Rigid Track

Generated by Doxygen 1.8.13

Contents

Chapter 1

Rigid Track Doxygen Documentation

1.1 Introduction

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

1.2 Rigid Track Installation

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

1.3 Source Code

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

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

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

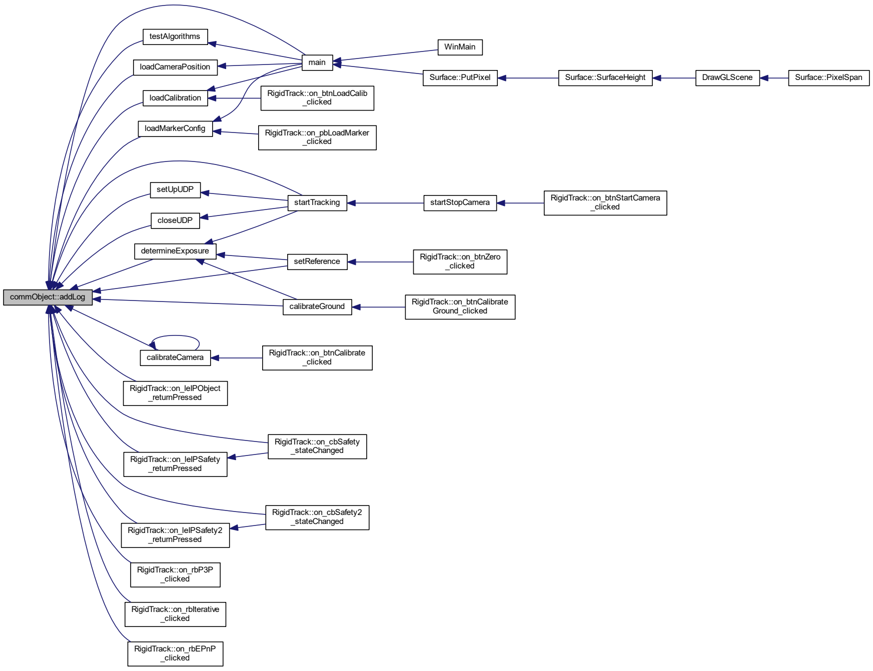
5.1.2.1 addLog()

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

Definition at line 20 of file communication.cpp.

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

Here is the caller graph for this function:


testAlgorithms
loadCameraPosition
loadCalibration
loadMarkerConfig
main
WinMain
Surface::PutPixel
Surface::SurfaceHeight
DrawGLScene
Surface::PixelSpan
RigidTrack::on_btnLoadCalib _clicked
RigidTrack::on_pbLoadMarker _clicked
setUpUDP
closeUDP
determineExposure
startTracking
startStopCamera
RigidTrack::on_btnStartCamera _clicked
setReference
RigidTrack::on_btnZero _clicked
commObject::addLog
calibrateGround
RigidTrack::on_btnCalibrate Ground_clicked
calibrateCamera
RigidTrack::on_btnCalibrate _clicked
RigidTrack::on_leIPObject _returnPressed
RigidTrack::on_cbSafety _stateChanged
RigidTrack::on_leIPSafety _returnPressed
RigidTrack::on_cbSafety2 _stateChanged
RigidTrack::on_leIPSafety2 _returnPressed
RigidTrack::on_rbP3P _clicked
RigidTrack::on_rbIterative _clicked
RigidTrack::on_rbEPnP _clicked

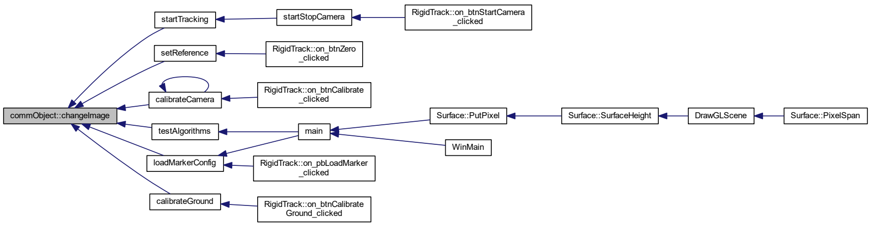
5.1.2.2 changeImage()

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

Definition at line 14 of file communication.cpp.

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

Here is the caller graph for this function:



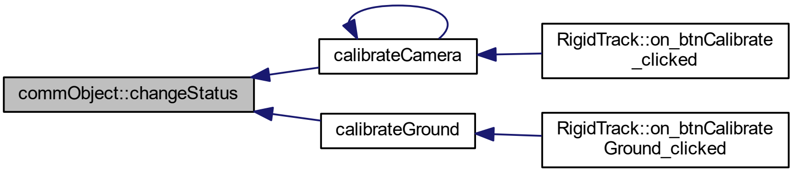
5.1.2.3 changeStatus()

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

Definition at line 8 of file communication.cpp.

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

Here is the caller graph for this function:


commObject::changeStatus
calibrateCamera
RigidTrack::on_btnCalibrate_clicked
calibrateGround
RigidTrack::on_btnCalibrateGround_clicked

5.1.2.4 clearLog()

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

Definition at line 26 of file communication.cpp.

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

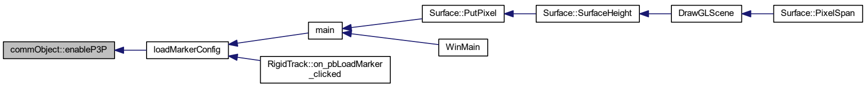
5.1.2.5 enableP3P()

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

Definition at line 32 of file communication.cpp.

```
{
    emit P3Penabled(value);
    QCoreApplication::processEvents();
}
```

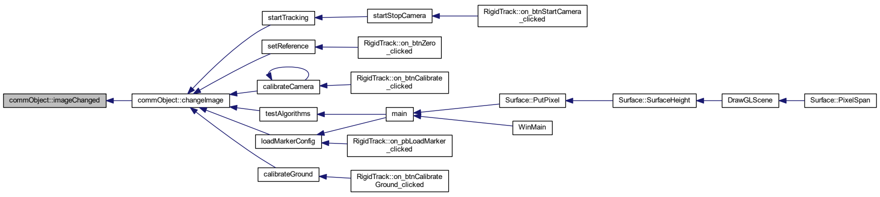
Here is the caller graph for this function:



5.1.2.6 imageChanged

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

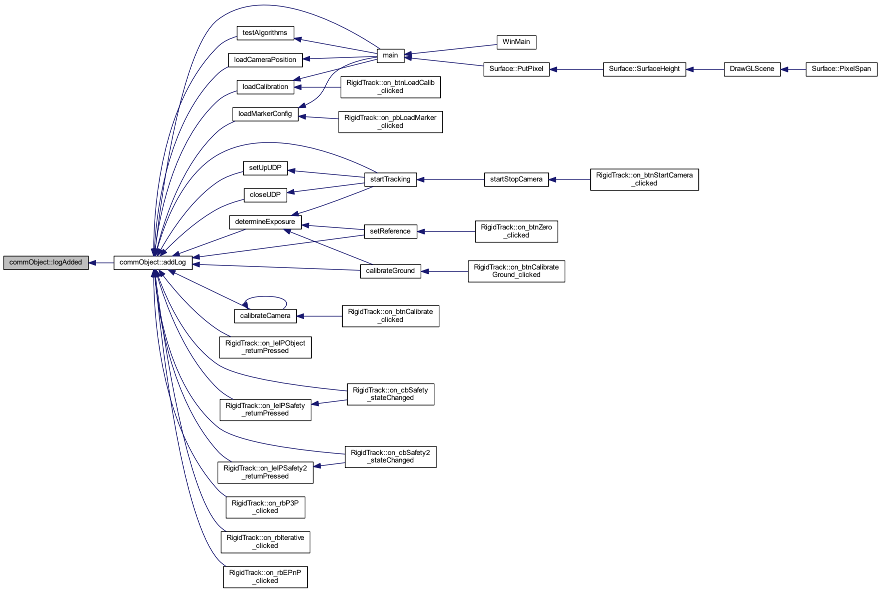
Here is the caller graph for this function:



5.1.2.7 logAdded

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

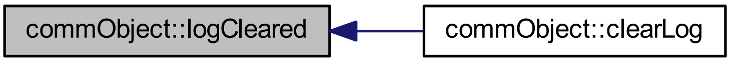
Here is the caller graph for this function:


testAlgorithms
main
WinMain
loadCameraPosition
Surface::PutPixel
Surface::SurfaceHeight
DrawGLScene
Surface::PixelSpan
loadCalibration
RigidTrack::on_btnLoadCalib _clicked
loadMarkerConfig
RigidTrack::on_pbLoadMarker _clicked
setUpUDP
startTracking
startStopCamera
RigidTrack::on_btnStartCamera _clicked
closeUDP
determineExposure
setReference
RigidTrack::on_btnZero _clicked
commObject::logAdded
commObject::addLog
calibrateGround
RigidTrack::on_btnCalibrate Ground_clicked
calibrateCamera
RigidTrack::on_btnCalibrate _clicked
RigidTrack::on_leIPObject _returnPressed
RigidTrack::on_cbSafety _stateChanged
RigidTrack::on_leIPSafety _returnPressed
RigidTrack::on_cbSafety2 _stateChanged
RigidTrack::on_leIPSafety2 _returnPressed
RigidTrack::on_rbP3P _clicked
RigidTrack::on_rbIterative _clicked
RigidTrack::on_rbEPnP _clicked

5.1.2.8 logCleared

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

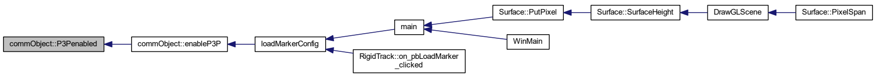
Here is the caller graph for this function:



5.1.2.9 P3Penabled

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

Here is the caller graph for this function:



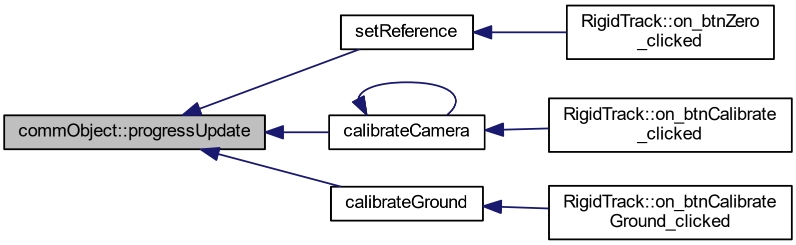
5.1.2.10 progressUpdate()

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

Definition at line 38 of file communication.cpp.

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

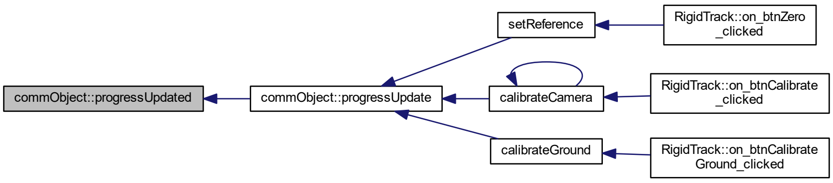
Here is the caller graph for this function:


setReference
RigidTrack::on_btnZero _clicked
commObject::progressUpdate
calibrateCamera
RigidTrack::on_btnCalibrate _clicked
calibrateGround
RigidTrack::on_btnCalibrate Ground_clicked

5.1.2.11 progressUpdated

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

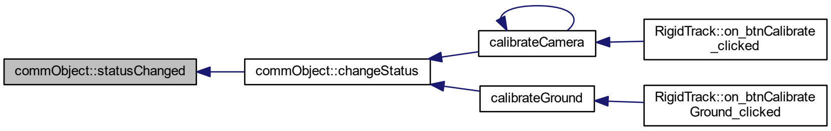
Here is the caller graph for this function:


setReference
RigidTrack::on_btnZero _clicked
commObject::progressUpdated
commObject::progressUpdate
calibrateCamera
RigidTrack::on_btnCalibrate _clicked
calibrateGround
RigidTrack::on_btnCalibrate Ground_clicked

5.1.2.12 statusChanged

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

Here is the caller graph for this function:


commObject::statusChanged
commObject::changeStatus
calibrateCamera
calibrateGround
RigidTrack::on_btnCalibrate_clicked
RigidTrack::on_btnCalibrateGround_clicked

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 max←Attempts=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 _modelest.h.

```
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_rbIterative_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.

```
      : QMainWindow(parent)
{
      ui.setupUi(this);

}
```

5.3.3 Member Function Documentation

5.3.3.1 clearLog

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

Definition at line 42 of file RigidTrack.cpp.

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

5.3.3.2 enableP3P

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

Definition at line 229 of file RigidTrack.cpp.

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

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.

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

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.

```
{
    QString command = "explorer.exe " + QDir::currentPath().replace("/", "\\");
    QProcess::startDetached(command);

}
```

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.

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

5.3.3.6 on_actionShow_Help_triggered

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

Definition at line 160 of file RigidTrack.cpp.

```
{

    //!/< append help.pdf to the path since this is the documentation in html format
    QString qtStrFile = QDir::currentPath().replace("/", "\\") + "\\help.pdf";

    //!/< open the documentation help file in the standard browser
    QDesktopServices::openUrl(QUrl::fromLocalFile(qtStrFile));
}
```

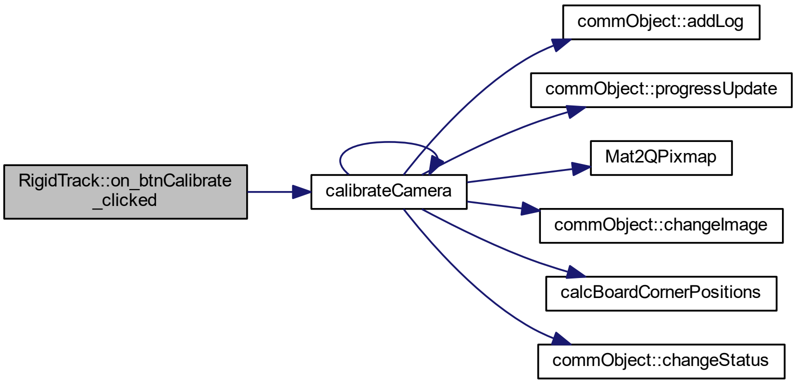
5.3.3.7 on_btnCalibrate_clicked

```
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:



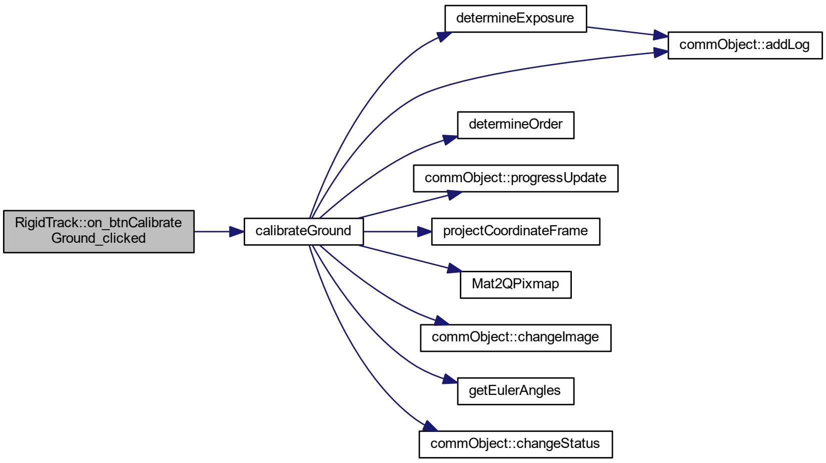
5.3.3.8 on_btnCalibrateGround_clicked

```
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:



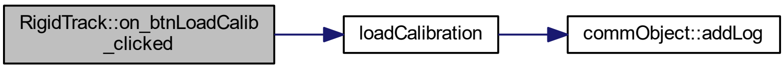
5.3.3.9 on_btnLoadCalib_clicked

```
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:



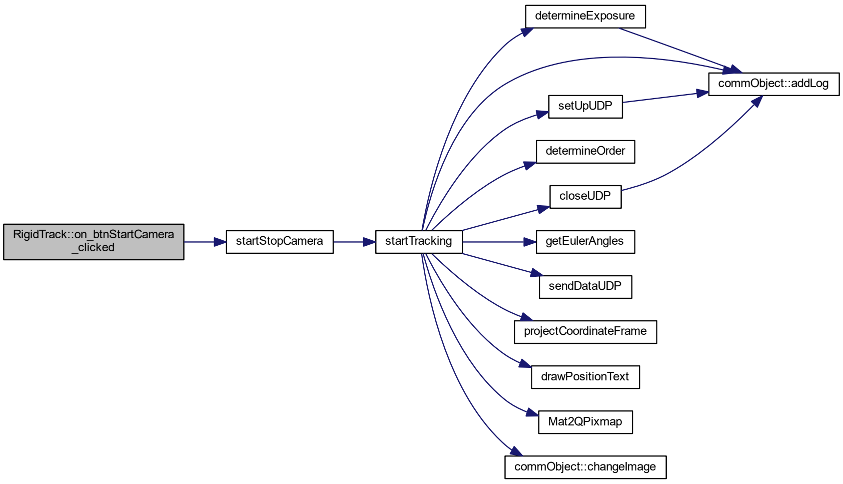
5.3.3.10 on_btnStartCamera_clicked

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

Definition at line 260 of file RigidTrack.cpp.

```
{
    if(RigidTrack::ui.btnStartCamera->text() == "Start Tracking")
    {
        RigidTrack::ui.btnStartCamera->setText("Stop Tracking");
    }
    else
    {
        RigidTrack::ui.btnStartCamera->setText("Start Tracking");
    }
    startStopCamera();
}
```

Here is the call graph for this function:



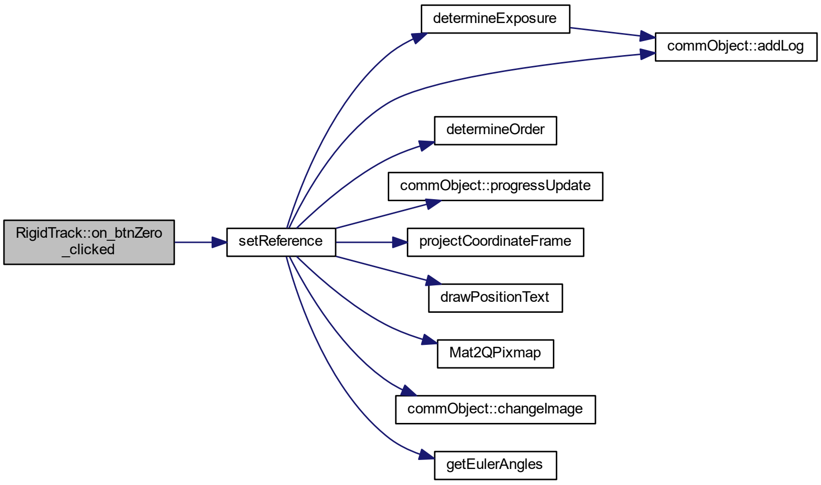
5.3.3.11 on_btnZero_clicked

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

Definition at line 27 of file RigidTrack.cpp.

```
{
    setReference();
}
```

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.

```
{
    if (state)
    {
        invertZ = -1;
    }
    else
    {
        invertZ = 1;
    }
}
```

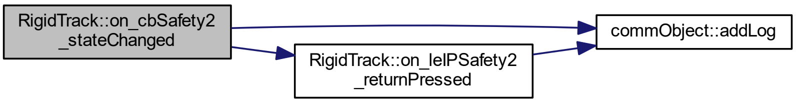
5.3.3.13 on_cbSafety2_stateChanged

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

Definition at line 187 of file RigidTrack.cpp.

```
{
    RigidTrack::ui.leIPSafety2->setEnabled(state);
    safety2Enable = state;
    if (state)
    {
        commObj.addLog("Enabled second Receiver");
        on_leIPSafety2_returnPressed();
    }
    else
    {
        commObj.addLog("Disabled second Receiver");
    }
}
```

Here is the call graph for this function:


RigidTrack::on_cbSafety2_stateChanged
RigidTrack::on_leIPSafety2_returnPressed
commObject::addLog

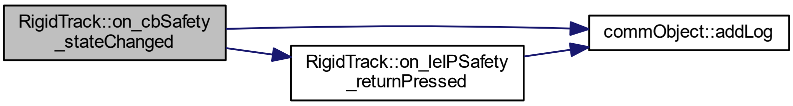
5.3.3.14 on_cbSafety_stateChanged

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

Definition at line 170 of file RigidTrack.cpp.

```
{
    RigidTrack::ui.dsbDimension->setEnabled(state);
    RigidTrack::ui.sbAngle->setEnabled(state);
    safetyEnable = state;
    RigidTrack::ui.leIPSafety->setEnabled(state);
    if (state)
    {
        commObj.addLog("Enabled Safety Area Protection");
        on_leIPSafety_returnPressed();
    }
    else
    {
        commObj.addLog("Disabled Safety Area Protection");
    }
}
```

Here is the call graph for this function:


RigidTrack::on_cbSafety_stateChanged
RigidTrack::on_leIPSafety_returnPressed
commObject::addLog

5.3.3.15 on_dsbDimension_valueChanged

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

Definition at line 202 of file RigidTrack.cpp.

```
{
    safetyBoxLength = d;
}
```

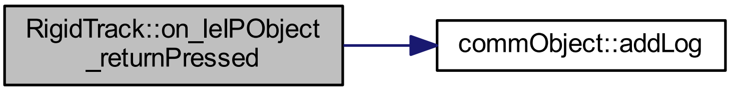
5.3.3.16 on_leIPObject_returnPressed

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

Definition at line 68 of file RigidTrack.cpp.

```
{
    try{
    QString adress = RigidTrack::ui.leIPObject->text();
    IPAdressObject = QHostAddress(adress.split(":")[0]);
    if (IPAdressObject.isNull() || adress.split(":").length() == 1 || adress.split(":")[1]==0)
    {
        throw 2;
    }
    portObject = adress.split(":")[1].toInt();
    commObj.addLog("Object IP changed to:");
    commObj.addLog(IPAdressObject.toString());
    commObj.addLog("Object Port changed to:");
    commObj.addLog(QString::number(portObject));
    }
    catch (...)
    {
        commObj.addLog("Error Changing the IP Adress or Port! Restored Standard Values 192.168.0.1:9155");
        IPAdressObject = QHostAddress("192.168.0.1");
        portObject = 9155;
        RigidTrack::ui.leIPObject->setText("192.168.0.1:9155");
    }
}
```

Here is the call graph for this function:



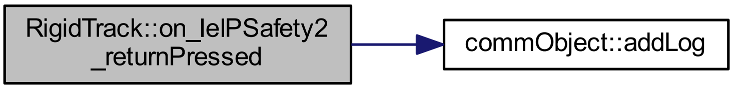
5.3.3.17 on_leIPSafety2_returnPressed

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

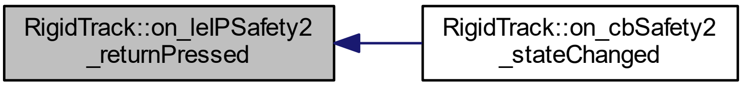
Definition at line 117 of file RigidTrack.cpp.

```
118 {
119     try {
120     QString adress = RigidTrack::ui.leIPSafety2->text();
121     IPAdressSafety2 = QHostAddress(adress.split(":")[0]);
122     if (IPAdressSafety2.isNull() || adress.split(":").length() == 1 || adress.split(":")[1]
      == 0)
123     {
124         throw 2;
125     }
126     portSafety2 = adress.split(":")[1].toInt();
127     commObj.addLog("Receiver 2 IP changed to:");
128     commObj.addLog(IPAdressSafety2.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 Adress or Port! Restored Standard Values
       192.168.0.1:9155");
135         IPAdressSafety2 = 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:



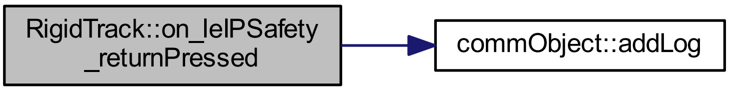
5.3.3.18 on_leIPSafety_returnPressed

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

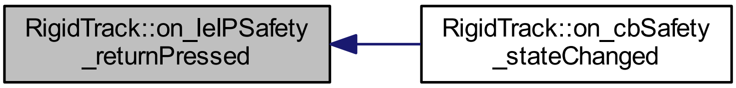
Definition at line 93 of file RigidTrack.cpp.

