



RobotEye Class Library

Reference Manual

Version 1.0.62
Mon May 30 2011

Ocular Robotics Pty. Ltd.
Level 3, 12-14 Ormonde Pde
Hurstville NSW 2220
Australia

www.ocularrobotics.com

Contents

1	RobotEye Class Library Reference Manual	1
2	RobotEye Conventions	3
2.1	RobotEye Coordinate System Definitions	3
2.2	RobotEye Angle Conventions	5
3	RobotEye Network Setup	7
3.1	RobotEye IP Address	7
3.2	Ethernet as a Control Bus	7
3.3	Configuring Your Firewall	9
4	Installation	11
4.1	Windows	11
4.2	Linux	12
5	Using the RobotEye API	13
5.1	Controller Connections	13
5.2	API Usage	13
5.2.1	The RobotEye Library Namespace	14
5.2.2	A Simple Example	14
5.2.3	Error Handling	14
5.2.4	A Motion Example	15
5.3	Example Code	15
6	Building Applications	17
6.1	Setting Up Your Project	17
6.1.1	Using CMake	17
6.1.2	Manually Setting up Projects	19
6.1.2.1	Microsoft Visual Studio	19
7	Namespace Documentation	21
7.1	ocular Namespace Reference	21
7.1.1	Detailed Description	21
8	Class Documentation	23
8.1	ocular::RobotEye Class Reference	23
8.1.1	Detailed Description	25
8.1.2	Member Enumeration Documentation	25
8.1.2.1	axisIndex_t	25
8.1.2.2	controlMode_t	25
8.1.2.3	digitalBit_t	25
8.1.2.4	eyeIndex_t	26
8.1.3	Constructor & Destructor Documentation	26
8.1.3.1	RobotEye	26
8.1.3.2	~RobotEye	26
8.1.4	Member Function Documentation	27

8.1.4.1	GetAccelerationParams	27
8.1.4.2	GetApertureAngles	27
8.1.4.3	GetLibraryVersion	27
8.1.4.4	GetSerialNumber	27
8.1.4.5	GetSystemInfo	28
8.1.4.6	Home	28
8.1.4.7	LogControllerData	28
8.1.4.8	ReceiveControllerData	29
8.1.4.9	SetAccelerationParams	29
8.1.4.10	SetApertureAngles	30
8.1.4.11	SetApertureAngles	30
8.1.4.12	SetApertureRates	31
8.1.4.13	SetBoundedElevationScan	32
8.1.4.14	SetControlSettings	33
8.1.4.15	SetDigitalOutput	33
8.1.4.16	SetPiecewiseLinearScan	34
8.1.4.17	SetRegionScan	34
8.1.4.18	StopMotion	35
8.1.4.19	TrackApertureAngles	36
8.2	ocular::RobotEyeCallback Class Reference	36
8.2.1	Detailed Description	36
8.2.2	Member Function Documentation	37
8.2.2.1	ReceiveData	37

Chapter 1

RobotEye Class Library Reference Manual



This document is the programmer's reference for Ocular Robotics' Pty. Ltd. RobotEye driver and its Application Programming Interface (API).

This driver supports all OEM versions of the RobotEye. It is implemented as a cross-platform C++ class library with simple API.

Chapter 2

RobotEye Conventions

2.1 RobotEye Coordinate System Definitions

The RobotEye device uses a right-handed (or 'positive') coordinate system fixed to the centre of the 'head' of the RobotEye with the aperture centred on the y axis when the aperture is at zero degrees azimuth and zero degrees elevation. The x axis is constrained to the horizontal plane, and the z axis 'up' with respect to x and y . The aperture angles (usually specified as the 'azimuthAngle' and 'elevationAngle') are angular offsets in azimuth and elevation from the y axis.

The following diagrams summarise the coordinate system used for the RobotEye system.

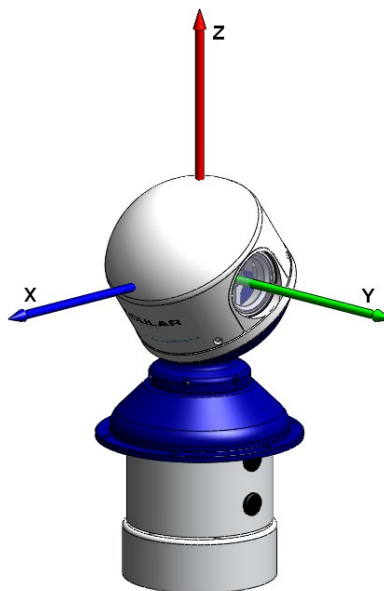


Figure 2.1: The RobotEye Coordinate System

The diagram above shows the right-handed frame used for the RobotEye.

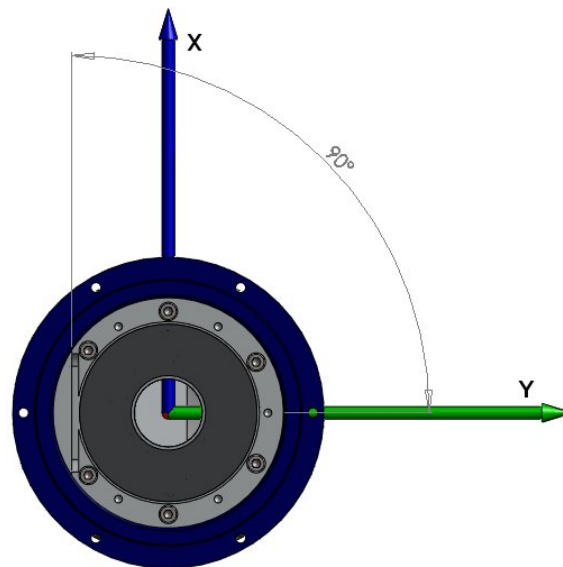


Figure 2.2: The RobotEye Azimuth Reference Surface

The diagram above shows the reference used for the direction of the y axis. The y axis, which is the 'azimuthAngle' zero reference is defined as being perpendicular to the encoder mounting surfaces.

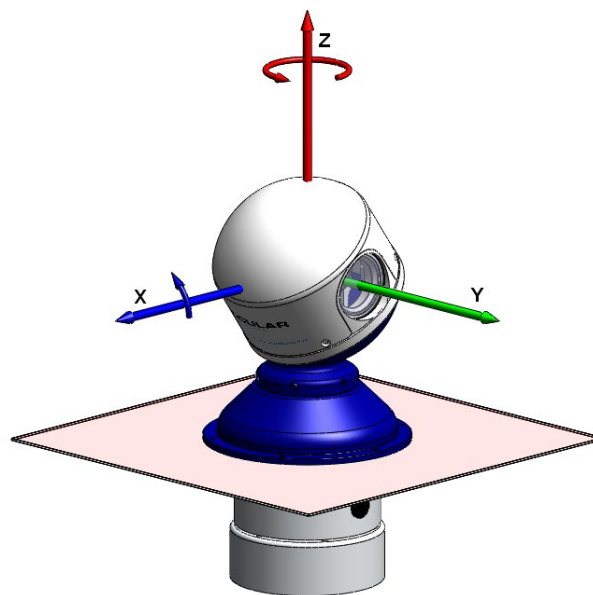


Figure 2.3: The RobotEye Elevation Reference Surface and Azimuth and Elevation Rotation Sign

The final diagram shows two things;

- The aperture zero elevation angle lies in a plane parallel to the to the RobotEye mounting flange mounting surface.
- The sign of the aperture angles, and angular rates. AzimuthAngle is more positive anti-clockwise, and ElevationAngle is positive upwards.

2.2 RobotEye Angle Conventions

The RobotEye library uses the following conventions when dealing with angles.

- Angles passed to the library must be expressed in degrees
- Functions that command the RobotEye to an absolute position will always take the shortest (or acute angle) path from the current location to the new location. i.e. if the RobotEye azimuth angle is currently 359 degrees, and it is commanded to 1 degree, it will only move 2 degrees in azimuth.
- Azimuth angles must be constrained to the domain 0 -> 360 degrees, where zero is defined as per the [RobotEye Coordinate System Definitions](#).
- Elevation angles must be constrained to the domain -maxElevation -> maxElevation, where maxElevation is typically 35 degrees. This angle can be varied for custom RobotEye devices. Zero degrees elevation is horizontal, as per the definitions given in the [RobotEye Coordinate System Definitions](#).

Chapter 3

RobotEye Network Setup

3.1 RobotEye IP Address

The RobotEye will come preconfigured with a static IP address on a private IPV4 subnet. See [IPv4 private addresses](#) for more information. The specific IP address will be provided with the RobotEye packaging documentation. Please contact Ocular Robotics Pty Ltd prior to delivery if you require a specific IP address for your device.

