ROBOTEYE C++ LIBRARY

Reference Manual

VERSION 2.4.1.319

WED 5TH AUG, 2015

Ocular Robotics Pty Ltd
Unit F1, 13-15 Forrester Street
Kingsgrove, NSW 2208
Sydney, Australia

www.ocularrobotics.com

Contents

Chapter 1

RobotEye C++ 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.

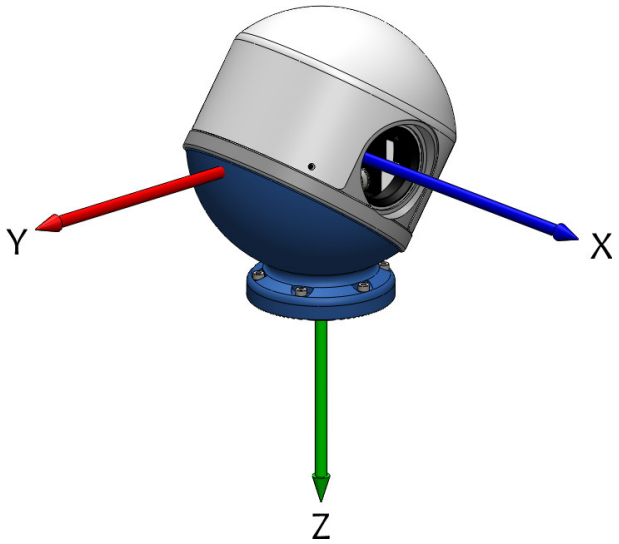


Figure 2.1: The RobotEye Coordinate System

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

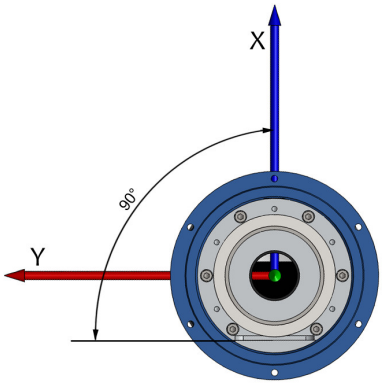


Figure 2.2: The RobotEye Azimuth Reference Surface

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

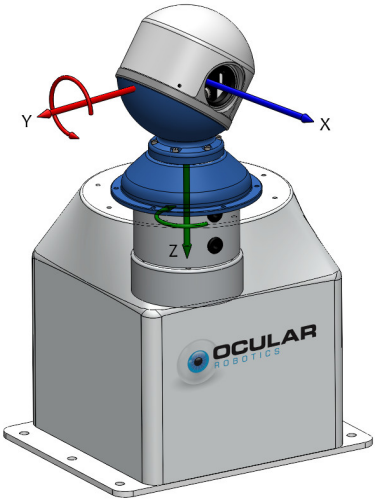


Figure 2.3: The RobotEye Elevation Reference Surface and Azimuth and Elevation Rotation Sign

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 clockwise when viewed from above, 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. The IP address of an eye can be changed using the RobotEyeManager utility available through the RobotEyeTools installer (recommended), or programatically with the RobotEyeManager class.

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 seperate 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 seperate network interface card on a seperate subnet.

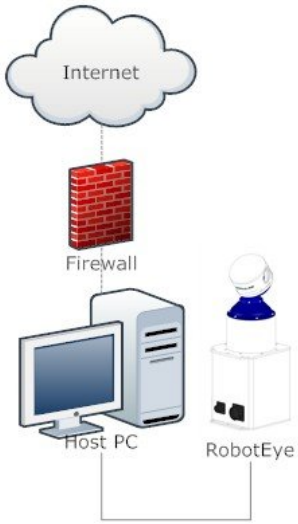


Figure 3.1: Network Schematic for Connecting a Host PC to a Single RobotEye

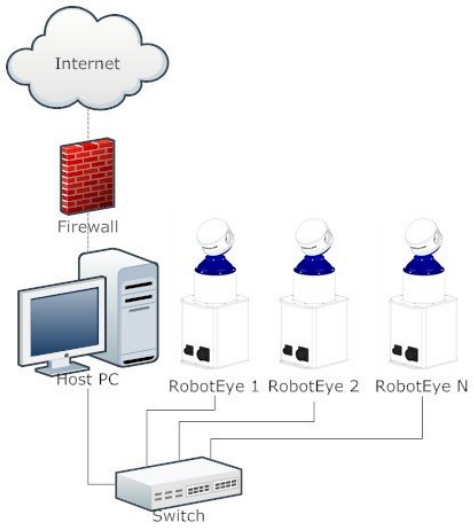


Figure 3.2: Network Schematic for Connecting a Host PC to Multiple RobotEyes

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.

