



ROBOT EYE PYTHON LIBRARY

Reference Manual

VERSION 3.5.623

FRI 21ST MAY, 2021

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

www.ocularrobotics.com

Contents

1	RobotEye Python 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	8
4	Namespace Index	11
4.1	Packages	11
5	Hierarchical Index	13
5.1	Class Hierarchy	13
6	Class Index	15
6.1	Class List	15
7	Namespace Documentation	17
7.1	RobotEye Namespace Reference	17
7.1.1	Detailed Description	17
7.2	RobotEyeCommon Namespace Reference	17
7.2.1	Detailed Description	17
8	Class Documentation	19
8.1	RobotEye.LensController.Lenscontroller Class Reference	19
8.1.1	Detailed Description	20
8.1.2	Constructor & Destructor Documentation	20
8.1.2.1	__init__	20
8.1.3	Member Function Documentation	20
8.1.3.1	ChangeIP	20
8.1.3.2	GetControllerVersion	20
8.1.3.3	GetLens	21
8.1.3.4	MoveAperture	21
8.1.3.5	MoveFocus	21
8.1.3.6	Reboot	21
8.1.3.7	Restart	22
8.1.3.8	SetAperture	22

8.1.3.9	SetFocus	22
8.2	RobotEye.RobotEyeCommon.OcularErrorCode Class Reference	23
8.2.1	Detailed Description	23
8.3	RobotEye.RobotEye.RobotEye Class Reference	24
8.3.1	Detailed Description	25
8.3.2	Constructor & Destructor Documentation	25
8.3.2.1	__init__	25
8.3.3	Member Function Documentation	25
8.3.3.1	GetApertureAngles	25
8.3.3.2	GetSerial	26
8.3.3.3	GetStabilisedRoll	26
8.3.3.4	GetStabilisedTarget	26
8.3.3.5	GetStabilisedTargetDistance	26
8.3.3.6	Home	27
8.3.3.7	SetAcceleration	27
8.3.3.8	SetApertureAngles	27
8.3.3.9	SetMotors	28
8.3.3.10	SetStabilisedSpeed	28
8.3.3.11	SetStabilisedTarget	28
8.3.3.12	StartStabilisation	29
8.3.3.13	Stop	29
8.3.3.14	StopStabilisation	29
8.3.3.15	TrackApertureAngles	29

Chapter 1

RobotEye Python 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 Python 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 'azimuth-Angle' 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.

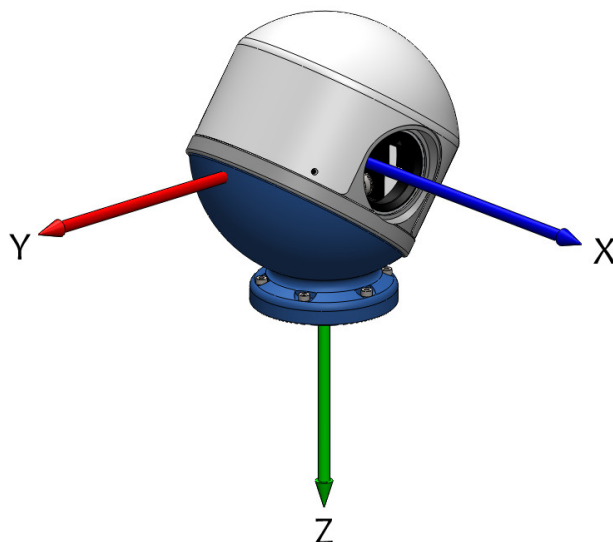


Figure 2.1: The RobotEye Coordinate System

The diagram above shows the right-handed frame used for the [RobotEye](#).

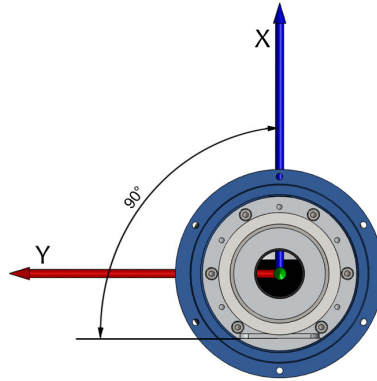


Figure 2.2: The RobotEye Azimuth Reference Surface

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.

