

book

**b**

## Rigid Track

Generated by Doxygen 1.8.13



## Table of Contents

1	<a href="#">Rigid Track Doxygen Documentation</a>	- 1 -
1.1	<a href="#">Introduction</a>	- 1 -
1.2	<a href="#">Rigid Track Installation</a>	- 1 -
1.3	<a href="#">Source Code</a>	- 1 -
2	<a href="#">File Index</a>	- 3 -
2.1	<a href="#">File List</a>	- 3 -
3	<a href="#">File Documentation</a>	- 5 -
3.1	<a href="#">RigidTrack/main.cpp File Reference</a>	- 5 -
3.1.1	<a href="#">Detailed Description</a>	- 9 -
3.1.2	<a href="#">Function Documentation</a>	- 9 -
3.1.3	<a href="#">Variable Documentation</a>	- 40 -
3.2	<a href="#">RigidTrack/main.h File Reference</a>	- 41 -
3.2.1	<a href="#">Detailed Description</a>	- 43 -
3.2.2	<a href="#">Function Documentation</a>	- 43 -
3.2.3	<a href="#">Variable Documentation</a>	- 69 -
3.3	<a href="#">RigidTrack/RigidTrack.cpp File Reference</a>	- 70 -
3.3.1	<a href="#">Detailed Description</a>	- 70 -
3.4	<a href="#">RigidTrack/RigidTrack.h File Reference</a>	- 70 -
3.4.1	<a href="#">Detailed Description</a>	- 71 -
	<a href="#">Index</a>	- 73 -



# 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:

RigidTrack/ <a href="#">main.cpp</a>	
Rigid Track main file that contains most functionality .....	- 5 -
RigidTrack/ <a href="#">main.h</a>	
Header file for <a href="#">main.cpp</a> .....	- 41 -
RigidTrack/ <a href="#">RigidTrack.cpp</a>	
Rigid Track GUI source that contains functions for GUI events .....	- 70 -
RigidTrack/ <a href="#">RigidTrack.h</a>	
Rigid Track GUI source header with Qt Signals and Slots .....	- 70 -



## Chapter 3

# File Documentation

### 3.1 RigidTrack/main.cpp File Reference

Rigid Track main file that contains most functionality.

```
#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>
```



- 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 functionality.

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	<i>boardSize</i>	denotes how many squares are in each direction.
in	<i>squareSize</i>	is the square length in millimeters.
out	<i>corners</i>	returns the square corners in millimeters.

Definition at line 238 of file main.cpp.

```
239 {
240     corners.clear();
241
242     for (int i = 0; i < boardSize.height; ++i)
243         for (int j = 0; j < boardSize.width; ++j)
244             corners.push_back(Point3f(float(j*squareSize), float(i*squareSize), 0));
245 }
```

Here is the caller graph for this function:



### 3.1.2.2 calibrateGround()

```
int calibrateGround ( )
```

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

Definition at line 1581 of file main.cpp.



```

1582 {
1583     //! initialize the variables with starting values
1584     gotOrder = false;
1585     posRef = 0;
1586     eulerRef = 0;
1587     RmatRef = 0;
1588     Rvec = RvecOriginal;
1589     Tvec = TvecOriginal;
1590
1591     determineExposure();
1592
1593     ss.str("");
1594     commObj.addLog("Started ground calibration");
1595
1596     CameraLibrary_EnableDevelopment();
1597     //! Initialize Camera SDK ===
1598     CameraLibrary::CameraManager::X();
1599
1600     //! At this point the Camera SDK is actively looking for all connected cameras and will initialize
1601     //! them on it's own.
1602
1603     //! Get a connected camera =====
1604     CameraManager::X().WaitForInitialization();
1605     Camera *camera = CameraManager::X().GetCamera();
1606
1607     //! If no device connected, pop a message box and exit ===
1608     if (camera == 0)
1609     {
1610         commObj.addLog("No camera found!");
1611         return 1;
1612     }
1613
1614     //! Determine camera resolution to size application window =====
1615     int cameraWidth = camera->Width();
1616     int cameraHeight = camera->Height();
1617     camera->GetDistortionModel(distModel);
1618     cv::Mat matFrame(cv::Size(cameraWidth, cameraHeight), CV_8UC1);
1619
1620     //! Set camera mode to precision mode, it directly provides marker coordinates
1621     camera->SetVideoType(Core::PrecisionMode);
1622
1623     //! Start camera output ===
1624     camera->Start();
1625
1626     //! Turn on some overlay text so it's clear things are =====
1627     //! working even if there is nothing in the camera's view. =====
1628     //! Set some other parameters as well of the camera
1629     camera->SetTextOverlay(true);
1630     camera->SetFrameRate(intFrameRate);
1631     camera->SetIntensity(intIntensity);
1632     camera->SetIRFilter(true);
1633     camera->SetContinuousIR(false);
1634     camera->SetHighPowerMode(false);
1635
1636     //! sample some frames and calculate the position and attitude. then average those values and use that
1637     as zero position
1638     int numberSamples = 0;
1639     int numberToSample = 200;
1640     double projectionError = 0;
1641     while (numberSamples < numberToSample)
1642     {
1643         //! Fetch a new frame from the camera =====
1644         Frame *frame = camera->GetFrame();
1645
1646         if (frame)
1647         {
1648             //! Ok, we've received a new frame, lets do something
1649             !! with it.
1650             if (frame->ObjectCount() == numberMarkers)
1651             {
1652                 !!for(int i=0; i<frame->ObjectCount(); i++)
1653                 for (int i = 0; i < numberMarkers; i++)
1654                 {
1655                     cObject *obj = frame->Object(i);
1656                     list_points2dUnsorted[i] = cv::Point2d(obj->X(), obj->Y());
1657                 }
1658
1659                 if (gotOrder == false)
1660                 {
1661                     determineOrder();
1662                 }
1663
1664                 //! sort the 2d points with the correct indices as found in the preceeding order
1665                 determination algorithm
1666                 for (int w = 0; w < numberMarkers; w++)
1667                 {

```

```

1667         list_points2d[w] = list_points2dUnsorted[
1668         pointOrderIndices[w]];
1669         list_points2dOld = list_points2dUnsorted;
1670
1671         ///Compute the pose from the 3D-2D correspondences
1672         solvePnP(list_points3d, list_points2d,
1673         cameraMatrix, distCoeffs, Rvec, Tvec, useGuess,
1674         methodPNP);
1675
1676         /// project the marker 3d points with the solution into the camera image CoSy and calculate
1677         difference to true camera image
1678         projectPoints(list_points3d, Rvec, Tvec,
1679         cameraMatrix, distCoeffs, list_points2dProjected);
1680         projectionError = norm(list_points2dProjected,
1681         list_points2d);
1682
1683         if (projectionError > 3)
1684         {
1685             commObj.addLog("Reprojection error is bigger than 3 pixel. Correct marker
1686             configuration loaded?\nMarker position measured precisely?");
1687             frame->Release();
1688             return 1;
1689         }
1690
1691         double maxValue = 0;
1692         double minValue = 0;
1693         minMaxLoc(Tvec.at<double>(2), &minValue, &maxValue);
1694
1695         if (maxValue > 10000 || minValue < 0)
1696         {
1697             commObj.addLog("Negative z distance, thats not possible. Start the set zero
1698             routine again and check marker configurations.");
1699             frame->Release();
1700             return 1;
1701         }
1702
1703         if (norm(positionOld) - norm(Tvec) < 0.05)    ///!<Iterative Method needs time
1704         to converge to solution
1705         {
1706             add(posRef, Tvec, posRef);
1707             add(eulerRef, Rvec, eulerRef);    ///!< That are not the values of yaw,
1708             roll and pitch yet! Rodriguez has to be called first.
1709             numberSamples++;    ///!<-- one sample more :D
1710             commObj.progressUpdate(numberSamples * 100 / numberToSample);
1711         }
1712         positionOld = Tvec;
1713
1714         Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
1715         for (int i = 0; i < numberMarkers; i++)
1716         {
1717             circle(cFrame, Point(list_points2d[i].x,
1718             list_points2d[i].y), 6, Scalar(0, 225, 0), 3);
1719         }
1720         projectCoordinateFrame(cFrame);
1721         projectPoints(list_points3d, Rvec, Tvec,
1722         cameraMatrix, distCoeffs, list_points2d);
1723         for (int i = 0; i < numberMarkers; i++)
1724         {
1725             circle(cFrame, Point(list_points2d[i].x,
1726             list_points2d[i].y), 3, Scalar(225, 0, 0), 3);
1727         }
1728
1729         QPixmap QPFrame;
1730         QPFrame = Mat2QPixmap(cFrame);
1731         commObj.changeImage(QPFrame);
1732         QApplication::processEvents();
1733     }
1734     frame->Release();
1735 }
1736
1737 ///! Release camera ----
1738 camera->Release();
1739
1740 ///!Divide by the number of samples to get the mean of the reference position
1741 divide(posRef, numberToSample, posRef);
1742 divide(eulerRef, numberToSample, eulerRef);    ///!< eulerRef is here in Axis Angle
1743 notation
1744
1745 Rodrigues(eulerRef, RmatRef);    ///!< axis angle to rotation matrix
1746
1747 getEulerAngles(RmatRef, eulerRef);    ///!< rotation matrix to euler
1748 ss.str("");
1749 ss << "RmatRef is:\n";

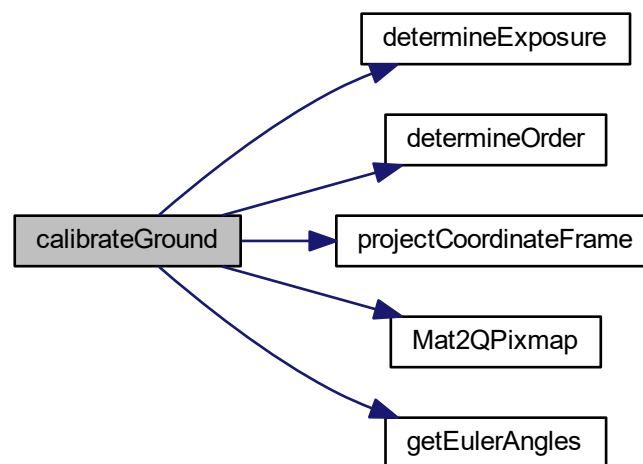
```

```

1740     ss << RmatRef << "\n";
1741     ss << "Reference Position is:\n";
1742     ss << posRef << "[mm] \n";
1743     ss << "Reference Euler angles are:\n";
1744     ss << eulerRef << "[deg] \n";
1745
1746     //!< Save the obtained calibration coefficients in a file for later use
1747     QString fileName = QFileDialog::getSaveFileName(nullptr, "Save ground calibration file", "
referenceData.xml", "Calibration File (*.xml);;All Files (*)");
1748     FileStorage fs(fileName.toUtf8().constData(), FileStorage::WRITE);
1749     fs << "M_NC" << RmatRef;
1750     fs << "eulerRef" << eulerRef;
1751     strBuf = fs.releaseAndGetString();
1752     commObj.changeStatus(QString::fromStdString(strBuf));
1753     commObj.addLog("Saved ground calibration!");
1754     commObj.progressUpdate(0);
1755     return 0;
1756 }

```

Here is the call graph for this function:



### 3.1.2.3 closeUDP()

```
void closeUDP ( )
```

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

Definition at line 1191 of file main.cpp.

```

1192 {
1193     //!< check if the socket is open and if yes close it
1194     if (udpSocketObject->isOpen())
1195     {
1196         udpSocketObject->close();
1197     }
1198 }

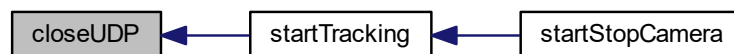
```

```

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

```

Here is the caller graph for this function:



### 3.1.2.4 determineExposure()

```
int determineExposure ( )
```

Get the optimal exposure for the camera. For that find the minimum and maximum exposure were the right number of markers are detected. Then the mean of those two values is used as exposure.

Definition at line 1380 of file main.cpp.

```

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

```

```

1414     //!< Turn on some overlay text so it's clear things are =====
1415     //!< working even if there is nothing in the camera's view. =====
1416     camera->SetTextOverlay(true);
1417     camera->SetExposure(intExposure);    //!< set the camera exposure
1418     camera->SetIntensity(intIntensity);  //!< set the camera infrared LED intensity
1419     camera->SetFrameRate(intFrameRate); //!< set the camera framerate to 100 Hz
1420     camera->SetIRFilter(true);    //!< enable the filter that blocks visible light and only passes infrared
light
1421     camera->SetHighPowerMode(true); //!< enable high power mode of the leds
1422     camera->SetContinuousIR(false); //!< enable continuous LED light
1423     camera->SetThreshold(intThreshold); //!< set threshold for marker detection
1424
1425     //!

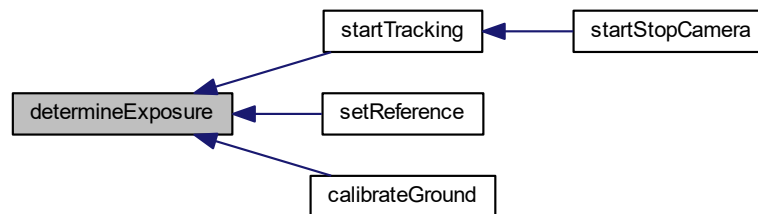
```

```

1492         commObj.addLog("Was not able to detect the right amount of markers.");
1493         ///! Release camera ==--
1494         camera->Release();
1495         return 1;
1496     }
1497     else ///! all markers and not more or less are found
1498     {
1499         frame->Release();
1500         intExposure = (minExposure + maxExposure) / 2.0;
1501         commObj.addLog("Found the correct number of markers.");
1502         commObj.addLog("Exposure set to:");
1503         commObj.addLog(QString::number(intExposure));
1504         break;
1505     }
1506 }
1507 }
1508
1509 camera->Release();
1510 return 0;
1511
1512 }

```

Here is the caller graph for this function:



### 3.1.2.5 determineOrder()

```
void determineOrder ( )
```

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

Definition at line 1516 of file `main.cpp`.

```

1517 {
1518     ///! determine the 3D-2D correspondences that are crucial for the PnP algorithm
1519     ///! Try every possible correspondence and solve PnP
1520     ///! Then project the 3D marker points into the 2D camera image and check the difference
1521     ///! between projected points and points as seen by the camera
1522     ///! the correspondence with the smallest difference is probably the correct one
1523
1524     ///! the difference between true 2D points and projected points is super big
1525     minPointDistance = 5000;
1526     std::sort(pointOrderIndices, pointOrderIndices + 4);
1527
1528     ///! now try every possible permutation of correspondence
1529     do {
1530         ///! reset the starting values for solvePnP
1531         Rvec = RvecOriginal;
1532         Tvec = TvecOriginal;
1533
1534         ///! sort the 2d points with the current permutation

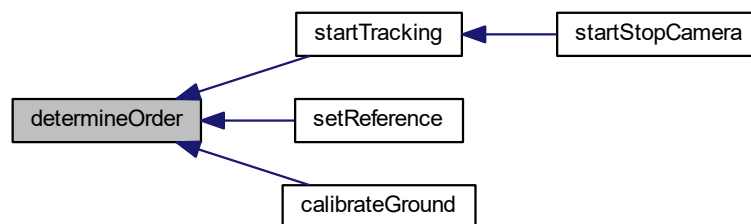
```

```

1535         for (int m = 0; m < numberMarkers; m++)
1536         {
1537             list_points2d[m] = list_points2dUnsorted[
pointOrderIndices[m]];
1538         }
1539
1540         ///! Call solve PnP with P3P since its more robust and sufficient for start value determination
1541         solvePnP(list_points3d, list_points2d,
cameraMatrix, distCoeffs, Rvec, Tvec, useGuess, SOLVEPNP_P3P);
1542
1543         ///! set the current difference of all point correspondences to zero
1544         currentPointDistance = 0;
1545
1546         ///! project the 3D points with the solvePnP solution onto 2D
1547         projectPoints(list_points3d, Rvec, Tvec,
cameraMatrix, distCoeffs, list_points2dProjected);
1548
1549         ///! now compute the absolute difference (error)
1550         for (int n = 0; n < numberMarkers; n++)
1551         {
1552             currentPointDistance += norm(list_points2d[n] -
list_points2dProjected[n]);
1553         }
1554
1555         ///! if the difference with the current permutation is smaller than the smallest value till now
1556         ///! it is probably the more correct permutation
1557         if (currentPointDistance < minPointDistance)
1558         {
1559             minPointDistance = currentPointDistance;    ///!< set the
smallest value of difference to the current one
1560             for (int b = 0; b < numberMarkers; b++)    ///!< now save the better permutation
1561             {
1562                 pointOrderIndicesNew[b] = pointOrderIndices[b];
1563             }
1564         }
1565
1566     }
1567
1568     ///! try every permutation
1569     while (std::next_permutation(pointOrderIndices,
pointOrderIndices + 4));
1570
1571     ///! now that the correct order is found assign it to the indices array
1572     for (int w = 0; w < numberMarkers; w++)
1573     {
1574         pointOrderIndices[w] = pointOrderIndicesNew[w];
1575     }
1576     gotOrder = true;
1577 }

