Tobii Gaze SDK

Developer's Guide

General Concepts

This is the developer's guide for the Tobii Gaze Software Development Kit (SDK). It provides conceptual information needed for developing gaze enabled applications.

Table of Contents

Table of Figures

The developer's guide consists of two parts: One generic part (this document) which describes the Tobii Gaze SDK on a higher level, and one API specific part which describes the Tobii Gaze SDK on more detailed level. The API specific documents are included with the respective Tobii Gaze SDK packages.

Introduction

The Tobii Gaze Software Development Kit (SDK) enables software developers to build gaze enabled applications.

NOTE. Tobii recommends the EyeX SDK, rather than the Gaze SDK, for all uses except in the following cases:

- *in embedded or single-application environments where your application has full control of the system;*
- *on platforms where the EyeX Engine is not yet available.*

For more information about the EyeX SDK, please see the Tobii Developer Zone.

The Gaze SDK is designed for integration into professional quality software. The key design principles have been minimalism and transparency, rather than ease of use and hiding of complexity. It features a C API that makes it accessible from virtually all modern programming languages, as well as higher level languages such as the .NET languages (e.g. C#).

The Gaze SDK contains libraries, header files for accessing the libraries (for the C API), sample applications, and documentation. It also includes some tools for configuring the eye tracker.

This guide assumes that the reader is familiar with either C/C++ or C#/.NET, and with software development in general.

Supported platforms and eye trackers

The Tobii Gaze SDK supports applications running on Microsoft Windows 7 and 8, on the x86 (Win32) and x64 platforms. The Gaze SDK enables the development of regular desktop applications but not Windows Store apps on Windows 8. The Tobii Gaze SDK is also available for Android and Linux platforms.

The Gaze SDK has been developed and tested for use with the Tobii REX eye tracker and the Tobii EyeX Controller, although it will also work with most other Tobii eye trackers.

Note that if your application uses the eye tracker analytically rather than for human-computer interaction, you should use the Tobii Analytics SDK instead.

Getting started

It is highly recommended to make sure that you have a working eye tracker setup prior to doing anything else. Please see the documentation for your eye tracker at the Tobii Developer Zone for instructions.

The recommended way to commence writing gaze enabled applications using the Tobii Gaze SDK, once the eye tracker setup is complete, is to read through this developer's guide, then to try the code samples and browse through the source code to get an idea of how everything fits together. It is, of course also possible to start with the sample code and work from there.

Tobii Gaze API reference

The TobiiGazeCore library is the core component of the Gaze SDK. It provides API functions for:

- Discovering connected eye trackers.
- Initializing and connecting to an eye tracker.
- Subscribing to gaze data.
- Getting information about the eye tracker device and the track box.
- Calibrating the eye tracker as well as getting/setting the calibration data.
- Setting and getting the active display area.

The library also includes some utility functions such as logging and version queries.

Eye tracker URLs

Eye trackers are identified using Unified Resource Locators (URLs). The Gaze SDK supports both USB-HID eye trackers and TCP/IP eye trackers. The scheme part of the URL indicates which protocol to use for connecting to the eye tracker.

The URL syntax for an eye tracker connected to using the USB-HID protocol is:

```
tobii-ttp://<ID>
```

where the ID is a unique identifier for the eye tracker device. The angle brackets are only for clarity in this document and are not part of the actual URL.

The URL for a TCP/IP eye tracker uses the tet-tcp scheme instead of tobii-ttp, and the host part (i.e. the part that follows the double slashes) can be either an IP address or a hostname that the operating system can resolve into an IP address using for example a DNS lookup.

Eye tracker event loop

All communication between the application and the eye tracker is performed asynchronously. The synchronous API functions hide the asynchronous behavior, but it is still there behind the scenes.

The TobiiGazeCore library uses a dedicated thread for processing messages received from the eye tracker. This thread is referred to as the event loop.

As an application developer you have two alternatives how to create and run the event loop:

- The first alternative is to create the event loop thread and then call the *run event loop* function from that thread. This alternative gives you full control over your threads and thread creation.
- The second alternative is to call the *run event loop on internal thread* function. This function creates an event loop thread and runs the event loop on it. This alternative is convenient if you do not want to create the thread yourself.

All callbacks from the asynchronous API function calls are called on the event loop thread.