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

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

Chapter 5

Class Index

5.1 Class List

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

Chapter 6

Class Documentation

6.1 ocular::ocular_rbe_obs_t Struct Reference

`#include <RobotEyeCommon.h>`

Public Attributes

- double azimuth
- double elevation
- double range
- unsigned short int intensity

6.1.1 Detailed Description

The Ocular Range Bearing Elevation observation type. This type is used by the RobotEye library to return range-bearing-elevation observations from Ocular Robotics laser scanner products.

6.1.2 Member Data Documentation

6.1.2.1 double ocular::ocular_rbe_obs_t::azimuth

The azimuth angle of the observation in degrees from 0 to 360

6.1.2.2 double ocular::ocular_rbe_obs_t::elevation

The elevation angle of the observation in degrees from -35 to 35

6.1.2.3 unsigned short int ocular::ocular_rbe_obs_t::intensity

The intensity recorded for this observation. When not present, will be set to 0

6.1.2.4 double ocular::ocular_rbe_obs_t::range

The range of the observation in meters

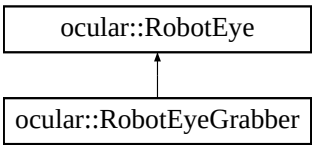
The documentation for this struct was generated from the following file:

- include/roboteye/RobotEyeCommon.h

6.2 ocular::RobotEye Class Reference

`#include <RobotEye.h>`

Inheritance diagram for ocular::RobotEye:



Public Member Functions

- RobotEye (std::string eyeIP)
- ocular_error_t StartLaser (unsigned short Freq, unsigned short Averaging, bool Intensity, unsigned short TargetPort)
- ocular_error_t StopLaser ()
- ocular_error_t Home ()
- ocular_error_t Home (ocular::RobotEyeNotificationCallbackClass *callbackPtr)
- ocular_error_t Stop ()
- ocular_error_t StartFullFieldScan (double AzRate, unsigned short NLines)
- ocular_error_t StartBoundedElevationScan (double AzRate, double elMin, double elMax, unsigned short NLines)
- ocular_error_t StartRegionScan (double AzRate, double azMin, double azMax, double elMin, double elMax, unsigned short NLines)
- ocular_error_t SetApertureAngles (double Az, double El, double Speed)
- ocular_error_t SetApertureAngles (double Az, double El, double Speed, ocular::RobotEyeNotificationCallbackClass *callbackPtr)
- ocular_error_t TrackApertureAngles (double Az, double El, double Speed)

- ocular_error_t SetAcceleration (double Acceleration)
- ocular_error_t SetIPSettings (std::string Serial, std::string DesiredIP)
- ocular_error_t GetApertureAngles (double &Az, double &El)
- ocular_error_t GetSerial (std::string &serial)
- ocular_error_t GetLastBlockingError (void)
- ocular_error_t GetLastNonBlockingError (void)
- unsigned int GetResponseTimeout (void)
- void SetResponseTimeout (unsigned int timeout)
- unsigned int GetMotionTimeout (void)
- void SetMotionTimeout (unsigned int timeout)
- ocular_error_t SetLaserGain (int gain)

Static Public Member Functions

- static std::string GetErrorString (ocular_error_t errorCode)

6.2.1 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.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 ocular::RobotEye::RobotEye (std::string *eyeIP*)

The default RobotEye constructor. All resource acquisition is done during construction.

Parameters

eyeIP	- the IP address of the RobotEye expressed as a string i.e. "10.-1.1.200"

Exceptions

std::runtime_error	- likely causes; incorrect network setup, configuration errors.

