

Rigid Track

Generated by Doxygen 1.8.13

Contents

1	Rigid Track Doxygen Documentation	1
1.1	Introduction	1
1.2	Rigid Track Installation	1
1.3	Source Code	1
2	Hierarchical Index	3
2.1	Class Hierarchy	3
3	Class Index	5
3.1	Class List	5
4	File Index	7
4.1	File List	7
5	Class Documentation	9
5.1	commObject Class Reference	9
5.1.1	Detailed Description	9
5.1.2	Member Function Documentation	9
5.1.2.1	addLog()	10
5.1.2.2	changeImage()	10
5.1.2.3	changeStatus()	11
5.1.2.4	clearLog()	11
5.1.2.5	enableP3P()	12
5.1.2.6	imageChanged	12
5.1.2.7	logAdded	12

5.1.2.8	logCleared	13
5.1.2.9	P3Penabled	13
5.1.2.10	progressUpdate()	14
5.1.2.11	progressUpdated	14
5.1.2.12	statusChanged	15
5.2	CvModelEstimator2 Class Reference	15
5.2.1	Detailed Description	16
5.2.2	Constructor & Destructor Documentation	16
5.2.2.1	CvModelEstimator2()	16
5.2.2.2	~CvModelEstimator2()	16
5.2.3	Member Function Documentation	16
5.2.3.1	checkSubset()	16
5.2.3.2	computeReprojError()	17
5.2.3.3	findInliers()	17
5.2.3.4	getSubset()	17
5.2.3.5	refine()	17
5.2.3.6	runKernel()	18
5.2.3.7	runLMeDS()	18
5.2.3.8	runRANSAC()	18
5.2.3.9	setSeed()	18
5.2.4	Member Data Documentation	18
5.2.4.1	checkPartialSubsets	18
5.2.4.2	maxBasicSolutions	19
5.2.4.3	modelPoints	19
5.2.4.4	modelSize	19
5.2.4.5	rng	19
5.3	RigidTrack Class Reference	19
5.3.1	Detailed Description	20
5.3.2	Constructor & Destructor Documentation	20
5.3.2.1	RigidTrack()	20

5.3.3	Member Function Documentation	21
5.3.3.1	clearLog	21
5.3.3.2	enableP3P	21
5.3.3.3	on_actionAbout_Rigid_Track_triggered	21
5.3.3.4	on_actionOpen_Installation_Folder_triggered	22
5.3.3.5	on_actionOpen_Log_Folder_triggered	22
5.3.3.6	on_actionShow_Help_triggered	22
5.3.3.7	on_btnCalibrate_clicked	23
5.3.3.8	on_btnCalibrateGround_clicked	23
5.3.3.9	on_btnLoadCalib_clicked	24
5.3.3.10	on_btnStartCamera_clicked	25
5.3.3.11	on_btnZero_clicked	25
5.3.3.12	on_cbInvert_stateChanged	26
5.3.3.13	on_cbSafety2_stateChanged	26
5.3.3.14	on_cbSafety_stateChanged	27
5.3.3.15	on_dsbDimension_valueChanged	28
5.3.3.16	on_leIPObject_returnPressed	28
5.3.3.17	on_leIPSafety2_returnPressed	29
5.3.3.18	on_leIPSafety_returnPressed	30
5.3.3.19	on_pbLoadMarker_clicked	31
5.3.3.20	on_rbEPnP_clicked	31
5.3.3.21	on_rblterative_clicked	32
5.3.3.22	on_rbP3P_clicked	32
5.3.3.23	on_sbAngle_valueChanged	33
5.3.3.24	on_sbHeadingOffset_valueChanged	33
5.3.3.25	progressUpdate	33
5.3.3.26	setImage	34
5.3.3.27	setLog	34
5.4	Surface Class Reference	34
5.4.1	Detailed Description	35

5.4.2	Constructor & Destructor Documentation	35
5.4.2.1	Surface()	35
5.4.2.2	~Surface()	36
5.4.3	Member Function Documentation	36
5.4.3.1	CalculateSize()	36
5.4.3.2	GetBuffer()	37
5.4.3.3	GetTexture()	37
5.4.3.4	Height()	38
5.4.3.5	PixelSpan()	38
5.4.3.6	PutPixel()	39
5.4.3.7	RebindTexture()	39
5.4.3.8	Resize()	40
5.4.3.9	SurfaceHeight()	41
5.4.3.10	SurfaceWidth()	42
5.4.3.11	Width()	42
6	File Documentation	43
6.1	RigidTrack/_modelest.h File Reference	43
6.2	RigidTrack/communication.cpp File Reference	43
6.3	RigidTrack/communication.h File Reference	44
6.4	RigidTrack/DoxygenMain.md File Reference	45
6.5	RigidTrack/main.cpp File Reference	45
6.5.1	Detailed Description	49
6.5.2	Function Documentation	50
6.5.2.1	calcBoardCornerPositions()	50
6.5.2.2	calibrateCamera()	51
6.5.2.3	calibrateGround()	53
6.5.2.4	closeUDP()	57
6.5.2.5	determineExposure()	57
6.5.2.6	determineOrder()	60
6.5.2.7	drawPositionText()	61

6.5.2.8	getEulerAngles()	62
6.5.2.9	loadCalibration()	63
6.5.2.10	loadCameraPosition()	64
6.5.2.11	loadMarkerConfig()	65
6.5.2.12	main()	67
6.5.2.13	Mat2QPixmap()	69
6.5.2.14	projectCoordinateFrame()	70
6.5.2.15	sendDataUDP()	71
6.5.2.16	setHeadingOffset()	71
6.5.2.17	setReference()	72
6.5.2.18	setUpUDP()	75
6.5.2.19	startStopCamera()	76
6.5.2.20	startTracking()	78
6.5.2.21	testAlgorithms()	83
6.5.3	Variable Documentation	86
6.5.3.1	BACKBUFFER_BITSPERPIXEL	86
6.5.3.2	camera_started	86
6.5.3.3	cameraMatrix	86
6.5.3.4	commObj	86
6.5.3.5	coordinateFrame	87
6.5.3.6	coordinateFrameProjected	87
6.5.3.7	currentMinIndex	87
6.5.3.8	currentPointDistance	87
6.5.3.9	data	87
6.5.3.10	datagram	88
6.5.3.11	distCoeffs	88
6.5.3.12	distModel	88
6.5.3.13	eulerAngles	88
6.5.3.14	eulerRef	88
6.5.3.15	exitRequested	89

6.5.3.16	frameTime	89
6.5.3.17	gotOrder	89
6.5.3.18	headingOffset	89
6.5.3.19	intExposure	89
6.5.3.20	intFrameRate	90
6.5.3.21	intIntensity	90
6.5.3.22	intThreshold	90
6.5.3.23	invertZ	90
6.5.3.24	IPAdressObject	90
6.5.3.25	IPAdressSafety	91
6.5.3.26	IPAdressSafety2	91
6.5.3.27	list_points2d	91
6.5.3.28	list_points2dDifference	91
6.5.3.29	list_points2dOld	91
6.5.3.30	list_points2dProjected	92
6.5.3.31	list_points2dUnsorted	92
6.5.3.32	list_points3d	92
6.5.3.33	logDate	92
6.5.3.34	logfile	92
6.5.3.35	logFileName	93
6.5.3.36	logName	93
6.5.3.37	M_CN	93
6.5.3.38	M_HeadingOffset	93
6.5.3.39	methodPNP	93
6.5.3.40	minPointDistance	94
6.5.3.41	numberMarkers	94
6.5.3.42	pointOrderIndices	94
6.5.3.43	pointOrderIndicesNew	94
6.5.3.44	portObject	94
6.5.3.45	portSafety	95

6.5.3.46	portSafety2	95
6.5.3.47	position	95
6.5.3.48	positionOld	95
6.5.3.49	posRef	95
6.5.3.50	Rmat	96
6.5.3.51	RmatRef	96
6.5.3.52	Rvec	96
6.5.3.53	RvecOriginal	96
6.5.3.54	safety2Enable	96
6.5.3.55	safetyAngle	97
6.5.3.56	safetyBoxLength	97
6.5.3.57	safetyEnable	97
6.5.3.58	ss	97
6.5.3.59	strBuf	97
6.5.3.60	timeFirstFrame	98
6.5.3.61	timeOld	98
6.5.3.62	Tvec	98
6.5.3.63	TvecOriginal	98
6.5.3.64	udpSocketObject	98
6.5.3.65	udpSocketSafety	99
6.5.3.66	udpSocketSafety2	99
6.5.3.67	useGuess	99
6.5.3.68	velocity	99
6.6	RigidTrack/main.h File Reference	100
6.6.1	Detailed Description	102
6.6.2	Function Documentation	102
6.6.2.1	calibrateCamera()	103
6.6.2.2	calibrateGround()	105
6.6.2.3	closeUDP()	109
6.6.2.4	determineExposure()	109

6.6.2.5	determineOrder()	112
6.6.2.6	drawPositionText()	113
6.6.2.7	loadCalibration()	114
6.6.2.8	loadCameraPosition()	115
6.6.2.9	loadMarkerConfig()	116
6.6.2.10	projectCoordinateFrame()	118
6.6.2.11	sendDataUDP()	119
6.6.2.12	setHeadingOffset()	120
6.6.2.13	setReference()	121
6.6.2.14	setUpUDP()	124
6.6.2.15	startStopCamera()	125
6.6.2.16	startTracking()	126
6.6.2.17	testAlgorithms()	131
6.6.3	Variable Documentation	134
6.6.3.1	commObj	134
6.6.3.2	invertZ	134
6.6.3.3	IPAdressObject	134
6.6.3.4	IPAdressSafety	135
6.6.3.5	IPAdressSafety2	135
6.6.3.6	methodPNP	135
6.6.3.7	portObject	135
6.6.3.8	portSafety	135
6.6.3.9	portSafety2	136
6.6.3.10	safety2Enable	136
6.6.3.11	safetyAngle	136
6.6.3.12	safetyBoxLength	136
6.6.3.13	safetyEnable	136
6.7	RigidTrack/precomp.hpp File Reference	137
6.7.1	Macro Definition Documentation	137
6.7.1.1	GET_OPTIMIZED	137

6.8	RigidTrack/resource.h File Reference	138
6.8.1	Macro Definition Documentation	138
6.8.1.1	IDI_ICON1	138
6.9	RigidTrack/RigidTrack.cpp File Reference	138
6.9.1	Detailed Description	138
6.10	RigidTrack/RigidTrack.h File Reference	139
6.10.1	Detailed Description	139
6.11	RigidTrack/supportcode.cpp File Reference	140
6.11.1	Function Documentation	141
6.11.1.1	CBTHookProc()	141
6.11.1.2	CloseWindow()	142
6.11.1.3	CreateAppWindow()	143
6.11.1.4	DrawGLScene()	146
6.11.1.5	FullscreenToggle()	147
6.11.1.6	InitGL()	148
6.11.1.7	LoadGLTextures()	149
6.11.1.8	main()	149
6.11.1.9	PopWaitingDialog()	151
6.11.1.10	PumpMessages()	152
6.11.1.11	ReSizeGLScene()	152
6.11.1.12	TimerProc()	153
6.11.1.13	WinMain()	153
6.11.1.14	WndProc()	154
6.11.2	Variable Documentation	156
6.11.2.1	gActive	156
6.11.2.2	gFullscreen	156
6.11.2.3	gSoftwareDecimate	156
6.11.2.4	gWindowHeight	156
6.11.2.5	gWindowWidth	157
6.11.2.6	hDC	157

6.11.2.7	hHook	157
6.11.2.8	hInstance	157
6.11.2.9	hRC	157
6.11.2.10	hWnd	158
6.11.2.11	keys	158
6.11.2.12	texture	158
6.11.2.13	windowHeight	158
6.11.2.14	windowName	158
6.11.2.15	windowWidth	158
6.12	RigidTrack/supportcode.h File Reference	159
6.12.1	Macro Definition Documentation	160
6.12.1.1	BYTESPERPIXEL	160
6.12.1.2	RGBA	160
6.12.1.3	WIN32_LEAN_AND_MEAN	160
6.12.2	Function Documentation	161
6.12.2.1	CloseWindow()	161
6.12.2.2	CreateAppWindow()	162
6.12.2.3	DrawGLScene()	164
6.12.2.4	FullscreenToggle()	166
6.12.2.5	PopWaitingDialog()	166
6.12.2.6	PumpMessages()	167
6.12.2.7	WndProc()	167
6.12.3	Variable Documentation	169
6.12.3.1	gActive	169
6.12.3.2	gFullscreen	169
6.12.3.3	hDC	169
6.12.3.4	keys	169

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

Hierarchical Index

2.1 Class Hierarchy

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

CvModelEstimator2	15
QMainWindow	
RigidTrack	19
QObject	
commObject	9
Surface	34

Chapter 3

Class Index

3.1 Class List

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

commObject	9
CvModelEstimator2	15
RigidTrack	19
Surface	34

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

RigidTrack/_modelest.h	43
RigidTrack/communication.cpp	43
RigidTrack/communication.h	44
RigidTrack/main.cpp	
Rigid Track main file that contains most functionality	45
RigidTrack/main.h	
Header file for main.cpp	100
RigidTrack/precomp.hpp	137
RigidTrack/resource.h	138
RigidTrack/RigidTrack.cpp	
Rigid Track GUI source that contains functions for GUI events	138
RigidTrack/RigidTrack.h	
Rigid Track GUI source header with Qt Signals and Slots	139
RigidTrack/supportcode.cpp	140
RigidTrack/supportcode.h	159

Chapter 5

Class Documentation

5.1 commObject Class Reference

```
#include <communication.h>
```

Inherits QObject.

Signals

- void [statusChanged](#) (QString newText)
- void [imageChanged](#) (QPixmap image)
- void [logAdded](#) (QString LogText)
- void [logCleared](#) ()
- void [P3Penabled](#) (bool value)
- void [progressUpdated](#) (int value)

Public Member Functions

- void [changeStatus](#) (QString newText)
- void [changeImage](#) (QPixmap image)
- void [addLog](#) (QString LogText)
- void [clearLog](#) ()
- void [enableP3P](#) (bool value)
- void [progressUpdate](#) (int value)

5.1.1 Detailed Description

Definition at line 7 of file communication.h.

5.1.2 Member Function Documentation

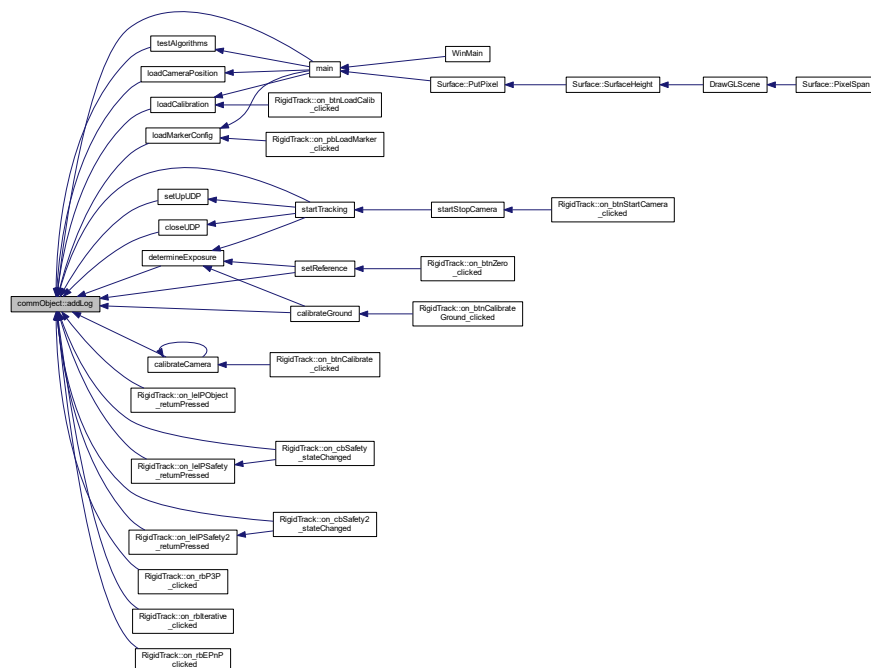
5.1.2.1 addLog()

```
void commObject::addLog (
    QString LogText )
```

Definition at line 20 of file communication.cpp.

```
20
21
22     emit logAdded(LogText);
23     QApplication::processEvents();
24 }
```

Here is the caller graph for this function:



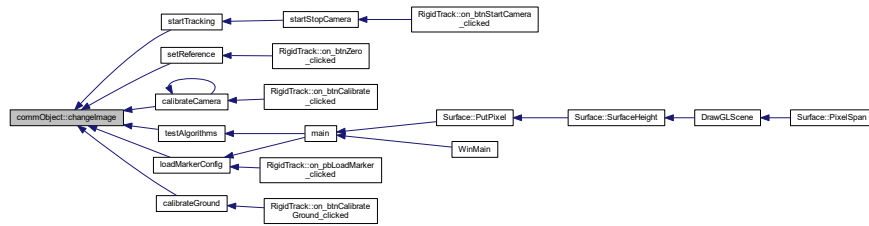
5.1.2.2 changelImage()

```
void commObject::changeImage (
    QPixmap image )
```

Definition at line 14 of file communication.cpp.

```
14
15
16     emit imageChanged(image);
17     QApplication::processEvents();
18 }
```

Here is the caller graph for this function:



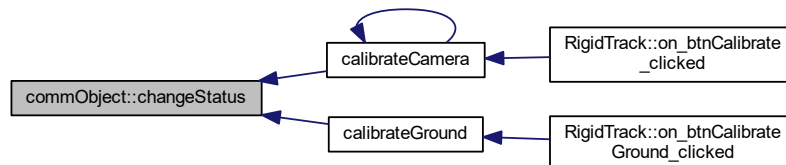
5.1.2.3 changeStatus()

```
void commObject::changeStatus (
    QString newText )
```

Definition at line 8 of file communication.cpp.

```
8
9
10     emit statusChanged(newText);
11     QApplication::processEvents();
12 }
```

Here is the caller graph for this function:



5.1.2.4 clearLog()

```
void commObject::clearLog ( )
```

Definition at line 26 of file communication.cpp.

```
26
27
28     emit logCleared();
29     QApplication::processEvents();
30 }
```

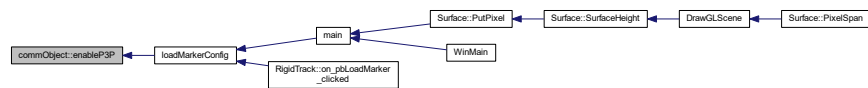
5.1.2.5 enableP3P()

```
void commObject::enableP3P (
    bool value )
```

Definition at line 32 of file communication.cpp.

```
33 {
34     emit P3Penabled(value);
35     QApplication::processEvents();
36 }
```

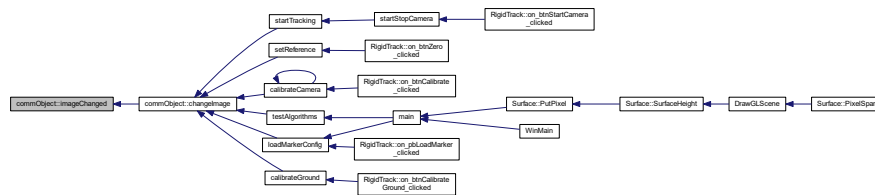
Here is the caller graph for this function:



5.1.2.6 imageChanged

```
void commObject::imageChanged (
    QPixmap image ) [signal]
```

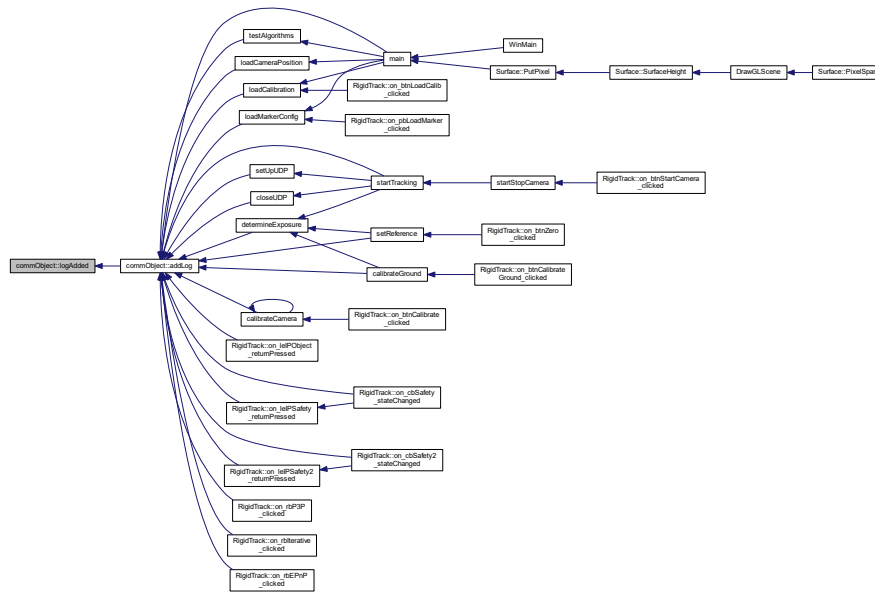
Here is the caller graph for this function:



5.1.2.7 logAdded

```
void commObject::logAdded (
    QString LogText ) [signal]
```


Here is the caller graph for this function:



5.1.2.8 logCleared

```
void commObject::logCleared ( ) [signal]
```

Here is the caller graph for this function:



5.1.2.9 P3Penabled

```
void commObject::P3Penabled (
    bool value ) [signal]
```

Here is the caller graph for this function:



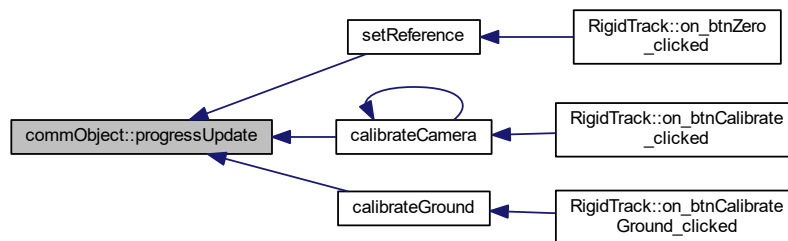
5.1.2.10 progressUpdate()

```
void commObject::progressUpdate (
    int value )
```

Definition at line 38 of file communication.cpp.

```
39 {
40     emit progressUpdated(value);
41     QApplication::processEvents();
42 }
```

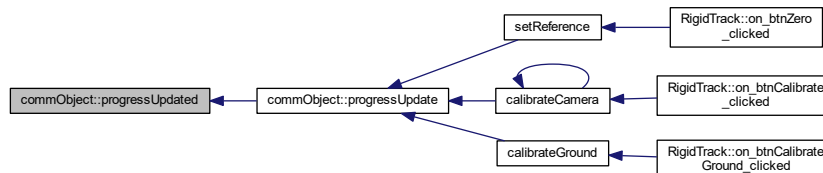
Here is the caller graph for this function:



5.1.2.11 progressUpdated

```
void commObject::progressUpdated (
    int value ) [signal]
```

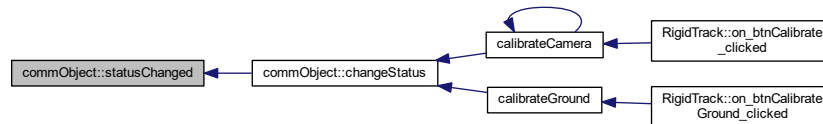
Here is the caller graph for this function:



5.1.2.12 statusChanged

```
void commObject::statusChanged (
    QString newText ) [signal]
```

Here is the caller graph for this function:



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

- RigidTrack/[communication.h](#)
- RigidTrack/[communication.cpp](#)

5.2 CvModelEstimator2 Class Reference

```
#include <_modelest.h>
```

Public Member Functions

- [CvModelEstimator2](#) (int _modelPoints, CvSize _modelSize, int _maxBasicSolutions)
- virtual [~CvModelEstimator2](#) ()
- virtual int [runKernel](#) (const CvMat *m1, const CvMat *m2, CvMat *model)=0
- virtual bool [runLMeDS](#) (const CvMat *m1, const CvMat *m2, CvMat *model, CvMat *mask, double confidence=0.99, int maxIters=2000)
- virtual bool [runRANSAC](#) (const CvMat *m1, const CvMat *m2, CvMat *model, CvMat *mask, double threshold, double confidence=0.99, int maxIters=2000)
- virtual bool [refine](#) (const CvMat *, const CvMat *, CvMat *, int)
- virtual void [setSeed](#) (int64 seed)

Protected Member Functions

- virtual void [computeReprojError](#) (const CvMat *m1, const CvMat *m2, const CvMat *model, CvMat *error)=0
- virtual int [findInliers](#) (const CvMat *m1, const CvMat *m2, const CvMat *model, CvMat *error, CvMat *mask, double threshold)
- virtual bool [getSubset](#) (const CvMat *m1, const CvMat *m2, CvMat *ms1, CvMat *ms2, int maxAttempts=1000)
- virtual bool [checkSubset](#) (const CvMat *ms1, int count)

Protected Attributes

- `CvRNG` `rng`
- `int` `modelPoints`
- `CvSize` `modelSize`
- `int` `maxBasicSolutions`
- `bool` `checkPartialSubsets`

5.2.1 Detailed Description

Definition at line 48 of file `_modelest.h`.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 `CvModelEstimator2()`

```
CvModelEstimator2::CvModelEstimator2 (
    int _modelPoints,
    CvSize _modelSize,
    int _maxBasicSolutions )
```

5.2.2.2 `~CvModelEstimator2()`

```
virtual CvModelEstimator2::~~CvModelEstimator2 ( ) [virtual]
```

5.2.3 Member Function Documentation

5.2.3.1 `checkSubset()`

```
virtual bool CvModelEstimator2::checkSubset (
    const CvMat * ms1,
    int count ) [protected], [virtual]
```

5.2.3.2 computeReprojError()

```
virtual void CvModelEstimator2::computeReprojError (
    const CvMat * m1,
    const CvMat * m2,
    const CvMat * model,
    CvMat * error ) [protected], [pure virtual]
```

5.2.3.3 findInliers()

```
virtual int CvModelEstimator2::findInliers (
    const CvMat * m1,
    const CvMat * m2,
    const CvMat * model,
    CvMat * error,
    CvMat * mask,
    double threshold ) [protected], [virtual]
```

5.2.3.4 getSubset()

```
virtual bool CvModelEstimator2::getSubset (
    const CvMat * m1,
    const CvMat * m2,
    CvMat * ms1,
    CvMat * ms2,
    int maxAttempts = 1000 ) [protected], [virtual]
```

5.2.3.5 refine()

```
virtual bool CvModelEstimator2::refine (
    const CvMat * ,
    const CvMat * ,
    CvMat * ,
    int ) [inline], [virtual]
```

Definition at line 60 of file `_modelesth`.

```
60 { return true; }
```

5.2.3.6 runKernel()

```
virtual int CvModelEstimator2::runKernel (  
    const CvMat * m1,  
    const CvMat * m2,  
    CvMat * model ) [pure virtual]
```

5.2.3.7 runLMeDS()

```
virtual bool CvModelEstimator2::runLMeDS (  
    const CvMat * m1,  
    const CvMat * m2,  
    CvMat * model,  
    CvMat * mask,  
    double confidence = 0.99,  
    int maxIters = 2000 ) [virtual]
```

5.2.3.8 runRANSAC()

```
virtual bool CvModelEstimator2::runRANSAC (  
    const CvMat * m1,  
    const CvMat * m2,  
    CvMat * model,  
    CvMat * mask,  
    double threshold,  
    double confidence = 0.99,  
    int maxIters = 2000 ) [virtual]
```

5.2.3.9 setSeed()

```
virtual void CvModelEstimator2::setSeed (  
    int64 seed ) [virtual]
```

5.2.4 Member Data Documentation

5.2.4.1 checkPartialSubsets

```
bool CvModelEstimator2::checkPartialSubsets [protected]
```

Definition at line 77 of file `_modelest.h`.

5.2.4.2 maxBasicSolutions

```
int CvModelEstimator2::maxBasicSolutions [protected]
```

Definition at line 76 of file `_modelest.h`.

5.2.4.3 modelPoints

```
int CvModelEstimator2::modelPoints [protected]
```

Definition at line 74 of file `_modelest.h`.

5.2.4.4 modelSize

```
CvSize CvModelEstimator2::modelSize [protected]
```

Definition at line 75 of file `_modelest.h`.

5.2.4.5 rng

```
CvRNG CvModelEstimator2::rng [protected]
```

Definition at line 73 of file `_modelest.h`.

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

- RigidTrack/[_modelest.h](#)

5.3 RigidTrack Class Reference

```
#include <RigidTrack.h>
```

Inherits QMainWindow.

Public Slots

- void [on_btnStartCamera_clicked](#) ()
- void [on_btnZero_clicked](#) ()
- void [on_btnCalibrate_clicked](#) ()
- void [setImage](#) (QPixmap image)
- void [clearLog](#) ()
- void [progressUpdate](#) (int value)
- void [on_btnLoadCalib_clicked](#) ()
- void [setLog](#) (QString logText)
- void [on_sbHeadingOffset_valueChanged](#) (double d)
- void [on_leIPObject_returnPressed](#) ()
- void [on_leIPSafety_returnPressed](#) ()
- void [on_leIPSafety2_returnPressed](#) ()
- void [on_rbP3P_clicked](#) ()
- void [on_rblterative_clicked](#) ()
- void [on_rbEPnP_clicked](#) ()
- void [on_actionShow_Help_triggered](#) ()
- void [on_cbSafety_stateChanged](#) (int state)
- void [on_cbSafety2_stateChanged](#) (int state)
- void [on_dsbDimension_valueChanged](#) (double d)
- void [on_sbAngle_valueChanged](#) (int i)
- void [on_pbLoadMarker_clicked](#) ()
- void [on_cbInvert_stateChanged](#) (int state)
- void [enableP3P](#) (bool value)
- void [on_btnCalibrateGround_clicked](#) ()
- void [on_actionOpen_Log_Folder_triggered](#) ()
- void [on_actionAbout_Rigid_Track_triggered](#) ()
- void [on_actionOpen_Installation_Folder_triggered](#) ()

Public Member Functions

- [RigidTrack](#) (QWidget *parent=Q_NULLPTR)

5.3.1 Detailed Description

Definition at line 17 of file RigidTrack.h.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 RigidTrack()

```
RigidTrack::RigidTrack (
    QWidget * parent = Q_NULLPTR )
```

Definition at line 20 of file RigidTrack.cpp.

```
21     : QMainWindow(parent)
22 {
23     ui.setupUi(this);
24
25 }
```


5.3.3 Member Function Documentation

5.3.3.1 clearLog

```
void RigidTrack::clearLog ( ) [slot]
```

Definition at line 42 of file RigidTrack.cpp.

```
43 {  
44     ui.listLog->reset();  
45 }
```

5.3.3.2 enableP3P

```
void RigidTrack::enableP3P (  
    bool value ) [slot]
```

Definition at line 229 of file RigidTrack.cpp.

```
230 {  
231     RigidTrack::ui.rbP3P->setEnabled(value);  
232 }
```

5.3.3.3 on_actionAbout_Rigid_Track_triggered

```
void RigidTrack::on_actionAbout_Rigid_Track_triggered ( ) [slot]
```

Definition at line 245 of file RigidTrack.cpp.

```
246 {  
247     QMessageBox msgBox;  
248     msgBox.setWindowTitle("About Rigid Track");  
249     msgBox.setText("Rigid Track\nInstitute for Flight System Dynamics\nVersion:\t 1.0\nAuthor:\t Florian  
Wachter\nBuild Date:\t " + QString(__DATE__));  
250     msgBox.exec();  
251 }
```

5.3.3.4 on_actionOpen_Installation_Folder_triggered

```
void RigidTrack::on_actionOpen_Installation_Folder_triggered ( ) [slot]
```

Definition at line 253 of file RigidTrack.cpp.

```
254 {  
255     QString command = "explorer.exe " + QDir::currentPath().replace("/", "\\");  
256     QProcess::startDetached(command);  
257  
258 }
```

5.3.3.5 on_actionOpen_Log_Folder_triggered

```
void RigidTrack::on_actionOpen_Log_Folder_triggered ( ) [slot]
```

Definition at line 239 of file RigidTrack.cpp.

```
240 {  
241     QString command = "explorer.exe " + QDir::currentPath().replace("/", "\\") + "\\logs";  
242     QProcess::startDetached(command);  
243 }
```

5.3.3.6 on_actionShow_Help_triggered

```
void RigidTrack::on_actionShow_Help_triggered ( ) [slot]
```

Definition at line 160 of file RigidTrack.cpp.

```
161 {  
162  
163     ///< append help.pdf to the path since this is the documentation in html format  
164     QString qtStrFile = QDir::currentPath().replace("/", "\\") + "\\help.pdf";  
165  
166     ///< open the documentation help file in the standard browser  
167     QDesktopServices::openUrl(QUrl::fromLocalFile(qtStrFile));  
168 }
```

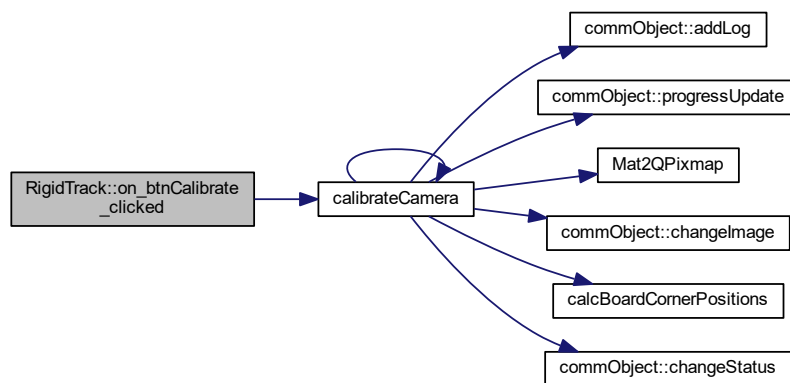
5.3.3.7 on_btnCalibrate_clicked

```
void RigidTrack::on_btnCalibrate_clicked ( ) [slot]
```

Definition at line 32 of file RigidTrack.cpp.

```
33 {  
34     calibrateCamera();  
35 }
```

Here is the call graph for this function:



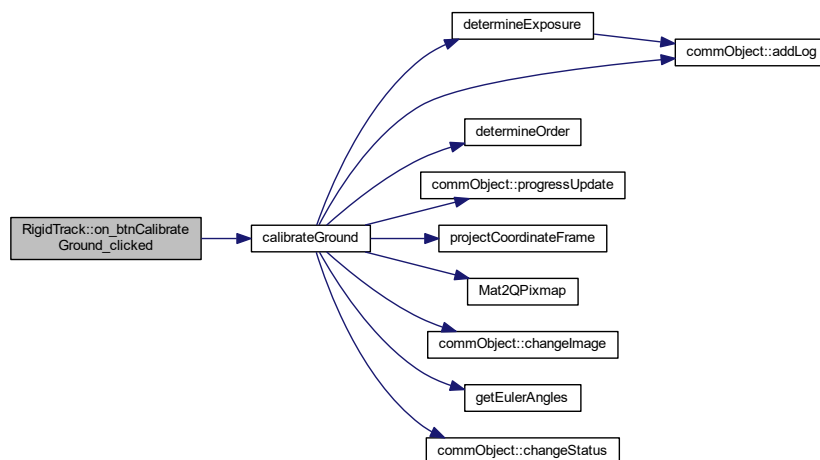
5.3.3.8 on_btnCalibrateGround_clicked

```
void RigidTrack::on_btnCalibrateGround_clicked ( ) [slot]
```

Definition at line 234 of file RigidTrack.cpp.

```
235 {  
236     calibrateGround();  
237 }
```

Here is the call graph for this function:



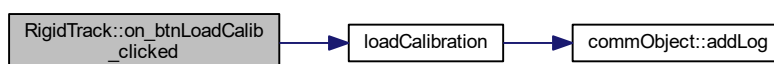
5.3.3.9 on_btnLoadCalib_clicked

```
void RigidTrack::on_btnLoadCalib_clicked ( ) [slot]
```

Definition at line 52 of file `RigidTrack.cpp`.

```
53 {
54     loadCalibration(1);
55 }
```

Here is the call graph for this function:



5.3.3.10 on_btnStartCamera_clicked

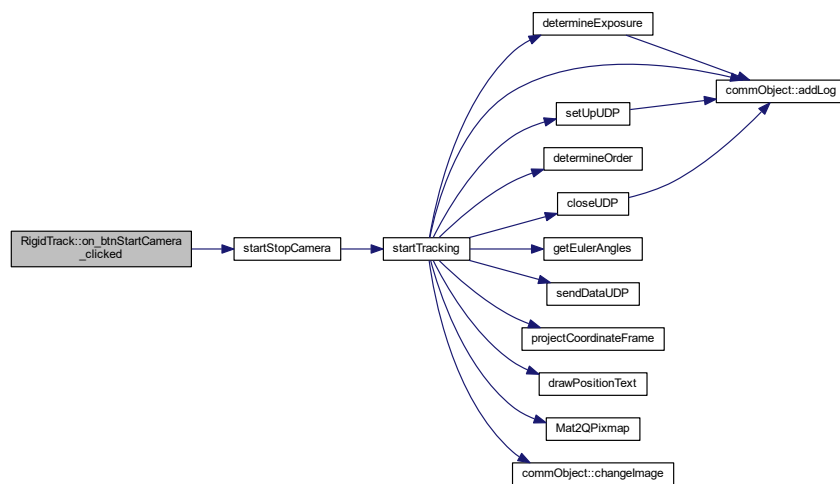
```
void RigidTrack::on_btnStartCamera_clicked ( ) [slot]
```