Synchronous and asynchronous functions

Several API functions exist both as synchronous (blocking) and asynchronous (non-blocking) variants. The synchronous functions block until a result has been received from the eye tracker or a timeout has occurred.

The asynchronous versions of the functions all expect a callback function pointer (or equivalent) as an argument. The callback is used to indicate when an operation has finished and get the result of the operation. Generally, consider the following when writing the callback function:

- Do not call synchronous API functions from callback functions, since doing so causes the event loop thread to hang. On the other hand, calling asynchronous API functions from callback functions works just fine. In fact, asynchronous operations are commonly chained together and run in a sequence, with the callback from one operation invoking the next operation.
- Do not perform any lengthy operations in a callback function, as that would block the event loop thread from receiving and processing more messages. If your data processing takes considerable time, you should copy the data and post the operation to another thread.

Coordinate systems

The API makes use of several coordinate systems. This section describes the coordinate systems in some detail: how they are defined and what they are used for.

User Coordinate System (UCS)

All the data available from Tobii eye trackers that describe **3D space coordinates** are provided in the *User Coordinate System,* or *UCS* for short. The UCS is a millimeter-based system with its origin at the centre of the frontal surface of the eye tracker (see **Error! Reference source not found.**). The oordinate axes are oriented as follows: the x-axis points horizontally towards the user's right, the y-axis points vertically towards the user's up and the z-axis points towards the user, perpendicular to the front surface of the eye tracker.

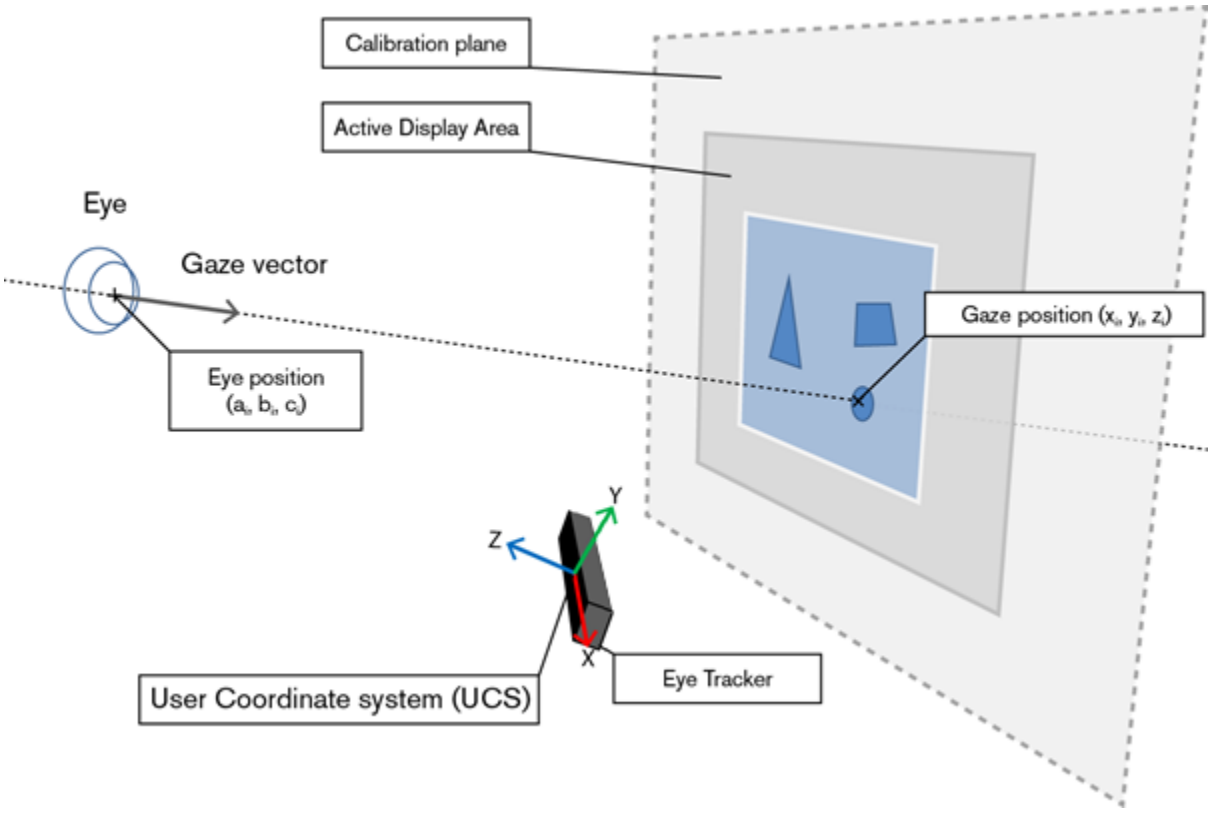


Figure 1. User Coordinate System (UCS)