3.2 Ethernet as a Control Bus

It is strongly recommended that the ethernet interface to the RobotEye be considered a time-critical control bus. In practice this means that in an ideal situation, only the host computer (the PC where the user application is running) and the RobotEye are connected together via the ethernet connection. If the host PC must be connected to another network (i.e. a corporate intranet, or the internet), it should be via a separate network adapter on a different subnet.

It is possible to connect multiple RobotEye devices in a single network. In this case, it is recommended that the network be fully switched to avoid potential data 'collisions' on the network.

The following diagrams illustrate the preferred network setup for one or multiple RobotEyes. Note again that it is recommended that if the host PC must be connected to another network, that it should be done using a separate network interface card on a separate subnet.

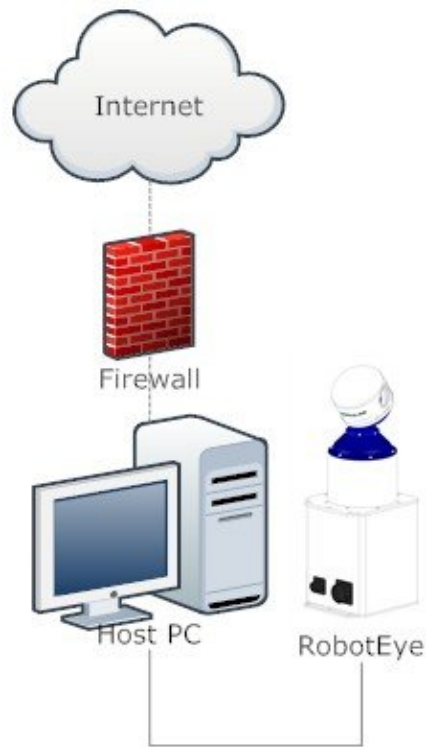


Figure 3.1: Network Schematic for Connecting a Host PC to a Single RobotEye

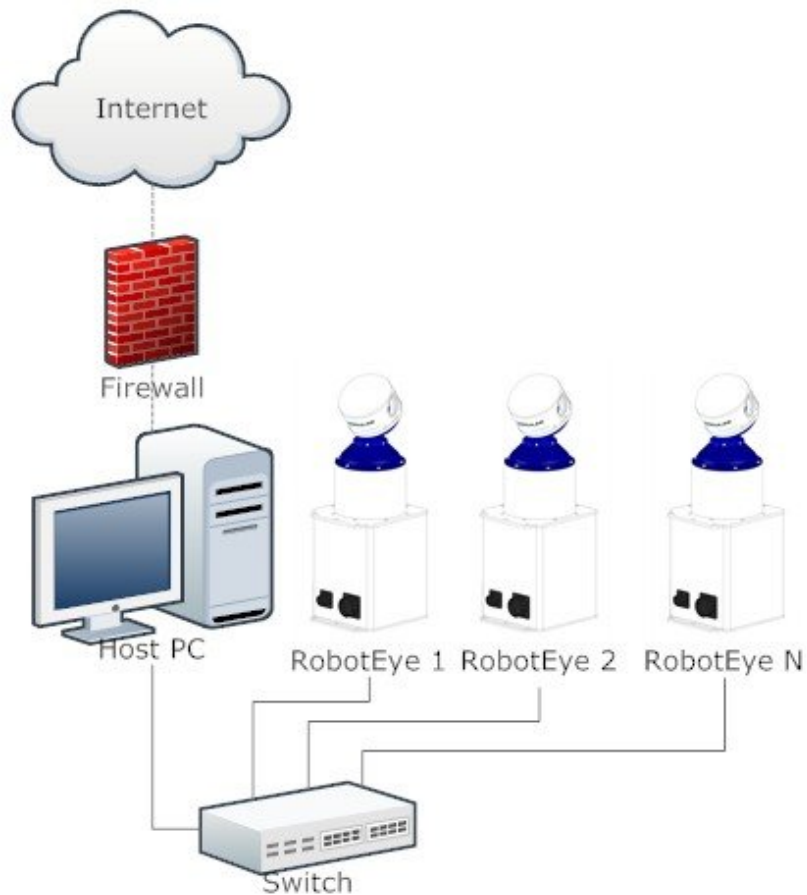


Figure 3.2: Network Schematic for Connecting a Host PC to Multiple RobotEyes

3.3 Configuring Your Firewall

In general, if the advice given in [Ethernet as a Control Bus](#) is followed, then it should be acceptable to disable any firewalls on the network adapter connected to the RobotEye device, as it is on a private network not accessible to the outside world. This is the simplest option if you are having network related issues.

Alternatively, if you must retain a firewall on the given network adapter, then make sure that UDP ports 60000-60007, 50000, and 67 are given exceptions in the firewall program (i.e. allow bidirectional communication on those UDP ports).

Chapter 4

Installation

4.1 Windows

On Windows platforms, the driver is installed using an installer program. The installer filename will be of the form 'RobotEyeLib-0.8.0-win32.exe', where '0.8.0' indicates the driver version number, and 'win32' indicates the target platform. By default, the driver is built as a 32-bit dll, however a 64-bit version may be requested if necessary.

To perform the installation under windows, simply double click the installer application, and follow the on-screen instructions. Note - you may need administrator privileges to run the installer.

The following figure shows the installer interface



Figure 4.1: The RobotEye Class Library Windows Installer

After accepting the Ocular Robotics license agreement, you will be prompted to enter an installation directory at the following screen.

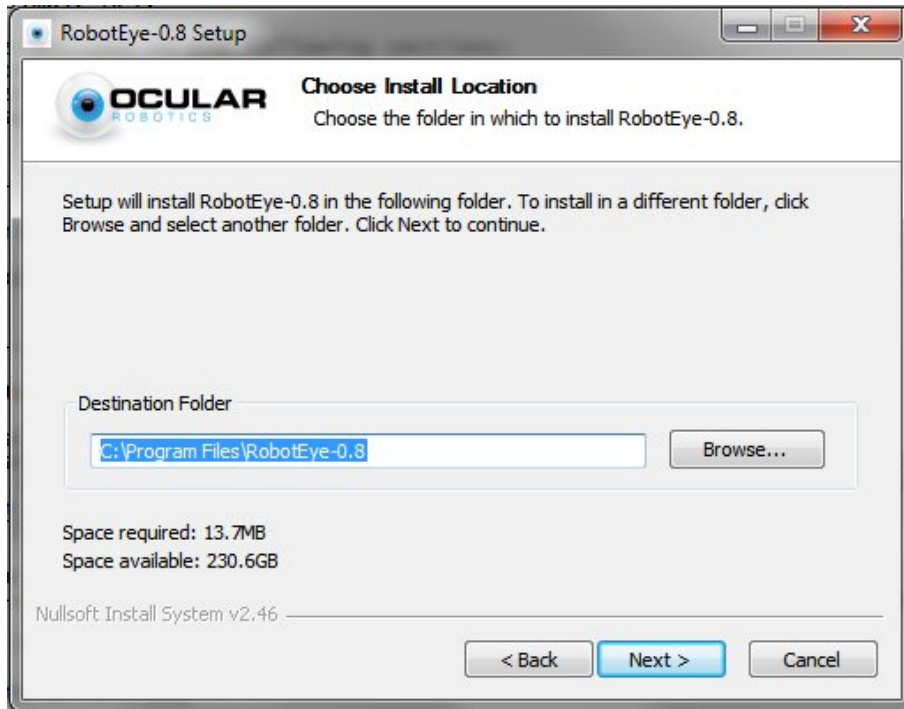


Figure 4.2: The Default RobotEye Class Library Windows Installation Directory

This directory will then contain all of the RobotEye library files: the driver, documentation and examples. In a Windows installation, a start menu group with links to the driver documentation will also be created.

4.2 Linux

The RobotEye library is distributed under Linux in two forms:

- as a '.deb' Debian Package, and
- as a '.rpm' Red Hat Package

Under Linux, the library will be installed to /opt/RobotEye-version, and is installed by invoking the relevant package manager.

Chapter 5

Using the RobotEye API

5.1 Controller Connections

The RobotEye controller supports up to eight simultaneous network connections. This allows parallelisation of certain tasks; for example it is possible to write an application that controls RobotEye trajectories, while another application is simultaneously logging aperture information.

There are some important caveats however;

- Only a single RobotEye object can exist in a given application; Attempting to create a second RobotEye object will result in an exception being thrown
- Only the first RobotEye object to connect to the controller can be *active*
- Any *passive* connections can only *listen* to data from the controller. They cannot be used to attempt to control the RobotEye device; doing so will result in an exception being thrown

An *active* connection is defined as one that can control the RobotEye device (*active* status will only be conferred to the *first* connection, so connection order is critical). A *passive* connection is defined as all other connections.