```
{
    try{
    QString adress = RigidTrack::ui.leIPSafety->text();
    IPAdressSafety = QHostAddress(adress.split(":")[0]);
    if (IPAdressSafety.isNull() || adress.split(":").length() == 1 || adress.split(":")[1] ==
    0)
    {
         throw 2;
     }
     portSafety = adress.split(":")[1].toInt();
     commObj.addLog("Safety Switch IP changed to:");
     commObj.addLog(IPAdressSafety.toString());
     commObj.addLog("Safety Switch Port changed to:");
     commObj.addLog(QString::number(portSafety));
     }
     catch (...)
     {
         commObj.addLog("Error Changing the IP Adress or Port! Restored Standard Values
    192.168.0.1:9155");
         IPAdressSafety = QHostAddress("192.168.0.1");
         portSafety = 9155;
         RigidTrack::ui.leIPSafety->setText("192.168.0.1:9155");
     }
     }
```

Here is the call graph for this function:



Here is the caller graph for this function:



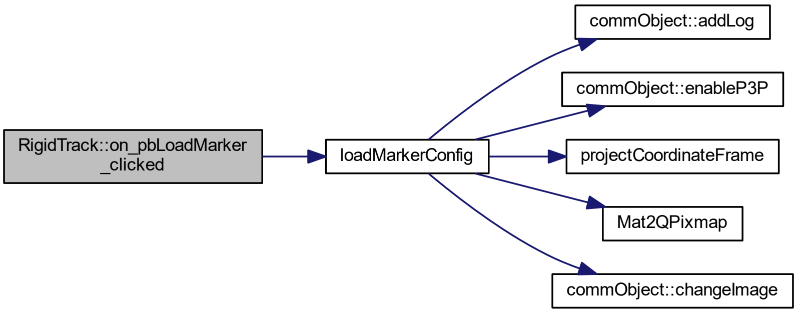
5.3.3.19 on_pbLoadMarker_clicked

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

Definition at line 212 of file RigidTrack.cpp.

```
{
    loadMarkerConfig(1);
}
```

Here is the call graph for this function:



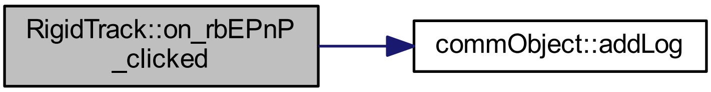
5.3.3.20 on_rbEPnP_clicked

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

Definition at line 154 of file RigidTrack.cpp.

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

Here is the call graph for this function:



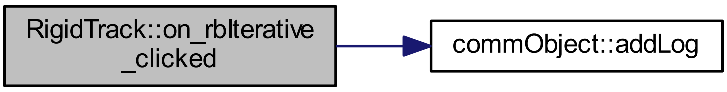
5.3.3.21 on_rbIterative_clicked

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

Definition at line 148 of file RigidTrack.cpp.

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

Here is the call graph for this function:



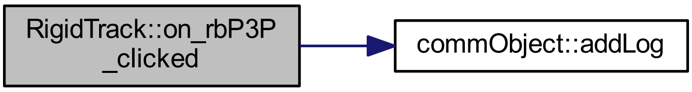
5.3.3.22 on_rbP3P_clicked

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

Definition at line 142 of file RigidTrack.cpp.

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

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.

```
{
    safetyAngle = i;
}
```

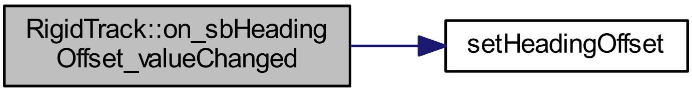
5.3.3.24 on_sbHeadingOffset_valueChanged

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

Definition at line 63 of file RigidTrack.cpp.

```
{
    setHeadingOffset(d);
}
```

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.

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

5.3.3.26 setImage

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

Definition at line 37 of file RigidTrack.cpp.

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

5.3.3.27 setLog

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

Definition at line 57 of file RigidTrack.cpp.

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

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()

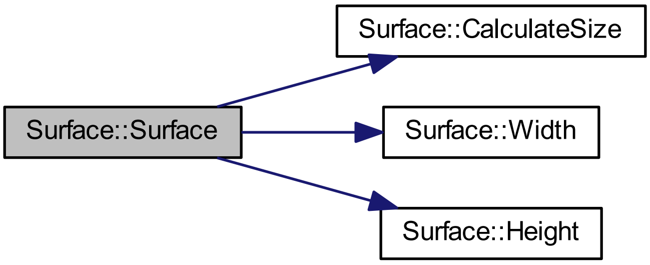
```
Surface::Surface (
              int Width,
              int Height )
```

/<==============================

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 *
      BYTESPERPIXEL);
430 
431     memset(buffer, 0, mSurfaceWidth * mSurfaceHeight * BYTESPERPIXEL);
432 
433     if(buffer==0)
434         throw(" nable to allocate surface buffer");
435 
436     glGenTextures(1, &mTexture);
437     if(mTexture==0)
438         throw(" nable to gen OpenGL texture");
439 
440 
441     glBindTexture(GL_TEXTURE_2D, mTexture);
442     glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8, mSurfaceWidth, mSurfaceHeight, 0, GL_RGBA, GL_UNSIGNED_BYTE,
      buffer);
443     glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
444     glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
445 }
```

Here is the call graph for this function:



5.4.2.2 ∼Surface()

```
Surface::∼Surface ( )
```

Definition at line 447 of file supportcode.cpp.

```
{

}
```

5.4.3 Member Function Documentation

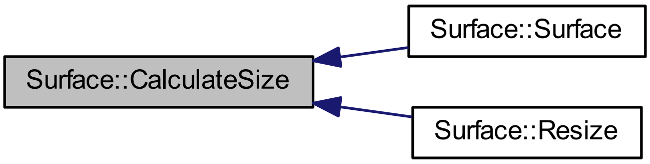
5.4.3.1 CalculateSize()

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

Definition at line 494 of file supportcode.cpp.

```
{
    int mSize = 1;

    while(mSize<Width)
        mSize*=2;

    return mSize;
}
```

Here is the caller graph for this function:



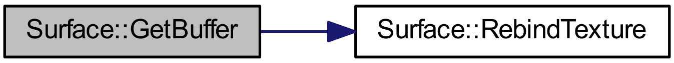
5.4.3.2 GetBuffer()

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

Definition at line 40 of file supportcode.h.

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

Here is the call graph for this function:



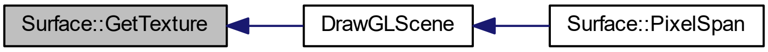
5.4.3.3 GetTexture()

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

Definition at line 504 of file supportcode.cpp.

```
505 {
506     //!/<if(mDirty==true)
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:



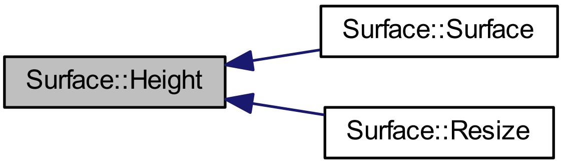
5.4.3.4 Height()

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

Definition at line 35 of file supportcode.h.

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

Here is the caller graph for this function:



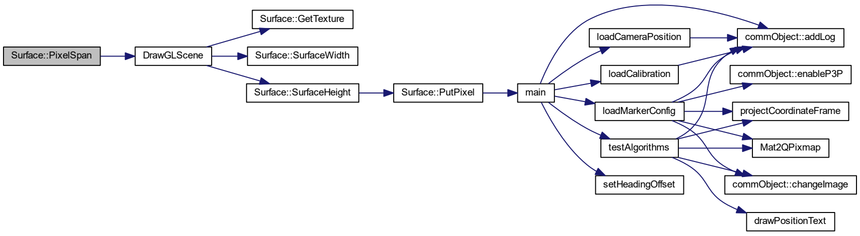
5.4.3.5 PixelSpan()

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

Definition at line 42 of file supportcode.h.

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

Here is the call graph for this function:



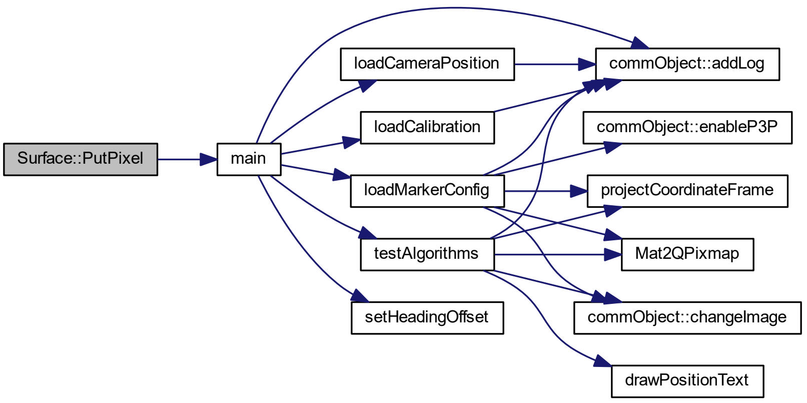
5.4.3.6 PutPixel()

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

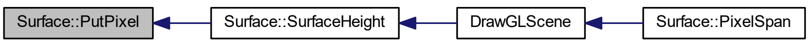
Definition at line 514 of file supportcode.cpp.

```
{
    if(X>=0 && Y>=0 && X<mWidth && Y<mHeight)
    {
        unsigned int *point = (unsigned int*)buffer + (Y*mSpan)+X;
        *point = Color;
        mDirty = true;
    }
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



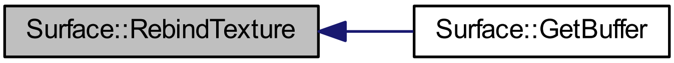
5.4.3.7 RebindTexture()

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

Definition at line 452 of file supportcode.cpp.

```
{
    glGenTextures(1, &mTexture);
    if(mTexture==0)
        throw(" nable to gen OpenGL texture");


    glBindTexture(GL_TEXTURE_2D, mTexture);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8, mSurfaceWidth, mSurfaceHeight, 0, GL_RGBA, GL_UNSIGNED_BYTE, buffer);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
}
```

Here is the caller graph for this function:



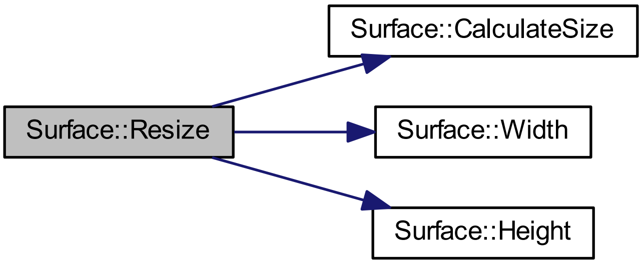
5.4.3.8 Resize()

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

Definition at line 466 of file supportcode.cpp.

```
{
    if(Width==mWidth && Height==mHeight)
        return;

    if(Width<1 || Height<1)
        return;

    int newSize = CalculateSize(Width);

    if(newSize>mSurfaceWidth || newSize>mSurfaceHeight)
    {
        if(buffer!=0)
            free(buffer);

        mSurfaceWidth = newSize;
        mSurfaceHeight = newSize;
        mSpan = mSurfaceWidth;
        mWidth = Width;
        mHeight = Height;
        buffer = (unsigned char*) malloc(mSurfaceWidth * mSurfaceHeight * BYTESPERPIXEL);
        if(buffer==0)
            throw("Unable to allocate surface buffer");

        mDirty = true;
    }
}
```

Here is the call graph for this function:



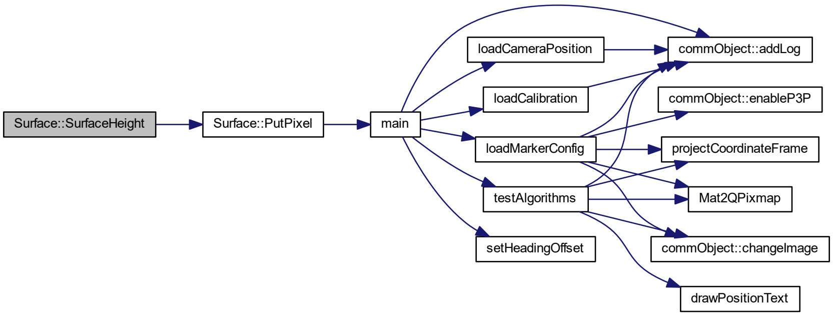
5.4.3.9 SurfaceHeight()

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

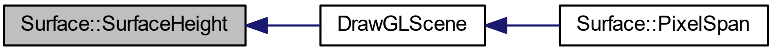
Definition at line 37 of file supportcode.h.

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

Here is the call graph for this function:



Here is the caller graph for this function:



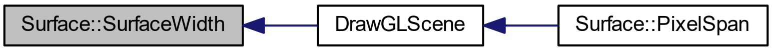
5.4.3.10 SurfaceWidth()

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

Definition at line 36 of file supportcode.h.

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

Here is the caller graph for this function:



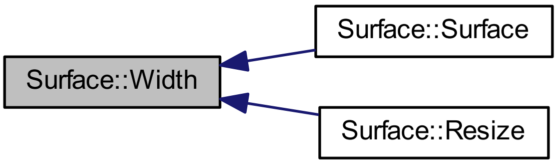
5.4.3.11 Width()

```
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

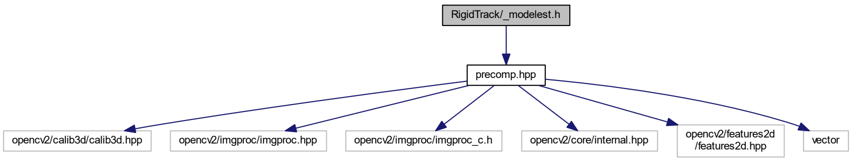
Chapter 6

File Documentation

6.1 RigidTrack/_modelest.h File Reference

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

Include dependency graph for _modelest.h:


RigidTrack/_modelest.h
precomp.hpp
opencv2/calib3d/calib3d.hpp
opencv2/imgproc/imgproc.hpp
opencv2/imgproc/imgproc_c.h
opencv2/core/internal.hpp
opencv2/features2d
/features2d.hpp
vector

Classes

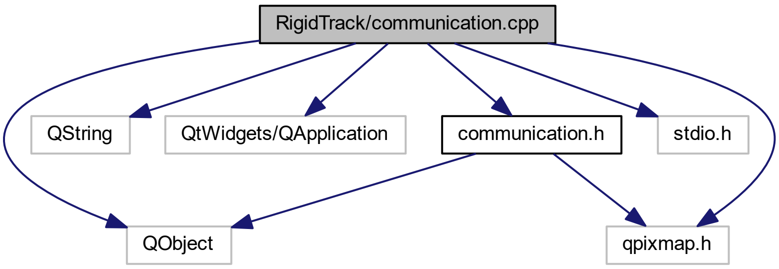
- class CvModelEstimator2

6.2 RigidTrack/communication.cpp File Reference

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

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

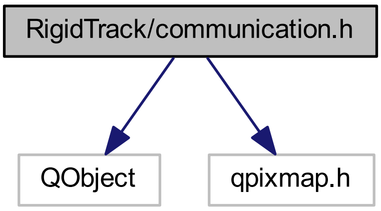
Include dependency graph for communication.cpp:



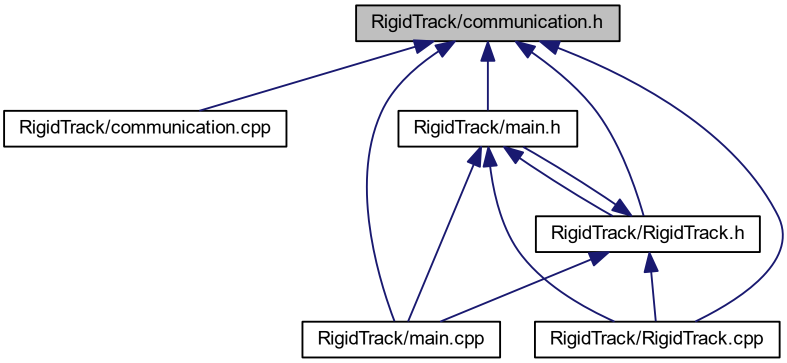
6.3 RigidTrack/communication.h File Reference

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

Include dependency graph for communication.h:



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


RigidTrack/communication.h
RigidTrack/communication.cpp
RigidTrack/main.h
RigidTrack/RigidTrack.h
RigidTrack/main.cpp
RigidTrack/RigidTrack.cpp

Classes

- class commObject

6.4 RigidTrack/DoxygenMain.md File Reference

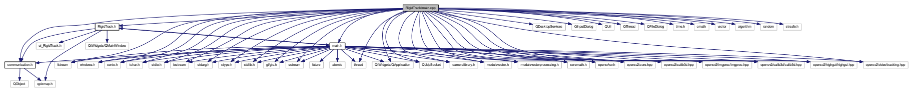
6.5 RigidTrack/main.cpp File Reference

Rigid Track main file that contains most functionallity.

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

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

Include dependency graph for main.cpp:



Functions

- int main (int argc, char ∗argv[ ])

    main initialises the GUI and values for the marker position etc

- QPixmap Mat2QPixmap (cv::Mat src)
- void calcBoardCornerPositions (Size boardSize, float squareSize, std::vector< Point3f > &corners)
- void getEulerAngles (Mat &rotCamerMatrix, Vec3d &eulerAngles)
- int startTracking ()
- void startStopCamera ()

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

- int setReference ()
- int calibrateCamera ()

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

- void loadCalibration (int method)
- void testAlgorithms ()
- void projectCoordinateFrame (Mat pictureFrame)
- void setUpUDP ()

    Open the UDP ports for communication.

- void setHeadingOffset (double d)
- void sendDataUDP (cv::Vec3d &Position, cv::Vec3d &Euler)
- void closeUDP ()
- void loadMarkerConfig (int method)
- void drawPositionText (cv::Mat &Picture, cv::Vec3d &Position, cv::Vec3d &Euler, double error)
- void loadCameraPosition ()
- int determineExposure ()
- void determineOrder ()
- int calibrateGround ()

Variables

- commObject commObj

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

  is the safety feature enabled
- bool safety2Enable = false

  is the second receiver enabled
- double safetyBoxLength = 1.5

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

  bank and pitch angle protection in degrees
- bool exitRequested = true

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  set to true and the algorithm uses the last result as starting value
- int methodPNP = 0

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

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

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

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

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

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

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

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

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

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

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

  new correspondence from list_points3d and list_points_2d
- double currentPointDistance = 5000

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

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

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

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

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

  camera matrix of the camera
- Mat distCoeffs

distortion coefficients of the camera

- Core::DistortionModel distModel

  distortion model of the camera
- QUdpSocket ∗ udpSocketObject

  socket for the communication with receiver 1
- QUdpSocket ∗ udpSocketSafety

  socket for the communication with safety receiver
- QUdpSocket ∗ udpSocketSafety2

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

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

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

  IPv4 adress of receiver 2.
- int portObject = 9155

  Port of receiver 1.
- int portSafety = 9155

  Port of the safety receiver.
- int portSafety2 = 9155

  Port of receiver 2.
- QByteArray datagram

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

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

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

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

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

  Filename for the logfiles.
- std::string logName

  Filename for the logfiles as standard string.
- SYSTEMTIME logDate

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

  file handler for writing the log file

6.5.1 Detailed Description

Rigid Track main file that contains most functionallity.

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

Author

Florian J.T. Wachter

Version

1.0

Date

April, 8th 2017

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.

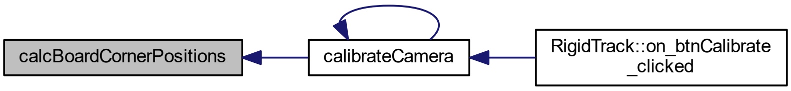
Parameters

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

Definition at line 229 of file main.cpp.