6.2.3 Member Function Documentation

6.2.3.1 ocular_error_t ocular::RobotEye::GetApertureAngles (double & *Az,* double & *El*)

A method to capture the current position of the RobotEye. This command will query the robot eye for it's current position. If the return value is NO_ERROR then Az and El will contain the current eye location.

Parameters

out	*&Az*	- The target location to store the eye azimuth in degrees
out	*&El*	- The target location to store the eye elevation in degrees

Note

This command should not be called at rates higher than 25 Hz. Future versions of the library will support a high-frequency streaming form of this command for rapid updates.

6.2.3.2 static std::string ocular::RobotEye::GetErrorString (ocular_error_t *errorCode*) [static]

A method to translate an ocular_error_t error code into a human-readable descriptive string.

Parameters

errorCode	- The error code to translate. If the error is not recognised, the string "Unknown Error" will be returned.

6.2.3.3 ocular_error_t ocular::RobotEye::GetLastBlockingError (void)

A method to retrieve the last error code returned by the RobotEye to a blocking transaction.

6.2.3.4 ocular_error_t ocular::RobotEye::GetLastNonBlockingError (void)

A method to retrieve the last error code returned by the Robot Eye to a non-blocking transaction.

6.2.3.5 unsigned int ocular::RobotEye::GetMotionTimeout (void)

A method to retrieve the current timeout for motion transactions. "Motion" transactions are commands where the response from the eye does not come until

the commanded motion is complete, such as Set Aperture Angles and Home.

6.2.3.6 unsigned int ocular::RobotEye::GetResponseTimeout (void)

A method to retrieve the current timeout for response transactions. "Response" transactions are the vast majority of the transactions with the RobotEye, where the response to the command is immediate.

6.2.3.7 ocular_error_t ocular::RobotEye::GetSerial (std::string & *serial*)

A method to acquire the serial number of the target RobotEye. This command will query the robot eye for it's serial number. If the return value is NO_ERROR, then serial will contain the serial number of the RobotEye.

Parameters

out	*&serial*	- The target location to store the eye serial string

6.2.3.8 ocular_error_t ocular::RobotEye::Home ( )

Perform homing routine. This command will cause a RobotEye to execute it's homing routine. The overloaded variant with no arguments is a blocking command, and will not return until the motion is complete or the eye has responded with an error.

Note

This is a blocking function call, and will not return until the home routine has been completed, and all motion ceased.
This command uses the motion timeout, as it is a movement command waiting for the completion of the move.
This command can interrupt an in-progress non-blocking SetAperture-Angles or Home command. When this is done, the callback for the previously executing non-blocking command will return with ERR_TIMEOUT.

6.2.3.9 ocular_error_t ocular::RobotEye::Home (ocular::-RobotEyeNotificationCallbackClass * *callbackPtr*)

Perform homing routine. This command will cause a RobotEye to execute it's homing routine. The overloaded variant with a callback handle is a *non-blocking* command, and will return immediately. The callback will notify the sender of successful or otherwise completion of the move.

Parameters

callbackPtr	- This is a pointer to a RobotEyeCallbackClass. The user should implement a derived-class with an implementation of the NotificationCallback virtual function to be executed when the motion is complete or an error has occured.

Note

This command uses the motion timeout, as it is a movement command waiting for the completion of the move.
A non-blocking motion command may be interrupted with any blocking motion command such as the scan patterns, SetApertureAngles, TrackApertureAngles, Stop or Home. When interrupted in this manner, the callback will fire when the motion timeout occurs with an ERR_TIMEOUT argument.

6.2.3.10 ocular_error_t ocular::RobotEye::SetAcceleration (double *Acceleration*)

Modify acceleration limits. 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

Acceleration	- the new acceleration in deg/s/s

6.2.3.11 ocular_error_t ocular::RobotEye::SetApertureAngles (double *Az,* double *El,* double *Speed*)

Blocking synchronised motion command. 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. It is intended to be the standard blocking method for pointing a Robot Eye.