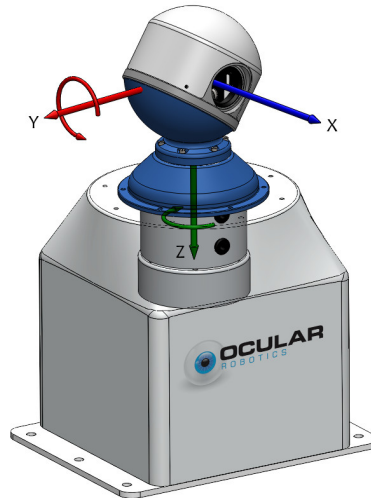


Figure 2.3: The RobotEye Elevation Reference Surface and Azimuth and Elevation Rotation Sign

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 179 degrees, and it is commanded to 1 degree, it will only move 2 degrees in azimuth.
- Azimuth angles are defined as -180 -> 180 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 I-PV4 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.

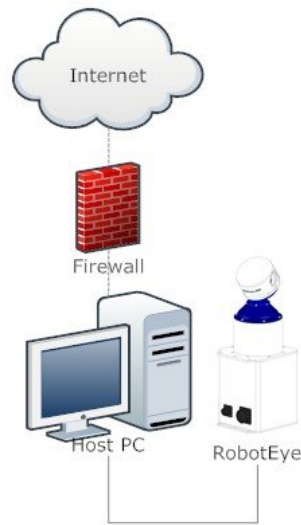


Figure 3.1: Network Schematic for Connecting a Host PC to a Single RobotEye

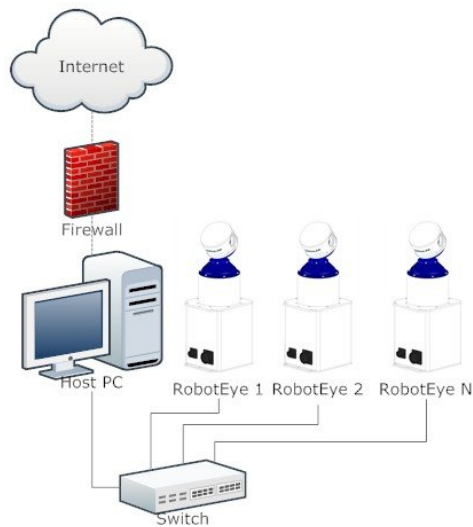


Figure 3.2: Network Schematic for Connecting a Host PC to Multiple RobotEyes

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

Namespace Index

4.1 Packages

Here are the packages with brief descriptions (if available):

RobotEye	
Module with RobotEye Driver Class	17
RobotEyeCommon	
Module with supporting class/methods for parsing the Robot- Eye API responses	17

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

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

RobotEye.LensController.Lenscontroller	19
RobotEye.RobotEye.RobotEye	24
Enum	
RobotEye.RobotEyeCommon.OcularErrorCode	23

Chapter 6

Class Index

6.1 Class List

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

RobotEye.LensController.Lenscontroller	
The Lens Controller class	19
RobotEye.RobotEyeCommon.OcularErrorCode	
The Ocular Error Code enumerated type	23
RobotEye.RobotEye.RobotEye	
The RobotEye class	24

Chapter 7

Namespace Documentation

7.1 RobotEye Namespace Reference

module with [RobotEye](#) Driver Class

7.1.1 Detailed Description

module with [RobotEye](#) Driver Class

7.2 RobotEyeCommon Namespace Reference

module with supporting class/methods for parsing the [RobotEye](#) API responses

7.2.1 Detailed Description

module with supporting class/methods for parsing the [RobotEye](#) API responses

Chapter 8

Class Documentation

8.1 RobotEye.LensController.Lenscontroller Class Reference

The Lens Controller class.

