation is smaller than the smallest value till now
1538         //! it is probably the more correct permutation
1539         if (currentPointDistance < minPointDistance)
1540         {
1541             minPointDistance = currentPointDistance;    //!< set the
     smallest value of difference to the current one
1542             for (int b = 0; b < numberMarkers; b++)    //!< now safe the better permutation
1543             {
1544                 pointOrderIndicesNew[b] = pointOrderIndices[b];
1545             }
1546         }
1547 
1548 
1549     }
1550     //! try every permutation
1551     while (std::next_permutation(pointOrderIndices,
    pointOrderIndices + 4));
1552 
1553     //! now that the correct order is found assign it to the indices array
1554     for (int w = 0; w < numberMarkers; w++)
1555     {
1556         pointOrderIndices[w] = pointOrderIndicesNew[w];
1557     }
1558     gotOrder = true;
1559 }
```

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

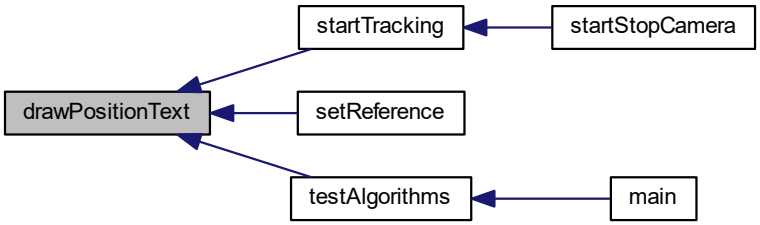
in	*Picture*	is the camera image in OpenCV matrix format.
in	*Position*	is the position of the tracked object in navigation CoSy.
in	*Euler*	are the Euler angles with respect to the navigation frame.
in	*error*	is the reprojection error of the pose estimation.

Definition at line 1315 of file main.cpp.

```
1316 {
1317     ss.str("");
1318     ss << "X: " << Position[0] << " m";
1319     putText(Picture, ss.str(), cv::Point(200, 440), 1, 1, cv::Scalar(255, 255, 255));
1320
1321     ss.str("");
1322     ss << "Y: " << Position[1] << " m";
1323     putText(Picture, ss.str(), cv::Point(200, 455), 1, 1, cv::Scalar(255, 255, 255));
1324
1325     ss.str("");
1326     ss << "Z: " << Position[2] << " m";
1327     putText(Picture, ss.str(), cv::Point(200, 470), 1, 1, cv::Scalar(255, 255, 255));
1328
1329     ss.str("");
1330     ss << "Heading: " << Euler[2] << " deg";
1331     putText(Picture, ss.str(), cv::Point(350, 440), 1, 1, cv::Scalar(255, 255, 255));
1332
1333     ss.str("");
1334     ss << "Pitch: " << Euler[1] << " deg";
1335     putText(Picture, ss.str(), cv::Point(350, 455), 1, 1, cv::Scalar(255, 255, 255));
1336
1337     ss.str("");
```

```
1338     ss << "Roll: " << Euler[0] << " deg";
1339     putText(Picture, ss.str(), cv::Point(350, 470), 1, 1, cv::Scalar(255, 255, 255));
1340
1341     ss.str("");
1342     ss << "Error: " << error << " px";
1343     putText(Picture, ss.str(), cv::Point(10, 470), 1, 1, cv::Scalar(255, 255, 255));
1344 }
```

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

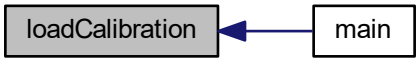
in	*method*	whether or not load the camera calibration from calibration.xml. If ==0 then yes, if != 0 then let the user select a different file.

Definition at line 923 of file main.cpp.

```
923                                 {
924
925     QString fileName;
926     if (method == 0)
927     {
928         fileName = "calibration.xml";
929     }
930     else
931     {
932         fileName = QFileDialog::getOpenFileName(nullptr, "Choose a previous saved calibration file", "", "
      Calibration Files (*.xml);;All Files (*)");
933         if (fileName.length() == 0)
934         {
935             fileName = "calibration.xml";
936         }
937     }
938     FileStorage fs;
939     fs.open(fileName.toUtf8().constData(), FileStorage::READ);
940     fs["CameraMatrix"] >> cameraMatrix;
941     fs["DistCoeff"] >> distCoeffs;
942     commObj.addLog("Loaded calibration from file:");
```

```
943     commObj.addLog(fileName);
944     ss.str("");
945     ss << "\nCamera Matrix is" << "\n" << cameraMatrix << "\n";
946     ss << "\nDistortion Coefficients are" << "\n" << distCoeffs << "\n";
947     commObj.addLog(QString::fromStdString(ss.str()));
948 }
```

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 1348 of file main.cpp.

```
1349 {
1350     //! Open the referenceData.xml that contains the rotation from camera CoSy to ground CoSy
1351     FileStorage fs;
1352     fs.open("referenceData.xml", FileStorage::READ);
1353     fs["M_NC"] >> M_CN;
1354     fs["M_NC"] >> RmatRef;
1355     fs["posRef"] >> posRef;
1356     fs["eulerRef"] >> eulerRef;
1357     commObj.addLog("Loaded reference pose.");
1358 }
```

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

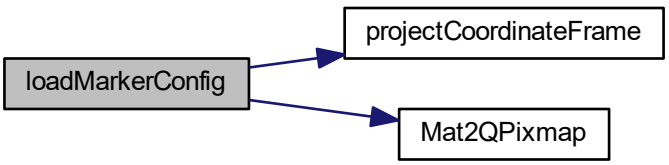
in	*method*	whether or not load the configuration from the markerStandard.xml. If ==0 load it, if != 0 let the user select a different file.

Definition at line 1195 of file main.cpp.