Parameters

Az	- Azimuth angle in degrees (0 to 360)
El	- Elevatiuon angle in degrees (-max to +max) where max depends on model (usually max = 35 degrees). 0 degrees is the horizontal plane.
Speed	- in Hz (the vector motion speed). This is an upper speed limit only, depending on the current acceleration limit the eye may not reach the specified speed during the motion.

Note

This is a blocking function call, and will not return until the specified aperture position has been reached, and all motion ceased.
This command uses the Motion timeout, as it is a movement command waiting for the completion of the move.
This command can interrupt an in-progress non-blocking SetApertureAngles or Home command. When this is done, the callback for the previously executing non-blocking command will return with ERR_TIMEOUT.

6.2.3.12 ocular_error_t ocular::RobotEye::SetApertureAngles (double *Az,* double *El,* double *Speed,* ocular::RobotEyeNotificationCallbackClass * *callbackPtr*)

Non-Blocking synchronised motion command. Method used to point the RobotEye aperture at a specific angle, specified in degrees, in the default RobotEye coordinate system. This function will return immediately, and the callback will be executed when either motion is complete, an error has occured, or the appropriate timeout occurs. It is intended to be the standard non-blocking method for pointing a Robot Eye.

Parameters

Az	- Azimuth angle in degrees (0 to 360)
El	- Elevatiuon angle in degrees (-max to +max) where max depends on model (usually max = 35 degrees). 0 degrees is the horizontal plane.
Speed	- in Hz (the vector motion speed). This is an upper speed limit only, depending on the current acceleration limit the eye may not reach the specified speed during the motion.
callbackPtr	- This is a pointer to a RobotEyeCallbackClass. The user should implement a derived-class with an implementation of the NotificationCallback virtual function to be executed when the motion is complete or an error has occured.

Note

This command uses the Motion timeout, as it is a movement command waiting for the completion of the move.
A non-blocking motion command may be interrupted with any blocking motion command such as the scan patterns, SetApertureAngles, TrackApertureAngles, Stop or Home. When interrupted in this manner, the callback will fire when the motion timeout occurs with an ERR_TIMEOUT argument.

6.2.3.13 ocular_error_t ocular::RobotEye::SetIPSettings (std::string *Serial,* std::string *DesiredIP*)

This method allows the IP address of the robot eye to be changed. Please ensure that the target address is accessable from the host machine. For assistance recovering lost eyes, please contact Ocular Robotics pty. ltd.

Parameters

Serial	- The Serial number of the eye to have it's IP changed. This is required as a safeguard against unintentional IP changes.
DesiredIP	- The desired IP address to change the eye to.

Note

When having it's IP changed, the eye will send the acknowledgement from the initial IP, then perform the address change. It will be required to construct a new RobotEye object to communicate with the eye on the new address.

6.2.3.14 ocular_error_t ocular::RobotEye::SetLaserGain (int *gain*)

Set the laser gain. This command will adjust the gain of the RE05's laser range finder. At power-on, the gain defaults to 0. This setting is not preserved through power-cycles of the device. High gains can result in improved ranging performance to long-range or low-reflectivity targets, but can also result in increased noise and false returns in the LIDAR data received.

Parameters

gain	- Desired laser gain value. Valid values are 0, 1, 2 and 3, where 3 is maximum gain, and 0 is minimum gain

6.2.3.15 void ocular::RobotEye::SetMotionTimeout (unsigned int *timeout*)

A method to set the current timeout for motion transactions. "Motion" transactions are commands where the response from the eye does not come until the commanded motion is complete, such as Set Aperture Angles and Home. Default value is 2500msec, and this should be adequate for the majority of applications.

Note

This parameter is a function of the RobotEye driver api and not of the Robot Eye device. The value is only updated for the life of the current RobotEye object.

A commanded timeout value of 0 will cause the RobotEye API to block indeifinately until a response from the RobotEye is recieved. As the UDP interface has no guarantees of delivery, and packet loss of both command and response packets is possible, this is not recommended as it may cause the API to hang.

6.2.3.16 void ocular::RobotEye::SetResponseTimeout (unsigned int *timeout*)

A method to set the current timeout for response transactions. "Response" transactions are the vast majority of the transactions with the RobotEye, where the response to the command is immediate. Default value is 500msec, and this should be adequate for the majority of applications.