5.2 API Usage

Under Windows, the RobotEye class library consists of the following components;

- A header file 'RobotEye.h' which contains the public interface of the library
- A header file 'RobotEyeCallback.h' which contains the virtual prototype for receiving feedback from the controller
- An import library 'RobotEye.lib' built for the Microsoft Visual Studio compiler
- A shared library 'RobotEye.dll'

In a Linux installation, the import library is not required, and the shared library will have a '.so' file extension, and have the 'lib' prefix.

5.2.1 The RobotEye Library Namespace

All C++ software provided by Ocular Robotics Pty. Ltd. is encapsulated within the namespace 'ocular'. Therefore, all usage of this class library must either explicitly use the 'ocular' namespace, or have the following line of code added before objects contained in the namespace can be used.

```
using namespace ocular;
```

In the examples that follow, the namespace is explicitly used.

5.2.2 A Simple Example

For the simplest possible working example, create a source file (say 'MyProject.cpp'), and include the RobotEye header file. Then, construct a RobotEye object, passing the IP Address of the relevant RobotEye as the input argument. The IP address of the RobotEye is included in the packaging documentation of each RobotEye.

This code would resemble the following;

```
#include <RobotEye.h>

int main( void )
{
    ocular::RobotEye myRobotEye( "169.254.99.205" );

    return 0;
}
```

The act of construction of the RobotEye object will perform all necessary resource acquisition and initialisation. If the RobotEye object is constructed successfully, it will be immediately ready to use.

When the RobotEye object is destructed (through exception, deletion, or simply going out of scope), all motion of the RobotEye will cease, and the motors will be turned off to reduce power drain.

5.2.3 Error Handling

The RobotEye library uses exceptions to indicate to a user when an error occurred, and what the source of the error is. It is therefore always recommended to use the C++ mechanism of try-catch blocks around the use of any RobotEye member functions. This will help to ensure that any programmatic errors are identified and rectified in a timely manner.

All RobotEye exceptions are derived from the C++ standard library 'exception' class, so user code only has to catch these exceptions. The class reference indicates which functions throw exceptions, and what the likely causes of those exceptions may be.

An example showing the use of an appropriate try-catch block is shown below;

```
#include <RobotEye.h>

#include <exception>
#include <iostream>

int main( void )
{
    try
    {
        ocular::RobotEye myRobotEye( "169.254.99.205" );
    }
    catch( std::exception & e )
    {

```

```

        std::cout << "Caught Exception: " << e.what() << std::endl;
        return 1;
    }

    return 0;
}

```

5.2.4 A Motion Example

This example demonstrates the use of the RobotEye object in its 'velocity mode'. The RobotEye is commanded to set its azimuth and elevation axis to specific speeds, then record the resulting motion to file for a period of 10 seconds before terminating.

```

#include <RobotEye.h>

#include <exception>
#include <iostream>

int main( void )
{
    try
    {
        // connect to RobotEye
        ocular::RobotEye myRobotEye( "169.254.99.205" );

        // home the RobotEye
        myRobotEye.Home();

        // Now command the aperture velocities
        myRobotEye.SetApertureRates( 1.1, 4.0 );

        // log 10000 ms of data to 'log.csv'
        myRobotEye.LogControllerData( "log.csv", 10000 );
    }
    catch( std::exception & e )
    {
        std::cout << "Caught Exception: " << e.what() << std::endl;
        return 1;
    }

    return 0;
}

```

5.3 Example Code

Please see the 'examples' subdirectory of the Robot Eye Library install directory. This directory contains many examples showing the use of the different control modes of the RobotEye device.

For information on compilation and running of these examples, see [Building Applications](#).

Chapter 6

Building Applications

6.1 Setting Up Your Project

This section describes how to set up a C++ project to use the RobotEye class library.

In general, the project only needs to know where the header file for the library is, and the name and location of the import library. In linux the shared object file (.so) doubles as the import library.

Note that once your project has been built, it will not execute unless the shared library is in one of the library loader search paths. See [this Wikipedia article](#) for a guide to the search paths used for your operating system.

6.1.1 Using CMake

CMake is a build tool that aids in setting up a project in an operating system and compiler independent manner. It is used by Ocular Robotics Pty. Ltd. on its internal projects, and therefore its use can greatly simplify the process of setting up a new project for use with the RobotEye library.

CMake can be downloaded from [the CMake website](#).

Linux users can use their favourite package manager to install CMake. It is recommended you also install CMake (a command line version of CMake which uses the curses library to emulate a simple GUI). There is also a QT based GUI for CMake which is identical to the Windows GUI.

CMake uses a script (CMakeLists.txt) to encode the rules about how a project should be built. These scripts will usually reside in the same directory as the source code for the project. For a simple application, it will look similar to the following;

```
# don't allow any version of CMake below this - or else it probably won't work
CMAKE_MINIMUM_REQUIRED( VERSION 2.6 )

# name for the project
PROJECT( Example1 )

# this project requires the RobotEye Lib, so find it
FIND_PACKAGE( RobotEye REQUIRED )

# set the include directory for the RobotEye lib
INCLUDE_DIRECTORIES( ${RobotEye_INCLUDE_DIR} )

# name the executable, and add source to it
ADD_EXECUTABLE( Example1 Example1.cpp )

# link to the RobotEye library
TARGET_LINK_LIBRARIES( Example1 ${RobotEye_LIBRARIES} )
```

The script above simply gives a name to the project, says that the RobotEye library is required, then adds the RobotEye include directory to the project, specifies that an executable 'Example1' should be built from 'Example1.cpp' and linked to the RobotEye import library. These CmakeLists.txt files are included with the examples supplied with the RobotEye library and can be very simply modified for your own projects.

The following steps outline how projects are built using CMake (identical for Windows and the Linux QT GUI).

- First, download and install CMake, following the instructions at [the CMake website](#).
- Run the CMake GUI
- Now tell CMake where the 'source code' is and where to build the 'binaries' (ideally these will be different directories i.e. an out-of-source build)
- Click 'Configure'
 - Answer the questions (create directory -> yes) (compiler -> Visual Studio 9 2008)
 - If the default RobotEye install paths are used, CMake should automatically find the RobotEye library resulting in the CMake GUI looking similar to the screen shot below

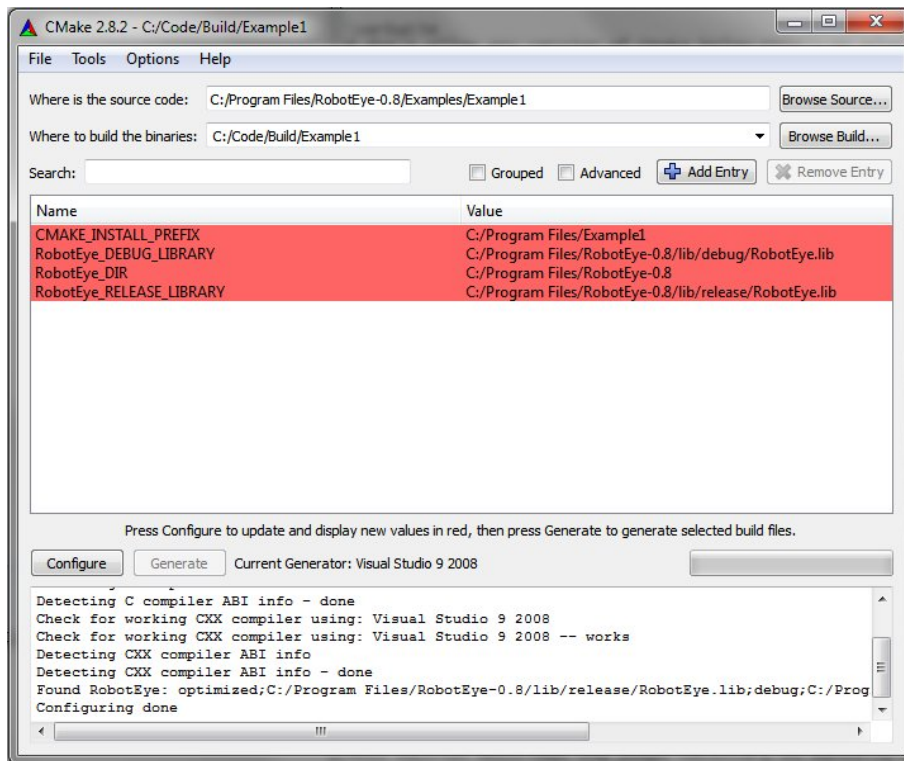


Figure 6.1: The CMake GUI for the 'Example1' Project

- Now click 'Configure' once more
- Now click 'Generate'

Congratulations! - you should now have a Visual Studio solution ready to build and execute in the directory you gave CMake as the 'binaries' directory (in the example "C:\Code\Build\Example1\Example1.sln")

- If you receive the CMake error "Error in configuration process...", it is likely to be because the RobotEye library was not found.