Active Display Coordinate System (ADCS)

The gaze data is mapped onto a **2D coordinate system** aligned with the *Active Display Area*. The Active Display Area is the display area excluding the monitor frame. The origin of the *Active Display Coordinate System* (ADCS) is at the upper left corner of the Active Display Area. The point (0, 0) denotes the upper left corner and (1, 1) the lower right corner of the Active Display Area.

Note that it is possible to get ADCS coordinates outside the Active Display Area, e.g. (-0.014, 1.057).

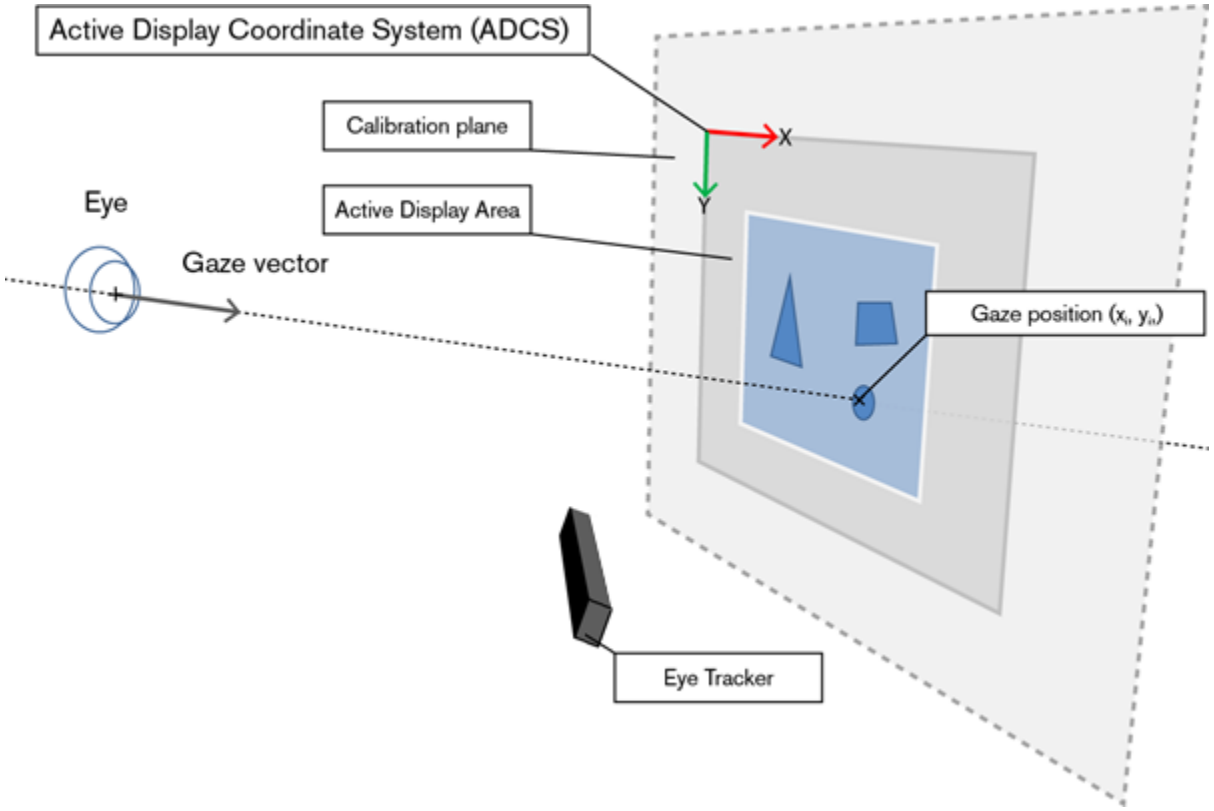


Figure 2. Active Display Coordinate System (ADCS)

Track Box Coordinate System (TBCS)

The track box is the volume in which the eye tracker is theoretically able to track the user's eyes. The user may move her head freely and still remain trackable as long as the eyes remain within the track box. The API provides functions to find out the size and position of the track box volume, and also where the eyes are located within it.