```

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

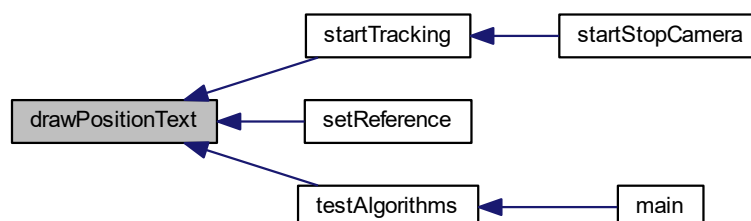
Definition at line 1333 of file main.cpp.

```

1334 {
1335     ss.str("");
1336     ss << "X: " << Position[0] << " m";
1337     putText(Picture, ss.str(), cv::Point(200, 440), 1, 1, cv::Scalar(255, 255, 255));
1338
1339     ss.str("");
1340     ss << "Y: " << Position[1] << " m";
1341     putText(Picture, ss.str(), cv::Point(200, 455), 1, 1, cv::Scalar(255, 255, 255));
1342
1343     ss.str("");
1344     ss << "Z: " << Position[2] << " m";
1345     putText(Picture, ss.str(), cv::Point(200, 470), 1, 1, cv::Scalar(255, 255, 255));
1346
1347     ss.str("");
1348     ss << "Heading: " << Euler[2]*180/3.1415 << " deg";
1349     putText(Picture, ss.str(), cv::Point(350, 440), 1, 1, cv::Scalar(255, 255, 255));
1350
1351     ss.str("");
1352     ss << "Pitch: " << Euler[1] * 180 / 3.1415 << " deg";
1353     putText(Picture, ss.str(), cv::Point(350, 455), 1, 1, cv::Scalar(255, 255, 255));
1354
1355     ss.str("");
1356     ss << "Roll: " << Euler[0] * 180 / 3.1415 << " deg";
1357     putText(Picture, ss.str(), cv::Point(350, 470), 1, 1, cv::Scalar(255, 255, 255));
1358
1359     ss.str("");
1360     ss << "Error: " << error << " px";
1361     putText(Picture, ss.str(), cv::Point(10, 470), 1, 1, cv::Scalar(255, 255, 255));
1362 }

```

Here is the caller graph for this function:





## 3.1.2.7 getEulerAngles()

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

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

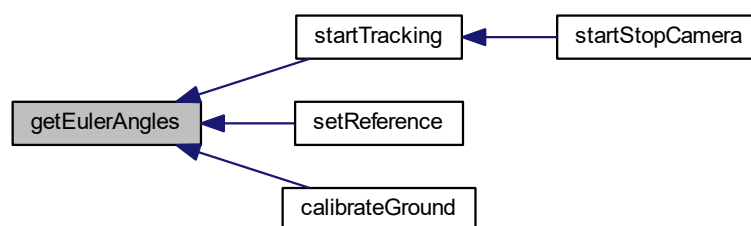
## Parameters

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

Definition at line 256 of file `main.cpp`.

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

Here is the caller graph for this function:



## 3.1.2.8 loadCalibration()

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

Load a previously saved camera calibration from a file.

## Parameters

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

Definition at line 941 of file main.cpp.

```

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

Here is the caller graph for this function:



### 3.1.2.9 loadCameraPosition()

```
void loadCameraPosition ( )
```

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

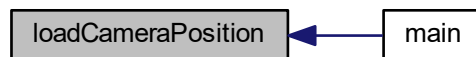
Definition at line 1366 of file main.cpp.

```

1367 {
1368     //! Open the referenceData.xml that contains the rotation from camera CoSy to ground CoSy
1369     FileStorage fs;
1370     fs.open("referenceData.xml", FileStorage::READ);
1371     fs["M_NC"] >> M_CN;
1372     fs["M_NC"] >> RmatRef;
1373     fs["posRef"] >> posRef;
1374     fs["eulerRef"] >> eulerRef;
1375     commObj.addLog("Loaded reference pose.");
1376 }

```

Here is the caller graph for this function:



### 3.1.2.10 loadMarkerConfig()

```

void loadMarkerConfig (
    int method )

```

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

#### Parameters

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

Definition at line 1213 of file main.cpp.

```

1214 {
1215     QString fileName;
1216     //! during start up of the programm load the standard marker configuration
1217     if (method == 0)
1218     {
1219         //! open the standard marker configuration file
1220         FileStorage fs;
1221         fs.open("markerStandard.xml", FileStorage::READ);
1222
1223         //! copy the values to the respective variables
1224         fs["numberMarkers"] >> numberMarkers;
1225
1226         //! initialize vectors with correct length depending on the number of markers
1227         list_points3d = std::vector<Point3d>(numberMarkers);
1228         list_points2d = std::vector<Point2d>(numberMarkers);
1229         list_points2dOld = std::vector<Point2d>(numberMarkers);
1230         list_points2dDifference = std::vector<double>(
            numberMarkers);
1231         list_points2dProjected = std::vector<Point2d>(
            numberMarkers);
1232         list_points2dUnsorted = std::vector<Point2d>(
            numberMarkers);
1233

```

```

1234         //!< save the marker locations in the points3d vector
1235         fs["list_points3d"] >> list_points3d;
1236         fs.release();
1237         commObj.addLog("Loaded marker configuration from file:");
1238         commObj.addLog(fileName);
1239
1240     }
1241
1242     }
1243     else
1244     {
1245         //!< if the load marker configuration button was clicked show a open file dialog
1246         fileName = QFileDialog::getOpenFileName(nullptr, "Choose a previous saved marker configuration file",
1247         "", "marker configuratio files (*.xml);;All Files (*)");
1248
1249         //!< was cancel or abort clicked
1250         if (fileName.length() == 0)
1251         {
1252             //!< if yes load the standard marker configuration
1253             fileName = "markerStandard.xml";
1254         }
1255
1256         //!< open the selected marker configuration file
1257         FileStorage fs;
1258         fs.open(fileName.toUtf8().constData(), FileStorage::READ);
1259
1260         //!< copy the values to the respective variables
1261         fs["numberMarkers"] >> numberMarkers;
1262
1263         //!< initialize vectors with correct length depending on the number of markers
1264         list_points3d = std::vector<Point3d>(numberMarkers);
1265         list_points2d = std::vector<Point2d>(numberMarkers);
1266         list_points2dOld = std::vector<Point2d>(numberMarkers);
1267         list_points2dDifference = std::vector<double>(numberMarkers);
1268         list_points2dProjected = std::vector<Point2d>(numberMarkers);
1269         list_points2dUnsorted = std::vector<Point2d>(numberMarkers);
1270
1271         //!< save the marker locations in the points3d vector
1272         fs["list_points3d"] >> list_points3d;
1273         fs.release();
1274         commObj.addLog("Loaded marker configuration from file:");
1275         commObj.addLog(fileName);
1276     }
1277
1278     //!< Print out the number of markers and their position to the GUI
1279     ss.str("");
1280     ss << "Number of Markers: " << numberMarkers << "\n";
1281     ss << "Marker 3D Points X,Y and Z [mm]: \n";
1282     for (int i = 0; i < numberMarkers; i++)
1283     {
1284         ss << "Marker " << i + 1 << ": \t" << list_points3d[i].x << "\t" << list_points3d[i].y << "\t" <<
1285         list_points3d[i].z << "\n";
1286     }
1287     commObj.addLog(QString::fromStdString(ss.str()));
1288
1289     //!< check if P3P algorithm can be enabled, it needs exactly 4 marker points to work
1290     if (numberMarkers == 4)
1291     {
1292         //!< if P3P is possible, let the user choose which algorithm he wants but keep iterative active
1293         methodPNP = 0;
1294         commObj.enableP3P(true);
1295     }
1296     else
1297     {
1298         //!< More (or less) marker than 4 loaded, P3P is not possible, hence user cant select P3P in GUI
1299         methodPNP = 0;
1300         commObj.enableP3P(false);
1301         commObj.addLog("P3P algorithm disabled, only works with 4 markers.");
1302     }
1303
1304     //!< now display the marker configuration in the camera view
1305     Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
1306
1307     //!< Set the camera pose parallel to the marker coordinate system
1308     Tvec.at<double>(0) = 0;
1309     Tvec.at<double>(1) = 0;
1310     Tvec.at<double>(2) = 4500;
1311     Rvec.at<double>(0) = 0 * 3.141592653589 / 180.0;
1312     Rvec.at<double>(1) = 0 * 3.141592653589 / 180.0;
1313     Rvec.at<double>(2) = -90. * 3.141592653589 / 180.0;
1314
1315     projectPoints(list_points3d, Rvec, Tvec, cameraMatrix,
1316     distCoeffs, list_points2dProjected);
1317     for (int i = 0; i < numberMarkers; i++)
1318     {
1319         circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 3, Scalar(255, 0, 0

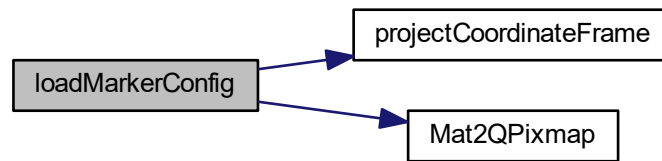
```

```

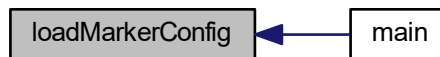
    ), 3);
1318     }
1319
1320     projectCoordinateFrame(cFrame);
1321     QPixmap QPFrame;
1322     QPFrame = Mat2QPixmap(cFrame);
1323     commObj.changeImage(QPFrame);
1324     QApplication::processEvents();
1325
1326 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.1.2.11 main()

```

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

```

`main` initialises the GUI and values for the marker position etc

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

## Parameters

in	<i>argc</i>	is not used.
in	<i>argv</i>	is also not used.

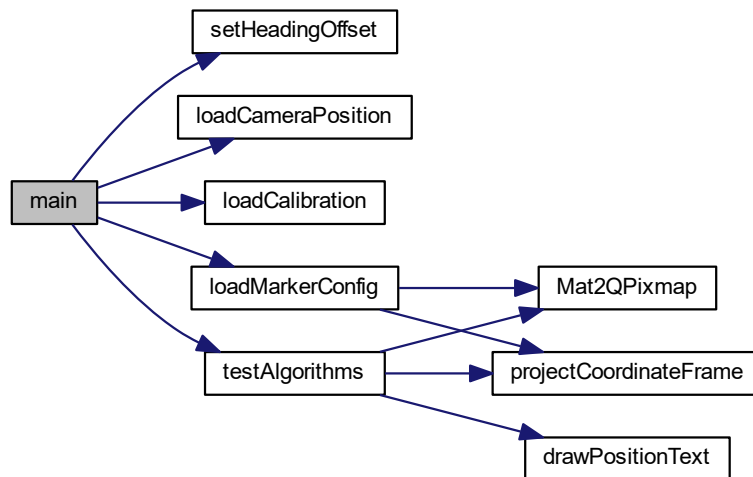
Definition at line 165 of file main.cpp.

```

166 {
167     QApplication a(argc, argv);
168     RigidTrack w;
169     w.show();    //!< show the GUI
170     //!< connect the Qt slots and signals for event handling
171     QObject::connect(&commObj, SIGNAL(statusChanged(QString)), &w, SLOT(setStatus(QString)),
Qt::DirectConnection);
172     QObject::connect(&commObj, SIGNAL(imageChanged(QPixmap)), &w, SLOT(setImage(QPixmap)),
Qt::DirectConnection);
173     QObject::connect(&commObj, SIGNAL(logAdded(QString)), &w, SLOT(setLog(QString)),
Qt::DirectConnection);
174     QObject::connect(&commObj, SIGNAL(logCleared()), &w, SLOT(clearLog(QString)),
Qt::DirectConnection);
175     QObject::connect(&commObj, SIGNAL(P3Penabled(bool)), &w, SLOT(enableP3P(bool)),
Qt::DirectConnection);
176     QObject::connect(&commObj, SIGNAL(progressUpdated(int)), &w, SLOT(progressUpdate(int)),
Qt::DirectConnection);
177
178     commObj.addLog("RigidTrack Version:");
179     commObj.addLog(QString::number(_MSC_FULL_VER));
180     commObj.addLog("Built on:");
181     commObj.addLog(QString(__DATE__));
182
183     //!< initial guesses for position and rotation, important for Iterative Method!
184     Tvec.at<double>(0) = 45;
185     Tvec.at<double>(1) = 45;
186     Tvec.at<double>(2) = 4500;
187     Rvec.at<double>(0) = 0 * 3.141592653589 / 180.0;
188     Rvec.at<double>(1) = 0 * 3.141592653589 / 180.0;
189     Rvec.at<double>(2) = -45 * 3.141592653589 / 180.0;
190
191     //!< Points that make up the marker CoSy axis system, hence one line in each axis direction
192     coordinateFrame = std::vector<Point3d>(4);
193     coordinateFrameProjected = std::vector<Point2d>(4);
194     coordinateFrame[0] = cv::Point3d(0, 0, 0);
195     coordinateFrame[1] = cv::Point3d(300, 0, 0);
196     coordinateFrame[2] = cv::Point3d(0, 300, 0);
197     coordinateFrame[3] = cv::Point3d(0, 0, 300);
198
199     position[0] = 1.1234;    //!< set position initial values
200     position[1] = 1.2345;    //!< set position initial values
201     position[2] = 1.3456;    //!< set position initial values
202
203     velocity[0] = 0.123;    //!< set velocity initial values
204     velocity[1] = 0.234;    //!< set velocity initial values
205     velocity[2] = 0.345;    //!< set velocity initial values
206
207     eulerAngles[0] = 1.002;  //!< set initial euler angles to arbitrary values for testing
208     eulerAngles[1] = 1.003;  //!< set initial euler angles to arbitrary values for testing
209     eulerAngles[2] = 1.004;  //!< set initial euler angles to arbitrary values for testing
210
211     setHeadingOffset(0.0);  //!< set the heading offset to 0
212
213     ss.precision(4);    //!< outputs in the log etc are limited to 3 decimal values
214
215     loadCameraPosition();  //!< load the rotation matrix from camera CoSy to ground CoSy
216     loadCalibration(0);    //!< load the calibration file with the camera intrinsics
217     loadMarkerConfig(0);  //!< load the standard marker configuration
218     testAlgorithms();    //!< test the algorithms and their accuracy
219
220     return a.exec();
221 }

```

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 QPixmap object for the GUI.

#### Parameters

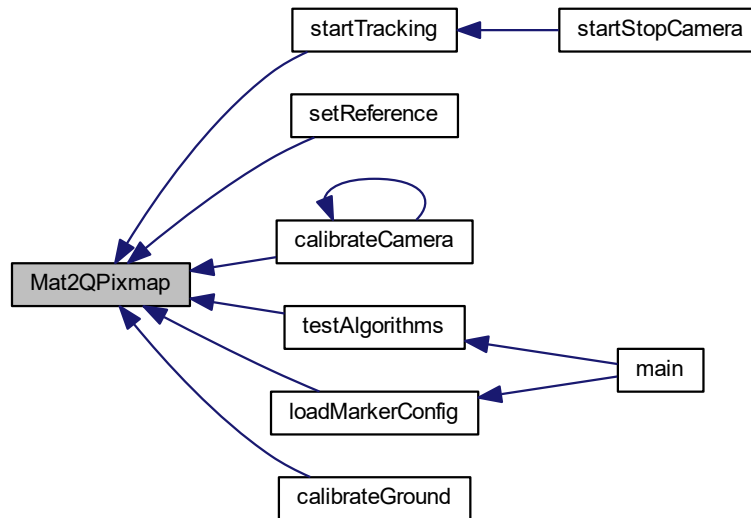
in	src	is the camera image represented as OpenCV matrix.
----	-----	---

Definition at line 225 of file main.cpp.

```

226 {
227     QImage dest((const uchar *)src.data, src.cols, src.rows, src.step, QImage::Format_RGB888);
228     dest.bits(); ///! enforce deep copy, see documentation
229     ///! of QImage::QImage ( const uchar * data, int width, int height, Format format )
230     QPixmap pixmapDest = QPixmap::fromImage(dest);
231     return pixmapDest;
232 }
  
```

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	<code>pictureFrame</code>	the image in which the CoSy frame should be pasted.
----	---------------------------	---

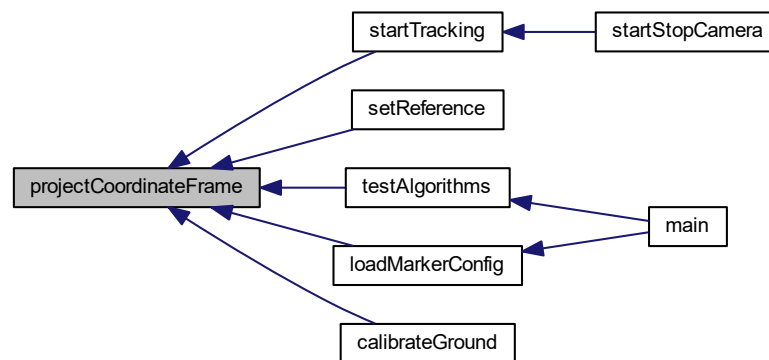
Definition at line 1099 of file main.cpp.

```

1100 {
1101     projectPoints(coordinateFrame, Rvec, Tvec,
1102                  cameraMatrix, distCoeffs, coordinateFrameProjected);
1103     line(pictureFrame, coordinateFrameProjected[0],
1104          coordinateFrameProjected[3], Scalar(0, 0, 255), 2); //!<z-axis
1105     line(pictureFrame, coordinateFrameProjected[0],
1106          coordinateFrameProjected[1], Scalar(255, 0, 0), 2); //!<x-axis
1107     line(pictureFrame, coordinateFrameProjected[0],
1108          coordinateFrameProjected[2], Scalar(0, 255, 0), 2); //!<y-axis
1109 }
```



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 startTracking function because its send rate is less than 100 Hz.

