

## Calibration

Generated by Doxygen 1.12.0



<b>1 Class Index</b>	<b>1</b>
1.1 Class List	1
<b>2 File Index</b>	<b>3</b>
2.1 File List	3
<b>3 Class Documentation</b>	<b>5</b>
3.1 BaslerCamera Class Reference	5
3.1.1 Detailed Description	5
3.1.2 Constructor & Destructor Documentation	6
3.1.2.1 BaslerCamera() [1/2]	6
3.1.2.2 BaslerCamera() [2/2]	6
3.1.2.3 ~BaslerCamera()	7
3.1.3 Member Function Documentation	7
3.1.3.1 getFrame()	7
3.1.4 Member Data Documentation	7
3.1.4.1 camera_timeout	7
3.2 Calibration Class Reference	8
3.2.1 Detailed Description	8
3.2.2 Constructor & Destructor Documentation	8
3.2.2.1 Calibration()	8
3.2.2.2 ~Calibration()	9
3.2.3 Member Function Documentation	9
3.2.3.1 calibrate()	9
3.2.3.2 drawChessboardCorners()	9
3.2.3.3 findChessboardCorners()	9
3.2.3.4 save()	10
3.3 CalibrationData Struct Reference	10
3.3.1 Detailed Description	11
3.3.2 Member Data Documentation	11
3.3.2.1 K_inv	11
3.3.2.2 P	11
3.3.2.3 pixel_height	11
3.3.2.4 pixel_width	11
3.3.2.5 R_inv	11
3.3.2.6 t	12
3.4 Chessboard Struct Reference	12
3.4.1 Detailed Description	12
3.4.2 Member Data Documentation	13
3.4.2.1 corners	13
3.4.2.2 horizontal_squares	13
3.4.2.3 pixel_height	13
3.4.2.4 pixel_width	13

3.4.2.5 square_size_mm . . . . .	13
3.4.2.6 start_pixel . . . . .	13
3.4.2.7 vertical_squares . . . . .	13
3.4.2.8 world_coordinates_mm . . . . .	14
3.5 Window Class Reference . . . . .	14
3.5.1 Detailed Description . . . . .	14
3.5.2 Constructor & Destructor Documentation . . . . .	14
3.5.2.1 Window() . . . . .	14
3.5.2.2 ~Window() . . . . .	15
3.5.3 Member Function Documentation . . . . .	15
3.5.3.1 load2Frames() . . . . .	15
3.5.3.2 loadFrame() [1/2] . . . . .	15
3.5.3.3 loadFrame() [2/2] . . . . .	16
<b>4 File Documentation</b> . . . . .	<b>17</b>
4.1 include/BaslerCamera.hpp File Reference . . . . .	17
4.2 BaslerCamera.hpp . . . . .	17
4.3 include/Calibration.hpp File Reference . . . . .	18
4.4 Calibration.hpp . . . . .	19
4.5 include/UI.hpp File Reference . . . . .	22
4.5.1 Macro Definition Documentation . . . . .	22
4.5.1.1 COLOR_BLUE . . . . .	22
4.5.1.2 COLOR_GREEN . . . . .	22
4.5.1.3 COLOR_GREY . . . . .	22
4.5.1.4 COLOR_RED . . . . .	22
4.5.1.5 COLOR_YELLOW . . . . .	23
4.6 UI.hpp . . . . .	23
4.7 src/main.cpp File Reference . . . . .	23
4.7.1 Function Documentation . . . . .	24
4.7.1.1 main() . . . . .	24
4.7.2 Variable Documentation . . . . .	24
4.7.2.1 lower_chessboard_car . . . . .	24
4.7.2.2 lower_chessboard_tower . . . . .	24
4.7.2.3 upper_chessboard_car . . . . .	24
4.7.2.4 upper_chessboard_tower . . . . .	24
<b>Index</b> . . . . .	<b>25</b>

# Chapter 1

## Class Index

### 1.1 Class List

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

<a href="#">BaslerCamera</a>	A class to interface with a Basler camera or emulate a camera using a video file . . . . .	5
<a href="#">Calibration</a>	Performs camera calibration using a series of chessboard patterns . . . . .	8
<a href="#">CalibrationData</a>	Contains data related to camera calibration . . . . .	10
<a href="#">Chessboard</a>	Represents a chessboard used for camera calibration . . . . .	12
<a href="#">Window</a>	A class to encapsulate an OpenCV window for displaying frames . . . . .	14



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

include/ <a href="#">BaslerCamera.hpp</a> . . . . .	17
include/ <a href="#">Calibration.hpp</a> . . . . .	18
include/ <a href="#">UI.hpp</a> . . . . .	22
src/ <a href="#">main.cpp</a> . . . . .	23





## Chapter 3

# Class Documentation

### 3.1 BaslerCamera Class Reference

A class to interface with a Basler camera or emulate a camera using a video file.

```
#include <BaslerCamera.hpp>
```

#### Public Member Functions

- [BaslerCamera](#) (float exposure\_time)  
*Constructs a [BaslerCamera](#) object for a real camera with the specified exposure time.*
- [BaslerCamera](#) (const std::string path\_to\_video\_file)  
*Constructs a [BaslerCamera](#) object to emulate a camera using a video file.*
- [~BaslerCamera](#) ()  
*Destructor for the [BaslerCamera](#) class.*
- cv::Mat [getFrame](#) ()  
*Retrieves the next frame from the camera or video file.*

#### Public Attributes

- int [camera\\_timeout](#) = 5000

#### 3.1.1 Detailed Description

A class to interface with a Basler camera or emulate a camera using a video file.

This class allows users to either interface with a physical Basler camera or emulate a camera by reading frames from a video file. It provides functionality to initialize the camera, retrieve frames, and handle resources properly.