```
{
    corners.clear();

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

Here is the caller graph for this function:



6.5.2.2 calibrateCamera()

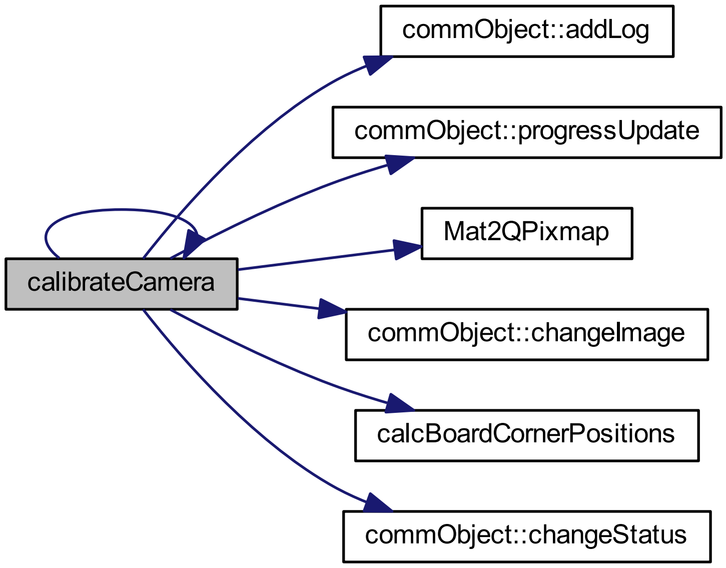
```
int calibrateCamera ( )
```

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

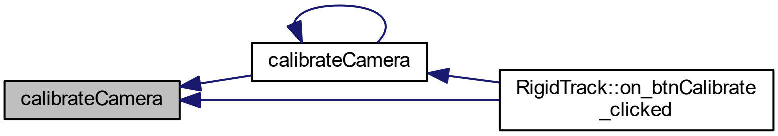
Definition at line 774 of file main.cpp.

```
{
    commObj.addLog("Started camera calibration. 80 pictures are going to be captured.");
    CameraLibrary_EnableDevelopment();

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

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

    //! Get a connected camera ================----
    CameraManager::X().WaitForInitialization();

    Camera *camera = CameraManager::X().GetCamera();
    if (camera == 0)
    {
        commObj.addLog("No camera found!");
        return 1;
    }

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

    //! Set Video Mode ==--

    //! We set the camera to Segment Mode here.  This mode is support by all of our products.
    //! Depending on what device you have connected you might want to consider a different
    //! video mode to achieve the best possible tracking quality.  All devices that support a
    //! mode that will achieve a better quality output with a mode other than Segment Mode are
    //! listed here along with what mode you should use if you're looking for the best head
    //! tracking:
    //!
    //!     V100:R1/R2    Precision Mode
    //!     TrackIR 5     Bit-Packed Precision Mode
    //!     V120          Precision Mode
    //!     TBar          Precision Mode
    //!     S250e         Precision Mode
    //!
    //! If you have questions about a new device that might be conspicuously missing here or
    //! have any questions about head tracking, email support or participate in our forums.

    camera->SetVideoType(Core::GrayscaleMode);

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

    //! Camera Matrix creation  ==--
    cameraMatrix = Mat::eye(3, 3, CV_64F);
    distCoeffs = Mat::zeros(8, 1, CV_64F);

    //! Ok, start main loop.  This loop fetches and displays   ===---
    //! camera frames.                                         ===---
    //! But first set some camera parameters
    camera->SetAGC(false);
    camera->SetAEC(false);
    camera->SetExposure(200);
    camera->SetIntensity(4);
    camera->SetFrameRate(30);
    camera->SetIRFilter(true);
    camera->SetContinuousIR(false);
    camera->SetHighPowerMode(false);

    int number_samples = 0;
    int imagesToSample = 80;

    std::vector<std::vector<Point2f> > imagePoints;
    std::vector<Point2f> pointBuf;
    bool found;
    Size boardSize(9, 6);
    Size imageSize(cameraWidth, cameraHeight);
    Mat Rvec(3, 1, DataType<double>::type);
    Mat Tvec(3, 1, DataType<double>::type);

    //! the user has to provide the size of one square in mm
    bool ok;
    int qsquareSize = QInputDialog::getInt(nullptr, "Chessboard size in mm", "Chessboard size in mm", 23, 1, 60, 1, &ok);
    float squareSize = 23;

    if (ok)
    {
        squareSize = qsquareSize;
    }

    QPixmap QPFrame;
    commObj.progressUpdate(0);
    while (number_samples < imagesToSample)
    {
        //! Fetch a new frame from the camera ===---
        cv::Mat matFrame(cv::Size(cameraWidth, cameraHeight), CV_8UC1);

        //! which is why we also set this constant to 8
        const int BACKBUFFER_BITSPERPIXEL = 8;

        //! later on, when we get the frame as usual:
        CameraLibrary::Frame * frame = camera->GetFrame();

        if (frame)
        {
            //! Lets have the Camera Library raster the camera's
            //! image into our texture.

            frame->Rasterize(cameraWidth, cameraHeight, matFrame.step, BACKBUFFER_BITSPERPIXEL, matFrame.data);
            QPFrame = Mat2QPixmap(matFrame);
            commObj.changeImage(QPFrame);
            found = findChessboardCorners(matFrame, boardSize, pointBuf, CV_CALIB_CB_ADAPTIVE_THRESH | CV_CALIB_CB_FAST_CHECK | CV_CALIB_CB_NORMALIZE_IMAGE);

            if (found)                //!< If done with success,
            {
                //! improve the found corners' coordinate accuracy for chessboard
                cornerSubPix(matFrame, pointBuf, Size(11, 11), Size(-1, -1), TermCriteria(CV_TERMCRIT_EPS + CV_TERMCRIT_ITER, 30, 0.1));

                imagePoints.push_back(pointBuf);
                number_samples += 1;
                commObj.addLog(QString::fromStdString(ss.str()));
                QCoreApplication::processEvents();
            }
            frame->Release();
            ss.str("");
            ss << "Samples found  =  " << number_samples;
            commObj.progressUpdate(number_samples * 100 / imagesToSample);
        }
        Sleep(2);
    }

    std::vector<std::vector<Point3f> > objectPoints(1);
    calcBoardCornerPositions(boardSize, squareSize, objectPoints[0]);
    objectPoints.resize(imagePoints.size(), objectPoints[0]);

    double rms = calibrateCamera(objectPoints, imagePoints, imageSize, cameraMatrix, distCoeffs, Rvec, Tvec);
    commObj.progressUpdate(0);
    //! Release camera ==--
    camera->Release();

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

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.2.3 calibrateGround()

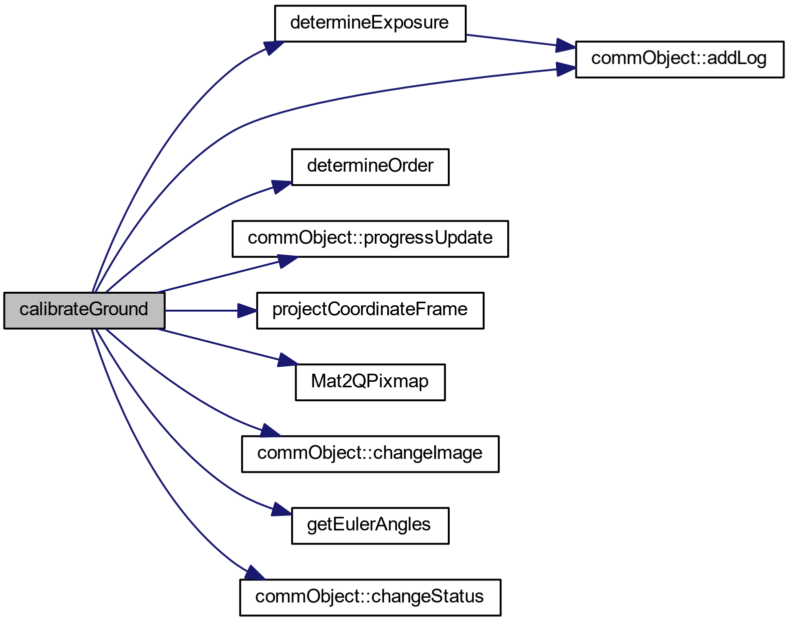
```
int calibrateGround ( )
```

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.

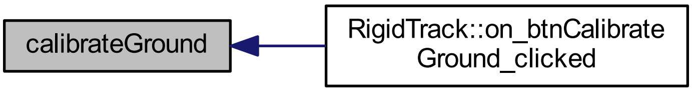
Definition at line 1563 of file main.cpp.

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

    determineExposure();

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

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

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

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

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

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

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

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

    //! Turn on some overlay text so it’s clear things are     ===---
    //! working even if there is nothing in the camera’s view. ===---
    //! Set some other parameters as well of the camera
    camera->SetTextOverlay(true);
    camera->SetFrameRate(intFrameRate);
    camera->SetIntensity(intIntensity);
    camera->SetIRFilter(true);
    camera->SetContinuousIR(false);
    camera->SetHighPowerMode(false);

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

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

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

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

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

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

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

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

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

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


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

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

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

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

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

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

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

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.2.4 closeUDP()

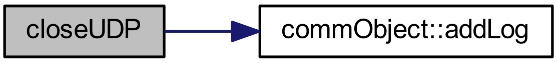
```
void closeUDP ( )
```

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.

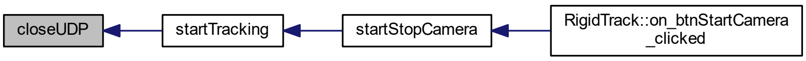
Definition at line 1173 of file main.cpp.

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

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.2.5 determineExposure()

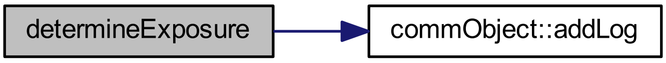
```
int determineExposure ( )
```

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.

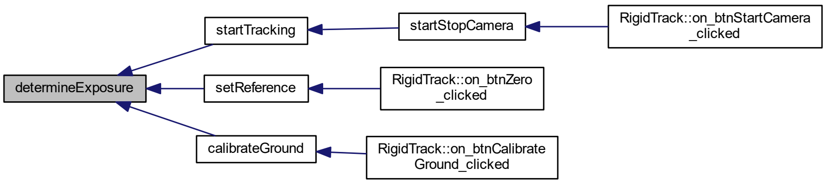
Definition at line 1362 of file main.cpp.

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

    CameraLibrary_EnableDevelopment();

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

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

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

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

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

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

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

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

    //!set exposure such that num markers are visible
    int numberObjects = 0;  //! Number of objects (markers) found in the current picture with the given exposure
    int minExposure = 1;    //! exposure when objects detected the first time is numberMarkers
    int maxExposure = 480;  //! exposure when objects detected is first time numberMarkers+1
    intExposure = minExposure;   //! set the exposure to the smallest value possible
    int numberTries = 0;    //! if the markers arent found after numberTries then there might be no markers at all in the real world

                            //! Determine minimum exposure, hence when are numberMarkers objects detected
    camera->SetExposure(intExposure);
    while (numberObjects != numberMarkers && numberTries < 48)
    {
        //! get a new camera frame
        Frame *frame = camera->GetFrame();
        if (frame) //! frame received
        {
            numberObjects = frame->ObjectCount();   //! how many objects are detected in the image
            if (numberObjects == numberMarkers) { minExposure = intExposure; frame->Release(); break; } //! if the right amount if markers is found, exit while loop
            //! not the right amount of markers was found so increase the exposure and try again
            numberTries++;
            intExposure += 10;
            camera->SetExposure(intExposure);
            ss.str("");
            ss << "Exposure: " << intExposure << "\t";
            ss << "Objects found: " << numberObjects;
            commObj.addLog(QString::fromStdString(ss.str()));
            frame->Release();
        }
    }

    //! Now determine maximum exposure, hence when are numberMarkers+1 objects detected
    numberTries = 0;    //! if the markers arent found after numberTries then there might be no markers at all in the real world
    intExposure = maxExposure;
    camera->SetExposure(intExposure);
    numberObjects = 0;
    while (numberObjects != numberMarkers && numberTries < 48)
    {
        Frame *frame = camera->GetFrame();
        if (frame)
        {
            numberObjects = frame->ObjectCount(); //! how many objects are detected in the image
            if (numberObjects == numberMarkers) { maxExposure = intExposure; frame->Release(); break; } //! if the right amount if markers is found, exit while loop

            //! not the right amount of markers was found so decrease the exposure and try again
            intExposure -= 10;
            numberTries++;
            camera->SetExposure(intExposure);
            ss.str("");
            ss << "Exposure: " << intExposure << "\t";
            ss << "Objects found: " << numberObjects;
            commObj.addLog(QString::fromStdString(ss.str()));
            frame->Release();
        }
    }

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

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

    camera->Release();
    return 0;

}
```

Here is the call graph for this function:



Here is the caller graph for this function:


determineExposure
startTracking
setReference
calibrateGround
startStopCamera
RigidTrack::on_btnStartCamera _clicked
RigidTrack::on_btnZero _clicked
RigidTrack::on_btnCalibrate Ground_clicked

6.5.2.6 determineOrder()

```
void determineOrder ( )
```

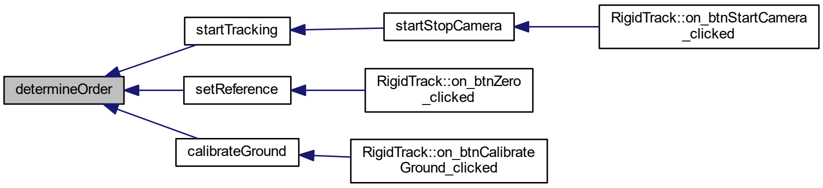
Compute the order of the marker points in 2D so they are the same as in the 3D array. Hence marker 1 must be in first place for both, list_points2d and list_points3d.

Definition at line 1498 of file main.cpp.

```
{
    //! determine the 3D-2D correspondences that are crucial for the PnP algorithm
    //! Try every possible correspondence and solve PnP
    //! Then project the 3D marker points into the 2D camera image and check the difference
    //! between projected points and points as seen by the camera
    //! the corresponce with the smallest difference is probably the correct one

        //! the difference between true 2D points and projected points is super big
    minPointDistance = 5000;
    std::sort(pointOrderIndices, pointOrderIndices + 4);

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

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

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

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

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

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

        //! if the difference with the current permutation is smaller than the smallest value till now
        //! it is probably the more correct permutation
        if (currentPointDistance < minPointDistance)
        {
            minPointDistance = currentPointDistance;    //!< set the
```

```
  smallest value of difference to the current one
            for (int b = 0; b < numberMarkers; b++)    //!< now safe the better permutation
            {
                pointOrderIndicesNew[b] = pointOrderIndices[b];
            }
        }


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

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

Here is the caller graph for this function:


determineOrder
startTracking
setReference
calibrateGround
startStopCamera
RigidTrack::on_btnStartCamera _clicked
RigidTrack::on_btnZero _clicked
RigidTrack::on_btnCalibrate Ground_clicked

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.

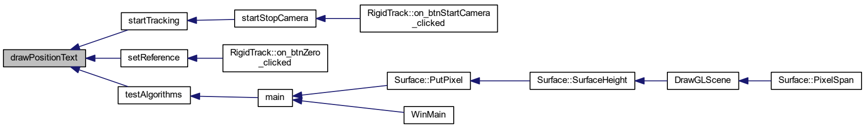
Parameters

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

Definition at line 1315 of file main.cpp.

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

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

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

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

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

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

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

Here is the caller graph for this function:



6.5.2.8 getEulerAngles()

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

Get the euler angles from a rotation matrix

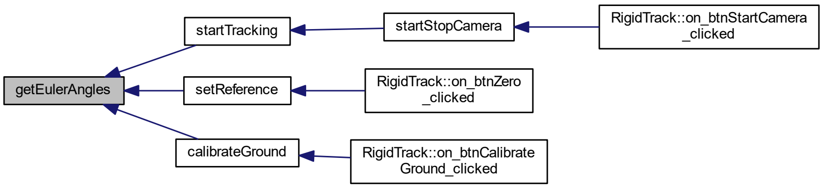
Parameters

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

Definition at line 241 of file main.cpp.

```
                                                            {

    Mat cameraMatrix, rotMatrix, transVect, rotMatrixX, rotMatrixY, rotMatrixZ;
    double* _r = rotCamerMatrix.ptr<double>();
    double projMatrix[12] = { _r[0],_r[1],_r[2],0,
        _r[3],_r[4],_r[5],0,
        _r[6],_r[7],_r[8],0 };

    decomposeProjectionMatrix(Mat(3, 4, CV_64FC1, projMatrix),
        cameraMatrix,
251         rotMatrix,
252         transVect,
253         rotMatrixX,
254         rotMatrixY,
255         rotMatrixZ,
256         eulerAngles);
257 }
```

Here is the caller graph for this function:


getEulerAngles
startTracking
setReference
calibrateGround
startStopCamera
RigidTrack::on_btnStartCamera_clicked
RigidTrack::on_btnZero_clicked
RigidTrack::on_btnCalibrateGround_clicked

6.5.2.9 loadCalibration()

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

Load a previously saved camera calibration from a file.

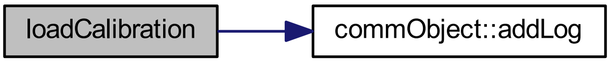
Parameters

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.

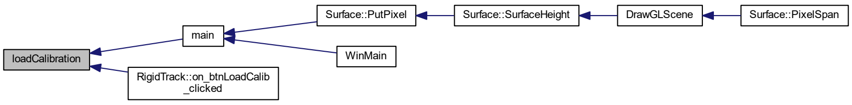
Definition at line 923 of file main.cpp.

```
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()

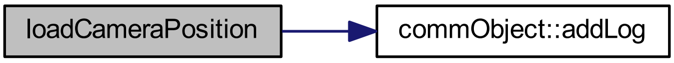
```
void loadCameraPosition ( )
```

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.

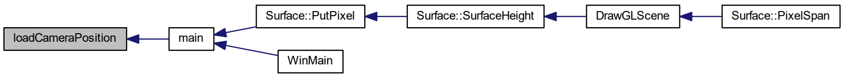
Definition at line 1348 of file main.cpp.

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

Here is the 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

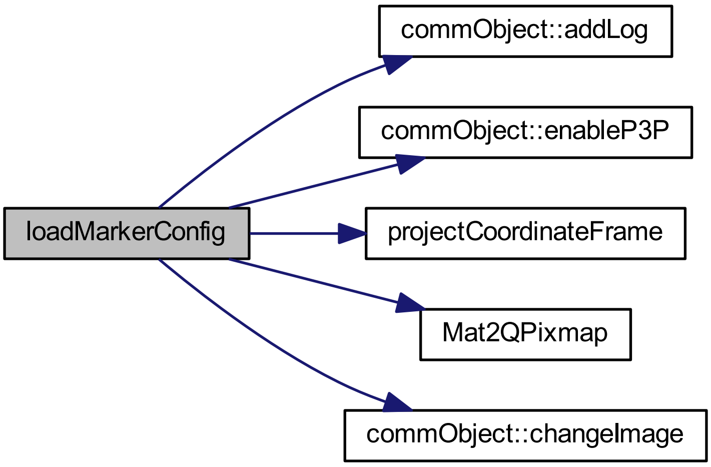
in	*method*	whether or not load the configuration from the markerStandard.xml. If ==0 load it, if != 0 let the user select a different file.

Definition at line 1195 of file main.cpp.

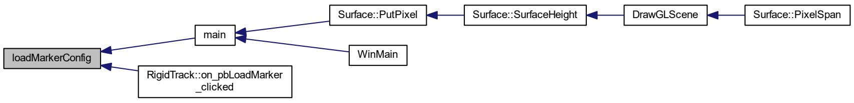
```
{
    QString fileName;
    //! during start up of the programm load the standard marker configuration
    if (method == 0)
    {
        //! open the standard marker configuration file
        FileStorage fs;
        fs.open("markerStandard.xml", FileStorage::READ);

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

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

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



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

        //! was cancel or abort clicked
        if (fileName.length() == 0)
        {
```

```
      //! if yes load the standard marker configuration
      fileName = "markerStandard.xml";
    }

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

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

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

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

  }

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

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

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

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

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

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

}
```

Here is the call graph for this function:



Here is the caller graph for this function:



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.

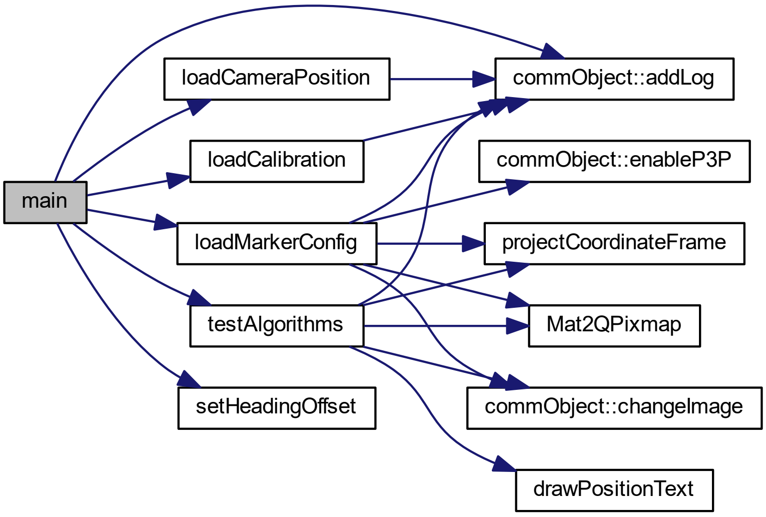
Parameters

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

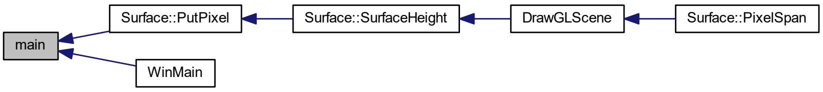
Definition at line 156 of file main.cpp.