- In this case, you will need to manually set the CMake variable 'RobotEye_DIR' to the top level directory of the RobotEye library (for example "C:/Program Files/RobotEye-0.8" - note the Unix style slashes)
- Then, as before, click 'Configure', followed by 'Generate'.

6.1.2 Manually Setting up Projects

6.1.2.1 Microsoft Visual Studio

This section describes the steps necessary to manually configure Microsoft Visual Studio to build a project using the RobotEye class library

The following steps will allow compilation and execution of any of the examples given in [Using the RobotEye API](#).

- In the Visual Studio IDE, create a new project using the following steps
 - File -> New -> Project
 - Visual C++ -> General -> Empty Project
 - Give a name and location for the project
 - Click OK
- In the Solution Explorer, right click the 'Source Files' folder then Add -> Existing Item
 - Navigate to the cpp file you wish to use, select it, then click OK
- Now set the project properties with the following steps
 - Project -> Properties (or Alt+F7)
 - Configuration Properties -> C/C++ -> General
 - Add the full path to the directory containing [RobotEye.h](#) to the 'Additional Include Directories' box (e.g. "C:\Program Files\RobotEye-0.8\include" (the quotation marks must be included))
 - Click Apply
 - Configuration Properties -> Linker -> Input
 - Add the full path to the RobotEye.lib file to the 'Additional Dependencies' box (e.g. "C:\Program Files\RobotEye-0.8\lib\debug\RobotEye.lib" (the quotation marks must be included))
 - Click Apply
 - Configuration Properties -> C/C++ -> Code Generation
 - Set the 'Runtime Library' to 'Multi-threaded DLL (/MD)' if linking against the release version of RobotEye.lib
 - Set the 'Runtime Library' to 'Multi-threaded Debug DLL (/MDd)' if linking against the debug version of RobotEye.lib
 - Click Apply
 - Click OK
- You can now build and run the project

Remember to put the RobotEye.dll somewhere in the library loader's search path.

Chapter 7

Namespace Documentation

7.1 ocular Namespace Reference

the Ocular Robotics Pty Ltd default namespace.

Classes

- class [RobotEye](#)
The [RobotEye](#) class.
- class [RobotEyeCallback](#)
The [RobotEyeCallback](#) class.

7.1.1 Detailed Description

the Ocular Robotics Pty Ltd default namespace. All C++ software from ocular robotics is within the ocular namespace. In your implementation you may choose to add the

```
using namespace ocular;
```

command to access the [RobotEye](#) class directly

Chapter 8

Class Documentation

8.1 ocular::RobotEye Class Reference

The [RobotEye](#) class.

```
#include <RobotEye.h>
```

Public Types

- enum [eyeIndex_t](#) {
[AllEyes](#) = -1, [FirstEye](#), [SecondEye](#), [ThirdEye](#),
[FourthEye](#) }
The eyeIndex enumerated type.
- enum [axisIndex_t](#) { [AzimuthAxis](#) = 0, [ElevationAxis](#) }
The axisIndex enumerated type.
- enum [digitalBit_t](#) {
[NoOutput](#) = 0, [Output1](#) = 1, [Output2](#) = 2, [Output3](#) = 4,
[Output4](#) = 8, [Output5](#) = 16, [Output6](#) = 32, [Output7](#) = 64,
[Output8](#) = 128 }
The digitalBit enumerated type.
- enum [controlMode_t](#) {
[ContourMode](#) = 0, [HomingMode](#), [PositionMode](#), [TrackingMode](#),
[VectorMode](#), [VelocityMode](#) }
The control mode enumerated type.

Public Member Functions

- [RobotEye](#) (std::string address)
The default [RobotEye](#) constructor.
- [~RobotEye](#) (void)
The default [RobotEye](#) destructor.
- std::string [GetLibraryVersion](#) (void)
Get the current library version.
- void [GetSystemInfo](#) (std::string &systemType, unsigned int &variant)

Get system information This includes the mnemonic that describes the [RobotEye](#) system currently under control.

- double [GetSerialNumber](#) (void)

Get the serial number of the [RobotEye](#) system.
- void [SetControlSettings](#) (const [controlMode_t](#) mode, const std::string config, const [axisIndex_t](#) axisIndex, const [eyeIndex_t](#) eyeIndex=FirstEye)

This method allows the default controller settings to be overwritten.
- void [SetAccelerationParams](#) (const [controlMode_t](#) mode, const double acceleration, const double deceleration, const [eyeIndex_t](#) eyeIndex=FirstEye)

This method allows the default controller acceleration settings to be overwritten.
- void [GetAccelerationParams](#) (const [controlMode_t](#) mode, double &acceleration, double &deceleration, const [eyeIndex_t](#) eyeIndex=FirstEye)

This method allows the current controller acceleration settings to be read.
- void [SetDigitalOutput](#) (const unsigned char bits=NoOutput)

This method allows a user to manipulate the general purpose digital output bits on the [RobotEye](#) controller (if available).
- void [LogControllerData](#) (const std::string &fileName, const unsigned long milliseconds, const [eyeIndex_t](#) eyeIndex=FirstEye)

Method used to log aperture angles to file.
- void [ReceiveControllerData](#) ([RobotEyeCallback](#) *callback, const unsigned long milliseconds, const [eyeIndex_t](#) eyeIndex=FirstEye)

Method used to pass aperture angles to a user callback.
- void [Home](#) (const double homingVelocity=1.5, const [eyeIndex_t](#) eyeIndex=FirstEye)

Method used to send the specified [RobotEye](#) to its home, or zero position.
- void [SetApertureRates](#) (const double azimuthRate, const double elevationRate, const [eyeIndex_t](#) eyeIndex=FirstEye)

Method used to put the [RobotEye](#) into its velocity mode.
- void [SetApertureAngles](#) (const double azimuth, const double elevation, const double apertureSpeed=1.5, const [eyeIndex_t](#) eyeIndex=FirstEye)

Method used to point the [RobotEye](#) aperture at a specific angle, specified in degrees, in the default [RobotEye](#) coordinate system.
- void [SetApertureAngles](#) (const std::vector< double > &azimuths, const std::vector< double > &elevations, const double apertureSpeed)

Overload of [SetApertureAngles\(\)](#) used to point the [RobotEye](#) aperture at a specific angle, specified in degrees, in the default [RobotEye](#) coordinate system.
- void [TrackApertureAngles](#) (const double azimuth, const double elevation, const double trackingSpeed=1.5, const [eyeIndex_t](#) eyeIndex=FirstEye)

Method used to point the [RobotEye](#) aperture at a specific angle, specified in degrees, in the default [RobotEye](#) coordinate system.
- void [SetBoundedElevationScan](#) (const double azimuthRate, const double lowerElevation, const double upperElevation, const double lineSpacing, const [eyeIndex_t](#) eyeIndex=FirstEye)

Method used to set up spiral trajectories in aperture space.
- void [SetRegionScan](#) (const double azimuthRate, const double initialAzimuth, const double initialElevation, const double deltaAzimuth, const double deltaElevation, const double lineSpacing, const [eyeIndex_t](#) eyeIndex=FirstEye)

Method used to set up and execute raster-like trajectories in aperture space.
- void [SetPiecewiseLinearScan](#) (const std::vector< double > &azimuthAngles, const std::vector< double > &elevationAngles, const std::vector< double > &velocities, const [eyeIndex_t](#) eyeIndex=FirstEye)

Method used to generate arbitrary trajectories in a piecewise linear fashion.
- void [StopMotion](#) (const [eyeIndex_t](#) eyeIndex=AllEyes)

Method used to stop/abort the motion of the specified [RobotEye](#).

- void [GetApertureAngles](#) (std::vector< double > &azimuth, std::vector< double > &elevation, const [eyeIndex_t](#) eyeIndex=AllEyes)

Method used to query the current aperture angles of the [RobotEye](#) device.

8.1.1 Detailed Description

The [RobotEye](#) class.

This class contains all of the public programmatic interfaces to the [RobotEye](#) system. The class can be instantiated multiple times to connect to [RobotEye](#) systems on different IP addresses. Only one instance of this class should be used per [RobotEye](#) system. The eyeIndex parameter will not normally be used - it is only for custom designs where multiple RobotEyes are connected to a single [RobotEye](#) controller.