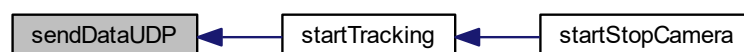
Definition at line 1172 of file main.cpp.

```

1173 {
1174     datagram.clear();
1175     QDataStream out(&datagram, QIODevice::WriteOnly);
1176     out.setVersion(QDataStream::Qt_4_3);
1177     out << (float)Position[0] << (float)Position[1] << (float)Position[2];
1178     out << (float)Euler[0] << (float)Euler[1] << (float)Euler[2]; ///! Roll Pitch Heading
1179     udpSocketObject->writeDatagram(datagram,
1180     IPAddressObject, portObject);
1181     ///! if second receiver is activated send it also the tracking data
1182     if (safety2Enable)
1183     {
1184         udpSocketSafety2->writeDatagram(datagram,
1185     IPAddressSafety2, portSafety2);
1186     }
1187 }

```

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.

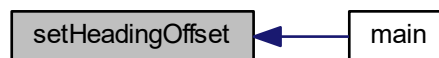
#### Parameters

in	<i>d</i>	denotes heading offset in degrees.
----	----------	------------------------------------

Definition at line 1140 of file main.cpp.

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

Here is the caller graph for this function:



### 3.1.2.16 setReference()

```
int setReference ( )
```

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

Definition at line 613 of file main.cpp.

```

614 {
615     /// initialize the variables with starting values
616     gotOrder = false;
617     posRef = 0;
618     eulerRef = 0;
619     RmatRef = 0;
620     Rvec = RvecOriginal;
621     Tvec = TvecOriginal;
622
623     determineExposure();
624
625     ss.str("");
626     commObj.addLog("Started reference coordinate determination.");
627
628     CameraLibrary_EnableDevelopment();
629     /// Initialize Camera SDK ===
630     CameraLibrary::CameraManager::X();
631
632     /// At this point the Camera SDK is actively looking for all connected cameras and will initialize
633     /// them on it's own.
634
635     /// Get a connected camera =====
636     CameraManager::X().WaitForInitialization();
637     Camera *camera = CameraManager::X().GetCamera();
638
639     /// If no device connected, pop a message box and exit ===
640     if (camera == 0)
641     {
642         commObj.addLog("No camera found!");
643         return 1;
644     }
645
646     /// Determine camera resolution to size application window =====
647     int cameraWidth = camera->Width();
648     int cameraHeight = camera->Height();
649     camera->GetDistortionModel(distModel);
650     cv::Mat matFrame(cv::Size(cameraWidth, cameraHeight), CV_8UC1);
651
652     /// Set camera mode to precision mode, it directly provides marker coordinates
653     camera->SetVideoType(Core::PrecisionMode);
654
655     /// Start camera output ===
656     camera->Start();
657
658     /// Turn on some overlay text so it's clear things are =====
659     /// working even if there is nothing in the camera's view. =====
660     /// Set some other parameters as well of the camera
661     camera->SetTextOverlay(true);
662     camera->SetFrameRate(intFrameRate);
663     camera->SetIntensity(intIntensity);
664     camera->SetIRFilter(true);
665     camera->SetContinuousIR(false);
666     camera->SetHighPowerMode(false);
667
668     /// sample some frames and calculate the position and attitude. then average those values and use that
669     /// as zero position
670     int numberSamples = 0;
671     int numberToSample = 200;
672     double projectionError = 0; ///< difference between the marker points as seen by the camera and the
673     /// projected marker points with Rvec and Tvec
674
675     while (numberSamples < numberToSample)
676     {
677         /// Fetch a new frame from the camera =====
678         Frame *frame = camera->GetFrame();
679
680         if (frame)
681         {
682             /// Ok, we've received a new frame, lets do something
683             /// with it.
684             if (frame->ObjectCount() == numberMarkers)
685             {
686                 for(int i=0; i<frame->ObjectCount(); i++)
687                 for (int i = 0; i < numberMarkers; i++)
688                 {
689                     cObject *obj = frame->Object(i);
690                     list_points2dUnsorted[i] = cv::Point2d(obj->X(), obj->Y());
691                 }
692
693                 if (gotOrder == false)
694                 {
695                     determineOrder();
696                 }
697
698                 /// sort the 2d points with the correct indices as found in the preceeding order
699                 determination_algorithm
700                 for (int w = 0; w < numberMarkers; w++)

```

```

698         {
699             list_points2d[w] = list_points2dUnsorted[
pointOrderIndices[w]];
700         }
701         list_points2dOld = list_points2dUnsorted;
702
703         ///Compute the pose from the 3D-2D correspondences
704         solvePnP(list_points3d, list_points2d,
cameraMatrix, distCoeffs, Rvec, Tvec, useGuess,
methodPNP);
705
706         /// project the marker 3d points with the solution into the camera image CoSy and calculate
difference to true camera image
707         projectPoints(list_points3d, Rvec, Tvec,
cameraMatrix, distCoeffs, list_points2dProjected);
708         projectionError = norm(list_points2dProjected,
list_points2d);
709
710         double maxValue = 0;
711         double minValue = 0;
712         minMaxLoc(Tvec.at<double>(2), &minValue, &maxValue);
713
714         if (maxValue > 10000 || minValue < 0)
715         {
716             ss.str("");
717             ss << "Negative z distance, thats not possible. Start the set zero routine again or
restart Programm.";
718             commObj.addLog(QString::fromStdString(ss.str()));
719             frame->Release();
720             return 1;
721         }
722
723         if (projectionError > 5)
724         {
725             commObj.addLog("Reprojection error is bigger than 5 pixel. Correct marker
configuration loaded?\nMarker position measured precisely?");
726             frame->Release();
727             return 1;
728         }
729
730         if (norm(positionOld) - norm(Tvec) < 0.05)    ///!<Iterative Method needs time
to converge to solution
731         {
732             add(posRef, Tvec, posRef);
733             add(eulerRef, Rvec, eulerRef); ///!< That are not the values of yaw,
roll and pitch yet! Rodriguez has to be called first.
734             numberSamples++;    ///!< one sample more :D
735             commObj.progressUpdate(numberSamples * 100 / numberToSample);
736         }
737         positionOld = Tvec;
738
739         Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
740         for (int i = 0; i < numberMarkers; i++)
741         {
742             circle(cFrame, Point(list_points2d[i].x,
list_points2d[i].y), 6, Scalar(0, 225, 0), 3);
743         }
744         projectCoordinateFrame(cFrame);
745         projectPoints(list_points3d, Rvec, Tvec,
cameraMatrix, distCoeffs, list_points2d);
746         for (int i = 0; i < numberMarkers; i++)
747         {
748             circle(cFrame, Point(list_points2d[i].x,
list_points2d[i].y), 3, Scalar(225, 0, 0), 3);
749         }
750         drawPositionText(cFrame, position,
eulerAngles, projectionError);
751
752         QPixmap QPFrame;
753         QPFrame = Mat2QPixmap(cFrame);
754         commObj.changeImage(QPFrame);
755         QCoreApplication::processEvents();
756
757     }
758     frame->Release();
759 }
760 }
761 ///! Release camera ==--
762 camera->Release();
763
764 ///!Divide by the number of samples to get the mean of the reference position
765 divide(posRef, numberToSample, posRef);
766 divide(eulerRef, numberToSample, eulerRef); ///!< eulerRef is here in Axis Angle
notation
767
768 Rodrigues(eulerRef, RmatRef);    ///!< axis angle to rotation matrix
769 ///!-- Euler Angles, finally

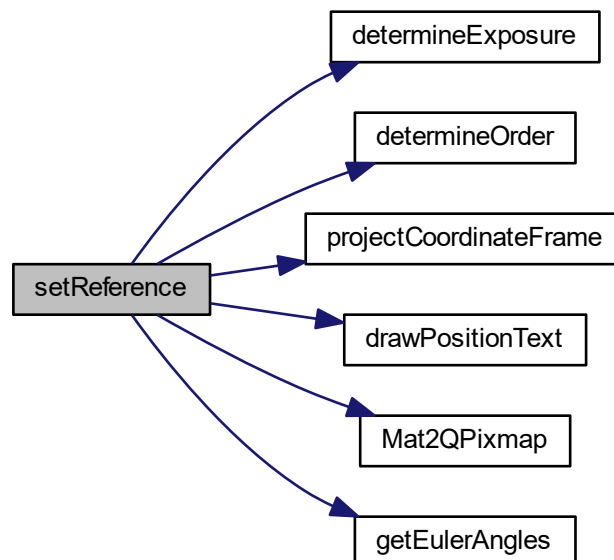
```

```

770     getEulerAngles(RmatRef, eulerRef); //!< rotation matrix to euler
771     ss.str("");
772     ss << "RmatRef is:\n";
773     ss << RmatRef << "\n";
774     ss << "Reference Position is:\n";
775     ss << posRef << "[mm] \n";
776     ss << "Reference Euler Angles are:\n";
777     ss << eulerRef << "[deg] \n";
778
779     //!< compute the difference between last obtained TVec and the average Value
780     //!< When it is large the iterative method has not converged properly so it is advised to start the
781     setReference() function once again
782     double error = norm(posRef) - norm(Tvec);
783     if (error > 5.0)
784     {
785         ss << "Caution, distance between reference position and last position is: " << error << "\n Start
786         the set zero routine once again.";
787     }
788     commObj.addLog(QString::fromStdString(ss.str()));
789     commObj.progressUpdate(0);
790     return 0;
791 }

```

Here is the call graph for this function:



### 3.1.2.17 startTracking()

```
int startTracking ( )
```

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

Definition at line 276 of file main.cpp.

```

276         {
277
278
279         gotOrder = false; //! The order of points, hence which entry in list_points3d corresponds to
which in list_points2d is not calculated yet
280         Rvec = RvecOriginal; //! Use the value of Rvec that was set in main() as starting value
for the solvePnP algorithm
281         Tvec = TvecOriginal; //! Use the value of Tvec that was set in main() as starting value
for the solvePnP algorithm
282         GetLocalTime(&logDate); //! Get the current date and time to name the log file
283
284         //! Concat the log file name as followed. The file is saved in the folder /logs in the Rigid Track
installation folder
285         logFileName = "../logs/positionLog_" + QString::number(logDate.wDay) + "_" +
QString::number(logDate.wMonth) + "_" + QString::number(logDate.wYear);
286         logFileName += "_" + QString::number(logDate.wHour) + "_" + QString::number(
logDate.wMinute) + "_" + QString::number(logDate.wSecond) + ".txt";
287         logName = logFileName.toString(); //! Convert the QString to a standard string
288
289         determineExposure(); //! Get the exposure where the right amount of markers is
detected
290
291         //! For OptiTrack Ethernet cameras, it's important to enable development mode if you
292         //! want to stop execution for an extended time while debugging without disconnecting
293         //! the Ethernet devices. Lets do that now:
294
295         CameraLibrary_EnableDevelopment();
296         CameraLibrary::CameraManager::X(); //! Initialize Camera SDK
297
298         //! At this point the Camera SDK is actively looking for all connected cameras and will initialize
299         //! them on it's own
300
301         //! Get a connected camera
302         CameraManager::X().WaitForInitialization();
303         Camera *camera = CameraManager::X().GetCamera();
304
305         //! If no camera can be found, inform user in message log and exit function
306         if (camera == 0)
307         {
308             commObj.addLog("No camera found!");
309             return 1;
310         }
311
312         //! Determine camera resolution to size application window
313         int cameraWidth = camera->Width();
314         int cameraHeight = camera->Height();
315
316         camera->SetVideoType(Core::PrecisionMode); //! Set the camera mode to precision mode, it used
greyscale information for marker property calculations
317
318         camera->Start(); //! Start camera output
319
320         //! Turn on some overlay text so it's clear things are
321         //! working even if there is nothing in the camera's view
322         camera->SetTextOverlay(true);
323         camera->SetExposure(intExposure); //! Set the camera exposure
324         camera->SetIntensity(intIntensity); //! Set the camera infrared LED intensity
325         camera->SetFrameRate(intFrameRate); //! Set the camera framerate to 100 Hz
326         camera->SetIRFilter(true); //! Enable the filter that blocks visible light and only passes infrared
light
327         camera->SetHighPowerMode(true); //! Enable high power mode of the LEDs
328         camera->SetContinuousIR(false); //! Disable continuous LED light
329         camera->SetThreshold(intThreshold); //! Set threshold for marker detection
330
331         //! Create a new matrix that stores the grayscale picture from the camera
332         Mat matFrame = Mat::zeros(cv::Size(cameraWidth, cameraHeight), CV_8UC1);
333         QPixmap QPFrame; //! QPixmap is the corresponding Qt class that saves images
334         //! Matrix that stores the colored picture, hence marker points, coordinate frame and reprojected
points
335         Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
336
337         int v = 0; //! Helper variable used to kick safety switch
338         //! Variables for the min and max values that are needed for sanity checks
339         double maxValue = 0;
340         double minValue = 0;
341         int framesDropped = 0; //! If a marker is not visible or accuracy is bad increase this counter
342         double projectionError = 0; //! Equals the quality of the tracking
343
344         setUpUDP(); //! Open sockets and ports for UDP communication
345
346         if (safetyEnable) //! If the safety feature is enabled send the starting message
347         {
348             //! Send enable message, hence send a 9 and then a 1
349             data.setNum((int)(9));
350             udpSocketSafety->write(data);
351             data.setNum((int)(1));
352             udpSocketSafety->write(data);