Definition at line 260 of file RigidTrack.cpp.

```

261 {
262     if(RigidTrack::ui.btnStartCamera->text() == "Start Tracking")
263     {
264         RigidTrack::ui.btnStartCamera->setText("Stop Tracking");
265     }
266     else
267     {
268         RigidTrack::ui.btnStartCamera->setText("Start Tracking");
269     }
270     startStopCamera();
271 }
```

Here is the call graph for this function:



5.3.3.11 on_btnZero_clicked

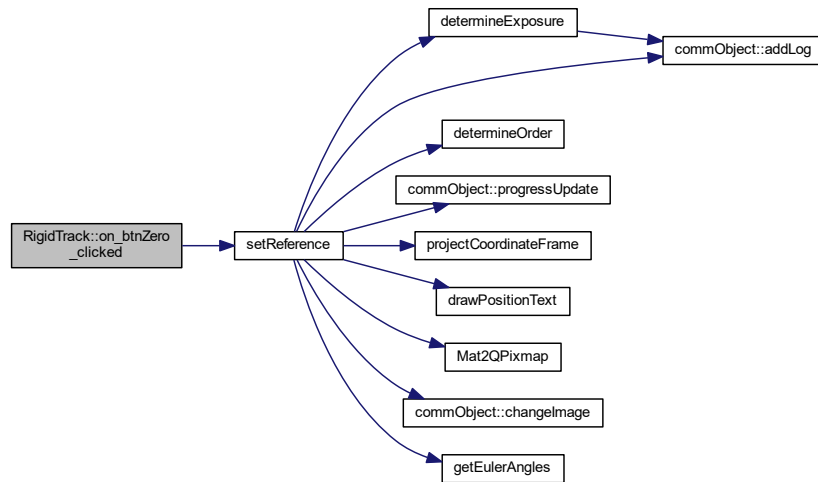
```
void RigidTrack::on_btnZero_clicked ( ) [slot]
```

Definition at line 27 of file RigidTrack.cpp.

```

28 {
29     setReference();
30 }
```

Here is the call graph for this function:



5.3.3.12 on_cbInvert_stateChanged

```
void RigidTrack::on_cbInvert_stateChanged (
    int state ) [slot]
```

Definition at line 217 of file `RigidTrack.cpp`.

```
218 {
219     if (state)
220     {
221         invertZ = -1;
222     }
223     else
224     {
225         invertZ = 1;
226     }
227 }
```

5.3.3.13 on_cbSafety2_stateChanged

```
void RigidTrack::on_cbSafety2_stateChanged (
    int state ) [slot]
```

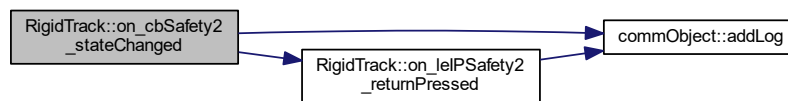
Definition at line 187 of file `RigidTrack.cpp`.

```

188 {
189     RigidTrack::ui.leIPSafety2->setEnabled(state);
190     safety2Enable = state;
191     if (state)
192     {
193         commObj.addLog("Enabled second Receiver");
194         on_leIPSafety2_returnPressed();
195     }
196     else
197     {
198         commObj.addLog("Disabled second Receiver");
199     }
200 }

```

Here is the call graph for this function:



5.3.3.14 on_cbSafety_stateChanged

```

void RigidTrack::on_cbSafety_stateChanged (
    int state ) [slot]

```

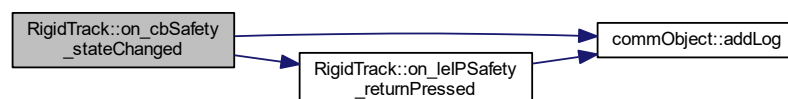
Definition at line 170 of file RigidTrack.cpp.

```

171 {
172     RigidTrack::ui.dsbDimension->setEnabled(state);
173     RigidTrack::ui.sbAngle->setEnabled(state);
174     safetyEnable = state;
175     RigidTrack::ui.leIPSafety->setEnabled(state);
176     if (state)
177     {
178         commObj.addLog("Enabled Safety Area Protection");
179         on_leIPSafety_returnPressed();
180     }
181     else
182     {
183         commObj.addLog("Disabled Safety Area Protection");
184     }
185 }

```

Here is the call graph for this function:



5.3.3.15 on_dsbDimension_valueChanged

```
void RigidTrack::on_dsbDimension_valueChanged (
    double d ) [slot]
```

Definition at line 202 of file RigidTrack.cpp.

```
203 {
204     safetyBoxLength = d;
205 }
```

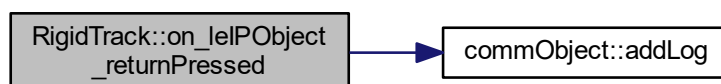
5.3.3.16 on_leIPObject_returnPressed

```
void RigidTrack::on_leIPObject_returnPressed ( ) [slot]
```

Definition at line 68 of file RigidTrack.cpp.

```
69 {
70     try{
71         QString address = RigidTrack::ui.leIPObject->text();
72         IPAdressObject = QHostAddress(address.split(":")[0]);
73         if (IPAdressObject.isNull() || address.split(":").length() == 1 || address.split(":")[1]==0
74     )
75     {
76         throw 2;
77     }
78     portObject = address.split(":")[1].toInt();
79     commObj.addLog("Object IP changed to:");
80     commObj.addLog(IPAdressObject.toString());
81     commObj.addLog("Object Port changed to:");
82     commObj.addLog(QString::number(portObject));
83 }
84 catch (...)
85 {
86     commObj.addLog("Error Changing the IP Adress or Port! Restored Standard Values
192.168.0.1:9155");
87     IPAdressObject = QHostAddress("192.168.0.1");
88     portObject = 9155;
89     RigidTrack::ui.leIPObject->setText("192.168.0.1:9155");
90 }
```

Here is the call graph for this function:



5.3.3.17 on_leIPSafety2_returnPressed

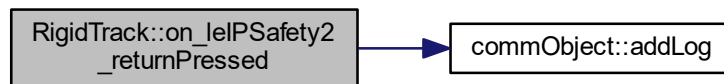
```
void RigidTrack::on_leIPSafety2_returnPressed ( ) [slot]
```

Definition at line 117 of file RigidTrack.cpp.

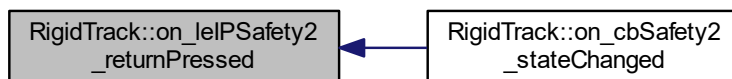
```

118 {
119     try {
120         QString address = RigidTrack::ui.leIPSafety2->text();
121         IPAddressSafety2 = QHostAddress(address.split(":")[0]);
122         if (IPAddressSafety2.isNull() || address.split(":").length() == 1 || address.split(":")[1]
== 0)
123         {
124             throw 2;
125         }
126         portSafety2 = address.split(":")[1].toInt();
127         commObj.addLog("Receiver 2 IP changed to:");
128         commObj.addLog(IPAddressSafety2.toString());
129         commObj.addLog("Receiver 2 Port changed to:");
130         commObj.addLog(QString::number(portSafety2));
131     }
132     catch (...)
133     {
134         commObj.addLog("Error Changing the IP Address or Port! Restored Standard Values
192.168.0.1:9155");
135         IPAddressSafety2 = QHostAddress("192.168.0.1");
136         portSafety2 = 9155;
137         RigidTrack::ui.leIPSafety2->setText("192.168.0.1:9155");
138     }
139 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.3.3.18 on_leIPSafety_returnPressed

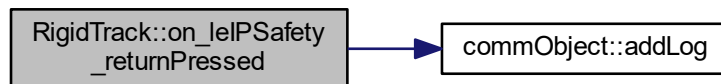
```
void RigidTrack::on_leIPSafety_returnPressed ( ) [slot]
```

Definition at line 93 of file RigidTrack.cpp.

```

94 {
95     try{
96         QString address = RigidTrack::ui.leIPSafety->text();
97         IPAdressSafety = QHostAddress(address.split(":")[0]);
98         if (IPAdressSafety.isNull() || address.split(":").length() == 1 || address.split(":")[1] ==
99             0)
100         {
101             throw 2;
102         }
103         portSafety = address.split(":")[1].toInt();
104         commObj.addLog("Safety Switch IP changed to:");
105         commObj.addLog(IPAdressSafety.toString());
106         commObj.addLog("Safety Switch Port changed to:");
107         commObj.addLog(QString::number(portSafety));
108     }
109     catch (...)
110     {
111         commObj.addLog("Error Changing the IP Adress or Port! Restored Standard Values
112         192.168.0.1:9155");
113         IPAdressSafety = QHostAddress("192.168.0.1");
114         portSafety = 9155;
115         RigidTrack::ui.leIPSafety->setText("192.168.0.1:9155");
116     }
117 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



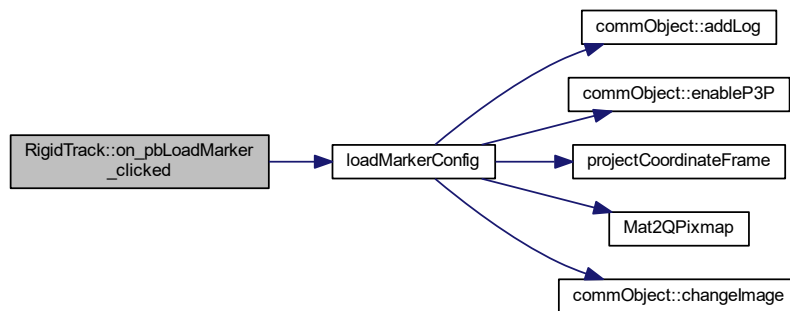
5.3.3.19 on_pbLoadMarker_clicked

```
void RigidTrack::on_pbLoadMarker_clicked ( ) [slot]
```

Definition at line 212 of file RigidTrack.cpp.

```
213 {
214     loadMarkerConfig(1);
215 }
```

Here is the call graph for this function:



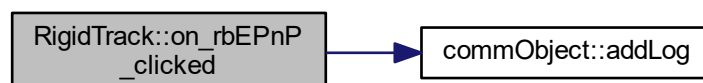
5.3.3.20 on_rbEPnP_clicked

```
void RigidTrack::on_rbEPnP_clicked ( ) [slot]
```

Definition at line 154 of file RigidTrack.cpp.

```
155 {
156     methodPNP = 1;
157     commObj.addLog("Changed PnP algorithm to EPnP");
158 }
```

Here is the call graph for this function:



5.3.3.21 on_rbIterative_clicked

```
void RigidTrack::on_rbIterative_clicked ( ) [slot]
```

Definition at line 148 of file RigidTrack.cpp.

```
149 {  
150     methodPNP = 0;  
151     commObj.addLog("Changed PnP algorithm to Iterative");  
152 }
```

Here is the call graph for this function:



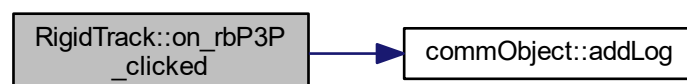
5.3.3.22 on_rbP3P_clicked

```
void RigidTrack::on_rbP3P_clicked ( ) [slot]
```

Definition at line 142 of file RigidTrack.cpp.

```
143 {  
144     methodPNP = 2;  
145     commObj.addLog("Changed PnP algorithm to P3P");  
146 }
```

Here is the call graph for this function:



5.3.3.23 on_sbAngle_valueChanged

```
void RigidTrack::on_sbAngle_valueChanged (
    int i ) [slot]
```

Definition at line 207 of file RigidTrack.cpp.

```
208 {
209     safetyAngle = i;
210 }
```

5.3.3.24 on_sbHeadingOffset_valueChanged

```
void RigidTrack::on_sbHeadingOffset_valueChanged (
    double d ) [slot]
```

Definition at line 63 of file RigidTrack.cpp.

```
64 {
65     setHeadingOffset(d);
66 }
```

Here is the call graph for this function:



5.3.3.25 progressUpdate

```
void RigidTrack::progressUpdate (
    int value ) [slot]
```

Definition at line 47 of file RigidTrack.cpp.

```
48 {
49     RigidTrack::ui.progressBar->setValue(value);
50 }
```

5.3.3.26 setImage

```
void RigidTrack::setImage (
    QPixmap image ) [slot]
```

Definition at line 37 of file RigidTrack.cpp.

```
38 {
39     ui.lbStatus->setPixmap(image);
40 }
```

5.3.3.27 setLog

```
void RigidTrack::setLog (
    QString logText ) [slot]
```

Definition at line 57 of file RigidTrack.cpp.

```
58 {
59     RigidTrack::ui.listLog->addItem(logText);
60     RigidTrack::ui.listLog->scrollToBottom();
61 }
```

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

- [RigidTrack/RigidTrack.h](#)
- [RigidTrack/RigidTrack.cpp](#)

5.4 Surface Class Reference

```
#include <supportcode.h>
```

Public Member Functions

- [Surface](#) (int [Width](#), int [Height](#))
/<=====
- [~Surface](#) ()
- GLuint [GetTexture](#) ()
- void [Resize](#) (int [Width](#), int [Height](#))
- int [CalculateSize](#) (int [Width](#))
- int [Width](#) ()
- int [Height](#) ()
- int [SurfaceWidth](#) ()
- int [SurfaceHeight](#) ()
- void [PutPixel](#) (int X, int Y, PIXEL Color)
- unsigned char * [GetBuffer](#) ()
- void [RebindTexture](#) ()
- int [PixelSpan](#) ()

5.4.1 Detailed Description

Definition at line 25 of file supportcode.h.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 Surface()

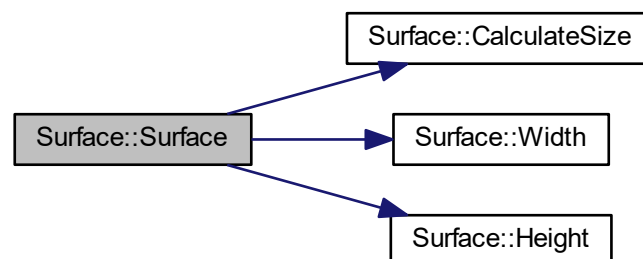
```
Surface::Surface (
    int Width,
    int Height )
```

/<=====

Definition at line 417 of file supportcode.cpp.

```
417                                     : buffer(0), mDirty(true)
418 {
419     //!/<== Use power of 2 texture sizes ==
420
421     mSurfaceWidth = CalculateSize(Width);
422     mSurfaceHeight = CalculateSize(Height);
423
424     mSpan = mSurfaceWidth;
425
426     mWidth = Width;
427     mHeight = Height;
428
429     buffer = (unsigned char*) malloc(mSurfaceWidth * mSurfaceHeight *
430     BYTESPERPIXEL);
431
432     memset(buffer, 0, mSurfaceWidth * mSurfaceHeight * BYTESPERPIXEL);
433
434     if(buffer==0)
435         throw(" nable to allocate surface buffer");
436
437     glGenTextures(1, &mTexture);
438     if(mTexture==0)
439         throw(" nable to gen OpenGL texture");
440
441     glBindTexture(GL_TEXTURE_2D, mTexture);
442     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8, mSurfaceWidth, mSurfaceHeight, 0, GL_RGBA, GL_UNSIGNED_BYTE,
443     buffer);
444     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
445     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
446 }
```

Here is the call graph for this function:



5.4.2.2 ~Surface()

```
Surface::~~Surface ( )
```

Definition at line 447 of file supportcode.cpp.

```
448 {  
449  
450 }
```

5.4.3 Member Function Documentation

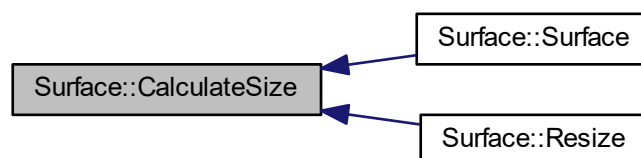
5.4.3.1 CalculateSize()

```
int Surface::CalculateSize (  
    int Width )
```

Definition at line 494 of file supportcode.cpp.

```
495 {  
496     int mSize = 1;  
497  
498     while (mSize<Width)  
499         mSize*=2;  
500  
501     return mSize;  
502 }
```

Here is the caller graph for this function:



5.4.3.2 GetBuffer()

```
unsigned char* Surface::GetBuffer ( ) [inline]
```

Definition at line 40 of file supportcode.h.

```
40 { return buffer; }
```

Here is the call graph for this function:



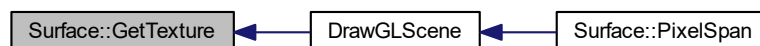
5.4.3.3 GetTexture()

```
GLuint Surface::GetTexture ( )
```

Definition at line 504 of file supportcode.cpp.

```
505 {  
506     ///507     {  
508         glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8, mSurfaceWidth, mSurfaceHeight, 0, GL_RGBA,  
GL_UNSIGNED_BYTE, buffer);  
509         mDirty = false;  
510     }  
511     return mTexture;  
512 }
```

Here is the caller graph for this function:



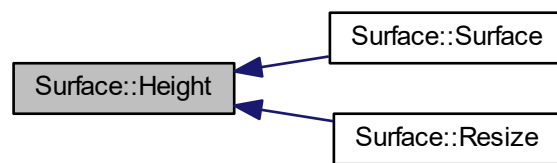
5.4.3.4 Height()

```
int Surface::Height ( ) [inline]
```

Definition at line 35 of file supportcode.h.

```
35 { return mHeight; }
```

Here is the caller graph for this function:



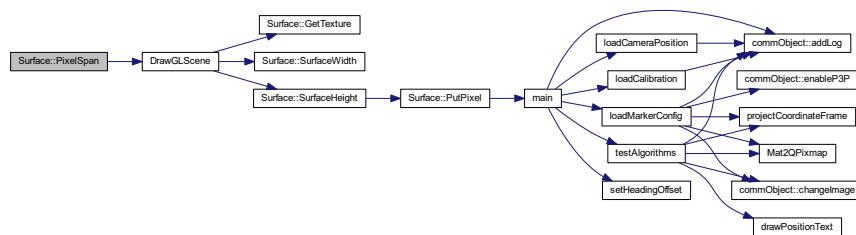
5.4.3.5 PixelSpan()

```
int Surface::PixelSpan ( ) [inline]
```

Definition at line 42 of file supportcode.h.

```
42 { return mSpan; }
```

Here is the call graph for this function:



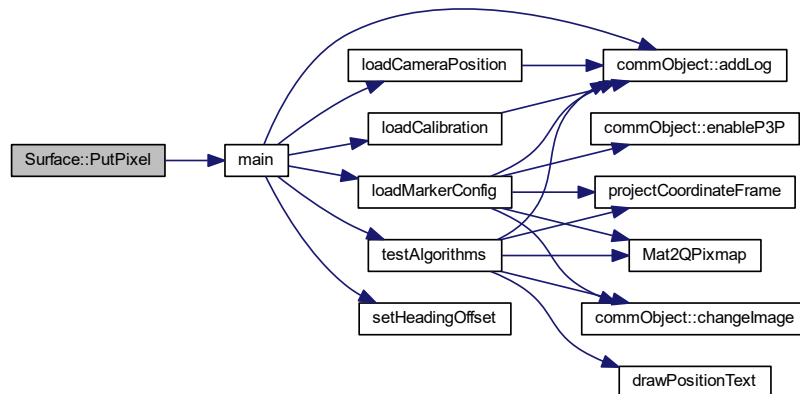
5.4.3.6 PutPixel()

```
void Surface::PutPixel (
    int X,
    int Y,
    PIXEL Color )
```

Definition at line 514 of file supportcode.cpp.

```
515 {
516     if(X>=0 && Y>=0 && X<mWidth && Y<mHeight)
517     {
518         unsigned int *point = (unsigned int*)buffer + (Y*mSpan)+X;
519         *point = Color;
520         mDirty = true;
521     }
522 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.4.3.7 RebindTexture()

```
void Surface::RebindTexture ( )
```

Definition at line 452 of file supportcode.cpp.

```

453 {
454     glGenTextures(1, &mTexture);
455     if(mTexture==0)
456         throw(" nable to gen OpenGL texture");
457
458
459     glBindTexture(GL_TEXTURE_2D, mTexture);
460     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8, mSurfaceWidth, mSurfaceHeight, 0, GL_RGBA, GL_UNSIGNED_BYTE,
buffer);
461     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
462     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
463 }

```

Here is the caller graph for this function:



5.4.3.8 Resize()

```

void Surface::Resize (
    int Width,
    int Height )

```

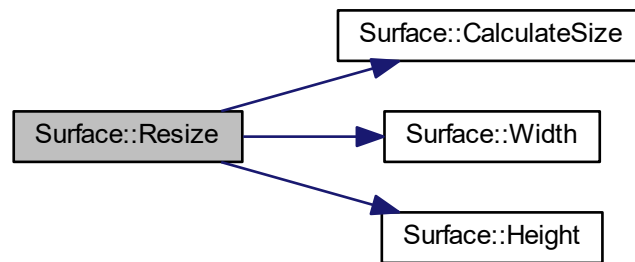
Definition at line 466 of file supportcode.cpp.

```

467 {
468     if(Width==mWidth && Height==mHeight)
469         return;
470
471     if(Width<1 || Height<1)
472         return;
473
474     int newSize = CalculateSize(Width);
475
476     if(newSize>mSurfaceWidth || newSize>mSurfaceHeight)
477     {
478         if(buffer!=0)
479             free(buffer);
480
481         mSurfaceWidth = newSize;
482         mSurfaceHeight = newSize;
483         mSpan = mSurfaceWidth;
484         mWidth = Width;
485         mHeight = Height;
486         buffer = (unsigned char*) malloc(mSurfaceWidth * mSurfaceHeight *
BYTESPERPIXEL);
487         if(buffer==0)
488             throw("Unable to allocate surface buffer");
489
490         mDirty = true;
491     }
492 }

```

Here is the call graph for this function:



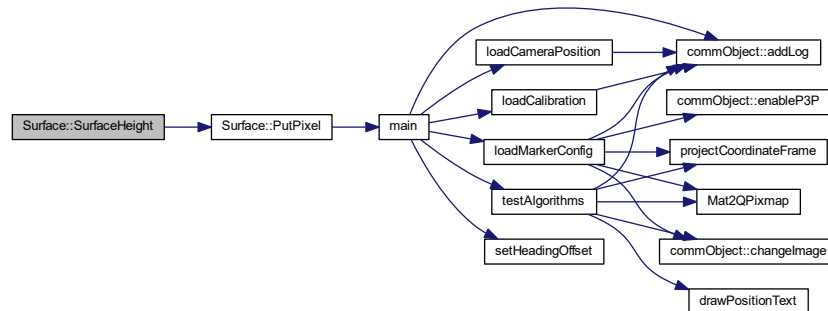
5.4.3.9 SurfaceHeight()

```
int Surface::SurfaceHeight ( ) [inline]
```

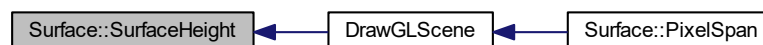
Definition at line 37 of file supportcode.h.

```
37 { return mSurfaceHeight; }
```

Here is the call graph for this function:



Here is the caller graph for this function:



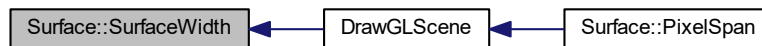
5.4.3.10 SurfaceWidth()

```
int Surface::SurfaceWidth ( ) [inline]
```

Definition at line 36 of file supportcode.h.

```
36 { return mSurfaceWidth; }
```

Here is the caller graph for this function:



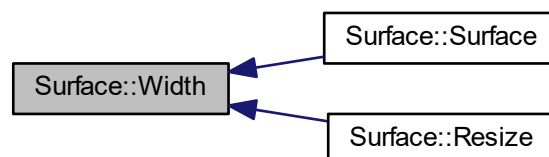
5.4.3.11 Width()

```
int Surface::Width ( ) [inline]
```

Definition at line 34 of file supportcode.h.

```
34 { return mWidth; }
```

Here is the caller graph for this function:



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

- RigidTrack/[supportcode.h](#)
- RigidTrack/[supportcode.cpp](#)

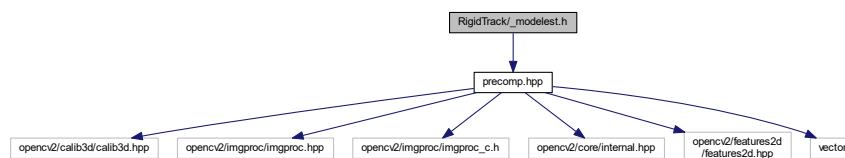
Chapter 6

File Documentation

6.1 RigidTrack/_modelest.h File Reference

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

Include dependency graph for _modelest.h:



Classes

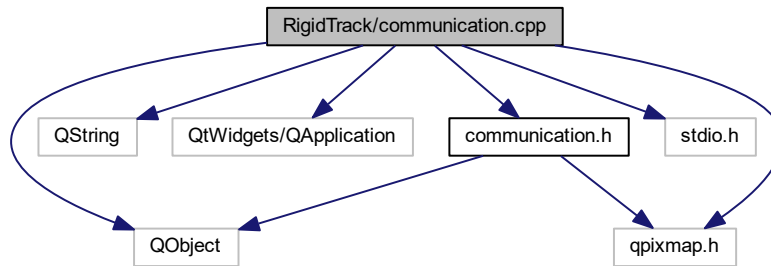
- class [CvModelEstimator2](#)

6.2 RigidTrack/communication.cpp File Reference

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

```
#include "communication.h"
```

Include dependency graph for communication.cpp:

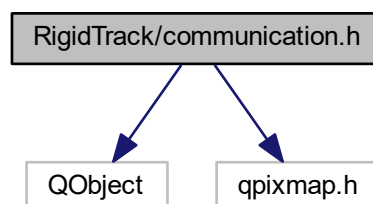


6.3 RigidTrack/communication.h File Reference

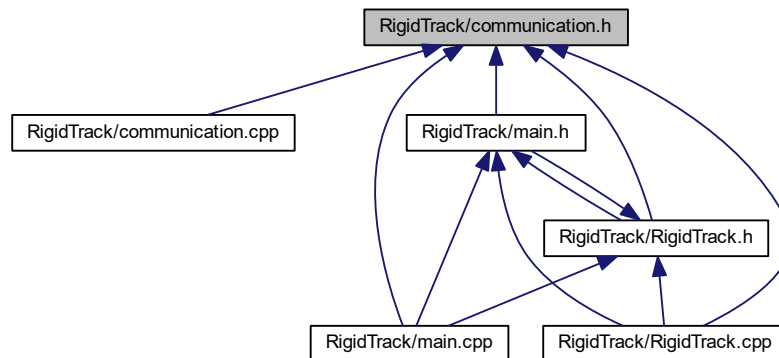
```
#include <QObject>
```

```
#include <qpixmap.h>
```

Include dependency graph for communication.h:



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



Classes

- class [commObject](#)

6.4 RigidTrack/DoxygenMain.md File Reference

6.5 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>

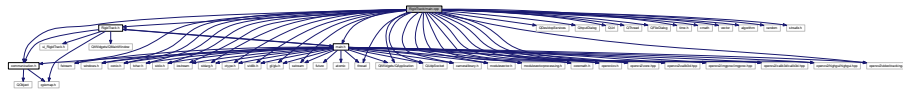
```

```

#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>
#include <algorithm>
#include <random>
#include <thread>
#include <strsafe.h>

```

Include dependency graph for main.cpp:



Functions

- int [main](#) (int argc, char *argv[])
main initialises the GUI and values for the marker position etc
- QPixmap [Mat2QPixmap](#) (cv::Mat src)
- void [calcBoardCornerPositions](#) (Size boardSize, float squareSize, std::vector< Point3f > &corners)
- void [getEulerAngles](#) (Mat &rotCamerMatrix, Vec3d &eulerAngles)
- int [startTracking](#) ()
- void [startStopCamera](#) ()
Start or stop the tracking depending on if the camera is currently running or not.
- int [setReference](#) ()
- int [calibrateCamera](#) ()
Start the camera calibration routine that computes the camera matrix and distortion coefficients.
- void [loadCalibration](#) (int method)
- void [testAlgorithms](#) ()
- void [projectCoordinateFrame](#) (Mat pictureFrame)
- void [setUpUDP](#) ()
Open the UDP ports for communication.
- void [setHeadingOffset](#) (double d)
- void [sendDataUDP](#) (cv::Vec3d &Position, cv::Vec3d &Euler)
- void [closeUDP](#) ()
- void [loadMarkerConfig](#) (int method)
- void [drawPositionText](#) (cv::Mat &Picture, cv::Vec3d &Position, cv::Vec3d &Euler, double error)
- void [loadCameraPosition](#) ()
- int [determineExposure](#) ()
- void [determineOrder](#) ()
- int [calibrateGround](#) ()

Variables

- `commObject commObj`
class that handles the communication from `main.cpp` to the GUI
- `bool safetyEnable = false`
is the safety feature enabled
- `bool safety2Enable = false`
is the second receiver enabled
- `double safetyBoxLength = 1.5`
length of the safety area cube in meters
- `int safetyAngle = 30`
bank and pitch angle protection in degrees
- `bool exitRequested = true`
variable if tracking loop should be exited
- `int invertZ = 1`
dummy variable to invert Z direction on request
- `double frameTime = 0.01`
100 Hz CoSy rate, is later on replaced with the hardware timestamp delivered by the camera
- `double timeOld = 0.0`
old time for finite differences velocity calculation. Is later on replaced with the hardware timestamp delivered by the camera
- `double timeFirstFrame = 0`
Time stamp of the first frame. This value is then subtracted for every other frame so the time in the log start at zero.
- `Vec3d position = Vec3d()`
position vector x,y,z for object position in O-CoSy, unit is meter
- `Vec3d eulerAngles = Vec3d()`
Roll Pitch Heading in this order, units in degrees.
- `Vec3d positionOld = Vec3d()`
old position in O-CoSy for finite differences velocity calculation
- `Vec3d velocity = Vec3d()`
velocity vector of object in o-CoSy in respect to o-CoSy
- `Vec3d posRef = Vec3d()`
initial position of object in camera CoSy
- `Vec3d eulerRef = Vec3d()`
initial euler angle of object respectivley to camera CoSy
- `double headingOffset = 0`
heading offset variable for aligning INS heading with tracking heading
- `int intIntensity = 15`
max infrared spot light intensity is 15 1-6 is strobe 7-15 is continuous 13 and 14 are meaningless
- `int intExposure = 1`
max is 480 increase if markers are badly visible but should be determined automatically during `setReference()`
- `int intFrameRate = 100`
CoSy rate of camera, maximum is 100 fps.
- `int intThreshold = 200`
threshold value for marker detection. If markers are badly visible lower this value but should not be necessary
- `Mat Rmat = (cv::Mat_<double>(3, 1) << 0.0, 0.0, 0.0)`
Rotation, translation etc. matrix for PnP results.
- `Mat RmatRef = (cv::Mat_<double>(3, 3) << 1., 0., 0., 0., 1., 0., 0., 0., 1.)`
reference rotation matrix from camera CoSy to marker CoSy
- `Mat M_CN = cv::Mat_<double>(3, 3)`
rotation matrix from camera to ground, fixed for given camera position

- Mat `M_HeadingOffset` = `cv::Mat_<double>(3, 3)`
rotation matrix that turns the ground system to the INS magnetic heading for alignment
- Mat `Rvec` = `(cv::Mat_<double>(3, 1) << 0.0, 0.0, 0.0)`
rotation vector (axis-angle notation) from camera CoSy to marker CoSy
- Mat `Tvec` = `(cv::Mat_<double>(3, 1) << 0.0, 0.0, 0.0)`
translation vector from camera CoSy to marker CoSy in camera CoSy
- Mat `RvecOriginal`
initial values as start values for algorithms and algorithm tests
- Mat `TvecOriginal`
initial values as start values for algorithms and algorithm tests
- bool `useGuess` = true
set to true and the algorithm uses the last result as starting value
- int `methodPNP` = 0
solvePNP algorithm 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP //!< 4 and 1 are the same and not implemented correctly by OpenCV
- int `numberMarkers` = 4
number of markers. Is loaded during start up from the marker configuration file
- `std::vector< Point3d >` `list_points3d`
marker positions in marker CoSy
- `std::vector< Point2d >` `list_points2d`
marker positions projected in 2D in camera image CoSy
- `std::vector< Point2d >` `list_points2dOld`
marker positions in previous picture in 2D in camera image CoSy
- `std::vector< double >` `list_points2dDifference`
difference of the old and new 2D marker position to determine the order of the points
- `std::vector< Point2d >` `list_points2dProjected`
3D marker points projected to 2D in camera image CoSy with the algorithm projectPoints
- `std::vector< Point2d >` `list_points2dUnsorted`
marker points in 2D camera image CoSy, sorted with increasing x (camera image CoSy) but not sorted to correspond with list_points3d
- `std::vector< Point3d >` `coordinateFrame`
coordinate visualisazion of marker CoSy
- `std::vector< Point2d >` `coordinateFrameProjected`
marker CoSy projected from 3D to 2D camera image CoSy
- int `pointOrderIndices` [] = { 0, 1, 2, 3 }
old correspondence from list_points3d and list_points_2d
- int `pointOrderIndicesNew` [] = { 0, 1, 2, 3 }
new correspondence from list_points3d and list_points_2d
- double `currentPointDistance` = 5000
distance from the projected 3D points (hence in 2d) to the real 2d marker positions in camera image CoSy
- double `minPointDistance` = 5000
minimum distance from the projected 3D points (hence in 2d) to the real 2d marker positions in camera image CoSy
- int `currentMinIndex` = 0
helper variable set to the point order that holds the current minimum point distance
- bool `gotOrder` = false
order of the list_points3d and list_points3d already tetermined or not, has to be done once
- bool `camera_started` = false
variable thats needed to exit the main while loop
- Mat `cameraMatrix`
camera matrix of the camera
- Mat `distCoeffs`

- distortion coefficients of the camera*
- Core::DistortionModel [distModel](#)
- distortion model of the camera*
- QUdpSocket * [udpSocketObject](#)
- socket for the communication with receiver 1*
- QUdpSocket * [udpSocketSafety](#)
- socket for the communication with safety receiver*
- QUdpSocket * [udpSocketSafety2](#)
- socket for the communication with receiver 3*
- QHostAddress [IPAdressObject](#) = QHostAddress("127.0.0.1")
- IPv4 adress of receiver 1.*
- QHostAddress [IPAdressSafety](#) = QHostAddress("192.168.4.1")
- IPv4 adress of safety receiver.*
- QHostAddress [IPAdressSafety2](#) = QHostAddress("192.168.4.4")
- IPv4 adress of receiver 2.*
- int [portObject](#) = 9155
- Port of receiver 1.*
- int [portSafety](#) = 9155
- Port of the safety receiver.*
- int [portSafety2](#) = 9155
- Port of receiver 2.*
- QByteArray [datagram](#)
- data package that is sent to receiver 1 and 2*
- QByteArray [data](#)
- data package that's sent to the safety receiver*
- const int [BACKBUFFER_BITSPERPIXEL](#) = 8
- 8 bit per pixel and greyscale image from camera*
- std::string [strBuf](#)
- buffer that holds the strings that are sent to the Qt GUI*
- std::stringstream [ss](#)
- stream that sends the strBuf buffer to the Qt GUI*
- QString [logFileName](#)
- Filename for the logfiles.*
- std::string [logName](#)
- Filename for the logfiles as standard string.*
- SYSTEMTIME [logDate](#)
- Systemtime struct that saves the current date and time thats needed for the log file name creation.*
- std::ofstream [logfile](#)
- file handler for writing the log file*

6.5.1 Detailed Description

Rigid Track main file that contains most functionality.

This file contains almost 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

6.5.2 Function Documentation**6.5.2.1 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 229 of file main.cpp.

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

Here is the caller graph for this function:



6.5.2.2 calibrateCamera()

```
int calibrateCamera ( )
```

Start the camera calibration routine that computes the camera matrix and distortion coefficients.