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

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

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

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

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

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

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

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

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

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

    return a.exec();
}
```

Here is the call graph for this function:



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 Pixmap object for the GUI.

Parameters

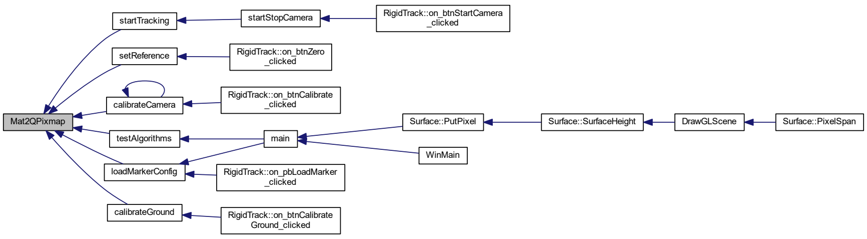
in	*src*	is the camera image represented as OpenCV matrix.

Definition at line 216 of file main.cpp.

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

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

Here is the caller graph for this function:



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.

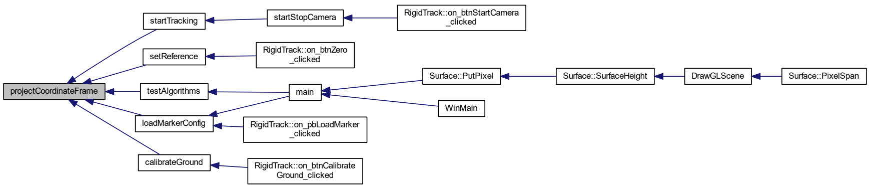
Parameters

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

Definition at line 1081 of file main.cpp.

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

Here is the caller graph for this function:



6.5.2.15 sendDataUDP()

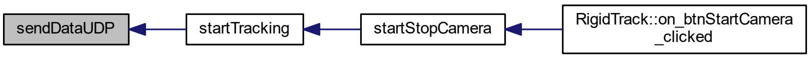
```
void sendDataUDP (
            cv::Vec3d & Position,
            cv::Vec3d & Euler )
```

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.

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

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

}
```

Here is the caller graph for this function:



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.

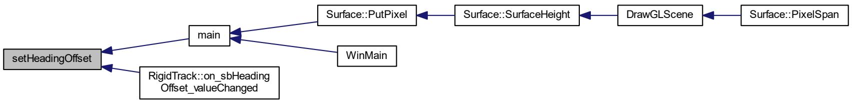
Parameters

in	*d*	denotes heading offset in degrees.

Definition at line 1122 of file main.cpp.

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

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

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

    //! Calculate rotation about z axis
    Mat R_z = (Mat_<double>(3, 3) <<
        cos(d), -sin(d), 0,
        sin(d), cos(d), 0,
        0, 0, 1);


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

Here is the caller graph for this function:



6.5.2.17 setReference()

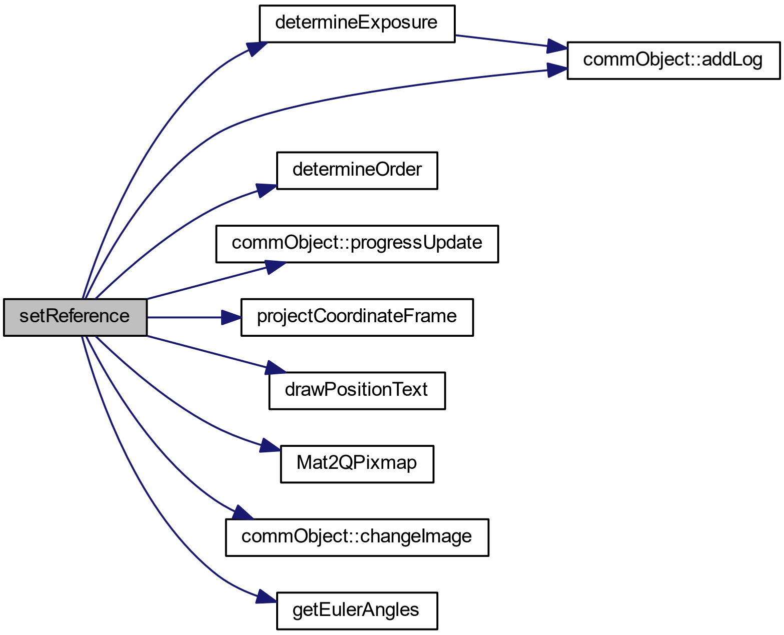
```
int setReference ( )
```

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.

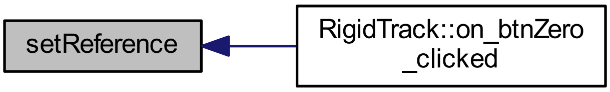
Definition at line 595 of file main.cpp.

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

    determineExposure();

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

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

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

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

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

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

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

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

    //! Turn on some overlay text so it’s clear things are     ===---
    //! working even if there is nothing in the camera’s view. ===---
    //! Set some other parameters as well of the camera
    camera->SetTextOverlay(true);
    camera->SetFrameRate(intFrameRate);
    camera->SetIntensity(intIntensity);
    camera->SetIRFilter(true);
    camera->SetContinuousIR(false);
    camera->SetHighPowerMode(false);

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

    //! compute the difference between last obtained TVec and the average Value
    //! When it is large the iterative method has not converged properly so it is advised to start the setReference() function once again
    double error = norm(posRef) - norm(Tvec);
    if (error > 5.0)
    {
        ss << "Caution, distance between reference position and last position is: " << error << "\n Start the set zero routine once again.";
    }
    commObj.addLog(QString::fromStdString(ss.str()));
    commObj.progressUpdate(0);
    return 0;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.2.18 setUpUDP()

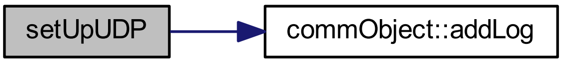
```
void setUpUDP ( )
```

Open the UDP ports for communication.

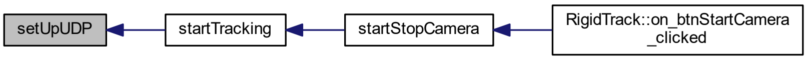
Definition at line 1090 of file main.cpp.

```
{
    //! Initialise the QDataStream that stores the data to be send
    QDataStream out(&datagram, QIODevice::WriteOnly);
    out.setVersion(QDataStream::Qt_4_3);

    //! Create UDP slots
    commObj.addLog("Opening UDP ports.");
    udpSocketObject = new QUdpSocket(0);
    udpSocketObject->connectToHost(IPAdressObject,
 portObject);
    commObj.addLog("Opened first receiver UDP port.");

    udpSocketSafety = new QUdpSocket(0);
    udpSocketSafety2 = new QUdpSocket(0);

    //! if the safety feature is activated open the udp port
    if (safetyEnable)
    {
        udpSocketSafety->connectToHost(IPAdressSafety,
 portSafety);
        commObj.addLog("Opened safety UDP port.");
    }

    //! if the second receiver feature is activated open the udp port
    if (safety2Enable)
    {
        udpSocketSafety2->connectToHost(IPAdressSafety2,
 portSafety2);
        commObj.addLog("Opened second receiver UDP port.");
    }
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.2.19 startStopCamera()

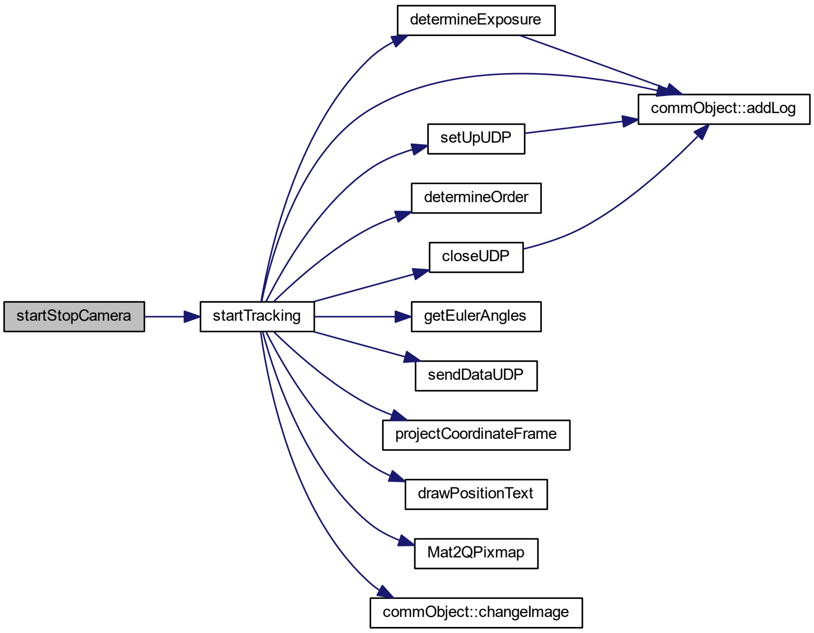
```
void startStopCamera ( )
```

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

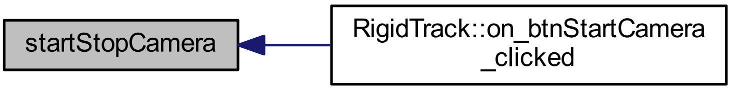
Definition at line 579 of file main.cpp.

```
{
    //! tracking is not running so start it
    if (exitRequested)
    {
        exitRequested = false;
        startTracking();
    }
    else  //!< tracking is currently running, set exitRequest to true so the while loop in startTracking() exits
    {
        exitRequested = true;
    }
}
```

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.

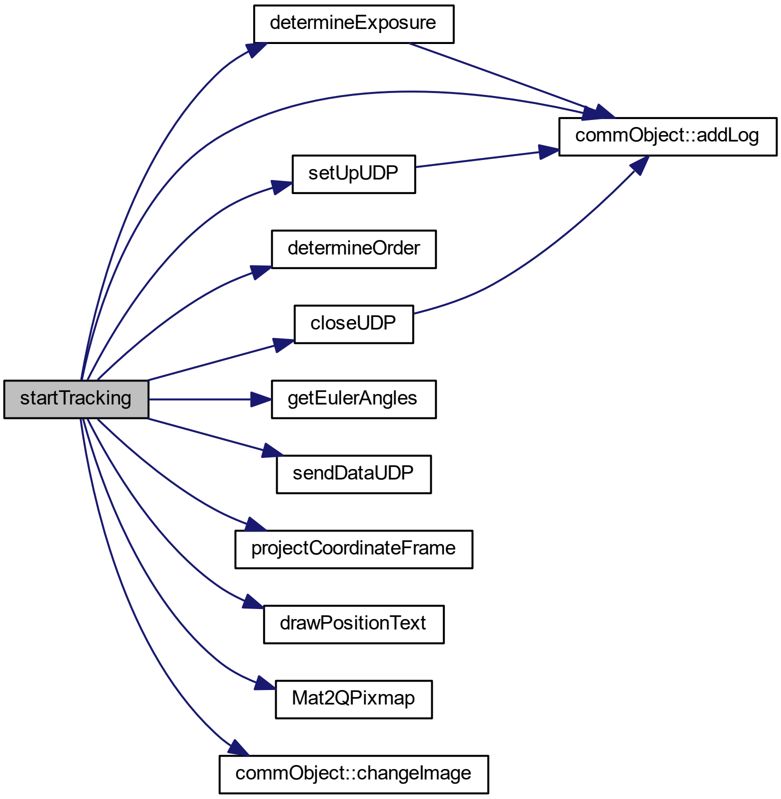
```
261                     {
262 
263 
264     gotOrder = false; //! The order of points, hence which entry in list_points3d corresponds to which in list_points2d is not calculated yet
265     Rvec = RvecOriginal; //! Use the value of Rvec that was set in main() as starting value for the solvePnP algorithm
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.toStdString(); //! Convert the QString to a standard string
273 
274     determineExposure(); //! Get the exposure where the right amount of markers is detected
275 
276     //! For OptiTrack Ethernet cameras, it's important to enable development mode if you
277     //! want to stop execution for an extended time while debugging without disconnecting
278     //! the Ethernet devices.  Lets do that now:
279 
280     CameraLibrary_EnableDevelopment();
281     CameraLibrary::CameraManager::X(); //! Initialize Camera SDK
282 
283     //! At this point the Camera SDK is actively looking for all connected cameras and will initialize
284     //! them on it's own
285 
286     //! Get a connected camera
287     CameraManager::X().WaitForInitialization();
288     Camera *camera = CameraManager::X().GetCamera();
289 
290     //! If no camera can be found, inform user in message log and exit function
291     if (camera == 0)
292     {
293         commObj.addLog("No camera found!");
294         return 1;
295     }
296 
297     //! Determine camera resolution to size application window
298     int cameraWidth = camera->Width();
299     int cameraHeight = camera->Height();
300 
301     camera->SetVideoType(Core::PrecisionMode);  //! Set the camera mode to precision mode, it used greyscale 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
```

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

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

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

    //! Fetch a new frame from the camera
    bool gotTime = false; //! Get the timestamp of the first frame. This time is subtracted from every subseeding frame so the time starts at 0 in the logs
    while (!gotTime) //! While no new frame is received loop
    {
        Frame *frame = camera->GetFrame(); //! Get a new camera frame
        if (frame)  //! There is actually a new frame
        {
            timeFirstFrame = frame->TimeStamp(); //! Get the time stamp for the first frame. It is subtracted for the following frames
            frame->Release();   //! Release the frame so the camera can continue
            gotTime = true; //! Exit the while loop
        }
    }

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

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

        if (frame) //! Did we got a new frame or does the camera still need more time
        {
            framesDropped++; //! Increase by one, if everything is okay it is decreased at the end of the loop again

            //! Only use this frame it the right number of markers is found in the picture
            if (frame->ObjectCount() == numberMarkers)
            {
                //! Get the marker points in 2D in the camera image frame and store them in the list_points2dUnsorted vector
                //! The order of points that come from the camera corresponds to the Y coordinate
                for (int i = 0; i < numberMarkers; i++)
                {
                    cObject *obj = frame->Object(i);
                    list_points2dUnsorted[i] = cv::Point2d(obj->X(), obj->Y());
                }

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

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

                //! The first time the 2D-3D corresspondence was determined with gotOrder was okay.
                //! But this order can change as the object moves and the marker objects appear in a
                //! different order in the frame->Object() array.
                //! The solution is that: When a marker point (in the camera image, hence in 2D) was at
                //! a position then it wont move that much from one frame to the other.
                //! So for the new frame we take a marker object and check which marker was closest this point
                //! in the old image frame? This is probably the same (true) marker. And we do that for every other marker as well.
                //! When tracking is good and no frames are dropped because of missing markers this should work every frame.
                for (int j = 0; j < numberMarkers; j++)
                {
                    minPointDistance = 5000; //! The sum of point distances is set to something unrealistic large
                    for (int k = 0; k < numberMarkers; k++)
                    {
                        //! Calculate N_2 norm of unsorted points minus old points
                        currentPointDistance = norm(
list_points2dUnsorted[pointOrderIndices[j]] -
list_points2dOld[k]);
                        //! If the norm is smaller than minPointDistance the correspondence is more likely
 to be correct
                        if (currentPointDistance <
minPointDistance)
                        {
                            //! Update the array that saves the new point order
                            minPointDistance =
currentPointDistance;
                            pointOrderIndicesNew[j] = k;
                        }
                    }
                }

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

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

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

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

                //! Increase the framesDropped variable if accuracy of tracking is too bad
                if (projectionError > 5)
                {
                    framesDropped++;
                }
                else
                {
                    framesDropped = 0;  //! Set number of subsequent frames dropped to zero because error
 is small enough and no marker was missing
                }

                //! Get the min and max values from TVec for sanity check
                minMaxLoc(Tvec.at<double>(2), &minValue, &maxValue);

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

                //! Next step is the transformation from camera CoSy to navigation CoSy
                //! Compute the relative object position from the reference position to the current one
                //! given in the camera CoSy: \f$ T_C^{NM} = Tvec - Tvec_{Ref} \f$
                subtract(Tvec, posRef, position);

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

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

                //! The difference of the reference rotation and the current rotation
                //! \f$ R_{ NM } = M_{ NC } \times R_{ CM } \f$
```

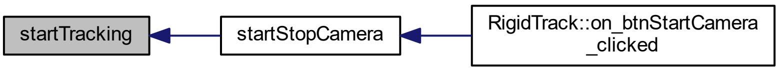
```
            Rmat = RmatRef.t() *Rmat;

            //! Euler Angles, finally
            getEulerAngles(Rmat, eulerAngles); //! Get the euler angles from the rotation matrix
            eulerAngles[2] += headingOffset; //! Add the heading offset to the heading angle

            //! Compute the velocity with finite differences. Only use is the log file. It is done here because the more precise time stamp can be used
            frameTime = frame->TimeStamp() - timeOld;   //! Time between the old frame and the current frame
            timeOld = frame->TimeStamp();    //! Set the old frame time to the current  one
            velocity[0] = (position[0] - positionOld[0]) / frameTime; //! Calculate the x velocity with finite differences
            velocity[1] = (position[1] - positionOld[1]) / frameTime; //! Calculate the y velocity with finite differences
            velocity[2] = (position[2] - positionOld[2]) / frameTime; //! Calculate the z velocity with finite differences
            positionOld = position;  //! Set the old position to the current one for next frame velocity calcuation

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

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

        //! Check if the position and euler angles are below the allowed value, if yes send OKAY signal (1), if not send shutdown signal (0)
        //! Absolute x, y and z position in navigation CoSy must be smaller than the allowed distance
        if (safetyEnable)
        {
            if ((abs(position[0]) < safetyBoxLength && abs(position[1]) < safetyBoxLength && abs(position[2]) < safetyBoxLength))
            {
                //! Absolute Euler angles must be smaller than allowed value. Heading is not considered

                if ((abs(eulerAngles[0]) < safetyAngle && abs(eulerAngles[1]) < safetyAngle))
                {
                    //! Send the OKAY signal to the desired computer every 5th time
                    if (v == 5) {
                        data.setNum((int)(1));
                        udpSocketSafety->write(data); //! Send the 1
                        v = 0; //! reset the counter that is needed for decimation to every 5th time step
                    }
                }
                //! The euler angles of the object exceeded the allowed euler angles, send the shutdown signal (0)
                else
                {
                    data.setNum((int)(0));  //! Send the shutdown signal, a 0
                    udpSocketSafety->write(data);
                    commObj.addLog("Object exceeded allowed Euler angles, shutdown signal sent."); //! Inform the user

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

            }
        }

        //! Inform the user if tracking system is disturbed (marker lost or so) or error was too big
        if (framesDropped > 10)
        {
            if (safetyEnable) //! Also send the shutdown signal
            {
                data.setNum((int)(0));  //! Send the shutdown signal, a 0
```

```
                    udpSocketSafety->write(data);
                }
                commObj.addLog("Lost marker points or precision was bad!"); //! Inform the user
                framesDropped = 0;
            }

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

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

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

            //! Project the marker points from 3D to the camera image frame (2d) with the computed pose
            projectPoints(list_points3d, Rvec, Tvec, cameraMatrix, distCoeffs, list_points2d);
            for (int i = 0; i < numberMarkers; i++)
            {
                //! Draw a circle around the projected points so the result can be better compared to the real marker position
                //! In the resulting picture those are the red dots
                circle(cFrame, Point(list_points2d[i].x, list_points2d[i].y), 3, Scalar(225, 0, 0), 3);
            }

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

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

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



6.5.2.21 testAlgorithms()

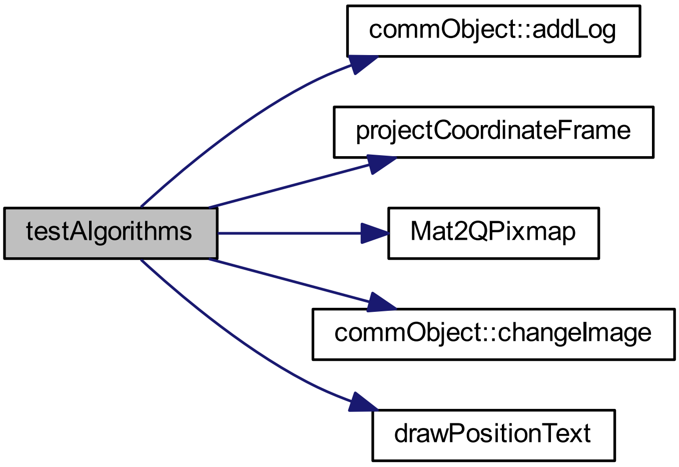
```
void testAlgorithms ( )
```

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.

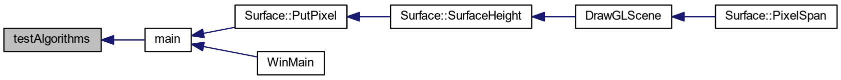
Definition at line 952 of file main.cpp.

```
{

    int _methodPNP;

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

    RvecOriginal = Rvec;
    TvecOriginal = Tvec;

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

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

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

    projectCoordinateFrame(cFrame);

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

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

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


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

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

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

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

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

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

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

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

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

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

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

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

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

    Rvec = RvecOriginal;
    Tvec = TvecOriginal;

}
```

Here is the call graph for this function:



Here is the caller graph for this function:



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 tetermined 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

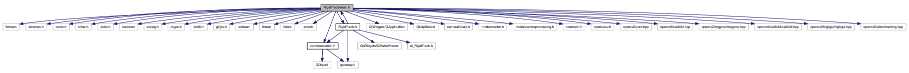
Definition at line 84 of file main.cpp.

6.6 RigidTrack/main.h File Reference

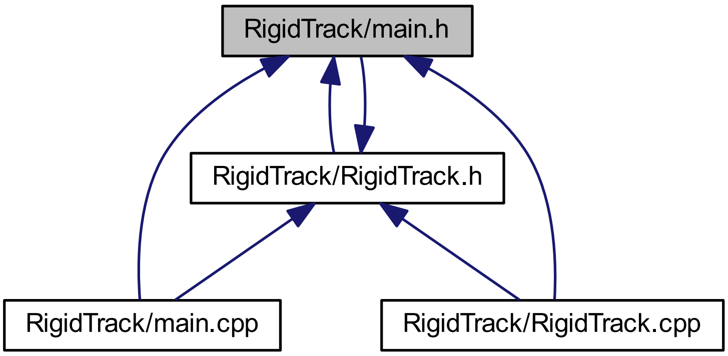
Header file for main.cpp.