#### Author

Anton Haes

### 3.1.2 Constructor & Destructor Documentation

#### 3.1.2.1 BaslerCamera() [1/2]

```
BaslerCamera::BaslerCamera (  
    float exposure_time) [inline]
```

Constructs a [BaslerCamera](#) object for a real camera with the specified exposure time.

Initializes the Basler camera, sets it to continuous acquisition mode, and configures the exposure time.

##### Author

Anton Haes

##### Parameters

<i>exposure_time</i>	The exposure time for the camera in microseconds.
----------------------	---

##### Exceptions

<i>std::runtime_error</i>	if the camera initialization fails.
---------------------------	-------------------------------------

#### 3.1.2.2 BaslerCamera() [2/2]

```
BaslerCamera::BaslerCamera (  
    const std::string path_to_video_file) [inline]
```

Constructs a [BaslerCamera](#) object to emulate a camera using a video file.

Opens the specified video file for reading frames.

##### Author

Anton Haes

##### Parameters

<i>path_to_video_file</i>	The path to the video file to be used for emulation.
---------------------------	--

##### Exceptions

<i>std::runtime_error</i>	if the video file cannot be opened.
---------------------------	-------------------------------------

### 3.1.2.3 ~BaslerCamera()

```
BaslerCamera::~~BaslerCamera () [inline]
```

Destructor for the [BaslerCamera](#) class.

Closes the camera and releases resources if the camera is not in emulation mode.

#### Author

Anton Haes

## 3.1.3 Member Function Documentation

### 3.1.3.1 getFrame()

```
cv::Mat BaslerCamera::getFrame () [inline]
```

Retrieves the next frame from the camera or video file.

If the camera is being emulated, it reads the next frame from the video file. If the camera is real, it grabs the latest frame from the camera.

#### Author

Anton Haes

#### Returns

cv::Mat The captured frame as an OpenCV Mat object. If the end of the video file is reached, it will return an empty frame.

#### Exceptions

<code>std::runtime_error</code>	if a frame cannot be retrieved or if the camera is not grabbing frames.
---------------------------------	---

## 3.1.4 Member Data Documentation

### 3.1.4.1 camera\_timeout

```
int BaslerCamera::camera_timeout = 5000
```

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

- include/[BaslerCamera.hpp](#)

## 3.2 Calibration Class Reference

Performs camera calibration using a series of chessboard patterns.

```
#include <Calibration.hpp>
```

### Public Member Functions

- [Calibration](#) ([BaslerCamera](#) \*cam)  
*Constructs a [Calibration](#) object with the given [BaslerCamera](#).*
- [~Calibration](#) ()
- void [findChessboardCorners](#) ([Chessboard](#) \*chessboards[])  
*Finds and records chessboard corners in images.*
- void [calibrate](#) ([Chessboard](#) \*chessboards[])  
*Performs camera calibration using the detected chessboard corners.*
- void [save](#) (const std::string &filename)  
*Saves the calibration data to a file.*
- void [drawChessboardCorners](#) (cv::Mat \*frame, [Chessboard](#) \*chessboards[])  
*Draws detected chessboard corners on the provided image.*

### 3.2.1 Detailed Description

Performs camera calibration using a series of chessboard patterns.

This class handles the process of camera calibration by finding chessboard corners in images, computing the camera's intrinsic and extrinsic parameters, and saving these parameters to a file. It also provides functionality to visualize the detected chessboard corners.

#### Author

Anton Haes

### 3.2.2 Constructor & Destructor Documentation

#### 3.2.2.1 Calibration()

```
Calibration::Calibration (  
    BaslerCamera * cam) [inline]
```

Constructs a [Calibration](#) object with the given [BaslerCamera](#).

#### Author

Anton Haes

#### Parameters

<i>cam</i>	Pointer to a <a href="#">BaslerCamera</a> object used to capture images for calibration.
------------	--

### 3.2.2.2 ~Calibration()

```
Calibration::~~Calibration () [inline]
```

## 3.2.3 Member Function Documentation

### 3.2.3.1 calibrate()

```
void Calibration::calibrate (  
    Chessboard * chessboards[]) [inline]
```

Performs camera calibration using the detected chessboard corners.

This method computes the camera's calibration parameters based on the corners detected in the chessboard images. It constructs matrices for calibration, performs matrix decompositions, and computes the intrinsic and extrinsic parameters.

#### Author

Anton Haes

#### Parameters

<i>chessboards</i>	Array of pointers to <a href="#">Chessboard</a> objects containing the detected corners and chessboard details.
--------------------	---

### 3.2.3.2 drawChessboardCorners()

```
void Calibration::drawChessboardCorners (  
    cv::Mat * frame,  
    Chessboard * chessboards[]) [inline]
```

Draws detected chessboard corners on the provided image.

This method visualizes the detected chessboard corners by drawing circles around them on the given image frame.

#### Author

Anton Haes

#### Parameters

<i>frame</i>	Pointer to a <code>cv::Mat</code> object representing the image where the corners will be drawn.
<i>chessboards</i>	Array of pointers to <a href="#">Chessboard</a> objects containing the detected corners.

### 3.2.3.3 findChessboardCorners()

```
void Calibration::findChessboardCorners (  
    Chessboard * chessboards[]) [inline]
```

Finds and records chessboard corners in images.

This method iterates through an array of [Chessboard](#) objects and captures images from the camera to find chessboard corners within specified regions. It displays the frames with detected corners and adjusts corner coordinates based on the full frame.

#### Author

Anton Haes

## Parameters

<i>chessboards</i>	Array of pointers to <a href="#">Chessboard</a> objects, each representing a calibration pattern.
--------------------	---

**3.2.3.4 save()**

```
void Calibration::save (
    const std::string & filename) [inline]
```

Saves the calibration data to a file.