```

```

353     }
354
355     /// Fetch a new frame from the camera
356     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
357     while (!gotTime) /// While no new frame is received loop
358     {
359         Frame *frame = camera->GetFrame(); /// Get a new camera frame
360         if (frame) /// There is actually a new frame
361         {
362             timeFirstFrame = frame->TimeStamp(); /// Get the time stamp for the first frame.
It is subtracted for the following frames
363             frame->Release(); /// Release the frame so the camera can continue
364             gotTime = true; /// Exit the while loop
365         }
366     }
367
368     /// Now enter the main loop that processes each frame and computes the pose, sends it and logs stuff
369     while (!exitRequested) /// Check if the user has not pressed "Stop Tracking" yet
370     {
371
372         Frame *frame = camera->GetFrame(); /// Fetch a new frame from the camera
373
374         if (frame) /// Did we got a new frame or does the camera still need more time
375         {
376             framesDropped++; /// Increase by one, if everything is okay it is decreased at the end of the
loop again
377
378             /// Only use this frame if the right number of markers is found in the picture
379             if (frame->ObjectCount() == numberMarkers)
380             {
381                 /// Get the marker points in 2D in the camera image frame and store them in the
list_points2dUnsorted vector
382                 /// The order of points that come from the camera corresponds to the Y coordinate
383                 for (int i = 0; i < numberMarkers; i++)
384                 {
385                     cObject *obj = frame->Object(i);
386                     list_points2dUnsorted[i] = cv::Point2d(obj->X(), obj->Y());
387                 }
388
389                 if (gotOrder == false) /// Was the order already determined? This is false for the
first frame and from then on true
390                 {
391                     determineOrder(); /// Now compute the order
392                 }
393
394                 /// Sort the 2d points with the correct indices as found in the preceeding order
determination algorithm
395                 for (int w = 0; w < numberMarkers; w++)
396                 {
397                     list_points2d[w] = list_points2dUnsorted[
pointOrderIndices[w]]; /// pointOrderIndices was calculated in determineOrder()
398                 }
399
400                 /// The first time the 2D-3D corresspondence was determined with gotOrder was okay.
401                 /// But this order can change as the object moves and the marker objects appear in a
402                 /// different order in the frame->Object() array.
403                 /// The solution is that: When a marker point (in the camera image, hence in 2D) was at
404                 /// a position then it wont move that much from one frame to the other.
405                 /// So for the new frame we take a marker object and check which marker was closest this
point
406                 /// in the old image frame? This is probably the same (true) marker. And we do that for
every other marker as well.
407                 /// When tracking is good and no frames are dropped because of missing markers this should
work every frame.
408                 for (int j = 0; j < numberMarkers; j++)
409                 {
410                     minPointDistance = 5000; /// The sum of point distances is set to
something unrealistic large
411                     for (int k = 0; k < numberMarkers; k++)
412                     {
413                         /// Calculate N_2 norm of unsorted points minus old points
414                         currentPointDistance = norm(
list_points2dUnsorted[pointOrderIndices[j]] -
list_points2dOld[k]);
415                         /// If the norm is smaller than minPointDistance the correspondence is more likely
to be correct
416                         if (currentPointDistance <
minPointDistance)
417                         {
418                             /// Update the array that saves the new point order
419                             minPointDistance =
currentPointDistance;
420                             pointOrderIndicesNew[j] = k;
421                         }
422                     }
423                 }

```

```

424
425     ///! Now the new order is found, set the point order to the new value
426     for (int k = 0; k < numberMarkers; k++)
427     {
428         pointOrderIndices[k] = pointOrderIndicesNew[k];
429         list_points2d[k] = list_points2dUnsorted[
pointOrderIndices[k]];
430     }
431
432     ///! Save the unsorted position of the marker points for the next loop
433     list_points2dOld = list_points2dUnsorted;
434
435     ///! Compute the object pose from the 3D-2D correspondences
436     solvePnP(list_points3d, list_points2d,
cameraMatrix, distCoeffs, Rvec, Tvec, useGuess,
methodPNP);
437
438     ///! Project the marker 3d points with the solution into the camera image CoSy and calculate
difference to true camera image
439     projectPoints(list_points3d, Rvec, Tvec,
cameraMatrix, distCoeffs, list_points2dProjected);
440     projectionError = norm(list_points2dProjected,
list_points2d); ///! Difference of true pose and found pose
441
442     ///! Increase the framesDropped variable if accuracy of tracking is too bad
443     if (projectionError > 5)
444     {
445         framesDropped++;
446     }
447     else
448     {
449         framesDropped = 0; ///! Set number of subsequent frames dropped to zero because error
is small enough and no marker was missing
450     }
451
452     ///! Get the min and max values from Tvec for sanity check
453     minMaxLoc(Tvec.at<double>(2), &minValue, &maxValue);
454
455     ///! Sanity check of values. negative z means the marker CoSy is behind the camera, that's
not possible.
456     if (minValue < 0)
457     {
458         commObj.addLog("Negative z distance, that is not possible. Start the set zero
routine again or restart Program.");
459         frame->Release(); ///! Release the frame so the camera can move on
460         camera->Release(); ///! Release the camera
461         closeUDP(); ///! Close all UDP connections so the programm can be closed later
on and no resources are locked
462         return 1; ///! Exit the function
463     }
464
465     ///! Next step is the transformation from camera CoSy to navigation CoSy
466     ///! Compute the relative object position from the reference position to the current one
467     ///! given in the camera CoSy: \f$ T_C^{NM} = Tvec - Tvec_{Ref} \f$
468     subtract(Tvec, posRef, position);
469
470     ///! Transform the position from the camera CoSy to the navigation CoSy with INS aligned
heading and convert from [mm] to [m]
471     ///! \f$ T_N^{NM} = M_{NC} \times T_C^{NM} \f$
472     Mat V = 0.001 * M_HeadingOffset * M_CN.t() * (Mat)
position;
473     position = V; ///! Position is the result of the preceeding calculation
474     position[2] *= invertZ; ///! Invert Z if check box in GUI is activated,
hence height above ground is considered
475
476     ///! Realtime angle between reference orientation and current orientation
477     Rodrigues(Rvec, Rmat); ///! Convert axis angle representation to ordinary rotation
matrix
478
479     ///! The difference of the reference rotation and the current rotation
480     ///! \f$ R_{NM} = M_{NC} \times R_{CM} \f$
481     Rmat = RmatRef.t() * Rmat;
482
483     ///! Euler Angles, finally
484     getEulerAngles(Rmat, eulerAngles); ///! Get the euler angles
from the rotation matrix
485     eulerAngles[2] += headingOffset; ///! Add the heading offset to the
heading angle
486
487     ///! 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
488     frameTime = frame->TimeStamp() - timeOld; ///! Time between the old frame
and the current frame
489     timeOld = frame->TimeStamp(); ///! Set the old frame time to the current one
490     velocity[0] = (position[0] - positionOld[0]) /
frameTime; ///! Calculate the x velocity with finite differences
491     velocity[1] = (position[1] - positionOld[1]) /

```



```

frameTime; //! Calculate the y velocity with finite differences
492     velocity[2] = (position[2] - positionOld[2]) /
frameTime; //! Calculate the z velocity with finite differences
493     positionOld = position; //! Set the old position to the current one for
next frame velocity calculation
494
495     eulerAngles[0] = eulerAngles[0] * -3.141592653589 / 180.0; //!
Convert the Euler angles from degrees to rad
496     eulerAngles[1] = eulerAngles[1] * -3.141592653589 / 180.0;
497     eulerAngles[2] = eulerAngles[2] * 3.141592653589 / 180.0;
498
499     //! Send position and Euler angles over WiFi with 100 Hz
500     sendDataUDP(position, eulerAngles);
501
502     //! Save the values in a log file, values are:
503     //! Time sinc tracking started Position Euler Angles Velocity
504     logfile.open(logName, std::ios::app); //! Open the log file, the folder is
RigidTrackInstallationFolder/logs
505     logfile << frame->TimeStamp() - timeFirstFrame << ";" <<
position[0] << ";" << position[1] << ";" << position[2] << ";";
506     logfile << eulerAngles[0] << ";" <<
eulerAngles[1] << ";" << eulerAngles[2] << ";";
507     logfile << velocity[0] << ";" << velocity[1] << ";" <<
velocity[2] << "\n";
508     logfile.close(); //! Close the file to save values
509 }
510
511 //! Check if the position and euler angles are below the allowed value, if yes send OKAY signal
(1), if not send shutdown signal (0)
512 //! Absolute x, y and z position in navigation CoSy must be smaller than the allowed distance
513 if (safetyEnable)
514 {
515     if ((abs(position[0]) < safetyBoxLength && abs(position[1]) <
safetyBoxLength && abs(position[2]) < safetyBoxLength))
516     {
517         //! Absolute Euler angles must be smaller than allowed value. Heading is not considered
518         if ((abs(eulerAngles[0]) < safetyAngle && abs(eulerAngles[1]) <
safetyAngle))
519         {
520             //! Send the OKAY signal to the desired computer every 5th time
521             if (v == 5) {
522                 data.setNum((int)(1));
523                 udpSocketSafety->write(data); //! Send the 1
524                 v = 0; //! reset the counter that is needed for decimation to every 5th time
step
525             }
526         }
527         //! The euler angles of the object exceeded the allowed euler angles, send the shutdown
signal (0)
528         else
529         {
530             data.setNum((int)(0)); //! Send the shutdown signal, a 0
531             udpSocketSafety->write(data);
532             commObj.addLog("Object exceeded allowed Euler angles, shutdown signal sent."
); //! Inform the user
533         }
534     }
535     //! The position of the object exceeded the allowed position, shut the object down
536     else
537     {
538         data.setNum((int)(0)); //! Send the shutdown signal, a 0
539         udpSocketSafety->write(data);
540         commObj.addLog("Object left allowed area, shutdown signal sent."); //! Inform
the user
541     }
542 }
543
544 //! Inform the user if tracking system is disturbed (marker lost or so) or error was too big
545 if (framesDropped > 10)
546 {
547     if (safetyEnable) //! Also send the shutdown signal
548     {
549         data.setNum((int)(0)); //! Send the shutdown signal, a 0
550         udpSocketSafety->write(data);
551     }
552     commObj.addLog("Lost marker points or precision was bad!"); //! Inform the user
framesDropped = 0;
553 }
554
555 //! Rasterize the frame so it can be shown in the GUI
556 frame->Rasterize(cameraWidth, cameraHeight, matFrame.step,
BACKBUFFER_BITSPERPIXEL, matFrame.data);
557
558 //! Convert the frame from greyscale as it comes from the camera to rgb color
559
560
561

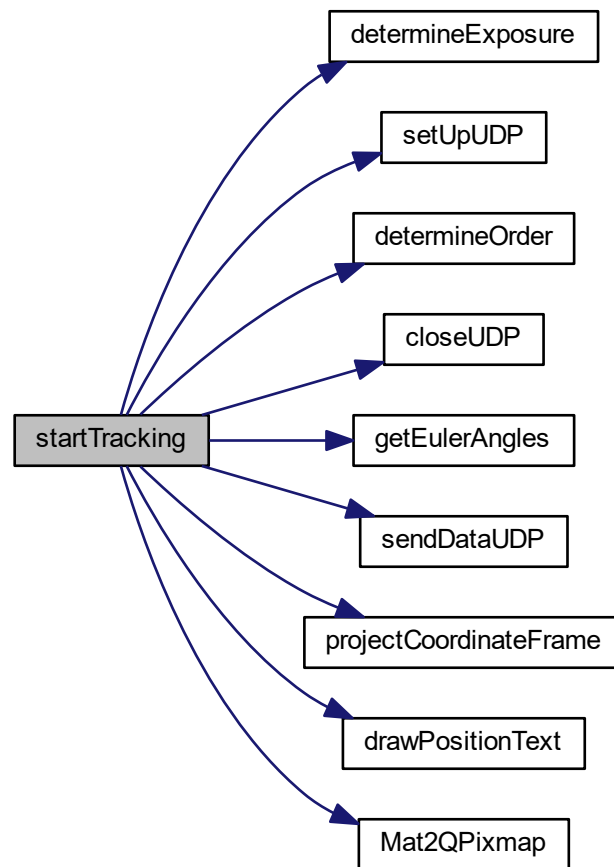
```

```

562         cvtColor(matFrame, cFrame, COLOR_GRAY2RGB);
563
564         //!< Project (draw) the marker CoSy origin into 2D and save it in the cFrame image
565         projectCoordinateFrame(cFrame);
566
567         //!< Project the marker points from 3D to the camera image frame (2d) with the computed pose
568         projectPoints(list_points3d, Rvec, Tvec,
cameraMatrix, distCoeffs, list_points2d);
569         for (int i = 0; i < numberMarkers; i++)
570         {
571             //!< Draw a circle around the projected points so the result can be better compared to the
real marker position
572             //!< In the resulting picture those are the red dots
573             circle(cFrame, Point(list_points2d[i].x,
list_points2d[i].y), 3, Scalar(225, 0, 0), 3);
574         }
575
576         //!< Write the current position, attitude and error values as text in the frame
577         drawPositionText(cFrame, position, eulerAngles, projectionError);
578
579         //!< Send the new camera picture to the GUI and call the GUI processing routine
580         QPixmap QPFrame;
581         QPFrame = Mat2QPixmap(cFrame);
582         commObj.changeImage(QPFrame); //!< Update the picture in the GUI
583         QApplication::processEvents(); //!< Give Qt time to handle everything
584
585         //!< Release the camera frame to fetch the new one
586         frame->Release();
587     }
588 }
589
590 //!< User choose to stop the tracking, clean things up
591 closeUDP(); //!< Close the UDP connections so resources are deallocated
592 camera->Release(); //!< Release camera
593 return 0;
594 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



### 3.1.2.18 testAlgorithms()

```
void testAlgorithms ( )
```

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

Definition at line 970 of file main.cpp.

```

971 {
972
973     int _methodPNP;
974
975     std::vector<Point2d> noise(numberMarkers);
976
977     RvecOriginal = Rvec;
978     TvecOriginal = Tvec;
979
980     projectPoints(list_points3d, Rvec, Tvec, cameraMatrix,
distCoeffs, list_points2dProjected);
981
982     ss.str("");
983     ss << "Unsorted Points 2D Projected \n";
984     ss << list_points2dProjected << "\n";
985     commObj.addLog(QString::fromStdString(ss.str()));
986
987     Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
988     for (int i = 0; i < numberMarkers; i++)
989     {
990         circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 6, Scalar(0, 255, 0
), 3);
991     }
992
993     projectCoordinateFrame(cFrame);
994
995     ss.str("");
996     ss << "=====\n";
997     ss << "===== Projected Points =====\n";
998     ss << list_points2dProjected << "\n";
999
1000     randn(noise, 0, 0.5);
1001     add(list_points2dProjected, noise, list_points2dProjected);
1002
1003     ss << "===== With Noise Points =====\n";
1004     ss << list_points2dProjected << "\n";
1005     commObj.addLog(QString::fromStdString(ss.str()));
1006
1007
1008     bool useGuess = true;
1009     _methodPNP = 0; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP //!< not used
1010
1011     solvePnP(list_points3d, list_points2dProjected, cameraMatrix,
distCoeffs, Rvec, Tvec, useGuess, _methodPNP);
1012
1013     ss.str("");
1014     ss << "=====\n";
1015     ss << "===== Iterative =====\n";
1016     ss << "rvec: " << "\n";
1017     ss << Rvec << "\n";
1018     ss << "tvec: " << "\n";
1019     ss << Tvec << "\n";
1020
1021     commObj.addLog(QString::fromStdString(ss.str()));
1022
1023     _methodPNP = 1; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP UPnP not used
1024     Rvec = cv::Mat::zeros(3, 1, CV_64F);
1025     Tvec = cv::Mat::zeros(3, 1, CV_64F);
1026     solvePnP(list_points3d, list_points2dProjected, cameraMatrix,
distCoeffs, Rvec, Tvec, useGuess, _methodPNP);
1027
1028     ss.str("");
1029     ss << "=====\n";
1030     ss << "===== EPNP =====\n";
1031     ss << "rvec: " << "\n";
1032     ss << Rvec << "\n";
1033     ss << "tvec: " << "\n";
1034     ss << Tvec << "\n";
1035
1036     projectPoints(list_points3d, Rvec, Tvec, cameraMatrix,
distCoeffs, list_points2dProjected);
1037     for (int i = 0; i < numberMarkers; i++)
1038     {
1039         circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 3, Scalar(255, 0, 0
), 3);
1040     }
1041     QPixmap QPFrame;
1042     QPFrame = Mat2QPixmap(cFrame);
1043     commObj.changeImage(QPFrame);