```
1196 {
1197     QString fileName;
1198     //! during start up of the programm load the standard marker configuration
1199     if (method == 0)
1200     {
1201         //! open the standard marker configuration file
1202         FileStorage fs;
1203         fs.open("markerStandard.xml", FileStorage::READ);
1204 
1205         //! copy the values to the respective variables
1206         fs["numberMarkers"] >> numberMarkers;
1207 
1208         //! inizialise vectors with correct length depending on the number of markers
1209         list_points3d = std::vector<Point3d>(numberMarkers);
1210         list_points2d = std::vector<Point2d>(numberMarkers);
1211         list_points2dOld = std::vector<Point2d>(numberMarkers);
1212         list_points2dDifference = std::vector<double>(
    numberMarkers);
1213         list_points2dProjected = std::vector<Point2d>(
    numberMarkers);
1214         list_points2dUnsorted = std::vector<Point2d>(
    numberMarkers);
1215 
1216         //! save the marker locations in the points3d vector
1217         fs["list_points3d"] >> list_points3d;
1218         fs.release();
1219         commObj.addLog("Loaded marker configuration from file:");
1220         commObj.addLog(fileName);
1221 
1222 
1223 
1224     }
1225     else
1226     {
1227         //! if the load marker configuration button was clicked show a open file dialog
1228         fileName = QFileDialog::getOpenFileName(nullptr, "Choose a previous saved marker configuration file
    ", "", "marker configuratio files (*.xml);;All Files (*)");
1229 
1230         //! was cancel or abort clicked
1231         if (fileName.length() == 0)
1232         {
1233             //! if yes load the standard marker configuration
1234             fileName = "markerStandard.xml";
1235         }
1236 
1237         //! open the selected marker configuration file
1238         FileStorage fs;
1239         fs.open(fileName.toUtf8().constData(), FileStorage::READ);
1240 
1241         //! copy the values to the respective variables
1242         fs["numberMarkers"] >> numberMarkers;
1243 
1244         //! inizialise vectors with correct length depending on the number of markers
1245         list_points3d = std::vector<Point3d>(numberMarkers);
1246         list_points2d = std::vector<Point2d>(numberMarkers);
1247         list_points2dOld = std::vector<Point2d>(numberMarkers);
1248         list_points2dDifference = std::vector<double>(numberMarkers);
1249         list_points2dProjected = std::vector<Point2d>(numberMarkers);
1250         list_points2dUnsorted = std::vector<Point2d>(numberMarkers);
1251 
1252         //! save the marker locations in the points3d vector
1253         fs["list_points3d"] >> list_points3d;
1254         fs.release();
1255         commObj.addLog("Loaded marker configuration from file:");
1256         commObj.addLog(fileName);
1257 
1258     }
1259 
1260     //! Print out the number of markers and their position to the GUI
1261     ss.str("");
1262     ss << "Number of Markers: " << numberMarkers << "\n";
1263     ss << "Marker 3D Points X,Y and Z [mm]: \n";
1264     for (int i = 0; i < numberMarkers; i++)
1265     {
```

```
1266         ss << "Marker " << i + 1 << ":\t" << list_points3d[i].x << "\t" << list_points3d[i].y << "\t" <<
     list_points3d[i].z << "\n";
1267     }
1268     commObj.addLog(QString::fromStdString(ss.str()));
1269 
1270     //! check if P3P algorithm can be enabled, it needs exactly 4 marker points to work
1271     if (numberMarkers == 4)
1272     {
1273         //! if P3P is possible, let the user choose which algorithm he wants but keep iterative active
1274         methodPNP = 0;
1275         commObj.enableP3P(true);
1276     }
1277     else
1278     {
1279         //! More (or less) marker than 4 loaded, P3P is not possible, hence user cant select P3P in GUI
1280         methodPNP = 0;
1281         commObj.enableP3P(false);
1282         commObj.addLog("P3P algorithm disabled, only works with 4 markers.");
1283     }
1284 
1285     //! now display the marker configuration in the camera view
1286     Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
1287 
1288     //! Set the camera pose parallel to the marker coordinate system
1289     Tvec.at<double>(0) = 0;
1290     Tvec.at<double>(1) = 0;
1291     Tvec.at<double>(2) = 4500;
1292     Rvec.at<double>(0) = 0 * 3.141592653589 / 180.0;
1293     Rvec.at<double>(1) = 0 * 3.141592653589 / 180.0;
1294     Rvec.at<double>(2) = -90. * 3.141592653589 / 180.0;
1295 
1296     projectPoints(list_points3d, Rvec, Tvec, cameraMatrix,
     distCoeffs, list_points2dProjected);
1297     for (int i = 0; i < numberMarkers; i++)
1298     {
1299         circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 3, Scalar(255, 0, 0
     ), 3);
1300     }
1301 
1302     projectCoordinateFrame(cFrame);
1303     QPixmap QPFrame;
1304     QPFrame = Mat2QPixmap(cFrame);
1305     commObj.changeImage(QPFrame);
1306     QCoreApplication::processEvents();
1307 
1308 }
```

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.

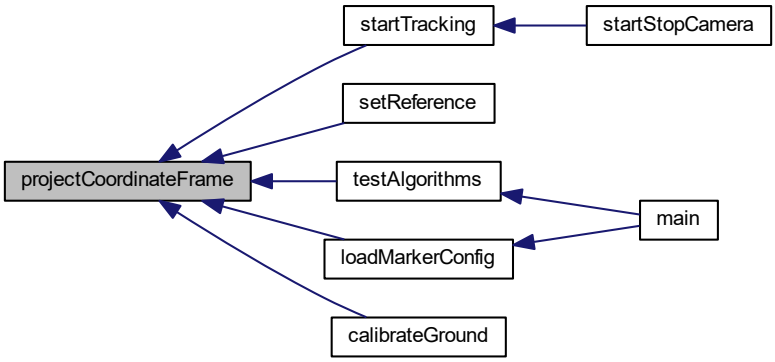
Parameters

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

Definition at line 1081 of file main.cpp.

```
1082 {
1083     projectPoints(coordinateFrame, Rvec, Tvec,
      cameraMatrix, distCoeffs, coordinateFrameProjected);
1084     line(pictureFrame, coordinateFrameProjected[0],
      coordinateFrameProjected[3], Scalar(0, 0, 255), 2); //!<z-axis
1085     line(pictureFrame, coordinateFrameProjected[0],
      coordinateFrameProjected[1], Scalar(255, 0, 0), 2); //!<x-axis
1086     line(pictureFrame, coordinateFrameProjected[0],
      coordinateFrameProjected[2], Scalar(0, 255, 0), 2); //!<y-axis
1087 }
```

Here is the caller graph for this function:



3.2.2.10 sendDataUDP()

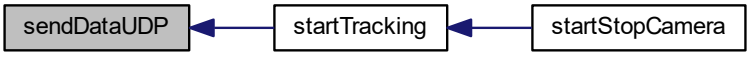
```
void sendDataUDP (
            cv::Vec3d & Position,
            cv::Vec3d & Euler )
```

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 1154 of file main.cpp.