This method writes the camera calibration parameters, including the projection matrix, intrinsic matrix, rotation matrix, and translation vector, to a specified binary file. It also includes image dimensions.

## Author

Anton Haes

## Parameters

<i>filename</i>	The name of the file where calibration data will be saved.
-----------------	--

## Exceptions

<i>std::runtime_error</i>	If the file cannot be opened or created for writing.
---------------------------	--

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

- include/[Calibration.hpp](#)

**3.3 CalibrationData Struct Reference**

Contains data related to camera calibration.

```
#include <Calibration.hpp>
```

**Public Attributes**

- Eigen::MatrixX<double> **P** = Eigen::MatrixX<double>::Zero(3, 4)  
*The projection matrix of the camera.*
- Eigen::MatrixX<double> **K\_inv**  
*The inverse of the intrinsic matrix of the camera.*
- Eigen::MatrixX<double> **R\_inv**  
*The inverse of the rotation matrix of the camera.*
- Eigen::MatrixX<double> **t**  
*The translation vector of the camera.*
- int **pixel\_width** = 1920  
*Pixel width of the calibrated camera.*
- int **pixel\_height** = 1200  
*Pixel height of the calibrated camera.*

### 3.3.1 Detailed Description

Contains data related to camera calibration.

This structure holds the intrinsic and extrinsic parameters of the camera obtained from the calibration process. It includes the projection matrix, inverse intrinsic matrix, inverse rotation matrix, translation vector, and image dimensions.

#### Author

Anton Haes

### 3.3.2 Member Data Documentation

#### 3.3.2.1 K\_inv

```
Eigen::MatrixXd CalibrationData::K_inv
```

The inverse of the intrinsic matrix of the camera.

#### 3.3.2.2 P

```
Eigen::MatrixXd CalibrationData::P = Eigen::MatrixXd::Zero(3, 4)
```

The projection matrix of the camera.

#### 3.3.2.3 pixel\_height

```
int CalibrationData::pixel_height = 1200
```

Pixel height of the calibrated camera.

#### 3.3.2.4 pixel\_width

```
int CalibrationData::pixel_width = 1920
```

Pixel width of the calibrated camera.

#### 3.3.2.5 R\_inv

```
Eigen::MatrixXd CalibrationData::R_inv
```

The inverse of the rotation matrix of the camera.

### 3.3.2.6 t

```
Eigen::MatrixXd CalibrationData::t
```

The translation vector of the camera.

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

- include/[Calibration.hpp](#)

## 3.4 Chessboard Struct Reference

Represents a chessboard used for camera calibration.

```
#include <Calibration.hpp>
```

### Public Attributes

- int [start\\_pixel](#) [2]  
*x, y pixel coordinates of upper left corner of chessboard\_region*
- int [pixel\\_width](#)  
*pixel width of the whole chessboard*
- int [pixel\\_height](#)  
*pixel height of the whole chessboard*
- int [world\\_coordinates\\_mm](#) [3]  
*x, y and z coordinates of upper left corner (= first element of std::vector<> corners)*
- int [horizontal\\_squares](#) = 8  
*Number of squares on the chessboard, in the horizontal direction.*
- int [vertical\\_squares](#) = 4  
*Number of squares on the chessboard, in the vertical direction.*
- int [square\\_size\\_mm](#) = 10  
*Real world size of each square, in millimeters.*
- std::vector< cv::Point2f > [corners](#)  
*Vector containing all the world coordinates of the [Chessboard](#).*

### 3.4.1 Detailed Description

Represents a chessboard used for camera calibration.

This structure contains information about the chessboard pattern, including its pixel coordinates, dimensions, and real-world location. It is used in the camera calibration process to map real-world coordinates to pixel coordinates and vice versa.

#### Author

Anton Haes



## 3.4.2 Member Data Documentation

### 3.4.2.1 corners

```
std::vector<cv::Point2f> Chessboard::corners
```

Vector containing all the world coordinates of the [Chessboard](#).

### 3.4.2.2 horizontal\_squares

```
int Chessboard::horizontal_squares = 8
```

Number of squares on the chessboard, in the horizontal direction.

### 3.4.2.3 pixel\_height

```
int Chessboard::pixel_height
```

pixel height of the whole chessboard

### 3.4.2.4 pixel\_width

```
int Chessboard::pixel_width
```

pixel width of the whole chessboard

### 3.4.2.5 square\_size\_mm

```
int Chessboard::square_size_mm = 10
```

Real world size of each square, in millimeters.

### 3.4.2.6 start\_pixel

```
int Chessboard::start_pixel[2]
```

x, y pixel coordinates of upper left corner of chessboard\_region

### 3.4.2.7 vertical\_squares

```
int Chessboard::vertical_squares = 4
```

Number of squares on the chessboard, in the vertical direction.

### 3.4.2.8 world\_coordinates\_mm

```
int Chessboard::world_coordinates_mm[3]
```

x, y and z coordinates of upper left corner (= first element of `std::vector<> corners`)

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

- [include/Calibration.hpp](#)

## 3.5 Window Class Reference

A class to encapsulate an OpenCV window for displaying frames.

```
#include <UI.hpp>
```

### Public Member Functions

- [Window](#) (const std::string &name)  
*Constructor that initializes the OpenCV window with the given name.*
- [~Window](#) ()  
*Destructor that destroys the OpenCV window.*
- int [loadFrame](#) (cv::Mat frame)  
*Loads and displays a frame in the OpenCV window.*
- int [loadFrame](#) (cv::Mat \*frame, int width, int height)  
*Loads, resizes, and displays a frame in the OpenCV window.*
- int [load2Frames](#) (cv::Mat \*frame\_left, cv::Mat \*frame\_right, int width, int height)  
*Loads, resizes, and displays a 2 frames next to each other in the OpenCV window.*

### 3.5.1 Detailed Description

A class to encapsulate an OpenCV window for displaying frames.