```
#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()

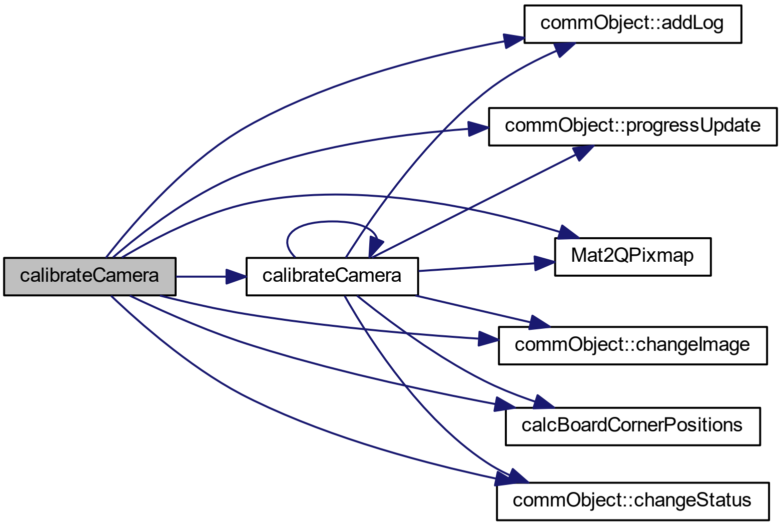
```
int calibrateCamera ( )
```

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

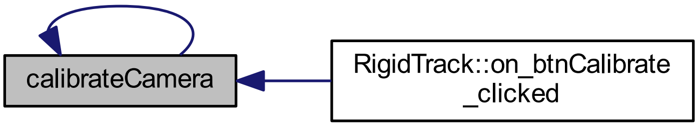
Definition at line 774 of file main.cpp.

```
{
    commObj.addLog("Started camera calibration. 80 pictures are going to be captured.");
    CameraLibrary_EnableDevelopment();

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

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

    //! Get a connected camera ================----
    CameraManager::X().WaitForInitialization();

    Camera *camera = CameraManager::X().GetCamera();
    if (camera == 0)
    {
        commObj.addLog("No camera found!");
        return 1;
    }

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

    //! Set Video Mode ==--

    //! We set the camera to Segment Mode here.  This mode is support by all of our products.
    //! Depending on what device you have connected you might want to consider a different
    //! video mode to achieve the best possible tracking quality.  All devices that support a
    //! mode that will achieve a better quality output with a mode other than Segment Mode are
    //! listed here along with what mode you should use if you’re looking for the best head
    //! tracking:
    //!
    //!     V100:R1/R2     Precision Mode
    //!     TrackIR 5      Bit-Packed Precision Mode
    //!     V120           Precision Mode
    //!     TBar           Precision Mode
    //!     S250e          Precision Mode
    //!
    //! If you have questions about a new device that might be conspicuously missing here or
    //! have any questions about head tracking, email support or participate in our forums.

    camera->SetVideoType(Core::GrayscaleMode);

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

    //! Camera Matrix creation  ==--
    cameraMatrix = Mat::eye(3, 3, CV_64F);
    distCoeffs = Mat::zeros(8, 1, CV_64F);

    //! Ok, start main loop.  This loop fetches and displays   ===---
    //! camera frames.                                         ===---
    //! But first set some camera parameters
    camera->SetAGC(false);
    camera->SetAEC(false);
    camera->SetExposure(200);
    camera->SetIntensity(4);
    camera->SetFrameRate(30);
    camera->SetIRFilter(true);
    camera->SetContinuousIR(false);
    camera->SetHighPowerMode(false);

    int number_samples = 0;
    int imagesToSample = 80;

    std::vector<std::vector<Point2f> > imagePoints;
    std::vector<Point2f> pointBuf;
    bool found;
    Size boardSize(9, 6);
    Size imageSize(cameraWidth, cameraHeight);
    Mat Rvec(3, 1, DataType<double>::type);
    Mat Tvec(3, 1, DataType<double>::type);

    //! the user has to provide the size of one square in mm
    bool ok;
    int qsquareSize = QInputDialog::getInt(nullptr, "Chessboard size in mm", "Chessboard size in mm", 23, 1, 60, 1, &ok);
    float squareSize = 23;

    if (ok)
    {
        squareSize = qsquareSize;
    }

    QPixmap QPFrame;
    commObj.progressUpdate(0);
    while (number_samples < imagesToSample)
    {
        //! Fetch a new frame from the camera ===---
        cv::Mat matFrame(cv::Size(cameraWidth, cameraHeight), CV_8UC1);

        //! which is why we also set this constant to 8
        const int BACKBUFFER_BITSPERPIXEL = 8;

        //! later on, when we get the frame as usual:
        CameraLibrary::Frame * frame = camera->GetFrame();

        if (frame)
        {
            //! Lets have the Camera Library raster the camera's
            //! image into our texture.

            frame->Rasterize(cameraWidth, cameraHeight, matFrame.step, BACKBUFFER_BITSPERPIXEL, matFrame.data);
            QPFrame = Mat2QPixmap(matFrame);
            commObj.changeImage(QPFrame);
            found = findChessboardCorners(matFrame, boardSize, pointBuf, CV_CALIB_CB_ADAPTIVE_THRESH | CV_CALIB_CB_FAST_CHECK | CV_CALIB_CB_NORMALIZE_IMAGE);

            if (found)                //!< If done with success,
            {
                //! improve the found corners' coordinate accuracy for chessboard
                cornerSubPix(matFrame, pointBuf, Size(11, 11), Size(-1, -1), TermCriteria(CV_TERMCRIT_EPS + CV_TERMCRIT_ITER, 30, 0.1));

                imagePoints.push_back(pointBuf);
                number_samples += 1;
                commObj.addLog(QString::fromStdString(ss.str()));
                QCoreApplication::processEvents();
            }
            frame->Release();
            ss.str("");
            ss << "Samples found  =  " << number_samples;
            commObj.progressUpdate(number_samples * 100 / imagesToSample);
        }
        Sleep(2);
    }

    std::vector<std::vector<Point3f> > objectPoints(1);
    calcBoardCornerPositions(boardSize, squareSize, objectPoints[0]);
    objectPoints.resize(imagePoints.size(), objectPoints[0]);

    double rms = calibrateCamera(objectPoints, imagePoints, imageSize, cameraMatrix, distCoeffs, Rvec, Tvec);
    commObj.progressUpdate(0);
    //! Release camera ==--
    camera->Release();

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

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.2.2 calibrateGround()

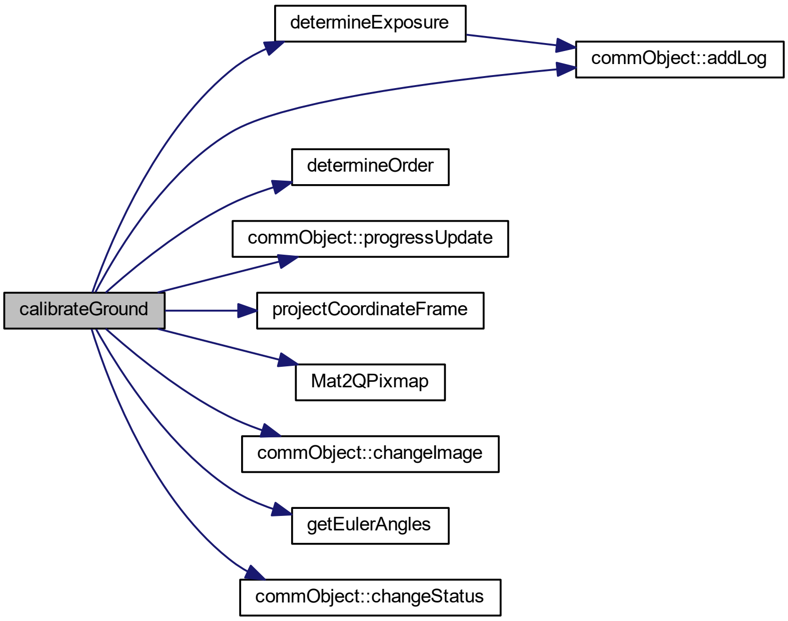
```
int calibrateGround ( )
```

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.

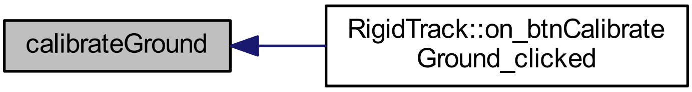
Definition at line 1563 of file main.cpp.

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

    determineExposure();

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

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

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

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

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

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

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

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

    //! Turn on some overlay text so it's clear things are     ===---
    //! working even if there is nothing in the camera's view. ===---
    //! Set some other parameters as well of the camera
    camera->SetTextOverlay(true);
    camera->SetFrameRate(intFrameRate);
    camera->SetIntensity(intIntensity);
    camera->SetIRFilter(true);
    camera->SetContinuousIR(false);
    camera->SetHighPowerMode(false);

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

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

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

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

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

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

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

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

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

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


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

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

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

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

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

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

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

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.2.3 closeUDP()

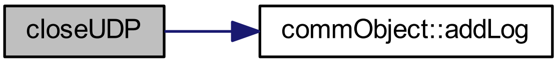
```
void closeUDP ( )
```

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.

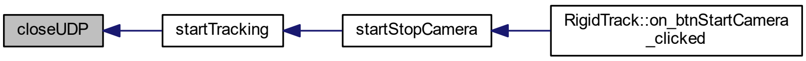
Definition at line 1173 of file main.cpp.

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

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.2.4 determineExposure()

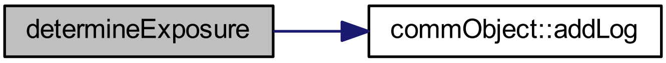
```
int determineExposure ( )
```

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.

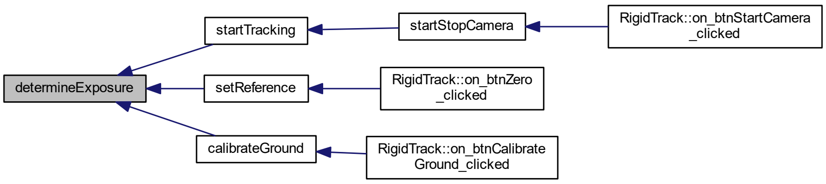
Definition at line 1362 of file main.cpp.

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

    CameraLibrary_EnableDevelopment();

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

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

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

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

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

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

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

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

    //!set exposure such that num markers are visible
    int numberObjects = 0;  //! Number of objects (markers) found in the current picture with the given exposure
    int minExposure = 1;    //! exposure when objects detected the first time is numberMarkers
    int maxExposure = 480;  //! exposure when objects detected is first time numberMarkers+1
    intExposure = minExposure;   //! set the exposure to the smallest value possible
    int numberTries = 0;    //! if the markers arent found after numberTries then there might be no markers at all in the real world

                            //! Determine minimum exposure, hence when are numberMarkers objects detected
    camera->SetExposure(intExposure);
    while (numberObjects != numberMarkers && numberTries < 48)
    {
        //! get a new camera frame
        Frame *frame = camera->GetFrame();
        if (frame) //! frame received
        {
            numberObjects = frame->ObjectCount();   //! how many objects are detected in the image
            if (numberObjects == numberMarkers) { minExposure = intExposure; frame->Release(); break; } //! if the right amount if markers is found, exit while loop
            //! not the right amount of markers was found so increase the exposure and try again
            numberTries++;
            intExposure += 10;
            camera->SetExposure(intExposure);
            ss.str("");
            ss << "Exposure: " << intExposure << "\t";
            ss << "Objects found: " << numberObjects;
            commObj.addLog(QString::fromStdString(ss.str()));
            frame->Release();
        }
    }

    //! Now determine maximum exposure, hence when are numberMarkers+1 objects detected
    numberTries = 0;    //! if the markers arent found after numberTries then there might be no markers at all in the real world
    intExposure = maxExposure;
    camera->SetExposure(intExposure);
    numberObjects = 0;
    while (numberObjects != numberMarkers && numberTries < 48)
    {
        Frame *frame = camera->GetFrame();
        if (frame)
        {
            numberObjects = frame->ObjectCount(); //! how many objects are detected in the image
            if (numberObjects == numberMarkers) { maxExposure = intExposure; frame->Release(); break; } //! if the right amount if markers is found, exit while loop

            //! not the right amount of markers was found so decrease the exposure and try again
            intExposure -= 10;
            numberTries++;
            camera->SetExposure(intExposure);
            ss.str("");
            ss << "Exposure: " << intExposure << "\t";
            ss << "Objects found: " << numberObjects;
            commObj.addLog(QString::fromStdString(ss.str()));
            frame->Release();
        }
    }

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

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

    camera->Release();
    return 0;

}
```

Here is the call graph for this function:



Here is the caller graph for this function:


determineExposure
startTracking
setReference
calibrateGround
startStopCamera
RigidTrack::on_btnStartCamera _clicked
RigidTrack::on_btnZero _clicked
RigidTrack::on_btnCalibrate Ground_clicked

6.6.2.5 determineOrder()

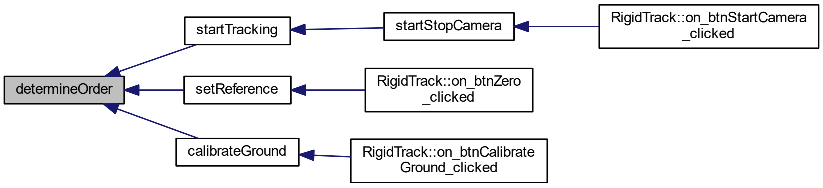
```
void determineOrder ( )
```

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.

```
{
    //! determine the 3D-2D correspondences that are crucial for the PnP algorithm
    //! Try every possible correspondence and solve PnP
    //! Then project the 3D marker points into the 2D camera image and check the difference
    //! between projected points and points as seen by the camera
    //! the corresponce with the smallest difference is probably the correct one

        //! the difference between true 2D points and projected points is super big
    minPointDistance = 5000;
    std::sort(pointOrderIndices, pointOrderIndices + 4);

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

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

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

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

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

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

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


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

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

Here is the caller graph for this function:


determineOrder
startTracking
setReference
calibrateGround
startStopCamera
RigidTrack::on_btnStartCamera _clicked
RigidTrack::on_btnZero _clicked
RigidTrack::on_btnCalibrate Ground_clicked

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.

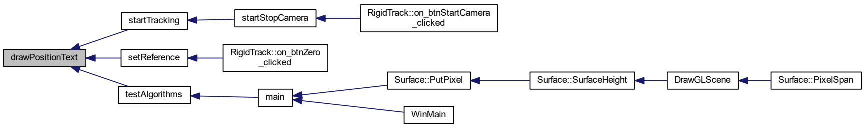
Parameters

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

Definition at line 1315 of file main.cpp.

```
{
    ss.str("");
    ss << "X: " << Position[0] << " m";
    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

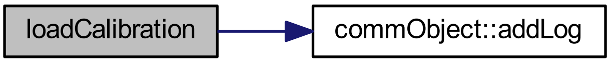
in	*method*	whether or not load the camera calibration from calibration.xml. If ==0 then yes, if != 0 then let the user select a different file.

Definition at line 923 of file main.cpp.

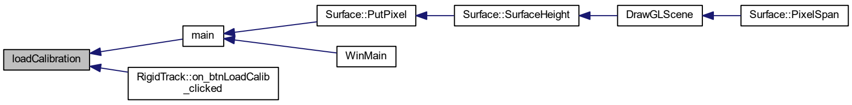
```
923                                   {
924
925     QString fileName;
926     if (method == 0)
927     {
928         fileName = "calibration.xml";
929     }
930     else
931     {
932         fileName = QFileDialog::getOpenFileName(nullptr, "Choose a previous saved calibration file", "", "
     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

```
            fileName = "calibration.xml";
        }
    }
    FileStorage fs;
    fs.open(fileName.toUtf8().constData(), FileStorage::READ);
    fs["CameraMatrix"] >> cameraMatrix;
    fs["DistCoeff"] >> distCoeffs;
    commObj.addLog("Loaded calibration from file:");
    commObj.addLog(fileName);
    ss.str("");
    ss << "\nCamera Matrix is" << "\n" << cameraMatrix << "\n";
    ss << "\nDistortion Coefficients are" << "\n" << distCoeffs << "\n";
    commObj.addLog(QString::fromStdString(ss.str()));
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.2.8 loadCameraPosition()

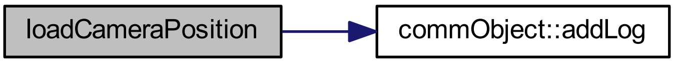
```
void loadCameraPosition ( )
```

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.

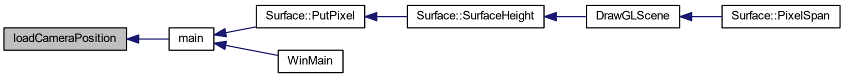
Definition at line 1348 of file main.cpp.

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

Here is the 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.

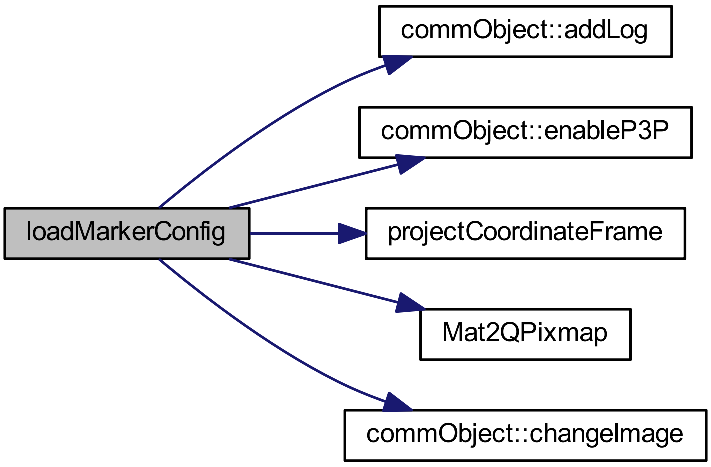
Parameters

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.

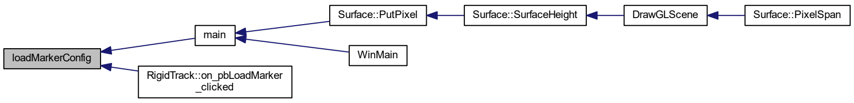
Definition at line 1195 of file main.cpp.

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

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

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

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



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

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

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

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

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

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

    }

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

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

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

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

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

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

}
```

Here is the call graph for this function:



Here is the caller graph for this function:



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.

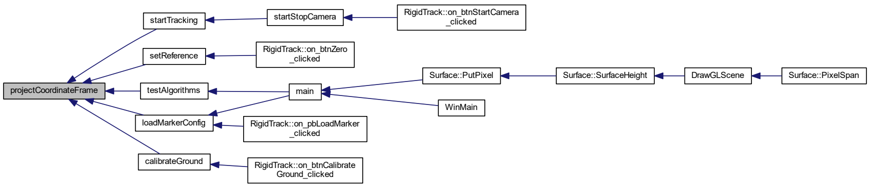
Parameters

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

Definition at line 1081 of file main.cpp.

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

Here is the caller graph for this function:



6.6.2.11 sendDataUDP()

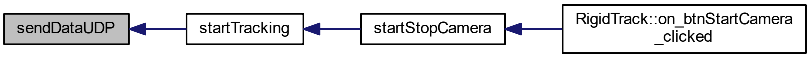
```
void sendDataUDP (
            cv::Vec3d & Position,
            cv::Vec3d & Euler )
```

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

Definition at line 1154 of file main.cpp.

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

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

}
```

Here is the caller graph for this function:



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.

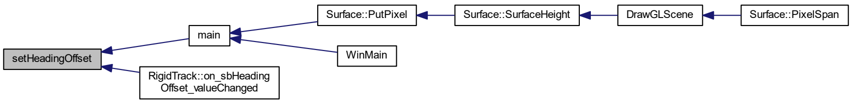
Parameters

in	*d*	denotes heading offset in degrees.

Definition at line 1122 of file main.cpp.

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

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

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

    //! Calculate rotation about z axis
    Mat R_z = (Mat_<double>(3, 3) <<
        cos(d), -sin(d), 0,
        sin(d), cos(d), 0,
        0, 0, 1);


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

Here is the caller graph for this function:



6.6.2.13 setReference()

```
int setReference ( )
```

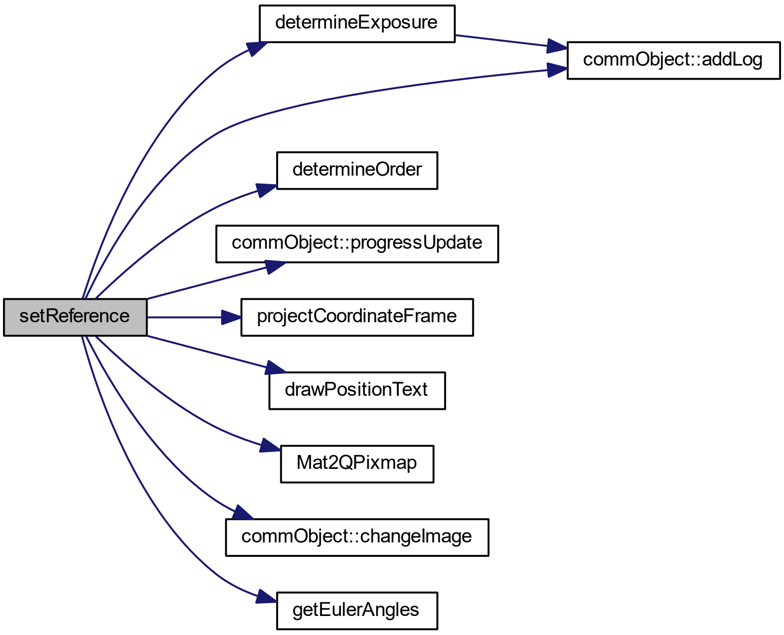
Determine the initial position of the object that serves as reference point or as ground frame origin. Computes the pose 200 times and then averages it. The position and attitude are from now on used as navigation CoSy.

Definition at line 595 of file main.cpp.

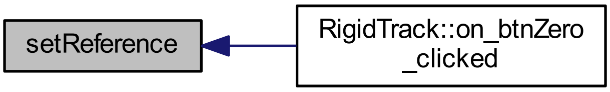
```
{
    //! initialize the variables with starting values
    gotOrder = false;
    posRef = 0;
    eulerRef = 0;
    RmatRef = 0;
    Rvec = RvecOriginal;
    Tvec = TvecOriginal;

    determineExposure();

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

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

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

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

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

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

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

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

    //! Turn on some overlay text so it's clear things are     ===---
    //! working even if there is nothing in the camera's view. ===---
    //! Set some other parameters as well of the camera
    camera->SetTextOverlay(true);
    camera->SetFrameRate(intFrameRate);
    camera->SetIntensity(intIntensity);
    camera->SetIRFilter(true);
    camera->SetContinuousIR(false);
    camera->SetHighPowerMode(false);

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

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

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

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

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

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

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

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

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

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

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

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

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

```

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

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

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

    //! compute the difference between last obtained TVec and the average Value
    //! When it is large the iterative method has not converged properly so it is advised to start the
   setReference() function once again
    double error = norm(posRef) - norm(Tvec);
    if (error > 5.0)
    {
        ss << "Caution, distance between reference position and last position is: " << error << "\n Start
   the set zero routine once again.";
    }
    commObj.addLog(QString::fromStdString(ss.str()));
    commObj.progressUpdate(0);
    return 0;
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.2.14 setUpUDP()

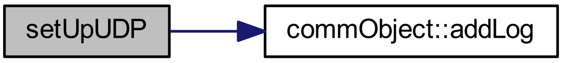
```
void setUpUDP ( )
```

Open the UDP ports for communication.