```
1155 {
1156     datagram.clear();
1157     QDataStream out(&datagram, QIODevice::WriteOnly);
1158     out.setVersion(QDataStream::Qt_4_3);
1159     out << (float)Position[0] << (float)Position[1] << (float)Position[2];
1160     out << (float)Euler[0] << (float)Euler[1] << (float)Euler[2]; //! Roll Pitch Heading
1161     udpSocketObject->writeDatagram(datagram,
      IPAdressObject, portObject);
1162
1163     //! if second receiver is activated send it also the tracking data
1164     if (safety2Enable)
1165     {
1166         udpSocketSafety2->writeDatagram(datagram,
      IPAdressSafety2, portSafety2);
1167     }
1168
1169 }
```

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 1122 of file main.cpp.

```
1123 {
1124     headingOffset = d;
1125     d = d * 3.141592653589 / 180.0; //! Convert heading offset from degrees to rad
1126
1127     //! Calculate rotation about x axis
1128     Mat R_x = (Mat_<double>(3, 3) <<
1129         1, 0, 0,
1130         0, 1, 0,
1131         0, 0, 1
1132         );
1133
1134     //! Calculate rotation about y axis
1135     Mat R_y = (Mat_<double>(3, 3) <<
1136         1, 0, 0,
1137         0, 1, 0,
1138         0, 0, 1
1139         );
1140
1141     //! Calculate rotation about z axis
1142     Mat R_z = (Mat_<double>(3, 3) <<
```

```
1143         cos(d), -sin(d), 0,
1144         sin(d), cos(d), 0,
1145         0, 0, 1);
1146
1147
1148     //! Combined rotation matrix
1149     M_HeadingOffset = R_z * R_y * R_x;
1150 }
```

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.

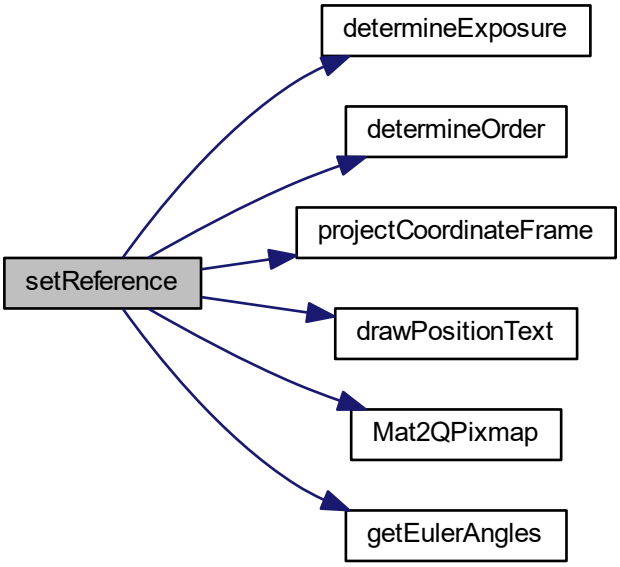
Definition at line 595 of file main.cpp.

```
596 {
597     //! initialize the variables with starting values
598     gotOrder = false;
599     posRef = 0;
600     eulerRef = 0;
601     RmatRef = 0;
602     Rvec = RvecOriginal;
603     Tvec = TvecOriginal;
604
605     determineExposure();
606
607     ss.str("");
608     commObj.addLog("Started reference coordinate determination.");
609
610     CameraLibrary_EnableDevelopment();
611     //! Initialize Camera SDK ===--
612     CameraLibrary::CameraManager::X();
613
614     //! At this point the Camera SDK is actively looking for all connected cameras and will initialize
615     //! them on it's own.
616
617     //! Get a connected camera ==================----
618     CameraManager::X().WaitForInitialization();
619     Camera *camera = CameraManager::X().GetCamera();
620
621     //! If no device connected, pop a message box and exit ==--
622     if (camera == 0)
623     {
624         commObj.addLog("No camera found!");
625         return 1;
626     }
627
628     //! Determine camera resolution to size application window ==----
629     int cameraWidth = camera->Width();
630     int cameraHeight = camera->Height();
631     camera->GetDistortionModel(distModel);
```