Definition at line 774 of file main.cpp.

```

775 {
776     commObj.addLog("Started camera calibration. 80 pictures are going to be captured.");
777     CameraLibrary_EnableDevelopment();
778
779     /// Initialize Camera SDK ===
780     CameraLibrary::CameraManager::X();
781
782     /// At this point the Camera SDK is actively looking for all connected cameras and will initialize
783     /// them on it's own.
784
785     /// Get a connected camera =====
786     CameraManager::X().WaitForInitialization();
787
788     Camera *camera = CameraManager::X().GetCamera();
789     if (camera == 0)
790     {
791         commObj.addLog("No camera found!");
792         return 1;
793     }
794
795     /// Determine camera resolution
796     int cameraWidth = camera->Width();
797     int cameraHeight = camera->Height();
798
799     /// Set Video Mode ===
800
801     /// We set the camera to Segment Mode here. This mode is support by all of our products.
802     /// Depending on what device you have connected you might want to consider a different
803     /// video mode to achieve the best possible tracking quality. All devices that support a
804     /// mode that will achieve a better quality output with a mode other than Segment Mode are
805     /// listed here along with what mode you should use if you're looking for the best head
806     /// tracking:
807     ///
808     ///      V100:R1/R2      Precision Mode
809     ///      TrackIR 5       Bit-Packed Precision Mode
810     ///      V120            Precision Mode
811     ///      TBar            Precision Mode
812     ///      S250e          Precision Mode
813     ///
814     /// If you have questions about a new device that might be conspicuously missing here or
815     /// have any questions about head tracking, email support or participate in our forums.
816
817     camera->SetVideoType(Core::GrayscaleMode);
818
819     /// Start camera output ===
820     camera->Start();
821
822     /// Camera Matrix creation ===
823     cameraMatrix = Mat::eye(3, 3, CV_64F);
824     distCoeffs = Mat::zeros(8, 1, CV_64F);
825
826     /// Ok, start main loop. This loop fetches and displays =====
827     /// camera frames.
828     /// But first set some camera parameters
829     camera->SetAGC(false);
830     camera->SetAEC(false);
831     camera->SetExposure(200);
832     camera->SetIntensity(4);
833     camera->SetFrameRate(30);
834     camera->SetIRFilter(true);
835     camera->SetContinuousIR(false);
836     camera->SetHighPowerMode(false);
837
838     int number_samples = 0;
839     int imagesToSample = 80;
840
841     std::vector<std::vector<Point2f>> > imagePoints;
842     std::vector<Point2f> pointBuf;
843     bool found;
844     Size boardSize(9, 6);
845     Size imageSize(cameraWidth, cameraHeight);
846     Mat Rvec(3, 1, DataType<double>::type);
847     Mat Tvec(3, 1, DataType<double>::type);
848

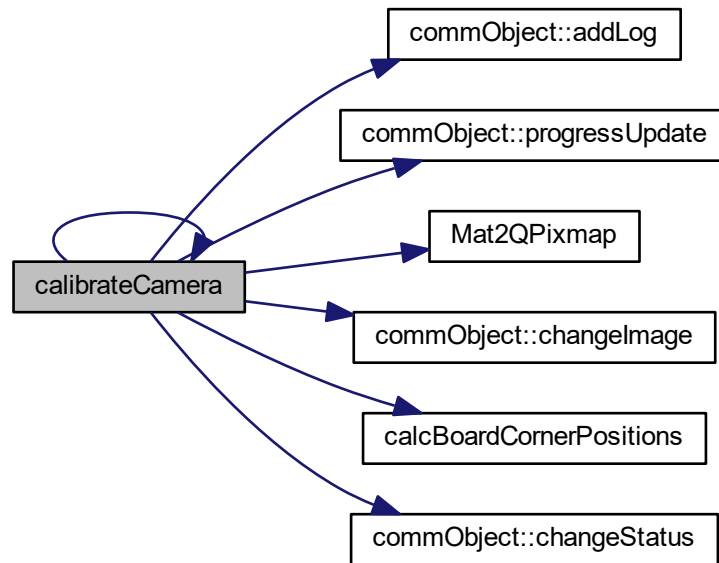
```

```

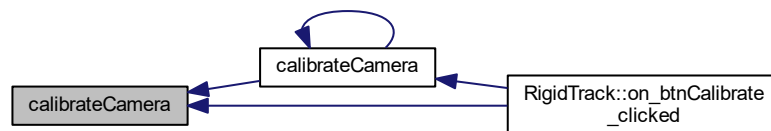
849     //! the user has to provide the size of one square in mm
850     bool ok;
851     int qsquareSize = QInputDialog::getInt(nullptr, "Chessboard size in mm", "Chessboard size in mm", 23, 1
, 60, 1, &ok);
852     float squareSize = 23;
853
854     if (ok)
855     {
856         squareSize = qsquareSize;
857     }
858
859     QPixmap QPFrame;
860     commObj.progressUpdate(0);
861     while (number_samples < imagesToSample)
862     {
863         //! Fetch a new frame from the camera =====
864         cv::Mat matFrame(cv::Size(cameraWidth, cameraHeight), CV_8UC1);
865
866         //! which is why we also set this constant to 8
867         const int BACKBUFFER_BITSPERPIXEL = 8;
868
869         //! later on, when we get the frame as usual:
870         CameraLibrary::Frame * frame = camera->GetFrame();
871
872         if (frame)
873         {
874             //! Lets have the Camera Library raster the camera's
875             //! image into our texture.
876
877             frame->Rasterize(cameraWidth, cameraHeight, matFrame.step, BACKBUFFER_BITSPERPIXEL, matFrame.
data);
878             QPFrame = Mat2QPixmap(matFrame);
879             commObj.changeImage(QPFrame);
880             found = findChessboardCorners(matFrame, boardSize, pointBuf, CV_CALIB_CB_ADAPTIVE_THRESH |
CV_CALIB_CB_FAST_CHECK | CV_CALIB_CB_NORMALIZE_IMAGE);
881
882             if (found)                //!< If done with success,
883             {
884                 //! improve the found corners' coordinate accuracy for chessboard
885                 cornerSubPix(matFrame, pointBuf, Size(11, 11), Size(-1, -1), TermCriteria(CV_TERMCRIT_EPS +
CV_TERMCRIT_ITER, 30, 0.1));
886
887                 imagePoints.push_back(pointBuf);
888                 number_samples += 1;
889                 commObj.addLog(QString::fromStdString(ss.str()));
890                 QCoreApplication::processEvents();
891             }
892             frame->Release();
893             ss.str("");
894             ss << "Samples found = " << number_samples;
895             commObj.progressUpdate(number_samples * 100 / imagesToSample);
896         }
897         Sleep(2);
898     }
899
900     std::vector<std::vector<Point3f> > objectPoints(1);
901     calcBoardCornerPositions(boardSize, squareSize, objectPoints[0]);
902     objectPoints.resize(imagePoints.size(), objectPoints[0]);
903
904     double rms = calibrateCamera(objectPoints, imagePoints, imageSize,
cameraMatrix, distCoeffs, Rvec, Tvec);
905     commObj.progressUpdate(0);
906     //! Release camera ----
907     camera->Release();
908
909     //! Save the obtained calibration coefficients in a file for later use
910     QString fileName = QFileDialog::getSaveFileName(nullptr, "Save calibration file", "", "Calibration File
(*.xml);;All Files (*)");
911     FileStorage fs(fileName.toUtf8().constData(), FileStorage::WRITE);
912     fs << "CameraMatrix" << cameraMatrix;
913     fs << "DistCoeff" << distCoeffs;
914     fs << "RMS" << rms;
915     strBuf = fs.releaseAndGetString();
916     commObj.changeStatus(QString::fromStdString(strBuf));
917     commObj.addLog("Saved calibration!");
918     return 0;
919 }

```


Here is the call graph for this function:



Here is the caller graph for this function:



6.5.2.3 calibrateGround()

```
int calibrateGround ( )
```

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

Definition at line 1563 of file `main.cpp`.

```

1564 {
1565     //! initialize the variables with starting values
1566     gotOrder = false;
1567     posRef = 0;
1568     eulerRef = 0;
1569     RmatRef = 0;
1570     Rvec = RvecOriginal;
1571     Tvec = TvecOriginal;
1572
1573     determineExposure();
1574
1575     ss.str("");
1576     commObj.addLog("Started ground calibration");
1577
1578     CameraLibrary_EnableDevelopment();
1579     //! Initialize Camera SDK ===
1580     CameraLibrary::CameraManager::X();
1581
1582     //! At this point the Camera SDK is actively looking for all connected cameras and will initialize
1583     //! them on it's own.
1584
1585     //! Get a connected camera =====
1586     CameraManager::X().WaitForInitialization();
1587     Camera *camera = CameraManager::X().GetCamera();
1588
1589     //! If no device connected, pop a message box and exit ===
1590     if (camera == 0)
1591     {
1592         commObj.addLog("No camera found!");
1593         return 1;
1594     }
1595
1596     //! Determine camera resolution to size application window =====
1597     int cameraWidth = camera->Width();
1598     int cameraHeight = camera->Height();
1599     camera->GetDistortionModel(distModel);
1600     cv::Mat matFrame(cv::Size(cameraWidth, cameraHeight), CV_8UC1);
1601
1602     //! Set camera mode to precision mode, it directly provides marker coordinates
1603     camera->SetVideoType(Core::PrecisionMode);
1604
1605     //! Start camera output ===
1606     camera->Start();
1607
1608     //! Turn on some overlay text so it's clear things are =====
1609     //! working even if there is nothing in the camera's view. =====
1610     //! Set some other parameters as well of the camera
1611     camera->SetTextOverlay(true);
1612     camera->SetFrameRate(intFrameRate);
1613     camera->SetIntensity(intIntensity);
1614     camera->SetIRFilter(true);
1615     camera->SetContinuousIR(false);
1616     camera->SetHighPowerMode(false);
1617
1618     //! sample some frames and calculate the position and attitude. then average those values and use that
1619     as zero position
1620     int numberSamples = 0;
1621     int numberToSample = 200;
1622     double projectionError = 0;
1623     while (numberSamples < numberToSample)
1624     {
1625         //! Fetch a new frame from the camera =====
1626         Frame *frame = camera->GetFrame();
1627
1628         if (frame)
1629         {
1630             //! Ok, we've received a new frame, lets do something
1631             //! with it.
1632             if (frame->ObjectCount() == numberMarkers)
1633             {
1634                 //!for(int i=0; i<frame->ObjectCount(); i++)
1635                 for (int i = 0; i < numberMarkers; i++)
1636                 {
1637                     cObject *obj = frame->Object(i);
1638                     list_points2dUnsorted[i] = cv::Point2d(obj->X(), obj->Y());
1639                 }
1640
1641                 if (gotOrder == false)
1642                 {
1643                     determineOrder();
1644                 }
1645
1646                 //! sort the 2d points with the correct indices as found in the preceeding order
1647                 determination algorithm
1648                 for (int w = 0; w < numberMarkers; w++)
1649                 {

```

```

1649         list_points2d[w] = list_points2dUnsorted[
1650             pointOrderIndices[w]];
1651         list_points2dOld = list_points2dUnsorted;
1652
1653         ///!Compute the pose from the 3D-2D correspondences
1654         solvePnP(list_points3d, list_points2d,
1655             cameraMatrix, distCoeffs, Rvec, Tvec, useGuess,
1656             methodPNP);
1657
1658         ///! project the marker 3d points with the solution into the camera image CoSy and calculate
1659         difference to true camera image
1660         projectPoints(list_points3d, Rvec, Tvec,
1661             cameraMatrix, distCoeffs, list_points2dProjected);
1662         projectionError = norm(list_points2dProjected,
1663             list_points2d);
1664
1665         if (projectionError > 3)
1666         {
1667             commObj.addLog("Reprojection error is bigger than 3 pixel. Correct marker
1668                 configuration loaded?\nMarker position measured precisely?");
1669             frame->Release();
1670             return 1;
1671         }
1672
1673         double maxValue = 0;
1674         double minValue = 0;
1675         minMaxLoc(Tvec.at<double>(2), &minValue, &maxValue);
1676
1677         if (maxValue > 10000 || minValue < 0)
1678         {
1679             commObj.addLog("Negative z distance, thats not possible. Start the set
1680                 zero routine again and check marker configurations.");
1681             frame->Release();
1682             return 1;
1683         }
1684
1685         if (norm(positionOld) - norm(Tvec) < 0.05)    ///!<Iterative Method needs time
1686             to converge to solution
1687         {
1688             add(posRef, Tvec, posRef);
1689             add(eulerRef, Rvec, eulerRef);    ///!< That are not the values of yaw,
1690             roll and pitch yet! Rodriguez has to be called first.
1691             numberSamples++;    ///!<-- one sample more :D
1692             commObj.progressUpdate(numberSamples * 100 / numberToSample);
1693         }
1694         positionOld = Tvec;
1695
1696         Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
1697         for (int i = 0; i < numberMarkers; i++)
1698         {
1699             circle(cFrame, Point(list_points2d[i].x,
1700                 list_points2d[i].y), 6, Scalar(0, 225, 0), 3);
1701             projectCoordinateFrame(cFrame);
1702             projectPoints(list_points3d, Rvec, Tvec,
1703                 cameraMatrix, distCoeffs, list_points2d);
1704             for (int i = 0; i < numberMarkers; i++)
1705             {
1706                 circle(cFrame, Point(list_points2d[i].x,
1707                     list_points2d[i].y), 3, Scalar(225, 0, 0), 3);
1708             }
1709
1710             QPixmap QPFrame;
1711             QPFrame = Mat2QPixmap(cFrame);
1712             commObj.changeImage(QPFrame);
1713             QApplication::processEvents();
1714         }
1715         frame->Release();
1716     }
1717
1718     ///! Release camera ----
1719     camera->Release();
1720
1721     ///!Divide by the number of samples to get the mean of the reference position
1722     divide(posRef, numberToSample, posRef);
1723     divide(eulerRef, numberToSample, eulerRef);    ///!< eulerRef is here in Axis Angle
1724     notation
1725
1726     Rodrigues(eulerRef, RmatRef);    ///!< axis angle to rotation matrix
1727
1728     getEulerAngles(RmatRef, eulerRef);    ///!< rotation matrix to euler
1729     ss.str("");
1730     ss << "RmatRef is:\n";

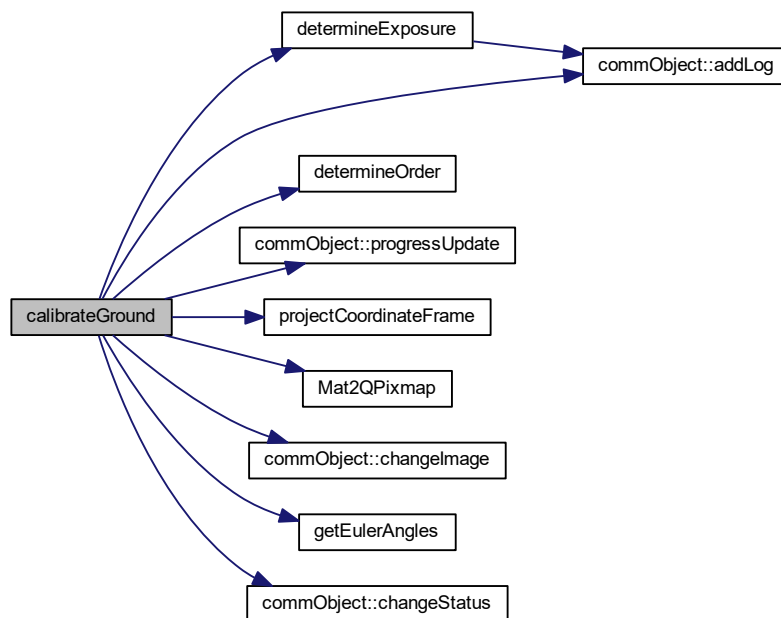
```

```

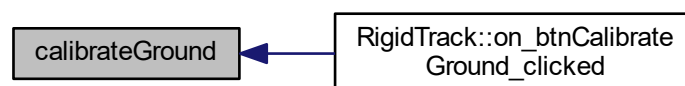
1722     ss << RmatRef << "\n";
1723     ss << "Reference Position is:\n";
1724     ss << posRef << "[mm] \n";
1725     ss << "Reference Euler angles are:\n";
1726     ss << eulerRef << "[deg] \n";
1727
1728     //! Save the obtained calibration coefficients in a file for later use
1729     QString fileName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

```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.2.4 closeUDP()

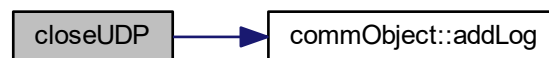
```
void closeUDP ( )
```

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

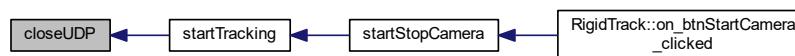
Definition at line 1173 of file main.cpp.

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

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.2.5 determineExposure()

```
int determineExposure ( )
```

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

Definition at line 1362 of file main.cpp.

```

1363 {
1364     ///! For OptiTrack Ethernet cameras, it's important to enable development mode if you
1365     ///! want to stop execution for an extended time while debugging without disconnecting
1366     ///! the Ethernet devices. Lets do that now:
1367
1368     CameraLibrary_EnableDevelopment();
1369
1370     ///! Initialize Camera SDK ---
1371     CameraLibrary::CameraManager::X();
1372
1373     ///! At this point the Camera SDK is actively looking for all connected cameras and will initialize
1374     ///! them on it's own.
1375
1376     ///! Get a connected camera =====
1377     CameraManager::X().WaitForInitialization();
1378     Camera *camera = CameraManager::X().GetCamera();
1379
1380     ///! If no device connected, pop a message box and exit ---
1381     if (camera == 0)
1382     {
1383         commObj.addLog("No camera found!");
1384         return 1;
1385     }
1386
1387     ///! Determine camera resolution to size application window -----
1388     int cameraWidth = camera->Width();
1389     int cameraHeight = camera->Height();
1390
1391     camera->SetVideoType(Core::PrecisionMode); ///! set the camera mode to precision mode, it used
greyscale information for marker property calculations
1392
1393     ///! Start camera output ---
1394     camera->Start();
1395
1396     ///! Turn on some overlay text so it's clear things are -----
1397     ///! working even if there is nothing in the camera's view. -----
1398     camera->SetTextOverlay(true);
1399     camera->SetExposure(intExposure); ///! set the camera exposure
1400     camera->SetIntensity(intIntensity); ///! set the camera infrared LED intensity
1401     camera->SetFrameRate(intFrameRate); ///! set the camera framerate to 100 Hz
1402     camera->SetIRFilter(true); ///! enable the filter that blocks visible light and only passes infrared
light
1403     camera->SetHighPowerMode(true); ///! enable high power mode of the leds
1404     camera->SetContinuousIR(false); ///! enable continuous LED light
1405     camera->SetThreshold(intThreshold); ///! set threshold for marker detection
1406
1407     ///!set exposure such that num markers are visible
1408     int numberObjects = 0; ///! Number of objects (markers) found in the current picture with the given
exposure
1409     int minExposure = 1; ///! exposure when objects detected the first time is numberMarkers
1410     int maxExposure = 480; ///! exposure when objects detected is first time numberMarkers+1
1411     intExposure = minExposure; ///! set the exposure to the smallest value possible
1412     int numberTries = 0; ///! if the markers arent found after numberTries then there might be no markers
at all in the real world
1413
1414     ///! Determine minimum exposure, hence when are numberMarkers objects detected
1415     camera->SetExposure(intExposure);
1416     while (numberObjects != numberMarkers && numberTries < 48)
1417     {
1418         ///! get a new camera frame
1419         Frame *frame = camera->GetFrame();
1420         if (frame) ///! frame received
1421         {
1422             numberObjects = frame->ObjectCount(); ///! how many objects are detected in the image
1423             if (numberObjects == numberMarkers) { minExposure =
intExposure; frame->Release(); break; } ///! if the right amount if markers is found, exit while
loop
1424             ///! not the right amount of markers was found so increase the exposure and try again
1425             numberTries++;
1426             intExposure += 10;
1427             camera->SetExposure(intExposure);
1428             ss.str("");
1429             ss << "Exposure: " << intExposure << "\t";
1430             ss << "Objects found: " << numberObjects;
1431             commObj.addLog(QString::fromStdString(ss.str()));
1432             frame->Release();
1433         }
1434     }
1435
1436     ///! Now determine maximum exposure, hence when are numberMarkers+1 objects detected
1437     numberTries = 0; ///! if the markers arent found after numberTries then there might be no markers at
all in the real world
1438     intExposure = maxExposure;
1439     camera->SetExposure(intExposure);
1440     numberObjects = 0;
1441     while (numberObjects != numberMarkers && numberTries < 48)
1442     {

```

```

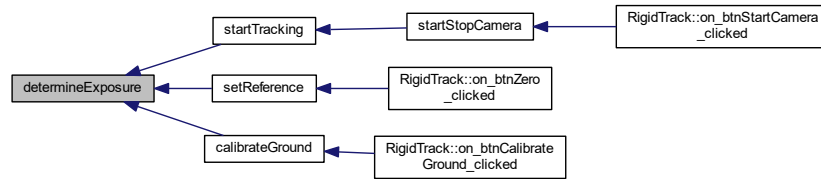
1443     Frame *frame = camera->GetFrame();
1444     if (frame)
1445     {
1446         numberObjects = frame->ObjectCount(); //! how many objects are detected in the image
1447         if (numberObjects == numberMarkers) { maxExposure =
intExposure; frame->Release(); break; } //! if the right amount of markers is found, exit while
        loop
1448
1449         //! not the right amount of markers was found so decrease the exposure and try again
1450         intExposure -= 10;
1451         numberTries++;
1452         camera->SetExposure(intExposure);
1453         ss.str("");
1454         ss << "Exposure: " << intExposure << "\t";
1455         ss << "Objects found: " << numberObjects;
1456         commObj.addLog(QString::fromStdString(ss.str()));
1457         frame->Release();
1458     }
1459 }
1460
1461 //! set the exposure to the mean of min and max exposure determined
1462 camera->SetExposure((minExposure + maxExposure) / 2.0);
1463
1464 //! and now check if the correct amount of markers is detected with that new value
1465 while (1)
1466 {
1467     Frame *frame = camera->GetFrame();
1468     if (frame)
1469     {
1470         numberObjects = frame->ObjectCount(); //! how many objects are detected in the image
1471         if (numberObjects != numberMarkers) //! are all markers and not more or less
detected in the image
1472         {
1473             frame->Release();
1474             commObj.addLog("Was not able to detect the right amount of markers.");
1475             //! Release camera ==--
1476             camera->Release();
1477             return 1;
1478         }
1479         else //! all markers and not more or less are found
1480         {
1481             frame->Release();
1482             intExposure = (minExposure + maxExposure) / 2.0;
1483             commObj.addLog("Found the correct number of markers.");
1484             commObj.addLog("Exposure set to:");
1485             commObj.addLog(QString::number(intExposure));
1486             break;
1487         }
1488     }
1489 }
1490
1491 camera->Release();
1492 return 0;
1493
1494 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.2.6 determineOrder()

```
void determineOrder ( )
```

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

Definition at line 1498 of file main.cpp.

```

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

```

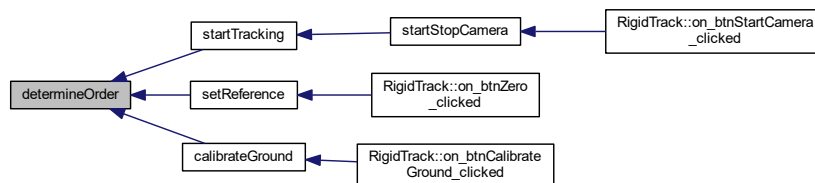


```

        smallest value of difference to the current one
1542         for (int b = 0; b < numberMarkers; b++)    //!< now safe the better permutation
1543         {
1544             pointOrderIndicesNew[b] = pointOrderIndices[b];
1545         }
1546     }
1547
1548 }
1549
1550 //!< try every permutation
1551 while (std::next_permutation(pointOrderIndices,
pointOrderIndices + 4));
1552
1553 //!< now that the correct order is found assign it to the indices array
1554 for (int w = 0; w < numberMarkers; w++)
1555 {
1556     pointOrderIndices[w] = pointOrderIndicesNew[w];
1557 }
1558 gotOrder = true;
1559 }

```

Here is the caller graph for this function:



6.5.2.7 drawPositionText()

```

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

```

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

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 1315 of file main.cpp.

```

1316 {
1317     ss.str("");
1318     ss << "X: " << Position[0] << " m";
1319     putText(Picture, ss.str(), cv::Point(200, 440), 1, 1, cv::Scalar(255, 255, 255));

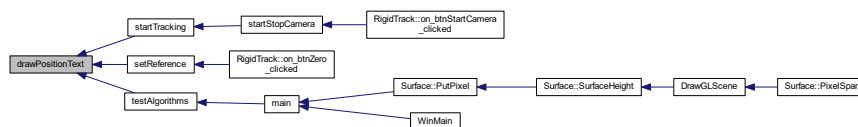
```

```

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

```

Here is the caller graph for this function:



6.5.2.8 getEulerAngles()

```

void getEulerAngles (
    Mat & rotCamerMatrix,
    Vec3d & eulerAngles )

```

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 rotCamerMatrix.

Definition at line 241 of file main.cpp.

```

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

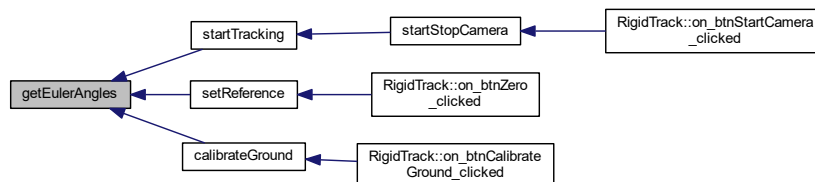
```

```

251     rotMatrix,
252     transVect,
253     rotMatrixX,
254     rotMatrixY,
255     rotMatrixZ,
256     eulerAngles);
257 }

```

Here is the caller graph for this function:



6.5.2.9 loadCalibration()

```

void loadCalibration (
    int method )

```

Load a previously saved camera calibration from a file.

Parameters

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

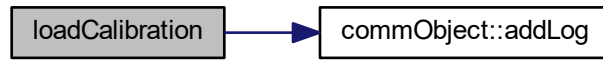
Definition at line 923 of file main.cpp.

```

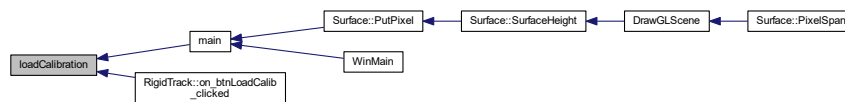
923                                     {
924
925     QString fileName;
926     if (method == 0)
927     {
928         fileName = "calibration.xml";
929     }
930     else
931     {
932         fileName = QFileDialog::getOpenFileName(nullptr, "Choose a previous saved calibration file", "", "
Calibration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

```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.2.10 loadCameraPosition()

```
void loadCameraPosition ( )
```

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

Definition at line 1348 of file main.cpp.

```

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

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.2.11 loadMarkerConfig()

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

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

Parameters

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

Definition at line 1195 of file main.cpp.

```

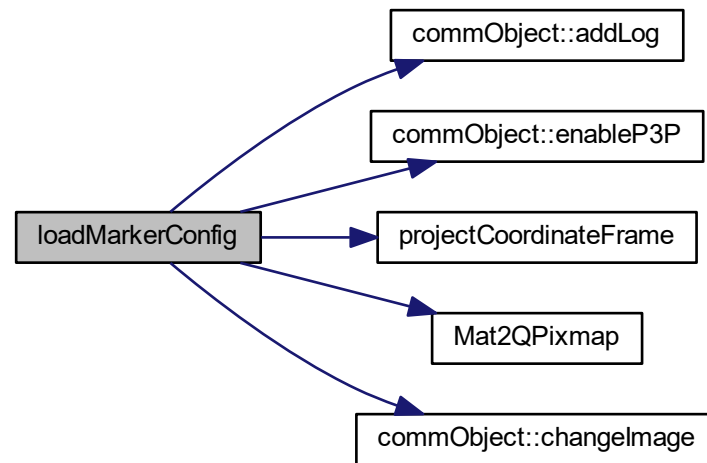
1196 {
1197     QString fileName;
1198     ///! during start up of the programm load the standard marker configuration
1199     if (method == 0)
1200     {
1201         ///! open the standard marker configuration file
1202         FileStorage fs;
1203         fs.open("markerStandard.xml", FileStorage::READ);
1204
1205         ///! copy the values to the respective variables
1206         fs["numberMarkers"] >> numberMarkers;
1207
1208         ///! initialize vectors with correct length depending on the number of markers
1209         list_points3d = std::vector<Point3d>(numberMarkers);
1210         list_points2d = std::vector<Point2d>(numberMarkers);
1211         list_points2d0ld = std::vector<Point2d>(numberMarkers);
1212         list_points2dDifference = std::vector<double>(
numberMarkers);
1213         list_points2dProjected = std::vector<Point2d>(
numberMarkers);
1214         list_points2dUnsorted = std::vector<Point2d>(
numberMarkers);
1215
1216         ///! save the marker locations in the points3d vector
1217         fs["list_points3d"] >> list_points3d;
1218         fs.release();
1219         commObj.addLog("Loaded marker configuration from file:");
1220         commObj.addLog(fileName);
1221
1222
1223     }
1224     else
1225     {
1226         ///! if the load marker configuration button was clicked show a open file dialog
1227         fileName = QFileDialog::getOpenFileName(nullptr, "Choose a previous saved marker configuration file
", "", "marker configuratio files (*.xml);;All Files (*)");
1228
1229         ///! was cancel or abort clicked
1230         if (fileName.length() == 0)
1231         {
1232
```

```

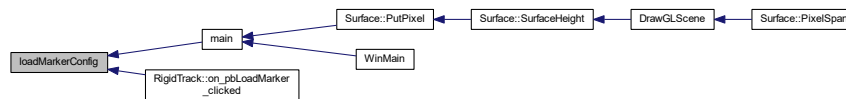
1233         //!< if yes load the standard marker configuration
1234         fileName = "markerStandard.xml";
1235     }
1236
1237     //!< open the selected marker configuration file
1238     FileStorage fs;
1239     fs.open(fileName.toUtf8().constData(), FileStorage::READ);
1240
1241     //!< copy the values to the respective variables
1242     fs["numberMarkers"] >> numberMarkers;
1243
1244     //!< initialize vectors with correct length depending on the number of markers
1245     list_points3d = std::vector<Point3d>(numberMarkers);
1246     list_points2d = std::vector<Point2d>(numberMarkers);
1247     list_points2dOld = std::vector<Point2d>(numberMarkers);
1248     list_points2dDifference = std::vector<double>(numberMarkers);
1249     list_points2dProjected = std::vector<Point2d>(numberMarkers);
1250     list_points2dUnsorted = std::vector<Point2d>(numberMarkers);
1251
1252     //!< save the marker locations in the points3d vector
1253     fs["list_points3d"] >> list_points3d;
1254     fs.release();
1255     commObj.addLog("Loaded marker configuration from file:");
1256     commObj.addLog(fileName);
1257
1258 }
1259
1260 //!< Print out the number of markers and their position to the GUI
1261 ss.str("");
1262 ss << "Number of Markers: " << numberMarkers << "\n";
1263 ss << "Marker 3D Points X,Y and Z [mm]: \n";
1264 for (int i = 0; i < numberMarkers; i++)
1265 {
1266     ss << "Marker " << i + 1 << ":\t" << list_points3d[i].x << "\t" << list_points3d[i].y << "\t" <<
list_points3d[i].z << "\n";
1267 }
1268 commObj.addLog(QString::fromStdString(ss.str()));
1269
1270 //!< check if P3P algorithm can be enabled, it needs exactly 4 marker points to work
1271 if (numberMarkers == 4)
1272 {
1273     //!< if P3P is possible, let the user choose which algorithm he wants but keep iterative active
1274     methodPNP = 0;
1275     commObj.enableP3P(true);
1276 }
1277 else
1278 {
1279     //!< More (or less) marker than 4 loaded, P3P is not possible, hence user cant select P3P in GUI
1280     methodPNP = 0;
1281     commObj.enableP3P(false);
1282     commObj.addLog("P3P algorithm disabled, only works with 4 markers.");
1283 }
1284
1285 //!< now display the marker configuration in the camera view
1286 Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
1287
1288 //!< Set the camera pose parallel to the marker coordinate system
1289 Tvec.at<double>(0) = 0;
1290 Tvec.at<double>(1) = 0;
1291 Tvec.at<double>(2) = 4500;
1292 Rvec.at<double>(0) = 0 * 3.141592653589 / 180.0;
1293 Rvec.at<double>(1) = 0 * 3.141592653589 / 180.0;
1294 Rvec.at<double>(2) = -90. * 3.141592653589 / 180.0;
1295
1296 projectPoints(list_points3d, Rvec, Tvec, cameraMatrix,
distCoeffs, list_points2dProjected);
1297 for (int i = 0; i < numberMarkers; i++)
1298 {
1299     circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 3, Scalar(255, 0, 0
), 3);
1300 }
1301
1302 projectCoordinateFrame(cFrame);
1303 QPixmap QPFrame;
1304 QPFrame = Mat2QPixmap(cFrame);
1305 commObj.changeImage(QPFrame);
1306 QCoreApplication::processEvents();
1307
1308 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.2.12 main()

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

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

First the GUI is set up with Signals and Slots, see Qt docu for how that works. Then some variables are initialized with arbitrary values. At last calibration and marker configuration etc. are loaded from xml files.

Parameters

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

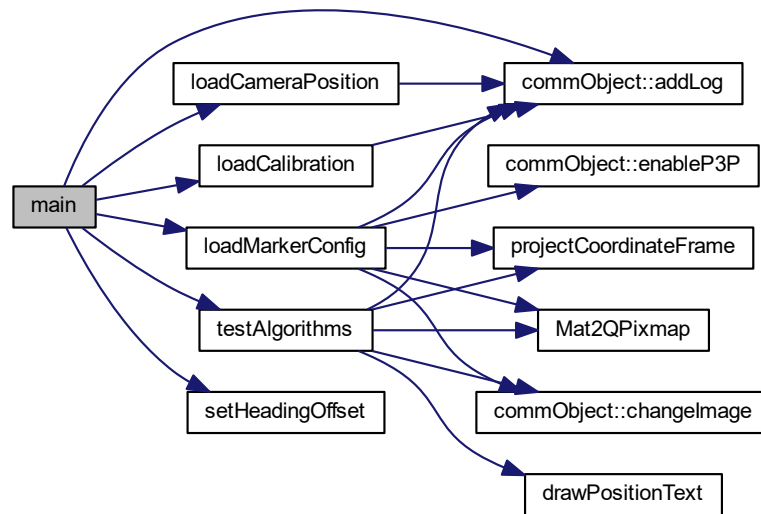
Definition at line 156 of file `main.cpp`.

```

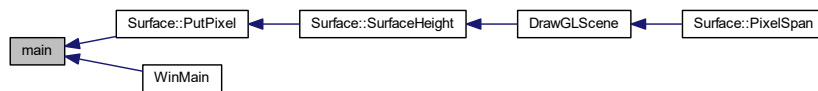
157 {
158     QApplication a(argc, argv);
159     RigidTrack w;
160     w.show();    //!< show the GUI
161     //!< connect the Qt slots and signals for event handling
162     QObject::connect(&commObj, SIGNAL(statusChanged(QString)), &w, SLOT(setStatus(QString)),
Qt::DirectConnection);
163     QObject::connect(&commObj,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

```


Here is the call graph for this function:



Here is the caller graph for this function:



6.5.2.13 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 216 of file main.cpp.

```

217 {
218     QImage dest((const uchar *)src.data, src.cols, src.rows, src.step, QImage::Format_RGB888);

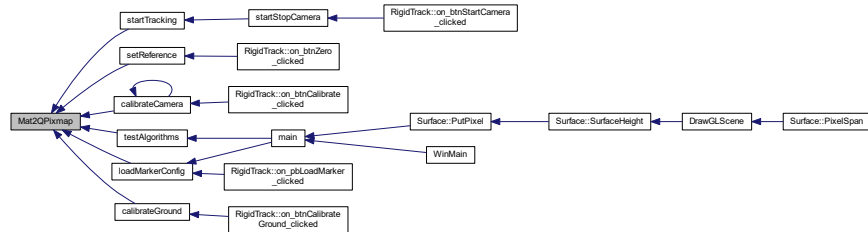
```

```

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

```

Here is the caller graph for this function:



6.5.2.14 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.
----	---------------------	---

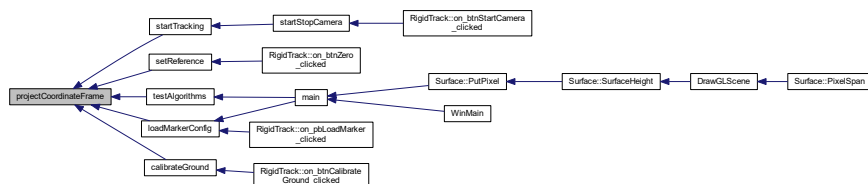
Definition at line 1081 of file main.cpp.

```

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

```

Here is the caller graph for this function:



6.5.2.15 sendDataUDP()

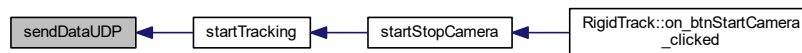
```
void sendDataUDP (
    cv::Vec3d & Position,
    cv::Vec3d & Euler )
```

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

Definition at line 1154 of file main.cpp.

```
1155 {
1156     datagram.clear();
1157     QDataStream out(&datagram, QIODevice::WriteOnly);
1158     out.setVersion(QDataStream::Qt_4_3);
1159     out << (float)Position[0] << (float)Position[1] << (float)Position[2];
1160     out << (float)Euler[0] << (float)Euler[1] << (float)Euler[2]; //! Roll Pitch Heading
1161     udpSocketObject->writeDatagram(datagram,
1162     IPAddressObject, portObject);
1163
1164     //! if second receiver is activated send it also the tracking data
1165     if (safety2Enable)
1166     {
1167         udpSocketSafety2->writeDatagram(datagram,
1168         IPAddressSafety2, portSafety2);
1169     }
```

Here is the caller graph for this function:



6.5.2.16 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 1122 of file main.cpp.

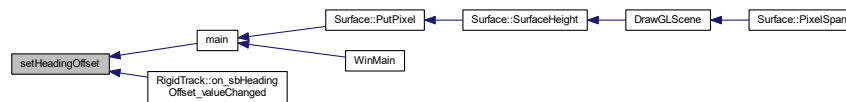
```
1123 {
1124     headingOffset = d;
1125     d = d * 3.141592653589 / 180.0; //! Convert heading offset from degrees to rad
1126 }
```

```

1127     //!< Calculate rotation about x axis
1128     Mat R_x = (Mat_<double>(3, 3) <<
1129         1, 0, 0,
1130         0, 1, 0,
1131         0, 0, 1
1132     );
1133
1134     //!< Calculate rotation about y axis
1135     Mat R_y = (Mat_<double>(3, 3) <<
1136         1, 0, 0,
1137         0, 1, 0,
1138         0, 0, 1
1139     );
1140
1141     //!< Calculate rotation about z axis
1142     Mat R_z = (Mat_<double>(3, 3) <<
1143         cos(d), -sin(d), 0,
1144         sin(d), cos(d), 0,
1145         0, 0, 1);
1146
1147
1148     //!< Combined rotation matrix
1149     M_HeadingOffset = R_z * R_y * R_x;
1150 }

```

Here is the caller graph for this function:



6.5.2.17 setReference()

```
int setReference ( )
```

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

Definition at line 595 of file main.cpp.