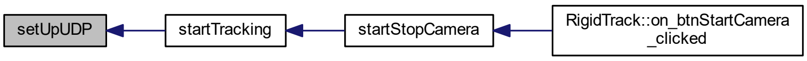
Definition at line 1090 of file main.cpp.

```
{
    //! Initialise the QDataStream that stores the data to be send
    QDataStream out(&datagram, QIODevice::WriteOnly);
    out.setVersion(QDataStream::Qt_4_3);

    //! Create UDP slots
    commObj.addLog("Opening UDP ports.");
    udpSocketObject = new QUdpSocket(0);
    udpSocketObject->connectToHost(IPAdressObject, portObject);
    commObj.addLog("Opened first receiver UDP port.");

    udpSocketSafety = new QUdpSocket(0);
    udpSocketSafety2 = new QUdpSocket(0);

    //! if the safety feature is activated open the udp port
    if (safetyEnable)
    {
        udpSocketSafety->connectToHost(IPAdressSafety, portSafety);
        commObj.addLog("Opened safety UDP port.");
    }

    //! if the second receiver feature is activated open the udp port
    if (safety2Enable)
    {
        udpSocketSafety2->connectToHost(IPAdressSafety2, portSafety2);
        commObj.addLog("Opened second receiver UDP port.");
    }
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.2.15 startStopCamera()

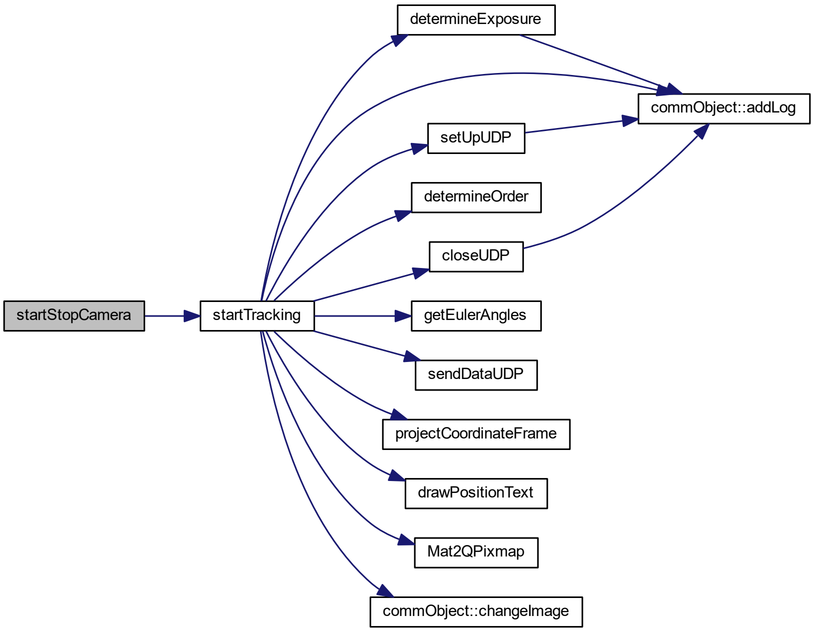
```
void startStopCamera ( )
```

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

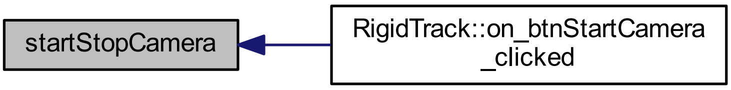
Definition at line 579 of file main.cpp.

```
{
    //! tracking is not running so start it
    if (exitRequested)
    {
        exitRequested = false;
        startTracking();
    }
    else  //!< tracking is currently running, set exitRequest to true so the while loop in startTracking() exits
    {
        exitRequested = true;
    }
}
```

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.

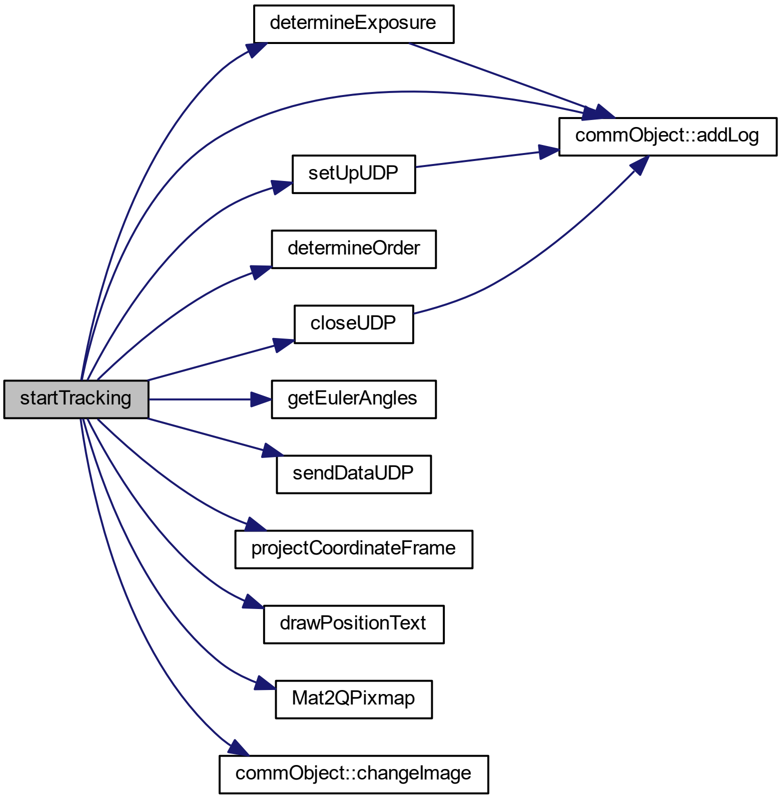
```
                    {


    gotOrder = false; //! The order of points, hence which entry in list_points3d corresponds to
    which in list_points2d is not calculated yet
    Rvec = RvecOriginal; //! Use the value of Rvec that was set in main() as starting value
    for the solvePnP algorithm
    Tvec = TvecOriginal; //! Use the value of Tvec that was set in main() as starting value
    for the solvePnP algorithm
    GetLocalTime(&logDate);  //! Get the current date and time to name the log file

    //! Concat the log file name as followed. The file is saved in the folder /logs in the Rigid Track
    installation folder
    logFileName = "./logs/positionLog_" + QString::number(logDate.wDay) + "_" +
   QString::number(logDate.wMonth) + "_" + QString::number(logDate.wYear);
    logFileName += "_" + QString::number(logDate.wHour) + "_" + QString::number(
   logDate.wMinute) + "_" + QString::number(logDate.wSecond) + ".txt";
    logName = logFileName.toStdString(); //! Convert the QString to a standard string

    determineExposure(); //! Get the exposure where the right amount of markers is
    detected

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

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

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

    //! Get a connected camera
    CameraManager::X().WaitForInitialization();
    Camera *camera = CameraManager::X().GetCamera();

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

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

```

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

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

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

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

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

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

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

 //! Fetch a new frame from the camera
 bool gotTime = false; //! Get the timestamp of the first frame. This time is subtracted from every subseeding frame so the time starts at 0 in the logs
 while (!gotTime) //! While no new frame is received loop
 {
     Frame *frame = camera->GetFrame(); //! Get a new camera frame
     if (frame)  //! There is actually a new frame
     {
         timeFirstFrame = frame->TimeStamp(); //! Get the time stamp for the first frame. It is subtracted for the following frames
         frame->Release();   //! Release the frame so the camera can continue
         gotTime = true; //! Exit the while loop
     }
 }

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

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

     if (frame) //! Did we got a new frame or does the camera still need more time
     {
         framesDropped++; //! Increase by one, if everything is okay it is decreased at the end of the loop again

         //! Only use this frame it the right number of markers is found in the picture
         if (frame->ObjectCount() == numberMarkers)
         {
             //! Get the marker points in 2D in the camera image frame and store them in the list_points2dUnsorted vector
             //! The order of points that come from the camera corresponds to the Y coordinate
             for (int i = 0; i < numberMarkers; i++)
             {
                 cObject *obj = frame->Object(i);
                 list_points2dUnsorted[i] = cv::Point2d(obj->X(), obj->Y());
             }

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

```

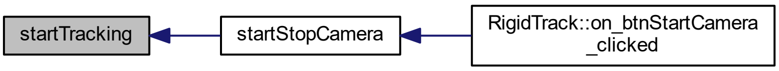
```
                //! Sort the 2d points with the correct indices as found in the preceeding order determination algorithm
                for (int w = 0; w < numberMarkers; w++)
                {
                    list_points2d[w] = list_points2dUnsorted[pointOrderIndices[w]]; //! pointOrderIndices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

                //! The first time the 2D-3D corresspondence was determined with gotOrder was okay.
                //! But this order can change as the object moves and the marker objects appear in a
                //! different order in the frame->Object() array.
                //! The solution is that: When a marker point (in the camera image, hence in 2D) was at
                //! a position then it wont move that much from one frame to the other.
                //! So for the new frame we take a marker object and check which marker was closest this point
                //! in the old image frame? This is probably the same (true) marker. And we do that for every other marker as well.
                //! When tracking is good and no frames are dropped because of missing markers this should work every frame.
                for (int j = 0; j < numberMarkers; j++)
                {
                    minPointDistance = 5000; //! The sum of point distances is set to something unrealistic large
                    for (int k = 0; k < numberMarkers; k++)
                    {
                        //! Calculate N_2 norm of unsorted points minus old points
                        currentPointDistance = norm(list_points2dUnsorted[pointOrderIndices[j]] - list_points2dOld[k]);
                        //! If the norm is smaller than minPointDistance the correspondence is more likely to be correct
                        if (currentPointDistance < minPointDistance)
                        {
                            //! Update the array that saves the new point order
                            minPointDistance = currentPointDistance;
                            pointOrderIndicesNew[j] = k;
                        }
                    }
                }

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

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

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

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

                //! Increase the framesDropped variable if accuracy of tracking is too bad
                if (projectionError > 5)
                {
                    framesDropped++;
                }
                else
                {
                    framesDropped = 0;  //! Set number of subsequent frames dropped to zero because error is small enough and no marker was missing
                }

                //! Get the min and max values from TVec for sanity check
                minMaxLoc(Tvec.at<double>(2), &minValue, &maxValue);

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

                //! Next step is the transformation from camera CoSy to navigation CoSy
                //! Compute the relative object position from the reference position to the current one
                //! given in the camera CoSy: \f$ T_C^{NM} = Tvec - Tvec_{Ref} \f$
                subtract(Tvec, posRef, position);

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

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

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

                //! Euler Angles, finally
                getEulerAngles(Rmat, eulerAngles); //! Get the euler angles from the rotation matrix
                eulerAngles[2] += headingOffset; //! Add the heading offset to the heading angle

                //! Compute the velocity with finite differences. Only use is the log file. It is done here because the more precise time stamp can be used
                frameTime = frame->TimeStamp() - timeOld;   //! Time between the old frame and the current frame
                timeOld = frame->TimeStamp();    //! Set the old frame time to the current  one
                velocity[0] = (position[0] - positionOld[0]) / frameTime; //! Calculate the x velocity with finite differences
                velocity[1] = (position[1] - positionOld[1]) / frameTime; //! Calculate the y velocity with finite differences
                velocity[2] = (position[2] - positionOld[2]) / frameTime; //! Calculate the z velocity with finite differences
                positionOld = position;  //! Set the old position to the current one for next frame velocity calcuation

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

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

            //! Check if the position and euler angles are below the allowed value, if yes send OKAY signal (1), if not send shutdown signal (0)
            //! Absolute x, y and z position in navigation CoSy must be smaller than the allowed distance
            if (safetyEnable)
            {
                if ((abs(position[0]) < safetyBoxLength && abs(position[1]) < safetyBoxLength && abs(position[2]) < safetyBoxLength))
                {
                    //! Absolute Euler angles must be smaller than allowed value. Heading is not considered

                    if ((abs(eulerAngles[0]) < safetyAngle && abs(eulerAngles[1]) < safetyAngle))
                    {
                        //! Send the OKAY signal to the desired computer every 5th time
                        if (v == 5) {
                            data.setNum((int)(1));
                            udpSocketSafety->write(data); //! Send the 1
                            v = 0; //! reset the counter that is needed for decimation to every 5th time step
                        }
                    }
                    //! The euler angles of the object exceeded the allowed euler angles, send the shutdown signal (0)
```

```
                    else
                    {
                        data.setNum((int)(0));  //! Send the shutdown signal, a 0
                        udpSocketSafety->write(data);
                        commObj.addLog("Object exceeded allowed Euler angles, shutdown signal sent."); //! Inform the user

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

                }
            }

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

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

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

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

            //! Project the marker points from 3D to the camera image frame (2d) with the computed pose
            projectPoints(list_points3d, Rvec, Tvec, cameraMatrix, distCoeffs, list_points2d);
            for (int i = 0; i < numberMarkers; i++)
            {
                //! Draw a circle around the projected points so the result can be better compared to the real marker position
                //! In the resulting picture those are the red dots
                circle(cFrame, Point(list_points2d[i].x, list_points2d[i].y), 3, Scalar(225, 0, 0), 3);
            }

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

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

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



6.6.2.17 testAlgorithms()

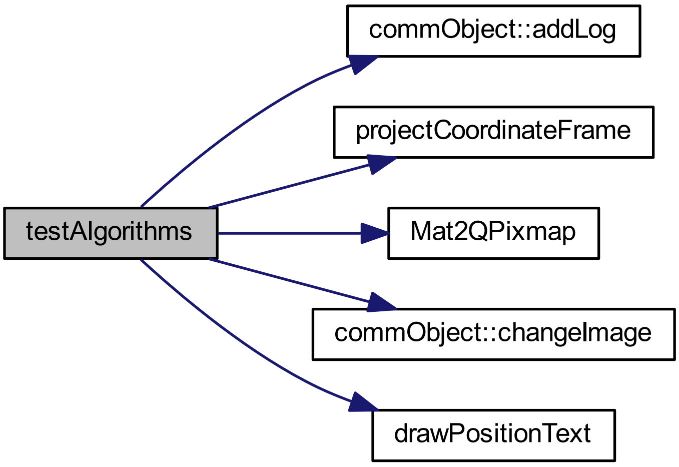
```
void testAlgorithms ( )
```

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.

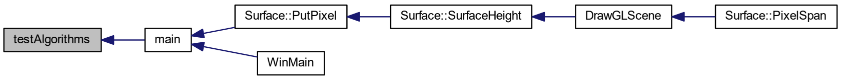
Definition at line 952 of file main.cpp.

```
{

    int _methodPNP;

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

    RvecOriginal = Rvec;
    TvecOriginal = Tvec;

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

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

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

    projectCoordinateFrame(cFrame);

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

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

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


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

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

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

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

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

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

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

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

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

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

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

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

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

    Rvec = RvecOriginal;
    Tvec = TvecOriginal;

}
```

Here is the call graph for this function:



Here is the caller graph for this function:



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 IPAdressObject

```
QHostAddress IPAdressObject
```

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
```

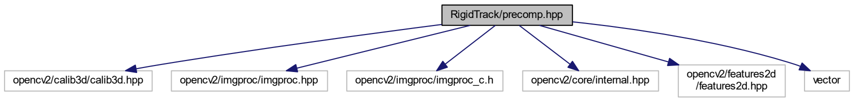
is the safety feature enabled

Definition at line 70 of file main.cpp.

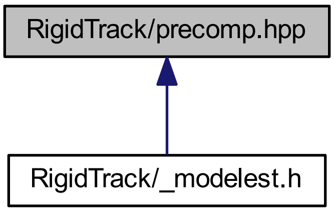
6.7 RigidTrack/precomp.hpp File Reference

```
#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:


RigidTrack/precomp.hpp
opencv2/calib3d/calib3d.hpp
opencv2/imgproc/imgproc.hpp
opencv2/imgproc/imgproc_c.h
opencv2/core/internal.hpp
opencv2/features2d
/features2d.hpp
vector

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 /<

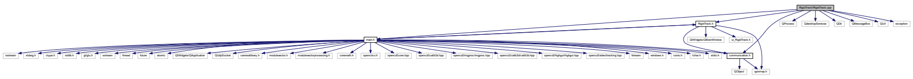
Definition at line 5 of file resource.h.

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

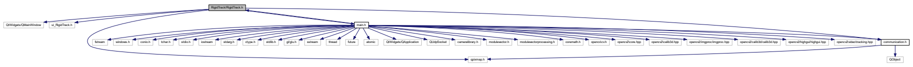
April, 8th 2017

6.10 RigidTrack/RigidTrack.h File Reference

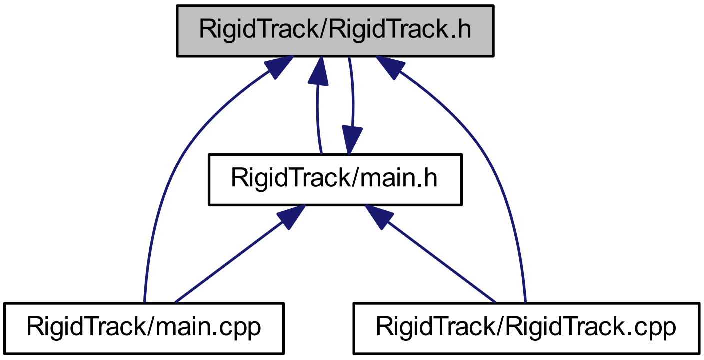
Rigid Track GUI source header with Qt Signals and Slots.

```
#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

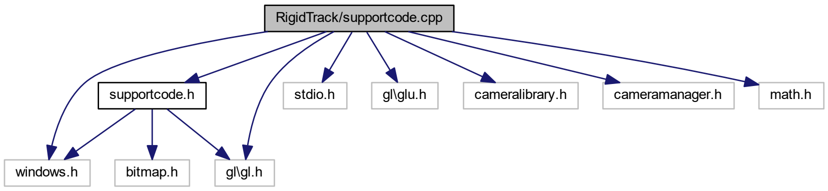
Date

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 nCmd↩ Show)

  /< 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

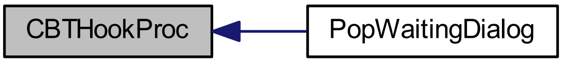
6.11.1.1 CBTHookProc()

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

Definition at line 575 of file supportcode.cpp.

```
{
    if (nCode < 0)
        return CallNextHookEx(hHook, nCode, wParam, lParam);

    if (nCode == HCBT_ACTIVATE)
    {
        HWND hWnd = reinterpret_cast<HWND>(wParam);
        SetWindowText(GetDlgItem(hWnd, IDOK), TEXT("Cancel"));
        return 0;
    }

    return CallNextHookEx(hHook, nCode, wParam, lParam);
}
```

Here is the caller graph for this function:



6.11.1.2 CloseWindow()

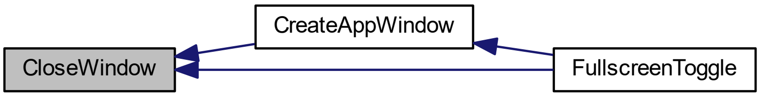
```
GLvoid CloseWindow (
              GLvoid   )
```

/< Properly Kill The Window

Definition at line 192 of file supportcode.cpp.

```
{
    if (gFullscreen)                                        //!/< Are We In Fullscreen Mode?
    {
        ChangeDisplaySettings(NULL,0);                      //!/< If So Switch Back To The Desktop
        ShowCursor(TRUE);                                   //!/< Show Mouse Pointer
    }

    if (hRC)                                            //!/< Do We Have A Rendering Context?
    {
        if (!wglMakeCurrent(NULL,NULL))                     //!/< Are We Able To Release The DC And RC Contexts?
        {
            MessageBox(NULL,L"Release Of DC And RC Failed.",L"SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
        }

        if (!wglDeleteContext(hRC))                         //!/< Are We Able To Delete The RC?
        {
            MessageBox(NULL, L"Release Rendering Context Failed.", L"SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
        }
        hRC=NULL;                                           //!/< Set RC To NULL
    }

    if (hDC && !ReleaseDC(hWnd,hDC))                     //!/< Are We Able To Release The DC
    {
        MessageBox(NULL, L"Release Device Context Failed.", L"SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
        hDC=NULL;                                           //!/< Set DC To NULL
    }

    if (hWnd && !DestroyWindow(hWnd))                       //!/< Are We Able To Destroy The Window?
    {
        MessageBox(NULL, L"Could Not Release hWnd.", L"SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
        hWnd=NULL;                                          //!/< Set hWnd To NULL
    }

    if (!UnregisterClass(L"OpenGL",hInstance))           //!/< Are We Able To Unregister Class
    {
        MessageBox(NULL, L"Could Not Unregister Class.", L"SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
        hInstance=NULL;                                        //!/< Set hInstance To NULL
    }
}
```

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 )
```

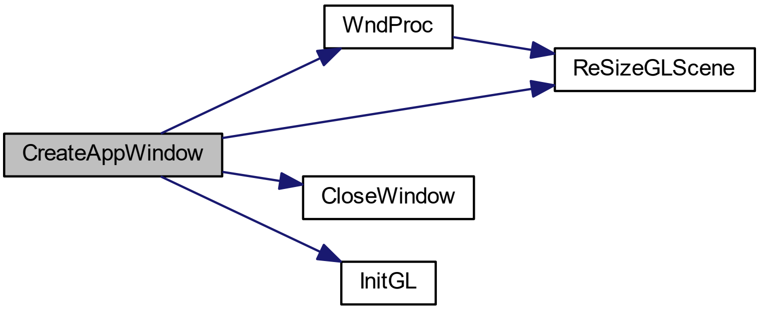
Definition at line 244 of file supportcode.cpp.