The coordinate system used to describe the position of the eyes within the track box is called the *Track Box Coordinate System (TBCS)*. The TBCS has its origin in the top, right corner, which is located closest to the eye tracker. The TBCS is a normalized coordinate system: the location of the (1, 1, 1) point is the lower, left corner furthest away from the eye tracker.

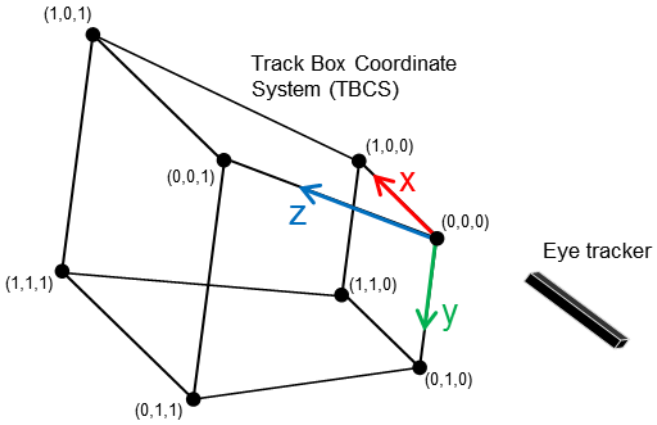


Figure 3. Track Box Coordinate System.

Gaze data

Once tracking has been started, the application will start receiving gaze data callbacks. Each gaze data packet contains the following data:

Timestamp. This is a value that indicates the time when the information used to produce the gaze data packet was sampled by the eye tracker. The value should be interpreted as a time interval in microseconds measured from an arbitrary point in time. The source of this timestamp is the internal clock in the eye tracker hardware.

Eye position from eye tracker. The eye position is provided for the left and right eye individually and describes the position of the eyeball in 3D space. The position is described in the UCS coordinate system as described above.

Eye position in track box. The eye position in the track box is provided for the left and right eye individually and gives the position of the eyeball within the track box volume. This data can be used to visualize the position of the eyes, which is commonly used as a tool to help the user positions themselves in front of the tracker.

Gaze point from eye tracker. The gaze point is provided for the left and right eye individually and describes the point on the calibration plane where the user's gaze is. Or, in more technical terms, the position of the intersection between the calibration plane and the line originating from the eye position point with the same direction as the gaze vector. The coordinates for the eye position and the gaze position are given in the User Coordinate System coordinate system. This is illustrated in Figure 4.

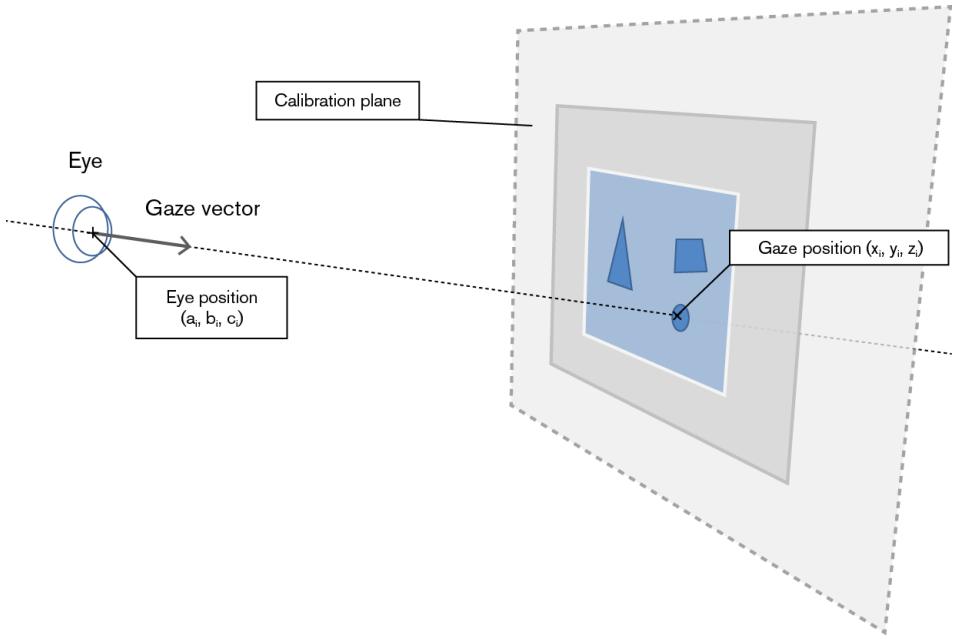


Figure 4. Eye position and gaze point.