Public Member Functions

- def [__init__](#)
The default [RobotEye](#) constructor.
- def [SetAperture](#)
SetAperture for lens controller Set desired value for aperture (in f-number)
- def [SetFocus](#)
Sends setFocus command to lens Controller.
- def [ChangeIP](#)
Changes IP of lens Controller.
- def [Reboot](#)
Soft restart LC-2.
- def [Restart](#)
Hard restart LC-2.
- def [GetControllerVersion](#)
Get the lens controller firmware version.
- def [MoveAperture](#)
Move Aperture motor on X step (one-quarter-stop f-number)
- def [MoveFocus](#)
Move Focus motor on X units where X could be as positive as negative values.
- def [GetLens](#)
Get lens name.

8.1.1 Detailed Description

The Lens Controller class.

This class contains all of the public programmatic interfaces to the Lens Controller system. The class can be instantiated multiple times to connect to lens Controller systems on different IP addresses. The response from each API is in JSON format which can be easily loaded using `json.loads`

8.1.2 Constructor & Destructor Documentation

8.1.2.1 `def RobotEye.LensController.Lenscontroller.__init__(self, lensControllerIP)`

The default [RobotEye](#) constructor.

Parameters

<i>lenscontrollerIP</i>	- the IP address of the Lens Controller expressed as a string i.e. "10.1.1.200"
-------------------------	---

Exceptions

<i>Connection-RefusedError</i>	- likely causes; incorrect network setup, configuration errors.
--------------------------------	---

8.1.3 Member Function Documentation

8.1.3.1 `def RobotEye.LensController.Lenscontroller.ChangeIP (self, newIp)`

Changes IP of lens Controller.

Parameters

<i>newIp</i>	
--------------	--

Returns

IPchanged, lens controller will set IP and restart It will be required to construct a new LensController object to communicate with the LC on the new address.

8.1.3.2 `def RobotEye.LensController.Lenscontroller.GetControllerVersion (self)`

Get the lens controller firmware version.

Returns

Returns the Lens Controller firmware version

8.1.3.3 def RobotEye.LensController.Lenscontroller.GetLens (self)

Get lens name.

Returns

JSON response with NO_ERR Error code and Lens:XXX, where XXX is the lens name stored in internal lens memory, lens should support this command.

8.1.3.4 def RobotEye.LensController.Lenscontroller.MoveAperture (self, xunits)

Move Aperture motor on X step (one-quarter-stop f-number)

Parameters

<i>xunits</i>	
---------------	--

Returns

“Iris=Y”, where Y – current value of Aperture or “errorAperLimits” when aperture value is not reachable

8.1.3.5 def RobotEye.LensController.Lenscontroller.MoveFocus (self, xunits)

Move Focus motor on X units where X could be as positive as negative values.

Parameters

<i>xunits</i>	
---------------	--

Returns

JSON object with errorCode NO_ERR and Focus:Y, where Y – current value of Focus motor or “errorFocus” when focus value is not reachable

8.1.3.6 def RobotEye.LensController.Lenscontroller.Reboot (self)

Soft restart LC-2.

Returns

OK

8.1.3.7 `def RobotEye.LensController.Lenscontroller.Restart (self)`

Hard restart LC-2.

Returns

OK

8.1.3.8 `def RobotEye.LensController.Lenscontroller.SetAperture (self, aperture)`

SetAperture for lens controller Set desired value for aperture (in f-number)

Parameters

<i>aperture</i>	Supported Values : [1.0, 1.1, 1.3, 1.4, 1.6, 1.8, 2.0, 2.2, 2.5, 2.8, 3.2, 3.5, 4.0, 4.5, 5.0, 5.6, 6.3, 7.1, 8.0, 9.0, 10, 11, 13, 14, 16, 18, 20, 22, 25, 29, 32, 36, 40, 45, 51, 57, 64, 72 ,80, 90]
-----------------	---

Returns

Response from the lens Controller in JSON format string

8.1.3.9 `def RobotEye.LensController.Lenscontroller.SetFocus (self, focus)`

Sends setFocus command to lens Controller.