```
632     cv::Mat matFrame(cv::Size(cameraWidth, cameraHeight), CV_8UC1);
633
634     //! Set camera mode to precision mode, it directly provides marker coordinates
635     camera->SetVideoType(Core::PrecisionMode);
636
637     //! Start camera output ==--
638     camera->Start();
639
640     //! Turn on some overlay text so it's clear things are     ===---
641     //! working even if there is nothing in the camera's view. ===---
642     //! Set some other parameters as well of the camera
643     camera->SetTextOverlay(true);
644     camera->SetFrameRate(intFrameRate);
645     camera->SetIntensity(intIntensity);
646     camera->SetIRFilter(true);
647     camera->SetContinuousIR(false);
648     camera->SetHighPowerMode(false);
649
650     //! sample some frames and calculate the position and attitude. then average those values and use that
      as zero position
651     int numberSamples = 0;
652     int numberToSample = 200;
653     double projectionError = 0; //!< difference between the marker points as seen by the camera and the
      projected marker points with Rvec and Tvec
654
655     while (numberSamples < numberToSample)
656     {
657         //! Fetch a new frame from the camera ===---
658         Frame *frame = camera->GetFrame();
659
660         if (frame)
661         {
662             //! Ok, we've received a new frame, lets do something
663             //! with it.
664             if (frame->ObjectCount() == numberMarkers)
665             {
666                 //!for(int i=0; i<frame->ObjectCount(); i++)
667                 for (int i = 0; i < numberMarkers; i++)
668                 {
669                     cObject *obj = frame->Object(i);
670                     list_points2dUnsorted[i] = cv::Point2d(obj->X(), obj->Y());
671                 }
672
673                 if (gotOrder == false)
674                 {
675                     determineOrder();
676                 }
677
678                 //! sort the 2d points with the correct indices as found in the preceeding order
      determination algorithm
679                 for (int w = 0; w < numberMarkers; w++)
680                 {
681                     list_points2d[w] = list_points2dUnsorted[
    pointOrderIndices[w]];
682                 }
683                 list_points2dOld = list_points2dUnsorted;
684
685                 //!Compute the pose from the 3D-2D corresponses
686                 solvePnP(list_points3d, list_points2d,
    cameraMatrix, distCoeffs, Rvec, Tvec, useGuess,
    methodPNP);
687
688                 //! project the marker 3d points with the solution into the camera image CoSy and calculate
      difference to true camera image
689                 projectPoints(list_points3d, Rvec, Tvec,
    cameraMatrix, distCoeffs, list_points2dProjected);
690                 projectionError = norm(list_points2dProjected,
    list_points2d);
691
692                 double maxValue = 0;
693                 double minValue = 0;
694                 minMaxLoc(Tvec.at<double>(2), &minValue, &maxValue);
695
696                 if (maxValue > 10000 || minValue < 0)
697                 {
698                     ss.str("");
699                     ss << "Negative z distance, thats not possible. Start the set zero routine again or
      restart Programm.";
700                     commObj.addLog(QString::fromStdString(ss.str()));
701                     frame->Release();
702                     return 1;
703                 }
704
705                 if (projectionError > 3)
706                 {
707                     commObj.addLog("Reprojection error is bigger than 3 pixel. Correct marker
      configuration loaded?\nMarker position measured precisely?");
```

```
708                     frame->Release();
709                     return 1;
710                 }
711 
712                 if (norm(positionOld) - norm(Tvec) < 0.05)   //!<Iterative Method needs time
     to converge to solution
713                 {
714                     add(posRef, Tvec, posRef);
715                     add(eulerRef, Rvec, eulerRef); //!< That are not the values of yaw,
     roll and pitch yet! Rodriguez has to be called first.
716                     numberSamples++;    //!<  one sample more :D
717                     commObj.progressUpdate(numberSamples * 100 / numberToSample);
718                 }
719                 positionOld = Tvec;
720 
721                 Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
722                 for (int i = 0; i < numberMarkers; i++)
723                 {
724                     circle(cFrame, Point(list_points2d[i].x,
    list_points2d[i].y), 6, Scalar(0, 225, 0), 3);
725                 }
726                 projectCoordinateFrame(cFrame);
727                 projectPoints(list_points3d, Rvec, Tvec,
    cameraMatrix, distCoeffs, list_points2d);
728                 for (int i = 0; i < numberMarkers; i++)
729                 {
730                     circle(cFrame, Point(list_points2d[i].x,
    list_points2d[i].y), 3, Scalar(225, 0, 0), 3);
731                 }
732                 drawPositionText(cFrame, position,
    eulerAngles, projectionError);
733 
734                 QPixmap QPFrame;
735                 QPFrame = Mat2QPixmap(cFrame);
736                 commObj.changeImage(QPFrame);
737                 QCoreApplication::processEvents();
738 
739             }
740             frame->Release();
741         }
742     }
743     //! Release camera ==--
744     camera->Release();
745 
746     //!Divide by the number of samples to get the mean of the reference position
747     divide(posRef, numberToSample, posRef);
748     divide(eulerRef, numberToSample, eulerRef); //!< eulerRef is here in Axis Angle
     notation
749 
750     Rodrigues(eulerRef, RmatRef);               //!< axis angle to rotation matrix
751     //!-- Euler Angles, finally
752     getEulerAngles(RmatRef, eulerRef); //!<  rotation matrix to euler
753     ss.str("");
754     ss << "RmatRef is:\n";
755     ss << RmatRef << "\n";
756     ss << "Reference Position is:\n";
757     ss << posRef << "[mm] \n";
758     ss << "Reference Euler Angles are:\n";
759     ss << eulerRef << "[deg] \n";
760 
761     //! compute the difference between last obtained TVec and the average Value
762     //! When it is large the iterative method has not converged properly so it is advised to start the
     setReference() function once again
763     double error = norm(posRef) - norm(Tvec);
764     if (error > 5.0)
765     {
766         ss << "Caution, distance between reference position and last position is: " << error << "\n Start
     the set zero routine once again.";
767     }
768     commObj.addLog(QString::fromStdString(ss.str()));
769     commObj.progressUpdate(0);
770     return 0;
771 }
```

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 261 of file main.cpp.

```
261                     {
262
263
264     gotOrder = false; //! The order of points, hence which entry in list_points3d corresponds to
        which in list_points2d is not calculated yet
265     Rvec = RvecOriginal; //! Use the value of Rvec that was set in main() as starting value
        for the solvePnP algorithm
266     Tvec = TvecOriginal; //! Use the value of Tvec that was set in main() as starting value
        for the solvePnP algorithm
267     GetLocalTime(&logDate);  //! Get the current date and time to name the log file
268
269     //! Concat the log file name as followed. The file is saved in the folder /logs in the Rigid Track
        installation folder
270     logFileName = "./logs/positionLog_" + QString::number(logDate.wDay) + "_" +
      QString::number(logDate.wMonth) + "_" + QString::number(logDate.wYear);
271     logFileName += "_" + QString::number(logDate.wHour) + "_" + QString::number(
      logDate.wMinute) + "_" + QString::number(logDate.wSecond) + ".txt";
272     logName = logFileName.toStdString(); //! Convert the QString to a standard string
273
274     determineExposure(); //! Get the exposure where the right amount of markers is
        detected
275
276     //! For OptiTrack Ethernet cameras, it's important to enable development mode if you
277     //! want to stop execution for an extended time while debugging without disconnecting
```