Author

Anton Haes

### 3.5.2 Constructor & Destructor Documentation

#### 3.5.2.1 Window()

```
Window::Window (
    const std::string & name) [inline]
```

Constructor that initializes the OpenCV window with the given name.

Author

Anton Haes

## Parameters

<i>name</i>	The name of the window.
-------------	-------------------------

### 3.5.2.2 ~Window()

```
Window::~~Window () [inline]
```

Destructor that destroys the OpenCV window.

## Author

Anton Haes

## 3.5.3 Member Function Documentation

### 3.5.3.1 load2Frames()

```
int Window::load2Frames (
    cv::Mat * frame_left,
    cv::Mat * frame_right,
    int width,
    int height) [inline]
```

Loads, resizes, and displays a 2 frames next to each other in the OpenCV window.

## Author

Anton Haes

## Parameters

<i>frame_left</i>	The first frame to display.
<i>frame_right</i>	The second frame to display
<i>width</i>	The width to resize both frames to.
<i>height</i>	The height to resize both frames to.

## Returns

Return the ASCII code of the key that was pressed, or -1 if no key was pressed.

### 3.5.3.2 loadFrame() [1/2]

```
int Window::loadFrame (
    cv::Mat * frame,
    int width,
    int height) [inline]
```

Loads, resizes, and displays a frame in the OpenCV window.

## Author

Anton Haes

## Parameters

<i>frame</i>	The frame to display.
<i>width</i>	The width to resize the frame to.
<i>height</i>	The height to resize the frame to.

**3.5.3.3 loadFrame()** [2/2]

```
int Window::loadFrame (  
    cv::Mat frame) [inline]
```

Loads and displays a frame in the OpenCV window.

## Author

Anton Haes

## Parameters

<i>frame</i>	The frame to display.
--------------	-----------------------

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

- [include/UI.hpp](#)

# Chapter 4

## File Documentation

### 4.1 include/BaslerCamera.hpp File Reference

```
#include <pylon/PylonIncludes.h>
#include <opencv2/opencv.hpp>
```

#### Classes

- class [BaslerCamera](#)

*A class to interface with a Basler camera or emulate a camera using a video file.*

### 4.2 BaslerCamera.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef BASLER_CAMERA_HPP
00002 #define BASLER_CAMERA_HPP
00003
00004 #include <pylon/PylonIncludes.h>
00005 #include <opencv2/opencv.hpp>
00006
00018 class BaslerCamera {
00019 public:
00020     // timeout when grabbing frames from the camera
00021     int camera_timeout = 5000;
00022
00034     BaslerCamera(float exposure_time) : is_emulated(false) {
00035         // Initialize Basler camera using Pylon API
00036         Pylon::PylonInitialize();
00037         camera = new Pylon::CInstantCamera(Pylon::CTlFactory::GetInstance().CreateFirstDevice());
00038         // sets up free-running continuous acquisition.
00039         camera->StartGrabbing(Pylon::GrabStrategy_LatestImageOnly);
00040         // configure exposure time
00041         GenApi::INodeMap& nodemap = camera->GetNodeMap();
00042         Pylon::CFloatParameter(nodemap, "ExposureTime").SetValue(exposure_time);
00043         // Specify the output pixel format.
00044         format_converter.OutputPixelFormat = Pylon::PixelFormat_BGR8packed;
00045     }
00046
00057     BaslerCamera(const std::string path_to_video_file) : is_emulated(true) {
00058         // Use OpenCV library to read and playback video file
00059         if (!video_capture.open(path_to_video_file)) {
00060             throw std::runtime_error("Error opening video file.");
00061         }
00062     }
00063
00071     ~BaslerCamera() {
```

```

00072         if (!is_emulated) {
00073             camera->Close();
00074             delete camera;
00075             Pylon::PylonTerminate();
00076         }
00077     }
00078
00090     cv::Mat getFrame() {
00091         // Case where the camera is being emulated
00092         if (is_emulated) {
00093             cv::Mat frame;
00094             // Read next frame from video
00095             bool successful = video_capture.read(frame);
00096             if (!successful) {
00097                 return cv::Mat(); // Return an empty Mat to indicate the end of the video file
00098             }
00099             return frame;
00100         }
00101
00102         // Case where the camera is not being emulated
00103         if (!camera->IsGrabbing()) {
00104             throw std::runtime_error("Camera is currently not grabbing frames.");
00105         }
00106         // Wait for an image and then retrieve it
00107         camera->RetrieveResult(camera_timeout, ptr_grab_result,
Pylon::TimeoutHandling_ThrowException);
00108         // Check if the image was grabbed successfully
00109         if (!ptr_grab_result->GrabSucceeded()) {
00110             throw std::runtime_error("Image not grabbed successfully from camera.");
00111         }
00112         // Convert image to pylonImage
00113         format_converter.Convert(pylon_image, ptr_grab_result);
00114         // Convert image to OpenCV Mat
00115         cv::Mat frame(ptr_grab_result->GetHeight(), ptr_grab_result->GetWidth(), CV_8UC3,
(uint8_t*)pylon_image.GetBuffer());
00116         // Copy the frame to ensure memory safety
00117         frame = frame.clone();
00118         return frame;
00119     }
00120
00121 private:
00122     bool is_emulated; // variable to indicate if the camera is being emulated with a video file
00123
00124     // Pylon objects for Basler camera
00125     Pylon::CInstantCamera* camera;
00126     Pylon::CGrabResultPtr ptr_grab_result;
00127     Pylon::CPylonImage pylon_image;
00128     Pylon::CImageFormatConverter format_converter;
00129
00130     // OpenCV object to read video file
00131     cv::VideoCapture video_capture;
00132 };
00133
00134 #endif // BASLER_CAMERA_HPP
00135

```

## 4.3 include/Calibration.hpp File Reference