8.1.2 Member Enumeration Documentation

8.1.2.1 enum ocular::RobotEye::axisIndex_t

The axisIndex enumerated type.

This enum is used to specify the axis that a particular command may apply to, possible values are: [ocular::RobotEye::AzimuthAxis](#) and [ocular::RobotEye::ElevationAxis](#)

Enumerator:

AzimuthAxis indicates command applies to the RobotEye's Azimuth axis

ElevationAxis indicates command applies to the RobotEye's Elevation axis

8.1.2.2 enum ocular::RobotEye::controlMode_t

The control mode enumerated type.

This enum is used to specify the control mode that any tuning parameters may apply to.

Enumerator:

ContourMode unused - reserved for future use

HomingMode indicates command applies to the homing control parameters

PositionMode unused - reserved for future use

TrackingMode indicates command applies to tracking mode (TrackApertureAngles)

VectorMode indicates command applies to coordinated axis control (i.e. SetApertureAngles, the scan modes etc.)

VelocityMode indicates command applies to velocity control of individual axes (i.e. SetApertureRates)

8.1.2.3 enum ocular::RobotEye::digitalBit_t

The digitalBit enumerated type.

This enum is used to specify the specific bits of digital output to be turned on

Enumerator:

NoOutput used to set all digital outputs to zero

- Output1* used to turn digital output 1 on
- Output2* used to turn digital output 2 on
- Output3* used to turn digital output 3 on
- Output4* used to turn digital output 4 on
- Output5* used to turn digital output 5 on
- Output6* used to turn digital output 6 on
- Output7* used to turn digital output 7 on
- Output8* used to turn digital output 8 on

8.1.2.4 enum `ocular::RobotEye::eyeIndex_t`

The `eyeIndex` enumerated type.

This enum is used to specify which `RobotEye` a function specifies. By default, for all public functions, the `FirstEye` value will be used. This should only need to be changed in custom designs where multiple `RobotEyes` are connected to a single `RobotEye` controller for highly synchronised motion. Possible values are `ocular::RobotEye::FirstEye`, `ocular::RobotEye::SecondEye`, `ocular::RobotEye::ThirdEye`, `ocular::RobotEye::FourthEye` and `ocular::RobotEye::AllEyes`

Enumerator:

- AllEyes* indicates command applies to all eyes connected to the controller
- FirstEye* indicates command applies to only the first eye connected to the controller
- SecondEye* indicates command applies to only the second eye connected to the controller
- ThirdEye* indicates command applies to only the third eye connected to the controller
- FourthEye* indicates command applies to only the fourth eye connected to the controller

8.1.3 Constructor & Destructor Documentation

8.1.3.1 `RobotEye::RobotEye (std::string address)`

The default `RobotEye` constructor.

All resource acquisition is done during construction.

Parameters

<i>address</i>	- the IP address of the <code>RobotEye</code> expressed as a string i.e. "1.1.1.1"
----------------	--

Exceptions

<i>std::runtime_error</i>	- likely causes; incorrect network setup, attempt to create multiple <code>RobotEye</code> objects with the same IP address, configuration errors.
---------------------------	--

8.1.3.2 `RobotEye::~~RobotEye (void)`

The default `RobotEye` destructor.

Turns `RobotEye` motors off and releases all allocated resources.

8.1.4 Member Function Documentation

8.1.4.1 void RobotEye::GetAccelerationParams (const controlMode_t mode, double & acceleration, double & deceleration, const eyeIndex_t eyeIndex = FirstEye)

This method allows the current controller acceleration settings to be read.

This is not intended to be a commonly used function - for a detailed description of usage, contact Ocular Robotics Pty. Ltd.

Parameters

<i>mode</i>	- the motor control mode the new settings should be read from
<i>acceleration</i>	- returns the acceleration in deg/s/s
<i>deceleration</i>	- returns the deceleration in deg/s/s
<i>eyeIndex</i>	- the specified RobotEye , by default the FirstEye (usually there will be only one per IP address)

Exceptions

<i>std::exception</i>	- likely causes; 'mode', or specified RobotEye does not exist
-----------------------	---

Note

This is a non-blocking function call and will return immediately

8.1.4.2 void RobotEye::GetApertureAngles (std::vector< double > & azimuth, std::vector< double > & elevation, const eyeIndex_t eyeIndex = AllEyes)

Method used to query the current aperture angles of the [RobotEye](#) device.

Parameters

<i>azimuth</i>	- in degrees, a vector of azimuth angles for the queried RobotEye(s)
<i>elevation</i>	- in degrees, a vector of elevation angles for the queried RobotEye(s)
<i>eyeIndex</i>	- the specified RobotEye , by default AllEyes

Exceptions

<i>std::exception</i>	- likely causes; controller errors
-----------------------	------------------------------------

8.1.4.3 std::string RobotEye::GetLibraryVersion (void)

Get the current library version.

Useful to check the dll or so version currently in use, particularly if there are multiple copies on a system, and the user is not sure which one is being loaded.

Returns

a string containing the current library version

8.1.4.4 double RobotEye::GetSerialNumber (void)

Get the serial number of the [RobotEye](#) system.

Returns

a double precision value representing the serial number of the system.

8.1.4.5 void RobotEye::GetSystemInfo (std::string & systemType, unsigned int & variant)

Get system information This includes the mnemonic that describes the [RobotEye](#) system currently under control.

This will be one of the Ocular Robotics 'marketing' names. i.e. RE01, RE02 etc, and the variant, 01, 02 etc.

Parameters

<i>systemType</i>	- a string containing the system mnemonic
<i>variant</i>	- an int describing the system variant (01,02 etc)

8.1.4.6 void RobotEye::Home (const double homingVelocity = 1.5, const eyeIndex_t eyeIndex = FirstEye)

Method used to send the specified [RobotEye](#) to its home, or zero position.

i.e. zero Azimuth and zero Elevation This should be done prior to using other motion commands to achieve accurate positioning.

Parameters

<i>homingVelocity</i>	- in Hz (rotations per second)
<i>eyeIndex</i>	- the specified RobotEye , by default the FirstEye (usually there will be only one per IP address)

Exceptions

<i>std::exception</i>	- likely causes; out-of-bounds on input values
-----------------------	--

Note

This is a blocking function call, and will not return until the home position has been reached, and all motion ceased.

8.1.4.7 void RobotEye::LogControllerData (const std::string & fileName, const unsigned long milliseconds, const eyeIndex_t eyeIndex = FirstEye)

Method used to log aperture angles to file.

Format is Comma Separated Values (csv), in the following form;

```
TimestampInMilliseconds,AzimuthAngleInDegrees1,ElevationAngleInDegrees1, ... ,AzimuthAngleInDegreesN,ElevationAngleInDegreesN
```

where N is the number of specified RobotEyes

Parameters

<i>fileName</i>	- the name of the file to record the data in.
<i>milliseconds</i>	- the length of time, in milliseconds, to record data for
<i>eyeIndex</i>	- the specified RobotEye , by default the FirstEye (usually there will be only one per IP address)

Exceptions

<i>std::exception</i>	- likely causes; controller timeout
-----------------------	-------------------------------------

Note

This is a blocking function call, and will not return until the data has been logged.

8.1.4.8 `void RobotEye::ReceiveControllerData (RobotEyeCallback * callback, const unsigned long milliseconds, const eyeIndex_t eyeIndex = FirstEye)`

Method used to pass aperture angles to a user callback.

Functionally very similar to the LogControllerData function

Parameters

<i>callback</i>	- the user defined callback. Must be derived from a RobotEyeCallback object, implementing the ReceiveData function.
<i>milliseconds</i>	- the length of time, in milliseconds, to spend in this function
<i>eyeIndex</i>	- the specified RobotEye , by default the FirstEye

Exceptions

<i>std::exception</i>	- likely causes; controller timeout
-----------------------	-------------------------------------

Note

This is a blocking function call, and will not return until the specified time has elapsed

8.1.4.9 `void RobotEye::SetAccelerationParams (const controlMode_t mode, const double acceleration, const double deceleration, const eyeIndex_t eyeIndex = FirstEye)`

This method allows the default controller acceleration settings to be overwritten.

This is not intended to be a commonly used function - for a detailed description of usage, contact Ocular Robotics Pty. Ltd.