```
278     //! the Ethernet devices.  Lets do that now:
279
280     CameraLibrary_EnableDevelopment();
281     CameraLibrary::CameraManager::X(); //! Initialize Camera SDK
282
283     //! At this point the Camera SDK is actively looking for all connected cameras and will initialize
284     //! them on it's own
285
286     //! Get a connected camera
287     CameraManager::X().WaitForInitialization();
288     Camera *camera = CameraManager::X().GetCamera();
289
290     //! If no camera can be found, inform user in message log and exit function
291     if (camera == 0)
292     {
293         commObj.addLog("No camera found!");
294         return 1;
295     }
296
297     //! Determine camera resolution to size application window
298     int cameraWidth = camera->Width();
299     int cameraHeight = camera->Height();
300
301     camera->SetVideoType(Core::PrecisionMode);  //! Set the camera mode to precision mode, it used
    greyscale imformation for marker property calculations
302
303     camera->Start(); //! Start camera output
304
305     //! Turn on some overlay text so it's clear things are
306     //! working even if there is nothing in the camera's view
307     camera->SetTextOverlay(true);
308     camera->SetExposure(intExposure);    //! Set the camera exposure
309     camera->SetIntensity(intIntensity); //! Set the camera infrared LED intensity
310     camera->SetFrameRate(intFrameRate); //! Set the camera framerate to 100 Hz
311     camera->SetIRFilter(true);  //! Enable the filter that blocks visible light and only passes infrared
    light
312     camera->SetHighPowerMode(true); //! Enable high power mode of the LEDs
313     camera->SetContinuousIR(false); //! Disable continuous LED light
314     camera->SetThreshold(intThreshold); //! Set threshold for marker detection
315
316     //! Create a new matrix that stores the grayscale picture from the camera
317     Mat matFrame = Mat::zeros(cv::Size(cameraWidth, cameraHeight), CV_8UC1);
318     QPixmap QPFrame; //! QPixmap is the corresponding Qt class that saves images
319     //! Matrix that stores the colored picture, hence marker points, coordinate frame and reprojected
    points
320     Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
321
322     int v = 0;  //! Helper variable used to kick safety switch
323     //! Variables for the min and max values that are needed for sanity checks
324     double maxValue = 0;
325     double minValue = 0;
326     int framesDropped = 0; //! Ff a marker is not visible or accuracy is bad increase this counter
327     double projectionError = 0; //! Equals the quality of the tracking
328
329     setUpUDP(); //! Open sockets and ports for UDP communication
330
331     if (safetyEnable) //! If the safety feature is enabled send the starting message
332     {
333         //! Send enable message, hence send a 9 and then a 1
334         data.setNum((int)(9));
335         udpSocketSafety->write(data);
336         data.setNum((int)(1));
337         udpSocketSafety->write(data);
338     }
339
340     //! Fetch a new frame from the camera
341     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
342     while (!gotTime) //! While no new frame is received loop
343     {
344         Frame *frame = camera->GetFrame(); //! Get a new camera frame
345         if (frame)  //! There is actually a new frame
346         {
347             timeFirstFrame = frame->TimeStamp(); //! Get the time stamp for the first frame.
    It is subtracted for the following frames
348             frame->Release();   //! Release the frame so the camera can continue
349             gotTime = true; //! Exit the while loop
350         }
351     }
352
353     //! Now enter the main loop that processes each frame and computes the pose, sends it and logs stuff
354     while (!exitRequested) //! Check if the user has not pressed "Stop Tracking" yet
355     {
356
357         Frame *frame = camera->GetFrame(); //! Fetch a new frame from the camera
358
359         if (frame) //! Did we got a new frame or does the camera still need more time
```

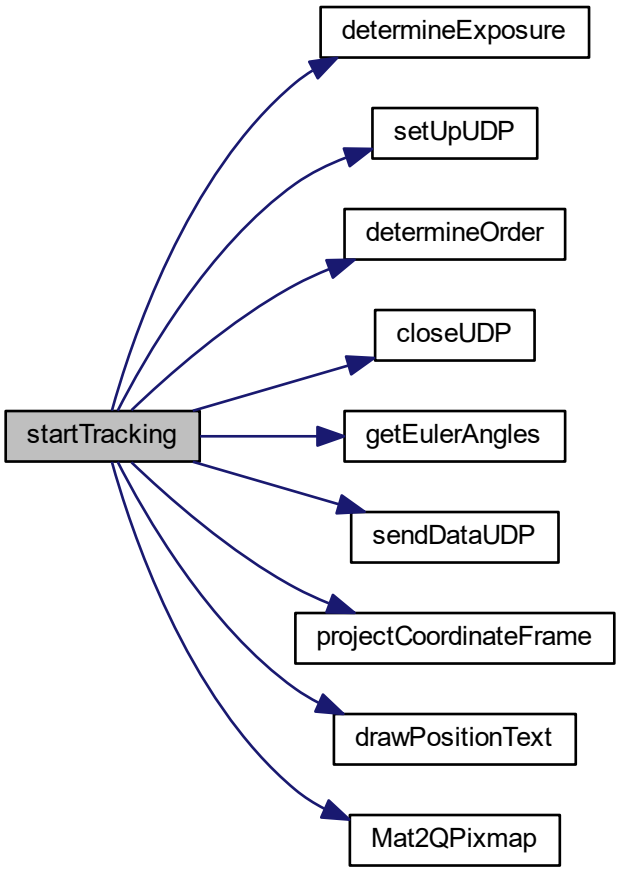
```
360         {
361             framesDropped++; //! Increase by one, if everything is okay it is decreased at the end of the
    loop again
362 
363             //! Only use this frame it the right number of markers is found in the picture