```

#include <opencv2/opencv.hpp>
#include <fstream>
#include <Dense>
#include "BaslerCamera.hpp"
#include "UI.hpp"

```

### Classes

- struct [Chessboard](#)  
*Represents a chessboard used for camera calibration.*
- struct [CalibrationData](#)  
*Contains data related to camera calibration.*
- class [Calibration](#)  
*Performs camera calibration using a series of chessboard patterns.*

## 4.4 Calibration.hpp

[Go to the documentation of this file.](#)

```

00001 #ifndef CALIBRATION_HPP
00002 #define CALIBRATION_HPP
00003
00004 #include <opencv2/opencv.hpp>
00005 #include <fstream> // library to write and read from files
00006 #include <Dense> // Eigen library for matrices
00007
00008 #include "BaslerCamera.hpp"
00009 #include "UI.hpp"
00010
00021 struct Chessboard {
00025     int start_pixel[2];
00029     int pixel_width;
00033     int pixel_height;
00037     int world_coordinates_mm[3];
00041     int horizontal_squares = 8;
00045     int vertical_squares = 4;
00049     int square_size_mm = 10;
00053     std::vector<cv::Point2f> corners;
00054 };
00055
00065 struct CalibrationData {
00069     Eigen::MatrixX_d P = Eigen::MatrixX_d::Zero(3, 4);
00073     Eigen::MatrixX_d K_inv;
00077     Eigen::MatrixX_d R_inv;
00081     Eigen::MatrixX_d t;
00085     int pixel_width = 1920;
00089     int pixel_height = 1200;
00090 };
00091
00102 class Calibration {
00103 public:
00104
00112     Calibration(BaslerCamera* cam) : camera(cam) {}
00113
00114     // Destructor for the Vision3D class
00115     ~Calibration() {}
00116
00128     void findChessboardCorners(Chessboard *chessboards[]) {
00129         Window window("Calibration");
00130
00131         for (int i = 0; chessboards[i] != NULL; i++) {
00132             bool found_chessboard = false;
00133             int start_x = chessboards[i]->start_pixel[0];
00134             int start_y = chessboards[i]->start_pixel[1];
00135             int width = chessboards[i]->pixel_width;
00136             int height = chessboards[i]->pixel_height;
00137             cv::Rect chessboard_region(start_x, start_y, width, height);
00138             cv::Size chessboard_size = cv::Size(chessboards[i]->horizontal_squares-1,
chessboards[i]->vertical_squares-1);
00139
00140             while (!found_chessboard) {
00141                 cv::Mat frame = camera->getFrame();
00142                 found_chessboard = cv::findChessboardCorners(frame(chessboard_region), chessboard_size
, chessboards[i]->corners, cv::CALIB_CB_ADAPTIVE_THRESH);
00143
00144                 cv::rectangle(frame, chessboard_region, COLOR_GREEN, 3);
00145                 window.loadFrame(frame, 768, 480);
00146             }
00147
00148             // Change chessboard coordinates to full frame instead of chessboard_region
00149             for (size_t j = 0; j < chessboards[i]->corners.size(); j++) {
00150                 chessboards[i]->corners[j].x += start_x;
00151                 chessboards[i]->corners[j].y += start_y;
00152             }
00153
00154             int index_top_left = 0;
00155             int index_top_right = chessboards[i]->horizontal_squares - 2;
00156             int index_bottom_left = (chessboards[i]->horizontal_squares-1) *
(chessboards[i]->vertical_squares-2);
00157
00158             // mirror chessboard so that x coordinates go from low to high
00159             if (chessboards[i]->corners[index_top_left].x >
chessboards[i]->corners[index_top_right].x) {
00160                 chessboards[i]->corners = mirrorXAxis(chessboards[i]);
00161             }
00162             // mirror chessboard so that y coordinates go from low to high
00163             if (chessboards[i]->corners[index_top_left].y >
chessboards[i]->corners[index_bottom_left].y) {
00164                 chessboards[i]->corners = mirrorYAxis(chessboards[i]);
00165             }
00166         }

```