```

596 {
597     //!< initialize the variables with starting values
598     gotOrder = false;
599     posRef = 0;
600     eulerRef = 0;
601     RmatRef = 0;
602     Rvec = RvecOriginal;
603     Tvec = TvecOriginal;
604
605     determineExposure();
606
607     ss.str("");
608     commObj.addLog("Started reference coordinate determination.");
609
610     CameraLibrary_EnableDevelopment();
611     //!< Initialize Camera SDK ---
612     CameraLibrary::CameraManager::X();
613
614     //!< At this point the Camera SDK is actively looking for all connected cameras and will initialize
615     //!< them on it's own.
616
617     //!< Get a connected camera =====
618     CameraManager::X().WaitForInitialization();
619     Camera *camera = CameraManager::X().GetCamera();

```

```

620
621     ///! If no device connected, pop a message box and exit ==--
622     if (camera == 0)
623     {
624         commObj.addLog("No camera found!");
625         return 1;
626     }
627
628     ///! Determine camera resolution to size application window ==----
629     int cameraWidth = camera->Width();
630     int cameraHeight = camera->Height();
631     camera->GetDistortionModel(distModel);
632     cv::Mat matFrame(cv::Size(cameraWidth, cameraHeight), CV_8UC1);
633
634     ///! Set camera mode to precision mode, it directly provides marker coordinates
635     camera->SetVideoType(Core::PrecisionMode);
636
637     ///! Start camera output ===
638     camera->Start();
639
640     ///! Turn on some overlay text so it's clear things are =====
641     ///! working even if there is nothing in the camera's view. =====
642     ///! Set some other parameters as well of the camera
643     camera->SetTextOverlay(true);
644     camera->SetFrameRate(intFrameRate);
645     camera->SetIntensity(intIntensity);
646     camera->SetIRFilter(true);
647     camera->SetContinuousIR(false);
648     camera->SetHighPowerMode(false);
649
650     ///! sample some frames and calculate the position and attitude. then average those values and use that
651     as zero position
652     int numberSamples = 0;
653     int numberToSample = 200;
654     double projectionError = 0; ///!< difference between the marker points as seen by the camera and the
655     projected marker points with Rvec and Tvec
656
657     while (numberSamples < numberToSample)
658     {
659         ///! Fetch a new frame from the camera ==----
660         Frame *frame = camera->GetFrame();
661
662         if (frame)
663         {
664             ///! Ok, we've received a new frame, lets do something
665             ///! with it.
666             if (frame->ObjectCount() == numberMarkers)
667             {
668                 for(int i=0; i<frame->ObjectCount(); i++)
669                 for (int i = 0; i < numberMarkers; i++)
670                 {
671                     cObject *obj = frame->Object(i);
672                     list_points2dUnsorted[i] = cv::Point2d(obj->X(), obj->Y());
673                 }
674
675                 if (gotOrder == false)
676                 {
677                     determineOrder();
678                 }
679
680                 ///! sort the 2d points with the correct indices as found in the preceeding order
681                 determination algorithm
682                 for (int w = 0; w < numberMarkers; w++)
683                 {
684                     list_points2d[w] = list_points2dUnsorted[
685                     pointOrderIndices[w]];
686                 }
687                 list_points2dOld = list_points2dUnsorted;
688
689                 ///!Compute the pose from the 3D-2D correspondences
690                 solvePnP(list_points3d, list_points2d,
691                 cameraMatrix, distCoeffs, Rvec, Tvec, useGuess,
692                 methodPNP);
693
694                 ///! project the marker 3d points with the solution into the camera image CoSy and calculate
695                 difference to true camera image
696                 projectPoints(list_points3d, Rvec, Tvec,
697                 cameraMatrix, distCoeffs, list_points2dProjected);
698                 projectionError = norm(list_points2dProjected,
699                 list_points2d);
700
701                 double maxValue = 0;
702                 double minValue = 0;
703                 minMaxLoc(Tvec.at<double>(2), &minValue, &maxValue);
704
705                 if (maxValue > 10000 || minValue < 0)
706                 {

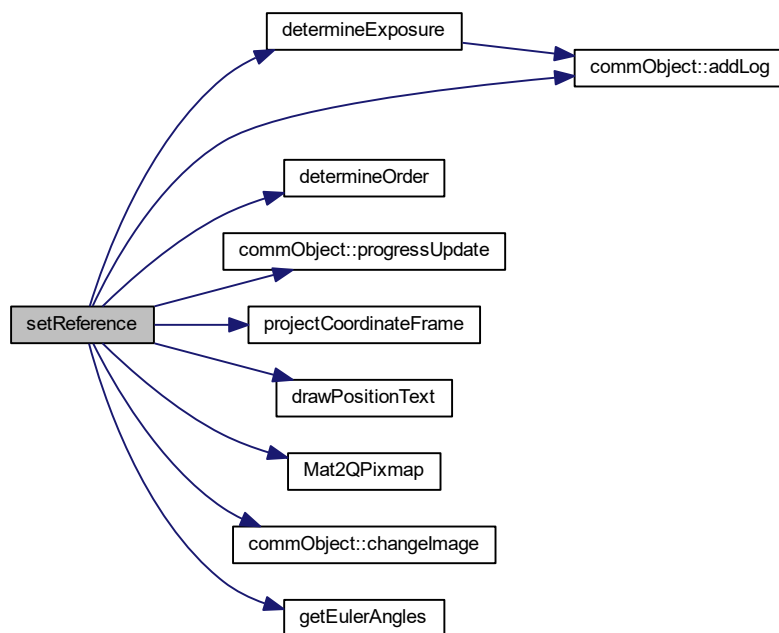
```

```

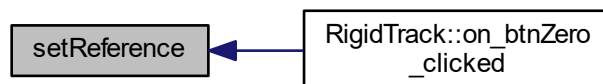
698         ss.str("");
699         ss << "Negative z distance, thats not possible. Start the set zero routine again or
restart Programm.";
700         commObj.addLog(QString::fromStdString(ss.str()));
701         frame->Release();
702         return 1;
703     }
704
705     if (projectionError > 3)
706     {
707         commObj.addLog("Reprojection error is bigger than 3 pixel. Correct marker
configuration loaded?\nMarker position measured precisely?");
708         frame->Release();
709         return 1;
710     }
711
712     if (norm(positionOld) - norm(Tvec) < 0.05)    //!< Iterative Method needs time
to converge to solution
713     {
714         add(posRef, Tvec, posRef);
715         add(eulerRef, Rvec, eulerRef); //!< That are not the values of yaw,
roll and pitch yet! Rodriguez has to be called first.
716         numberSamples++;    //!< one sample more :D
717         commObj.progressUpdate(numberSamples * 100 / numberToSample);
718     }
719     positionOld = Tvec;
720
721     Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
722     for (int i = 0; i < numberMarkers; i++)
723     {
724         circle(cFrame, Point(list_points2d[i].x,
list_points2d[i].y), 6, Scalar(0, 225, 0), 3);
725     }
726     projectCoordinateFrame(cFrame);
727     projectPoints(list_points3d, Rvec, Tvec,
cameraMatrix, distCoeffs, list_points2d);
728     for (int i = 0; i < numberMarkers; i++)
729     {
730         circle(cFrame, Point(list_points2d[i].x,
list_points2d[i].y), 3, Scalar(225, 0, 0), 3);
731     }
732     drawPositionText(cFrame, position,
eulerAngles, projectionError);
733
734     QPixmap QPFrame;
735     QPFrame = Mat2QPixmap(cFrame);
736     commObj.changeImage(QPFrame);
737     QApplication::processEvents();
738
739     }
740     frame->Release();
741 }
742 }
743
744 //!< Release camera ===
camera->Release();
745
746 //!< Divide by the number of samples to get the mean of the reference position
747 divide(posRef, numberToSample, posRef);
748 divide(eulerRef, numberToSample, eulerRef); //!< eulerRef is here in Axis Angle
notation
749
750 Rodrigues(eulerRef, RmatRef);    //!< axis angle to rotation matrix
751 //!<-- Euler Angles, finally
752 getEulerAngles(RmatRef, eulerRef); //!< rotation matrix to euler
753 ss.str("");
754 ss << "RmatRef is:\n";
755 ss << RmatRef << "\n";
756 ss << "Reference Position is:\n";
757 ss << posRef << "[mm] \n";
758 ss << "Reference Euler Angles are:\n";
759 ss << eulerRef << "[deg] \n";
760
761 //!< compute the difference between last obtained Tvec and the average Value
762 //!< When it is large the iterative method has not converged properly so it is advised to start the
setReference() function once again
763 double error = norm(posRef) - norm(Tvec);
764 if (error > 5.0)
765 {
766     ss << "Caution, distance between reference position and last position is: " << error << "\n Start
the set zero routine once again.";
767 }
768 commObj.addLog(QString::fromStdString(ss.str()));
769 commObj.progressUpdate(0);
770 return 0;
771 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.2.18 setUpUDP()

```
void setUpUDP ( )
```

Open the UDP ports for communication.

Definition at line 1090 of file main.cpp.

```

1091 {
1092     //! Initialise the QDataStream that stores the data to be send
1093     QDataStream out(&datagram, QIODevice::WriteOnly);
1094     out.setVersion(QDataStream::Qt_4_3);
1095
1096     //! Create UDP slots
1097     commObj.addLog("Opening UDP ports.");
1098     udpSocketObject = new QUdpSocket(0);
1099     udpSocketObject->connectToHost(IPAddressObject,
portObject);
1100     commObj.addLog("Opened first receiver UDP port.");
1101
1102     udpSocketSafety = new QUdpSocket(0);
1103     udpSocketSafety2 = new QUdpSocket(0);
1104
1105     //! if the safety feature is activated open the udp port
1106     if (safetyEnable)
1107     {
1108         udpSocketSafety->connectToHost(IPAdressSafety,
portSafety);
1109         commObj.addLog("Opened safety UDP port.");
1110     }
1111
1112     //! if the second receiver feature is activated open the udp port
1113     if (safety2Enable)
1114     {
1115         udpSocketSafety2->connectToHost(IPAdressSafety2,
portSafety2);
1116         commObj.addLog("Opened second receiver UDP port.");
1117     }
1118 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.2.19 startStopCamera()

```
void startStopCamera ( )
```

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

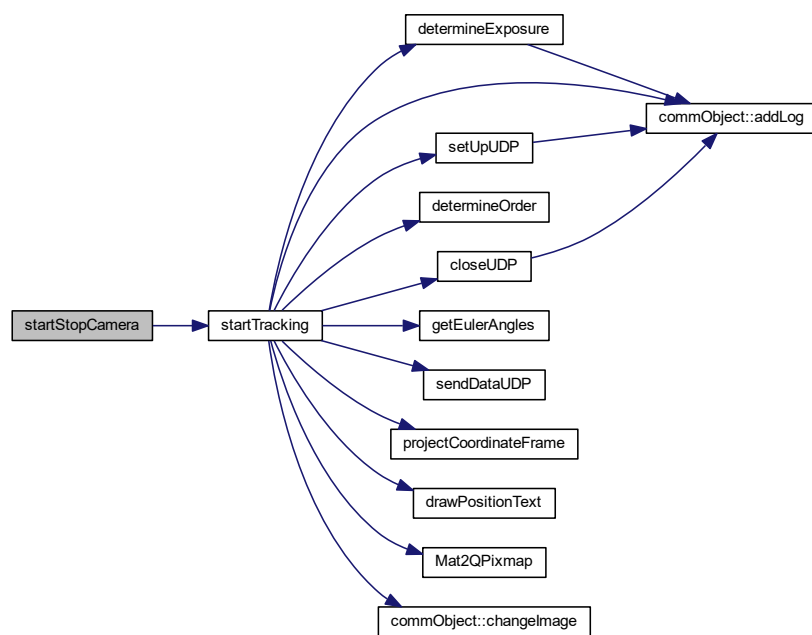
Definition at line 579 of file main.cpp.


```

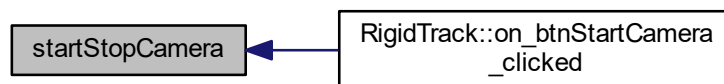
580 {
581     ///! tracking is not running so start it
582     if (exitRequested)
583     {
584         exitRequested = false;
585         startTracking();
586     }
587     else ///!< tracking is currently running, set exitRequest to true so the while loop in startTracking()
588     exits
589     {
590         exitRequested = true;
591     }
592 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.2.20 startTracking()

```
int startTracking ( )
```

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

Definition at line 261 of file main.cpp.

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

```

323     /// Variables for the min and max values that are needed for sanity checks
324     double maxValue = 0;
325     double minValue = 0;
326     int framesDropped = 0; /// If a marker is not visible or accuracy is bad increase this counter
327     double projectionError = 0; /// Equals the quality of the tracking
328
329     setUpUDP(); /// Open sockets and ports for UDP communication
330
331     if (safetyEnable) /// If the safety feature is enabled send the starting message
332     {
333         /// Send enable message, hence send a 9 and then a 1
334         data.setNum((int)(9));
335         udpSocketSafety->write(data);
336         data.setNum((int)(1));
337         udpSocketSafety->write(data);
338     }
339
340     /// Fetch a new frame from the camera
341     bool gotTime = false; /// Get the timestamp of the first frame. This time is subtracted from every
subsequent frame so the time starts at 0 in the logs
342     while (!gotTime) /// While no new frame is received loop
343     {
344         Frame *frame = camera->GetFrame(); /// Get a new camera frame
345         if (frame) /// There is actually a new frame
346         {
347             timeFirstFrame = frame->TimeStamp(); /// Get the time stamp for the first frame.
It is subtracted for the following frames
348             frame->Release(); /// Release the frame so the camera can continue
349             gotTime = true; /// Exit the while loop
350         }
351     }
352
353     /// Now enter the main loop that processes each frame and computes the pose, sends it and logs stuff
354     while (!exitRequested) /// Check if the user has not pressed "Stop Tracking" yet
355     {
356         Frame *frame = camera->GetFrame(); /// Fetch a new frame from the camera
357
358         if (frame) /// Did we got a new frame or does the camera still need more time
359         {
360             framesDropped++; /// Increase by one, if everything is okay it is decreased at the end of the
loop again
361
362             /// Only use this frame if the right number of markers is found in the picture
363             if (frame->ObjectCount() == numberMarkers)
364             {
365                 /// Get the marker points in 2D in the camera image frame and store them in the
list_points2dUnsorted vector
366                 /// The order of points that come from the camera corresponds to the Y coordinate
367                 for (int i = 0; i < numberMarkers; i++)
368                 {
369                     cObject *obj = frame->Object(i);
370                     list_points2dUnsorted[i] = cv::Point2d(obj->X(), obj->Y());
371                 }
372
373                 if (gotOrder == false) /// Was the order already determined? This is false for the
first frame and from then on true
374                 {
375                     determineOrder(); /// Now compute the order
376                 }
377
378                 /// Sort the 2d points with the correct indices as found in the preceeding order
determination algorithm
379                 for (int w = 0; w < numberMarkers; w++)
380                 {
381                     list_points2d[w] = list_points2dUnsorted[
pointOrderIndices[w]]; /// pointOrderIndices was calculated in determineOrder()
382                 }
383                 list_points2dOld = list_points2dUnsorted;
384
385                 /// The first time the 2D-3D corresspondence was determined with gotOrder was okay.
386                 /// But this order can change as the object moves and the marker objects appear in a
387                 /// different order in the frame->Object() array.
388                 /// The solution is that: When a marker point (in the camera image, hence in 2D) was at
389                 /// a position then it wont move that much from one frame to the other.
390                 /// So for the new frame we take a marker object and check which marker was closest this
point
391                 /// in the old image frame? This is probably the same (true) marker. And we do that for
every other marker as well.
392                 /// When tracking is good and no frames are dropped because of missing markers this should
work every frame.
393                 for (int j = 0; j < numberMarkers; j++)
394                 {
395                     minPointDistance = 5000; /// The sum of point distances is set to
something unrealistic large
396                     for (int k = 0; k < numberMarkers; k++)
397                     {
398

```

```

399             ///! Calculate N_2 norm of unsorted points minus old points
400             currentPointDistance = norm(
list_points2dUnsorted[pointOrderIndices[j]] -
list_points2dOld[k]);
401             ///! If the norm is smaller than minPointDistance the correspondence is more likely
to be correct
402             if (currentPointDistance <
minPointDistance)
403             {
404                 ///! Update the array that saves the new point order
405                 minPointDistance =
currentPointDistance;
406                 pointOrderIndicesNew[j] = k;
407             }
408         }
409     }
410
411     ///! Now the new order is found, set the point order to the new value
412     for (int k = 0; k < numberMarkers; k++)
413     {
414         pointOrderIndices[k] = pointOrderIndicesNew[k];
415         list_points2d[k] = list_points2dUnsorted[
pointOrderIndices[k]];
416     }
417
418     ///! Save the unsorted position of the marker points for the next loop
419     list_points2dOld = list_points2dUnsorted;
420
421     ///! Compute the object pose from the 3D-2D correspondences
422     solvePnP(list_points3d, list_points2d,
cameraMatrix, distCoeffs, Rvec, Tvec, useGuess,
methodPNP);
423
424     ///! Project the marker 3d points with the solution into the camera image CoSy and calculate
difference to true camera image
425     projectPoints(list_points3d, Rvec, Tvec,
cameraMatrix, distCoeffs, list_points2dProjected);
426     projectionError = norm(list_points2dProjected,
list_points2d); ///! Difference of true pose and found pose
427
428     ///! Increase the framesDropped variable if accuracy of tracking is too bad
429     if (projectionError > 5)
430     {
431         framesDropped++;
432     }
433     else
434     {
435         framesDropped = 0; ///! Set number of subsequent frames dropped to zero because error
is small enough and no marker was missing
436     }
437
438     ///! Get the min and max values from Tvec for sanity check
439     minMaxLoc(Tvec.at<double>(2), &minValue, &maxValue);
440
441     ///! Sanity check of values. negative z means the marker CoSy is behind the camera, that's
not possible.
442     if (minValue < 0)
443     {
444         commObj.addLog("Negative z distance, that is not possible. Start the set
zero routine again or restart Program.");
445         frame->Release(); ///! Release the frame so the camera can move on
446         camera->Release(); ///! Release the camera
447         closeUDP(); ///! Close all UDP connections so the programm can be closed later
on and no resources are locked
448         return 1; ///! Exit the function
449     }
450
451     ///! Next step is the transformation from camera CoSy to navigation CoSy
452     ///! Compute the relative object position from the reference position to the current one
453     ///! given in the camera CoSy: \f$ T_C^{NM} = Tvec - Tvec_{Ref} \f$
454     subtract(Tvec, posRef, position);
455
456     ///! Transform the position from the camera CoSy to the navigation CoSy with INS aligned
heading and convert from [mm] to [m]
457     ///! \f$ T_N^{NM} = M_{NC} \times T_C^{NM} \f$
458     Mat V = 0.001 * M_HeadingOffset * M_CN.t() * (Mat)
position;
459     position = V; ///! Position is the result of the preceeding calculation
460     position[2] *= invertZ; ///! Invert Z if check box in GUI is activated,
hence height above ground is considered
461
462     ///! Realtime angle between reference orientation and current orientation
463     Rodrigues(Rvec, Rmat); ///! Convert axis angle representation to ordinary rotation
matrix
464
465     ///! The difference of the reference rotation and the current rotation
466     ///! \f$ R_{NM} = M_{NC} \times R_{CM} \f$

```

```

467         Rmat = RmatRef.t() * Rmat;
468
469         //! Euler Angles, finally
470         getEulerAngles(Rmat, eulerAngles); //! Get the euler angles
from the rotation matrix
471         eulerAngles[2] += headingOffset; //! Add the heading offset to the
heading angle
472
473         //! Compute the velocity with finite differences. Only use is the log file. It is done here
because the more precise time stamp can be used
474         frameTime = frame->TimeStamp() - timeOld; //! Time between the old frame
and the current frame
475         timeOld = frame->TimeStamp(); //! Set the old frame time to the current one
476         velocity[0] = (position[0] - positionOld[0]) /
frameTime; //! Calculate the x velocity with finite differences
477         velocity[1] = (position[1] - positionOld[1]) /
frameTime; //! Calculate the y velocity with finite differences
478         velocity[2] = (position[2] - positionOld[2]) /
frameTime; //! Calculate the z velocity with finite differences
479         positionOld = position; //! Set the old position to the current one for
next frame velocity calculation
480
481         //! Send position and Euler angles over WiFi with 100 Hz
482         sendDataUDP(position, eulerAngles);
483
484         //! Save the values in a log file, values are:
485         //! Time sinc tracking started Position Euler Angles Velocity
486         logfile.open(logName, std::ios::app); //! Open the log file, the folder is
RigidTrackInstallationFolder/logs
487         logfile << frame->TimeStamp() - timeFirstFrame << ";" <<
position[0] << ";" << position[1] << ";" << position[2] << ";";
488         logfile << eulerAngles[0] << ";" <<
eulerAngles[1] << ";" << eulerAngles[2] << ";";
489         logfile << velocity[0] << ";" << velocity[1] << ";" <<
velocity[2] << "\n";
490         logfile.close(); //! Close the file to save values
491     }
492
493     //! Check if the position and euler angles are below the allowed value, if yes send OKAY signal
(1), if not send shutdown signal (0)
494     //! Absolute x, y and z position in navigation CoSy must be smaller than the allowed distance
495     if (safetyEnable)
496     {
497         if ((abs(position[0]) < safetyBoxLength && abs(position[1]) <
safetyBoxLength && abs(position[2]) < safetyBoxLength))
498         {
499             //! Absolute Euler angles must be smaller than allowed value. Heading is not considered
500             if ((abs(eulerAngles[0]) < safetyAngle && abs(eulerAngles[1]) <
safetyAngle))
501             {
502                 //! Send the OKAY signal to the desired computer every 5th time
503                 if (v == 5) {
504                     data.setNum((int)(1));
505                     udpSocketSafety->write(data); //! Send the 1
506                     v = 0; //! reset the counter that is needed for decimation to every 5th time
step
507                 }
508             }
509             //! The euler angles of the object exceeded the allowed euler angles, send the shutdown
signal (0)
510             else
511             {
512                 data.setNum((int)(0)); //! Send the shutdown signal, a 0
513                 udpSocketSafety->write(data);
514                 commObj.addLog("Object exceeded allowed Euler angles, shutdown signal
sent."); //! Inform the user
515             }
516         }
517     }
518     //! The position of the object exceeded the allowed position, shut the object down
519     else
520     {
521         data.setNum((int)(0)); //! Send the shutdown signal, a 0
522         udpSocketSafety->write(data);
523         commObj.addLog("Object left allowed area, shutdown signal sent."); //!
Inform the user
524     }
525 }
526
527 //! Inform the user if tracking system is disturbed (marker lost or so) or error was too big
528 if (framesDropped > 10)
529 {
530     if (safetyEnable) //! Also send the shutdown signal
531     {
532         data.setNum((int)(0)); //! Send the shutdown signal, a 0
533     }

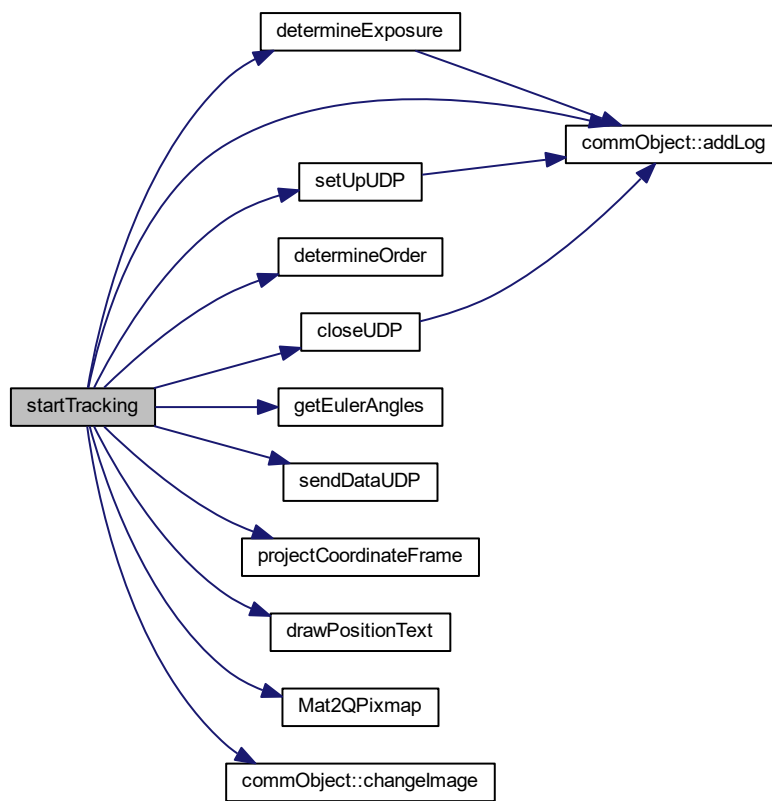
```

```

534         udpSocketSafety->write(data);
535     }
536     commObj.addLog("Lost marker points or precision was bad!"); //! Inform the
    user
537     framesDropped = 0;
538 }
539
540     //! Rasterize the frame so it can be shown in the GUI
541     frame->Rasterize(cameraWidth, cameraHeight, matFrame.step,
    BACKBUFFER_BITSPERPIXEL, matFrame.data);
542
543     //! Convert the frame from greyscale as it comes from the camera to rgb color
544     cvtColor(matFrame, cFrame, COLOR_GRAY2RGB);
545
546     //! Project (draw) the marker CoSy origin into 2D and save it in the cFrame image
547     projectCoordinateFrame(cFrame);
548
549     //! Project the marker points from 3D to the camera image frame (2d) with the computed pose
550     projectPoints(list_points3d, Rvec, Tvec,
    cameraMatrix, distCoeffs, list_points2d);
551     for (int i = 0; i < numberMarkers; i++)
552     {
553         //! Draw a circle around the projected points so the result can be better compared to the
    real marker position
554         //! In the resulting picture those are the red dots
555         circle(cFrame, Point(list_points2d[i].x,
    list_points2d[i].y), 3, Scalar(225, 0, 0), 3);
556     }
557
558     //! Write the current position, attitude and error values as text in the frame
559     drawPositionText(cFrame, position, eulerAngles, projectionError);
560
561     //! Send the new camera picture to the GUI and call the GUI processing routine
562     QPixmap QPFrame;
563     QPFrame = Mat2QPixmap(cFrame);
564     commObj.changeImage(QPFrame); //! Update the picture in the GUI
565     QApplication::processEvents(); //! Give Qt time to handle everything
566
567     //! Release the camera frame to fetch the new one
568     frame->Release();
569 }
570 }
571
572 //! User choose to stop the tracking, clean things up
573 closeUDP(); //! Close the UDP connections so resources are deallocated
574 camera->Release(); //! Release camera
575 return 0;
576 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.2.21 testAlgorithms()

```
void testAlgorithms ( )
```

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

Definition at line 952 of file main.cpp.

```

953 {
954
955     int _methodPNP;
956
957     std::vector<Point2d> noise(numberMarkers);
958
959     RvecOriginal = Rvec;
960     TvecOriginal = Tvec;
961
962     projectPoints(list_points3d, Rvec, Tvec, cameraMatrix,
distCoeffs, list_points2dProjected);
963
964     ss.str("");
965     ss << "Unsorted Points 2D Projected \n";
966     ss << list_points2dProjected << "\n";
967     commObj.addLog(QString::fromStdString(ss.str()));
968
969     Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
970     for (int i = 0; i < numberMarkers; i++)
971     {
972         circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 6, Scalar(0, 255, 0
), 3);
973     }
974
975     projectCoordinateFrame(cFrame);
976
977     ss.str("");
978     ss << "=====\n";
979     ss << "===== Projected Points =====\n";
980     ss << list_points2dProjected << "\n";
981
982     randn(noise, 0, 0.5);
983     add(list_points2dProjected, noise, list_points2dProjected);
984
985     ss << "===== With Noise Points =====\n";
986     ss << list_points2dProjected << "\n";
987     commObj.addLog(QString::fromStdString(ss.str()));
988
989     bool useGuess = true;
990     _methodPNP = 0; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP //!< not used
991
992     solvePnP(list_points3d, list_points2dProjected, cameraMatrix,
distCoeffs, Rvec, Tvec, useGuess, _methodPNP);
993
994     ss.str("");
995     ss << "=====\n";
996     ss << "===== Iterative =====\n";
997     ss << "rvec: " << "\n";
998     ss << Rvec << "\n";
999     ss << "tvec: " << "\n";
1000     ss << Tvec << "\n";
1001
1002     commObj.addLog(QString::fromStdString(ss.str()));
1003
1004     _methodPNP = 1; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP UPnP not used
1005     Rvec = cv::Mat::zeros(3, 1, CV_64F);
1006     Tvec = cv::Mat::zeros(3, 1, CV_64F);
1007     solvePnP(list_points3d, list_points2dProjected, cameraMatrix,
distCoeffs, Rvec, Tvec, useGuess, _methodPNP);
1008
1009     ss.str("");
1010     ss << "=====\n";
1011     ss << "===== EPNP =====\n";
1012     ss << "rvec: " << "\n";
1013     ss << Rvec << "\n";
1014     ss << "tvec: " << "\n";
1015     ss << Tvec << "\n";
1016
1017     projectPoints(list_points3d, Rvec, Tvec, cameraMatrix,
distCoeffs, list_points2dProjected);
1018     for (int i = 0; i < numberMarkers; i++)
1019     {
1020         circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 3, Scalar(255, 0, 0
), 3);
1021     }
1022     QPixmap QPFrame;
1023     QPFrame = Mat2QPixmap(cFrame);
1024     commObj.changeImage(QPFrame);
1025     QCoreApplication::processEvents();
1026     commObj.addLog(QString::fromStdString(ss.str()));
1027     if (numberMarkers == 4)
1028     {
1029         _methodPNP = 2; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP //!< not used
1030         Rvec = cv::Mat::zeros(3, 1, CV_64F);
1031         Tvec = cv::Mat::zeros(3, 1, CV_64F);
1032         solvePnP(list_points3d, list_points2dProjected,

```

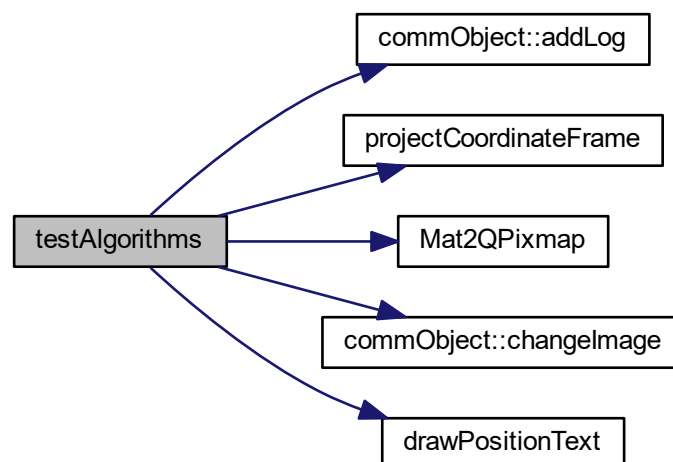


```

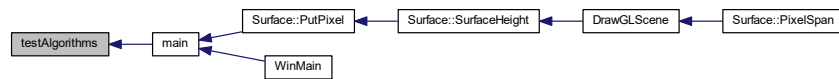
cameraMatrix, distCoeffs, Rvec, Tvec, useGuess, _methodPNP);
1034
1035     ss.str("");
1036     ss << "=====\n";
1037     ss << "===== P3P =====\n";
1038     ss << "rvec: " << "\n";
1039     ss << Rvec << "\n";
1040     ss << "tvec: " << "\n";
1041     ss << Tvec << "\n";
1042
1043     projectPoints(list_points3d,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

```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.3 Variable Documentation

6.5.3.1 BACKBUFFER_BITSPERPIXEL

```
const int BACKBUFFER_BITSPERPIXEL = 8
```

8 bit per pixel and greyscale image from camera

Definition at line 140 of file main.cpp.

6.5.3.2 camera_started

```
bool camera_started = false
```

variable thats needed to exit the main while loop

Definition at line 122 of file main.cpp.

6.5.3.3 cameraMatrix

```
Mat cameraMatrix
```

camera matrix of the camera

Definition at line 124 of file main.cpp.

6.5.3.4 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.

6.5.3.5 coordinateFrame

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

coordinate visualisazion of marker CoSy

Definition at line 113 of file main.cpp.

6.5.3.6 coordinateFrameProjected

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

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

Definition at line 114 of file main.cpp.

6.5.3.7 currentMinIndex

```
int currentMinIndex = 0
```

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

Definition at line 119 of file main.cpp.

6.5.3.8 currentPointDistance

```
double currentPointDistance = 5000
```

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

Definition at line 117 of file main.cpp.

6.5.3.9 data

```
QByteArray data
```

data package that's sent to the safety receiver

Definition at line 138 of file main.cpp.

6.5.3.10 datagram

```
QByteArray datagram
```

data package that is sent to receiver 1 and 2

Definition at line 137 of file main.cpp.

6.5.3.11 distCoeffs

```
Mat distCoeffs
```

distortion coefficients of the camera

Definition at line 125 of file main.cpp.

6.5.3.12 distModel

```
Core::DistortionModel distModel
```

distortion model of the camera

Definition at line 126 of file main.cpp.

6.5.3.13 eulerAngles

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

Roll Pitch Heading in this order, units in degrees.

Definition at line 82 of file main.cpp.

6.5.3.14 eulerRef

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

initial euler angle of object respectivley to camera CoSy

Definition at line 86 of file main.cpp.

6.5.3.15 exitRequested

```
bool exitRequested = true
```

variable if tracking loop should be exited

Definition at line 74 of file main.cpp.

6.5.3.16 frameTime

```
double frameTime = 0.01
```

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

Definition at line 77 of file main.cpp.

6.5.3.17 gotOrder

```
bool gotOrder = false
```

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

Definition at line 120 of file main.cpp.

6.5.3.18 headingOffset

```
double headingOffset = 0
```

heading offset variable for aligning INS heading with tracking heading

Definition at line 87 of file main.cpp.

6.5.3.19 intExposure

```
int intExposure = 1
```

max is 480 increase if markers are badly visible but should be determined automatically during [setReference\(\)](#)

Definition at line 90 of file main.cpp.

6.5.3.20 intFrameRate

```
int intFrameRate = 100
```

CoSy rate of camera, maximum is 100 fps.

Definition at line 91 of file main.cpp.

6.5.3.21 intIntensity

```
int intIntensity = 15
```

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

Definition at line 89 of file main.cpp.

6.5.3.22 intThreshold

```
int intThreshold = 200
```

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

Definition at line 92 of file main.cpp.

6.5.3.23 invertZ

```
int invertZ = 1
```

dummy variable to invert Z direction on request

Definition at line 75 of file main.cpp.

6.5.3.24 IPAdressObject

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

IPv4 adress of receiver 1.

Definition at line 131 of file main.cpp.

6.5.3.25 IPAdressSafety

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

IPv4 adress of safety receiver.

Definition at line 132 of file main.cpp.

6.5.3.26 IPAdressSafety2

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

IPv4 adress of receiver 2.

Definition at line 133 of file main.cpp.

6.5.3.27 list_points2d

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

marker positions projected in 2D in camera image CoSy

Definition at line 108 of file main.cpp.

6.5.3.28 list_points2dDifference

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

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

Definition at line 110 of file main.cpp.

6.5.3.29 list_points2dOld

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

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

Definition at line 109 of file main.cpp.

6.5.3.30 list_points2dProjected

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

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

Definition at line 111 of file main.cpp.

6.5.3.31 list_points2dUnsorted

```
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

Definition at line 112 of file main.cpp.

6.5.3.32 list_points3d

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

marker positions in marker CoSy

Definition at line 107 of file main.cpp.

6.5.3.33 logDate

```
SYSTEMTIME logDate
```

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

Definition at line 145 of file main.cpp.

6.5.3.34 logfile

```
std::ofstream logfile
```

file handler for writing the log file

Definition at line 146 of file main.cpp.

6.5.3.35 logFileName

```
QString logFileName
```

Filename for the logfiles.

Definition at line 143 of file main.cpp.

6.5.3.36 logName

```
std::string logName
```

Filename for the logfiles as standard string.

Definition at line 144 of file main.cpp.

6.5.3.37 M_CN

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

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

Definition at line 97 of file main.cpp.

6.5.3.38 M_HeadingOffset

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

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

Definition at line 98 of file main.cpp.

6.5.3.39 methodPNP

```
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

Definition at line 105 of file main.cpp.

6.5.3.40 minPointDistance

```
double minPointDistance = 5000
```

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

Definition at line 118 of file main.cpp.

6.5.3.41 numberMarkers

```
int numberMarkers = 4
```

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

Definition at line 106 of file main.cpp.

6.5.3.42 pointOrderIndices

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

old correspondence from list_points3d and list_points_2d

Definition at line 115 of file main.cpp.

6.5.3.43 pointOrderIndicesNew

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

new correspondence from list_points3d and list_points_2d

Definition at line 116 of file main.cpp.

6.5.3.44 portObject

```
int portObject = 9155
```

Port of receiver 1.

Definition at line 134 of file main.cpp.

6.5.3.45 portSafety

```
int portSafety = 9155
```

Port of the safety receiver.

Definition at line 135 of file main.cpp.

6.5.3.46 portSafety2

```
int portSafety2 = 9155
```

Port of receiver 2.

Definition at line 136 of file main.cpp.

6.5.3.47 position

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

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

Definition at line 81 of file main.cpp.

6.5.3.48 positionOld

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

old position in O-CoSy for finite differences velocity calculation

Definition at line 83 of file main.cpp.

6.5.3.49 posRef

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

initial position of object in camera CoSy

Definition at line 85 of file main.cpp.

6.5.3.50 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.

6.5.3.51 RmatRef

```
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

Definition at line 96 of file main.cpp.