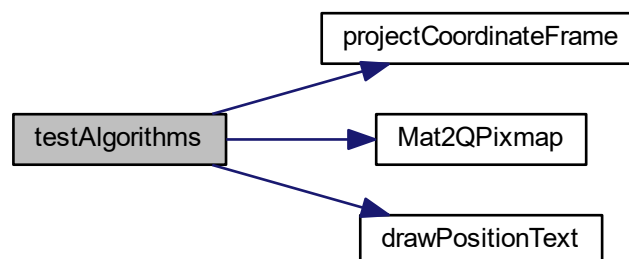
```

```

1044     QCoreApplication::processEvents();
1045     commObj.addLog(QString::fromStdString(ss.str()));
1046     if (numberMarkers == 4)
1047     {
1048         _methodPNP = 2; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP //!< not used
1049         Rvec = cv::Mat::zeros(3, 1, CV_64F);
1050         Tvec = cv::Mat::zeros(3, 1, CV_64F);
1051         solvePnP(list_points3d, list_points2dProjected,
1052                 cameraMatrix, distCoeffs, Rvec, Tvec, useGuess, _methodPNP);
1053
1054         ss.str("");
1055         ss << "=====\n";
1056         ss << "=====P3P=====\n";
1057         ss << "rvec: " << "\n";
1058         ss << Rvec << "\n";
1059         ss << "tvec: " << "\n";
1060         ss << Tvec << "\n";
1061
1062         projectPoints(list_points3d, Rvec, Tvec, cameraMatrix,
1063                     distCoeffs, list_points2dProjected);
1064         for (int i = 0; i < numberMarkers; i++)
1065         {
1066             circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 3, Scalar(255,
1067                                     0, 0), 3);
1068             double projectionError = norm(list_points2dProjected, list_points2d);
1069             putText(cFrame, "Testing Algorithms Finished", cv::Point(5, 420), 1, 1, cv::Scalar(255, 255, 255));
1070             drawPositionText(cFrame, position, eulerAngles, projectionError)
1071         };
1072
1073         QPixmap QPFrame;
1074         QPFrame = Mat2QPixmap(cFrame);
1075         commObj.changeImage(QPFrame);
1076         QCoreApplication::processEvents();
1077         commObj.addLog(QString::fromStdString(ss.str()));
1078     }
1079
1080     _methodPNP = 4; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP //!< not used
1081     Rvec = cv::Mat::zeros(3, 1, CV_64F);
1082     Tvec = cv::Mat::zeros(3, 1, CV_64F);
1083     solvePnP(list_points3d, list_points2dProjected, cameraMatrix,
1084             distCoeffs, Rvec, Tvec, useGuess, _methodPNP);
1085
1086     ss.str("");
1087     ss << "=====\n";
1088     ss << "=====UPNP=====\n";
1089     ss << "rvec: " << "\n";
1090     ss << Rvec << "\n";
1091     ss << "tvec: " << "\n";
1092     ss << Tvec << "\n";
1093
1094     commObj.addLog(QString::fromStdString(ss.str()));
1095
1096     Rvec = RvecOriginal;
1097     Tvec = TvecOriginal;
1098 }

```

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

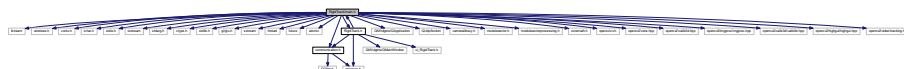
Definition at line 95 of file main.cpp.

## 3.2 RigidTrack/main.h File Reference

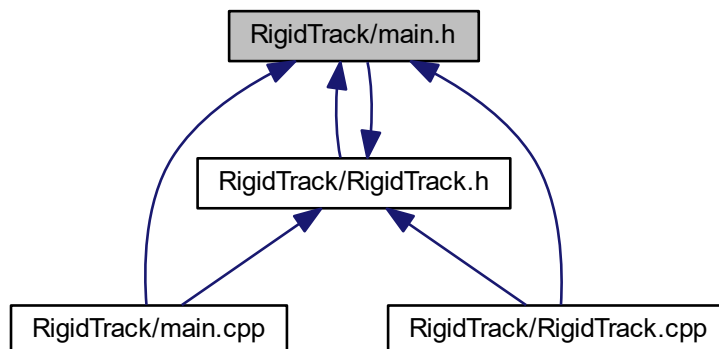
Header file for [main.cpp](#).

```
#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*
- QObject [commObj](#)  
*class that handles the communication from [main.cpp](#) to the GUI*

### 3.2.1 Detailed Description

Header file for [main.cpp](#).

#### Author

Florian J.T. Wachter

#### Version

1.0

#### Date

April, 8th 2017

### 3.2.2 Function Documentation

### 3.2.2.1 calibrateGround()

```
int calibrateGround ( )
```

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

Definition at line 1581 of file main.cpp.

```
1582 {
1583     //! initialize the variables with starting values
1584     gotOrder = false;
1585     posRef = 0;
1586     eulerRef = 0;
1587     RmatRef = 0;
1588     Rvec = RvecOriginal;
1589     Tvec = TvecOriginal;
1590
1591     determineExposure();
1592
1593     ss.str("");
1594     commObj.addLog("Started ground calibration");
1595
1596     CameraLibrary_EnableDevelopment();
1597     //! Initialize Camera SDK ===
1598     CameraLibrary::CameraManager::X();
1599
1600     //! At this point the Camera SDK is actively looking for all connected cameras and will initialize
1601     //! them on it's own.
1602
1603     //! Get a connected camera =====
1604     CameraManager::X().WaitForInitialization();
1605     Camera *camera = CameraManager::X().GetCamera();
1606
1607     //! If no device connected, pop a message box and exit ===
1608     if (camera == 0)
1609     {
1610         commObj.addLog("No camera found!");
1611         return 1;
1612     }
1613
1614     //! Determine camera resolution to size application window =====
1615     int cameraWidth = camera->Width();
1616     int cameraHeight = camera->Height();
1617     camera->GetDistortionModel(distModel);
1618     cv::Mat matFrame(cv::Size(cameraWidth, cameraHeight), CV_8UC1);
1619
1620     //! Set camera mode to precision mode, it directly provides marker coordinates
1621     camera->SetVideoType(Core::PrecisionMode);
1622
1623     //! Start camera output ===
1624     camera->Start();
1625
1626     //! Turn on some overlay text so it's clear things are =====
1627     //! working even if there is nothing in the camera's view. =====
1628     //! Set some other parameters as well of the camera
1629     camera->SetTextOverlay(true);
1630     camera->SetFrameRate(intFrameRate);
1631     camera->SetIntensity(intIntensity);
1632     camera->SetIRFilter(true);
1633     camera->SetContinuousIR(false);
1634     camera->SetHighPowerMode(false);
1635
1636     //! sample some frames and calculate the position and attitude. then average those values and use that
as zero position
1637     int numberSamples = 0;
1638     int numberToSample = 200;
1639     double projectionError = 0;
1640
1641     while (numberSamples < numberToSample)
1642     {
1643         //! Fetch a new frame from the camera =====
1644         Frame *frame = camera->GetFrame();
1645
1646         if (frame)
1647         {
1648             //! Ok, we've received a new frame, lets do something
1649             //! with it.
1650             if (frame->ObjectCount() == numberMarkers)
1651             {
```

```

1652         //!for(int i=0; i<frame->ObjectCount(); i++)
1653         for (int i = 0; i < numberMarkers; i++)
1654         {
1655             cObject *obj = frame->Object(i);
1656             list_points2dUnsorted[i] = cv::Point2d(obj->X(), obj->Y());
1657         }
1658
1659         if (gotOrder == false)
1660         {
1661             determineOrder();
1662         }
1663
1664         //! sort the 2d points with the correct indices as found in the preceeding order
1665         determination algorithm
1666         for (int w = 0; w < numberMarkers; w++)
1667         {
1668             list_points2d[w] = list_points2dUnsorted[
pointOrderIndices[w]];
1669         }
1670         list_points2dOld = list_points2dUnsorted;
1671
1672         //!Compute the pose from the 3D-2D correspondences
1673         solvePnP(list_points3d, list_points2d,
cameraMatrix, distCoeffs, Rvec, Tvec, useGuess,
methodPNP);
1674
1675         //! project the marker 3d points with the solution into the camera image CoSy and calculate
1676         difference to true camera image
1677         projectPoints(list_points3d, Rvec, Tvec,
cameraMatrix, distCoeffs, list_points2dProjected);
1678         projectionError = norm(list_points2dProjected,
list_points2d);
1679
1680         if (projectionError > 3)
1681         {
1682             commObj.addLog("Reprojection error is bigger than 3 pixel. Correct marker
configuration loaded?\nMarker position measured precisely?");
1683             frame->Release();
1684             return 1;
1685         }
1686
1687         double maxValue = 0;
1688         double minValue = 0;
1689         minMaxLoc(Tvec.at<double>(2), &minValue, &maxValue);
1690
1691         if (maxValue > 10000 || minValue < 0)
1692         {
1693             commObj.addLog("Negative z distance, thats not possible. Start the set zero
routine again and
1694             check marker configurations.");
1695             frame->Release();
1696             return 1;
1697         }
1698
1699         if (norm(positionOld) - norm(Tvec) < 0.05)    //!<Iterative Method needs time
to converge to solution
1700         {
1701             add(posRef, Tvec, posRef);
1702             add(eulerRef, Rvec, eulerRef);    //!< That are not the values of yaw,
roll and pitch yet! Rodriguez has to be called first.
1703             numberSamples++;    //!<-- one sample more :D
1704             commObj.progressUpdate(numberSamples * 100 / numberToSample);
1705         }
1706         positionOld = Tvec;
1707
1708         Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
1709         for (int i = 0; i < numberMarkers; i++)
1710         {
1711             circle(cFrame, Point(list_points2d[i].x,
list_points2d[i].y), 6, Scalar(0, 225, 0), 3);
1712         }
1713         projectCoordinateFrame(cFrame);
1714         projectPoints(list_points3d, Rvec, Tvec,
cameraMatrix, distCoeffs, list_points2d);
1715         for (int i = 0; i < numberMarkers; i++)
1716         {
1717             circle(cFrame, Point(list_points2d[i].x,
list_points2d[i].y), 3, Scalar(225, 0, 0), 3);
1718         }
1719
1720         QPixmap QPFrame;
1721         QPFrame = Mat2QPixmap(cFrame);
1722         commObj.changeImage(QPFrame);
1723         QCoreApplication::processEvents();
1724     }

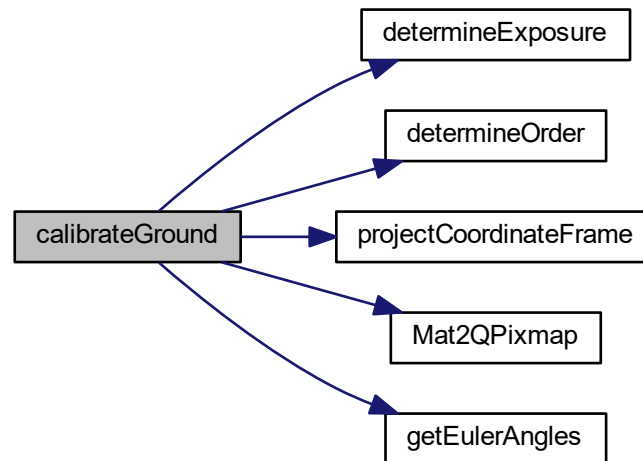
```

```

1725         frame->Release();
1726     }
1727 }
1728 //! Release camera ----
1729 camera->Release();
1730
1731 //! Divide by the number of samples to get the mean of the reference position
1732 divide(posRef, numberToSample, posRef);
1733 divide(eulerRef, numberToSample, eulerRef); //!< eulerRef is here in Axis Angle
notation
1734
1735 Rodrigues(eulerRef, RmatRef); //!< axis angle to rotation matrix
1736
1737 getEulerAngles(RmatRef, eulerRef); //!< rotation matrix to euler
1738 ss.str("");
1739 ss << "RmatRef is:\n";
1740 ss << RmatRef << "\n";
1741 ss << "Reference Position is:\n";
1742 ss << posRef << "[mm] \n";
1743 ss << "Reference Euler angles are:\n";
1744 ss << eulerRef << "[deg] \n";
1745
1746 //! Save the obtained calibration coefficients in a file for later use
1747 QString fileName = QFileDialog::getSaveFileName(nullptr, "Save ground calibration file", "
referenceData.xml", "Calibration File (*.xml);;All Files (*)");
1748 FileStorage fs(fileName.toUtf8().constData(), FileStorage::WRITE);
1749 fs << "M_NC" << RmatRef;
1750 fs << "eulerRef" << eulerRef;
1751 strBuf = fs.releaseAndGetString();
1752 commObj.changeStatus(QString::fromStdString(strBuf));
1753 commObj.addLog("Saved ground calibration!");
1754 commObj.progressUpdate(0);
1755 return 0;
1756 }

```

Here is the call graph for this function:



### 3.2.2.2 closeUDP()

```
void closeUDP ( )
```

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

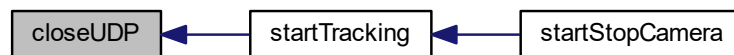
Definition at line 1191 of file main.cpp.

```

1192 {
1193     //! check if the socket is open and if yes close it
1194     if (udpSocketObject->isOpen())
1195     {
1196         udpSocketObject->close();
1197     }
1198
1199     if (udpSocketSafety->isOpen())
1200     {
1201         udpSocketSafety->close();
1202     }
1203
1204     if (udpSocketSafety2->isOpen())
1205     {
1206         udpSocketSafety2->close();
1207     }
1208     commObj.addLog("Closed all UDP ports.");
1209 }

```

Here is the caller graph for this function:



### 3.2.2.3 determineExposure()

```
int determineExposure ( )
```

Get the optimal exposure for the camera. For that find the minimum and maximum exposure were the right number of markers are detected. Then the mean of those two values is used as exposure.

Definition at line 1380 of file main.cpp.

```

1381 {
1382     //! For OptiTrack Ethernet cameras, it's important to enable development mode if you
1383     //! want to stop execution for an extended time while debugging without disconnecting
1384     //! the Ethernet devices. Lets do that now:
1385
1386     CameraLibrary_EnableDevelopment();
1387
1388     //! Initialize Camera SDK ===
1389     CameraLibrary::CameraManager::X();
1390
1391     //! At this point the Camera SDK is actively looking for all connected cameras and will initialize
1392     //! them on it's own.
1393
1394     //! Get a connected camera =====
1395     CameraManager::X().WaitForInitialization();
1396     Camera *camera = CameraManager::X().GetCamera();
1397
1398     //! If no device connected, pop a message box and exit ===
1399     if (camera == 0)

```

```

1400     {
1401         commObj.addLog("No camera found!");
1402         return 1;
1403     }
1404
1405     ///! Determine camera resolution to size application window =====
1406     int cameraWidth = camera->Width();
1407     int cameraHeight = camera->Height();
1408
1409     camera->SetVideoType(Core::PrecisionMode); ///! set the camera mode to precision mode, it used
greyscale information for marker property calculations
1410
1411         ///! Start camera output ===
1412     camera->Start();
1413
1414     ///! Turn on some overlay text so it's clear things are =====
1415     ///! working even if there is nothing in the camera's view. =====
1416     camera->SetTextOverlay(true);
1417     camera->SetExposure(intExposure); ///! set the camera exposure
1418     camera->SetIntensity(intIntensity); ///! set the camera infrared LED intensity
1419     camera->SetFrameRate(intFrameRate); ///! set the camera framerate to 100 Hz
1420     camera->SetIRFilter(true); ///! enable the filter that blocks visible light and only passes infrared
light
1421     camera->SetHighPowerMode(true); ///! enable high power mode of the leds
1422     camera->SetContinuousIR(false); ///! enable continuous LED light
1423     camera->SetThreshold(intThreshold); ///! set threshold for marker detection
1424
1425     ///!set exposure such that num markers are visible
1426     int numberObjects = 0; ///! Number of objects (markers) found in the current picture with the given
exposure
1427     int minExposure = 1; ///! exposure when objects detected the first time is numberMarkers
1428     int maxExposure = 480; ///! exposure when objects detected is first time numberMarkers+1
1429     intExposure = minExposure; ///! set the exposure to the smallest value possible
1430     int numberTries = 0; ///! if the markers arent found after numberTries then there might be no markers
at all in the real world
1431
1432         ///! Determine minimum exposure, hence when are numberMarkers objects detected
1433     camera->SetExposure(intExposure);
1434     while (numberObjects != numberMarkers && numberTries < 48)
1435     {
1436         ///! get a new camera frame
1437         Frame *frame = camera->GetFrame();
1438         if (frame) ///! frame received
1439         {
1440             numberObjects = frame->ObjectCount(); ///! how many objects are detected in the image
1441             if (numberObjects == numberMarkers) { minExposure =
intExposure; frame->Release(); break; } ///! if the right amount of markers is found, exit while
loop
1442             ///! not the right amount of markers was found so increase the exposure and try again
1443             numberTries++;
1444             intExposure += 10;
1445             camera->SetExposure(intExposure);
1446             ss.str("");
1447             ss << "Exposure: " << intExposure << "\t";
1448             ss << "Objects found: " << numberObjects;
1449             commObj.addLog(QString::fromStdString(ss.str()));
1450             frame->Release();
1451         }
1452     }
1453
1454     ///! Now determine maximum exposure, hence when are numberMarkers+1 objects detected
1455     numberTries = 0; ///! if the markers arent found after numberTries then there might be no markers at
all in the real world
1456     intExposure = maxExposure;
1457     camera->SetExposure(intExposure);
1458     numberObjects = 0;
1459     while (numberObjects != numberMarkers && numberTries < 48)
1460     {
1461         Frame *frame = camera->GetFrame();
1462         if (frame)
1463         {
1464             numberObjects = frame->ObjectCount(); ///! how many objects are detected in the image
1465             if (numberObjects == numberMarkers) { maxExposure =
intExposure; frame->Release(); break; } ///! if the right amount of markers is found, exit while
loop
1466             ///! not the right amount of markers was found so decrease the exposure and try again
1467             intExposure -= 10;
1468             numberTries++;
1469             camera->SetExposure(intExposure);
1470             ss.str("");
1471             ss << "Exposure: " << intExposure << "\t";
1472             ss << "Objects found: " << numberObjects;
1473             commObj.addLog(QString::fromStdString(ss.str()));
1474             frame->Release();
1475         }
1476     }
1477 }

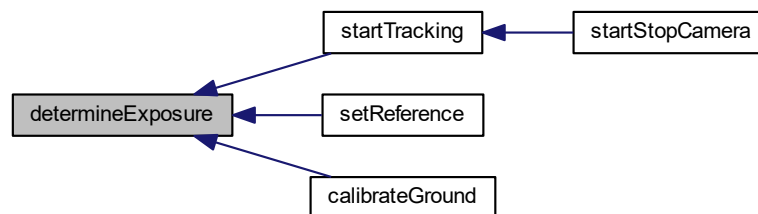
```

```

1478
1479     //! set the exposure to the mean of min and max exposure determined
1480     camera->SetExposure((minExposure + maxExposure) / 2.0);
1481
1482     //! and now check if the correct amount of markers is detected with that new value
1483     while (1)
1484     {
1485         Frame *frame = camera->GetFrame();
1486         if (frame)
1487         {
1488             numberObjects = frame->ObjectCount(); //! how many objects are detected in the image
1489             if (numberObjects != numberMarkers) //! are all markers and not more or less
1490             detected in the image
1491             {
1492                 frame->Release();
1493                 commObj.addLog("Was not able to detect the right amount of markers.");
1494                 //! Release camera ==--
1495                 camera->Release();
1496                 return 1;
1497             }
1498             else //! all markers and not more or less are found
1499             {
1500                 frame->Release();
1501                 intExposure = (minExposure + maxExposure) / 2.0;
1502                 commObj.addLog("Found the correct number of markers.");
1503                 commObj.addLog("Exposure set to:");
1504                 commObj.addLog(QString::number(intExposure));
1505                 break;
1506             }
1507         }
1508     }
1509     camera->Release();
1510     return 0;
1511 }
1512 }

```

Here is the caller graph for this function:



### 3.2.2.4 determineOrder()

```
void determineOrder ( )
```

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

Definition at line 1516 of file `main.cpp`.