```

00167
00168     }
00169
00170 void calibrate(Chessboard* chessboards[]) {
00171     // Count number of calibration points
00172     int num_points = 0;
00173     for (int i = 0; chessboards[i] != NULL; i++) {
00174         num_points += chessboards[i]->corners.size();
00175     }
00176
00177     // Create and fill Matrix A (Section 4.2 from preparatory research training)
00178     Eigen::MatrixXd A(num_points*2, 12);
00179     for (int k = 0; chessboards[k] != NULL; k++) {
00180         int num_cols = chessboards[k]->horizontal_squares-1;
00181         int num_rows = chessboards[k]->vertical_squares-1;
00182         double Ly = (double)chessboards[k]->world_coordinates_mm[1];
00183         double Lz = (double)chessboards[k]->world_coordinates_mm[2];
00184         for (int i = 0; i < num_rows; i++) {
00185             double Lx = (double)chessboards[k]->world_coordinates_mm[0]; // FIX THIS
00186             for (int j = 0; j < num_cols; j++) {
00187                 int u = chessboards[k]->corners[i * num_cols + j].x;
00188                 int v = chessboards[k]->corners[i * num_cols + j].y;
00189
00190                 int index = (k*chessboards[k]->corners.size()) + (i * num_cols + j);
00191                 A.row(2*index) << Lx, Ly, Lz, 1, 0, 0, 0, 0, -u*Lx, -u*Ly, -u*Lz, -u;
00192                 A.row(2*index+1) << 0, 0, 0, 0, Lx, Ly, Lz, 1, -v*Lx, -v*Ly, -v*Lz, -v;
00193
00194                 Lx += (double)chessboards[k]->square_size_mm;
00195             }
00196             Ly -= (double)chessboards[k]->square_size_mm;
00197         }
00198     }
00199
00200     // Find Eigenvectors and Eigenvalues (Section 4.2 from preparatory research training)
00201     Eigen::MatrixXd ATA = A.transpose() * A;
00202     Eigen::EigenSolver<Eigen::MatrixXd> es;
00203     Eigen::VectorXd eigenvalues(12, 1);
00204     Eigen::MatrixXd eigenvectors(12, 12);
00205
00206     es.compute(ATA, true);
00207     eigenvalues = es.eigenvalues().real();
00208     eigenvectors = es.eigenvectors().real();
00209
00210     // Find Eigenvalue and Eigenvector combination with smallest Loss value (Section 4.2 from
00211     preparatory research training)
00212     float min = FLT_MAX;
00213     uint8_t best_index = 0;
00214     for (uint8_t i = 0; i < 12; i++) {
00215         float lambda = eigenvalues.row(i).coeff(0, 0);
00216         Eigen::MatrixXd x_normalized = eigenvectors.col(i);
00217         float loss_func = (x_normalized.transpose() * ATA * x_normalized).coeff(0, 0) - lambda *
00218         ((x_normalized.transpose() * x_normalized).coeff(0, 0) - 1);
00219         if (loss_func < min) {
00220             best_index = i;
00221         }
00222     }
00223
00224     // Fill matrix P
00225     Eigen::VectorXd p = es.eigenvectors().real().col(best_index);
00226     for (uint8_t i = 0; i < 3; i++) {
00227         for (uint8_t j = 0; j < 4; j++) {
00228             calibration_data.P(i, j) = p(i*4+j);
00229         }
00230     }
00231
00232     // Create a new matrix P_1 by omitting the last column of P (section 3.1 from bachelor thesis)
00233     Eigen::MatrixXd P_1 = calibration_data.P.leftCols(calibration_data.P.cols() - 1);
00234     // Create a new matrix P_2 by extracting the last column of P (section 3.1 from bachelor
00235     thesis)
00236     Eigen::MatrixXd P_2 = calibration_data.P.col(calibration_data.P.cols() - 1);
00237
00238     // RQ-Decomposition of P_1 (according to section 2 of bachelor thesis)
00239     Eigen::MatrixXd pAT = P_1.colwise().reverse().transpose(); // cfr. Equation 1 of bachelor
00240     thesis
00241     Eigen::HouseholderQR<Eigen::MatrixXd> qr(pAT); // Perform QR-Decomposition
00242     Eigen::MatrixXd q_tilde = qr.householderQ();
00243     Eigen::MatrixXd r_tilde = qr.matrixQR().triangularView<Eigen::Upper>();
00244
00245     // Equation 8 from bachelor thesis
00246     Eigen::MatrixXd Q = q_tilde.transpose().colwise().reverse();
00247     Eigen::MatrixXd R = r_tilde.transpose().colwise().reverse().rowwise().reverse();
00248
00249     // fill calibration_data
00250     calibration_data.K_inv = R.inverse();
00251     calibration_data.R_inv = Q.inverse();
00252     calibration_data.t = calibration_data.K_inv * P_2; // see Equation 9 from bachelor thesis
00253 }

```



```

00260
00273 void save(const std::string& filename) {
00274     // Create file
00275     std::ofstream file(filename, std::ios::out | std::ios::binary);
00276     if (!file.is_open()) {
00277         throw std::runtime_error("Failed to open or create file for writing.");
00278     }
00279
00280     // Write pixel width and height
00281     file.write(reinterpret_cast<const char*>(&calibration_data.pixel_width), sizeof(int));
00282     file.write(reinterpret_cast<const char*>(&calibration_data.pixel_height), sizeof(int));
00283
00284     // Write matrix P
00285     int rows, cols;
00286     rows = calibration_data.P.rows();
00287     cols = calibration_data.P.cols();
00288     file.write(reinterpret_cast<const char*>(&rows), sizeof(int));
00289     file.write(reinterpret_cast<const char*>(&cols), sizeof(int));
00290     file.write(reinterpret_cast<const char*>(calibration_data.P.data()), rows * cols *
sizeof(double));
00291
00292     // Write matrix K_inv
00293     rows = calibration_data.K_inv.rows();
00294     cols = calibration_data.K_inv.cols();
00295     file.write(reinterpret_cast<const char*>(&rows), sizeof(int));
00296     file.write(reinterpret_cast<const char*>(&cols), sizeof(int));
00297     file.write(reinterpret_cast<const char*>(calibration_data.K_inv.data()), rows * cols *
sizeof(double));
00298
00299     // Write matrix R_inv
00300     rows = calibration_data.R_inv.rows();
00301     cols = calibration_data.R_inv.cols();
00302     file.write(reinterpret_cast<const char*>(&rows), sizeof(int));
00303     file.write(reinterpret_cast<const char*>(&cols), sizeof(int));
00304     file.write(reinterpret_cast<const char*>(calibration_data.R_inv.data()), rows * cols *
sizeof(double));
00305
00306     // Write matrix t
00307     rows = calibration_data.t.rows();
00308     cols = calibration_data.t.cols();
00309     file.write(reinterpret_cast<const char*>(&rows), sizeof(int));
00310     file.write(reinterpret_cast<const char*>(&cols), sizeof(int));
00311     file.write(reinterpret_cast<const char*>(calibration_data.t.data()), rows * cols *
sizeof(double));
00312
00313     file.close();
00314 }
00315
00326 void drawChessboardCorners(cv::Mat* frame, Chessboard* chessboards[]) {
00327     for (int i = 0; chessboards[i] != NULL; i++) {
00328         for (size_t j = 0; j < chessboards[i]->corners.size(); j++) {
00329             cv::circle(*frame, chessboards[i]->corners[j], 5, COLOR_GREEN, -1);
00330         }
00331     }
00332 }
00333
00334
00335
00336 private:
00337     BaslerCamera* camera;
00338     CalibrationData calibration_data;
00339
00352     std::vector<cv::Point2f> mirrorXAxis(Chessboard* chessboard) {
00353         std::vector<cv::Point2f> mirroredVector(chessboard->corners.size());
00354
00355         int num_cols = chessboard->horizontal_squares-1;
00356         int num_rows = chessboard->vertical_squares-1;
00357
00358         for (int i = 0; i < num_rows; i++) {
00359             for (int j = 0; j < num_cols; j++) {
00360                 mirroredVector[i * num_cols + j] = chessboard->corners[i * num_cols + num_cols - 1 -
j];
00361             }
00362         }
00363
00364         return mirroredVector;
00365     }
00366
00379     std::vector<cv::Point2f> mirrorYAxis(Chessboard* chessboard) {
00380         std::vector<cv::Point2f> mirroredVector(chessboard->corners.size());
00381
00382         int num_cols = chessboard->horizontal_squares-1;
00383         int num_rows = chessboard->vertical_squares-1;
00384
00385         for (int i = 0; i < num_rows; i++) {
00386             for (int j = 0; j < num_cols; j++) {
00387                 mirroredVector[i * num_cols + j] = chessboard->corners[(num_rows - 1 - i) * num_cols +

```