6.5.3.52 Rvec

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

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

Definition at line 99 of file main.cpp.

6.5.3.53 RvecOriginal

```
Mat RvecOriginal
```

initial values as start values for algorithms and algorithm tests

Definition at line 101 of file main.cpp.

6.5.3.54 safety2Enable

```
bool safety2Enable = false
```

is the second receiver enabled

Definition at line 71 of file main.cpp.

6.5.3.55 safetyAngle

```
int safetyAngle = 30
```

bank and pitch angle protection in degrees

Definition at line 73 of file main.cpp.

6.5.3.56 safetyBoxLength

```
double safetyBoxLength = 1.5
```

length of the safety area cube in meters

Definition at line 72 of file main.cpp.

6.5.3.57 safetyEnable

```
bool safetyEnable = false
```

is the safety feature enabled

Definition at line 70 of file main.cpp.

6.5.3.58 ss

```
std::stringstream ss
```

stream that sends the strBuf buffer to the Qt GUI

Definition at line 142 of file main.cpp.

6.5.3.59 strBuf

```
std::string strBuf
```

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

Definition at line 141 of file main.cpp.

6.5.3.60 timeFirstFrame

```
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.

Definition at line 79 of file main.cpp.

6.5.3.61 timeOld

```
double timeOld = 0.0
```

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

Definition at line 78 of file main.cpp.

6.5.3.62 Tvec

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

translation vector from camera CoSy to marker CoSy in camera CoSy

Definition at line 100 of file main.cpp.

6.5.3.63 TvecOriginal

```
Mat TvecOriginal
```

initial values as start values for algorithms and algorithm tests

Definition at line 102 of file main.cpp.

6.5.3.64 udpSocketObject

```
QUdpSocket* udpSocketObject
```

socket for the communication with receiver 1

Definition at line 128 of file main.cpp.

6.5.3.65 udpSocketSafety

```
QUdpSocket* udpSocketSafety
```

socket for the communication with safety receiver

Definition at line 129 of file main.cpp.

6.5.3.66 udpSocketSafety2

```
QUdpSocket* udpSocketSafety2
```

socket for the communication with receiver 3

Definition at line 130 of file main.cpp.

6.5.3.67 useGuess

```
bool useGuess = true
```

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

Definition at line 104 of file main.cpp.

6.5.3.68 velocity

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

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

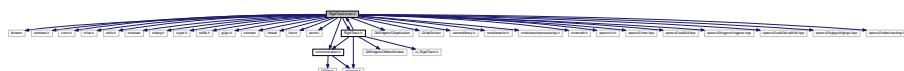
Definition at line 84 of file main.cpp.

6.6 RigidTrack/main.h File Reference

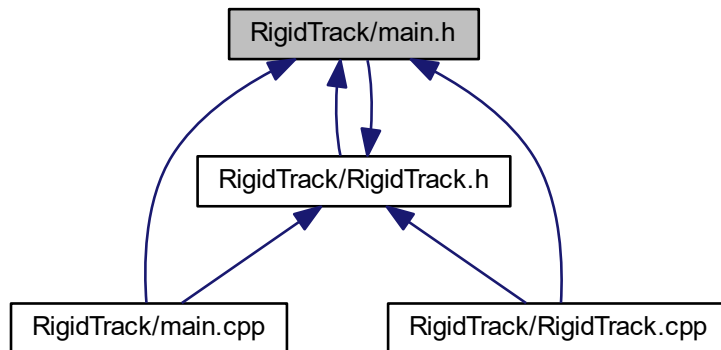
Header file for [main.cpp](#).

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

Include dependency graph for main.h:



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



Functions

- int [startTracking](#) ()
- void [startStopCamera](#) ()
 - Start or stop the tracking depending on if the camera is currently running or not.*
- int [setReference](#) ()
- int [calibrateCamera](#) ()
 - Start the camera calibration routine that computes the camera matrix and distortion coefficients.*
- void [loadCalibration](#) (int method)
- void [testAlgorithms](#) ()
- void [projectCoordinateFrame](#) (Mat pictureFrame)
- void [setUpUDP](#) ()
 - Open the UDP ports for communication.*
- void [setHeadingOffset](#) (double d)
- void [sendDataUDP](#) (cv::Vec3d &Position, cv::Vec3d &Euler)
- void [closeUDP](#) ()
- void [loadMarkerConfig](#) (int method)
- void [drawPositionText](#) (cv::Mat &Picture, cv::Vec3d &Position, cv::Vec3d &Euler, double error)
- void [loadCameraPosition](#) ()
- int [determineExposure](#) ()
- void [determineOrder](#) ()
- int [calibrateGround](#) ()

Variables

- int [methodPNP](#)
 - solvePNP algorithm 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP //!< 4 and 1 are the same and not implemented correctly by OpenCV*
- bool [safetyEnable](#)
 - is the safety feature enabled*
- bool [safety2Enable](#)

- is the second receiver enabled*
- double [safetyBoxLength](#)
length of the safety area cube in meters
- int [safetyAngle](#)
bank and pitch angle protection in degrees
- QHostAddress [IPAdressObject](#)
IPv4 adress of receiver 1.
- QHostAddress [IPAdressSafety](#)
IPv4 adress of safety receiver.
- QHostAddress [IPAdressSafety2](#)
IPv4 adress of receiver 2.
- int [portObject](#)
Port of receiver 1.
- int [portSafety](#)
Port of the safety receiver.
- int [portSafety2](#)
Port of receiver 2.
- int [invertZ](#)
dummy variable to invert Z direction on request
- [commObject](#) [commObj](#)
class that handles the communication from [main.cpp](#) to the GUI

6.6.1 Detailed Description

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

Author

Florian J.T. Wachter

Version

1.0

Date

April, 8th 2017

6.6.2 Function Documentation

6.6.2.1 calibrateCamera()

```
int calibrateCamera ( )
```

Start the camera calibration routine that computes the camera matrix and distortion coefficients.

Definition at line 774 of file main.cpp.

```

775 {
776     commObj.addLog("Started camera calibration. 80 pictures are going to be captured.");
777     CameraLibrary_EnableDevelopment();
778
779     /// Initialize Camera SDK ===
780     CameraLibrary::CameraManager::X();
781
782     /// At this point the Camera SDK is actively looking for all connected cameras and will initialize
783     /// them on it's own.
784
785     /// Get a connected camera =====
786     CameraManager::X().WaitForInitialization();
787
788     Camera *camera = CameraManager::X().GetCamera();
789     if (camera == 0)
790     {
791         commObj.addLog("No camera found!");
792         return 1;
793     }
794
795     /// Determine camera resolution
796     int cameraWidth = camera->Width();
797     int cameraHeight = camera->Height();
798
799     /// Set Video Mode ===
800
801     /// We set the camera to Segment Mode here. This mode is support by all of our products.
802     /// Depending on what device you have connected you might want to consider a different
803     /// video mode to achieve the best possible tracking quality. All devices that support a
804     /// mode that will achieve a better quality output with a mode other than Segment Mode are
805     /// listed here along with what mode you should use if you're looking for the best head
806     /// tracking:
807     ///
808     ///      V100:R1/R2    Precision Mode
809     ///      TrackIR 5     Bit-Packed Precision Mode
810     ///      V120          Precision Mode
811     ///      TBar          Precision Mode
812     ///      S250e         Precision Mode
813     ///
814     /// If you have questions about a new device that might be conspicuously missing here or
815     /// have any questions about head tracking, email support or participate in our forums.
816
817     camera->SetVideoType(Core::GrayscaleMode);
818
819     /// Start camera output ===
820     camera->Start();
821
822     /// Camera Matrix creation ===
823     cameraMatrix = Mat::eye(3, 3, CV_64F);
824     distCoeffs = Mat::zeros(8, 1, CV_64F);
825
826     /// Ok, start main loop. This loop fetches and displays =====
827     /// camera frames.
828     /// But first set some camera parameters
829     camera->SetAGC(false);
830     camera->SetAEC(false);
831     camera->SetExposure(200);
832     camera->SetIntensity(4);
833     camera->SetFrameRate(30);
834     camera->SetIRFilter(true);
835     camera->SetContinuousIR(false);
836     camera->SetHighPowerMode(false);
837
838     int number_samples = 0;
839     int imagesToSample = 80;
840
841     std::vector<std::vector<Point2f>> > imagePoints;
842     std::vector<Point2f> pointBuf;
843     bool found;
844     Size boardSize(9, 6);
845     Size imageSize(cameraWidth, cameraHeight);
846     Mat Rvec(3, 1, DataType<double>::type);
847     Mat Tvec(3, 1, DataType<double>::type);
848

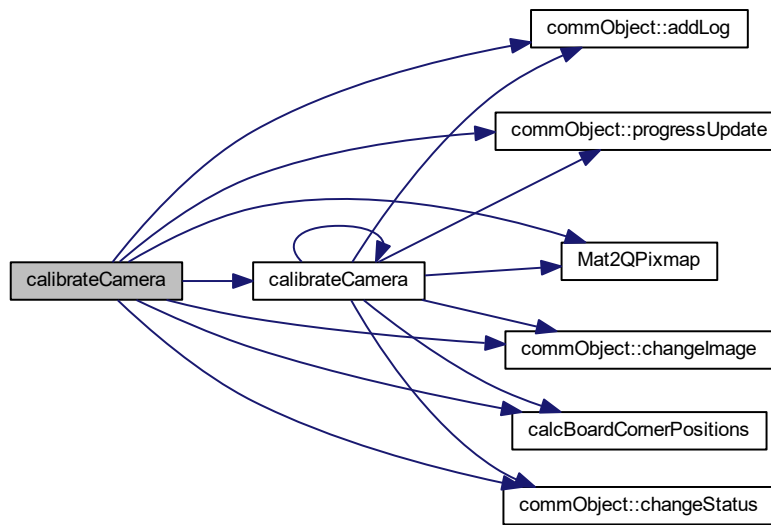
```

```

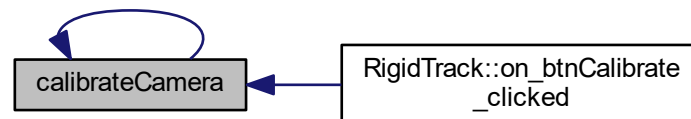
849     //! the user has to provide the size of one square in mm
850     bool ok;
851     int qsquareSize = QInputDialog::getInt(nullptr, "Chessboard size in mm", "Chessboard size in mm", 23, 1
, 60, 1, &ok);
852     float squareSize = 23;
853
854     if (ok)
855     {
856         squareSize = qsquareSize;
857     }
858
859     QPixmap QPFrame;
860     commObj.progressUpdate(0);
861     while (number_samples < imagesToSample)
862     {
863         //! Fetch a new frame from the camera =====
864         cv::Mat matFrame(cv::Size(cameraWidth, cameraHeight), CV_8UC1);
865
866         //! which is why we also set this constant to 8
867         const int BACKBUFFER_BITSPERPIXEL = 8;
868
869         //! later on, when we get the frame as usual:
870         CameraLibrary::Frame * frame = camera->GetFrame();
871
872         if (frame)
873         {
874             //! Lets have the Camera Library raster the camera's
875             //! image into our texture.
876
877             frame->Rasterize(cameraWidth, cameraHeight, matFrame.step, BACKBUFFER_BITSPERPIXEL, matFrame.
data);
878             QPFrame = Mat2QPixmap(matFrame);
879             commObj.changeImage(QPFrame);
880             found = findChessboardCorners(matFrame, boardSize, pointBuf, CV_CALIB_CB_ADAPTIVE_THRESH |
CV_CALIB_CB_FAST_CHECK | CV_CALIB_CB_NORMALIZE_IMAGE);
881
882             if (found)                //!< If done with success,
883             {
884                 //! improve the found corners' coordinate accuracy for chessboard
885                 cornerSubPix(matFrame, pointBuf, Size(11, 11), Size(-1, -1), TermCriteria(CV_TERMCRIT_EPS +
CV_TERMCRIT_ITER, 30, 0.1));
886
887                 imagePoints.push_back(pointBuf);
888                 number_samples += 1;
889                 commObj.addLog(QString::fromStdString(ss.str()));
890                 QCoreApplication::processEvents();
891             }
892             frame->Release();
893             ss.str("");
894             ss << "Samples found = " << number_samples;
895             commObj.progressUpdate(number_samples * 100 / imagesToSample);
896         }
897         Sleep(2);
898     }
899
900     std::vector<std::vector<Point3f> > objectPoints(1);
901     calcBoardCornerPositions(boardSize, squareSize, objectPoints[0]);
902     objectPoints.resize(imagePoints.size(), objectPoints[0]);
903
904     double rms = calibrateCamera(objectPoints, imagePoints, imageSize,
cameraMatrix, distCoeffs, Rvec, Tvec);
905     commObj.progressUpdate(0);
906     //! Release camera ----
907     camera->Release();
908
909     //! Save the obtained calibration coefficients in a file for later use
910     QString fileName = QFileDialog::getSaveFileName(nullptr, "Save calibration file", "", "Calibration File
(*.xml);;All Files (*)");
911     FileStorage fs(fileName.toUtf8().constData(), FileStorage::WRITE);
912     fs << "CameraMatrix" << cameraMatrix;
913     fs << "DistCoeff" << distCoeffs;
914     fs << "RMS" << rms;
915     strBuf = fs.releaseAndGetString();
916     commObj.changeStatus(QString::fromStdString(strBuf));
917     commObj.addLog("Saved calibration!");
918     return 0;
919 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.2.2 calibrateGround()

```
int calibrateGround ( )
```

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

Definition at line 1563 of file `main.cpp`.

```

1564 {
1565     //! initialize the variables with starting values
1566     gotOrder = false;
1567     posRef = 0;
1568     eulerRef = 0;
1569     RmatRef = 0;
1570     Rvec = RvecOriginal;
1571     Tvec = TvecOriginal;
1572
1573     determineExposure();
1574
1575     ss.str("");
1576     commObj.addLog("Started ground calibration");
1577
1578     CameraLibrary_EnableDevelopment();
1579     //! Initialize Camera SDK ===
1580     CameraLibrary::CameraManager::X();
1581
1582     //! At this point the Camera SDK is actively looking for all connected cameras and will initialize
1583     //! them on it's own.
1584
1585     //! Get a connected camera =====
1586     CameraManager::X().WaitForInitialization();
1587     Camera *camera = CameraManager::X().GetCamera();
1588
1589     //! If no device connected, pop a message box and exit ===
1590     if (camera == 0)
1591     {
1592         commObj.addLog("No camera found!");
1593         return 1;
1594     }
1595
1596     //! Determine camera resolution to size application window =====
1597     int cameraWidth = camera->Width();
1598     int cameraHeight = camera->Height();
1599     camera->GetDistortionModel(distModel);
1600     cv::Mat matFrame(cv::Size(cameraWidth, cameraHeight), CV_8UC1);
1601
1602     //! Set camera mode to precision mode, it directly provides marker coordinates
1603     camera->SetVideoType(Core::PrecisionMode);
1604
1605     //! Start camera output ===
1606     camera->Start();
1607
1608     //! Turn on some overlay text so it's clear things are =====
1609     //! working even if there is nothing in the camera's view. =====
1610     //! Set some other parameters as well of the camera
1611     camera->SetTextOverlay(true);
1612     camera->SetFrameRate(intFrameRate);
1613     camera->SetIntensity(intIntensity);
1614     camera->SetIRFilter(true);
1615     camera->SetContinuousIR(false);
1616     camera->SetHighPowerMode(false);
1617
1618     //! sample some frames and calculate the position and attitude. then average those values and use that
1619     as zero position
1620     int numberSamples = 0;
1621     int numberToSample = 200;
1622     double projectionError = 0;
1623     while (numberSamples < numberToSample)
1624     {
1625         //! Fetch a new frame from the camera =====
1626         Frame *frame = camera->GetFrame();
1627
1628         if (frame)
1629         {
1630             //! Ok, we've received a new frame, lets do something
1631             //! with it.
1632             if (frame->ObjectCount() == numberMarkers)
1633             {
1634                 //!for(int i=0; i<frame->ObjectCount(); i++)
1635                 for (int i = 0; i < numberMarkers; i++)
1636                 {
1637                     cObject *obj = frame->Object(i);
1638                     list_points2dUnsorted[i] = cv::Point2d(obj->X(), obj->Y());
1639                 }
1640
1641                 if (gotOrder == false)
1642                 {
1643                     determineOrder();
1644                 }
1645
1646                 //! sort the 2d points with the correct indices as found in the preceeding order
1647                 determination algorithm
1648                 for (int w = 0; w < numberMarkers; w++)
1649                 {

```

```

1649         list_points2d[w] = list_points2dUnsorted[
1650     pointOrderIndices[w]];
1651         list_points2dOld = list_points2dUnsorted;
1652
1653         ///Compute the pose from the 3D-2D correspondences
1654     solvePnP(list_points3d, list_points2d,
1655     cameraMatrix, distCoeffs, Rvec, Tvec, useGuess,
1656     methodPNP);
1657
1658     /// project the marker 3d points with the solution into the camera image CoSy and calculate
1659     difference to true camera image
1660     projectPoints(list_points3d, Rvec, Tvec,
1661     cameraMatrix, distCoeffs, list_points2dProjected);
1662     projectionError = norm(list_points2dProjected,
1663     list_points2d);
1664
1665     if (projectionError > 3)
1666     {
1667         commObj.addLog("Reprojection error is bigger than 3 pixel. Correct marker
1668     configuration loaded?\nMarker position measured precisely?");
1669         frame->Release();
1670         return 1;
1671     }
1672
1673     double maxValue = 0;
1674     double minValue = 0;
1675     minMaxLoc(Tvec.at<double>(2), &minValue, &maxValue);
1676
1677     if (maxValue > 10000 || minValue < 0)
1678     {
1679         commObj.addLog("Negative z distance, thats not possible. Start the set
1680     zero routine again and check marker configurations.");
1681         frame->Release();
1682         return 1;
1683     }
1684
1685     if (norm(positionOld) - norm(Tvec) < 0.05)    ///!<Iterative Method needs time
1686     to converge to solution
1687     {
1688         add(posRef, Tvec, posRef);
1689         add(eulerRef, Rvec, eulerRef);    ///!< That are not the values of yaw,
1690     roll and pitch yet! Rodriguez has to be called first.
1691         numberSamples++;    ///!<-- one sample more :D
1692         commObj.progressUpdate(numberSamples * 100 / numberToSample);
1693     }
1694     positionOld = Tvec;
1695
1696     Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
1697     for (int i = 0; i < numberMarkers; i++)
1698     {
1699         circle(cFrame, Point(list_points2d[i].x,
1700     list_points2d[i].y), 6, Scalar(0, 225, 0), 3);
1701     }
1702     projectCoordinateFrame(cFrame);
1703     projectPoints(list_points3d, Rvec, Tvec,
1704     cameraMatrix, distCoeffs, list_points2d);
1705     for (int i = 0; i < numberMarkers; i++)
1706     {
1707         circle(cFrame, Point(list_points2d[i].x,
1708     list_points2d[i].y), 3, Scalar(225, 0, 0), 3);
1709     }
1710
1711     QPixmap QPFrame;
1712     QPFrame = Mat2QPixmap(cFrame);
1713     commObj.changeImage(QPFrame);
1714     QApplication::processEvents();
1715
1716     }
1717     frame->Release();
1718 }
1719
1720     ///! Release camera ----
1721     camera->Release();
1722
1723     ///!Divide by the number of samples to get the mean of the reference position
1724     divide(posRef, numberToSample, posRef);
1725     divide(eulerRef, numberToSample, eulerRef);    ///!< eulerRef is here in Axis Angle
1726     notation
1727
1728     Rodrigues(eulerRef, RmatRef);    ///!< axis angle to rotation matrix
1729
1730     getEulerAngles(RmatRef, eulerRef);    ///!< rotation matrix to euler
1731     ss.str("");
1732     ss << "RmatRef is:\n";

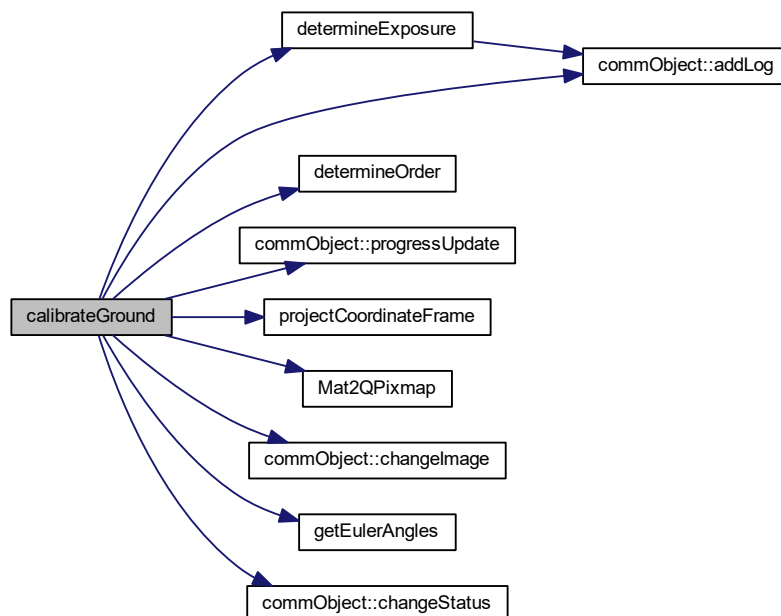
```

```

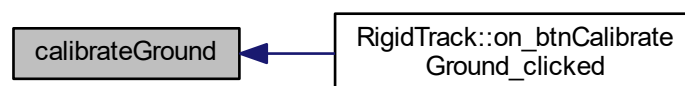
1722     ss << RmatRef << "\n";
1723     ss << "Reference Position is:\n";
1724     ss << posRef << "[mm] \n";
1725     ss << "Reference Euler angles are:\n";
1726     ss << eulerRef << "[deg] \n";
1727
1728     //!< Save the obtained calibration coefficients in a file for later use
1729     QString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

```

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.2.3 closeUDP()

```
void closeUDP ( )
```

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

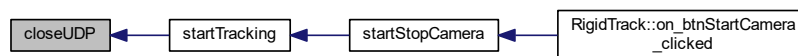
Definition at line 1173 of file main.cpp.

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

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.2.4 determineExposure()

```
int determineExposure ( )
```

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

Definition at line 1362 of file main.cpp.

```

1363 {
1364     ///! For OptiTrack Ethernet cameras, it's important to enable development mode if you
1365     ///! want to stop execution for an extended time while debugging without disconnecting
1366     ///! the Ethernet devices. Lets do that now:
1367
1368     CameraLibrary_EnableDevelopment();
1369
1370     ///! Initialize Camera SDK ---
1371     CameraLibrary::CameraManager::X();
1372
1373     ///! At this point the Camera SDK is actively looking for all connected cameras and will initialize
1374     ///! them on it's own.
1375
1376     ///! Get a connected camera =====
1377     CameraManager::X().WaitForInitialization();
1378     Camera *camera = CameraManager::X().GetCamera();
1379
1380     ///! If no device connected, pop a message box and exit ---
1381     if (camera == 0)
1382     {
1383         commObj.addLog("No camera found!");
1384         return 1;
1385     }
1386
1387     ///! Determine camera resolution to size application window -----
1388     int cameraWidth = camera->Width();
1389     int cameraHeight = camera->Height();
1390
1391     camera->SetVideoType(Core::PrecisionMode); ///! set the camera mode to precision mode, it used
greyscale information for marker property calculations
1392
1393     ///! Start camera output ---
1394     camera->Start();
1395
1396     ///! Turn on some overlay text so it's clear things are -----
1397     ///! working even if there is nothing in the camera's view. -----
1398     camera->SetTextOverlay(true);
1399     camera->SetExposure(intExposure); ///! set the camera exposure
1400     camera->SetIntensity(intIntensity); ///! set the camera infrared LED intensity
1401     camera->SetFrameRate(intFrameRate); ///! set the camera framerate to 100 Hz
1402     camera->SetIRFilter(true); ///! enable the filter that blocks visible light and only passes infrared
light
1403     camera->SetHighPowerMode(true); ///! enable high power mode of the leds
1404     camera->SetContinuousIR(false); ///! enable continuous LED light
1405     camera->SetThreshold(intThreshold); ///! set threshold for marker detection
1406
1407     ///!set exposure such that num markers are visible
1408     int numberObjects = 0; ///! Number of objects (markers) found in the current picture with the given
exposure
1409     int minExposure = 1; ///! exposure when objects detected the first time is numberMarkers
1410     int maxExposure = 480; ///! exposure when objects detected is first time numberMarkers+1
1411     intExposure = minExposure; ///! set the exposure to the smallest value possible
1412     int numberTries = 0; ///! if the markers arent found after numberTries then there might be no markers
at all in the real world
1413
1414     ///! Determine minimum exposure, hence when are numberMarkers objects detected
1415     camera->SetExposure(intExposure);
1416     while (numberObjects != numberMarkers && numberTries < 48)
1417     {
1418         ///! get a new camera frame
1419         Frame *frame = camera->GetFrame();
1420         if (frame) ///! frame received
1421         {
1422             numberObjects = frame->ObjectCount(); ///! how many objects are detected in the image
1423             if (numberObjects == numberMarkers) { minExposure =
intExposure; frame->Release(); break; } ///! if the right amount if markers is found, exit while
loop
1424             ///! not the right amount of markers was found so increase the exposure and try again
1425             numberTries++;
1426             intExposure += 10;
1427             camera->SetExposure(intExposure);
1428             ss.str("");
1429             ss << "Exposure: " << intExposure << "\t";
1430             ss << "Objects found: " << numberObjects;
1431             commObj.addLog(QString::fromStdString(ss.str()));
1432             frame->Release();
1433         }
1434     }
1435
1436     ///! Now determine maximum exposure, hence when are numberMarkers+1 objects detected
1437     numberTries = 0; ///! if the markers arent found after numberTries then there might be no markers at
all in the real world
1438     intExposure = maxExposure;
1439     camera->SetExposure(intExposure);
1440     numberObjects = 0;
1441     while (numberObjects != numberMarkers && numberTries < 48)
1442     {

```

```

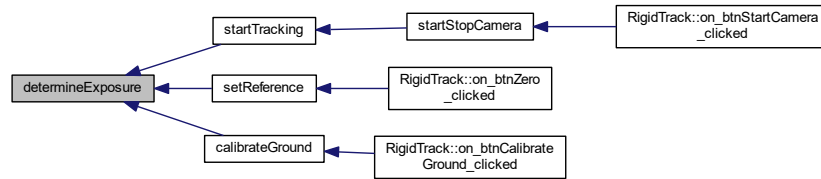
1443     Frame *frame = camera->GetFrame();
1444     if (frame)
1445     {
1446         numberObjects = frame->ObjectCount(); //!< how many objects are detected in the image
1447         if (numberObjects == numberMarkers) { maxExposure =
intExposure; frame->Release(); break; } //!< if the right amount of markers is found, exit while
        loop
1448
1449         //!< not the right amount of markers was found so decrease the exposure and try again
1450         intExposure -= 10;
1451         numberTries++;
1452         camera->SetExposure(intExposure);
1453         ss.str("");
1454         ss << "Exposure: " << intExposure << "\t";
1455         ss << "Objects found: " << numberObjects;
1456         commObj.addLog(QString::fromStdString(ss.str()));
1457         frame->Release();
1458     }
1459 }
1460
1461 //!< set the exposure to the mean of min and max exposure determined
1462 camera->SetExposure((minExposure + maxExposure) / 2.0);
1463
1464 //!< and now check if the correct amount of markers is detected with that new value
1465 while (1)
1466 {
1467     Frame *frame = camera->GetFrame();
1468     if (frame)
1469     {
1470         numberObjects = frame->ObjectCount(); //!< how many objects are detected in the image
1471         if (numberObjects != numberMarkers) //!< are all markers and not more or less
detected in the image
1472         {
1473             frame->Release();
1474             commObj.addLog("Was not able to detect the right amount of markers.");
1475             //!< Release camera ==--
1476             camera->Release();
1477             return 1;
1478         }
1479         else //!< all markers and not more or less are found
1480         {
1481             frame->Release();
1482             intExposure = (minExposure + maxExposure) / 2.0;
1483             commObj.addLog("Found the correct number of markers.");
1484             commObj.addLog("Exposure set to:");
1485             commObj.addLog(QString::number(intExposure));
1486             break;
1487         }
1488     }
1489 }
1490
1491 camera->Release();
1492 return 0;
1493
1494 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.2.5 determineOrder()

```
void determineOrder ( )
```

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

Definition at line 1498 of file main.cpp.

```

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

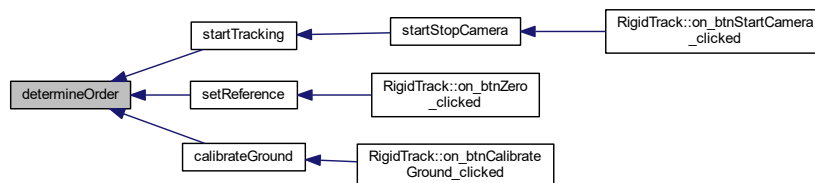
```

```

        smallest value of difference to the current one
1542         for (int b = 0; b < numberMarkers; b++)    //!< now safe the better permutation
1543         {
1544             pointOrderIndicesNew[b] = pointOrderIndices[b];
1545         }
1546     }
1547
1548 }
1549
1550 //!< try every permutation
1551 while (std::next_permutation(pointOrderIndices,
pointOrderIndices + 4));
1552
1553 //!< now that the correct order is found assign it to the indices array
1554 for (int w = 0; w < numberMarkers; w++)
1555 {
1556     pointOrderIndices[w] = pointOrderIndicesNew[w];
1557 }
1558 gotOrder = true;
1559 }

```

Here is the caller graph for this function:



6.6.2.6 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 1315 of file main.cpp.

```

1316 {
1317     ss.str("");
1318     ss << "X: " << Position[0] << " m";
1319     putText(Picture, ss.str(), cv::Point(200, 440), 1, 1, cv::Scalar(255, 255, 255));

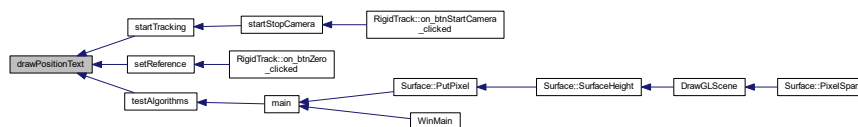
```

```

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

```

Here is the caller graph for this function:



6.6.2.7 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 923 of file main.cpp.

```

923                                     {
924
925     QString fileName;
926     if (method == 0)
927     {
928         fileName = "calibration.xml";
929     }
930     else
931     {
932         fileName = QFileDialog::getOpenFileName(nullptr, "Choose a previous saved calibration file", "", "
Calibration Files (*.xml);;All Files (*)");
933         if (fileName.length() == 0)
934         {

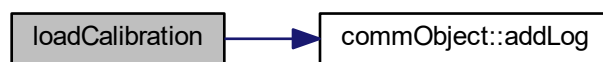
```

```

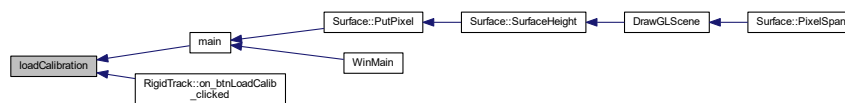
935         fileName = "calibration.xml";
936     }
937 }
938 FileStorage fs;
939 fs.open(fileName.toUtf8().constData(), FileStorage::READ);
940 fs["CameraMatrix"] >> cameraMatrix;
941 fs["DistCoeff"] >> distCoeffs;
942 commObj.addLog("Loaded calibration from file:");
943 commObj.addLog(fileName);
944 ss.str("");
945 ss << "\nCamera Matrix is" << "\n" << cameraMatrix << "\n";
946 ss << "\nDistortion Coefficients are" << "\n" << distCoeffs << "\n";
947 commObj.addLog(QString::fromStdString(ss.str()));
948 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.2.8 loadCameraPosition()

```
void loadCameraPosition ( )
```

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

Definition at line 1348 of file main.cpp.

```

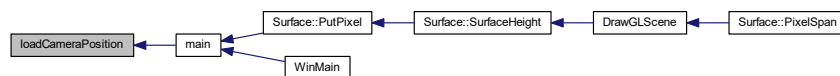
1349 {
1350     //! Open the referenceData.xml that contains the rotation from camera CoSy to ground CoSy
1351     FileStorage fs;
1352     fs.open("referenceData.xml", FileStorage::READ);
1353     fs["M_NC"] >> M_CN;
1354     fs["M_NC"] >> RmatRef;
1355     fs["posRef"] >> posRef;
1356     fs["eulerRef"]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

```

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.2.9 loadMarkerConfig()

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

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

Parameters

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

Definition at line 1195 of file main.cpp.

```

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

```



```

        numberMarkers);
1215
1216        ///! save the marker locations in the points3d vector
1217        fs["list_points3d"] >> list_points3d;
1218        fs.release();
1219        commObj.addLog("Loaded marker configuration from file:");
1220        commObj.addLog(fileName);
1221
1222    }
1223
1224    }
1225    else
1226    {
1227        ///! if the load marker configuration button was clicked show a open file dialog
1228        fileName = QFileDialog::getOpenFileName(nullptr, "Choose a previous saved marker configuration file", "", "marker configuratio files (*.xml);;All Files (*)");
1229
1230        ///! was cancel or abort clicked
1231        if (fileName.length() == 0)
1232        {
1233            ///! if yes load the standard marker configuration
1234            fileName = "markerStandard.xml";
1235        }
1236
1237        ///! open the selected marker configuration file
1238        FileStorage fs;
1239        fs.open(fileName.toUtf8().constData(), FileStorage::READ);
1240
1241        ///! copy the values to the respective variables
1242        fs["numberMarkers"] >> numberMarkers;
1243
1244        ///! initialize vectors with correct length depending on the number of markers
1245        list_points3d = std::vector<Point3d>(numberMarkers);
1246        list_points2d = std::vector<Point2d>(numberMarkers);
1247        list_points2dOld = std::vector<Point2d>(numberMarkers);
1248        list_points2dDifference = std::vector<double>(numberMarkers);
1249        list_points2dProjected = std::vector<Point2d>(numberMarkers);
1250        list_points2dUnsorted = std::vector<Point2d>(numberMarkers);
1251
1252        ///! save the marker locations in the points3d vector
1253        fs["list_points3d"] >> list_points3d;
1254        fs.release();
1255        commObj.addLog("Loaded marker configuration from file:");
1256        commObj.addLog(fileName);
1257    }
1258
1259    ///! Print out the number of markers and their position to the GUI
1260    ss.str("");
1261    ss << "Number of Markers: " << numberMarkers << "\n";
1262    ss << "Marker 3D Points X,Y and Z [mm]: \n";
1263    for (int i = 0; i < numberMarkers; i++)
1264    {
1265        ss << "Marker " << i + 1 << ":\t" << list_points3d[i].x << "\t" << list_points3d[i].y << "\t" << list_points3d[i].z << "\n";
1266    }
1267    commObj.addLog(QString::fromStdString(ss.str()));
1268
1269    ///! check if P3P algorithm can be enabled, it needs exactly 4 marker points to work
1270    if (numberMarkers == 4)
1271    {
1272        ///! if P3P is possible, let the user choose which algorithm he wants but keep iterative active
1273        methodPNP = 0;
1274        commObj.enableP3P(true);
1275    }
1276    else
1277    {
1278        ///! More (or less) marker than 4 loaded, P3P is not possible, hence user cant select P3P in GUI
1279        methodPNP = 0;
1280        commObj.enableP3P(false);
1281        commObj.addLog("P3P algorithm disabled, only works with 4 markers.");
1282    }
1283
1284    ///! now display the marker configuration in the camera view
1285    Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
1286
1287    ///! Set the camera pose parallel to the marker coordinate system
1288    Tvec.at<double>(0) = 0;
1289    Tvec.at<double>(1) = 0;
1290    Tvec.at<double>(2) = 4500;
1291    Rvec.at<double>(0) = 0 * 3.141592653589 / 180.0;
1292    Rvec.at<double>(1) = 0 * 3.141592653589 / 180.0;
1293    Rvec.at<double>(2) = -90. * 3.141592653589 / 180.0;
1294
1295    projectPoints(list_points3d, Rvec, Tvec, cameraMatrix,
1296    distCoeffs, list_points2dProjected);
1297    for (int i = 0; i < numberMarkers; i++)

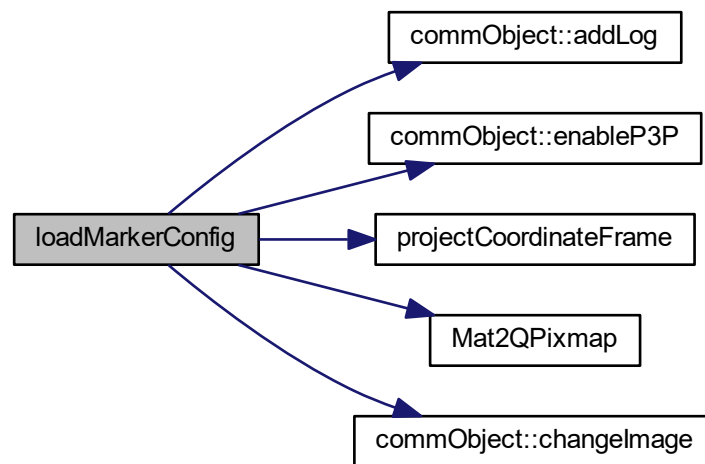
```

```

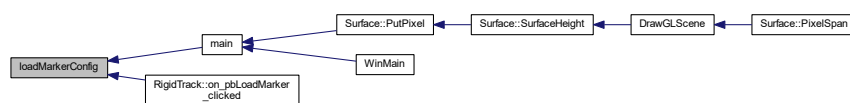
1298     {
1299         circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 3, Scalar(255, 0, 0
1300     ), 3);
1301     }
1302     projectCoordinateFrame(cFrame);
1303     QPixmap QPFrame;
1304     QPFrame = Mat2QPixmap(cFrame);
1305     commObj.changeImage(QPFrame);
1306     QApplication::processEvents();
1307
1308 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.2.10 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

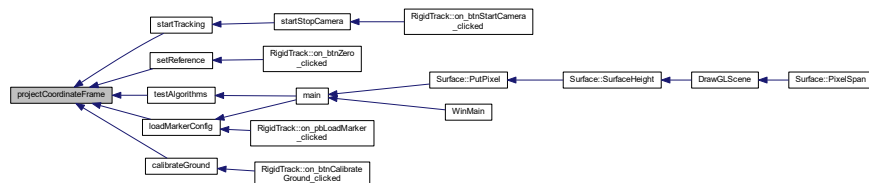
in	<i>pictureFrame</i>	the image in which the CoSy frame should be pasted.
----	---------------------	---

Definition at line 1081 of file main.cpp.