364             if (frame->ObjectCount() == numberMarkers)
365             {
366                 //! Get the marker points in 2D in the camera image frame and store them in the
    list_points2dUnsorted vector
367                 //! The order of points that come from the camera corresponds to the Y coordinate
368                 for (int i = 0; i < numberMarkers; i++)
369                 {
370                     cObject *obj = frame->Object(i);
371                     list_points2dUnsorted[i] = cv::Point2d(obj->X(), obj->Y());
372                 }
373 
374                 if (gotOrder == false) //! Was the order already determined? This is false for the
    first frame and from then on true
375                 {
376                     determineOrder(); //! Now compute the order
377                 }
378 
379                 //! Sort the 2d points with the correct indices as found in the preceeding order
    determination algorithm
380                 for (int w = 0; w < numberMarkers; w++)
381                 {
382                     list_points2d[w] = list_points2dUnsorted[
    pointOrderIndices[w]]; //! pointOrderIndices was calculated in determineOrder()
383                 }
384                 list_points2dOld = list_points2dUnsorted;
385 
386                 //! The first time the 2D-3D corresspondence was determined with gotOrder was okay.
387                 //! But this order can change as the object moves and the marker objects appear in a
388                 //! different order in the frame->Object() array.
389                 //! The solution is that: When a marker point (in the camera image, hence in 2D) was at
390                 //! a position then it wont move that much from one frame to the other.
391                 //! So for the new frame we take a marker object and check which marker was closest this
    point
392                 //! in the old image frame? This is probably the same (true) marker. And we do that for
    every other marker as well.
393                 //! When tracking is good and no frames are dropped because of missing markers this should
    work every frame.
394                 for (int j = 0; j < numberMarkers; j++)
395                 {
396                     minPointDistance = 5000; //! The sum of point distances is set to
    something unrealistic large
397                     for (int k = 0; k < numberMarkers; k++)
398                     {
399                         //! Calculate N_2 norm of unsorted points minus old points
400                         currentPointDistance = norm(
    list_points2dUnsorted[pointOrderIndices[j]] -
    list_points2dOld[k]);
401                         //! If the norm is smaller than minPointDistance the correspondence is more likely
    to be correct
402                         if (currentPointDistance <
    minPointDistance)
403                         {
404                             //! Update the array that saves the new point order
405                             minPointDistance =
    currentPointDistance;
406                             pointOrderIndicesNew[j] = k;
407                         }
408                     }
409                 }
410 
411                 //! Now the new order is found, set the point order to the new value
412                 for (int k = 0; k < numberMarkers; k++)
413                 {
414                     pointOrderIndices[k] = pointOrderIndicesNew[k];
415                     list_points2d[k] = list_points2dUnsorted[
    pointOrderIndices[k]];
416                 }
417 
418                 //! Save the unsorted position of the marker points for the next loop
419                 list_points2dOld = list_points2dUnsorted;
420 
421                 //!Compute the object pose from the 3D-2D corresponses
422                 solvePnP(list_points3d, list_points2d,
    cameraMatrix, distCoeffs, Rvec, Tvec, useGuess,
    methodPNP);
423 
424                 //! Project the marker 3d points with the solution into the camera image CoSy and calculate
    difference to true camera image
425                 projectPoints(list_points3d, Rvec, Tvec,
    cameraMatrix, distCoeffs, list_points2dProjected);
426                 projectionError = norm(list_points2dProjected,
    list_points2d); //! Difference of true pose and found pose
```