Gaze point on display. The gaze point is provided for the left and right eye individually. It is conceptually the same as the gaze point from the eye tracker, but expressed as a two-dimensional point on the calibration plane instead of as a point in 3D space. The position is given in the Active Display Coordinate System where (0, 0) denotes the upper left corner and (1, 1) the lower right corner of the Active Display Area.

Tracking status code. The tracking status code specifies how certain the eye tracker is about which eyes have been tracked. It is straight forward when none or both eyes are tracked. When only one eye is found, the eye tracker uses heuristic methods to determine which eye it is based on, e.g., where the eyes were last tracked and where in the track box the eye is positioned.

The possible values for the tracking status code are:

- NO_EYES_TRACKED
- BOTH_EYES_TRACKED
- ONLY_LEFT_EYE_TRACKED – Only one eye was found. The eye tracker is confident that it is the left eye.
- ONE_EYE_TRACKED_PROBABLY_LEFT – Only one eye was found. It is probably the left eye.
- ONE_EYE_TRACKED_UNKNOWN_WHICH – Only one eye was found. It could be either the left or the right eye.
- ONE_EYE_TRACKED_PROBABLY_RIGHT - Only one eye was found. It is probably the right eye.
- ONLY_RIGHT_EYE_TRACKED – Only one eye was found. The eye tracker is confident that it is the right eye.

The recommended approach for gaze interaction applications is to only use gaze data packets where the eye tracker is fairly confident about which eye(s) are being tracked. This means that gaze data for the left eye can be used for the following tracking status codes:

- BOTH_EYES_TRACKED
- ONLY_LEFT_EYE_TRACKED
- ONE_EYE_TRACKED_PROBABLY_LEFT

And gaze data for the right eye can be used for the following tracking status codes:

- BOTH_EYES_TRACKED
- ONLY_RIGHT_EYE_TRACKED
- ONE_EYE_TRACKED_PROBABLY_RIGHT

Configuring the Active Display Area

An eye tracker needs to know where the screen is relative to itself in order to produce meaningful gaze point data. Eye trackers that are not built into display monitors (e.g. the Tobii X-series eye trackers) must be configured for use with a particular display monitor, projector, or other planar surface.

Note. If your application is intended to run on a computer where the Tobii EyeX Engine is installed, then you should let the EyeX Engine take care of the active display area configuration.

The configuration consists of three points in space corresponding to the top left, bottom left, and top right corners of the Active Display Area. If a monitor is used, the coordinates used are the locations of the corner pixels of the display area. The three points are to be given in the User Coordinate System (UCS) illustrated by the colored vectors in Figure 5. Note that this coordinate system is relative to the eye tracker, so that tilting or moving the eye tracker invalidates the Active Display Area configuration.