```

1082 {
1083     projectPoints(coordinateFrame, Rvec, Tvec,
1084                 cameraMatrix, distCoeffs, coordinateFrameProjected);
1085     line(pictureFrame, coordinateFrameProjected[0],
1086         coordinateFrameProjected[3], Scalar(0, 0, 255), 2); ///z-axis
1087     line(pictureFrame, coordinateFrameProjected[0],
1088         coordinateFrameProjected[1], Scalar(255, 0, 0), 2); ///x-axis
1089     line(pictureFrame, coordinateFrameProjected[0],
1090         coordinateFrameProjected[2], Scalar(0, 255, 0), 2); ///y-axis
1091 }
```

Here is the caller graph for this function:



6.6.2.11 sendDataUDP()

```

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

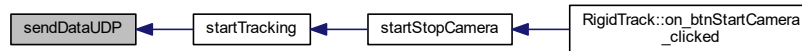
Send the position and attitude over UDP to every receiver, the safety receiver is handled on its own in the startTracking function because its send rate is less than 100 Hz.

Definition at line 1154 of file main.cpp.

```

1155 {
1156     datagram.clear();
1157     QDataStream out(&datagram, QIODevice::WriteOnly);
1158     out.setVersion(QDataStream::Qt_4_3);
1159     out << (float)Position[0] << (float)Position[1] << (float)Position[2];
1160     out << (float)Euler[0] << (float)Euler[1] << (float)Euler[2]; ///Roll Pitch Heading
1161     udpSocketObject->writeDatagram(datagram,
1162     IPAddressObject, portObject);
1163     ///if second receiver is activated send it also the tracking data
1164     if (safety2Enable)
1165     {
1166         udpSocketSafety2->writeDatagram(datagram,
1167         IPAddressSafety2, portSafety2);
1168     }
1169 }
```

Here is the caller graph for this function:



6.6.2.12 setHeadingOffset()

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

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

Parameters

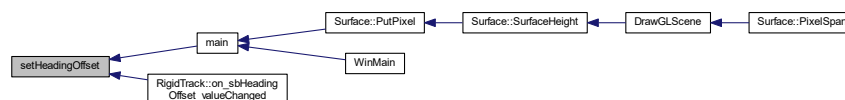
in	<i>d</i>	denotes heading offset in degrees.
----	----------	------------------------------------

Definition at line 1122 of file main.cpp.

```

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

Here is the caller graph for this function:



6.6.2.13 setReference()

```
int setReference ( )
```

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

Definition at line 595 of file main.cpp.

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

```

666         //!for(int i=0; i<frame->ObjectCount(); i++)
667         for (int i = 0; i < numberMarkers; i++)
668         {
669             cObject *obj = frame->Object(i);
670             list_points2dUnsorted[i] = cv::Point2d(obj->X(), obj->Y());
671         }
672
673         if (gotOrder == false)
674         {
675             determineOrder();
676         }
677
678         //! sort the 2d points with the correct indices as found in the preceeding order
679         determination algorithm
680         for (int w = 0; w < numberMarkers; w++)
681         {
682             list_points2d[w] = list_points2dUnsorted[
pointOrderIndices[w]];
683         }
684         list_points2dOld = list_points2dUnsorted;
685
686         //!Compute the pose from the 3D-2D correspondences
687         solvePnP(list_points3d, list_points2d,
cameraMatrix, distCoeffs, Rvec, Tvec, useGuess,
methodPNP);
688
689         //! project the marker 3d points with the solution into the camera image CoSy and calculate
        difference to true camera image
690         projectPoints(list_points3d, Rvec, Tvec,
cameraMatrix, distCoeffs, list_points2dProjected);
691         projectionError = norm(list_points2dProjected,
list_points2d);
692
693         double maxValue = 0;
694         double minValue = 0;
695         minMaxLoc(Tvec.at<double>(2), &minValue, &maxValue);
696
697         if (maxValue > 10000 || minValue < 0)
698         {
699             ss.str("");
700             ss << "Negative z distance, thats not possible. Start the set zero routine again or
restart Programm.";
701             commObj.addLog(QString::fromStdString(ss.str()));
702             frame->Release();
703             return 1;
704         }
705
706         if (projectionError > 3)
707         {
708             commObj.addLog("Reprojection error is bigger than 3 pixel. Correct marker
configuration loaded?\nMarker position measured precisely?");
709             frame->Release();
710             return 1;
711         }
712
713         if (norm(positionOld) - norm(Tvec) < 0.05)    //!<Iterative Method needs time
        to converge to solution
714         {
715             add(posRef, Tvec, posRef);
716             add(eulerRef, Rvec, eulerRef);    //!< That are not the values of yaw,
roll and pitch yet! Rodriguez has to be called first.
717             numberSamples++;    //!< one sample more :D
718             commObj.progressUpdate(numberSamples * 100 / numberToSample);
719         }
720         positionOld = Tvec;
721
722         Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
723         for (int i = 0; i < numberMarkers; i++)
724         {
725             circle(cFrame, Point(list_points2d[i].x,
list_points2d[i].y), 6, Scalar(0, 225, 0), 3);
726         }
727         projectCoordinateFrame(cFrame);
728         projectPoints(list_points3d, Rvec, Tvec,
cameraMatrix, distCoeffs, list_points2d);
729         for (int i = 0; i < numberMarkers; i++)
730         {
731             circle(cFrame, Point(list_points2d[i].x,
list_points2d[i].y), 3, Scalar(225, 0, 0), 3);
732         }
733         drawPositionText(cFrame, position,
eulerAngles, projectionError);
734
735         QPixmap QPFrame;
736         QPFrame = Mat2QPixmap(cFrame);
737         commObj.changeImage(QPFrame);
738         QApplication::processEvents();

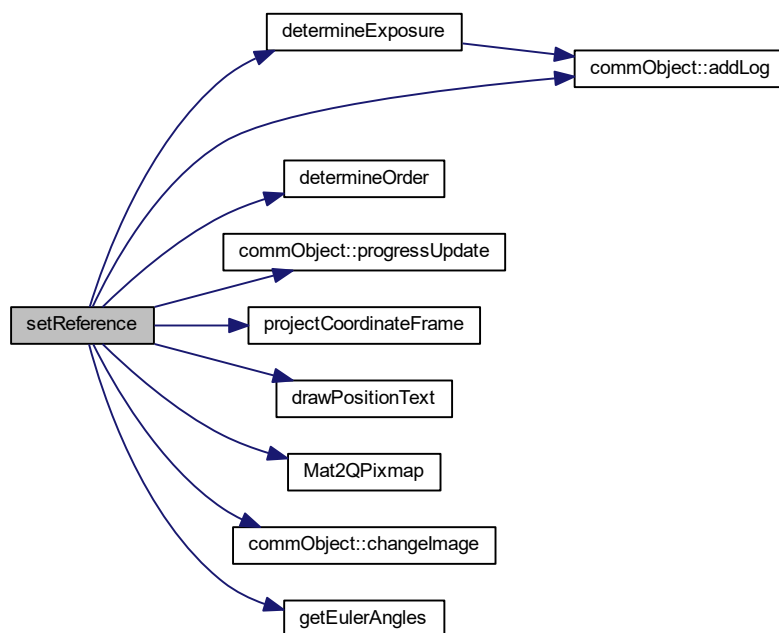
```

```

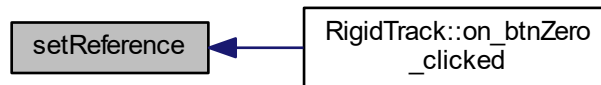
738
739     }
740     frame->Release();
741 }
742 }
743 ///! Release camera ==--
744 camera->Release();
745
746 ///! Divide by the number of samples to get the mean of the reference position
747 divide(posRef, numberToSample, posRef);
748 divide(eulerRef, numberToSample, eulerRef); ///!< eulerRef is here in Axis Angle
notation
749
750 Rodrigues(eulerRef, RmatRef); ///!< axis angle to rotation matrix
751 ///!-- Euler Angles, finally
752 getEulerAngles(RmatRef, eulerRef); ///!< rotation matrix to euler
753 ss.str("");
754 ss << "RmatRef is:\n";
755 ss << RmatRef << "\n";
756 ss << "Reference Position is:\n";
757 ss << posRef << "[mm] \n";
758 ss << "Reference Euler Angles are:\n";
759 ss << eulerRef << "[deg] \n";
760
761 ///! compute the difference between last obtained TVec and the average Value
762 ///! When it is large the iterative method has not converged properly so it is advised to start the
setReference() function once again
763 double error = norm(posRef) - norm(Tvec);
764 if (error > 5.0)
765 {
766     ss << "Caution, distance between reference position and last position is: " << error << "\n Start
the set zero routine once again.";
767 }
768 commObj.addLog(QString::fromStdString(ss.str()));
769 commObj.progressUpdate(0);
770 return 0;
771 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.2.14 setUpUDP()

```
void setUpUDP ( )
```

Open the UDP ports for communication.

Definition at line 1090 of file main.cpp.

```

1091 {
1092     //! Initialise the QDataStream that stores the data to be send
1093     QDataStream out(&datagram, QIODevice::WriteOnly);
1094     out.setVersion(QDataStream::Qt_4_3);
1095
1096     //! Create UDP slots
1097     commObj.addLog("Opening UDP ports.");
1098     udpSocketObject = new QUdpSocket(0);
1099     udpSocketObject->connectToHost(IPAddressObject,
portObject);
1100     commObj.addLog("Opened first receiver UDP port.");
1101
1102     udpSocketSafety = new QUdpSocket(0);
1103     udpSocketSafety2 = new QUdpSocket(0);
1104
1105     //! if the safety feature is activated open the udp port
1106     if (safetyEnable)
1107     {
1108         udpSocketSafety->connectToHost(IPAddressSafety,
portSafety);
1109         commObj.addLog("Opened safety UDP port.");
1110     }
1111
1112     //! if the second receiver feature is activated open the udp port
1113     if (safety2Enable)
1114     {
1115         udpSocketSafety2->connectToHost(IPAddressSafety2,
portSafety2);
1116         commObj.addLog("Opened second receiver UDP port.");
1117     }
1118 }
  
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.2.15 startStopCamera()

```
void startStopCamera ( )
```

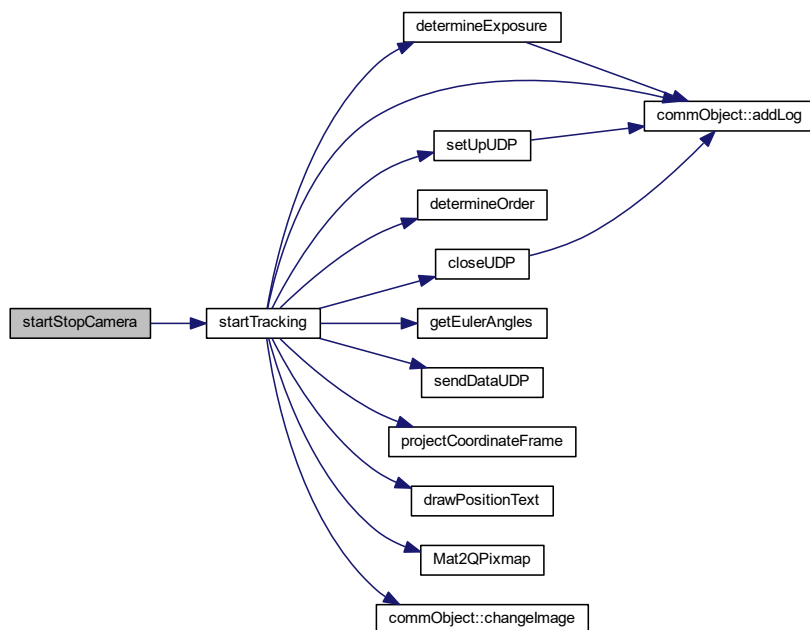
Start or stop the tracking depending on if the camera is currently running or not.

Definition at line 579 of file main.cpp.

```

580 {
581     ///! tracking is not running so start it
582     if (exitRequested)
583     {
584         exitRequested = false;
585         startTracking();
586     }
587     else ///!< tracking is currently running, set exitRequest to true so the while loop in startTracking()
588     exits
589     {
590         exitRequested = true;
591     }
592 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.2.16 startTracking()

```
int startTracking ( )
```

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

Definition at line 261 of file main.cpp.

```

261         {
262
263
264         gotOrder = false; //!< The order of points, hence which entry in list_points3d corresponds to
                which in list_points2d is not calculated yet
265         Rvec = RvecOriginal; //!< Use the value of Rvec that was set in main() as starting value
                for the solvePnP algorithm
266         Tvec = TvecOriginal; //!< Use the value of Tvec that was set in main() as starting value
                for the solvePnP algorithm
267         GetLocalTime(&logDate); //!< Get the current date and time to name the log file
268
269         //!< Concat the log file name as followed. The file is saved in the folder /logs in the Rigid Track
                installation folder
270         logFileName = "./logs/positionLog_" + QString::number(logDate.wDay) + "_" +
                QString::number(logDate.wMonth) + "_" + QString::number(logDate.wYear);
271         logFileName += "_" + QString::number(logDate.wHour) + "_" + QString::number(
                logDate.wMinute) + "_" + QString::number(logDate.wSecond) + ".txt";
272         logName = logFileName.toString(); //!< Convert the QString to a standard string
273
274         determineExposure(); //!< Get the exposure where the right amount of markers is
                detected
275
276         //!< For OptiTrack Ethernet cameras, it's important to enable development mode if you
277         //!< want to stop execution for an extended time while debugging without disconnecting
278         //!< the Ethernet devices. Lets do that now:
279
280         CameraLibrary_EnableDevelopment();
281         CameraLibrary::CameraManager::X(); //!< Initialize Camera SDK
282
283         //!< At this point the Camera SDK is actively looking for all connected cameras and will initialize
284         //!< them on it's own
285
286         //!< Get a connected camera
287         CameraManager::X().WaitForInitialization();
288         Camera *camera = CameraManager::X().GetCamera();
289
290         //!< If no camera can be found, inform user in message log and exit function
291         if (camera == 0)
292         {
293             commObj.addLog("No camera found!");
294             return 1;
295         }
296
297         //!< Determine camera resolution to size application window
298         int cameraWidth = camera->Width();
299         int cameraHeight = camera->Height();
  
```

```

300
301     camera->SetVideoType(Core::PrecisionMode); /// Set the camera mode to precision mode, it used
greyscale information for marker property calculations
302
303     camera->Start(); /// Start camera output
304
305     /// Turn on some overlay text so it's clear things are
306     /// working even if there is nothing in the camera's view
307     camera->SetTextOverlay(true);
308     camera->SetExposure(intExposure); /// Set the camera exposure
309     camera->SetIntensity(intIntensity); /// Set the camera infrared LED intensity
310     camera->SetFrameRate(intFrameRate); /// Set the camera framerate to 100 Hz
311     camera->SetIRFilter(true); /// Enable the filter that blocks visible light and only passes infrared
light
312     camera->SetHighPowerMode(true); /// Enable high power mode of the LEDs
313     camera->SetContinuousIR(false); /// Disable continuous LED light
314     camera->SetThreshold(intThreshold); /// Set threshold for marker detection
315
316     /// Create a new matrix that stores the grayscale picture from the camera
317     Mat matFrame = Mat::zeros(cv::Size(cameraWidth, cameraHeight), CV_8UC1);
318     QPixmap QPFrame; /// QPixmap is the corresponding Qt class that saves images
319     /// Matrix that stores the colored picture, hence marker points, coordinate frame and reprojected
points
320     Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
321
322     int v = 0; /// Helper variable used to kick safety switch
323     /// Variables for the min and max values that are needed for sanity checks
324     double maxValue = 0;
325     double minValue = 0;
326     int framesDropped = 0; /// If a marker is not visible or accuracy is bad increase this counter
327     double projectionError = 0; /// Equals the quality of the tracking
328
329     setUpUDP(); /// Open sockets and ports for UDP communication
330
331     if (safetyEnable) /// If the safety feature is enabled send the starting message
332     {
333         /// Send enable message, hence send a 9 and then a 1
334         data.setNum((int)(9));
335         udpSocketSafety->write(data);
336         data.setNum((int)(1));
337         udpSocketSafety->write(data);
338     }
339
340     /// Fetch a new frame from the camera
341     bool gotTime = false; /// Get the timestamp of the first frame. This time is subtracted from every
subseeding frame so the time starts at 0 in the logs
342     while (!gotTime) /// While no new frame is received loop
343     {
344         Frame *frame = camera->GetFrame(); /// Get a new camera frame
345         if (frame) /// There is actually a new frame
346         {
347             timeFirstFrame = frame->TimeStamp(); /// Get the time stamp for the first frame.
It is subtracted for the following frames
348             frame->Release(); /// Release the frame so the camera can continue
349             gotTime = true; /// Exit the while loop
350         }
351     }
352
353     /// Now enter the main loop that processes each frame and computes the pose, sends it and logs stuff
354     while (!exitRequested) /// Check if the user has not pressed "Stop Tracking" yet
355     {
356         Frame *frame = camera->GetFrame(); /// Fetch a new frame from the camera
357
358         if (frame) /// Did we got a new frame or does the camera still need more time
359         {
360             framesDropped++; /// Increase by one, if everything is okay it is decreased at the end of the
loop again
361
362             /// Only use this frame if the right number of markers is found in the picture
363             if (frame->ObjectCount() == numberMarkers)
364             {
365                 /// Get the marker points in 2D in the camera image frame and store them in the
list_points2dUnsorted vector
366                 /// The order of points that come from the camera corresponds to the Y coordinate
367                 for (int i = 0; i < numberMarkers; i++)
368                 {
369                     cObject *obj = frame->Object(i);
370                     list_points2dUnsorted[i] = cv::Point2d(obj->X(), obj->Y());
371                 }
372
373                 if (gotOrder == false) /// Was the order already determined? This is false for the
first frame and from then on true
374                 {
375                     determineOrder(); /// Now compute the order
376                 }
377             }
378

```

```

379         //! Sort the 2d points with the correct indices as found in the preceeding order
determination algorithm
380         for (int w = 0; w < numberMarkers; w++)
381         {
382             list_points2d[w] = list_points2dUnsorted[
pointOrderIndices[w]]; //! pointOrderIndices was calculated in determineOrder()
383         }
384         list_points2dOld = list_points2dUnsorted;
385
386         //! The first time the 2D-3D corresspondence was determined with gotOrder was okay.
387         //! But this order can change as the object moves and the marker objects appear in a
388         //! different order in the frame->Object() array.
389         //! The solution is that: When a marker point (in the camera image, hence in 2D) was at
390         //! a position then it wont move that much from one frame to the other.
391         //! So for the new frame we take a marker object and check which marker was closest this
point
392         //! in the old image frame? This is probably the same (true) marker. And we do that for
every other marker as well.
393         //! When tracking is good and no frames are dropped because of missing markers this should
work every frame.
394         for (int j = 0; j < numberMarkers; j++)
395         {
396             minPointDistance = 5000; //! The sum of point distances is set to
something unrealistic large
397             for (int k = 0; k < numberMarkers; k++)
398             {
399                 //! Calculate N_2 norm of unsorted points minus old points
400                 currentPointDistance = norm(
list_points2dUnsorted[pointOrderIndices[j]] -
list_points2dOld[k]);
401                 //! If the norm is smaller than minPointDistance the correspondence is more likely
to be correct
402                 if (currentPointDistance <
minPointDistance)
403                 {
404                     //! Update the array that saves the new point order
405                     minPointDistance =
currentPointDistance;
406                     pointOrderIndicesNew[j] = k;
407                 }
408             }
409         }
410
411         //! Now the new order is found, set the point order to the new value
412         for (int k = 0; k < numberMarkers; k++)
413         {
414             pointOrderIndices[k] = pointOrderIndicesNew[k];
415             list_points2d[k] = list_points2dUnsorted[
pointOrderIndices[k]];
416         }
417
418         //! Save the unsorted position of the marker points for the next loop
419         list_points2dOld = list_points2dUnsorted;
420
421         //! Compute the object pose from the 3D-2D correspondences
422         cameraMatrix, solvePnP(list_points3d, list_points2d,
distCoeffs, Rvec, Tvec, useGuess,
methodPNP);
423
424         //! Project the marker 3d points with the solution into the camera image CoSy and calculate
difference to true camera image
425         projectPoints(list_points3d, Rvec, Tvec,
cameraMatrix, distCoeffs, list_points2dProjected);
426         projectionError = norm(list_points2dProjected,
list_points2d); //! Difference of true pose and found pose
427
428         //! Increase the framesDropped variable if accuracy of tracking is too bad
429         if (projectionError > 5)
430         {
431             framesDropped++;
432         }
433         else
434         {
435             framesDropped = 0; //! Set number of subsequent frames dropped to zero because error
is small enough and no marker was missing
436         }
437
438         //! Get the min and max values from Tvec for sanity check
439         minMaxLoc(Tvec.at<double>(2), &minValue, &maxValue);
440
441         //! Sanity check of values. negative z means the marker CoSy is behind the camera, that's
not possible.
442         if (minValue < 0)
443         {
444             commObj.addLog("Negative z distance, that is not possible. Start the set
zero routine again or restart Program.");
445             frame->Release(); //! Release the frame so the camera can move on

```

```

446         camera->Release(); /// Release the camera
447         closeUDP(); /// Close all UDP connections so the programm can be closed later
on and no resources are locked
448         return 1; /// Exit the function
449     }
450
451     /// Next step is the transformation from camera CoSy to navigation CoSy
452     /// Compute the relative object position from the reference position to the current one
453     /// given in the camera CoSy: \f$ T_C^{NM} = Tvec - Tvec_{Ref} \f$
454     subtract(Tvec, posRef, position);
455
456     /// Transform the position from the camera CoSy to the navigation CoSy with INS alligned
heading and convert from [mm] to [m]
457     /// \f$ T_N^{NM} = M_{NC} \backslashtimes T_C^{NM} \f$
458     Mat V = 0.001 * M_HeadingOffset * M_CN.t() * (Mat)
position;
459     position = V; /// Position is the result of the preceeding calculation
460     position[2] *= invertZ; /// Invert Z if check box in GUI is activated,
hence height above ground is considered
461
462     /// Realtime angle between reference orientation and current orientation
463     Rodrigues(Rvec, Rmat); /// Convert axis angle respresentation to ordinary rotation
matrix
464
465     /// The difference of the reference rotation and the current rotation
466     /// \f$ R_{NM} = M_{NC} \backslashtimes R_{CM} \f$
467     Rmat = RmatRef.t() * Rmat;
468
469     /// Euler Angles, finally
470     getEulerAngles(Rmat, eulerAngles); /// Get the euler angles
from the rotation matrix
471     eulerAngles[2] += headingOffset; /// Add the heading offset to the
heading angle
472
473     /// Compute the velocity with finite differences. Only use is the log file. It is done here
because the more precise time stamp can be used
474     frameTime = frame->TimeStamp() - timeOld; /// Time between the old frame
and the current frame
475     timeOld = frame->TimeStamp(); /// Set the old frame time to the current one
476     velocity[0] = (position[0] - positionOld[0]) /
frameTime; /// Calculate the x velocity with finite differences
477     velocity[1] = (position[1] - positionOld[1]) /
frameTime; /// Calculate the y velocity with finite differences
478     velocity[2] = (position[2] - positionOld[2]) /
frameTime; /// Calculate the z velocity with finite differences
479     positionOld = position; /// Set the old position to the current one for
next frame velocity calcaution
480
481     /// Send position and Euler angles over WiFi with 100 Hz
482     sendDataUDP(position, eulerAngles);
483
484     /// Save the values in a log file, values are:
485     /// Time sinc tracking started Position Euler Angles Velocity
486     logfile.open(logName, std::ios::app); /// Open the log file, the folder is
RigidTrackInstallationFolder/logs
487     logfile << frame->TimeStamp() - timeFirstFrame << ";" <<
position[0] << ";" << position[1] << ";" << position[2] << ";";
488     logfile << eulerAngles[0] << ";" <<
eulerAngles[1] << ";" << eulerAngles[2] << ";";
489     logfile << velocity[0] << ";" << velocity[1] << ";" <<
velocity[2] << "\n";
490     logfile.close(); /// Close the file to save values
491 }
492
493     /// Check if the position and euler angles are below the allowed value, if yes send OKAY signal
(1), if not send shutdown signal (0)
494     /// Absolute x, y and z position in navigation CoSy must be smaller than the allowed distance
495     if (safetyEnable)
496     {
497         if ((abs(position[0]) < safetyBoxLength && abs(position[1]) <
safetyBoxLength && abs(position[2]) < safetyBoxLength))
498         {
499             /// Absolute Euler angles must be smaller than allowed value. Heading is not considered
500             if ((abs(eulerAngles[0]) < safetyAngle && abs(eulerAngles[1]) <
safetyAngle))
501             {
502                 /// Send the OKAY signal to the desired computer every 5th time
503                 if (v == 5) {
504                     data.setNum((int)(1));
505                     udpSocketSafety->write(data); /// Send the 1
506                     v = 0; /// reset the counter that is needed for decimation to every 5th time
step
507                 }
508             }
509             /// The euler angles of the object exceeded the allowed euler angles, send the shutdown
signal (0)

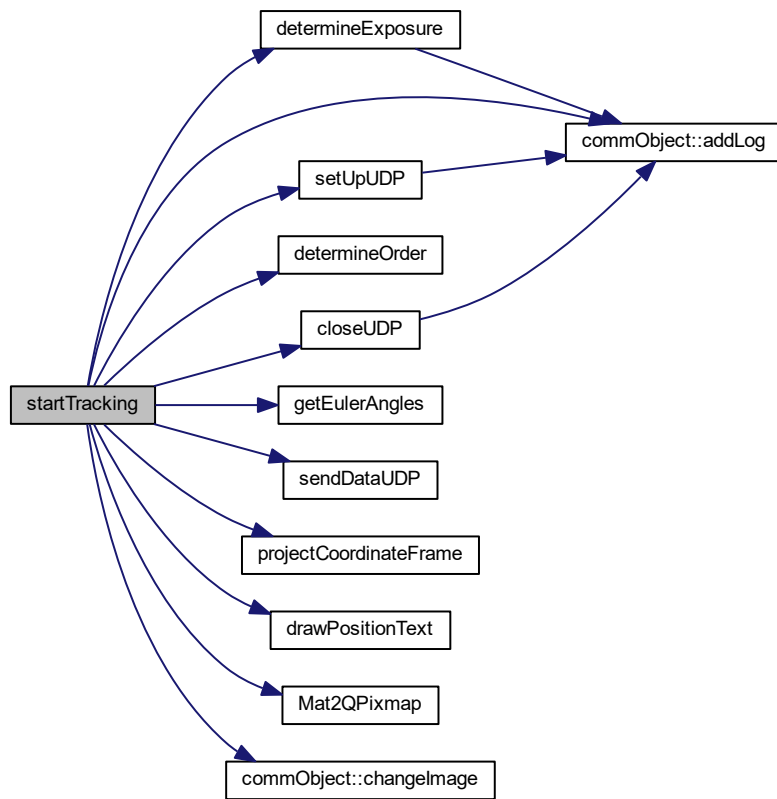
```

```

510         else
511         {
512             data.setNum((int) (0)); /// Send the shutdown signal, a 0
513             udpSocketSafety->write(data);
514             commObj.addLog("Object exceeded allowed Euler angles, shutdown signal
sent."); /// Inform the user
515         }
516     }
517     /// The position of the object exceeded the allowed position, shut the object down
518     else
519     {
520         data.setNum((int) (0)); /// Send the shutdown signal, a 0
521         udpSocketSafety->write(data);
522         commObj.addLog("Object left allowed area, shutdown signal sent."); ///
Inform the user
523     }
524 }
525 }
526 }
527
528 /// Inform the user if tracking system is disturbed (marker lost or so) or error was too big
529 if (framesDropped > 10)
530 {
531     if (safetyEnable) /// Also send the shutdown signal
532     {
533         data.setNum((int) (0)); /// Send the shutdown signal, a 0
534         udpSocketSafety->write(data);
535     }
536     commObj.addLog("Lost marker points or precision was bad!"); /// Inform the
user
537     framesDropped = 0;
538 }
539
540 /// Rasterize the frame so it can be shown in the GUI
541 frame->Rasterize(cameraWidth, cameraHeight, matFrame.step,
BACKBUFFER_BITSPERPIXEL, matFrame.data);
542
543 /// Convert the frame from greyscale as it comes from the camera to rgb color
544 cvtColor(matFrame, cFrame, COLOR_GRAY2RGB);
545
546 /// Project (draw) the marker CoSy origin into 2D and save it in the cFrame image
547 projectCoordinateFrame(cFrame);
548
549 /// Project the marker points from 3D to the camera image frame (2d) with the computed pose
550 projectPoints(list_points3d, Rvec, Tvec,
cameraMatrix, distCoeffs, list_points2d);
551 for (int i = 0; i < numberMarkers; i++)
552 {
553     /// Draw a circle around the projected points so the result can be better compared to the
real marker position
554     /// In the resulting picture those are the red dots
555     circle(cFrame, Point(list_points2d[i].x,
list_points2d[i].y), 3, Scalar(225, 0, 0), 3);
556 }
557
558 /// Write the current position, attitude and error values as text in the frame
559 drawPositionText(cFrame, position, eulerAngles, projectionError);
560
561 /// Send the new camera picture to the GUI and call the GUI processing routine
562 QPixmap QPFrame;
563 QPFrame = Mat2QPixmap(cFrame);
564 commObj.changeImage(QPFrame); /// Update the picture in the GUI
565 QApplication::processEvents(); /// Give Qt time to handle everything
566
567 /// Release the camera frame to fetch the new one
568 frame->Release();
569 }
570 }
571
572 /// User choose to stop the tracking, clean things up
573 closeUDP(); /// Close the UDP connections so resources are deallocated
574 camera->Release(); /// Release camera
575 return 0;
576 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.2.17 testAlgorithms()

```
void testAlgorithms ( )
```

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

Definition at line 952 of file `main.cpp`.

```

953 {
954
955     int _methodPNP;
956
957     std::vector<Point2d> noise(numberMarkers);
958
959     RvecOriginal = Rvec;
960     TvecOriginal = Tvec;
961
962     projectPoints(list_points3d, Rvec, Tvec, cameraMatrix,
distCoeffs, list_points2dProjected);
963
964     ss.str("");
965     ss << "Unsorted Points 2D Projected \n";
966     ss << list_points2dProjected << "\n";
967     commObj.addLog(QString::fromStdString(ss.str()));
968
969     Mat cFrame(480, 640, CV_8UC3, Scalar(0, 0, 0));
970     for (int i = 0; i < numberMarkers; i++)
971     {
972         circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 6, Scalar(0, 255, 0
), 3);
973     }
974
975     projectCoordinateFrame(cFrame);
976
977     ss.str("");
978     ss << "=====\n";
979     ss << "===== Projected Points =====\n";
980     ss << list_points2dProjected << "\n";
981
982     randn(noise, 0, 0.5);
983     add(list_points2dProjected, noise, list_points2dProjected);
984
985     ss << "===== With Noise Points =====\n";
986     ss << list_points2dProjected << "\n";
987     commObj.addLog(QString::fromStdString(ss.str()));
988
989     bool useGuess = true;
990     _methodPNP = 0; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP //!< not used
991
992     solvePnP(list_points3d, list_points2dProjected, cameraMatrix,
distCoeffs, Rvec, Tvec, useGuess, _methodPNP);
993
994     ss.str("");
995     ss << "=====\n";
996     ss << "===== Iterative =====\n";
997     ss << "rvec: " << "\n";
998     ss << Rvec << "\n";
999     ss << "tvec: " << "\n";
1000     ss << Tvec << "\n";
1001
1002     commObj.addLog(QString::fromStdString(ss.str()));
1003
1004     _methodPNP = 1; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP UPnP not used
1005     Rvec = cv::Mat::zeros(3, 1, CV_64F);
1006     Tvec = cv::Mat::zeros(3, 1, CV_64F);
1007     solvePnP(list_points3d, list_points2dProjected, cameraMatrix,
distCoeffs, Rvec, Tvec, useGuess, _methodPNP);
1008
1009     ss.str("");
1010     ss << "=====\n";
1011     ss << "===== EPNP =====\n";
1012     ss << "rvec: " << "\n";
1013     ss << Rvec << "\n";
1014     ss << "tvec: " << "\n";
1015     ss << Tvec << "\n";
1016
1017     projectPoints(list_points3d, Rvec, Tvec, cameraMatrix,
distCoeffs, list_points2dProjected);
1018     for (int i = 0; i < numberMarkers; i++)
1019     {
1020         circle(cFrame, Point(list_points2dProjected[i].x, list_points2dProjected[i].y), 3, Scalar(255, 0, 0
), 3);
1021     }
1022     QPixmap QPFrame;
1023     QPFrame = Mat2QPixmap(cFrame);
1024     commObj.changeImage(QPFrame);
1025     QCoreApplication::processEvents();
1026     commObj.addLog(QString::fromStdString(ss.str()));
1027     if (numberMarkers == 4)
1028     {
1029         _methodPNP = 2; //!< 0 = iterative 1 = EPNP 2 = P3P 4 = UPNP //!< not used
1030         Rvec = cv::Mat::zeros(3, 1, CV_64F);
1031         Tvec = cv::Mat::zeros(3, 1, CV_64F);
1032         solvePnP(list_points3d, list_points2dProjected,

```

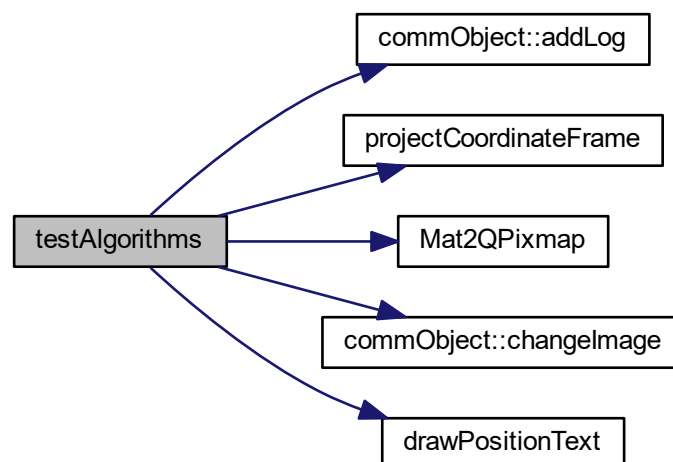


```

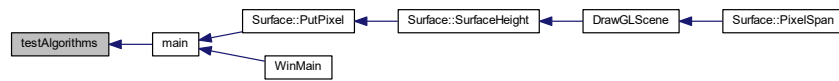
cameraMatrix, distCoeffs, Rvec, Tvec, useGuess, _methodPNP);
1034
1035     ss.str("");
1036     ss << "=====\n";
1037     ss << "===== P3P =====\n";
1038     ss << "rvec: " << "\n";
1039     ss << Rvec << "\n";
1040     ss << "tvec: " << "\n";
1041     ss << Tvec << "\n";
1042
1043     projectPoints(list_points3d,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

```

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.3 Variable Documentation

6.6.3.1 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`.

6.6.3.2 invertZ

`int` `invertZ`

dummy variable to invert Z direction on request

Definition at line 75 of file `main.cpp`.

6.6.3.3 IPAddressObject

`QHostAddress` `IPAddressObject`

IPv4 adress of receiver 1.

Definition at line 131 of file `main.cpp`.

6.6.3.4 IPAdressSafety

`QHostAddress IPAdressSafety`

IPv4 adress of safety receiver.

Definition at line 132 of file main.cpp.

6.6.3.5 IPAdressSafety2

`QHostAddress IPAdressSafety2`

IPv4 adress of receiver 2.

Definition at line 133 of file main.cpp.

6.6.3.6 methodPNP

`int methodPNP`

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

Definition at line 105 of file main.cpp.

6.6.3.7 portObject

`int portObject`

Port of receiver 1.

Definition at line 134 of file main.cpp.

6.6.3.8 portSafety

`int portSafety`

Port of the safety receiver.

Definition at line 135 of file main.cpp.

6.6.3.9 portSafety2

```
int portSafety2
```

Port of receiver 2.

Definition at line 136 of file main.cpp.

6.6.3.10 safety2Enable

```
bool safety2Enable
```

is the second receiver enabled

Definition at line 71 of file main.cpp.

6.6.3.11 safetyAngle

```
int safetyAngle
```

bank and pitch angle protection in degrees

Definition at line 73 of file main.cpp.

6.6.3.12 safetyBoxLength

```
double safetyBoxLength
```

length of the safety area cube in meters

Definition at line 72 of file main.cpp.

6.6.3.13 safetyEnable

```
bool safetyEnable
```

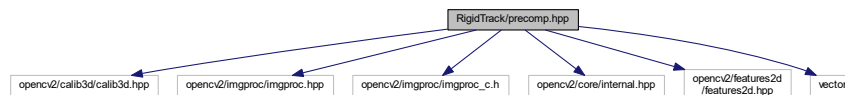
is the safety feature enabled

Definition at line 70 of file main.cpp.

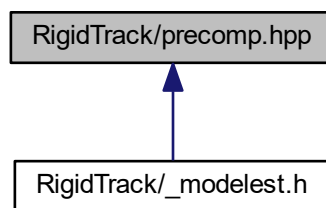
6.7 RigidTrack/precomp.hpp File Reference

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

Include dependency graph for precomp.hpp:



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



Macros

- #define [GET_OPTIMIZED](#)(func) (func)

6.7.1 Macro Definition Documentation

6.7.1.1 GET_OPTIMIZED

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

Definition at line 59 of file precomp.hpp.

6.8 RigidTrack/resource.h File Reference

Macros

- `#define IDI_ICON1 101`
/<{{NO_DEPENDENCIES}} /< Von Microsoft Visual C++ generierte Includedatei. /< Verwendet durch RigidTrack.rc /<

6.8.1 Macro Definition Documentation

6.8.1.1 IDI_ICON1

```
#define IDI_ICON1 101
```

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

Definition at line 5 of file resource.h.

6.9 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:
```



6.9.1 Detailed Description

Rigid Track GUI source that contains functions for GUI events.