Parameters

timeout	- Desired timeout in milliseconds.

Note

This parameter is a function of the RobotEye driver api and not of the Robot Eye device. The value is only updated for the life of the current RobotEye object.
A commanded timeout value of 0 will cause the RobotEye API to block indeifinately until a response from the RobotEye is recieved. As the UDP interface has no guarantees of delivery, and packet loss of both command and response packets is possible, this is not recommended as it may cause the API to hang.

6.2.3.17 ocular_error_t ocular::RobotEye::StartBoundedElevationScan (double *AzRate,* double *elMin,* double *elMax,* unsigned short *NLines*)

Begin executing a bounded-elevation scan pattern. 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 'AzRate'. The density of the spiral is set by the 'NLines' 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.

Parameters

AzRate	- in Hz (rotations per second)
elMin	- in degrees: the lower elevation bound of the spiral scan.
elMax	- in degrees: the upper elevation bound of the spiral scan.
NLines	- The number of lines to perform over the elevation range.

Note

It takes approximately 3 revolutions to start the spiral, and, at most, 1/4 of a revolution to respond to new commands.

6.2.3.18 ocular_error_t ocular::RobotEye::StartFullFieldScan (double *AzRate,* unsigned short *NLines*)

Begin executing a full-field scan pattern. Method used to set up spiral trajectories in aperture space. When viewed in azimuth-elevation space, this function will create a spiral from the minimum to maximum elevation and back again continuously at the speed 'AzRate'. The density of the spiral is set by the 'N-Lines' 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.

Parameters

AzRate	- in Hz (rotations per second)
NLines	- The number of lines to perform over the elevation range.

Note

It takes approximately 3 revolutions to start the spiral, and, at most, 1/4 of a revolution to respond to new commands.

6.2.3.19 ocular_error_t ocular::RobotEye::StartLaser (unsigned short *Freq,* unsigned short *Averaging,* bool *Intensity,* unsigned short *TargetPort*)

Start the laser in an RE05 for an external listener. This command will activate the laser module within an RE05, and begin streaming range-bearing-elevation packets to the desired port on the machine transmitting this command. It is intended to be used with a stand-alone listener such as the Point Cloud Library RE05_Grabber class. Use the alternative form with a callback as an argument if it is desired to receive the laser data within this RobotEye object.

Parameters

Freq	- The laser sampling rate in Hz. Valid values are from 1 to 10,000, or 30,000 with no intensity
Averaging	- The number of laser samples to average to produce each range measurement. Not supported for 30 kHz sample rate.
Intensity	- Enable streaming of intensity values along with range-bearing-elevation observations. Not supported for 30 kHz sample rate.
TargetPort	- The target port number on the host machine for laser data streaming. This should correspond to the port number of an already created data listener.

6.2.3.20 ocular_error_t ocular::RobotEye::StartRegionScan (double *AzRate,* double *azMin,* double *azMax,* double *elMin,* double *elMax,* unsigned short *NLines*)

Begin executing a region scan pattern. Method used to set up and execute raster-like trajectories in aperture space. When viewed in azimuth-elevation space, this function will scan within a rectangle defined by the given azimuth and elevation limits. The density of the raster is given by the 'NLines' parameter which defines the number of horizontal lines to scan within the elevation range. A subsequent call to this, or another motion function will be responded to within 3 scan lines.

Parameters

AzRate	- speed in Hz (rotations per second). This is an upper speed limit only, depending on the current acceleration limit the eye may not reach the specified speed during the motion.
azMin	- in degrees: the lower azimuth bound of the scan
azMax	- in degrees: the upper azimuth bound of the scan
elMin	- in degrees: the lower elevation bound of the scan.
elMax	- in degrees: the upper elevation bound of the scan.
NLines	- The number of lines to perform over the elevation range.

Note

It may take up to 3 scan lines to respond to new commands.