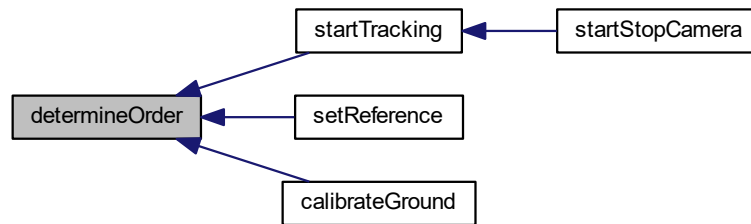
```

1517 {
1518     //! determine the 3D-2D correspondences that are crucial for the PnP algorithm
1519     //! Try every possible correspondence and solve PnP
1520     //! Then project the 3D marker points into the 2D camera image and check the difference
1521     //! between projected points and points as seen by the camera
1522     //! the correspondence with the smallest difference is probably the correct one
1523
1524     //! the difference between true 2D points and projected points is super big
1525     minPointDistance = 5000;
1526     std::sort(pointOrderIndices, pointOrderIndices + 4);
1527
1528     //! now try every possible permutation of correspondence
1529     do {
1530         //! reset the starting values for solvePnP
1531         Rvec = RvecOriginal;
1532         Tvec = TvecOriginal;
1533
1534         //! sort the 2d points with the current permutation
1535         for (int m = 0; m < numberMarkers; m++)
1536         {
1537             list_points2d[m] = list_points2dUnsorted[
1538                 pointOrderIndices[m]];
1539         }
1540
1541         //! Call solve PnP with P3P since its more robust and sufficient for start value determination
1542         solvePnP(list_points3d, list_points2d,
1543             cameraMatrix, distCoeffs, Rvec, Tvec, useGuess, SOLVEPNP_P3P);
1544
1545         //! set the current difference of all point correspondences to zero
1546         currentPointDistance = 0;
1547
1548         //! project the 3D points with the solvePnP solution onto 2D
1549         projectPoints(list_points3d, Rvec, Tvec,
1550             cameraMatrix, distCoeffs, list_points2dProjected);
1551
1552         //! now compute the absolute difference (error)
1553         for (int n = 0; n < numberMarkers; n++)
1554         {
1555             currentPointDistance += norm(list_points2d[n] -
1556                 list_points2dProjected[n]);
1557         }
1558
1559         //! if the difference with the current permutation is smaller than the smallest value till now
1560         //! it is probably the more correct permutation
1561         if (currentPointDistance < minPointDistance)
1562         {
1563             minPointDistance = currentPointDistance;    //!< set the
1564             smallest value of difference to the current one
1565             for (int b = 0; b < numberMarkers; b++)    //!< now save the better permutation
1566             {
1567                 pointOrderIndicesNew[b] = pointOrderIndices[b];
1568             }
1569         }
1570     }
1571     while (std::next_permutation(pointOrderIndices,
1572         pointOrderIndices + 4));
1573
1574     //! now that the correct order is found assign it to the indices array
1575     for (int w = 0; w < numberMarkers; w++)
1576     {
1577         pointOrderIndices[w] = pointOrderIndicesNew[w];
1578     }
1579     gotOrder = true;
1580 }

```



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

Definition at line 1333 of file main.cpp.

```

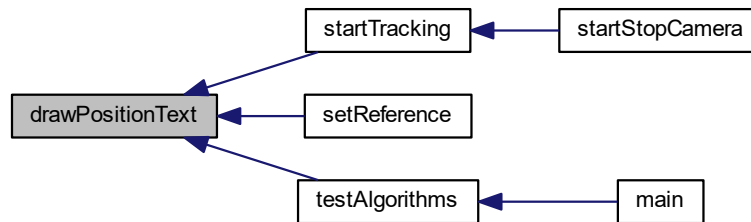
1334 {
1335     ss.str("");
1336     ss << "X: " << Position[0] << " m";
1337     putText(Picture, ss.str(), cv::Point(200, 440), 1, 1, cv::Scalar(255, 255, 255));
1338
1339     ss.str("");
1340     ss << "Y: " << Position[1] << " m";
1341     putText(Picture, ss.str(), cv::Point(200, 455), 1, 1, cv::Scalar(255, 255, 255));
1342
1343     ss.str("");
1344     ss << "Z: " << Position[2] << " m";
1345     putText(Picture, ss.str(), cv::Point(200, 470), 1, 1, cv::Scalar(255, 255, 255));
1346
1347     ss.str("");
1348     ss << "Heading: " << Euler[2]*180/3.1415 << " deg";
1349     putText(Picture, ss.str(), cv::Point(350, 440), 1, 1, cv::Scalar(255, 255, 255));
1350
1351     ss.str("");
1352     ss << "Pitch: " << Euler[1] * 180 / 3.1415 << " deg";
1353     putText(Picture, ss.str(), cv::Point(350, 455), 1, 1, cv::Scalar(255, 255, 255));
1354
1355     ss.str("");
  
```

```

1356     ss << "Roll: " << Euler[0] * 180 / 3.1415 << " deg";
1357     putText(Picture, ss.str(), cv::Point(350, 470), 1, 1, cv::Scalar(255, 255, 255));
1358
1359     ss.str("");
1360     ss << "Error: " << error << " px";
1361     putText(Picture, ss.str(), cv::Point(10, 470), 1, 1, cv::Scalar(255, 255, 255));
1362 }

```

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	<i>method</i>	whether or not load the camera calibration from calibration.xml. If ==0 then yes, if != 0 then let the user select a different file.
----	---------------	--

Definition at line 941 of file main.cpp.

```

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

```

```

961     commObj.addLog(fileName);
962     ss.str("");
963     ss << "\nCamera Matrix is" << "\n" << cameraMatrix << "\n";
964     ss << "\nDistortion Coefficients are" << "\n" << distCoeffs << "\n";
965     commObj.addLog(QString::fromStdString(ss.str()));
966 }

```

Here is the caller graph for this function:



### 3.2.2.7 loadCameraPosition()

```
void loadCameraPosition ( )
```

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

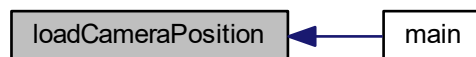
Definition at line 1366 of file main.cpp.

```

1367 {
1368     //! Open the referenceData.xml that contains the rotation from camera CoSy to ground CoSy
1369     FileStorage fs;
1370     fs.open("referenceData.xml", FileStorage::READ);
1371     fs["M_NC"] >> M_CN;
1372     fs["M_NC"] >> RmatRef;
1373     fs["posRef"] >> posRef;
1374     fs["eulerRef"] >> eulerRef;
1375     commObj.addLog("Loaded reference pose.");
1376 }

```

Here is the caller graph for this function:



### 3.2.2.8 loadMarkerConfig()

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

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

## Parameters

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

Definition at line 1213 of file main.cpp.

```

1214 {
1215     QString fileName;
1216     ///! during start up of the programm load the standard marker configuration
1217     if (method == 0)
1218     {
1219         ///! open the standard marker configuration file
1220         FileStorage fs;
1221         fs.open("markerStandard.xml", FileStorage::READ);
1222
1223         ///! copy the values to the respective variables
1224         fs["numberMarkers"] >> numberMarkers;
1225
1226         ///! initialize vectors with correct length depending on the number of markers
1227         list_points3d = std::vector<Point3d>(numberMarkers);
1228         list_points2d = std::vector<Point2d>(numberMarkers);
1229         list_points2dOld = std::vector<Point2d>(numberMarkers);
1230         list_points2dDifference = std::vector<double>(
numberMarkers);
1231         list_points2dProjected = std::vector<Point2d>(
numberMarkers);
1232         list_points2dUnsorted = std::vector<Point2d>(
numberMarkers);
1233
1234         ///! save the marker locations in the points3d vector
1235         fs["list_points3d"] >> list_points3d;
1236         fs.release();
1237         commObj.addLog("Loaded marker configuration from file:");
1238         commObj.addLog(fileName);
1239
1240
1241     }
1242     else
1243     {
1244         ///! if the load marker configuration button was clicked show a open file dialog
1245         fileName = QFileDialog::getOpenFileName(nullptr, "Choose a previous saved marker configuration file
", "", "marker configuratio files (*.xml);;All Files (*)");
1246
1247         ///! was cancel or abort clicked
1248         if (fileName.length() == 0)
1249         {
1250             ///! if yes load the standard marker configuration
1251             fileName = "markerStandard.xml";
1252         }
1253
1254         ///! open the selected marker configuration file
1255         FileStorage fs;
1256         fs.open(fileName.toUtf8().constData(), FileStorage::READ);
1257
1258         ///! copy the values to the respective variables
1259         fs["numberMarkers"] >> numberMarkers;
1260
1261         ///! initialize vectors with correct length depending on the number of markers
1262         list_points3d = std::vector<Point3d>(numberMarkers);
1263         list_points2d = std::vector<Point2d>(numberMarkers);
1264         list_points2dOld = std::vector<Point2d>(numberMarkers);
1265         list_points2dDifference = std::vector<double>(numberMarkers);
1266         list_points2dProjected = std::vector<Point2d>(numberMarkers);
1267         list_points2dUnsorted = std::vector<Point2d>(numberMarkers);
1268
1269         ///! save the marker locations in the points3d vector
1270         fs["list_points3d"] >> list_points3d;
1271         fs.release();
1272         commObj.addLog("Loaded marker configuration from file:");
1273         commObj.addLog(fileName);
1274
1275     }
1276 }
1277
1278 ///! Print out the number of markers and their position to the GUI
1279 ss.str("");
1280 ss << "Number of Markers: " << numberMarkers << "\n";
1281 ss << "Marker 3D Points X,Y and Z [mm]: \n";
1282 for (int i = 0; i < numberMarkers; i++)
1283 {

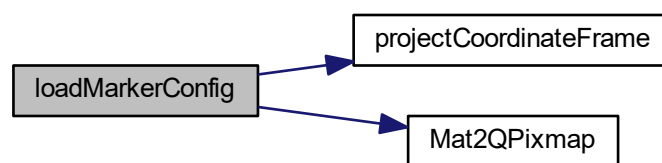
```

```

1284         ss << "Marker " << i + 1 << ":\t" << list_points3d[i].x << "\t" << list_points3d[i].y << "\t" <<
list_points3d[i].z << "\n";
1285     }
1286     commObj.addLog(QString::fromStdString(ss.str()));
1287
1288     //! check if P3P algorithm can be enabled, it needs exactly 4 marker points to work
1289     if (numberMarkers == 4)
1290     {
1291         //! if P3P is possible, let the user choose which algorithm he wants but keep iterative active
1292         methodPNP = 0;
1293         commObj.enableP3P(true);
1294     }
1295     else
1296     {
1297         //! More (or less) marker than 4 loaded, P3P is not possible, hence user cant select P3P in GUI
1298         methodPNP = 0;
1299         commObj.enableP3P(false);
1300         commObj.addLog("P3P algorithm disabled, only works with 4 markers.");
1301     }
1302
1303     //! now display the marker configuration in the camera view
1304     Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
1305
1306     //! Set the camera pose parallel to the marker coordinate system
1307     Tvec.at<double>(0) = 0;
1308     Tvec.at<double>(1) = 0;
1309     Tvec.at<double>(2) = 4500;
1310     Rvec.at<double>(0) = 0 * 3.141592653589 / 180.0;
1311     Rvec.at<double>(1) = 0 * 3.141592653589 / 180.0;
1312     Rvec.at<double>(2) = -90. * 3.141592653589 / 180.0;
1313
1314     projectPoints(list_points3d, Rvec, Tvec, cameraMatrix,
distCoeffs, list_points2dProjected);
1315     for (int i = 0; i < numberMarkers; i++)
1316     {
1317         circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 3, Scalar(255, 0, 0
), 3);
1318     }
1319
1320     projectCoordinateFrame(cFrame);
1321     QPixmap QPFrame;
1322     QPFrame = Mat2QPixmap(cFrame);
1323     commObj.changeImage(QPFrame);
1324     QApplication::processEvents();
1325
1326 }

```

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

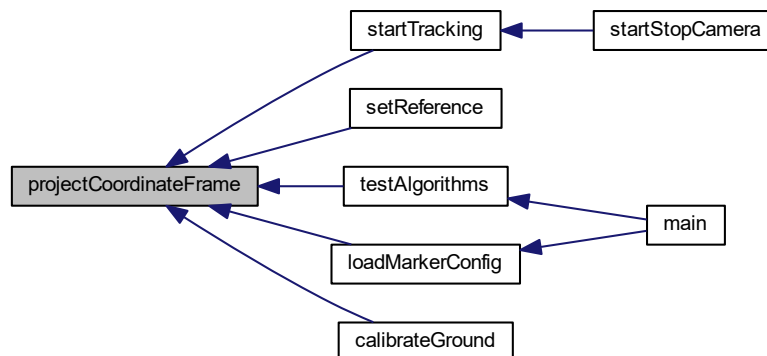
in	<i>pictureFrame</i>	the image in which the CoSy frame should be pasted.
----	---------------------	---

Definition at line 1099 of file main.cpp.

```
1100 {
1101     projectPoints(coordinateFrame, Rvec, Tvec,
1102                 cameraMatrix, distCoeffs, coordinateFrameProjected);
1102     line(pictureFrame, coordinateFrameProjected[0],
1103         coordinateFrameProjected[3], Scalar(0, 0, 255), 2); //!

```

Here is the caller graph for this function:



### 3.2.2.10 sendDataUDP()

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

Send the position and attitude over UDP to every receiver, the safety receiver is handled on its own in the start↔Tracking function because its send rate is less than 100 Hz.