```
    jl;  
00388         }  
00389     }  
00390  
00391     return mirroredVector;  
00392 }  
00393 };  
00394  
00395 #endif // CALIBRATION_HPP
```

## 4.5 include/UI.hpp File Reference

```
#include <opencv2/opencv.hpp>
```

### Classes

- class [Window](#)  
*A class to encapsulate an OpenCV window for displaying frames.*

### Macros

- #define [COLOR\\_RED](#) cv::Scalar(0, 0, 255)
- #define [COLOR\\_GREEN](#) cv::Scalar(0, 255, 0)
- #define [COLOR\\_BLUE](#) cv::Scalar(255, 0, 0)
- #define [COLOR\\_YELLOW](#) cv::Scalar(0, 255, 255)
- #define [COLOR\\_GREY](#) cv::Scalar(127, 127, 127)

### 4.5.1 Macro Definition Documentation

#### 4.5.1.1 COLOR\_BLUE

```
#define COLOR_BLUE cv::Scalar(255, 0, 0)
```

#### 4.5.1.2 COLOR\_GREEN

```
#define COLOR_GREEN cv::Scalar(0, 255, 0)
```

#### 4.5.1.3 COLOR\_GREY

```
#define COLOR_GREY cv::Scalar(127, 127, 127)
```

#### 4.5.1.4 COLOR\_RED

```
#define COLOR_RED cv::Scalar(0, 0, 255)
```

### 4.5.1.5 COLOR\_YELLOW

```
#define COLOR_YELLOW cv::Scalar(0, 255, 255)
```

## 4.6 UI.hpp

[Go to the documentation of this file.](#)

```
00001 #ifndef UI_HPP
00002 #define UI_HPP
00003
00004 #include <opencv2/opencv.hpp>
00005
00006 #define COLOR_RED      cv::Scalar(0, 0, 255)
00007 #define COLOR_GREEN    cv::Scalar(0, 255, 0)
00008 #define COLOR_BLUE     cv::Scalar(255, 0, 0)
00009 #define COLOR_YELLOW   cv::Scalar(0, 255, 255)
00010 #define COLOR_GREY     cv::Scalar(127, 127, 127)
00011
00017 class Window {
00018 public:
00019
00027     Window(const std::string& name) : window_name(name) {
00028         cv::namedWindow(window_name, cv::WINDOW_AUTOSIZE);
00029     }
00030
00036     ~Window() {
00037         cv::destroyWindow(window_name);
00038     }
00039
00047     int loadFrame(cv::Mat frame) {
00048         cv::imshow(window_name, frame);
00049         return cv::waitKey(1);
00050     }
00051
00061     int loadFrame(cv::Mat* frame, int width, int height) {
00062         cv::resize(*frame, *frame, cv::Size(width, height));
00063         cv::imshow(window_name, *frame);
00064         return cv::waitKey(1);
00065     }
00066
00079     int load2Frames(cv::Mat* frame_left, cv::Mat* frame_right, int width, int height) {
00080         // resize the left frame to match the output size
00081         cv::resize(*frame_left, *frame_left, cv::Size(width, height));
00082         // resize the right frame to match the output size
00083         cv::resize(*frame_right, *frame_right, cv::Size(width, height));
00084         // create a new frame, and fill it with the left and right frame
00085         cv::Mat combined_frame(width*2, height, CV_8UC3);
00086         cv::hconcat(*frame_left, *frame_right, combined_frame);
00087         // show the frame
00088         cv::imshow(window_name, combined_frame);
00089         return cv::waitKey(1);
00090     }
00091
00092 private:
00093     std::string window_name; // The name of the OpenCV window.
00094 };
00095
00096 #endif // UI_HPP
00097
```

## 4.7 src/main.cpp File Reference