6.2.3.21 ocular_error_t ocular::RobotEye::Stop ( )

Stop any in-progress motion. This command will cause a RobotEye to immediately cease motion. It can be used to terminate an in-progress non-blocking Set Aperture Angles or Home instruction, in which case the non-blocking callback will return ocular::ocular_error_t::ERR_TIMEOUT

6.2.3.22 ocular_error_t ocular::RobotEye::StopLaser ( )

Stop the laser in an RE05. This command will de-activate the laser module within an RE05, and stop the data streaming.

6.2.3.23 ocular_error_t ocular::RobotEye::TrackApertureAngles (double *Az,* double *El,* double *Speed*)

Rapid-response motion command. 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 where extremely low-latency motion control is required.

Parameters

Az	- in degrees (0 to 360)
El	- in degrees (-max to +max) where max depends on model (usually max = 35 degrees). 0 degrees is the horizontal plane.
Speed	- in Hz (the vector motion speed)

Note

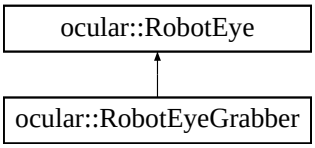
This is a non-blocking function call, and will return immediately.
This command will cause the eye to begin motion more rapidly than the Set Aperture Angles command, however it does not provide a notification when motion is complete, and it may have longer settling times than an equivalent Set Aperture Angles command.

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

- include/roboteye/RobotEye.h

6.3 ocular::RobotEyeGrabber Class Reference

Inheritance diagram for ocular::RobotEyeGrabber:



Public Member Functions

- **RobotEyeGrabber** (std::string EyeIP)
- ocular_error_t StartLaser (unsigned short Freq, unsigned short Averaging, bool Intensity, ocular::RobotEyeLaserDataCallbackClass *callbackPtr)
- void StartListening (ocular::RobotEyeLaserDataCallbackClass *callbackPtr)

Additional Inherited Members

6.3.1 Member Function Documentation

6.3.1.1 ocular_error_t ocular::RobotEyeGrabber::StartLaser (unsigned short *Freq*, unsigned short *Averaging*, bool *Intensity*, ocular::RobotEyeLaserDataCallbackClass * *callbackPtr*)

Start the laser in an RobotEye Lidar. This command will activate the laser module within an RobotEye Lidar, and begin streaming range-bearing-elevation packets to this RobotEyeGrabber object. Each time a packet is received, the data will be converted into a std::vector of ocular::ocular_rbe_obs_t strucutres, and the LaserDataCallback method within the provided callback class will be executed.

This is an overload of the base RobotEye class RobotEye::StartLaser which automatically creates the receiver and fills in the appropriate port number argument.

Parameters

Freq	- The laser sampling rate in Hz. Valid values are from 1 to 10,000, or 30,000 with no intensity
Averaging	- The number of laser samples to average to produce each range measurement. Not supported for 30 kHz sample rate.
Intensity	- Enable streaming of intensity values along with range-bearing-elevation observations. Not supported for 30 kHz sample rate.
callbackPtr	- A pointer to the derived callback class who's LaserDataCallback method is to be executed when data is received.

6.3.1.2 void ocular::RobotEyeGrabber::StartListening (ocular::RobotEyeLaserDataCallbackClass * *callbackPtr*)

Start listening to laser data on the callback without starting the laser. This command will activate the listening for external incoming laser data into this RobotEyeGrabber object without starting the laser itself. This should only be used when this application is not starting the laser itself, but only receiving data from the laser. Each time a packet is received, the data will be converted into a std::vector of ocular::ocular_rbe_obs_t strucutres, and the LaserDataCallback method within the provided callback class will be executed.

Parameters

callbackPtr	- A pointer to the derived callback class who's LaserDataCallback method is to be executed when data is received.

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

- include/roboteye/RobotEyeGrabber.h

6.4 ocular::RobotEyeLaserDataCallbackClass Class Reference

Callback class for Laser Scanner Data.

`#include <RobotEyeCallbacks.h>`

Public Member Functions

- virtual void **LaserDataCallback** (std::vector< ocular::ocular_rbe_obs_t > observations, unsigned int timestamp=0)=0

6.4.1 Detailed Description

Callback class for Laser Scanner Data.