Author

Florian J.T. Wachter

Version

1.0

Date

April, 8th 2017

6.10 RigidTrack/RigidTrack.h File Reference

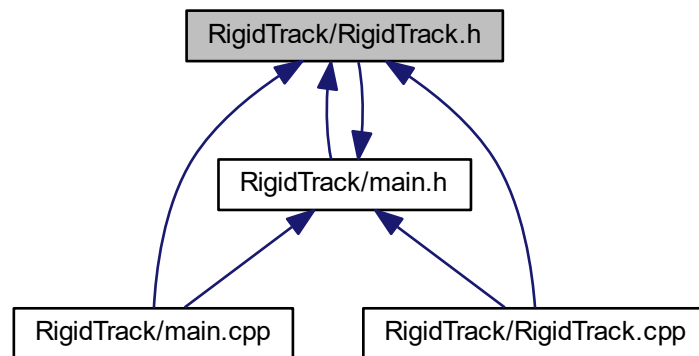
Rigid Track GUI source header with Qt Signals and Slots.

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

Include dependency graph for RigidTrack.h:



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



Classes

- class [RigidTrack](#)

6.10.1 Detailed Description

Rigid Track GUI source header with Qt Signals and Slots.

Author

Florian J.T. Wachter

Version

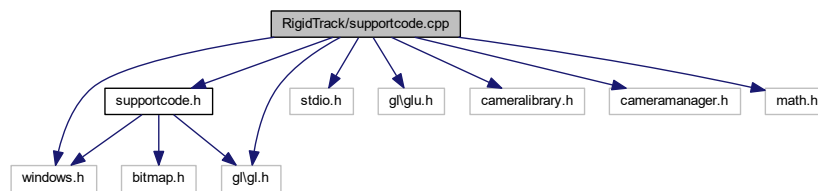
1.0

Date

April, 8th 2017

6.11 RigidTrack/supportcode.cpp File Reference

```
#include <windows.h>
#include <stdio.h>
#include <gl\gl.h>
#include <gl\glu.h>
#include "cameralibrary.h"
#include "cameramanager.h"
#include "math.h"
#include "supportcode.h"
Include dependency graph for supportcode.cpp:
```



Functions

- int [LoadGLTextures](#) ()
 - /< Permanent Rendering Context*
- GLvoid [ReSizeGLScene](#) (GLsizei width, GLsizei height)
 - /< Resize And Initialize The GL Window*
- int [InitGL](#) (GLvoid)
 - /< All Setup For OpenGL Goes Here*
- int [DrawGLScene](#) ([Surface](#) *surf, int width, int height)
- LRESULT CALLBACK [WndProc](#) (HWND [hWnd](#), UINT [uMsg](#), WPARAM [wParam](#), LPARAM [lParam](#))
 - /< Additional Message Information*
- GLvoid [CloseWindow](#) (GLvoid)
 - /< Properly Kill The Window*
- BOOL [CreateAppWindow](#) (const char *title, int width, int height, int bits, bool fullscreenflag)
- int [main](#) (int argc, char *argv[])
 - main initialises the GUI and values for the marker position etc*
- int WINAPI [WinMain](#) (HINSTANCE [hInstance](#), HINSTANCE [hPrevInstance](#), LPSTR [lpCmdLine](#), int [nCmdShow](#))
 - /< Window Show State*
- bool [FullscreenToggle](#) ()
- bool [PumpMessages](#) ()
- LRESULT CALLBACK [CBTHookProc](#) (int [nCode](#), WPARAM [wParam](#), LPARAM [lParam](#))
- VOID CALLBACK [TimerProc](#) (HWND [hWnd](#), UINT [uMsg](#), UINT [idEvent](#), DWORD [dwTime](#))
- bool [PopWaitingDialog](#) ()

Variables

- int `gWindowWidth`

```

/*****
- /<== This is boiler-plate code for bringing up the application's window and /<== initializing OpenGL,
and an OpenGL surface class for rendering the camera /<== image as a quad using the 3D hardware.
*****/

```
- int `gWindowHeight`
- bool `gFullscreen` = FALSE

```

/< Window Active Flag Set To TRUE By Default

```
- bool `gActive` = TRUE

```

/< Array Used For Scanning Keyboard

```
- bool `keys` [256]

```

/< Private GDI Device Context

```
- HDC `hDC` =NULL
- GLuint `texture` [1]

```

/< Fullscreen Flag Set To Fullscreen Mode By Default

```
- int `gSoftwareDecimate` = 0

```

/< Storage For One Texture ( NEW )

```
- HWND `hWnd` =NULL

```

/< Private GDI Device Context

```
- HINSTANCE `hInstance`

```

/< Holds Our Window Handle

```
- HGLRC `hRC` =NULL

```

/< Holds The Instance Of The Application

```
- int `windowWidth`
- int `windowHeight`
- const char * `windowName`
- HHOOK `hHook` = NULL

```

/<== Code to pop a simple dialog for 'waiting for cameras' using a message box and /<== no resources required for
this sample application.

```

6.11.1 Function Documentation

6.11.1.1 CBTHookProc()

```

LRESULT CALLBACK CBTHookProc (
    int nCode,
    WPARAM wParam,
    LPARAM lParam )

```

Definition at line 575 of file supportcode.cpp.

```

576 {
577     if (nCode < 0)
578         return CallNextHookEx(hHook, nCode, wParam, lParam);
579
580     if (nCode == HCBT_ACTIVATE)
581     {
582         HWND hWnd = reinterpret_cast<HWND>(wParam);
583         SetWindowText(GetDlgItem(hWnd, IDOK), TEXT("Cancel"));
584         return 0;
585     }
586
587     return CallNextHookEx(hHook, nCode, wParam, lParam);
588 }

```

Here is the caller graph for this function:



6.11.1.2 CloseWindow()

```
GLvoid CloseWindow (
    GLvoid )
```

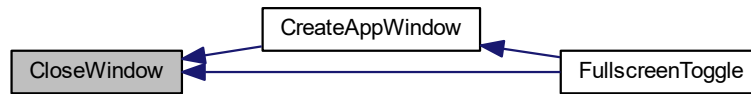
/< Properly Kill The Window

Definition at line 192 of file supportcode.cpp.

```

193 {
194     if (gFullscreen)                                //!< Are We In Fullscreen Mode?
195     {
196         ChangeDisplaySettings(NULL,0);                //!< If So Switch Back To The Desktop
197         ShowCursor(TRUE);                             //!< Show Mouse Pointer
198     }
199
200     if (hRC)                                          //!< Do We Have A Rendering Context?
201     {
202         if (!wglMakeCurrent(NULL,NULL))              //!< Are We Able To Release The DC And RC
203             Contexts?
204             {
205                 MessageBox(NULL,L"Release Of DC And RC Failed.",L"SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
206             }
207         if (!wglDeleteContext(hRC))                  //!< Are We Able To Delete The RC?
208         {
209             MessageBox(NULL, L"Release Rendering Context Failed.", L"SHUTDOWN ERROR",MB_OK |
210             MB_ICONINFORMATION);
211         }
212         hRC=NULL;                                    //!< Set RC To NULL
213     }
214     if (hDC && !ReleaseDC(hWnd,hDC))                //!< Are We Able To Release The DC
215     {
216         MessageBox(NULL, L"Release Device Context Failed.", L"SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
217         hDC=NULL;                                    //!< Set DC To NULL
218     }
219
220     if (hWnd && !DestroyWindow(hWnd))                //!< Are We Able To Destroy The Window?
221     {
222         MessageBox(NULL, L"Could Not Release hWnd.", L"SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
223         hWnd=NULL;                                    //!< Set hWnd To NULL
224     }
225
226     if (!UnregisterClass(L"OpenGL",hInstance))        //!< Are We Able To Unregister Class
227     {
228         MessageBox(NULL, L"Could Not Unregister Class.", L"SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
229         hInstance=NULL;                               //!< Set hInstance To NULL
230     }
231 }
```

Here is the caller graph for this function:



6.11.1.3 CreateAppWindow()

```

BOOL CreateAppWindow (
    const char * title,
    int width,
    int height,
    int bits,
    bool fullscreenflag )
  
```

Definition at line 244 of file supportcode.cpp.

```

245 {
246     windowHeight = width;
247     windowHeight = height;
248     windowName = title;
249
250     GLuint PixelFormat;          ///< Holds The Results After Searching For A Match
251     WNDCLASS wc;                 ///< Windows Class Structure
252     DWORD dwExStyle;             ///< Window Extended Style
253     DWORD dwStyle;               ///< Window Style
254     RECT WindowRect;             ///< Grabs Rectangle Upper Left / Lower Right Values
255     WindowRect.left=(long)0;      ///< Set Left Value To 0
256     WindowRect.right=(long)width; ///< Set Right Value To Requested Width
257     WindowRect.top=(long)0;       ///< Set Top Value To 0
258     WindowRect.bottom=(long)height; ///< Set Bottom Value To Requested Height
259
260     gFullscreen=fullscreenflag;   ///< Set The Global Fullscreen Flag
261
262     HICON hIcon = (HICON) LoadImage(0, IDI_WINLOGO, IMAGE_ICON, 0, 0, LR_SHARED);
263
264     hInstance = GetModuleHandle(NULL);          ///< Grab An Instance For Our
265     Window
266     wc.style = CS_HREDRAW | CS_VREDRAW | CS_OWNDC;    ///< Redraw On Size, And Own DC For
267     Window.
268     wc.lpfnWndProc = (WNDPROC) WndProc;              ///< WndProc Handles Messages
269     wc.cbClsExtra = 0;                               ///< No Extra Window Data
270     wc.cbWndExtra = 0;                               ///< No Extra Window Data
271     wc.hInstance = hInstance;                        ///< Set The Instance
272     wc.hIcon = 0;
273     wc.hCursor = LoadCursor(NULL, IDC_ARROW);        ///< Load The Arrow Pointer
274     wc.hbrBackground = NULL;                        ///< No Background Required For GL
275     wc.lpszMenuName = NULL;                          ///< We Don't Want A Menu
276     wc.lpszClassName = L"OpenGL";                   ///< Set The Class Name
277
278     if (!RegisterClass(&wc))                        ///< Attempt To Register The Window Class
279     {
280         MessageBox(NULL, L"Failed To Register The Window Class.", L"ERROR", MB_OK|MB_ICONEXCLAMATION);
281         return FALSE;                                ///< Return FALSE
282     }
283
284     if (gFullscreen)                                  ///< Attempt Fullscreen Mode?
285     {
286         DEVMODE dmScreenSettings;                    ///< Device Mode
287         memset(&dmScreenSettings, 0, sizeof(dmScreenSettings)); ///< Makes Sure Memory's Cleared
288         dmScreenSettings.dmSize=sizeof(dmScreenSettings); ///< Size Of The Devmode Structure
  
```

```

287     dmScreenSettings.dmPelsWidth      = width;                ///< Selected Screen Width
288     dmScreenSettings.dmPelsHeight     = height;              ///< Selected Screen Height
289     dmScreenSettings.dmBitsPerPel     = bits;                ///< Selected Bits Per Pixel
290     dmScreenSettings.dmFields=DM_BITSPERPEL|DM_PELSWIDTH|DM_PELSHEIGHT;
291
292     ///< Try To Set Selected Mode And Get Results. NOTE: CDS_FULLSCREEN Gets Rid Of Start Bar.
293     if (ChangeDisplaySettings(&dmScreenSettings,CDS_FULLSCREEN)!=DISP_CHANGE_SUCCESSFUL)
294     {
295         ///< If The Mode Fails, Offer Two Options. Quit Or Use Windowed Mode.
296         if (MessageBox(NULL, L"The Requested Fullscreen Mode Is Not Supported By\nYour Video Card. Use
Windowed Mode Instead?",L"NeHe GL",MB_YESNO|MB_ICONEXCLAMATION)==IDYES)
297         {
298             gFullscreen=FALSE;                ///< Windowed Mode Selected. Fullscreen = FALSE
299         }
300         else
301         {
302             ///< Pop Up A Message Box Letting User Know The Program Is Closing.
303             MessageBox(NULL, L"Program Will Now Close.",L" RROR",MB_OK|MB_ICONSTOP);
304             return FALSE;                    ///< Return FALSE
305         }
306     }
307 }
308
309 if (gFullscreen)                                ///< Are We Still In
Fullscreen Mode?
310 {
311     dwExStyle=WS_EX_APPWINDOW;                ///< Window Extended Style
312     dwStyle=WS_POPUP;                        ///< Windows Style
313     ShowCursor(FALSE);                        ///< Hide Mouse Pointer
314 }
315 else
316 {
317     dwExStyle=WS_EX_APPWINDOW | WS_EX_WINDOWEDGE;    ///< Window Extended Style
318     dwStyle=WS_OVERLAPPEDWINDOW;                    ///< Windows Style
319 }
320
321 AdjustWindowRectEx(&WindowRect, dwStyle, FALSE, dwExStyle);    ///< Adjust Window To True Requested
Size
322
323 ///< Create The Window
324 if (!hWnd=CreateWindowEx( dwExStyle,                ///< Extended Style For The Window
325     L" penGL",                ///< Class Name
326     L"title",                ///< Window Title
327     dwStyle |                ///< Defined Window Style
328     WS_CLIPSIBLINGS |        ///< Required Window Style
329     WS_CLIPCHILDREN,         ///< Required Window Style
330     0, 0,                    ///< Window Position
331     WindowRect.right-WindowRect.left,    ///< Calculate Window Width
332     WindowRect.bottom-WindowRect.top,    ///< Calculate Window Height
333     NULL,                    ///< No Parent Window
334     NULL,                    ///< No Menu
335     hInstance,                ///< Instance
336     NULL))                    ///< Dont Pass Anything To WM_CREATE
337 {
338     CloseWindow();                ///< Reset The Display
339     MessageBox(NULL,L" indow Creation Error.",L"Error", MB_OK|MB_ICONEXCLAMATION);
340     return FALSE;                ///< Return FALSE
341 }
342
343 static PIXELFORMATDESCRIPTOR pfd=                ///< pfd Tells Windows How We Want Things To Be
344 {
345     sizeof(PIXELFORMATDESCRIPTOR),            ///< Size Of This Pixel Format Descriptor
346     1,                ///< Version Number
347     PFD_DRAW_TO_WINDOW |                ///< Format Must Support Window
348     PFD_SUPPORT_OPENGL |                ///< Format Must Support OpenGL
349     PFD_DOUBLEBUFFER,                    ///< Must Support Double Buffering
350     PFD_TYPE_RGBA,                ///< Request An RGBA Format
351     bits,                ///< Select Our Color Depth
352     0, 0, 0, 0, 0, 0,                ///< Color Bits Ignored
353     0,                ///< No Alpha Buffer
354     0,                ///< Shift Bit Ignored
355     0,                ///< No Accumulation Buffer
356     0, 0, 0, 0,                ///< Accumulation Bits Ignored
357     16,                ///< 16Bit Z-Buffer (Depth Buffer)
358     0,                ///< No Stencil Buffer
359     0,                ///< No Auxiliary Buffer
360     PFD_MAIN_PLANE,                ///< Main Drawing Layer
361     0,                ///< Reserved
362     0, 0, 0                ///< Layer Masks Ignored
363 };
364
365 if (!hDC=GetDC(hWnd))                ///< Did We Get A Device Context?
366 {
367     CloseWindow();                ///< Reset The Display
368     MessageBox(NULL,L" an't Create A GL Device Context.",L" RROR",MB_OK|MB_ICONEXCLAMATION);
369     return FALSE;                ///< Return FALSE
370 }

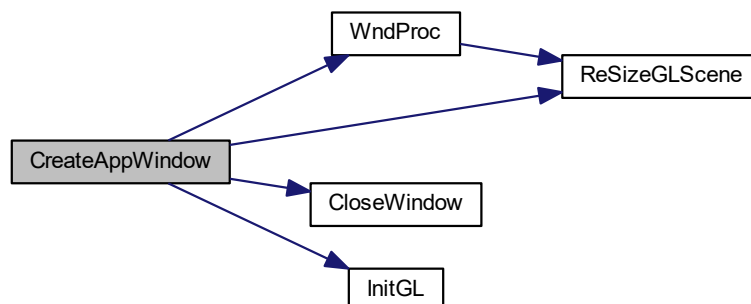
```

```

371
372  if (!PixelFormat=ChoosePixelFormat(hDC,&pfd))  ///!< Did Windows Find A Matching Pixel Format?
373  {
374      CloseWindow();                          ///!< Reset The Display
375      MessageBox(NULL, L" an't Find A Suitable PixelFormat.", L" RROR",MB_OK|MB_ICONEXCLAMATION);
376      return FALSE;                          ///!< Return FALSE
377  }
378
379  if(!SetPixelFormat(hDC,PixelFormat,&pfd))      ///!< Are We Able To Set The Pixel Format?
380  {
381      CloseWindow();                          ///!< Reset The Display
382      MessageBox(NULL, L" an't Set The PixelFormat.", L" RROR",MB_OK|MB_ICONEXCLAMATION);
383      return FALSE;                          ///!< Return FALSE
384  }
385
386  if (!hRC=wglCreateContext(hDC))              ///!< Are We Able To Get A Rendering Context?
387  {
388      CloseWindow();                          ///!< Reset The Display
389      MessageBox(NULL, L" an't Create A GL Rendering Context.", L" RROR",MB_OK|MB_ICONEXCLAMATION);
390      return FALSE;                          ///!< Return FALSE
391  }
392
393  if(!wglMakeCurrent(hDC,hRC))                 ///!< Try To Activate The Rendering Context
394  {
395      CloseWindow();                          ///!< Reset The Display
396      MessageBox(NULL, L" an't Activate The GL Rendering Context.", L" RROR",MB_OK|MB_ICONEXCLAMATION);
397      return FALSE;                          ///!< Return FALSE
398  }
399
400  ShowWindow(hWnd,SW_SHOW);                   ///!< Show The Window
401  SetForegroundWindow(hWnd);                   ///!< Slightly Higher Priority
402  SetFocus(hWnd);                             ///!< Sets Keyboard Focus To The Window
403  ResizeGLScene(width, height);               ///!< Set Up Our Perspective GL Screen
404
405  if (!InitGL())                              ///!< Initialize Our Newly Created GL Window
406  {
407      CloseWindow();                          ///!< Reset The Display
408      MessageBox(NULL, L" nitialization Failed.", L" RROR",MB_OK|MB_ICONEXCLAMATION);
409      return FALSE;                          ///!< Return FALSE
410  }
411
412  return TRUE;                                ///!< Success
413 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



6.11.1.4 DrawGLScene()

```

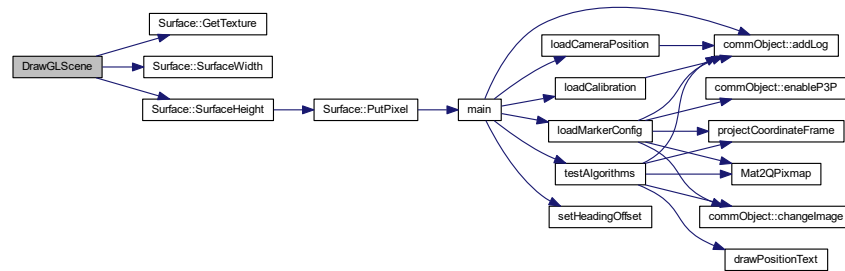
int DrawGLScene (
    Surface * surf,
    int width,
    int height )
  
```

Definition at line 76 of file supportcode.cpp.

```

77 {
78     if(surf==NULL)
79         return true;
80
81     static bool frameThrottler = true;
82     frameThrottler=!frameThrottler;
83
84     if(frameThrottler) ///<== Only display every other frame in case VSYNC is enabled. Otherwise
85         return true;    ///<== application would get behind
86
87     int pixelWidth = width;
88     int pixelHeight= height;
89
90     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); ///< Clear The Screen And The Depth Buffer
91     glLoadIdentity();                                ///< Reset The View
92
93     static GLuint ff = 0;
94
95     int tex = surf->GetTexture();
96     if(tex==0)
97         tex=ff;
98     else
99         ff=tex;
100     if(tex==0)
101         return true;
102
103     glBindTexture(GL_TEXTURE_2D, tex);
104
105     glBegin(GL_QUADS);
106         glTexCoord2f(0.0f, 0.0f); glVertex3f( 0, 0, 0);
107         glTexCoord2f((GLfloat)pixelWidth/(GLfloat)surf->SurfaceWidth(), 0.0f); glVertex3f( (
108         GLfloat)gWindowWidth, 0, 0);
109         glTexCoord2f((GLfloat)pixelWidth/(GLfloat)surf->SurfaceWidth(), (GLfloat)pixelHeight/(
110         GLfloat)surf->SurfaceHeight()); glVertex3f( (GLfloat)gWindowWidth, (GLfloat)
111         gWindowHeight, 0);
112         glTexCoord2f(0.0f, (GLfloat)pixelHeight/(GLfloat)surf->SurfaceHeight()); glVertex3f( 0
113         , (GLfloat) gWindowHeight, 0);
114     glEnd();
115
116     SwapBuffers(hDC);                                ///< Swap Buffers (Double Buffering)
117
118     return true;                                     ///< Keep Going
119 }
  
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.11.1.5 FullscreenToggle()

```
bool FullscreenToggle ( )
```

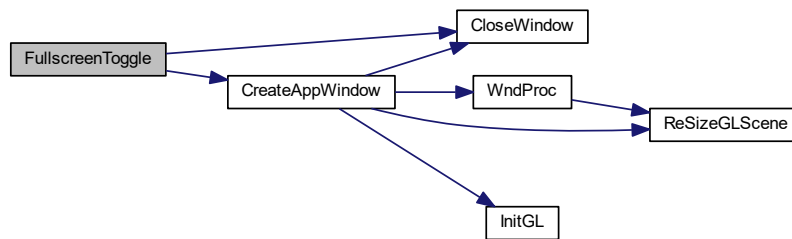
Definition at line 536 of file supportcode.cpp.

```

537 {
538     keys[VK_F1]=FALSE;                ///< If So Make Key FALSE
539     CloseWindow();                    ///< Kill Our Current Window
540     gFullscreen=!gFullscreen;         ///< Toggle Fullscreen / Windowed Mode
541
542     if (!CreateAppWindow(windowName,windowWidth,
543         windowHeight,32,gFullscreen))
544     {
545         MessageBox(0, L"Unable to toggle to full screen",L"Error", MB_OK);
546         return 1;
547     }
548     return true;
549 }

```

Here is the call graph for this function:



6.11.1.6 InitGL()

```
int InitGL (
    GLvoid )
```

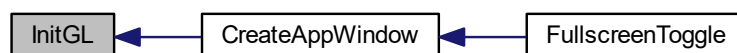
/< All Setup For OpenGL Goes Here

Definition at line 62 of file supportcode.cpp.

```

63 {
64     glEnable(GL_TEXTURE_2D);           //!< Enable Texture Mapping ( NEW )
65     glShadeModel(GL_SMOOTH);          //!< Enable Smooth Shading
66     glClearColor(0.0f, 0.0f, 0.0f, 0.5f); //!< Black Background
67     glClearDepth(1.0f);                //!< Depth Buffer Setup
68     glEnable(GL_DEPTH_TEST);           //!< Enables Depth Testing
69     glDepthFunc(GL_LEQUAL);            //!< The Type Of Depth Testing To Do
70     glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
71     glEnable(GL_BLEND);
72     glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST); //!< Really Nice Perspective Calculations
73     return TRUE;                       //!< Initialization Went OK
74 }
```

Here is the caller graph for this function:



6.11.1.7 LoadGLTextures()

```
int LoadGLTextures ( )
```

```
/< Permanent Rendering Context
```

```
/< Load Bitmaps And Convert To Textures
```

Definition at line 37 of file supportcode.cpp.

```
38 {
39     return 0;                                //!< Return The Status
40 }
```

6.11.1.8 main()

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

main initialises the GUI and values for the marker position etc

First the GUI is set up with Signals and Slots, see Qt docu for how that works. Then some variables are initialized with arbitrary values. At last calibration and marker configuration etc. are loaded from xml files.

Parameters

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

Definition at line 156 of file main.cpp.

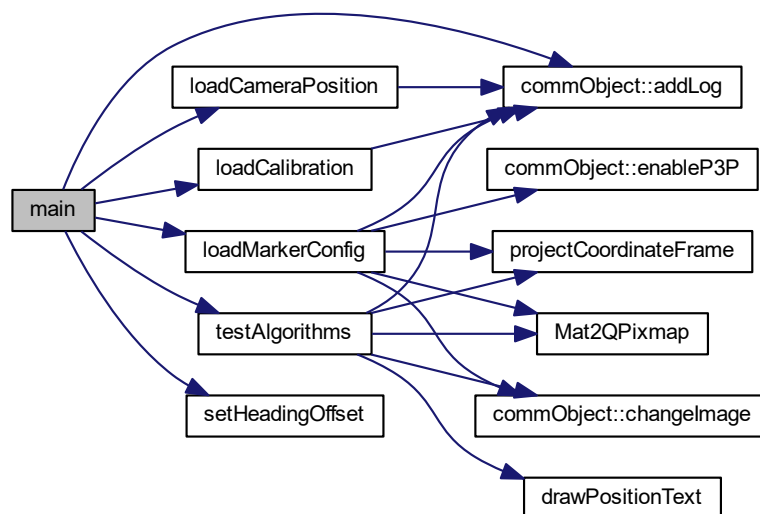
```
157 {
158     QApplication a(argc, argv);
159     RigidTrack w;
160     w.show();    //!< show the GUI
161     //!< connect the Qt slots and signals for event handling
162     QObject::connect(&commObj, SIGNAL(statusChanged(QString)), &w, SLOT(setStatus(QString)),
163         Qt::DirectConnection);
163     QObject::connect(&commObj, SIGNAL(imageChanged(QPixmap)), &w, SLOT(setImage(QPixmap)),
164         Qt::DirectConnection);
164     QObject::connect(&commObj, SIGNAL(logAdded(QString)), &w, SLOT(setLog(QString)),
165         Qt::DirectConnection);
165     QObject::connect(&commObj, SIGNAL(logCleared()), &w, SLOT(clearLog(QString)),
166         Qt::DirectConnection);
166     QObject::connect(&commObj, SIGNAL(P3Penabled(bool)), &w, SLOT(enableP3P(bool)),
167         Qt::DirectConnection);
167     QObject::connect(&commObj, SIGNAL(progressUpdated(int)), &w, SLOT(progressUpdate(int)),
168         Qt::DirectConnection);
168
169     commObj.addLog("RigidTrack Version:");
170     commObj.addLog(QString::number(_MSC_FULL_VER));
171     commObj.addLog("Built on:");
172     commObj.addLog(QString(__DATE__));
173
174     //!< initial guesses for position and rotation, important for Iterative Method!
175     Tvec.at<double>(0) = 45;
176     Tvec.at<double>(1) = 45;
177     Tvec.at<double>(2) = 4500;
```

```

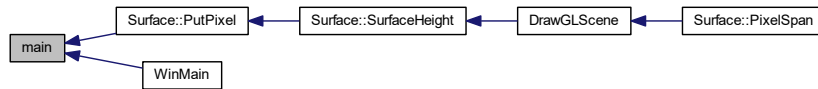
178   Rvec.at<double>(0) = 0 * 3.141592653589 / 180.0;
179   Rvec.at<double>(1) = 0 * 3.141592653589 / 180.0;
180   Rvec.at<double>(2) = -45 * 3.141592653589 / 180.0;
181
182   ///! Points that make up the marker CoSy axis system, hence one line in each axis direction
183   coordinateFrame = std::vector<Point3d>(4);
184   coordinateFrameProjected = std::vector<Point2d>(4);
185   coordinateFrame[0] = cv::Point3d(0, 0, 0);
186   coordinateFrame[1] = cv::Point3d(300, 0, 0);
187   coordinateFrame[2] = cv::Point3d(0, 300, 0);
188   coordinateFrame[3] = cv::Point3d(0, 0, 300);
189
190   position[0] = 1.1234;    ///!< set position initial values
191   position[1] = 1.2345;    ///!< set position initial values
192   position[2] = 1.3456;    ///!< set position initial values
193
194   velocity[0] = 0.123;    ///!< set velocity initial values
195   velocity[1] = 0.234;    ///!< set velocity initial values
196   velocity[2] = 0.345;    ///!< set velocity initial values
197
198   eulerAngles[0] = 1.002; ///!< set initial euler angles to arbitrary values for testing
199   eulerAngles[1] = 1.003; ///!< set initial euler angles to arbitrary values for testing
200   eulerAngles[2] = 1.004; ///!< set initial euler angles to arbitrary values for testing
201
202   setHeadingOffset(0.0); ///!< set the heading offset to 0
203
204   ss.precision(4); ///!< outputs in the log etc are limited to 3 decimal values
205
206   loadCameraPosition(); ///!< load the rotation matrix from camera CoSy to ground CoSy
207   loadCalibration(0); ///!< load the calibration file with the camera intrinsics
208   loadMarkerConfig(0); ///!< load the standard marker configuration
209   testAlgorithms(); ///!< test the algorithms and their accuracy
210
211   return a.exec();
212 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



6.11.1.9 PopWaitingDialog()

```
bool PopWaitingDialog ( )
```

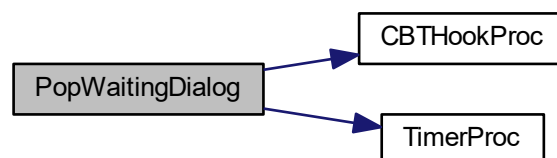
Definition at line 615 of file supportcode.cpp.

```

616 {
617     ///<== hook in so we can create a message box that has only a 'Cancel' button ==--
618
619     hHook = SetWindowsHookEx(WH_CBT, reinterpret_cast<HOOKPROC>(&
        CBTHookProc), NULL, GetCurrentThreadId());
620
621     UINT_PTR nTimer = SetTimer(0, 100, 3000, (TIMERPROC) TimerProc);
622     int iResult = MessageBox( 0, L"waiting for connected cameras...", L"Camera Initialization", MB_OK );
623
624     if( iResult == IDOK )
625     {
626         ///<== user has clicked the cancel button ===
627         UnhookWindowsHookEx(hHook);
628         return false;
629     }
630
631     KillTimer(0, nTimer);
632
633     return true;
634 }

```

Here is the call graph for this function:



6.11.1.10 PumpMessages()

bool PumpMessages ()

Definition at line 551 of file supportcode.cpp.

```

552 {
553     MSG msg;
554
555     if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))    ////< Is There A Message Waiting?
556     {
557         if (msg.message==WM_QUIT)                    ////< Have We Received A Quit Message?
558             return false;
559         else                                           ////< If Not, Deal With Window Messages
560         {
561             TranslateMessage(&msg);                  ////< Translate The Message
562             DispatchMessage(&msg);                   ////< Dispatch The Message
563         }
564     }
565
566     return true;
567 }
```

6.11.1.11 ReSizeGLScene()

GLvoid ReSizeGLScene (
 GLsizei width,
 GLsizei height)

/< Resize And Initialize The GL Window

Definition at line 42 of file supportcode.cpp.

```

43 {
44     if (height==0)                                ////< Prevent A Divide By Zero By
45     {
46         height=1;                                ////< Making Height Equal One
47     }
48
49     glViewport(0,0,width,height);                 ////< Reset The Current Viewport
50
51     glMatrixMode(GL_PROJECTION);                  ////< Select The Projection Matrix
52     glLoadIdentity();                             ////< Reset The Projection Matrix
53
54     gWindowWidth = width;
55     gWindowHeight = height;
56
57     glOrtho(0,gWindowWidth,gWindowHeight,0,100,-100);
58     glMatrixMode(GL_MODELVIEW);                  ////< Select The Modelview Matrix
59     glLoadIdentity();                             ////< Reset The Modelview Matrix
60 }
```

Here is the caller graph for this function:



6.11.1.12 TimerProc()

```

VOID CALLBACK TimerProc (
    HWND hWnd,
    UINT uMsg,
    UINT idEvent,
    DWORD dwTime )

```

Definition at line 590 of file supportcode.cpp.

```

591 {
592     CameraLibrary::CameraList list;
593
594     bool found = false;
595
596     for( int i=0; i<list.Count(); i++ )
597     {
598         if( list[i].State()==CameraLibrary::Initialized )
599         {
600             found = true;
601         }
602     }
603
604     if(found==true)
605     {
606         HWND hWndActive = GetActiveWindow();
607
608         if( hWndActive!=0 )
609         {
610             SendMessage(hWndActive, WM_COMMAND, IDCANCEL, 0);
611         }
612     }
613 }

```

Here is the caller graph for this function:



6.11.1.13 WinMain()

```

int WINAPI WinMain (
    HINSTANCE hInstance,
    HINSTANCE hPrevInstance,
    LPSTR lpCmdLine,
    int nCmdShow )

```

/< Window Show State

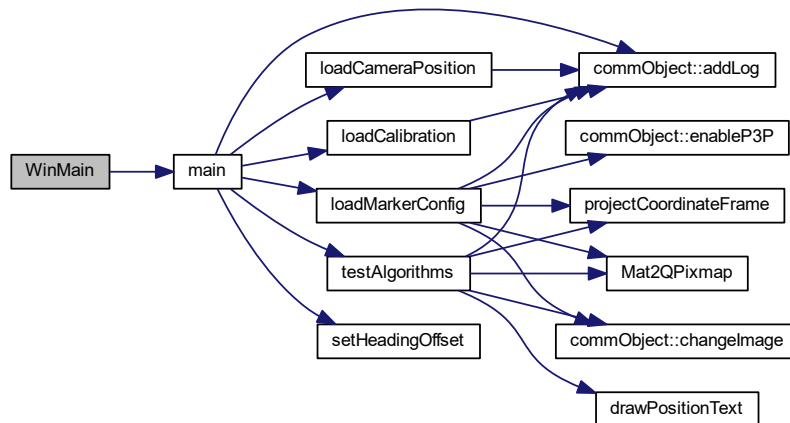
Parameters

<i>hPrevInstance</i>	/< Instance
<i>lpCmdLine</i>	/< Previous Instance
<i>nCmdShow</i>	/< Command Line Parameters

Definition at line 528 of file supportcode.cpp.

```
532 {
533     return main(0,0);
534 }
```

Here is the call graph for this function:



6.11.1.14 WndProc()

```
LRESULT CALLBACK WndProc (
    HWND hWnd,
    UINT uMsg,
    WPARAM wParam,
    LPARAM lParam )
```

/< Additional Message Information

Parameters

<i>uMsg</i>	/< Handle For This Window
<i>wParam</i>	/< Message For This Window
<i>lParam</i>	/< Additional Message Information

Definition at line 118 of file supportcode.cpp.

```

122 {
123     switch (uMsg)
124     {
125         case WM_ACTIVATE:
126             {
127                 if (!HIWORD(wParam))
128                     //!/< Check For Windows Messages
129                     //!/< Watch For Window Activate Message
130                     //!/< Check Minimization State
131             }
132     }
133 }
```

```

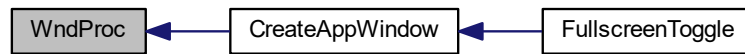
128         {
129             gActive=TRUE;                                //!< Program Is Active
130         }
131     else
132     {
133         gActive=FALSE;                                //!< Program Is No Longer Active
134     }
135
136     return 0;                                           //!< Return To The Message Loop
137 }
138
139 case WM_POWERBROADCAST:
140     if(wParam == PBT_APMRESUMESUSPEND)
141     {
142         CameraLibrary::CameraManager::X().PrepareForSuspend();
143     }
144     if(wParam == PBT_APMRESUMEAUTOMATIC)
145     {
146         CameraLibrary::CameraManager::X().ResumeFromSuspend();
147     }
148     break;
149
150 case WM_SYSCOMMAND:                                    //!< Intercept System Commands
151     {
152         switch (wParam)                                //!< Check System Calls
153         {
154             case SC_SCREENSAVE:                        //!< Screensaver Trying To Start?
155             case SC_MONITORPOWER:                      //!< Monitor Trying To Enter Powersave?
156                 return 0;                               //!< Prevent From Happening
157             }
158         break;                                           //!< Exit
159     }
160
161 case WM_CLOSE:                                         //!< Did We Receive A Close Message?
162     {
163         PostQuitMessage(0);                            //!< Send A Quit Message
164         return 0;                                       //!< Jump Back
165     }
166
167 case WM_KEYDOWN:                                       //!< Is A Key Being Held Down?
168     {
169         keys[wParam] = TRUE;                          //!< If So, Mark It As TRUE
170         return 0;                                       //!< Jump Back
171     }
172
173 case WM_KEYUP:                                         //!< Has A Key Been Released?
174     {
175         keys[wParam] = FALSE;                         //!< If So, Mark It As FALSE
176         return 0;                                       //!< Jump Back
177     }
178 case WM_MOVE:
179     return 0;                                           //!< Jump Back
180 case WM_PAINT:
181     return 0;
182 case WM_SIZE:                                         //!< Resize The OpenGL Window
183     {
184         ReSizeGLScene(LOWORD(lParam),HIWORD(lParam)); //!< LoWord=Width, HiWord=Height
185         return 0;                                       //!< Jump Back
186     }
187 }
188
189 //!< Pass All Unhandled Messages To DefWindowProc
190 return DefWindowProc(hWnd,uMsg,wParam,lParam);
191 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



6.11.2 Variable Documentation

6.11.2.1 `gActive`

```
bool gActive = TRUE
```

`/< Array Used For Scanning Keyboard`

Definition at line 26 of file `supportcode.cpp`.