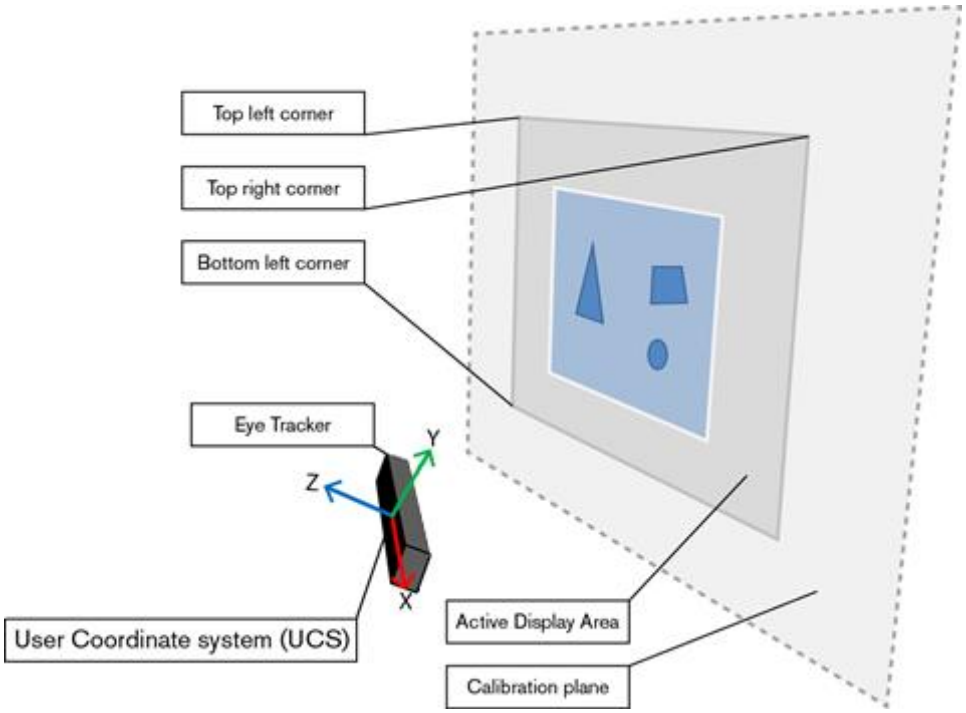


Figure 5. Eye tracker configuration schematic.

Note: It is important that the provided points correspond to the two sides of a rectangle. If the correct coordinates are not provided to the eye tracker, this will result in offsets in the gaze data.

It is also possible to query the eye tracker for its current Active Display Area configuration. This can be done for all eye trackers – not only X-series eye trackers.

Note: The eye tracker writes the Active Display Area configuration to persistent storage. It will be in use until a new Active Display Area configuration is written, regardless of whether the eye tracker loses power.

User calibration

In order to compute the gaze point and gaze direction with high accuracy, the eye tracker firmware adapts its algorithms to the person sitting in front of the tracker. This is referred to as calibrating the eye tracker for the user. It is typically performed by letting the user gaze at points located at some reference points on the active display area. Calibration should be performed for individual users and for different reading or visual aids (glasses or contacts). It is also recommended to calibrate for individual eye tracker devices and eye tracker/active display area setups.

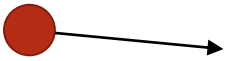
Note. If your application is intended to run on a computer where the Tobii EyeX Engine is installed, then you should let the EyeX Engine take care of the user profile handling and the calibrations.

The calibration is reset when the eye tracker loses power, and when a USB connected eye tracker loses its connection to the host computer. The API provides functions for saving and loading calibrations.

Calibration procedure

The calibration of the eye tracker is typically done as follows:

1. A small animated object is displayed on the screen to catch the user's attention.



2. When it arrives at the calibration point, let the point rest for about 0.5 seconds to give the user a chance to focus. Shrink the object to focus the gaze.

3. When shrunk, tell the eye tracker to start collecting data for the specific calibration point.
4. Wait for the eye tracker to finish calibration data collection on the current position.
5. Enlarge the object again.
6. Repeat steps 1-5 for all desired calibration points.

The animation in step 1 should not be too fast, nor should the shrinking in step 2 be too fast. Otherwise the user may not be able to get a good calibration result due to the fact that he or she has no time to focus the gaze on the target before the eye tracker starts collecting calibration data.

The normal number of calibration points is 2, 5, or 9; more points can be used but the calibration quality will not increase significantly for more than 9 points. Usually 5 points yields a good result and is not experienced as too intrusive by the user.

Configuring the calibration points

The location of the calibration points is decided by the client application. A typical calibration pattern for 5 points can be seen in Figure 6.

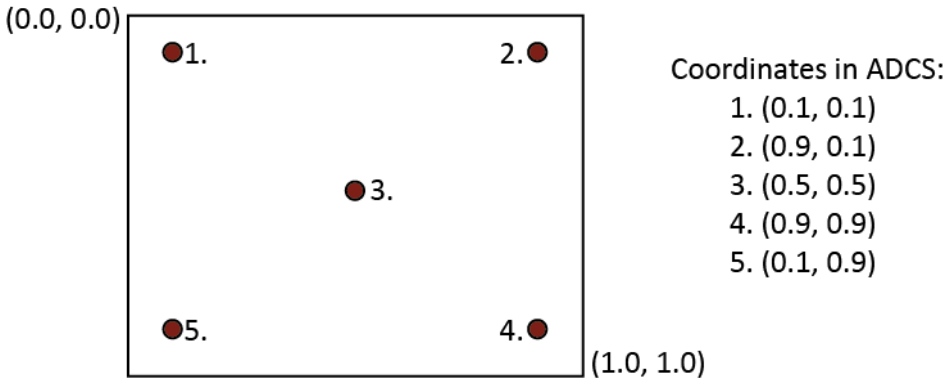


Figure 6. Standard 5-point Calibration Pattern.