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

- include/roboteye/RobotEyeCallbacks.h

6.5 ocular::RobotEyeManager Class Reference

`#include <RobotEyeManager.h>`

Public Member Functions

- RobotEyeManager ()
- ocular_error_t SearchForEyes (unsigned int timeout, bool blocking=true, ocular::RobotEyeManagerCallback *callbackPtr=NULL)
- void StopSearching (void)
- ocular_error_t WaitForTransactionComplete (void)
- bool CheckIP (std::string ip)
- bool IsKnownSerial (std::string serial)
- ocular_error_t SetIP (std::string serial, std::string desiredIP)
- std::string GetIPFromSerial (std::string serial)
- RobotEyeMap GetFoundEyeList (void)
- std::string GetFormattedFoundEyeList (void)
- void ResetEyeList (void)
- unsigned int GetNumEyesInList (void)

6.5.1 Detailed Description

The RobotEyeManager class. This class provides functionality to search a network for eyes, and to update the serial numbers of any eyes present on the network. The basic search functionality is implemented through the SearchForEyes method, and an internal RobotEyeMap of eyes found since construction or the last call to ResetEyeList. It uses full broadcast packets for searching, so in most cases will find eyes outside the subnet of the network adapters in use, however firewalls and local system configuration may prevent this if not correctly configured.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 ocular::RobotEyeManager::RobotEyeManager ( )

Default constructor for RobotEyeManager class. This will acquire all required resources for management of all eyes on a network.

6.5.3 Member Function Documentation

6.5.3.1 bool ocular::RobotEyeManager::CheckIP (std::string *ip*)

Helper method to check for valid IPV4 address. This is a simple helper method that confirms the input string is of the form W.X.Y.Z where W, X, Y and Z are all within the range 0:255 inclusive.

Parameters

ip	- IP address string to check for validity

Returns

True if ip is a valid IPV4 address, false otherwise.

6.5.3.2 std::string ocular::RobotEyeManager::GetFormattedFoundEyeList (void)

Get list of eyes formatted as a std::string for display. This method will format the current RobotEyeMap in a tabular manner for display. Intended for use in console applications.

6.5.3.3 RobotEyeMap ocular::RobotEyeManager::GetFoundEyeList (void)

Get list of eyes found by SearchForEyes. This method will return a map of strings containing the list of found robot eye devices on the network and their

associated IP addresses. To fill this list, a prior call to SearchForEyes must be made.

6.5.3.4 std::string ocular::RobotEyeManager::GetIPFromSerial (std::string *serial*)

Get the IP address of an eye from a known serial number. This method is used to retrieve the IPv4 address of a robot eye previously seen by a SearchForEyes command as a string.

Parameters

serial	- The serial number of the eye for which the IP is desired

Returns

The IPv4 address of the requested eye. If the eye has not been seen on the network, an empty string will be returned.

6.5.3.5 unsigned int ocular::RobotEyeManager::GetNumEyesInList (void)

Get number of eyes currently in the RobotEyeMap. Method to extract number of eyes seen on the network as a result of SearchForEyes calls since startup or the last call to ResetEyeList.

6.5.3.6 bool ocular::RobotEyeManager::IsKnownSerial (std::string *serial*)

Check whether serial has been seen. Check internal map of serial numbers to check whether the desired serial has been seen on the network as a result of a previous or ongoing SearchForEyes command.

Parameters

serial	- Serial number to check internal map for.

Returns

True if provided serial number has been seen on the network, false otherwise.

6.5.3.7 void ocular::RobotEyeManager::ResetEyeList (void)

Erase the current internal RobotEyeMap. This method will clear the current internal map of found robot eyes. It is the only method by which eyes which may have previously been on the network but are no longer present can be found.

6.5.3.8 ocular_error_t ocular::RobotEyeManager::SearchForEyes (unsigned int *timeout,* bool *blocking* = `true`, ocular::RobotEyeManagerCallback * *callbackPtr* = `NULL`)

Method to search for eyes present on the network. This is a method for searching for any Robot Eye devices present on the local network. It has flexible behaviour based on the arguments provided. The default configuration is as a blocking call which will search for the specified timeout, and then return. The UDP search command will be transmitted twice per second for the period of the timeout. If blocking is set to false, then this call will return immediately, and the search will proceed asynchronously in a separate thread space for the specified timeout. If a pointer to a RobotEyeManagerCallback is provided as an argument, then this callback will be executed on completion of the specified timeout.