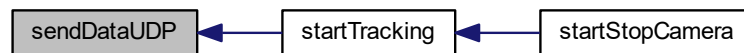
Definition at line 1172 of file main.cpp.

```

1173 {
1174     datagram.clear();
1175     QDataStream out(&datagram, QIODevice::WriteOnly);
1176     out.setVersion(QDataStream::Qt_4_3);
1177     out << (float)Position[0] << (float)Position[1] << (float)Position[2];
1178     out << (float)Euler[0] << (float)Euler[1] << (float)Euler[2]; ///! Roll Pitch Heading
1179     udpSocketObject->writeDatagram(datagram,
    IPAddressObject, portObject);
1180
1181     ///! if second receiver is activated send it also the tracking data
1182     if (safety2Enable)
1183     {
1184         udpSocketSafety2->writeDatagram(datagram,
    IPAddressSafety2, portSafety2);
1185     }
1186
1187 }

```

Here is the caller graph for this function:



### 3.2.2.11 setHeadingOffset()

```

void setHeadingOffset (
    double d )

```

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

#### Parameters

in	$d$	denotes heading offset in degrees.
----	-----	------------------------------------

Definition at line 1140 of file main.cpp.

```

1141 {
1142     headingOffset = d;
1143     d = d * 3.141592653589 / 180.0; ///! Convert heading offset from degrees to rad
1144
1145     ///! Calculate rotation about x axis
1146     Mat R_x = (Mat_<double>(3, 3) <<
1147         1, 0, 0,
1148         0, 1, 0,
1149         0, 0, 1
1150     );
1151
1152     ///! Calculate rotation about y axis
1153     Mat R_y = (Mat_<double>(3, 3) <<
1154         1, 0, 0,
1155         0, 1, 0,
1156         0, 0, 1
1157     );
1158
1159     ///! Calculate rotation about z axis
1160     Mat R_z = (Mat_<double>(3, 3) <<

```

```

1161         cos(d), -sin(d), 0,
1162         sin(d), cos(d), 0,
1163         0, 0, 1);
1164
1165
1166     //! Combined rotation matrix
1167     M_HeadingOffset = R_z * R_y * R_x;
1168 }

```

Here is the caller graph for this function:



### 3.2.2.12 setReference()

```
int setReference ( )
```

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

Definition at line 613 of file main.cpp.

```

614 {
615     //! initialize the variables with starting values
616     gotOrder = false;
617     posRef = 0;
618     eulerRef = 0;
619     RmatRef = 0;
620     Rvec = RvecOriginal;
621     Tvec = TvecOriginal;
622
623     determineExposure();
624
625     ss.str("");
626     commObj.addLog("Started reference coordinate determination.");
627
628     CameraLibrary_EnableDevelopment();
629     //! Initialize Camera SDK ---
630     CameraLibrary::CameraManager::X();
631
632     //! At this point the Camera SDK is actively looking for all connected cameras and will initialize
633     //! them on it's own.
634
635     //! Get a connected camera =====
636     CameraManager::X().WaitForInitialization();
637     Camera *camera = CameraManager::X().GetCamera();
638
639     //! If no device connected, pop a message box and exit ---
640     if (camera == 0)
641     {
642         commObj.addLog("No camera found!");
643         return 1;
644     }
645
646     //! Determine camera resolution to size application window -----
647     int cameraWidth = camera->Width();
648     int cameraHeight = camera->Height();
649     camera->GetDistortionModel(distModel);

```



```

650     cv::Mat matFrame(cv::Size(cameraWidth, cameraHeight), CV_8UC1);
651
652     ///! Set camera mode to precision mode, it directly provides marker coordinates
653     camera->SetVideoType(Core::PrecisionMode);
654
655     ///! Start camera output ===
656     camera->Start();
657
658     ///! Turn on some overlay text so it's clear things are =====
659     ///! working even if there is nothing in the camera's view. =====
660     ///! Set some other parameters as well of the camera
661     camera->SetTextOverlay(true);
662     camera->SetFrameRate(intFrameRate);
663     camera->SetIntensity(intIntensity);
664     camera->SetIRFilter(true);
665     camera->SetContinuousIR(false);
666     camera->SetHighPowerMode(false);
667
668     ///! sample some frames and calculate the position and attitude. then average those values and use that
    as zero position
669     int numberSamples = 0;
670     int numberToSample = 200;
671     double projectionError = 0; ///!< difference between the marker points as seen by the camera and the
    projected marker points with Rvec and Tvec
672
673     while (numberSamples < numberToSample)
674     {
675         ///! Fetch a new frame from the camera =====
676         Frame *frame = camera->GetFrame();
677
678         if (frame)
679         {
680             ///! Ok, we've received a new frame, lets do something
681             ///! with it.
682             if (frame->ObjectCount() == numberMarkers)
683             {
684                 ///!for(int i=0; i<frame->ObjectCount(); i++)
685                 for (int i = 0; i < numberMarkers; i++)
686                 {
687                     cObject *obj = frame->Object(i);
688                     list_points2dUnsorted[i] = cv::Point2d(obj->X(), obj->Y());
689                 }
690
691                 if (gotOrder == false)
692                 {
693                     determineOrder();
694                 }
695
696                 ///! sort the 2d points with the correct indices as found in the preceeding order
        determination algorithm
697                 for (int w = 0; w < numberMarkers; w++)
698                 {
699                     list_points2d[w] = list_points2dUnsorted[
pointOrderIndices[w]];
700                 }
701                 list_points2dOld = list_points2dUnsorted;
702
703                 ///!Compute the pose from the 3D-2D correspondences
704                 solvePnP(list_points3d, list_points2d,
cameraMatrix, distCoeffs, Rvec, Tvec, useGuess,
methodPNP);
705
706                 ///! project the marker 3d points with the solution into the camera image CoSy and calculate
        difference to true camera image
707                 projectPoints(list_points3d, Rvec, Tvec,
cameraMatrix, distCoeffs, list_points2dProjected);
708                 projectionError = norm(list_points2dProjected,
list_points2d);
709
710                 double maxValue = 0;
711                 double minValue = 0;
712                 minMaxLoc(Tvec.at<double>(2), &minValue, &maxValue);
713
714                 if (maxValue > 10000 || minValue < 0)
715                 {
716                     ss.str("");
717                     ss << "Negative z distance, thats not possible. Start the set zero routine again or
restart Programm.";
718                     commObj.addLog(QString::fromStdString(ss.str()));
719                     frame->Release();
720                     return 1;
721                 }
722
723                 if (projectionError > 5)
724                 {
725                     commObj.addLog("Reprojection error is bigger than 5 pixel. Correct marker
configuration loaded?\nMarker position measured precisely?");

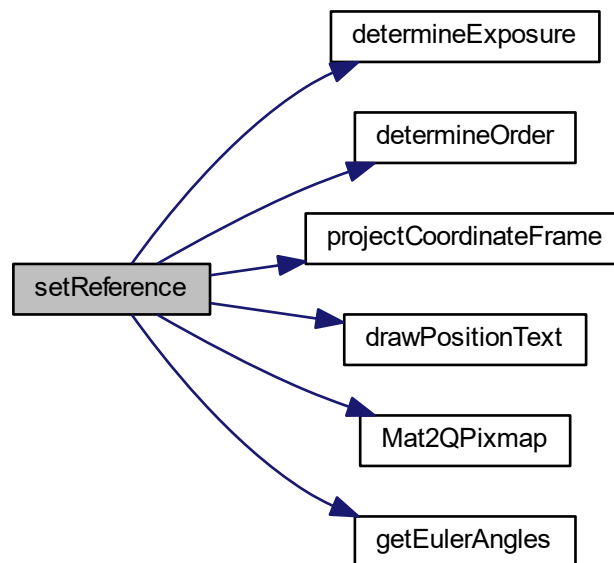
```

```

726         frame->Release();
727         return 1;
728     }
729
730     if (norm(positionOld) - norm(Tvec) < 0.05)    //!< Iterative Method needs time
to converge to solution
731     {
732         add(posRef, Tvec, posRef);
733         add(eulerRef, Rvec, eulerRef); //!< That are not the values of yaw,
roll and pitch yet! Rodriguez has to be called first.
734         numberSamples++;    //!< one sample more :D
735         commObj.progressUpdate(numberSamples * 100 / numberToSample);
736     }
737     positionOld = Tvec;
738
739     Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
740     for (int i = 0; i < numberMarkers; i++)
741     {
742         circle(cFrame, Point(list_points2d[i].x,
list_points2d[i].y), 6, Scalar(0, 225, 0), 3);
743     }
744     projectCoordinateFrame(cFrame);
745     projectPoints(list_points3d, Rvec, Tvec,
cameraMatrix, distCoeffs, list_points2d);
746     for (int i = 0; i < numberMarkers; i++)
747     {
748         circle(cFrame, Point(list_points2d[i].x,
list_points2d[i].y), 3, Scalar(225, 0, 0), 3);
749     }
750     drawPositionText(cFrame, position,
eulerAngles, projectionError);
751
752     QPixmap QPFrame;
753     QPFrame = Mat2QPixmap(cFrame);
754     commObj.changeImage(QPFrame);
755     QApplication::processEvents();
756
757     }
758     frame->Release();
759 }
760 }
761 //!< Release camera ===
762 camera->Release();
763
764 //!< Divide by the number of samples to get the mean of the reference position
765 divide(posRef, numberToSample, posRef);
766 divide(eulerRef, numberToSample, eulerRef); //!< eulerRef is here in Axis Angle
notation
767
768 Rodrigues(eulerRef, RmatRef);    //!< axis angle to rotation matrix
769 //!<-- Euler Angles, finally
770 getEulerAngles(RmatRef, eulerRef); //!< rotation matrix to euler
771 ss.str("");
772 ss << "RmatRef is:\n";
773 ss << RmatRef << "\n";
774 ss << "Reference Position is:\n";
775 ss << posRef << "[mm] \n";
776 ss << "Reference Euler Angles are:\n";
777 ss << eulerRef << "[deg] \n";
778
779 //!< compute the difference between last obtained Tvec and the average Value
780 //!< When it is large the iterative method has not converged properly so it is advised to start the
setReference() function once again
781 double error = norm(posRef) - norm(Tvec);
782 if (error > 5.0)
783 {
784     ss << "Caution, distance between reference position and last position is: " << error << "\n Start
the set zero routine once again.";
785 }
786 commObj.addLog(QString::fromStdString(ss.str()));
787 commObj.progressUpdate(0);
788 return 0;
789 }

```

Here is the call graph for this function:



### 3.2.2.13 startTracking()

```
int startTracking ( )
```

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

Definition at line 276 of file main.cpp.

```

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

```

```

293     /// the Ethernet devices. Lets do that now:
294
295     CameraLibrary_EnableDevelopment();
296     CameraLibrary::CameraManager::X(); /// Initialize Camera SDK
297
298     /// At this point the Camera SDK is actively looking for all connected cameras and will initialize
299     /// them on it's own
300
301     /// Get a connected camera
302     CameraManager::X().WaitForInitialization();
303     Camera *camera = CameraManager::X().GetCamera();
304
305     /// If no camera can be found, inform user in message log and exit function
306     if (camera == 0)
307     {
308         commObj.addLog("No camera found!");
309         return 1;
310     }
311
312     /// Determine camera resolution to size application window
313     int cameraWidth = camera->Width();
314     int cameraHeight = camera->Height();
315
316     camera->SetVideoType(Core::PrecisionMode); /// Set the camera mode to precision mode, it used
    greyscale information for marker property calculations
317
318     camera->Start(); /// Start camera output
319
320     /// Turn on some overlay text so it's clear things are
321     /// working even if there is nothing in the camera's view
322     camera->SetTextOverlay(true);
323     camera->SetExposure(intExposure); /// Set the camera exposure
324     camera->SetIntensity(intIntensity); /// Set the camera infrared LED intensity
325     camera->SetFrameRate(intFrameRate); /// Set the camera framerate to 100 Hz
326     camera->SetIRFilter(true); /// Enable the filter that blocks visible light and only passes infrared
    light
327     camera->SetHighPowerMode(true); /// Enable high power mode of the LEDs
328     camera->SetContinuousIR(false); /// Disable continuous LED light
329     camera->SetThreshold(intThreshold); /// Set threshold for marker detection
330
331     /// Create a new matrix that stores the grayscale picture from the camera
332     Mat matFrame = Mat::zeros(cv::Size(cameraWidth, cameraHeight), CV_8UC1);
333     QPixmap QPFrame; /// QPixmap is the corresponding Qt class that saves images
334     /// Matrix that stores the colored picture, hence marker points, coordinate frame and reprojected
    points
335     Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
336
337     int v = 0; /// Helper variable used to kick safety switch
338     /// Variables for the min and max values that are needed for sanity checks
339     double maxValue = 0;
340     double minValue = 0;
341     int framesDropped = 0; /// If a marker is not visible or accuracy is bad increase this counter
342     double projectionError = 0; /// Equals the quality of the tracking
343
344     setUpUDP(); /// Open sockets and ports for UDP communication
345
346     if (safetyEnable) /// If the safety feature is enabled send the starting message
347     {
348         /// Send enable message, hence send a 9 and then a 1
349         data.setNum((int)(9));
350         udpSocketSafety->write(data);
351         data.setNum((int)(1));
352         udpSocketSafety->write(data);
353     }
354
355     /// Fetch a new frame from the camera
356     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
357     while (!gotTime) /// While no new frame is received loop
358     {
359         Frame *frame = camera->GetFrame(); /// Get a new camera frame
360         if (frame) /// There is actually a new frame
361         {
362             timeFirstFrame = frame->TimeStamp(); /// Get the time stamp for the first frame.
    It is subtracted for the following frames
363             frame->Release(); /// Release the frame so the camera can continue
364             gotTime = true; /// Exit the while loop
365         }
366     }
367
368     /// Now enter the main loop that processes each frame and computes the pose, sends it and logs stuff
369     while (!exitRequested) /// Check if the user has not pressed "Stop Tracking" yet
370     {
371
372         Frame *frame = camera->GetFrame(); /// Fetch a new frame from the camera
373
374         if (frame) /// Did we got a new frame or does the camera still need more time

```

```

375         {
376             framesDropped++; ///! Increase by one, if everything is okay it is decreased at the end of the
loop again
377
378             ///! Only use this frame if the right number of markers is found in the picture
379             if (frame->ObjectCount() == numberMarkers)
380             {
381                 ///! Get the marker points in 2D in the camera image frame and store them in the
list_points2dUnsorted vector
382                 ///! The order of points that come from the camera corresponds to the Y coordinate
383                 for (int i = 0; i < numberMarkers; i++)
384                 {
385                     cObject *obj = frame->Object(i);
386                     list_points2dUnsorted[i] = cv::Point2d(obj->X(), obj->Y());
387                 }
388
389                 if (gotOrder == false) ///! Was the order already determined? This is false for the
first frame and from then on true
390                 {
391                     determineOrder(); ///! Now compute the order
392                 }
393
394                 ///! Sort the 2d points with the correct indices as found in the preceeding order
determination algorithm
395                 for (int w = 0; w < numberMarkers; w++)
396                 {
397                     list_points2d[w] = list_points2dUnsorted[
pointOrderIndices[w]]; ///! pointOrderIndices was calculated in determineOrder()
398                 }
399
400                 ///! The first time the 2D-3D corresspondence was determined with gotOrder was okay.
///! But this order can change as the object moves and the marker objects appear in a
///! different order in the frame->Object() array.
401                 ///! The solution is that: When a marker point (in the camera image, hence in 2D) was at
///! a position then it wont move that much from one frame to the other.
402                 ///! So for the new frame we take a marker object and check which marker was closest this
point
403                 ///! in the old image frame? This is probably the same (true) marker. And we do that for
every other marker as well.
404                 ///! When tracking is good and no frames are dropped because of missing markers this should
work every frame.
405                 for (int j = 0; j < numberMarkers; j++)
406                 {
407                     minPointDistance = 5000; ///! The sum of point distances is set to
something unrealistic large
408                     for (int k = 0; k < numberMarkers; k++)
409                     {
410                         ///! Calculate N_2 norm of unsorted points minus old points
411                         currentPointDistance = norm(
list_points2dUnsorted[pointOrderIndices[j]] -
list_points2dOld[k]);
412                     }
413                     ///! If the norm is smaller than minPointDistance the correspondence is more likely
to be correct
414                     if (currentPointDistance <
minPointDistance)
415                     {
416                         ///! Update the array that saves the new point order
417                         minPointDistance =
currentPointDistance;
418                         pointOrderIndicesNew[j] = k;
419                     }
420                 }
421
422                 ///! Now the new order is found, set the point order to the new value
423                 for (int k = 0; k < numberMarkers; k++)
424                 {
425                     pointOrderIndices[k] = pointOrderIndicesNew[k];
426                     list_points2d[k] = list_points2dUnsorted[
pointOrderIndices[k]];
427                 }
428
429                 ///! Save the unsorted position of the marker points for the next loop
list_points2dOld = list_points2dUnsorted;
430
431                 ///! Compute the object pose from the 3D-2D correspondes
solvePnP(list_points3d, list_points2d,
cameraMatrix, distCoeffs, Rvec, Tvec, useGuess,
methodPNP);
432
433                 ///! Project the marker 3d points with the solution into the camera image CoSy and calculate
difference to true camera image
434                 projectPoints(list_points3d, Rvec, Tvec,
cameraMatrix, distCoeffs, list_points2dProjected);
435                 projectionError = norm(list_points2dProjected,
list_points2d); ///! Difference of true pose and found pose
436
437
438
439
440
441