```
{
    windowWidth  = width;
    windowHeight = height;
    windowName   = title;

    GLuint      PixelFormat;            //!< Holds The Results After Searching For A Match
    WNDCLASS    wc;                     //!< Windows Class Structure
    DWORD       dwExStyle;              //!< Window Extended Style
    DWORD       dwStyle;                //!< Window Style
    RECT        WindowRect;             //!< Grabs Rectangle Upper Left / Lower Right Values
    WindowRect.left=(long)0;            //!< Set Left Value To 0
    WindowRect.right=(long)width;       //!< Set Right Value To Requested Width
    WindowRect.top=(long)0;             //!< Set Top Value To 0
    WindowRect.bottom=(long)height;     //!< Set Bottom Value To Requested Height

    gFullscreen=fullscreenflag;          //!< Set The Global Fullscreen Flag

    HICON hIcon = (HICON)LoadImage(0,IDI_WINLOGO,IMAGE_ICON,0,0,LR_SHARED);

    hInstance           = GetModuleHandle(NULL);                //!< Grab An Instance For Our
   Window
    wc.style             = CS_HREDRAW | CS_VREDRAW | CS_OWNDC;   //!< Redraw On Size, And Own DC For
   Window.
    wc.lpfnWndProc       = (WNDPROC) WndProc;                  //!< WndProc Handles Messages
    wc.cbClsExtra        = 0;                                    //!< No Extra Window Data
    wc.cbWndExtra        = 0;                                    //!< No Extra Window Data
    wc.hInstance         = hInstance;                           //!< Set The Instance
    wc.hIcon             = 0;
    wc.hCursor           = LoadCursor(NULL, IDC_ARROW);          //!< Load The Arrow Pointer
    wc.hbrBackground     = NULL;                                 //!< No Background Required For GL
    wc.lpszMenuName      = NULL;                                 //!< We Don't Want A Menu
    wc.lpszClassName     = L"OpenGL";                                //!< Set The Class Name

    if (!RegisterClass(&wc))                                     //!< Attempt To Register The Window Class
    {
        MessageBox(NULL, L"Failed To Register The Window Class.",L"ERROR",MB_OK|MB_ICONEXCLAMATION);
        return FALSE;                                            //!< Return FALSE
    }

    if (gFullscreen)                                              //!< Attempt Fullscreen Mode?
    {
        DEVMODE dmScreenSettings;                                //!< Device Mode
        memset(&dmScreenSettings,0,sizeof(dmScreenSettings));    //!< Makes Sure Memory's Cleared
        dmScreenSettings.dmSize=sizeof(dmScreenSettings);        //!< Size Of The Devmode Structure
```

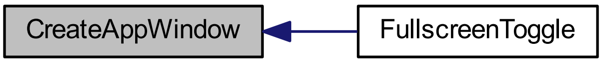
```
     dmScreenSettings.dmPelsWidth    = width;                //!< Selected Screen Width
     dmScreenSettings.dmPelsHeight   = height;               //!< Selected Screen Height
     dmScreenSettings.dmBitsPerPel   = bits;                 //!< Selected Bits Per Pixel
     dmScreenSettings.dmFields=DM_BITSPERPEL|DM_PELSWIDTH|DM_PELSHEIGHT;

     //!< Try To Set Selected Mode And Get Results.  NOTE: CDS_FULLSCREEN Gets Rid Of Start Bar.
     if (ChangeDisplaySettings(&dmScreenSettings,CDS_FULLSCREEN)!=DISP_CHANGE_SUCCESSFUL)
     {
         //!< If The Mode Fails, Offer Two Options.  Quit Or Use Windowed Mode.
         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)
         {
             gFullscreen=FALSE;       //!< Windowed Mode Selected.  Fullscreen = FALSE
         }
         else
         {
             //!< Pop Up A Message Box Letting User Know The Program Is Closing.
             MessageBox(NULL, L"Program Will Now Close.",L" RROR",MB_OK|MB_ICONSTOP);
             return FALSE;                                   //!< Return FALSE
         }
     }
 }

 if (gFullscreen)                                             //!< Are We Still In Fullscreen Mode?
 {
     dwExStyle=WS_EX_APPWINDOW;                              //!< Window Extended Style
     dwStyle=WS_POPUP;                                       //!< Windows Style
     ShowCursor(FALSE);                                      //!< Hide Mouse Pointer
 }
 else
 {
     dwExStyle=WS_EX_APPWINDOW | WS_EX_WINDOWEDGE;           //!< Window Extended Style
     dwStyle=WS_OVERLAPPEDWINDOW;                            //!< Windows Style
 }

 AdjustWindowRectEx(&WindowRect, dwStyle, FALSE, dwExStyle);     //!< Adjust Window To True Requested Size

 //!< Create The Window
 if (!(hWnd=CreateWindowEx(  dwExStyle,                          //!< Extended Style For The Window
     L" penGL",                          //!< Class Name
                             L"title",                               //!< Window Title
                             dwStyle |                           //!< Defined Window Style
                             WS_CLIPSIBLINGS |                   //!< Required Window Style
                             WS_CLIPCHILDREN,                    //!< Required Window Style
                             0, 0,                               //!< Window Position
                             WindowRect.right-WindowRect.left,   //!< Calculate Window Width
                             WindowRect.bottom-WindowRect.top,   //!< Calculate Window Height
                             NULL,                               //!< No Parent Window
                             NULL,                               //!< No Menu
                             hInstance,                         //!< Instance
                             NULL)))                             //!< Dont Pass Anything To WM_CREATE
 {
     CloseWindow();                               //!< Reset The Display
     MessageBox(NULL,L" indow Creation Error.",L"Error", MB_OK|MB_ICONEXCLAMATION);
     return FALSE;                               //!< Return FALSE
 }

 static  PIXELFORMATDESCRIPTOR pfd=              //!< pfd Tells Windows How We Want Things To Be
 {
     sizeof(PIXELFORMATDESCRIPTOR),              //!< Size Of This Pixel Format Descriptor
     1,                                          //!< Version Number
     PFD_DRAW_TO_WINDOW |                        //!< Format Must Support Window
     PFD_SUPPORT_OPENGL |                        //!< Format Must Support OpenGL
     PFD_DOUBLEBUFFER,                           //!< Must Support Double Buffering
     PFD_TYPE_RGBA,                              //!< Request An RGBA Format
     bits,                                       //!< Select Our Color Depth
     0, 0, 0, 0, 0, 0,                           //!< Color Bits Ignored
     0,                                          //!< No Alpha Buffer
     0,                                          //!< Shift Bit Ignored
     0,                                          //!< No Accumulation Buffer
     0, 0, 0, 0,                                 //!< Accumulation Bits Ignored
     16,                                         //!< 16Bit Z-Buffer (Depth Buffer)
     0,                                          //!< No Stencil Buffer
     0,                                          //!< No Auxiliary Buffer
     PFD_MAIN_PLANE,                             //!< Main Drawing Layer
     0,                                          //!< Reserved
     0, 0, 0                                     //!< Layer Masks Ignored
 };

 if (!(hDC=GetDC(hWnd)))                          //!< Did We Get A Device Context?
 {
     CloseWindow();                               //!< Reset The Display
     MessageBox(NULL,L" an’t Create A GL Device Context.",L" RROR",MB_OK|MB_ICONEXCLAMATION);
     return FALSE;                               //!< Return FALSE
 }
```

```

    if (!(PixelFormat=ChoosePixelFormat(hDC,&pfd)))  //!< Did Windows Find A Matching Pixel Format?
    {
        CloseWindow();                               //!< Reset The Display
        MessageBox(NULL, L" an't Find A Suitable PixelFormat.", L" RROR",MB_OK|MB_ICONEXCLAMATION);
        return FALSE;                               //!< Return FALSE
    }

    if(!SetPixelFormat(hDC,PixelFormat,&pfd))        //!< Are We Able To Set The Pixel Format?
    {
        CloseWindow();                               //!< Reset The Display
        MessageBox(NULL, L" an't Set The PixelFormat.", L" RROR",MB_OK|MB_ICONEXCLAMATION);
        return FALSE;                               //!< Return FALSE
    }

    if (!(hRC=wglCreateContext(hDC)))             //!< Are We Able To Get A Rendering Context?
    {
        CloseWindow();                               //!< Reset The Display
        MessageBox(NULL, L" an't Create A GL Rendering Context.", L" RROR",MB_OK|MB_ICONEXCLAMATION);
        return FALSE;                               //!< Return FALSE
    }

    if(!wglMakeCurrent(hDC,hRC))                  //!< Try To Activate The Rendering Context
    {
        CloseWindow();                               //!< Reset The Display
        MessageBox(NULL, L" an't Activate The GL Rendering Context.", L" RROR",MB_OK|MB_ICONEXCLAMATION);
        return FALSE;                               //!< Return FALSE
    }

    ShowWindow(hWnd,SW_SHOW);                       //!< Show The Window
    SetForegroundWindow(hWnd);                      //!< Slightly Higher Priority
    SetFocus(hWnd);                                 //!< Sets Keyboard Focus To The Window
    ReSizeGLScene(width, height);                  //!< Set Up Our Perspective GL Screen

    if (!InitGL())                                    //!< Initialize Our Newly Created GL Window
    {
        CloseWindow();                               //!< Reset The Display
        MessageBox(NULL, L" nitialization Failed.", L" RROR",MB_OK|MB_ICONEXCLAMATION);
        return FALSE;                               //!< Return FALSE
    }

    return TRUE;                                    //!< Success
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



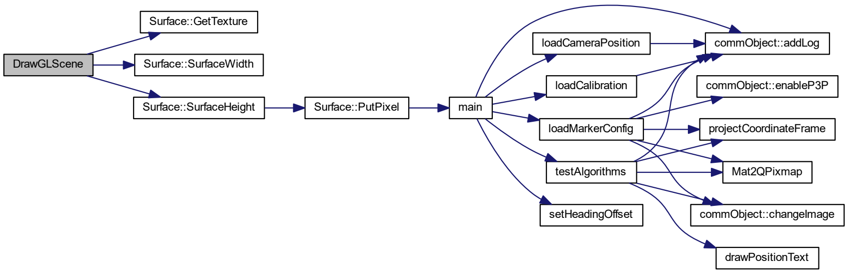
6.11.1.4 DrawGLScene()

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

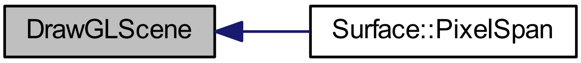
Definition at line 76 of file supportcode.cpp.

```
{
    if(surf==NULL)
        return true;

    static bool frameThrottler = true;
    frameThrottler=!frameThrottler;

    if(frameThrottler)  //!/<== Only display every other frame in case VSYNC is enabled.  Otherwise
        return true;    //!/<== application would get behind

    int pixelWidth = width;
    int pixelHeight= height;

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); //!/< Clear The Screen And The Depth Buffer
    glLoadIdentity();                                   //!/< Reset The View

    static GLuint ff = 0;

    int tex = surf->GetTexture();
    if(tex==0)
        tex=ff;
    else
        ff=tex;
    if(tex==0)
        return true;

    glBindTexture(GL_TEXTURE_2D, tex);

    glBegin(GL_QUADS);
        glTexCoord2f(0.0f, 0.0f); glVertex3f( 0,     0, 0);
        glTexCoord2f((GLfloat)pixelWidth/(GLfloat)surf->SurfaceWidth(), 0.0f); glVertex3f( (
  GLfloat)gWindowWidth,  0, 0);
        glTexCoord2f((GLfloat)pixelWidth/(GLfloat)surf->SurfaceWidth(), (GLfloat)pixelHeight/(
  GLfloat)surf->SurfaceHeight()); glVertex3f( (GLfloat)gWindowWidth,(GLfloat)
  gWindowHeight, 0);
        glTexCoord2f(0.0f, (GLfloat)pixelHeight/(GLfloat)surf->SurfaceHeight()); glVertex3f( 0
  , (GLfloat) gWindowHeight, 0);
    glEnd();

    SwapBuffers(hDC);                    //!/< Swap Buffers (Double Buffering)

    return true;                                        //!/< Keep Going
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



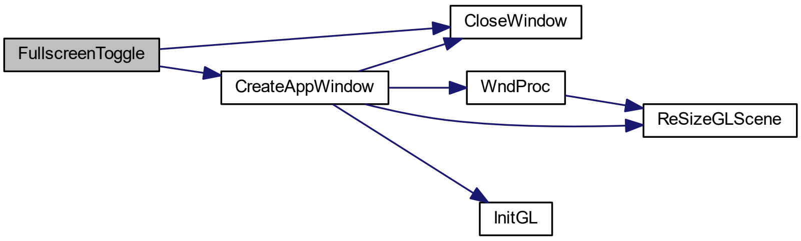
6.11.1.5 FullscreenToggle()

```
bool FullscreenToggle ( )
```

Definition at line 536 of file supportcode.cpp.

```
{
    keys[VK_F1]=FALSE;                    //!/< If So Make Key FALSE
    CloseWindow();                         //!/< Kill Our Current Window
    gFullscreen=!gFullscreen;           //!/< Toggle Fullscreen / Windowed Mode

    if (!CreateAppWindow(windowName,windowWidth,windowHeight,32,gFullscreen))
    {
        MessageBox(0, L"Unable to toggle to full screen",L"Error", MB_OK);
        return 1;
    }

    return true;
}
```

Here is the call graph for this function:



6.11.1.6 InitGL()

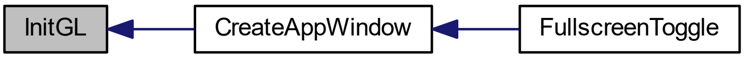
```
int InitGL (
            GLvoid  )
```

/< All Setup For OpenGL Goes Here

Definition at line 62 of file supportcode.cpp.

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

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.

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

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.

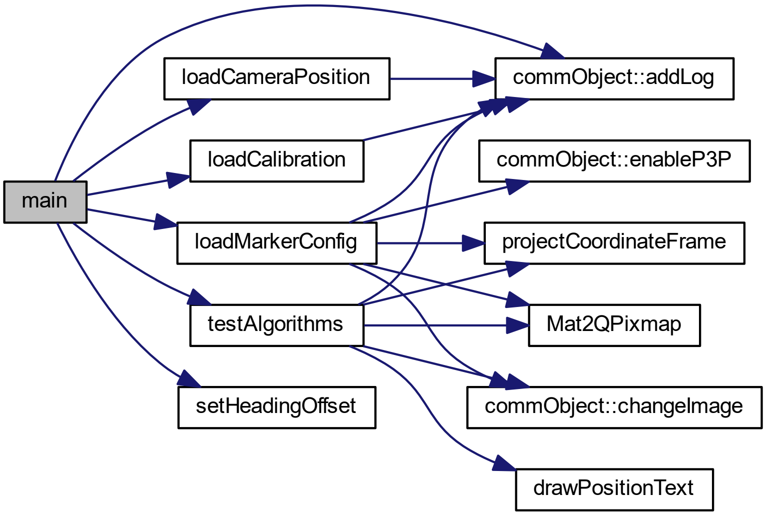
Parameters

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

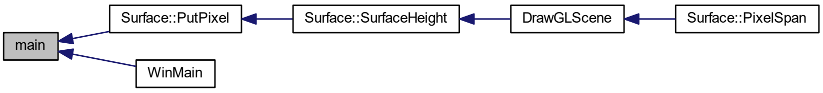
Definition at line 156 of file main.cpp.

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

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

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

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

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

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

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

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

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

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

    return a.exec();
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



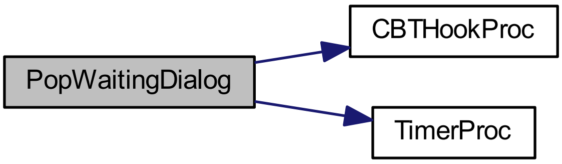
6.11.1.9 PopWaitingDialog()

```
bool PopWaitingDialog ( )
```

Definition at line 615 of file supportcode.cpp.

```
{
    //!/<== hook in so we can create a message box that has only a ’Cancel’ button ==--

    hHook = SetWindowsHookEx(WH_CBT, reinterpret_cast<HOOKPROC>(&
  CBTHookProc), NULL, GetCurrentThreadId());

    UINT_PTR nTimer = SetTimer(0, 100, 3000,(TIMERPROC) TimerProc);
    int iResult = MessageBox( 0, L"waiting for connected cameras...", L"Camera Initialization", MB_OK );

    if( iResult == IDOK )
    {
        //!/<== user has clicked the cancel button ==--
        UnhookWindowsHookEx(hHook);
        return false;
    }

    KillTimer(0, nTimer);

    return true;
}
```

Here is the call graph for this function:



6.11.1.10 PumpMessages()

```
bool PumpMessages ( )
```

Definition at line 551 of file supportcode.cpp.

```
{
    MSG msg;

    if (PeekMessage(&msg,NULL,0,0,PM_REMOVE))   //!/< Is There A Message Waiting?
    {
        if (msg.message==WM_QUIT)               //!/< Have We Received A Quit Message?
            return false;
        else                                    //!/< If Not, Deal With Window Messages
        {
            TranslateMessage(&msg);             //!/< Translate The Message
            DispatchMessage(&msg);              //!/< Dispatch The Message
        }
    }

    return true;
}
```

6.11.1.11 ReSizeGLScene()

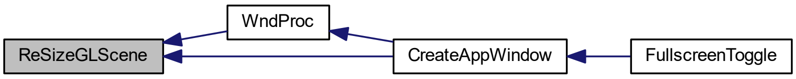
```
GLvoid ReSizeGLScene (
              GLsizei width,
              GLsizei height )
```

/< Resize And Initialize The GL Window

Definition at line 42 of file supportcode.cpp.

```
{
    if (height==0)                                      //!/< Prevent A Divide By Zero By
    {
        height=1;                                       //!/< Making Height Equal One
    }

    glViewport(0,0,width,height);                       //!/< Reset The Current Viewport

    glMatrixMode(GL_PROJECTION);                        //!/< Select The Projection Matrix
    glLoadIdentity();                                   //!/< Reset The Projection Matrix

    gWindowWidth = width;
    gWindowHeight = height;

    glOrtho(0,gWindowWidth,gWindowHeight,0,100,-100);
    glMatrixMode(GL_MODELVIEW);                         //!/< Select The Modelview Matrix
    glLoadIdentity();                                   //!/< Reset The Modelview Matrix
}
```

Here is the caller graph for this function:



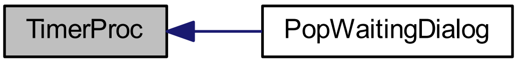
6.11.1.12 TimerProc()

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

Definition at line 590 of file supportcode.cpp.

```
{
    CameraLibrary::CameraList list;

    bool found = false;

    for( int i=0; i<list.Count(); i++ )
    {
        if( list[i].State()==CameraLibrary::Initialized )
        {
            found = true;
        }
    }

    if(found==true)
    {
        HWND hWndActive = GetActiveWindow();

        if( hWndActive!=0 )
        {
            SendMessage(hWndActive, WM_COMMAND, IDCANCEL, 0);
        }
    }
}
```

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

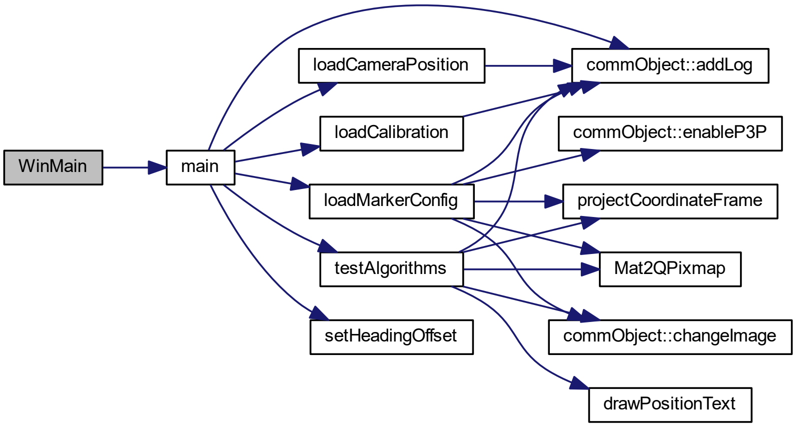
Parameters

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

Definition at line 528 of file supportcode.cpp.

```
{
    return main(0,0);
}
```

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

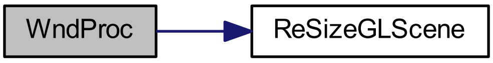
Parameters

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

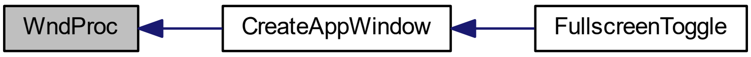
Definition at line 118 of file supportcode.cpp.