```
427
428                 //! Increase the framesDropped variable if accuracy of tracking is too bad
429                 if (projectionError > 5)
430                 {
431                     framesDropped++;
432                 }
433                 else
434                 {
435                     framesDropped = 0;  //! Set number of subsequent frames dropped to zero because error
    is small enough and no marker was missing
436                 }
437
438                 //! Get the min and max values from TVec for sanity check
439                 minMaxLoc(Tvec.at<double>(2), &minValue, &maxValue);
440
441                 //! Sanity check of values. negative z means the marker CoSy is behind the camera, that's
    not possible.
442                 if (minValue < 0)
443                 {
444                     commObj.addLog("Negative z distance, that is not possible. Start the set zero
    routine again or restart Program.");
445                     frame->Release(); //! Release the frame so the camera can move on
446                     camera->Release(); //! Release the camera
447                     closeUDP(); //! Close all UDP connections so the programm can be closed later
    on and no resources are locked
448                     return 1; //! Exit the function
449                 }
450
451                 //! Next step is the transformation from camera CoSy to navigation CoSy
452                 //! Compute the relative object position from the reference position to the current one
453                 //! given in the camera CoSy: \f$ T_C^{NM} = Tvec - Tvec_{Ref} \f$
454                 subtract(Tvec, posRef, position);
455
456                 //! Transform the position from the camera CoSy to the navigation CoSy with INS alligned
    heading and convert from [mm] to [m]
457                 //! \f$ T_N^{NM} = M_{NC} \times T_C^{NM} \f$
458                 Mat V = 0.001 * M_HeadingOffset * M_CN.t() * (Mat)
    position;
459                 position = V;   //! Position is the result of the preceeding calculation
460                 position[2] *= invertZ;  //! Invert Z if check box in GUI is activated,
    hence height above ground is considered
461
462                 //! Realtive angle between reference orientation and current orientation
463                 Rodrigues(Rvec, Rmat);  //! Convert axis angle respresentation to ordinary rotation
    matrix
464
465                 //! The difference of the reference rotation and the current rotation
466                 //! \f$ R_{ NM } = M_{ NC } \times R_{ CM } \f$
467                 Rmat = RmatRef.t() *Rmat;
468
469                 //! Euler Angles, finally
470                 getEulerAngles(Rmat, eulerAngles); //! Get the euler angles
    from the rotation matrix
471                 eulerAngles[2] += headingOffset; //! Add the heading offset to the
    heading angle
472
473                 //! 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
474                 frameTime = frame->TimeStamp() - timeOld;   //! Time between the old frame
    and the current frame
475                 timeOld = frame->TimeStamp();    //! Set the old frame time to the current  one
476                 velocity[0] = (position[0] - positionOld[0]) /
    frameTime; //! Calculate the x velocity with finite differences
477                 velocity[1] = (position[1] - positionOld[1]) /
    frameTime; //! Calculate the y velocity with finite differences
478                 velocity[2] = (position[2] - positionOld[2]) /
    frameTime; //! Calculate the z velocity with finite differences
479                 positionOld = position;  //! Set the old position to the current one for
    next frame velocity calcuation
480
481                 //! Send position and Euler angles over WiFi with 100 Hz
482                 sendDataUDP(position, eulerAngles);
483
484                 //! Save the values in a log file, values are:
485                 //! Time sinc tracking started  Position    Euler Angles    Velocity
486                 logfile.open(logName, std::ios::app); //! Open the log file, the folder is
    RigidTrackInstallationFolder/logs
487                 logfile << frame->TimeStamp() - timeFirstFrame << ";" <<
    position[0] << ";" << position[1] << ";" << position[2] << ";";
488                 logfile << eulerAngles[0] << ";" <<
    eulerAngles[1] << ";" << eulerAngles[2] << ";";
489                 logfile << velocity[0] << ";" << velocity[1] << ";" <<
    velocity[2] << "\n";
490                 logfile.close(); //! Close the file to save values
491             }
492
493             //! Check if the position and euler angles are below the allowed value, if yes send OKAY signal
```

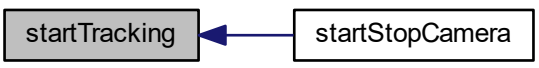
```
 (1), if not send shutdown signal (0)
494             //! Absolute x, y and z position in navigation CoSy must be smaller than the allowed distance
495             if (safetyEnable)
496             {
497                 if ((abs(position[0]) < safetyBoxLength && abs(position[1]) <
    safetyBoxLength && abs(position[2]) < safetyBoxLength))
498                 {
499                     //! Absolute Euler angles must be smaller than allowed value. Heading is not considered
500                     if ((abs(eulerAngles[0]) < safetyAngle && abs(eulerAngles[1]) <
    safetyAngle))
501                     {
502                         //! Send the OKAY signal to the desired computer every 5th time
503                         if (v == 5) {
504                             data.setNum((int)(1));
505                             udpSocketSafety->write(data); //! Send the 1
506                             v = 0; //! reset the counter that is needed for decimation to every 5th time
     step
507                         }
508                     }
509                     //! The euler angles of the object exceeded the allowed euler angles, send the shutdown
     signal (0)
510                     else
511                     {
512                         data.setNum((int)(0));  //! Send the shutdown signal, a 0
513                         udpSocketSafety->write(data);
514                         commObj.addLog("Object exceeded allowed Euler angles, shutdown signal sent."
    ); //! Inform the user
515
516                     }
517                 }
518                 //! The position of the object exceeded the allowed position, shut the object down
519                 else
520                 {
521                     data.setNum((int)(0));  //! Send the shutdown signal, a 0
522                     udpSocketSafety->write(data);
523                     commObj.addLog("Object left allowed area, shutdown signal sent."); //! Inform
     the user
524
525                 }
526             }
527
528             //! Inform the user if tracking system is disturbed (marker lost or so) or error was too big
529             if (framesDropped > 10)
530             {
531                 if (safetyEnable) //! Also send the shutdown signal
532                 {
533                     data.setNum((int)(0));  //! Send the shutdown signal, a 0
534                     udpSocketSafety->write(data);
535                 }
536                 commObj.addLog("Lost marker points or precision was bad!"); //! Inform the user
537                 framesDropped = 0;
538             }
539
540             //! Rasterize the frame so it can be shown in the GUI
541             frame->Rasterize(cameraWidth, cameraHeight, matFrame.step,
    BACKBUFFER_BITSPERPIXEL, matFrame.data);
542
543             //! Convert the frame from greyscale as it comes from the camera to rgb color
544             cvtColor(matFrame, cFrame, COLOR_GRAY2RGB);
545
546             //! Project (draw) the marker CoSy origin into 2D and save it in the cFrame image
547             projectCoordinateFrame(cFrame);
548
549             //! Project the marker points from 3D to the camera image frame (2d) with the computed pose
550             projectPoints(list_points3d, Rvec, Tvec,
    cameraMatrix, distCoeffs, list_points2d);
551             for (int i = 0; i < numberMarkers; i++)
552             {
553                 //! Draw a circle around the projected points so the result can be better compared to the
     real marker position
554                 //! In the resulting picture those are the red dots
555                 circle(cFrame, Point(list_points2d[i].x,
    list_points2d[i].y), 3, Scalar(225, 0, 0), 3);
556             }
557
558             //! Write the current position, attitude and error values as text in the frame
559             drawPositionText(cFrame, position, eulerAngles, projectionError);
560
561             //! Send the new camera picture to the GUI and call the GUI processing routine
562             QPixmap QPFrame;
563             QPFrame = Mat2QPixmap(cFrame);
564             commObj.changeImage(QPFrame); //! Update the picture in the GUI
565             QCoreApplication::processEvents(); //! Give Qt time to handle everything
566
567             //! Release the camera frame to fetch the new one
568             frame->Release();
```