Parameters

<i>focus</i>	
--------------	--

Returns

Response from the lens Controller returned as is

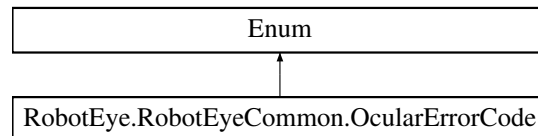
The documentation for this class was generated from the following file:

- RobotEye/LensController.py

8.2 RobotEye.RobotEyeCommon.OcularErrorCode Class Reference

The Ocular Error Code enumerated type.

Inheritance diagram for RobotEye.RobotEyeCommon.OcularErrorCode:



Static Public Attributes

- int **NO_ERR** = 0
- int **ERR_INVALID_N_ARGS** = 1
- int **ERR_ARG_OUT_OF_RANGE** = 2
- int **ERR_NOT_READY** = 3
- int **ERR_NOT_HOMED** = 4
- int **ERR_INVALID_ARG** = 5
- int **ERR_UNKNOWN_CMD** = 6
- int **ERR_UNSUPPORTED_CMD** = 7
- int **ERR_SCAN_TOO_SPARSE** = 8
- int **ERR_BUSY_CLIENT** = 9
- int **ERR_BAD_FLASH_PAGE** = 10
- int **ERR_BAD_FLASH_KEY** = 11
- int **ERR_STABILISATION_RUNNING** = 12
- int **ERR_VERSION_MISMATCH** = 13
- int **ERR_INVALID_CALLBACK** = 14
- int **ERR_BUSY** = 15
- int **ERR_TIMEOUT** = 16
- int **ERR_OTHER** = 17

8.2.1 Detailed Description

The Ocular Error Code enumerated type.

This enum is used to handle the various error codes returned by either the robot eye device or EyeLib internal operations. For more details on error codes from 0x00 to 0x08, please see the Robot Eye UDP Communications Specification.

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

- RobotEye/RobotEyeCommon.py

8.3 RobotEye.RobotEye.RobotEye Class Reference

The [RobotEye](#) class.

Public Member Functions

- def [__init__](#)
The default [RobotEye](#) constructor.
- def [Home](#)
Perform homing routine.
- def [Stop](#)
Stop any in-progress motion.
- def [SetApertureAngles](#)
Blocking synchronised motion command.
- def [TrackApertureAngles](#)
Rapid-response motion command.
- def [SetAcceleration](#)
Modify acceleration limits.
- def [GetApertureAngles](#)
A method to capture the current position of the [RobotEye](#).
- def [GetSerial](#)
A method to acquire the serial number of the target [RobotEye](#).
- def [StartStabilisation](#)
Start the stabilisation.
- def [StopStabilisation](#)
Stop the stabilisation.
- def [SetStabilisedTarget](#)
Set the stabilisation target.
- def [GetStabilisedTarget](#)
Get the stabilisation target.
- def [GetStabilisedTargetDistance](#)
Get the stabilisation target distance.
- def [GetStabilisedRoll](#)
Get the stabilisation roll.
- def [SetStabilisedSpeed](#)
Set the stabilisation speed.
- def [GetLastBlockingError](#)
A method to retrieve the last error code returned by the [RobotEye](#) to a blocking transaction.
- def [SetMotors](#)
Set the motors on or off.

8.3.1 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. The response from each API is in JSON format which can be easily loaded using `json.loads`

8.3.2 Constructor & Destructor Documentation

8.3.2.1 `def RobotEye.RobotEye.RobotEye.__init__(self, eyeip)`

The default [RobotEye](#) constructor.