If the search is done with blocking set to false, it is possible to specify a timeout of 0. This will cause the asynchronous search to proceed indefinately. If a callaback is provided, this callback will be executed with each re-transmission of the search request, providing a regular, asynchronous update of of the eyes seen on the network since the last call to ResetEyeList.

Only one transaction can be in-progress for each RobotEyeManager class at any time. In-progress transactions can be stopped with the StopSearching method.

Parameters

timeout	- The overall timeout for the search function in milliseconds. For blocking = true, 0 is an invalid value. For blocking = false, 0 indicates a request for an ongoing asynchronous search.
blocking	- Whether to block on timeout of the search or not. Defaults to true.
callbackPtr	- Callback to execute on completion of search, or on re-transmission of search for ongoing asynchronous searches.

6.5.3.9 ocular_error_t ocular::RobotEyeManager::SetIP (std::string *serial,* std::string *desiredIP*)

Set the IP address of the target eye. This method enables the setting of the IP address of a target eye on the network. There is no requirement that the specified serial number has been seen on the network as the result of a SearchForEyes command.

Parameters

serial	- Serial number of the eye who's I.P. is to be changed.
desiredIP	- Desired IP address of the eye.

Note

This command does not check for IP address conflicts. Setting the IP of an eye to a currently occupied IP address will cause undefined behaviour.

6.5.3.10 void ocular::RobotEyeManager::StopSearching (void)

Method to stop an in-progress search. A call to a this method will stop any in-progress search for eyes. If called from a different thread, it can also be used to interrupt a blocking SearchForEyes call or a SetIP call. This call will cause all relevant behaviours to occur as-if the specified timeout for the stopped transaction had occured.

6.5.3.11 ocular_error_t ocular::RobotEyeManager::WaitForTransactionComplete (void)

Method to wait for completion of asynchronous transaction. This method will block until an asynchronous SearchForEyes transaction is complete. If no search is currently in progress it will return immediately with a NO_ERR code. If the search in progress is an ongoing asychronous search, it will return immediately with an ERR_BUSY code, as such a call will never timeout, and this function would therefore never return.

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

- include/roboteye/RobotEyeManager.h

6.6 ocular::RobotEyeManagerCallback Class Reference

`#include <RobotEyeManager.h>`

Public Member Functions

- virtual void ManagerCallbackFcn (RobotEyeMap foundEyes)=0

6.6.1 Detailed Description

A class used to define a callback for eye searches. This class has a single pure virtual member fuction, and is used to derive callback classes. The callback is executed by the RobotEyeManager::SearchForEyes method, for more details see the documentation for that method.

6.6.2 Member Function Documentation

6.6.2.1 virtual void ocular::RobotEyeManagerCallback::ManagerCallbackFcn (RobotEyeMap *foundEyes*) [pure virtual]

RobotEyeManager Callback Function. This pure virtual function should be implemented in a class derived from the RobotEyeManagerCallback class to enable the callback functionality of the RobotEyeManager::SearchForEyes method.

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

- include/roboteye/RobotEyeManager.h

6.7 ocular::RobotEyeNotificationCallbackClass Class Reference

Callback class for notification callbacks from asynchronous commands.

`#include <RobotEyeCallbacks.h>`

Public Member Functions

- virtual void NotificationCallback (ocular_error_t ErrCode)=0

6.7.1 Detailed Description

Callback class for notification callbacks from asynchronous commands.

6.7.2 Member Function Documentation

6.7.2.1 virtual void ocular::RobotEyeNotificationCallbackClass::NotificationCallback (ocular_error_t *ErrCode*) [pure virtual]

Pure Virtual callback function used for generating callback notifications on some motion commands

Parameters

ErrCode	- The error code appropriate to the asynchronous command made. ErrCode of NO_ERR indicates success.

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

- include/roboteye/RobotEyeCallbacks.h