```
{
    switch (uMsg)                                   //!/< Check For Windows Messages
    {
        case WM_ACTIVATE:                           //!/< Watch For Window Activate Message
        {
            if (!HIWORD(wParam))                    //!/< Check Minimization State
            {
                gActive=TRUE;                        //!/< Program Is Active
            }
            else
            {
                gActive=FALSE;                       //!/< Program Is No Longer Active
            }

            return 0;                               //!/< Return To The Message Loop
        }

        case WM_POWERBROADCAST:
            if(wParam == PBT_APMSUSPEND)
            {
                CameraLibrary::CameraManager::X().PrepareForSuspend();
            }
            if(wParam == PBT_APMRESUMEAUTOMATIC)
            {
                CameraLibrary::CameraManager::X().ResumeFromSuspend();
            }
            break;

        case WM_SYSCOMMAND:                         //!/< Intercept System Commands
        {
            switch (wParam)                         //!/< Check System Calls
            {
                case SC_SCREENSAVE:                 //!/< Screensaver Trying To Start?
                case SC_MONITORPOWER:               //!/< Monitor Trying To Enter Powersave?
                return 0;                           //!/< Prevent From Happening
            }
            break;                                  //!/< Exit
        }

        case WM_CLOSE:                              //!/< Did We Receive A Close Message?
        {
            PostQuitMessage(0);                     //!/< Send A Quit Message
            return 0;                               //!/< Jump Back
        }

        case WM_KEYDOWN:                            //!/< Is A Key Being Held Down?
        {
            keys[wParam] = TRUE;                    //!/< If So, Mark It As TRUE
            return 0;                               //!/< Jump Back
        }

        case WM_KEYUP:                              //!/< Has A Key Been Released?
        {
            keys[wParam] = FALSE;                   //!/< If So, Mark It As FALSE
            return 0;                               //!/< Jump Back
        }
        case WM_MOVE:
            return 0;                               //!/< Jump Back
        case WM_PAINT:
            return 0;
        case WM_SIZE:                               //!/< Resize The OpenGL Window
        {
            ReSizeGLScene(LOWORD(lParam),HIWORD(lParam));  //!/< LoWord=Width, HiWord=Height
            return 0;                               //!/< Jump Back
        }
    }

    //!/< Pass All Unhandled Messages To DefWindowProc
    return DefWindowProc(hWnd,uMsg,wParam,lParam);
}
```

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

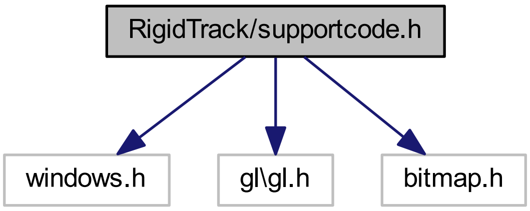
```
int windowWidth
```

Definition at line 240 of file supportcode.cpp.

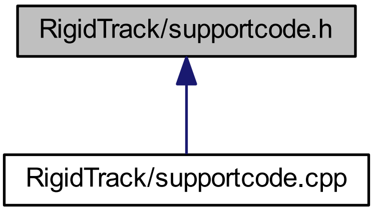
6.12 RigidTrack/supportcode.h File Reference

```
#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()

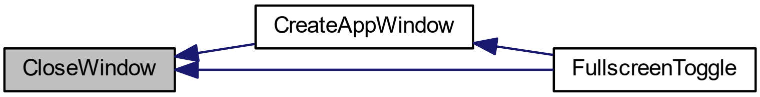
```
GLvoid CloseWindow (
              GLvoid   )
```

/< Properly Kill The Window

Definition at line 192 of file supportcode.cpp.

```
{
    if (gFullscreen)                                        //!/< Are We In Fullscreen Mode?
    {
        ChangeDisplaySettings(NULL,0);                      //!/< If So Switch Back To The Desktop
        ShowCursor(TRUE);                                   //!/< Show Mouse Pointer
    }

    if (hRC)                                            //!/< Do We Have A Rendering Context?
    {
        if (!wglMakeCurrent(NULL,NULL))                     //!/< Are We Able To Release The DC And RC Contexts?
        {
            MessageBox(NULL,L"Release Of DC And RC Failed.",L"SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
        }

        if (!wglDeleteContext(hRC))                         //!/< Are We Able To Delete The RC?
        {
            MessageBox(NULL, L"Release Rendering Context Failed.", L"SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
        }
        hRC=NULL;                                           //!/< Set RC To NULL
    }

    if (hDC && !ReleaseDC(hWnd,hDC))                    //!/< Are We Able To Release The DC
    {
        MessageBox(NULL, L"Release Device Context Failed.", L"SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
        hDC=NULL;                                           //!/< Set DC To NULL
    }

    if (hWnd && !DestroyWindow(hWnd))                       //!/< Are We Able To Destroy The Window?
    {
        MessageBox(NULL, L"Could Not Release hWnd.", L"SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
        hWnd=NULL;                                          //!/< Set hWnd To NULL
    }

    if (!UnregisterClass(L"OpenGL",hInstance))             //!/< Are We Able To Unregister Class
    {
        MessageBox(NULL, L"Could Not Unregister Class.", L"SHUTDOWN ERROR",MB_OK | MB_ICONINFORMATION);
        hInstance=NULL;                                       //!/< Set hInstance To NULL
    }
}
```

Here is the caller graph for this function:



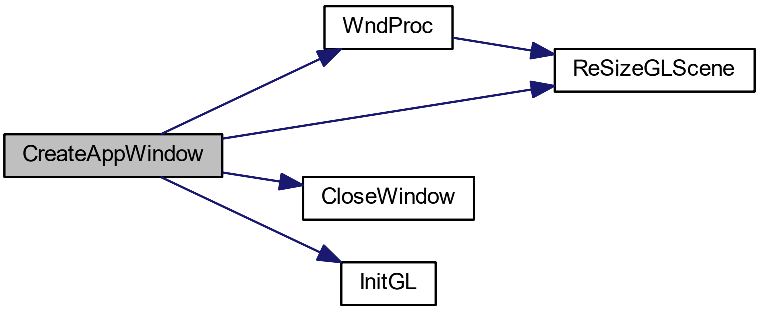
6.12.2.2 CreateAppWindow()

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

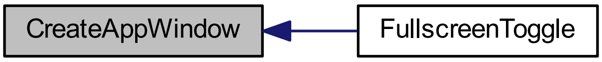
Definition at line 244 of file supportcode.cpp.

```
{
    windowWidth  = width;
    windowHeight = height;
    windowName   = title;

    GLuint      PixelFormat;            //!< Holds The Results After Searching For A Match
    WNDCLASS    wc;                     //!< Windows Class Structure
    DWORD       dwExStyle;              //!< Window Extended Style
    DWORD       dwStyle;                //!< Window Style
    RECT        WindowRect;             //!< Grabs Rectangle Upper Left / Lower Right Values
    WindowRect.left=(long)0;            //!< Set Left Value To 0
    WindowRect.right=(long)width;       //!< Set Right Value To Requested Width
    WindowRect.top=(long)0;             //!< Set Top Value To 0
    WindowRect.bottom=(long)height;     //!< Set Bottom Value To Requested Height

    gFullscreen=fullscreenflag;          //!< Set The Global Fullscreen Flag

    HICON hIcon = (HICON)LoadImage(0,IDI_WINLOGO,IMAGE_ICON,0,0,LR_SHARED);

    hInstance          = GetModuleHandle(NULL);                //!< Grab An Instance For Our Window
    wc.style            = CS_HREDRAW | CS_VREDRAW | CS_OWNDC;   //!< Redraw On Size, And Own DC For Window.
    wc.lpfnWndProc      = (WNDPROC) WndProc;                 //!< WndProc Handles Messages
    wc.cbClsExtra       = 0;                                    //!< No Extra Window Data
    wc.cbWndExtra       = 0;                                    //!< No Extra Window Data
    wc.hInstance        = hInstance;                           //!< Set The Instance
    wc.hIcon            = 0;
    wc.hCursor          = LoadCursor(NULL, IDC_ARROW);          //!< Load The Arrow Pointer
    wc.hbrBackground    = NULL;                                 //!< No Background Required For GL
    wc.lpszMenuName     = NULL;                                 //!< We Don't Want A Menu
    wc.lpszClassName    = L"OpenGL";                                //!< Set The Class Name

    if (!RegisterClass(&wc))                                    //!< Attempt To Register The Window Class
    {
        MessageBox(NULL, L"Failed To Register The Window Class.",L"ERROR",MB_OK|MB_ICONEXCLAMATION);
        return FALSE;                                           //!< Return FALSE
    }

    if (gFullscreen)                                             //!< Attempt Fullscreen Mode?
    {
        DEVMODE dmScreenSettings;                               //!< Device Mode
        memset(&dmScreenSettings,0,sizeof(dmScreenSettings));   //!< Makes Sure Memory's Cleared
        dmScreenSettings.dmSize=sizeof(dmScreenSettings);       //!< Size Of The Devmode Structure
        dmScreenSettings.dmPelsWidth    = width;                //!< Selected Screen Width
        dmScreenSettings.dmPelsHeight   = height;               //!< Selected Screen Height
        dmScreenSettings.dmBitsPerPel   = bits;                 //!< Selected Bits Per Pixel
        dmScreenSettings.dmFields=DM_BITSPERPEL|DM_PELSWIDTH|DM_PELSHEIGHT;

        //!< Try To Set Selected Mode And Get Results.  NOTE: CDS_FULLSCREEN Gets Rid Of Start Bar.
        if (ChangeDisplaySettings(&dmScreenSettings,CDS_FULLSCREEN)!=DISP_CHANGE_SUCCESSFUL)
        {
            //!< If The Mode Fails, Offer Two Options.  Quit Or Use Windowed Mode.
            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)
            {
                gFullscreen=FALSE;       //!< Windowed Mode Selected.  Fullscreen = FALSE
            }
            else
            {
                //!< Pop Up A Message Box Letting User Know The Program Is Closing.
                MessageBox(NULL, L"Program Will Now Close.",L" RROR",MB_OK|MB_ICONSTOP);
                return FALSE;                                   //!< Return FALSE
            }
        }
    }

    if (gFullscreen)                                             //!< Are We Still In Fullscreen Mode?
    {
        dwExStyle=WS_EX_APPWINDOW;                              //!/< Window Extended Style
        dwStyle=WS_POPUP;                                       //!/< Windows Style
        ShowCursor(FALSE);                                      //!/< Hide Mouse Pointer
    }
    else
    {
        dwExStyle=WS_EX_APPWINDOW | WS_EX_WINDOWEDGE;           //!/< Window Extended Style
        dwStyle=WS_OVERLAPPEDWINDOW;                            //!/< Windows Style
    }

    AdjustWindowRectEx(&WindowRect, dwStyle, FALSE, dwExStyle);     //!/< Adjust Window To True Requested
   Size

    //!/< Create The Window
    if (!(hWnd=CreateWindowEx(  dwExStyle,                          //!/< Extended Style For The Window
        L" penGL",                          //!/< Class Name
                                L"title",                               //!/< Window Title
                                dwStyle |                           //!/< Defined Window Style
                                WS_CLIPSIBLINGS |                   //!/< Required Window Style
                                WS_CLIPCHILDREN,                    //!/< Required Window Style
                                0, 0,                               //!/< Window Position
                                WindowRect.right-WindowRect.left,   //!/< Calculate Window Width
                                WindowRect.bottom-WindowRect.top,   //!/< Calculate Window Height
                                NULL,                               //!/< No Parent Window
                                NULL,                               //!/< No Menu
                                hInstance,                         //!/< Instance
                                NULL)))                             //!/< Dont Pass Anything To WM_CREATE
    {
        CloseWindow();                               //!/< Reset The Display
        MessageBox(NULL,L" indow Creation Error.",L"Error", MB_OK|MB_ICONEXCLAMATION);
        return FALSE;                               //!/< Return FALSE
    }

    static  PIXELFORMATDESCRIPTOR pfd=              //!/< pfd Tells Windows How We Want Things To Be
    {
        sizeof(PIXELFORMATDESCRIPTOR),              //!/< Size Of This Pixel Format Descriptor
        1,                                          //!/< Version Number
        PFD_DRAW_TO_WINDOW |                        //!/< Format Must Support Window
        PFD_SUPPORT_OPENGL |                        //!/< Format Must Support OpenGL
        PFD_DOUBLEBUFFER,                           //!/< Must Support Double Buffering
        PFD_TYPE_RGBA,                              //!/< Request An RGBA Format
        bits,                                       //!/< Select Our Color Depth
        0, 0, 0, 0, 0, 0,                           //!/< Color Bits Ignored
        0,                                          //!/< No Alpha Buffer
        0,                                          //!/< Shift Bit Ignored
        0,                                          //!/< No Accumulation Buffer
        0, 0, 0, 0,                                 //!/< Accumulation Bits Ignored
        16,                                         //!/< 16Bit Z-Buffer (Depth Buffer)
        0,                                          //!/< No Stencil Buffer
        0,                                          //!/< No Auxiliary Buffer
        PFD_MAIN_PLANE,                             //!/< Main Drawing Layer
        0,                                          //!/< Reserved
        0, 0, 0                                     //!/< Layer Masks Ignored
    };

    if (!(hDC=GetDC(hWnd)))                          //!/< Did We Get A Device Context?
    {
        CloseWindow();                               //!/< Reset The Display
        MessageBox(NULL,L" an't Create A GL Device Context.",L" RROR",MB_OK|MB_ICONEXCLAMATION);
        return FALSE;                               //!/< Return FALSE
    }

    if (!(PixelFormat=ChoosePixelFormat(hDC,&pfd)))  //!/< Did Windows Find A Matching Pixel Format?
    {
        CloseWindow();                               //!/< Reset The Display
        MessageBox(NULL, L" an't Find A Suitable PixelFormat.", L" RROR",MB_OK|MB_ICONEXCLAMATION);
        return FALSE;                               //!/< Return FALSE
    }

    if(!SetPixelFormat(hDC,PixelFormat,&pfd))        //!/< Are We Able To Set The Pixel Format?
    {
        CloseWindow();                               //!/< Reset The Display
        MessageBox(NULL, L" an't Set The PixelFormat.", L" RROR",MB_OK|MB_ICONEXCLAMATION);
        return FALSE;                               //!/< Return FALSE
    }

    if (!(hRC=wglCreateContext(hDC)))             //!/< Are We Able To Get A Rendering Context?
    {
        CloseWindow();                               //!/< Reset The Display
        MessageBox(NULL, L" an't Create A GL Rendering Context.", L" RROR",MB_OK|MB_ICONEXCLAMATION);
        return FALSE;                               //!/< Return FALSE
    }

    if(!wglMakeCurrent(hDC,hRC))                  //!/< Try To Activate The Rendering Context
    {
        CloseWindow();                               //!/< Reset The Display
        MessageBox(NULL, L" an’t Activate The GL Rendering Context.", L" RROR",MB_OK|MB_ICONEXCLAMATION);
        return FALSE;                               //!/< Return FALSE
    }

    ShowWindow(hWnd,SW_SHOW);                       //!/< Show The Window
    SetForegroundWindow(hWnd);                      //!/< Slightly Higher Priority
    SetFocus(hWnd);                                 //!/< Sets Keyboard Focus To The Window
    ReSizeGLScene(width, height);                  //!/< Set Up Our Perspective GL Screen

    if (!InitGL())                                    //!/< Initialize Our Newly Created GL Window
    {
        CloseWindow();                               //!/< Reset The Display
        MessageBox(NULL, L" nitialization Failed.", L" RROR",MB_OK|MB_ICONEXCLAMATION);
        return FALSE;                               //!/< Return FALSE
    }

    return TRUE;                                    //!/< Success
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



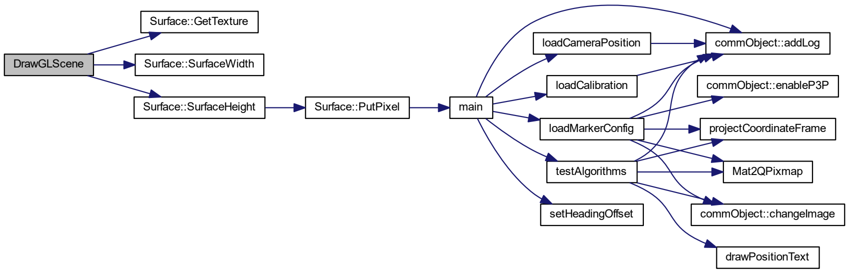
6.12.2.3 DrawGLScene()

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

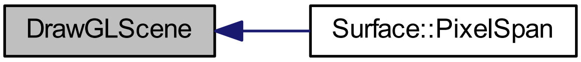
Definition at line 76 of file supportcode.cpp.

```
{
    if(surf==NULL)
        return true;

    static bool frameThrottler = true;
    frameThrottler=!frameThrottler;

    if(frameThrottler)  //!/<== Only display every other frame in case VSYNC is enabled.  Otherwise
        return true;    //!/<== application would get behind

    int pixelWidth = width;
    int pixelHeight= height;

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); //!/< Clear The Screen And The Depth Buffer
    glLoadIdentity();                                   //!/< Reset The View

    static GLuint ff = 0;

    int tex = surf->GetTexture();
    if(tex==0)
        tex=ff;
    else
        ff=tex;
    if(tex==0)
        return true;

    glBindTexture(GL_TEXTURE_2D, tex);

    glBegin(GL_QUADS);
        glTexCoord2f(0.0f, 0.0f); glVertex3f( 0,    0, 0);
        glTexCoord2f((GLfloat)pixelWidth/(GLfloat)surf->SurfaceWidth(), 0.0f); glVertex3f( (GLfloat)gWindowWidth,  0, 0);
        glTexCoord2f((GLfloat)pixelWidth/(GLfloat)surf->SurfaceWidth(), (GLfloat)pixelHeight/(GLfloat)surf->SurfaceHeight()); glVertex3f( (GLfloat)gWindowWidth,(GLfloat)gWindowHeight, 0);
        glTexCoord2f(0.0f, (GLfloat)pixelHeight/(GLfloat)surf->SurfaceHeight()); glVertex3f( 0, (GLfloat) gWindowHeight, 0);
    glEnd();

    SwapBuffers(hDC);                    //!/< Swap Buffers (Double Buffering)

    return true;                                        //!/< Keep Going
}
```

Here is the call graph for this function:



Here is the caller graph for this function:



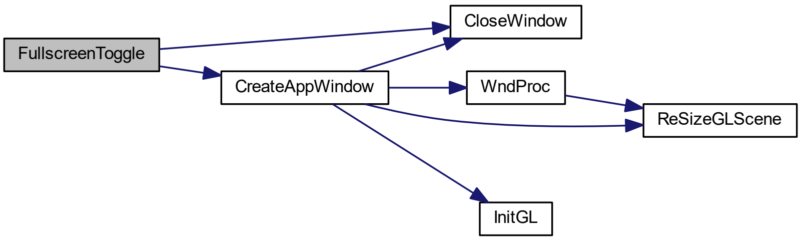
6.12.2.4 FullscreenToggle()

```
bool FullscreenToggle ( )
```

Definition at line 536 of file supportcode.cpp.

```
{
    keys[VK_F1]=FALSE;                    //!/< If So Make Key FALSE
    CloseWindow();                         //!/< Kill Our Current Window
    gFullscreen=!gFullscreen;          //!/< Toggle Fullscreen / Windowed Mode

    if (!CreateAppWindow(windowName,windowWidth, windowHeight,32,gFullscreen))
    {
        MessageBox(0, L"Unable to toggle to full screen",L"Error", MB_OK);
        return 1;
    }

    return true;
}
```

Here is the call graph for this function:



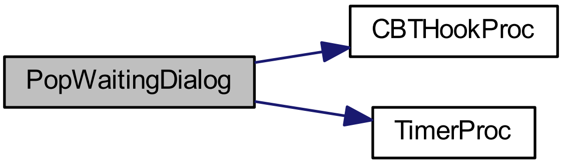
6.12.2.5 PopWaitingDialog()

```
bool PopWaitingDialog ( )
```

Definition at line 615 of file supportcode.cpp.

```
{
    //!/<== hook in so we can create a message box that has only a ’Cancel’ button ==--

    hHook = SetWindowsHookEx(WH_CBT, reinterpret_cast<HOOKPROC>(&CBTHookProc), NULL, GetCurrentThreadId());

    UINT_PTR nTimer = SetTimer(0, 100, 3000,(TIMERPROC) TimerProc);
    int iResult = MessageBox( 0, L"waiting for connected cameras...", L"Camera Initialization", MB_OK );

    if( iResult == IDOK )
    {
        //!/<== user has clicked the cancel button ==--
        UnhookWindowsHookEx(hHook);
        return false;
    }

    KillTimer(0, nTimer);

    return true;
}
```

Here is the call graph for this function:



6.12.2.6 PumpMessages()

```
bool PumpMessages ( )
```

Definition at line 551 of file supportcode.cpp.

```
{
    MSG msg;

    if (PeekMessage(&msg,NULL,0,0,PM_REMOVE))   //!/< Is There A Message Waiting?
    {
        if (msg.message==WM_QUIT)               //!/< Have We Received A Quit Message?
            return false;
        else                                    //!/< If Not, Deal With Window Messages
        {
            TranslateMessage(&msg);             //!/< Translate The Message
            DispatchMessage(&msg);              //!/< Dispatch The Message
        }
    }

    return true;
}
```

6.12.2.7 WndProc()

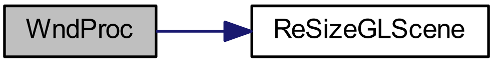
```
LRESULT CALLBACK WndProc (
             HWND ,
             UINT ,
             WPARAM ,
             LPARAM  )
```

/< Additional Message Information

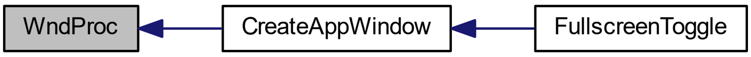
Definition at line 118 of file supportcode.cpp.

```
{
    switch (uMsg)                                   //!/< Check For Windows Messages
    {
        case WM_ACTIVATE:                           //!/< Watch For Window Activate Message
        {
            if (!HIWORD(wParam))                    //!/< Check Minimization State
            {
                gActive=TRUE;                        //!/< Program Is Active
            }
            else
            {
                gActive=FALSE;                       //!/< Program Is No Longer Active
            }

            return 0;                               //!/< Return To The Message Loop
        }

        case WM_POWERBROADCAST:
            if(wParam == PBT_APMSUSPEND)
            {
                CameraLibrary::CameraManager::X().PrepareForSuspend();
            }
            if(wParam == PBT_APMRESUMEAUTOMATIC)
            {
                CameraLibrary::CameraManager::X().ResumeFromSuspend();
            }
            break;

        case WM_SYSCOMMAND:                         //!/< Intercept System Commands
        {
            switch (wParam)                         //!/< Check System Calls
            {
                case SC_SCREENSAVE:                 //!/< Screensaver Trying To Start?
                case SC_MONITORPOWER:               //!/< Monitor Trying To Enter Powersave?
                return 0;                           //!/< Prevent From Happening
            }
            break;                                  //!/< Exit
        }

        case WM_CLOSE:                              //!/< Did We Receive A Close Message?
        {
            PostQuitMessage(0);                     //!/< Send A Quit Message
            return 0;                               //!/< Jump Back
        }

        case WM_KEYDOWN:                            //!/< Is A Key Being Held Down?
        {
            keys[wParam] = TRUE;                    //!/< If So, Mark It As TRUE
            return 0;                               //!/< Jump Back
        }

        case WM_KEYUP:                              //!/< Has A Key Been Released?
        {
            keys[wParam] = FALSE;                   //!/< If So, Mark It As FALSE
            return 0;                               //!/< Jump Back
        }
        case WM_MOVE:
            return 0;                               //!/< Jump Back
        case WM_PAINT:
            return 0;
        case WM_SIZE:                               //!/< Resize The OpenGL Window
        {
            ReSizeGLScene(LOWORD(lParam),HIWORD(lParam));  //!/< LoWord=Width, HiWord=Height
            return 0;                               //!/< Jump Back
        }
    }

    //!/< Pass All Unhandled Messages To DefWindowProc
    return DefWindowProc(hWnd,uMsg,wParam,lParam);
}
```

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