Parameters

<i>eyeip</i>	- the IP address of the RobotEye expressed as a string i.e. "10.1.1.200"
--------------	--

Exceptions

<i>Connection-RefusedError</i>	- likely causes; incorrect network setup, configuration errors.
--------------------------------	---

8.3.3 Member Function Documentation

8.3.3.1 `def RobotEye.RobotEye.RobotEye.GetApertureAngles (self)`

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 in a JSON format

Returns

JSON format string with Az and El and The error code

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

8.3.3.2 `def RobotEye.RobotEye.RobotEye.GetSerial (self)`

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](#) in JSON response

Returns

JSON response with serial

8.3.3.3 `def RobotEye.RobotEye.RobotEye.GetStabilisedRoll (self)`

Get the stabilisation roll.

This command will get the roll needed to apply to the camera frame.

Returns

roll in JSON string - in degrees.

8.3.3.4 `def RobotEye.RobotEye.RobotEye.GetStabilisedTarget (self)`

Get the stabilisation target.

This command will get the stabilisation target.

Returns

JSON string with yaw pitch and roll if `NO_ERROR` or just error code

8.3.3.5 `def RobotEye.RobotEye.RobotEye.GetStabilisedTargetDistance (self)`

Get the stabilisation target distance.

This command will get the distance error of the stabilisation target.

Returns

distance in JSON. At target, distance will be 0, the farther away it is, the closer to 1.

8.3.3.6 `def RobotEye.RobotEye.RobotEye.Home (self)`

Perform homing routine.

This command will cause a [RobotEye](#) to execute it's homing routine. The over-loaded 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 SetApertureAngles or Home command. When this is done, the callback for the previously executing non-blocking command will return with ERR_TIMEOUT.

8.3.3.7 `def RobotEye.RobotEye.RobotEye.SetAcceleration (self, 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

<i>Acceleration</i>	- the new acceleration in rev/s/s
---------------------	-----------------------------------

8.3.3.8 `def RobotEye.RobotEye.RobotEye.SetApertureAngles (self, Az, El, 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

<i>Az</i>	- Azimuth angle in degrees (-180 to 180)
<i>El</i>	- Elevatiuon angle in degrees (-max to +max) where max depends on model (usually max = 35 degrees). 0 degrees is the horizontal plane.

<i>Speed</i>	- 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.

8.3.3.9 def RobotEye.RobotEye.RobotEye.SetMotors (self, on)

Set the motors on or off.

This command will power on or off both motors. At power-on, the motors default to on. This setting is not preserved through power-cycles of the device.

Parameters

<i>on</i>	- Desired state of the motors. Valid values are true or false. True turns motors on, and false turns them off.
-----------	--

8.3.3.10 def RobotEye.RobotEye.RobotEye.SetStabilisedSpeed (self, speed)

Set the stabilisation speed.

This command will set the speed used to apply the stabilisation movements.

Returns

speed with error Code in JSON format - in degrees per second.

8.3.3.11 def RobotEye.RobotEye.RobotEye.SetStabilisedTarget (self, yaw, pitch, roll)

Set the stabilisation target.

This command will set the stabilisation target.

Parameters

<i>yaw</i>	- in degrees.
<i>pitch</i>	- in degrees.
<i>roll</i>	- in degrees.

8.3.3.12 `def RobotEye.RobotEye.RobotEye.StartStabilisation (self)`

Start the stabilisation.

This command will activate the stabilisation module.

8.3.3.13 `def RobotEye.RobotEye.RobotEye.Stop (self)`

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 ERR_TIMEOUT

8.3.3.14 `def RobotEye.RobotEye.RobotEye.StopStabilisation (self)`

Stop the stabilisation.

This command will deactivate the stabilisation module.

8.3.3.15 `def RobotEye.RobotEye.RobotEye.TrackApertureAngles (self, Az, El, 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

<i>Az</i>	- in degrees (-180 to 180)
<i>El</i>	- in degrees (-max to +max) where max depends on model (usually max = 35 degrees). 0 degrees is the horizontal plane.
<i>Speed</i>	- 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:

- RobotEye/RobotEye.py