```
569         }
570     }
571
572     //! User choose to stop the tracking, clean things up
573     closeUDP(); //! Close the UDP connections so resources are deallocated
574     camera->Release();  //! Release camera
575     return 0;
576 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



3.2.2.14 testAlgorithms()

```
void testAlgorithms ( )
```

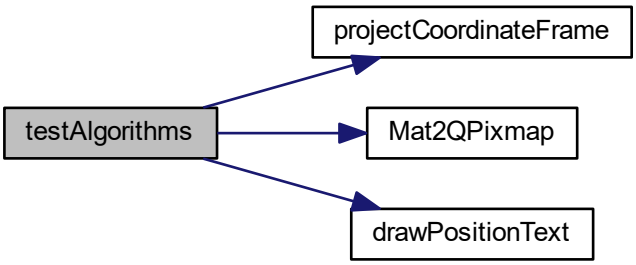
Project some points from 3D to 2D and then check the accuracy of the algorithms. Mainly to generate something that can be shown in the camera view so the user knows everything loaded correctly.

Definition at line 952 of file main.cpp.

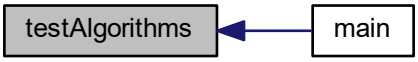
```
953 {
954
955     int _methodPNP;
956
957     std::vector<Point2d> noise(numberMarkers);
958
959     RvecOriginal = Rvec;
960     TvecOriginal = Tvec;
961
962     projectPoints(list_points3d, Rvec, Tvec, cameraMatrix,
    distCoeffs, list_points2dProjected);
963
964     ss.str("");
965     ss << "Unsorted Points 2D Projected \n";
966     ss << list_points2dProjected << "\n";
967     commObj.addLog(QString::fromStdString(ss.str()));
968
969     Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
970     for (int i = 0; i < numberMarkers; i++)
971     {
972         circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 6, Scalar(0, 255, 0
    ), 3);
973     }
974
975     projectCoordinateFrame(cFrame);
976
977     ss.str("");
978     ss << "=======================================================\n";
979     ss << "================= Projected Points ==================\n";
980     ss << list_points2dProjected << "\n";
981
982     randn(noise, 0, 0.5);
983     add(list_points2dProjected, noise, list_points2dProjected);
984
985     ss << "================ With Noise Points ==================\n";
986     ss << list_points2dProjected << "\n";
987     commObj.addLog(QString::fromStdString(ss.str()));
988
989
990     bool useGuess = true;
991     _methodPNP = 0; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP  //!< not used
992
993     solvePnP(list_points3d, list_points2dProjected, cameraMatrix,
    distCoeffs, Rvec, Tvec, useGuess, _methodPNP);
994
995     ss.str("");
996     ss << "=======================================================\n";
997     ss << "==================== Iterative ====================\n";
998     ss << "rvec: " << "\n";
999     ss << Rvec << "\n";
1000     ss << "tvec: " << "\n";
1001     ss << Tvec << "\n";
1002
1003     commObj.addLog(QString::fromStdString(ss.str()));
1004
1005     _methodPNP = 1; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP UPnP not used
1006     Rvec = cv::Mat::zeros(3, 1, CV_64F);
1007     Tvec = cv::Mat::zeros(3, 1, CV_64F);
1008     solvePnP(list_points3d, list_points2dProjected, cameraMatrix,
    distCoeffs, Rvec, Tvec, useGuess, _methodPNP);
1009
1010     ss.str("");
1011     ss << "=======================================================\n";
1012     ss << "====================    EPNP    ====================\n";
1013     ss << "rvec: " << "\n";
1014     ss << Rvec << "\n";
1015     ss << "tvec: " << "\n";
1016     ss << Tvec << "\n";
1017
1018     projectPoints(list_points3d, Rvec, Tvec, cameraMatrix,
    distCoeffs, list_points2dProjected);
1019     for (int i = 0; i < numberMarkers; i++)
```

```
1020     {
1021         circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 3, Scalar(255, 0, 0
     ), 3);
1022     }
1023     QPixmap QPFrame;
1024     QPFrame = Mat2QPixmap(cFrame);
1025     commObj.changeImage(QPFrame);
1026     QCoreApplication::processEvents();
1027     commObj.addLog(QString::fromStdString(ss.str()));
1028     if (numberMarkers == 4)
1029     {
1030         _methodPNP = 2; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP  //!< not used
1031         Rvec = cv::Mat::zeros(3, 1, CV_64F);
1032         Tvec = cv::Mat::zeros(3, 1, CV_64F);
1033         solvePnP(list_points3d, list_points2dProjected,
     cameraMatrix, distCoeffs, Rvec, Tvec, useGuess, _methodPNP);
1034
1035         ss.str("");
1036         ss << "=========================================================\n";
1037         ss << "====================     P3P    ====================\n";
1038         ss << "rvec: " << "\n";
1039         ss << Rvec << "\n";
1040         ss << "tvec: " << "\n";
1041         ss << Tvec << "\n";
1042
1043         projectPoints(list_points3d, Rvec, Tvec, cameraMatrix,
     distCoeffs, list_points2dProjected);
1044         for (int i = 0; i < numberMarkers; i++)
1045         {
1046             circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 3, Scalar(255,
     0, 0), 3);
1047         }
1048         double projectionError = norm(list_points2dProjected, list_points2d);
1049         putText(cFrame, "Testing Algorithms Finished", cv::Point(5, 420), 1, 1, cv::Scalar(255, 255, 255));
1050         drawPositionText(cFrame, position, eulerAngles, projectionError)
     ;
1051
1052         QPixmap QPFrame;
1053         QPFrame = Mat2QPixmap(cFrame);
1054         commObj.changeImage(QPFrame);
1055         QCoreApplication::processEvents();
1056         commObj.addLog(QString::fromStdString(ss.str()));
1057     }
1058
1059     _methodPNP = 4; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP  //!< not used
1060     Rvec = cv::Mat::zeros(3, 1, CV_64F);
1061     Tvec = cv::Mat::zeros(3, 1, CV_64F);
1062     solvePnP(list_points3d, list_points2dProjected, cameraMatrix,
     distCoeffs, Rvec, Tvec, useGuess, _methodPNP);
1063
1064     ss.str("");
1065     ss << "=========================================================\n";
1066     ss << "====================    UPNP    ====================\n";
1067     ss << "rvec: " << "\n";
1068     ss << Rvec << "\n";
1069     ss << "tvec: " << "\n";
1070     ss << Tvec << "\n";
1071
1072     commObj.addLog(QString::fromStdString(ss.str()));
1073
1074     Rvec = RvecOriginal;
1075     Tvec = TvecOriginal;
1076
1077 }
```

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.

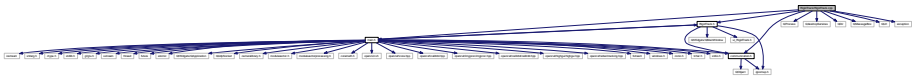
Definition at line 68 of file main.cpp.

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

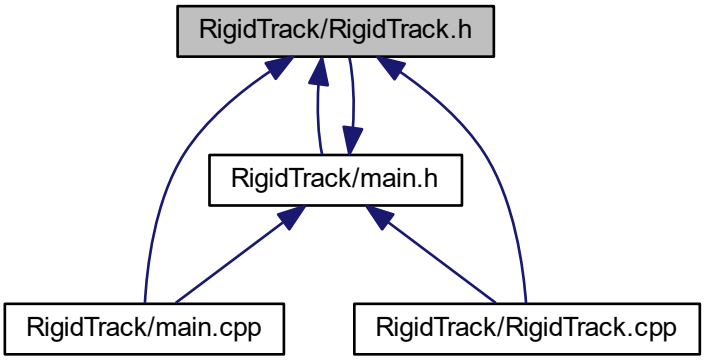
3.4 RigidTrack/RigidTrack.h File Reference

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