Parameters

<i>mode</i>	- the motor control mode the new settings should be applied to
<i>acceleration</i>	- the new acceleration in deg/s/s
<i>deceleration</i>	- the new deceleration in deg/s/s
<i>eyeIndex</i>	- the specified RobotEye , by default the FirstEye (usually there will be only one per IP address)

Exceptions

<i>std::exception</i>	- likely causes; 'mode', or specified RobotEye does not exist, or out-of-bounds input
-----------------------	---

Note

This is a non-blocking function call and will return immediately

8.1.4.10 void RobotEye::SetApertureAngles (const double *azimuth*, const double *elevation*, const double *apertureSpeed* = 1.5, const eyeIndex_t *eyeIndex* = FirstEye)

Method used to point the [RobotEye](#) aperture at a specific angle, specified in degrees, in the default [RobotEye](#) coordinate system.

The function waits for the motion to complete before returning control to the caller.

Parameters

<i>azimuth</i>	- in degrees (0 to 360)
<i>elevation</i>	- in degrees (-max to +max) where max depends on model (usually max = 35 degrees). 0 degrees is the horizontal plane.
<i>apertureSpeed</i>	- in Hz (the vector motion speed)
<i>eyeIndex</i>	- the specified RobotEye , by default the FirstEye (usually there will be only one per IP address)

Exceptions

<i>std::exception</i>	- likely causes; out-of-bounds on input values
-----------------------	--

Note

This is a blocking function call, and will not return until the specified aperture position has been reached, and all motion ceased.

8.1.4.11 void RobotEye::SetApertureAngles (const std::vector< double > & *azimuths*, const std::vector< double > & *elevations*, const double *apertureSpeed*)

Overload of [SetApertureAngles\(\)](#) used to point the [RobotEye](#) aperture at a specific angle, specified in degrees, in the default [RobotEye](#) coordinate system.

The function waits for the motion to complete before returning control to the caller. This method is intended to be used primarily for multi-RobotEye systems where coordinated motions to different aperture angles are desired. The RobotEyes should reach their destination simultaneously

Parameters

<i>azimuths</i>	- in degrees (0 to 360), a vector of angles where the zero'th element is for the First eye, the 1st element for the Second Eye, etc.
<i>elevations</i>	- in degrees (-max to +max) where max depends on model (usually max = 35 degrees). 0 degrees is the horizontal plane. A vector of angles where the zero'th element is for the First eye, the 1st element for the Second Eye, etc.
<i>apertureSpeed</i>	- in Hz (the vector motion speed)

Exceptions

<i>std::exception</i>	- likely causes; out-of-bounds on input values, more elements in vector than RobotEyes connected to controller
-----------------------	--

Note

This is a blocking function call, and will not return until the specified aperture position has been reached, and all motion ceased.

8.1.4.12 void RobotEye::SetApertureRates (const double *azimuthRate*, const double *elevationRate*, const eyeIndex_t *eyeIndex* = FirstEye)

Method used to put the RobotEye into its velocity mode.

In this mode, the aperture velocities are specified (in Hz). i.e. a 1.0 azimuthRate will result in 1 rotation per second. An elevationRate of 1.0 will result in the aperture sweeping from its lowest to highest extent in 1 second. This mode is a useful and simple method for creating sinusoidal sweeps of the aperture across its full extents.

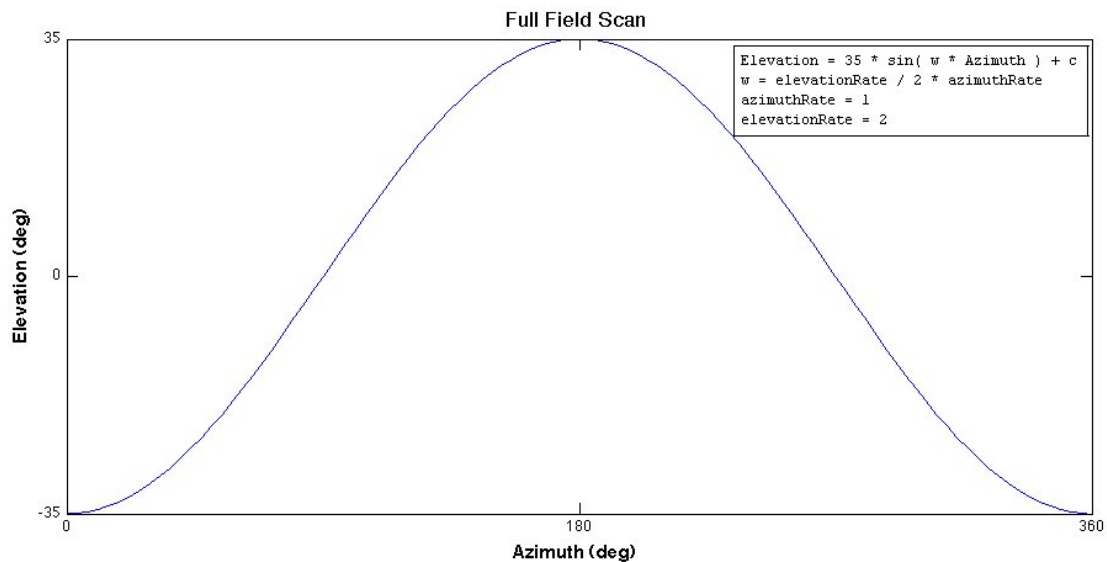


Figure 8.1: The trajectory of the RobotEye in Aperture (Azimuth-Elevation) space resulting from a call to SetApertureRates. The trajectory shown is for a 1 second period with the given input parameters. Note that the trajectory could start at any location.

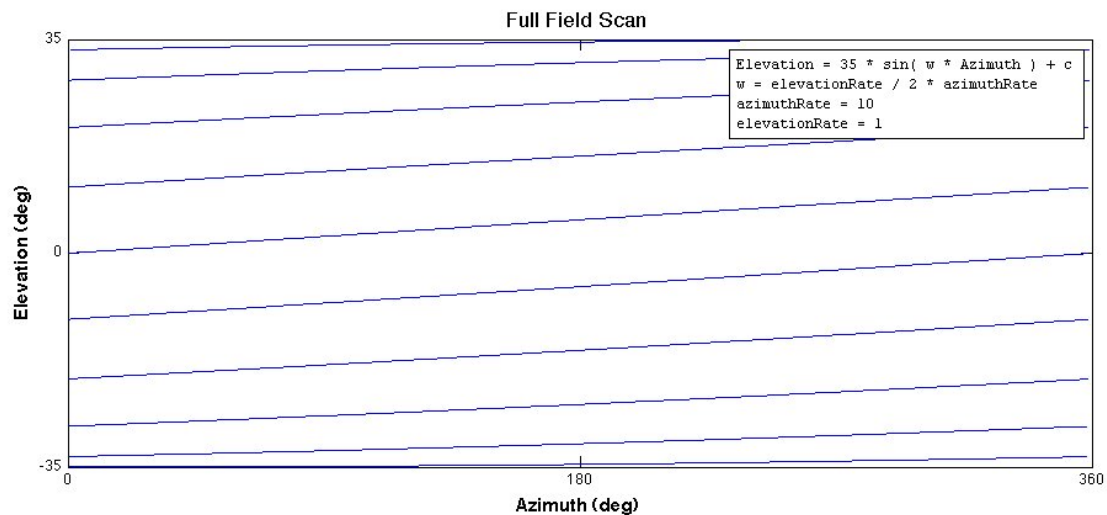


Figure 8.2: Another trajectory of the RobotEye in Aperture (Azimuth-Elevation) space resulting from a call to SetApertureRates. The trajectory shown is for a 1 second period with the given input parameters. Note that the trajectory could start at any location.

Parameters

<i>azimuthRate</i>	- in Hz (rotations per second)
<i>elevationRate</i>	- in Hz (vertical sweeps per second)
<i>eyeIndex</i>	- the specified RobotEye , by default the FirstEye (usually there will be only one per IP address)

Exceptions

<i>std::exception</i>	- likely causes; out-of-bounds on input values
-----------------------	--

Note

This is a non-blocking function call and will return immediately

8.1.4.13 void RobotEye::SetBoundedElevationScan (const double *azimuthRate*, const double *lowerElevation*, const double *upperElevation*, const double *lineSpacing*, const eyeIndex_t *eyeIndex* = FirstEye)

Method used to set up spiral trajectories in aperture space.

When viewed in azimuth-elevation space, this function will create a spiral from 'lowerElevation' to 'upperElevation' and back again continuously at the speed 'azimuthRate'. The density of the spiral is set by the 'lineSpacing' parameter. Note that it takes approximately 3 revolutions in azimuth of the [RobotEye](#) to transition into this mode. A subsequent call to this, or another motion function will be responded to within 1/4 of a revolution in azimuth.