```

```

442         //!< Increase the framesDropped variable if accuracy of tracking is too bad
443         if (projectionError > 5)
444         {
445             framesDropped++;
446         }
447         else
448         {
449             framesDropped = 0; //!< Set number of subsequent frames dropped to zero because error
is small enough and no marker was missing
450         }
451
452         //!< Get the min and max values from Tvec for sanity check
453         minMaxLoc(Tvec.at<double>(2), &minValue, &maxValue);
454
455         //!< Sanity check of values. negative z means the marker CoSy is behind the camera, that's
not possible.
456         if (minValue < 0)
457         {
458             commObj.addLog("Negative z distance, that is not possible. Start the set zero
routine again or restart Program.");
459             frame->Release(); //!< Release the frame so the camera can move on
460             camera->Release(); //!< Release the camera
461             closeUDP(); //!< Close all UDP connections so the programm can be closed later
on and no resources are locked
462             return 1; //!< Exit the function
463         }
464
465         //!< Next step is the transformation from camera CoSy to navigation CoSy
466         //!< Compute the relative object position from the reference position to the current one
467         //!< given in the camera CoSy: \f$ T_C^{NM} = Tvec - Tvec_{Ref} \f$
468         subtract(Tvec, posRef, position);
469
470         //!< Transform the position from the camera CoSy to the navigation CoSy with INS aligned
heading and convert from [mm] to [m]
471         //!< \f$ T_N^{NM} = M_{NC} \times T_C^{NM} \f$
472         Mat V = 0.001 * M_HeadingOffset * M_CN.t() * (Mat)
position;
473         position = V; //!< Position is the result of the preceeding calculation
474         position[2] *= invertZ; //!< Invert Z if check box in GUI is activated,
hence height above ground is considered
475
476         //!< Realtime angle between reference orientation and current orientation
477         Rodrigues(Rvec, Rmat); //!< Convert axis angle representation to ordinary rotation
matrix
478
479         //!< The difference of the reference rotation and the current rotation
480         //!< \f$ R_{NM} = M_{NC} \times R_{CM} \f$
481         Rmat = RmatRef.t() * Rmat;
482
483         //!< Euler Angles, finally
484         getEulerAngles(Rmat, eulerAngles); //!< Get the euler angles
from the rotation matrix
485         eulerAngles[2] += headingOffset; //!< Add the heading offset to the
heading angle
486
487         //!< 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
488         frameTime = frame->TimeStamp() - timeOld; //!< Time between the old frame
and the current frame
489         timeOld = frame->TimeStamp(); //!< Set the old frame time to the current one
490         velocity[0] = (position[0] - positionOld[0]) /
frameTime; //!< Calculate the x velocity with finite differences
491         velocity[1] = (position[1] - positionOld[1]) /
frameTime; //!< Calculate the y velocity with finite differences
492         velocity[2] = (position[2] - positionOld[2]) /
frameTime; //!< Calculate the z velocity with finite differences
493         positionOld = position; //!< Set the old position to the current one for
next frame velocity calculation
494
495         eulerAngles[0] = eulerAngles[0] * -3.141592653589 / 180.0; //!<
Convert the Euler angles from degrees to rad
496         eulerAngles[1] = eulerAngles[1] * -3.141592653589 / 180.0;
497         eulerAngles[2] = eulerAngles[2] * 3.141592653589 / 180.0;
498
499         //!< Send position and Euler angles over WiFi with 100 Hz
500         sendDataUDP(position, eulerAngles);
501
502         //!< Save the values in a log file, values are:
503         //!< Time sinc tracking started Position Euler Angles Velocity
504         logfile.open(logName, std::ios::app); //!< Open the log file, the folder is
RigidTrackInstallationFolder/logs
505         logfile << frame->TimeStamp() - timeFirstFrame << ";" <<
position[0] << ";" << position[1] << ";" << position[2] << ";";
506         logfile << eulerAngles[0] << ";" <<
eulerAngles[1] << ";" << eulerAngles[2] << ";";
507         logfile << velocity[0] << ";" << velocity[1] << ";" <<
velocity[2] << "\n";

```

```

508         logfile.close(); //! Close the file to save values
509     }
510
511     //! Check if the position and euler angles are below the allowed value, if yes send OKAY signal
    (1), if not send shutdown signal (0)
512     //! Absolute x, y and z position in navigation CoSy must be smaller than the allowed distance
513     if (safetyEnable)
514     {
515         if ((abs(position[0]) < safetyBoxLength && abs(position[1]) <
safetyBoxLength && abs(position[2]) < safetyBoxLength))
516         {
517             //! Absolute Euler angles must be smaller than allowed value. Heading is not considered
518             if ((abs(eulerAngles[0]) < safetyAngle && abs(eulerAngles[1]) <
safetyAngle))
519             {
520                 //! Send the OKAY signal to the desired computer every 5th time
521                 if (v == 5) {
522                     data.setNum((int)(1));
523                     udpSocketSafety->write(data); //! Send the 1
524                     v = 0; //! reset the counter that is needed for decimation to every 5th time
                    step
                    }
525             }
526             //! The euler angles of the object exceeded the allowed euler angles, send the shutdown
527             signal (0)
528             else
529             {
530                 data.setNum((int)(0)); //! Send the shutdown signal, a 0
531                 udpSocketSafety->write(data);
532                 commObj.addLog("Object exceeded allowed Euler angles, shutdown signal sent."
); //! Inform the user
533             }
534         }
535     }
536     //! The position of the object exceeded the allowed position, shut the object down
537     else
538     {
539         data.setNum((int)(0)); //! Send the shutdown signal, a 0
540         udpSocketSafety->write(data);
541         commObj.addLog("Object left allowed area, shutdown signal sent."); //! Inform
the user
542     }
543 }
544 }
545
546 //! Inform the user if tracking system is disturbed (marker lost or so) or error was too big
547 if (framesDropped > 10)
548 {
549     if (safetyEnable) //! Also send the shutdown signal
550     {
551         data.setNum((int)(0)); //! Send the shutdown signal, a 0
552         udpSocketSafety->write(data);
553     }
554     commObj.addLog("Lost marker points or precision was bad!"); //! Inform the user
555     framesDropped = 0;
556 }
557
558 //! Rasterize the frame so it can be shown in the GUI
559 frame->Rasterize(cameraWidth, cameraHeight, matFrame.step,
BACKBUFFER_BITS_PER_PIXEL, matFrame.data);
560
561 //! Convert the frame from greyscale as it comes from the camera to rgb color
562 cvtColor(matFrame, cFrame, COLOR_GRAY2RGB);
563
564 //! Project (draw) the marker CoSy origin into 2D and save it in the cFrame image
565 projectCoordinateFrame(cFrame);
566
567 //! Project the marker points from 3D to the camera image frame (2d) with the computed pose
568 projectPoints(list_points3d, Rvec, Tvec,
cameraMatrix, distCoeffs, list_points2d);
569 for (int i = 0; i < numberMarkers; i++)
570 {
571     //! Draw a circle around the projected points so the result can be better compared to the
real marker position
572     //! In the resulting picture those are the red dots
573     circle(cFrame, Point(list_points2d[i].x,
list_points2d[i].y), 3, Scalar(225, 0, 0), 3);
574 }
575
576 //! Write the current position, attitude and error values as text in the frame
577 drawPositionText(cFrame, position, eulerAngles, projectionError);
578
579 //! Send the new camera picture to the GUI and call the GUI processing routine
580 QPixmap QPFrame;
581 QPFrame = Mat2QPixmap(cFrame);
582 commObj.changeImage(QPFrame); //! Update the picture in the GUI

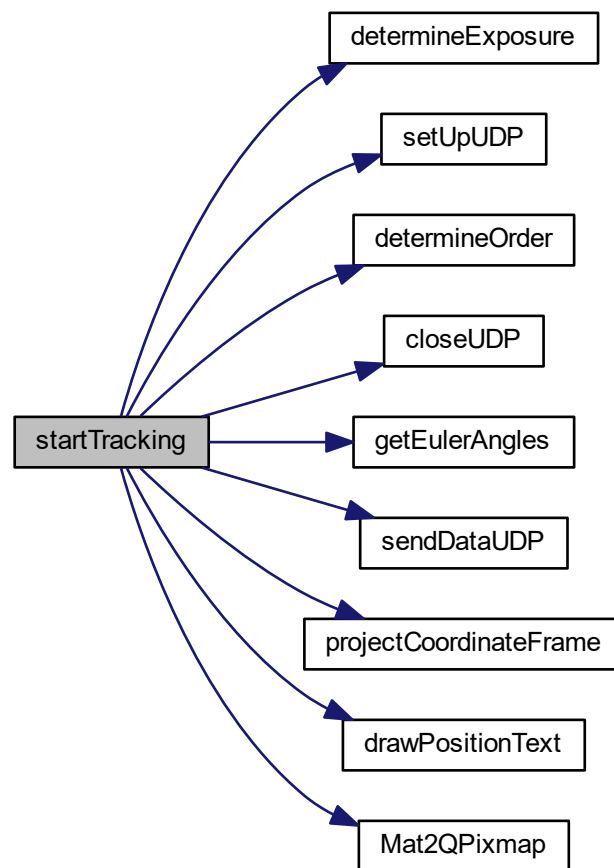
```

```

583         QApplication::processEvents(); /// Give Qt time to handle everything
584
585         /// Release the camera frame to fetch the new one
586         frame->Release();
587     }
588 }
589
590 /// User choose to stop the tracking, clean things up
591 closeUDP(); /// Close the UDP connections so resources are deallocated
592 camera->Release(); /// Release camera
593 return 0;
594 }

```

Here is the call graph for this function:



Here is the caller graph for this function:





## 3.2.2.14 testAlgorithms()

```
void testAlgorithms ( )
```

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

Definition at line 970 of file main.cpp.

```

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

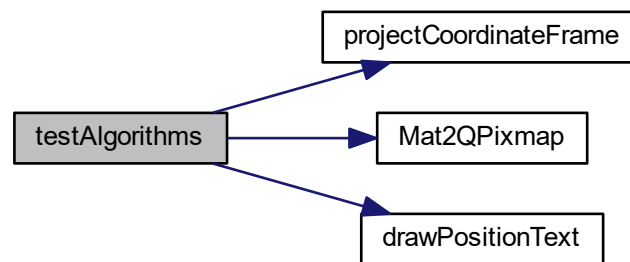
```

```

1035
1036     projectPoints(list_points3d, Rvec, Tvec, cameraMatrix,
distCoeffs, list_points2dProjected);
1037     for (int i = 0; i < numberMarkers; i++)
1038     {
1039         circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 3, Scalar(255, 0, 0
), 3);
1040     }
1041     QPixmap QPFrame;
1042     QPFrame = Mat2QPixmap(cFrame);
1043     commObj.changeImage(QPFrame);
1044     QApplication::processEvents();
1045     commObj.addLog(QString::fromStdString(ss.str()));
1046     if (numberMarkers == 4)
1047     {
1048         _methodPNP = 2; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP //!< not used
1049         Rvec = cv::Mat::zeros(3, 1, CV_64F);
1050         Tvec = cv::Mat::zeros(3, 1, CV_64F);
1051         solvePnP(list_points3d, list_points2dProjected,
cameraMatrix, distCoeffs, Rvec, Tvec, useGuess, _methodPNP);
1052
1053         ss.str("");
1054         ss << "=====\n";
1055         ss << "===== P3P =====\n";
1056         ss << "rvec: " << "\n";
1057         ss << Rvec << "\n";
1058         ss << "tvec: " << "\n";
1059         ss << Tvec << "\n";
1060
1061         projectPoints(list_points3d, Rvec, Tvec, cameraMatrix,
distCoeffs, list_points2dProjected);
1062         for (int i = 0; i < numberMarkers; i++)
1063         {
1064             circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 3, Scalar(255,
0, 0), 3);
1065         }
1066         double projectionError = norm(list_points2dProjected, list_points2d);
1067         putText(cFrame, "Testing Algorithms Finished", cv::Point(5, 420), 1, 1, cv::Scalar(255, 255, 255));
1068         drawPositionText(cFrame, position, eulerAngles, projectionError)
;
1069
1070         QPixmap QPFrame;
1071         QPFrame = Mat2QPixmap(cFrame);
1072         commObj.changeImage(QPFrame);
1073         QApplication::processEvents();
1074         commObj.addLog(QString::fromStdString(ss.str()));
1075     }
1076
1077     _methodPNP = 4; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP //!< not used
1078     Rvec = cv::Mat::zeros(3, 1, CV_64F);
1079     Tvec = cv::Mat::zeros(3, 1, CV_64F);
1080     solvePnP(list_points3d, list_points2dProjected, cameraMatrix,
distCoeffs, Rvec, Tvec, useGuess, _methodPNP);
1081
1082     ss.str("");
1083     ss << "=====\n";
1084     ss << "===== UPNP =====\n";
1085     ss << "rvec: " << "\n";
1086     ss << Rvec << "\n";
1087     ss << "tvec: " << "\n";
1088     ss << Tvec << "\n";
1089
1090     commObj.addLog(QString::fromStdString(ss.str()));
1091
1092     Rvec = RvecOriginal;
1093     Tvec = TvecOriginal;
1094
1095 }

```

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

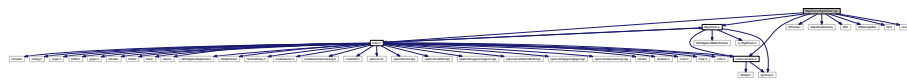
Now declare variables that are used across the [main.cpp](#) file. Basically almost every variable used is declared here.

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

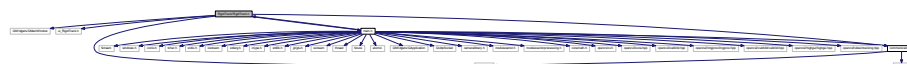
##### Date

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:
```







# Index

- calcBoardCornerPositions
  - main.cpp, [9](#)
- calibrateGround
  - main.cpp, [10](#)
  - main.h, [43](#)
- closeUDP
  - main.cpp, [13](#)
  - main.h, [46](#)
- commObj
  - main.cpp, [40](#)
  - main.h, [69](#)
- determineExposure
  - main.cpp, [14](#)
  - main.h, [47](#)
- determineOrder
  - main.cpp, [16](#)
  - main.h, [49](#)
- drawPositionText
  - main.cpp, [17](#)
  - main.h, [51](#)
- getEulerAngles
  - main.cpp, [18](#)
- loadCalibration
  - main.cpp, [19](#)
  - main.h, [52](#)
- loadCameraPosition
  - main.cpp, [20](#)
  - main.h, [53](#)
- loadMarkerConfig
  - main.cpp, [21](#)
  - main.h, [53](#)
- main
  - main.cpp, [23](#)
- main.cpp
  - calcBoardCornerPositions, [9](#)
  - calibrateGround, [10](#)
  - closeUDP, [13](#)
  - commObj, [40](#)
  - determineExposure, [14](#)
  - determineOrder, [16](#)
  - drawPositionText, [17](#)
  - getEulerAngles, [18](#)
  - loadCalibration, [19](#)
  - loadCameraPosition, [20](#)
  - loadMarkerConfig, [21](#)
  - main, [23](#)
  - Mat2QPixmap, [25](#)
  - projectCoordinateFrame, [26](#)
  - Rmat, [40](#)
  - sendDataUDP, [27](#)
  - setHeadingOffset, [27](#)
  - setReference, [28](#)
  - startTracking, [31](#)
  - testAlgorithms, [37](#)
- main.h
  - calibrateGround, [43](#)
  - closeUDP, [46](#)
  - commObj, [69](#)
  - determineExposure, [47](#)
  - determineOrder, [49](#)
  - drawPositionText, [51](#)
  - loadCalibration, [52](#)
  - loadCameraPosition, [53](#)
  - loadMarkerConfig, [53](#)
  - projectCoordinateFrame, [56](#)
  - sendDataUDP, [56](#)
  - setHeadingOffset, [57](#)
  - setReference, [58](#)
  - startTracking, [61](#)
  - testAlgorithms, [67](#)
- Mat2QPixmap
  - main.cpp, [25](#)
- projectCoordinateFrame
  - main.cpp, [26](#)
  - main.h, [56](#)
- RigidTrack/RigidTrack.cpp, [70](#)
- RigidTrack/RigidTrack.h, [70](#)
- RigidTrack/main.cpp, [5](#)
- RigidTrack/main.h, [41](#)
- Rmat
  - main.cpp, [40](#)
- sendDataUDP
  - main.cpp, [27](#)
  - main.h, [56](#)
- setHeadingOffset
  - main.cpp, [27](#)
  - main.h, [57](#)
- setReference
  - main.cpp, [28](#)
  - main.h, [58](#)
- startTracking
  - main.cpp, [31](#)
  - main.h, [61](#)

testAlgorithms  
  main.cpp, [37](#)  
  main.h, [67](#)