NOTE: All points are given in normalized coordinates in such a way that (0.0, 0.0) corresponds to the upper left corner and (1.0, 1.0) corresponds to the lower right corner of the Active Display Area. When choosing the calibration points it is important to consider the following:

- The calibration points should span an area that is as large as or larger than the area where the gaze controlled application or the stimuli is to be shown in order to ensure good interaction.
- The calibration points must be positioned within the area that is trackable by the eye tracker and be within the Active Display Area.

The calibration state

To be able to perform a calibration the client application must first enter the calibration state. The calibration state is an exclusive state which can only be held by one client at a time. It is entered by calling the function `calibration_start` and is left by calling `calibration_stop`.

Some operations can only be performed during the calibration state, e.g., `calibration_add_point`, `calibration_remove_point` and `calibration_compute_and_set`.

Other operations such as `set_calibration` or `get_calibration` work at any time. However, `set_calibration` can only be called by the client which is currently in the calibration state. Best practice is to call `calibration_start` and check whether it reports that another client is currently calibrating rather than caching the result of the calibration events.

Calibration buffers

The eye tracker firmware uses two buffers to keep track of the calibration data:

- The temporary calibration buffer. This buffer is used only during calibration. This is where data is added or removed.
- The active calibration buffer. This buffer contains the data for the calibration that is currently set.

The active calibration buffer is modified either by a call to `set_calibration`, which copies data from the client, or a successful call to `calibration_compute_and_set`, which computes calibration parameters based on the data in the temporary buffer and then copies the data into the active buffer.

Calibration plots

There are many possible ways of visualizing calibration results. The Tobii calibration plot visualization is a concise representation of a performed calibration and it usually looks something like what is shown in Figure 7.

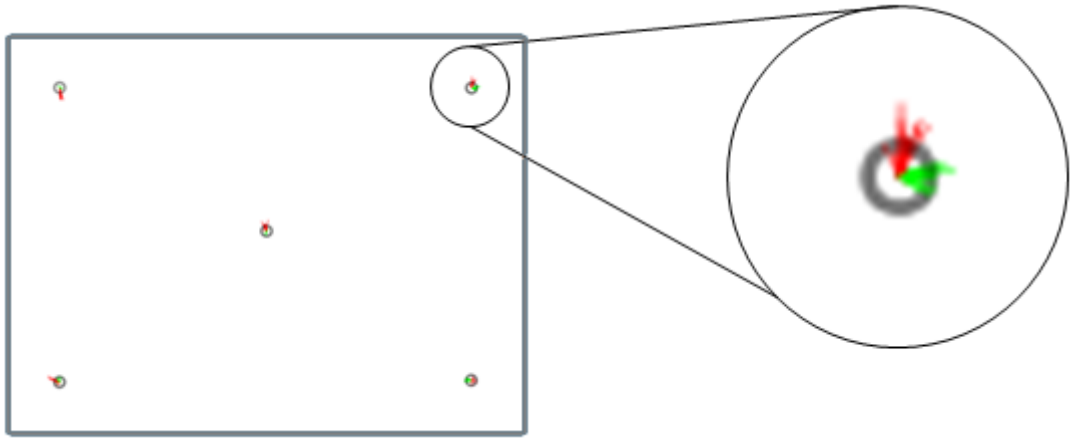


Figure 7. 5-point calibration plot with combined data for both the left and the right eye.

The calibration plot shows the offset between the mapped gaze samples and the calibration points based on the best possible adaptation of the eye model to the collected values done by the eye tracker during calibration. In Figure 7, the samples are drawn as red and green where the lines represent the offset between the sample points and the calibration points. The red lines represent samples from the left eye and the green lines represent samples from the right eye. The grey circles are the actual calibration points. The data displayed in the plot is made available to client applications through the language bindings. This allows for alternative visualizations of calibration results as well as the traditional visualization in Figure 7.