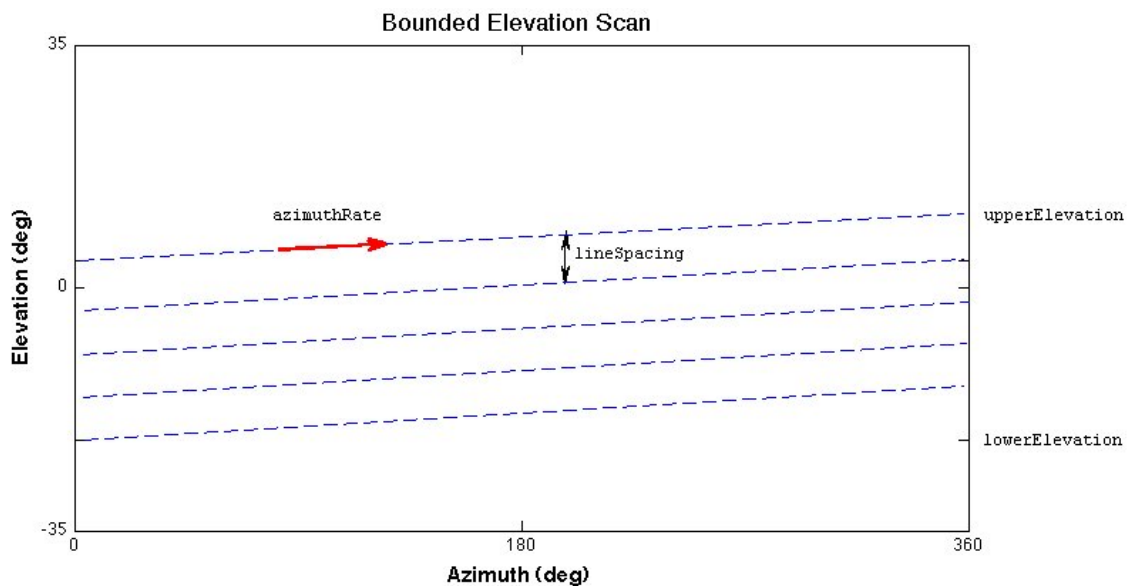


Figure 8.3: The Bounded Elevation Scan trajectory of the RobotEye in Aperture (Azimuth-Elevation) space, showing the relationship to the function's input parameters. Scan direction is bottom to top, and back again.

Parameters

<i>azimuthRate</i>	- in Hz (rotations per second)
<i>lowerElevation</i>	- in degrees: the lower elevation bound of the spiral scan.
<i>upperElevation</i>	- in degrees: the upper elevation bound of the spiral scan.
<i>lineSpacing</i>	- the vertical spacing, in degrees, between the (roughly) horizontal lines in the scan.
<i>eyeIndex</i>	- the specified RobotEye , by default the FirstEye (AllEyes is not valid in this mode)

Exceptions

<i>std::exception</i>	- likely causes; out-of-bounds on input values
-----------------------	--

Note

This is a non-blocking function call, and will return immediately. Note, as above that it takes approximately 3 revolutions to start the spiral, and, at most, 1/4 of a revolution to respond to new commands. Also note that this command cannot be given to multiple RobotEyes simultaneously.

8.1.4.14 void RobotEye::SetControlSettings (const controlMode_t mode, const std::string config, const axisIndex_t axisIndex, const eyeIndex_t eyeIndex = FirstEye)

This method allows the default controller settings to be overwritten.

This is not intended to be a commonly used function - for a detailed description of usage, contact Ocular Robotics Pty. Ltd.

Parameters

<i>mode</i>	- the motor control mode the new settings should be applied to
<i>config</i>	- the 'name' of the settings object to be used to overwrite the defaults for the specified mode
<i>axisIndex</i>	- the specified axis, no default, indicates which axis the settings should be applied to.
<i>eyeIndex</i>	- the specified RobotEye , by default the FirstEye (usually there will be only one per IP address)

Exceptions

<i>std::exception</i>	- likely causes; 'mode', 'config' or specified RobotEye does not exist, incorrectly formatted configuration files.
-----------------------	--

Note

This is a non-blocking function call and will return immediately

8.1.4.15 void RobotEye::SetDigitalOutput (const unsigned char bits = NoOutput)

This method allows a user to manipulate the general purpose digital output bits on the [RobotEye](#) controller (if available).

Any unspecified outputs are set to zero. On destruction of the [RobotEye](#) object, all outputs will be set to zero.

Parameters

<i>bits</i>	- the digital bits to be turned 'on' - formed by logically 'or'ing the members of the digitalBits_t enumerated type (e.g. SetDigitalOutput(ocular::RobotEye::Output4 ocular::RobotEye::Output8); to set outputs 4 and 8)
-------------	--

Exceptions

<i>std::exception</i>	- likely causes; called from passive connection
-----------------------	---

Note

This is a non-blocking function call and will return immediately

8.1.4.16 `void RobotEye::SetPiecewiseLinearScan (const std::vector< double > & azimuthAngles, const std::vector< double > & elevationAngles, const std::vector< double > & velocities, const eyeIndex_t eyeIndex = FirstEye)`

Method used to generate arbitrary trajectories in a piecewise linear fashion.

The method is passed a list of desired azimuth and elevation angles that define the path through aperture space. A vector of velocities is also given to complete the specification - where each velocity element corresponds to the desired velocity for the path segment starting at that location. By default the trajectory will loop via the shortest path back to the beginning on completion.

Parameters

<i>azimuthAngles</i>	- in degrees: the list of azimuth angles that define the endpoints of the linear segments
<i>elevationAngles</i>	- in degrees: the list of elevation angles that define the endpoints of the linear segments
<i>velocities</i>	- in Hz: the desired vector speed for the path segment starting at that location
<i>eyeIndex</i>	- the specified RobotEye , by default the FirstEye (AllEyes is not valid in this mode)

Exceptions

<i>std::exception</i>	- likely causes; out-of-bounds on input values
-----------------------	--

Note

This is a non-blocking function call, and will return immediately. Note, that it takes approximately 10 linear segments to respond to new commands. Also note that this command cannot be given to multiple RobotEyes simultaneously.

8.1.4.17 `void RobotEye::SetRegionScan (const double azimuthRate, const double initialAzimuth, const double initialElevation, const double deltaAzimuth, const double deltaElevation, const double lineSpacing, const eyeIndex_t eyeIndex = FirstEye)`

Method used to set up and execute raster-like trajectories in aperture space.

When viewed in azimuth-elevation space, this function will scan from the top-left corner of a rectangle defined by the initial Azimuth and Elevation (top-left corner) and the delta Azimuth and Elevation parameters (the width and height respectively). The density of the raster is given by the 'numberOfLines' parameter which defines the number of horizontal lines to scan within 'deltaElevation'. A subsequent call to this, or another motion function will be responded to within 3 scan lines.

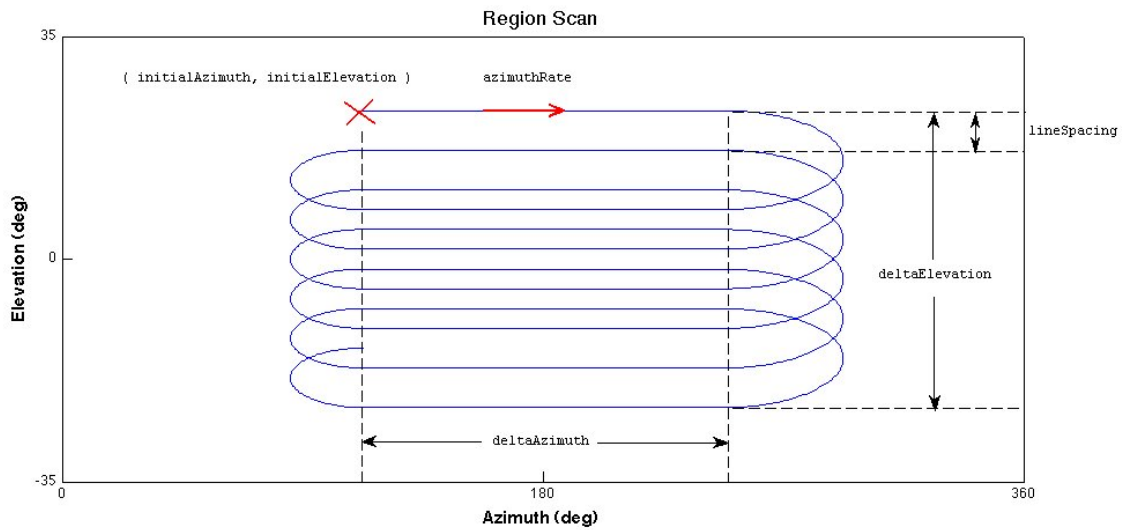


Figure 8.4: The partial trajectory of the RobotEye in Aperture (Azimuth-Elevation) space resulting in a call to SetRegionScan, showing the relationship to the function's input parameters.