6.11.2.2 `gFullscreen`

```
bool gFullscreen = FALSE
```

`/< Window Active Flag Set To TRUE By Default`

Definition at line 27 of file `supportcode.cpp`.

6.11.2.3 `gSoftwareDecimate`

```
int gSoftwareDecimate = 0
```

`/< Storage For One Texture (NEW)`

Definition at line 29 of file `supportcode.cpp`.

6.11.2.4 `gWindowHeight`

```
int gWindowHeight
```

Definition at line 19 of file `supportcode.cpp`.

6.11.2.5 gWindowWidth

```
int gWindowWidth
```

```

/*****
- /<== This is boiler-plate code for bringing up the application's window and /<== initializing OpenGL,
and an OpenGL surface class for rendering the camera /<== image as a quad using the 3D hardware.
/*****
-

```

```

/< Header File For Windows /< Header File For Standard Input/Output /< Header File For The OpenGL32 Library
/< Header File For The GLu32 Library

```

Definition at line 18 of file supportcode.cpp.

6.11.2.6 hDC

```
HDC hDC =NULL
```

Definition at line 24 of file supportcode.cpp.

6.11.2.7 hHook

```
HHOOK hHook = NULL
```

/<== Code to pop a simple dialog for 'waiting for cameras' using a message box and /<== no resources required for this sample application.

Definition at line 573 of file supportcode.cpp.

6.11.2.8 hInstance

```
HINSTANCE hInstance
```

/< Holds Our Window Handle

Definition at line 34 of file supportcode.cpp.

6.11.2.9 hRC

```
HGLRC hRC =NULL
```

/< Holds The Instance Of The Application

Definition at line 35 of file supportcode.cpp.

6.11.2.10 hWnd

```
HWND hWnd =NULL
```

/< Private GDI Device Context

Definition at line 33 of file supportcode.cpp.

6.11.2.11 keys

```
bool keys
```

/< Private GDI Device Context

Definition at line 25 of file supportcode.cpp.

6.11.2.12 texture

```
GLuint texture[1]
```

/< Fullscreen Flag Set To Fullscreen Mode By Default

Definition at line 28 of file supportcode.cpp.

6.11.2.13 windowHeight

```
int windowHeight
```

Definition at line 241 of file supportcode.cpp.

6.11.2.14 windowName

```
const char* windowName
```

Definition at line 242 of file supportcode.cpp.

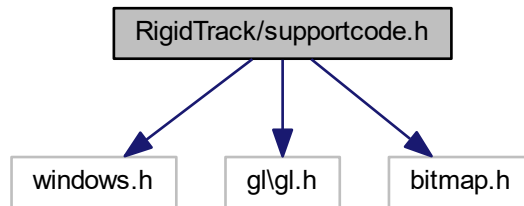
6.11.2.15 windowWidth

```
int windowWidth
```

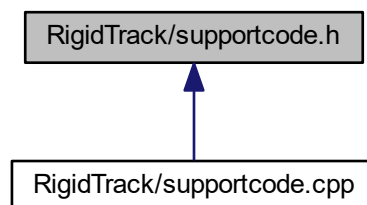
Definition at line 240 of file supportcode.cpp.

6.12 RigidTrack/supportcode.h File Reference

```
#include <windows.h>
#include <gl\gl.h>
#include "bitmap.h"
Include dependency graph for supportcode.h:
```



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



Classes

- class [Surface](#)

Macros

- #define [WIN32_LEAN_AND_MEAN](#)
- #define [BYTESPERPIXEL](#) 4
- #define [RGBA](#)(x, y, z, a) ((a<<24)|(x<<16)|(y<<8)|z)

Functions

- LRESULT CALLBACK [WndProc](#) (HWND, UINT, WPARAM, LPARAM)
/< Additional Message Information*
- BOOL [CreateAppWindow](#) (const char *title, int width, int height, int bits, bool fullscreenflag)
- GLvoid [CloseWindow](#) (GLvoid)
/< Properly Kill The Window*
- bool [PumpMessages](#) ()
- bool [FullscreenToggle](#) ()
- bool [PopWaitingDialog](#) ()
- int [DrawGLScene](#) (Surface *surf, int width, int height)

Variables

- HDC [hDC](#)
- bool [keys](#) [256]
- bool [gActive](#)
- bool [gFullscreen](#)

6.12.1 Macro Definition Documentation

6.12.1.1 BYTESPERPIXEL

```
#define BYTESPERPIXEL 4
```

Definition at line 17 of file supportcode.h.

6.12.1.2 RGBA

```
#define RGBA(  
    x,  
    y,  
    z,  
    a ) ((a<<24)|(x<<16)|(y<<8)|z)
```

Definition at line 18 of file supportcode.h.

6.12.1.3 WIN32_LEAN_AND_MEAN

```
#define WIN32_LEAN_AND_MEAN
```

Definition at line 3 of file supportcode.h.

6.12.2 Function Documentation

6.12.2.1 CloseWindow()

```
GLvoid CloseWindow (
    GLvoid )
```

/< Properly Kill The Window

Definition at line 192 of file supportcode.cpp.

```
193 {
194     if (gFullscreen)                                //!< Are We In Fullscreen Mode?
195     {
196         ChangeDisplaySettings(NULL,0);                //!< If So Switch Back To The Desktop
197         ShowCursor(TRUE);                            //!< Show Mouse Pointer
198     }
199
200     if (hRC)                                          //!< Do We Have A Rendering Context?
201     {
202         if (!wglMakeCurrent(NULL,NULL))              //!< Are We Able To Release The DC And RC
203             Contexts?
204             {
205                 MessageBox(NULL,L"Release Of DC And RC Failed.",L"SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
206             }
207         if (!wglDeleteContext(hRC))                  //!< Are We Able To Delete The RC?
208         {
209             MessageBox(NULL, L"Release Rendering Context Failed.", L"SHUTDOWN ERROR",MB_OK |
210             MB_ICONINFORMATION);
211         }
212         hRC=NULL;                                    //!< Set RC To NULL
213     }
214
215     if (hDC && !ReleaseDC(hWnd,hDC))                  //!< Are We Able To Release The DC
216     {
217         MessageBox(NULL, L"Release Device Context Failed.", L"SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
218         hDC=NULL;                                    //!< Set DC To NULL
219     }
220
221     if (hWnd && !DestroyWindow(hWnd))                  //!< Are We Able To Destroy The Window?
222     {
223         MessageBox(NULL, L"Could Not Release hWnd.", L"SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
224         hWnd=NULL;                                    //!< Set hWnd To NULL
225     }
226
227     if (!UnregisterClass(L"OpenGL",hInstance))          //!< Are We Able To Unregister Class
228     {
229         MessageBox(NULL, L"Could Not Unregister Class.", L"SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
230         hInstance=NULL;                                //!< Set hInstance To NULL
231     }
```

Here is the caller graph for this function:



6.12.2.2 CreateAppWindow()

```

BOOL CreateAppWindow (
    const char * title,
    int width,
    int height,
    int bits,
    bool fullscreenflag )

```

Definition at line 244 of file supportcode.cpp.

```

245 {
246     windowWidth = width;
247     windowHeight = height;
248     windowName = title;
249
250     GLuint PixelFormat;          ///< Holds The Results After Searching For A Match
251     WNDCLASS wc;                 ///< Windows Class Structure
252     DWORD dwExStyle;             ///< Window Extended Style
253     DWORD dwStyle;               ///< Window Style
254     RECT WindowRect;             ///< Grabs Rectangle Upper Left / Lower Right Values
255     WindowRect.left=(long)0;      ///< Set Left Value To 0
256     WindowRect.right=(long)width; ///< Set Right Value To Requested Width
257     WindowRect.top=(long)0;       ///< Set Top Value To 0
258     WindowRect.bottom=(long)height; ///< Set Bottom Value To Requested Height
259
260     gFullscreen=fullscreenflag;   ///< Set The Global Fullscreen Flag
261
262     HICON hIcon = (HICON) LoadImage(0, IDI_WINLOGO, IMAGE_ICON, 0, 0, LR_SHARED);
263
264     hInstance = GetModuleHandle(NULL);          ///< Grab An Instance For Our
Window
265     wc.style = CS_HREDRAW | CS_VREDRAW | CS_OWNDC;  ///< Redraw On Size, And Own DC For
Window.
266     wc.lpfnWndProc = (WNDPROC) WndProc;           ///< WndProc Handles Messages
267     wc.cbClsExtra = 0;                            ///< No Extra Window Data
268     wc.cbWndExtra = 0;                             ///< No Extra Window Data
269     wc.hInstance = hInstance;                      ///< Set The Instance
270     wc.hIcon = 0;
271     wc.hCursor = LoadCursor(NULL, IDC_ARROW);      ///< Load The Arrow Pointer
272     wc.hbrBackground = NULL;                      ///< No Background Required For GL
273     wc.lpszMenuName = NULL;                        ///< We Don't Want A Menu
274     wc.lpszClassName = L"OpenGL";                 ///< Set The Class Name
275
276     if (!RegisterClass(&wc))                      ///< Attempt To Register The Window Class
277     {
278         MessageBox(NULL, L"Failed To Register The Window Class.", L"ERROR", MB_OK|MB_ICONEXCLAMATION);
279         return FALSE;                               ///< Return FALSE
280     }
281
282     if (gFullscreen)                               ///< Attempt Fullscreen Mode?
283     {
284         DEVMODE dmScreenSettings;                  ///< Device Mode
285         memset(&dmScreenSettings, 0, sizeof(dmScreenSettings)); ///< Makes Sure Memory's Cleared
286         dmScreenSettings.dmSize=sizeof(dmScreenSettings); ///< Size Of The Devmode Structure
287         dmScreenSettings.dmPelsWidth = width;      ///< Selected Screen Width
288         dmScreenSettings.dmPelsHeight = height;    ///< Selected Screen Height
289         dmScreenSettings.dmBitsPerPel = bits;      ///< Selected Bits Per Pixel
290         dmScreenSettings.dmFields=DM_BITSPERPEL|DM_PELSWIDTH|DM_PELSHEIGHT;
291
292         ///< Try To Set Selected Mode And Get Results. NOTE: CDS_FULLSCREEN Gets Rid Of Start Bar.
293         if (ChangeDisplaySettings(&dmScreenSettings, CDS_FULLSCREEN) != DISP_CHANGE_SUCCESSFUL)
294         {
295             ///< If The Mode Fails, Offer Two Options. Quit Or Use Windowed Mode.
296             if (MessageBox(NULL, L"The Requested Fullscreen Mode Is Not Supported By\nYour Video Card. Use
Windowed Mode Instead?", L"NeHe GL", MB_YESNO|MB_ICONEXCLAMATION) == IDYES)
297             {
298                 gFullscreen=FALSE;                ///< Windowed Mode Selected. Fullscreen = FALSE
299             }
300             else
301             {
302                 ///< Pop Up A Message Box Letting User Know The Program Is Closing.
303                 MessageBox(NULL, L"Program Will Now Close.", L"ERROR", MB_OK|MB_ICONSTOP);
304                 return FALSE;                      ///< Return FALSE
305             }
306         }
307     }
308
309     if (gFullscreen)                               ///< Are We Still In
Fullscreen Mode?

```

```

310 {
311     dwExStyle=WS_EX_APPWINDOW;           ///< Window Extended Style
312     dwStyle=WS_POPUP;                    ///< Windows Style
313     ShowCursor(FALSE);                   ///< Hide Mouse Pointer
314 }
315 else
316 {
317     dwExStyle=WS_EX_APPWINDOW | WS_EX_WINDOWEDGE;           ///< Window Extended Style
318     dwStyle=WS_OVERLAPPEDWINDOW;                             ///< Windows Style
319 }
320
321 AdjustWindowRectEx(&WindowRect, dwStyle, FALSE, dwExStyle);    ///< Adjust Window To True Requested
Size
322
323 ///< Create The Window
324 if (!hWnd=CreateWindowEx( dwExStyle,           ///< Extended Style For The Window
325     L" penGL",                               ///< Class Name
326     L"title",                                ///< Window Title
327     dwStyle |                                ///< Defined Window Style
328     WS_CLIPSIBLINGS |                        ///< Required Window Style
329     WS_CLIPCHILDREN,                         ///< Required Window Style
330     0, 0,                                     ///< Window Position
331     WindowRect.right-WindowRect.left,        ///< Calculate Window Width
332     WindowRect.bottom-WindowRect.top,        ///< Calculate Window Height
333     NULL,                                     ///< No Parent Window
334     NULL,                                     ///< No Menu
335     hInstance,                               ///< Instance
336     NULL))                                   ///< Dont Pass Anything To WM_CREATE
337 {
338     CloseWindow();                           ///< Reset The Display
339     MessageBox(NULL,L" indow Creation Error.",L"Error", MB_OK|MB_ICONEXCLAMATION);
340     return FALSE;                            ///< Return FALSE
341 }
342
343 static PIXELFORMATDESCRIPTOR pfd=           ///< pfd Tells Windows How We Want Things To Be
344 {
345     sizeof(PIXELFORMATDESCRIPTOR),          ///< Size Of This Pixel Format Descriptor
346     1,                                       ///< Version Number
347     PFD_DRAW_TO_WINDOW |                   ///< Format Must Support Window
348     PFD_SUPPORT_OPENGL |                  ///< Format Must Support OpenGL
349     PFD_DOUBLEBUFFER,                     ///< Must Support Double Buffering
350     PFD_TYPE_RGBA,                         ///< Request An RGBA Format
351     bits,                                   ///< Select Our Color Depth
352     0, 0, 0, 0, 0, 0,                     ///< Color Bits Ignored
353     0,                                     ///< No Alpha Buffer
354     0,                                     ///< Shift Bit Ignored
355     0,                                     ///< No Accumulation Buffer
356     0, 0, 0, 0,                           ///< Accumulation Bits Ignored
357     16,                                    ///< 16Bit Z-Buffer (Depth Buffer)
358     0,                                     ///< No Stencil Buffer
359     0,                                     ///< No Auxiliary Buffer
360     PFD_MAIN_PLANE,                       ///< Main Drawing Layer
361     0,                                     ///< Reserved
362     0, 0, 0                               ///< Layer Masks Ignored
363 };
364
365 if (!hDC=GetDC(hWnd))                      ///< Did We Get A Device Context?
366 {
367     CloseWindow();                          ///< Reset The Display
368     MessageBox(NULL,L" an't Create A GL Device Context.",L" RROR",MB_OK|MB_ICONEXCLAMATION);
369     return FALSE;                           ///< Return FALSE
370 }
371
372 if (!(PixelFormat=ChoosePixelFormat(hDC,&pfd))    ///< Did Windows Find A Matching Pixel Format?
373 {
374     CloseWindow();                          ///< Reset The Display
375     MessageBox(NULL, L" an't Find A Suitable PixelFormat.", L" RROR",MB_OK|MB_ICONEXCLAMATION);
376     return FALSE;                           ///< Return FALSE
377 }
378
379 if(!SetPixelFormat(hDC,PixelFormat,&pfd))        ///< Are We Able To Set The Pixel Format?
380 {
381     CloseWindow();                          ///< Reset The Display
382     MessageBox(NULL, L" an't Set The PixelFormat.", L" RROR",MB_OK|MB_ICONEXCLAMATION);
383     return FALSE;                           ///< Return FALSE
384 }
385
386 if (!hRC=wglCreateContext(hDC))               ///< Are We Able To Get A Rendering Context?
387 {
388     CloseWindow();                          ///< Reset The Display
389     MessageBox(NULL, L" an't Create A GL Rendering Context.", L" RROR",MB_OK|MB_ICONEXCLAMATION);
390     return FALSE;                           ///< Return FALSE
391 }
392
393 if(!wglMakeCurrent(hDC,hRC))                  ///< Try To Activate The Rendering Context
394 {
395     CloseWindow();                          ///< Reset The Display

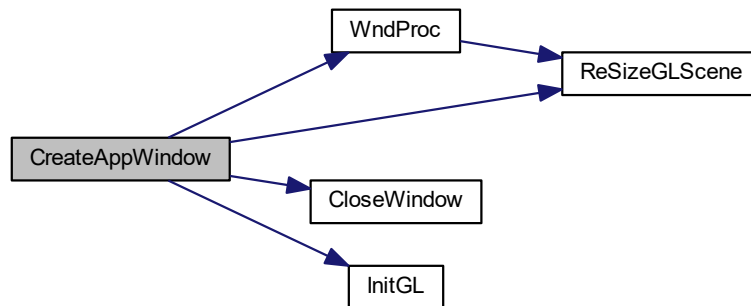
```

```

396     MessageBox(NULL, L" an't Activate The GL Rendering Context.", L" RROR",MB_OK|MB_ICONEXCLAMATION);
397     return FALSE;                                     //!< Return FALSE
398 }
399
400 ShowWindow(hWnd,SW_SHOW);                             //!< Show The Window
401 SetForegroundWindow(hWnd);                             //!< Slightly Higher Priority
402 SetFocus(hWnd);                                       //!< Sets Keyboard Focus To The Window
403 ReSizeGLScene(width, height);                         //!< Set Up Our Perspective GL Screen
404
405 if (!InitGL())                                       //!< Initialize Our Newly Created GL Window
406 {
407     CloseWindow();                                   //!< Reset The Display
408     MessageBox(NULL, L" nitialization Failed.", L" RROR",MB_OK|MB_ICONEXCLAMATION);
409     return FALSE;                                   //!< Return FALSE
410 }
411
412 return TRUE;                                         //!< Success
413 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



6.12.2.3 DrawGLScene()

```

int DrawGLScene (
    Surface * surf,
    int width,
    int height )

```

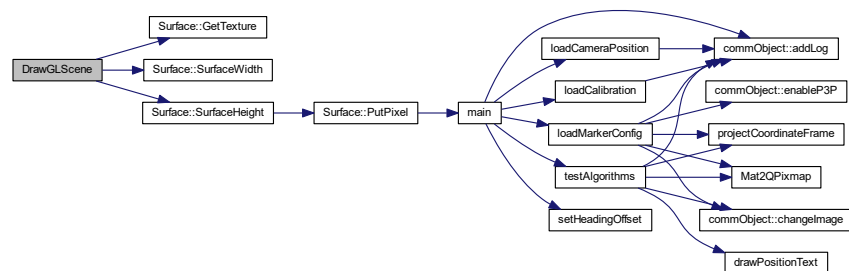
Definition at line 76 of file supportcode.cpp.


```

77 {
78     if (surf==NULL)
79         return true;
80
81     static bool frameThrottler = true;
82     frameThrottler=!frameThrottler;
83
84     if (frameThrottler) //!<<== Only display every other frame in case VSYNC is enabled. Otherwise
85         return true;    //!<<== application would get behind
86
87     int pixelWidth = width;
88     int pixelHeight= height;
89
90     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); //!<< Clear The Screen And The Depth Buffer
91     glLoadIdentity();                                   //!<< Reset The View
92
93     static GLuint ff = 0;
94
95     int tex = surf->GetTexture();
96     if (tex==0)
97         tex=ff;
98     else
99         ff=tex;
100    if (tex==0)
101        return true;
102
103    glBindTexture(GL_TEXTURE_2D, tex);
104
105    glBegin(GL_QUADS);
106        glTexCoord2f(0.0f, 0.0f); glVertex3f( 0,    0, 0);
107        glTexCoord2f((GLfloat)pixelWidth/(GLfloat)surf->SurfaceWidth(), 0.0f); glVertex3f( (
108        GLfloat)gWindowWidth, 0, 0);
109        glTexCoord2f((GLfloat)pixelWidth/(GLfloat)surf->SurfaceWidth(), (GLfloat)pixelHeight/(
110        GLfloat)surf->SurfaceHeight()); glVertex3f( (GLfloat)gWindowWidth, (GLfloat)
111        gWindowHeight, 0);
112        glTexCoord2f(0.0f, (GLfloat)pixelHeight/(GLfloat)surf->SurfaceHeight()); glVertex3f( 0
113        , (GLfloat) gWindowHeight, 0);
114    glEnd();
115
116    SwapBuffers (hDC);                                //!<< Swap Buffers (Double Buffering)
117
118    return true;                                       //!<< Keep Going
119 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



6.12.2.4 FullscreenToggle()

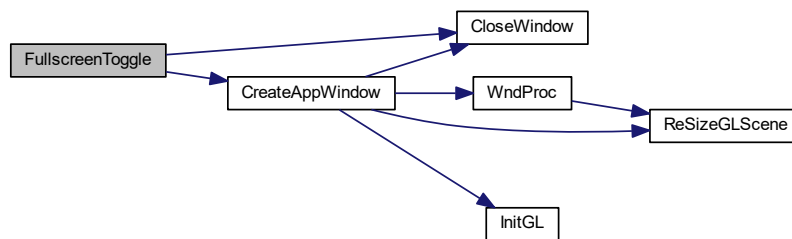
```
bool FullscreenToggle ( )
```

Definition at line 536 of file supportcode.cpp.

```

537 {
538     keys[VK_F1]=FALSE;                ///< If So Make Key FALSE
539     CloseWindow();                   ///< Kill Our Current Window
540     gFullscreen=!gFullscreen;        ///< Toggle Fullscreen / Windowed Mode
541
542     if (!CreateAppWindow(windowName,windowWidth,
543         windowHeight,32,gFullscreen))
544     {
545         MessageBox(0, L"Unable to toggle to full screen",L"Error", MB_OK);
546         return 1;
547     }
548     return true;
549 }
```

Here is the call graph for this function:



6.12.2.5 PopWaitingDialog()

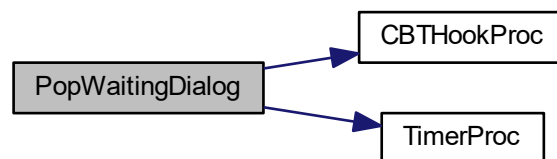
```
bool PopWaitingDialog ( )
```

Definition at line 615 of file supportcode.cpp.

```

616 {
617     ///<== hook in so we can create a message box that has only a 'Cancel' button ===
618
619     hHook = SetWindowsHookEx(WH_CBT, reinterpret_cast<HOOKPROC>(&
620         CBTHookProc), NULL, GetCurrentThreadId());
621
622     UINT_PTR nTimer = SetTimer(0, 100, 3000, (TIMERPROC) TimerProc);
623     int iResult = MessageBox( 0, L"waiting for connected cameras...", L"Camera Initialization", MB_OK );
624
625     if( iResult == IDOK )
626     {
627         ///<== user has clicked the cancel button ===
628         UnhookWindowsHookEx(hHook);
629         return false;
630     }
631     KillTimer(0, nTimer);
632     return true;
633 }
634 }
```

Here is the call graph for this function:



6.12.2.6 PumpMessages()

```
bool PumpMessages ( )
```

Definition at line 551 of file supportcode.cpp.

```

552 {
553     MSG msg;
554
555     if (PeekMessage (&msg, NULL, 0, 0, PM_REMOVE))    ////< Is There A Message Waiting?
556     {
557         if (msg.message==WM_QUIT)                    ////< Have We Received A Quit Message?
558             return false;
559         else                                          ////< If Not, Deal With Window Messages
560         {
561             TranslateMessage (&msg);                ////< Translate The Message
562             DispatchMessage (&msg);                ////< Dispatch The Message
563         }
564     }
565
566     return true;
567 }
```

6.12.2.7 WndProc()

```

LRESULT CALLBACK WndProc (
    HWND ,
    UINT ,
    WPARAM ,
    LPARAM )
```

/< Additional Message Information

Definition at line 118 of file supportcode.cpp.

```

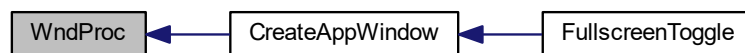
122 {
123     switch (uMsg)                                //!<< Check For Windows Messages
124     {
125         case WM_ACTIVATE:                        //!<< Watch For Window Activate Message
126         {
127             if (!HIWORD(wParam))                //!<< Check Minimization State
128             {
129                 gActive=TRUE;                    //!<< Program Is Active
130             }
131             else
132             {
133                 gActive=FALSE;                    //!<< Program Is No Longer Active
134             }
135         }
136         return 0;                                //!<< Return To The Message Loop
137     }
138
139     case WM_POWERBROADCAST:
140         if(wParam == PBT_APMSSUSPEND)
141         {
142             CameraLibrary::CameraManager::X().PrepareForSuspend();
143         }
144         if(wParam == PBT_APMRESUMEAUTOMATIC)
145         {
146             CameraLibrary::CameraManager::X().ResumeFromSuspend();
147         }
148         break;
149
150     case WM_SYSCOMMAND:                          //!<< Intercept System Commands
151     {
152         switch (wParam)                          //!<< Check System Calls
153         {
154             case SC_SCREENSAVE:                  //!<< Screensaver Trying To Start?
155             case SC_MONITORPOWER:                //!<< Monitor Trying To Enter Powersave?
156                 return 0;                        //!<< Prevent From Happening
157             }
158             break;                                //!<< Exit
159         }
160
161     case WM_CLOSE:                              //!<< Did We Receive A Close Message?
162     {
163         PostQuitMessage(0);                      //!<< Send A Quit Message
164         return 0;                                //!<< Jump Back
165     }
166
167     case WM_KEYDOWN:                            //!<< Is A Key Being Held Down?
168     {
169         keys[wParam] = TRUE;                     //!<< If So, Mark It As TRUE
170         return 0;                                //!<< Jump Back
171     }
172
173     case WM_KEYUP:                              //!<< Has A Key Been Released?
174     {
175         keys[wParam] = FALSE;                    //!<< If So, Mark It As FALSE
176         return 0;                                //!<< Jump Back
177     }
178     case WM_MOVE:                                //!<< Jump Back
179         return 0;
180     case WM_PAINT:                                //!<< Jump Back
181         return 0;
182     case WM_SIZE:                                //!<< Resize The OpenGL Window
183     {
184         ReSizeGLScene(LOWORD(lParam),HIWORD(lParam)); //!<< LoWord=Width, HiWord=Height
185         return 0;                                //!<< Jump Back
186     }
187 }
188
189 //!<< Pass All Unhandled Messages To DefWindowProc
190 return DefWindowProc(hWnd,uMsg,wParam,lParam);
191 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



6.12.3 Variable Documentation

6.12.3.1 gActive

```
bool gActive
```

6.12.3.2 gFullscreen

```
bool gFullscreen
```

6.12.3.3 hDC

```
HDC hDC
```

Definition at line 24 of file `supportcode.cpp`.

6.12.3.4 keys

```
bool keys[256]
```


Index

BACKBUFFER_BITSPERPIXEL

main.cpp, [40](#)

calcBoardCornerPositions

main.cpp, [15](#)

calibrateCamera

main.cpp, [16](#)

main.h, [54](#)

calibrateGround

main.cpp, [17](#)

main.h, [55](#)

camera_started

main.cpp, [40](#)

cameraMatrix

main.cpp, [41](#)

closeUDP

main.cpp, [19](#)

main.h, [57](#)

commObj

main.cpp, [41](#)

main.h, [74](#)

coordinateFrame

main.cpp, [41](#)

coordinateFrameProjected

main.cpp, [41](#)

currentMinIndex

main.cpp, [41](#)

currentPointDistance

main.cpp, [41](#)

data

main.cpp, [42](#)

datagram

main.cpp, [42](#)

Debug/moc_RigidTrack.cpp

QT_MOC_LITERAL, [7](#)

Debug/moc_communication.cpp

QT_MOC_LITERAL, [5](#)

determineExposure

main.cpp, [20](#)

main.h, [57](#)

determineOrder

main.cpp, [22](#)

main.h, [59](#)

distCoeffs

main.cpp, [42](#)

distModel

main.cpp, [42](#)

drawPositionText

main.cpp, [23](#)

main.h, [60](#)

eulerAngles

main.cpp, [42](#)

eulerRef

main.cpp, [42](#)

exitRequested

main.cpp, [43](#)

frameTime

main.cpp, [43](#)

GET_OPTIMIZED

precomp.hpp, [77](#)

getEulerAngles

main.cpp, [23](#)

gotOrder

main.cpp, [43](#)

headingOffset

main.cpp, [43](#)

IDI_ICON1

resource.h, [78](#)

IPAddressObject

main.cpp, [44](#)

main.h, [75](#)

IPAddressSafety

main.cpp, [44](#)

main.h, [75](#)

IPAddressSafety2

main.cpp, [44](#)

main.h, [75](#)

intExposure

main.cpp, [43](#)

intFrameRate

main.cpp, [43](#)

intIntensity

main.cpp, [44](#)

intThreshold

main.cpp, [44](#)

invertZ

main.cpp, [44](#)

main.h, [75](#)

list_points2d

main.cpp, [45](#)

list_points2dDifference

main.cpp, [45](#)

list_points2dOld

main.cpp, [45](#)

- list_points2dProjected
 - main.cpp, 45
- list_points2dUnsorted
 - main.cpp, 45
- list_points3d
 - main.cpp, 45
- loadCalibration
 - main.cpp, 24
 - main.h, 61
- loadCameraPosition
 - main.cpp, 25
 - main.h, 61
- loadMarkerConfig
 - main.cpp, 25
 - main.h, 62
- logDate
 - main.cpp, 46
- logFileName
 - main.cpp, 46
- logName
 - main.cpp, 46
- logfile
 - main.cpp, 46
- M_CN
 - main.cpp, 46
- M_HeadingOffset
 - main.cpp, 46
- main
 - main.cpp, 27
- main.cpp
 - BACKBUFFER_BITSPERPIXEL, 40
 - calcBoardCornerPositions, 15
 - calibrateCamera, 16
 - calibrateGround, 17
 - camera_started, 40
 - cameraMatrix, 41
 - closeUDP, 19
 - commObj, 41
 - coordinateFrame, 41
 - coordinateFrameProjected, 41
 - currentMinIndex, 41
 - currentPointDistance, 41
 - data, 42
 - datagram, 42
 - determineExposure, 20
 - determineOrder, 22
 - distCoeffs, 42
 - distModel, 42
 - drawPositionText, 23
 - eulerAngles, 42
 - eulerRef, 42
 - exitRequested, 43
 - frameTime, 43
 - getEulerAngles, 23
 - gotOrder, 43
 - headingOffset, 43
 - IPAdressObject, 44
 - IPAdressSafety, 44
 - IPAdressSafety2, 44
 - intExposure, 43
 - intFrameRate, 43
 - intIntensity, 44
 - intThreshold, 44
 - invertZ, 44
 - list_points2d, 45
 - list_points2dDifference, 45
 - list_points2dOld, 45
 - list_points2dProjected, 45
 - list_points2dUnsorted, 45
 - list_points3d, 45
 - loadCalibration, 24
 - loadCameraPosition, 25
 - loadMarkerConfig, 25
 - logDate, 46
 - logFileName, 46
 - logName, 46
 - logfile, 46
 - M_CN, 46
 - M_HeadingOffset, 46
 - main, 27
 - Mat2QPixmap, 28
 - methodPNP, 47
 - minPointDistance, 47
 - numberMarkers, 47
 - pointOrderIndices, 47
 - pointOrderIndicesNew, 47
 - portObject, 47
 - portSafety, 48
 - portSafety2, 48
 - posRef, 48
 - position, 48
 - positionOld, 48
 - projectCoordinateFrame, 29
 - Rmat, 48
 - RmatRef, 49
 - Rvec, 49
 - RvecOriginal, 49
 - safety2Enable, 49
 - safetyAngle, 49
 - safetyBoxLength, 49
 - safetyEnable, 50
 - sendDataUDP, 29
 - setHeadingOffset, 30
 - setReference, 30
 - setUpUDP, 32
 - ss, 50
 - startStopCamera, 33
 - startTracking, 34
 - strBuf, 50
 - testAlgorithms, 39
 - timeFirstFrame, 50
 - timeOld, 50
 - Tvec, 50
 - TvecOriginal, 51
 - udpSocketObject, 51
 - udpSocketSafety, 51

- udpSocketSafety2, 51
- useGuess, 51
- velocity, 51
- main.h
 - calibrateCamera, 54
 - calibrateGround, 55
 - closeUDP, 57
 - commObj, 74
 - determineExposure, 57
 - determineOrder, 59
 - drawPositionText, 60
 - IPAdressObject, 75
 - IPAdressSafety, 75
 - IPAdressSafety2, 75
 - invertZ, 75
 - loadCalibration, 61
 - loadCameraPosition, 61
 - loadMarkerConfig, 62
 - methodPNP, 75
 - portObject, 75
 - portSafety, 76
 - portSafety2, 76
 - projectCoordinateFrame, 63
 - safety2Enable, 76
 - safetyAngle, 76
 - safetyBoxLength, 76
 - safetyEnable, 76
 - sendDataUDP, 64
 - setHeadingOffset, 64
 - setReference, 65
 - setUpUDP, 67
 - startStopCamera, 67
 - startTracking, 68
 - testAlgorithms, 74
- Mat2QPixmap
 - main.cpp, 28
- methodPNP
 - main.cpp, 47
 - main.h, 75
- minPointDistance
 - main.cpp, 47
- numberMarkers
 - main.cpp, 47
- pointOrderIndices
 - main.cpp, 47
- pointOrderIndicesNew
 - main.cpp, 47
- portObject
 - main.cpp, 47
 - main.h, 75
- portSafety
 - main.cpp, 48
 - main.h, 76
- portSafety2
 - main.cpp, 48
 - main.h, 76
- posRef
 - main.cpp, 48
- position
 - main.cpp, 48
- positionOld
 - main.cpp, 48
- precomp.hpp
 - GET_OPTIMIZED, 77
- projectCoordinateFrame
 - main.cpp, 29
 - main.h, 63
- qCleanupResources_RigidTrack
 - qrc_RigidTrack.cpp, 9
- qInitResources_RigidTrack
 - qrc_RigidTrack.cpp, 9
- QT_MOC_LITERAL
 - Debug/moc_RigidTrack.cpp, 7
 - Debug/moc_communication.cpp, 5
 - Release/moc_RigidTrack.cpp, 8
 - Release/moc_communication.cpp, 6
- QT_RCC_MANGLE_NAMESPACE
 - qrc_RigidTrack.cpp, 8
- QT_RCC_PREPEND_NAMESPACE
 - qrc_RigidTrack.cpp, 8
- qrc_RigidTrack.cpp
 - qCleanupResources_RigidTrack, 9
 - qInitResources_RigidTrack, 9
 - QT_RCC_MANGLE_NAMESPACE, 8
 - QT_RCC_PREPEND_NAMESPACE, 8
- Release/moc_RigidTrack.cpp
 - QT_MOC_LITERAL, 8
- Release/moc_communication.cpp
 - QT_MOC_LITERAL, 6
- resource.h
 - IDI_ICON1, 78
- RigidTrack/_modelest.h, 3
- RigidTrack/GeneratedFiles/Debug/moc_RigidTrack.cpp, 7
- RigidTrack/GeneratedFiles/Debug/moc_communication.↔
cpp, 5
- RigidTrack/GeneratedFiles/Release/moc_RigidTrack.↔
cpp, 7
- RigidTrack/GeneratedFiles/Release/moc_communication.↔
cpp, 6
- RigidTrack/GeneratedFiles/qrc_RigidTrack.cpp, 8
- RigidTrack/GeneratedFiles/ui_RigidTrack.h, 10
- RigidTrack/RigidTrack.cpp, 78
- RigidTrack/RigidTrack.h, 79
- RigidTrack/communication.cpp, 3
- RigidTrack/communication.h, 4
- RigidTrack/main.cpp, 11
- RigidTrack/main.h, 52
- RigidTrack/precomp.hpp, 77
- RigidTrack/resource.h, 78
- Rmat
 - main.cpp, 48
- RmatRef
 - main.cpp, 49

Rvec
 main.cpp, 49

RvecOriginal
 main.cpp, 49

safety2Enable
 main.cpp, 49
 main.h, 76

safetyAngle
 main.cpp, 49
 main.h, 76

safetyBoxLength
 main.cpp, 49
 main.h, 76

safetyEnable
 main.cpp, 50
 main.h, 76

sendDataUDP
 main.cpp, 29
 main.h, 64

setHeadingOffset
 main.cpp, 30
 main.h, 64

setReference
 main.cpp, 30
 main.h, 65

setUpUDP
 main.cpp, 32
 main.h, 67

ss
 main.cpp, 50

startStopCamera
 main.cpp, 33
 main.h, 67

startTracking
 main.cpp, 34
 main.h, 68

strBuf
 main.cpp, 50

testAlgorithms
 main.cpp, 39
 main.h, 74

timeFirstFrame
 main.cpp, 50

timeOld
 main.cpp, 50

Tvec
 main.cpp, 50

TvecOriginal
 main.cpp, 51

udpSocketObject
 main.cpp, 51

udpSocketSafety
 main.cpp, 51

udpSocketSafety2
 main.cpp, 51

useGuess
 main.cpp, 51

velocity
 main.cpp, 51