```
#include <opencv2/opencv.hpp>
#include "BaslerCamera.hpp"
#include "UI.hpp"
#include "Calibration.hpp"
```

### Functions

- `int main` (int argc, char \*\*argv)

## Variables

- [Chessboard lower\\_chessboard\\_tower](#)
- [Chessboard upper\\_chessboard\\_tower](#)
- [Chessboard lower\\_chessboard\\_car](#)
- [Chessboard upper\\_chessboard\\_car](#)

## 4.7.1 Function Documentation

### 4.7.1.1 main()

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

## 4.7.2 Variable Documentation

### 4.7.2.1 lower\_chessboard\_car

[Chessboard](#) lower\_chessboard\_car

#### Initial value:

```
= {  
    .start_pixel = {685, 500},  
    .pixel_width = 580,  
    .pixel_height = 160,  
    .world_coordinates_mm = {-30, 20, 26}  
}
```

### 4.7.2.2 lower\_chessboard\_tower

[Chessboard](#) lower\_chessboard\_tower

#### Initial value:

```
= {  
    .start_pixel = {685, 675},  
    .pixel_width = 580,  
    .pixel_height = 275,  
    .world_coordinates_mm = {-30, 20, 5}  
}
```

### 4.7.2.3 upper\_chessboard\_car

[Chessboard](#) upper\_chessboard\_car

#### Initial value:

```
= {  
    .start_pixel = {775, 365},  
    .pixel_width = 420,  
    .pixel_height = 115,  
    .world_coordinates_mm = {-30, 60, 36}  
}
```

### 4.7.2.4 upper\_chessboard\_tower

[Chessboard](#) upper\_chessboard\_tower

#### Initial value:

```
= {  
    .start_pixel = {720, 500},  
    .pixel_width = 520,  
    .pixel_height = 175,  
    .world_coordinates_mm = {-30, 60, 15}  
}
```

# Index

- ~BaslerCamera
  - BaslerCamera, [6](#)
- ~Calibration
  - Calibration, [8](#)
- ~Window
  - Window, [15](#)
- BaslerCamera, [5](#)
  - ~BaslerCamera, [6](#)
  - BaslerCamera, [6](#)
  - camera\_timeout, [7](#)
  - getFrame, [7](#)
- calibrate
  - Calibration, [9](#)
- Calibration, [8](#)
  - ~Calibration, [8](#)
  - calibrate, [9](#)
  - Calibration, [8](#)
  - drawChessboardCorners, [9](#)
  - findChessboardCorners, [9](#)
  - save, [10](#)
- CalibrationData, [10](#)
  - K\_inv, [11](#)
  - P, [11](#)
  - pixel\_height, [11](#)
  - pixel\_width, [11](#)
  - R\_inv, [11](#)
  - t, [11](#)
- camera\_timeout
  - BaslerCamera, [7](#)
- Chessboard, [12](#)
  - corners, [13](#)
  - horizontal\_squares, [13](#)
  - pixel\_height, [13](#)
  - pixel\_width, [13](#)
  - square\_size\_mm, [13](#)
  - start\_pixel, [13](#)
  - vertical\_squares, [13](#)
  - world\_coordinates\_mm, [13](#)
- COLOR\_BLUE
  - UI.hpp, [22](#)
- COLOR\_GREEN
  - UI.hpp, [22](#)
- COLOR\_GREY
  - UI.hpp, [22](#)
- COLOR\_RED
  - UI.hpp, [22](#)
- COLOR\_YELLOW
  - UI.hpp, [22](#)
- corners
  - Chessboard, [13](#)
- drawChessboardCorners
  - Calibration, [9](#)
- findChessboardCorners
  - Calibration, [9](#)
- getFrame
  - BaslerCamera, [7](#)
- horizontal\_squares
  - Chessboard, [13](#)
- include/BaslerCamera.hpp, [17](#)
- include/Calibration.hpp, [18](#), [19](#)
- include/UI.hpp, [22](#), [23](#)
- K\_inv
  - CalibrationData, [11](#)
- load2Frames
  - Window, [15](#)
- loadFrame
  - Window, [15](#), [16](#)
- lower\_chessboard\_car
  - main.cpp, [24](#)
- lower\_chessboard\_tower
  - main.cpp, [24](#)
- main
  - main.cpp, [24](#)
- main.cpp
  - lower\_chessboard\_car, [24](#)
  - lower\_chessboard\_tower, [24](#)
  - main, [24](#)
  - upper\_chessboard\_car, [24](#)
  - upper\_chessboard\_tower, [24](#)
- P
  - CalibrationData, [11](#)
- pixel\_height
  - CalibrationData, [11](#)
  - Chessboard, [13](#)
- pixel\_width
  - CalibrationData, [11](#)
  - Chessboard, [13](#)
- R\_inv
  - CalibrationData, [11](#)

- save
  - Calibration, [10](#)
- square\_size\_mm
  - Chessboard, [13](#)
- src/main.cpp, [23](#)
- start\_pixel
  - Chessboard, [13](#)
- t
  - CalibrationData, [11](#)
- UI.hpp
  - COLOR\_BLUE, [22](#)
  - COLOR\_GREEN, [22](#)
  - COLOR\_GREY, [22](#)
  - COLOR\_RED, [22](#)
  - COLOR\_YELLOW, [22](#)
- upper\_chessboard\_car
  - main.cpp, [24](#)
- upper\_chessboard\_tower
  - main.cpp, [24](#)
- vertical\_squares
  - Chessboard, [13](#)
- Window, [14](#)
  - ~Window, [15](#)
  - load2Frames, [15](#)
  - loadFrame, [15](#), [16](#)
  - Window, [14](#)
- world\_coordinates\_mm
  - Chessboard, [13](#)