Parameters

<i>azimuthRate</i>	- speed in Hz (rotations per second)
<i>initialAzimuth</i>	- in degrees: the Azimuth coordinate of the top-left corner of the scan
<i>initialElevation</i>	- in degrees: the Elevation coordinate of the top-left corner of the scan
<i>deltaAzimuth</i>	- in degrees: the 'width' of the raster-like scan
<i>deltaElevation</i>	- in degrees: the 'height' of the raster-like scan
<i>lineSpacing</i>	- the vertical spacing, in degrees, between the horizontal lines in the scan.
<i>eyeIndex</i>	- the specified RobotEye , by default the FirstEye (AllEyes is not valid in this mode)

Exceptions

<i>std::exception</i>	- likely causes; out-of-bounds on input values
-----------------------	--

Note

This is a non-blocking function call, and will return immediately. Note, as above that it takes approximately 3 scan lines to respond to new commands. Also note that this command cannot be given to multiple RobotEyes simultaneously.

8.1.4.18 void RobotEye::StopMotion (const eyeIndex_t eyeIndex = AllEyes)

Method used to stop/abort the motion of the specified [RobotEye](#).

Parameters

<i>eyeIndex</i>	- the specified RobotEye , by default AllEyes
-----------------	---

Exceptions

<i>std::exception</i>	- likely causes; out-of-bounds on input values
-----------------------	--

Note

This is a blocking function call - it will only return to the caller once motion of the specified [RobotEye](#)

has ceased.

8.1.4.19 void RobotEye::TrackApertureAngles (const double *azimuth*, const double *elevation*, const double *trackingSpeed* = 1.5, const eyeIndex_t *eyeIndex* = FirstEye)

Method used to point the [RobotEye](#) aperture at a specific angle, specified in degrees, in the default [RobotEye](#) coordinate system.

This function sends setpoints to the [RobotEye](#) controller. Previous setpoints are aborted once new setpoints are sent. Use this method for tracking dynamically moving targets, or similar.

Parameters

<i>azimuth</i>	- in degrees (0 to 360)
<i>elevation</i>	- in degrees (-max to +max) where max depends on model (usually max = 35 degrees). 0 degrees is the horizontal plane.
<i>trackingSpeed</i>	- in Hz (the vector motion speed)
<i>eyeIndex</i>	- the specified RobotEye , by default the FirstEye (usually there will be only one per IP address)

Exceptions

<i>std::exception</i>	- likely causes; out-of-bounds on input values
-----------------------	--

Note

This is a non-blocking function call, and will return immediately.

8.2 ocular::RobotEyeCallback Class Reference

The [RobotEyeCallback](#) class.

```
#include <RobotEyeCallback.h>
```

Public Member Functions

- [RobotEyeCallback](#) ()
The default [RobotEyeCallback](#) constructor.
- virtual [~RobotEyeCallback](#) ()
The default [RobotEyeCallback](#) destructor.
- virtual void [ReceiveData](#) (unsigned int timestamp, double azimuth, double elevation)=0
This virtual method should be overridden by derived user class(es).

8.2.1 Detailed Description

The [RobotEyeCallback](#) class.

This class is a pure virtual class that defines the interfaces used to pass data from the [RobotEye](#) controller to user code. A user should derive their own classes from this one, and override the pure virtual members.

Note

- this callback is executed in the threadspace of the [RobotEye](#) library. If user code is multi-threaded, it is the responsibility of the user to ensure their callback is threadsafe.

8.2.2 Member Function Documentation

8.2.2.1 `virtual void ocular::RobotEyeCallback::ReceiveData (unsigned int timestamp, double azimuth, double elevation)` [pure virtual]

This virtual method should be overridden by derived user class(es).

It is called by the [RobotEye](#) object whenever new data is available.

Parameters

<i>timestamp</i>	- the controller time when the measurements were taken
<i>azimuth</i>	- in degrees: the azimuth at 'timestamp'
<i>elevation</i>	- in degrees: the elevation at 'timestamp'

Exceptions

<code>std::runtime_error</code>	- likely causes; controller timeout
---------------------------------	-------------------------------------

Index

- ~RobotEye
 - ocular::RobotEye, 26
- AllEyes
 - ocular::RobotEye, 26
- axisIndex_t
 - ocular::RobotEye, 25
- AzimuthAxis
 - ocular::RobotEye, 25
- ContourMode
 - ocular::RobotEye, 25
- controlMode_t
 - ocular::RobotEye, 25
- digitalBit_t
 - ocular::RobotEye, 25
- ElevationAxis
 - ocular::RobotEye, 25
- eyeIndex_t
 - ocular::RobotEye, 26
- FirstEye
 - ocular::RobotEye, 26
- FourthEye
 - ocular::RobotEye, 26
- GetAccelerationParams
 - ocular::RobotEye, 27
- GetApertureAngles
 - ocular::RobotEye, 27
- GetLibraryVersion
 - ocular::RobotEye, 27
- GetSerialNumber
 - ocular::RobotEye, 27
- GetSystemInfo
 - ocular::RobotEye, 28
- Home
 - ocular::RobotEye, 28
- HomingMode
 - ocular::RobotEye, 25
- LogControllerData
 - ocular::RobotEye, 28
- NoOutput
 - ocular::RobotEye, 25
- ocular, 21
- ocular::RobotEye, 23
 - ~RobotEye, 26
 - AllEyes, 26
 - axisIndex_t, 25
 - AzimuthAxis, 25
 - ContourMode, 25
 - controlMode_t, 25
 - digitalBit_t, 25
 - ElevationAxis, 25
 - eyeIndex_t, 26
 - FirstEye, 26
 - FourthEye, 26
 - GetAccelerationParams, 27
 - GetApertureAngles, 27
 - GetLibraryVersion, 27
 - GetSerialNumber, 27
 - GetSystemInfo, 28
 - Home, 28
 - HomingMode, 25
 - LogControllerData, 28
 - NoOutput, 25
 - Output1, 25
 - Output2, 26
 - Output3, 26
 - Output4, 26
 - Output5, 26
 - Output6, 26
 - Output7, 26
 - Output8, 26
 - PositionMode, 25
 - ReceiveControllerData, 29
 - RobotEye, 26
 - SecondEye, 26
 - SetAccelerationParams, 29
 - SetApertureAngles, 29, 30
 - SetApertureRates, 30
 - SetBoundedElevationScan, 32
 - SetControlSettings, 33
 - SetDigitalOutput, 33
 - SetPiecewiseLinearScan, 33
 - SetRegionScan, 34
 - StopMotion, 35
 - ThirdEye, 26
 - TrackApertureAngles, 36
 - TrackingMode, 25
 - VectorMode, 25

- VelocityMode, 25
- ocular::RobotEyeCallback, 36
 - ReceiveData, 37
- Output1
 - ocular::RobotEye, 25
- Output2
 - ocular::RobotEye, 26
- Output3
 - ocular::RobotEye, 26
- Output4
 - ocular::RobotEye, 26
- Output5
 - ocular::RobotEye, 26
- Output6
 - ocular::RobotEye, 26
- Output7
 - ocular::RobotEye, 26
- Output8
 - ocular::RobotEye, 26

- PositionMode
 - ocular::RobotEye, 25

- ReceiveControllerData
 - ocular::RobotEye, 29
- ReceiveData
 - ocular::RobotEyeCallback, 37
- RobotEye
 - ocular::RobotEye, 26

- SecondEye
 - ocular::RobotEye, 26
- SetAccelerationParams
 - ocular::RobotEye, 29
- SetApertureAngles
 - ocular::RobotEye, 29, 30
- SetApertureRates
 - ocular::RobotEye, 30
- SetBoundedElevationScan
 - ocular::RobotEye, 32
- SetControlSettings
 - ocular::RobotEye, 33
- SetDigitalOutput
 - ocular::RobotEye, 33
- SetPiecewiseLinearScan
 - ocular::RobotEye, 33
- SetRegionScan
 - ocular::RobotEye, 34
- StopMotion
 - ocular::RobotEye, 35

- ThirdEye
 - ocular::RobotEye, 26
- TrackApertureAngles
 - ocular::RobotEye, 36
- TrackingMode
 - ocular::RobotEye, 25

- VectorMode
 - ocular::RobotEye, 25
- VelocityMode
 - ocular::RobotEye, 25