**ATUAV-RT and Endat Connector Documentation**

Oliver Schmid - [oliver.schmd@gmail.com](mailto:oliver.schmd@gmail.com)

July 2012

**Abstract**

This report explains how Tobii eye tracking data is processed by EMDAT in real-time.

**Introduction**

This is the documentation for ATUAV-RT and program used by the UBC ATUAV project to process eyetracking events in real time using the EMDAT library.

All files referred to are stored at: <https://github.com/ATUAV/ATUAV>

**Dependencies**

ATUAV-RT has 3 outside and 1 internal dependency:

- Tobii SDK 3.0 RC 1 <http://www.tobii.com/en/eye-tracking-research/global/landingpages/analysis-sdk-30rc/>

- FixDet <http://www.sis.uta.fi/~csolsp/projects.php>

- NDesk.Options <http://www.ndesk.org/Options>

- Emdat : Visual C++ project written specifically for ATUAV-RT to connect C# to Python.

**Main Classes**

These 2 classes setup the framework of the program, but do none of the actual data processing.

**Program.cs**

This is the main class. It handles command line argument parsing, Tobii eye-tracker connection, and initializing the fixation detector and EMDAT connector classes.

Important Methods

- Main : parses arguments and looks for Tobii eye trackers.

- EyetrackerFound: triggered when an eyetracker is found. Syncs CPU and eyetracker clocks. Initializes the fixation detector and has it subscribe to the eye tracker's gaze data events. Initializes the EMDAT processor and has it subscribe to eyetracker and fixation events. This method also has commented out sections showing how to use GazeDataHandlers like ConsolePrinter that perform an operation on every gaze data event and handlers like WindowingConsolePrinter that perform operations on chunks of data.

**EyetrackerConnector.cs**

Handles connecting to an eyetracker

Important Methods

- Eyetracker: gets the connected eyetracker object, use this object to subscribe to events (GazeDataReceived, FramerateChanged).

- Connect: connects to an eyetracker and starts tracking events. Handles exceptions by disconnecting from eye tracker (I have yet to encounter an exception).

- Disconnect: disconnects and disposes eyetracker.

**Gaze Data Handlers**

This collection of classes define different ways to handle gaze data events from the eyetracker. For instance the FixationDetector class uses them as input to determine the occurrence of fixations in real time. For a better understanding of the inheritance of different gaze data handlers see the included class diagram (ClassDiagram.cd) in the project.

**GazeDataHandler.cs**

Base class for classes that handle gaze data events from the eyetracker.

Important Methods

- constructor: requires a SyncManager object to be passed in so that timestamps can be converted from the eye tracker’s internal clock to local CPU time.

- GazeDataReceived: abstract method defining how to handle a gaze data event.

**GazeDataSynchronizedHandler.cs**

Base class for classes that handle gaze data events when CPU/eyetracker clocks are synchronized. This means that timestamps can be reliably converted between eyetracker and CPU time.

Important Methods

- GazeDataReceivedSynchronized: abstract method defining how to handle a synchronized gaze data event

**ConsolePrinter.cs**

Gaze data handler that prints every event it receives to console. Can also listen to FixationEnd events.

Important Methods

- Print: prints a gaze data item or fixation event to console.

**FixationDetector.cs**

Detects fixations using FixDet project written by Oleg Špakov at the University of Tampere in Tampere, Finland. Documentation and code available at <http://www.sis.uta.fi/~csolsp/projects.php>

Important Methods

- constructor: initializes FixDet to something similar to the default Tobii Fixation Filter. See documentation for available settings.

- FixDetector: gets the fixation detector, use for subscribing to FixationEnd, FixationStart, and FixationUpdate events (FixationEnd being the most important and useful).

- GazeDataReceivedSynchronized: converts microsecond timestamp to milliseconds and offsets it so that the first detected event occurs at time 0, converts normalized screen coordinates (for x-axis, 0 is left of screen, 1 is right; for y-axi: 0 is top, 1 is bottom) into pixels, averages left and right eye gaze point (where on the screen the user is looking) as per default setting of Tobii Studio. Currently this method also filters events with low validity scores, although the Tobii documentation doesn’t explicitly say how their fixation detector handles low validity events it advises users of the SDK to ignore anything with a validity code of 2 or higher (pg 34 of SDK 3.0 RC 1 documentation). Gaze point averaging behaviour when only one eye is valid is to not average and instead use only that eye’s gaze point. Again, I’m unsure whether this is what Tobii Studio’s fixation filter does.

**Windowing Gaze Data Handlers**

These gaze data handlers also implement the interface WindowingHandler which defines methods for starting, stopping and processing data collected in a window of time.

**WindowingHandler.cs**

Interface for windowing gaze data handlers.

Important Methods

- StartWindow: starts data collection, before this is called events should be dropped.

- StopWindow: stops data collection, after this is called events should be dropped. However, events should not be deleted after this is called.

- ProcessWindow: pauses data collection, processes accumulated data and starts a new window. Locking should be used to ensure that no event is dropped while this method is run and that events are only processed in the windows they occurred. For an example of correct implementation see WindowingConsolePrinter. Takes as input a bool (keepData) to be used to differentiate between separated or accumulating windows. Separated windows (keepData=false) should only process events that occurred during that window. Accumulating windows (keepData=true) should process events that occurred during that window or any window prior.

**WindowingConsolePrinter.cs**

Gaze data handler that collects gaze data in dynamically sized windows. When called to process a window it prints all the events it has collected to console.

Important Methods

- PrintData: prints all collected data to console, does not clear accumulated data.

**EmdatProcessor.cs**

Calls the Python EMDAT library to process gaze, fixation and Area of Interest data into features used for machine learning. Calling Python from C# is done through Emdat C++ Project.

Important Methods

- AoiFilePath: gets and sets the filepath for AOI definitions, reads that file so that the contents can be passed to EMDAT when ProcessWindow is called

- ProcessWindow: Should generate an ID for the window (could just use an incrementing counter “seg1”, “seg2”, “seg3”, …), converts GazeDataItems into string form (these should be in the same format as when Tobii Studio does a text export: tab separated columns, same number of columns, same order, new lines separating each event), converts fixations into string form (these should be in the same format as when Tobii Studio exports a FixationData.tsv).

**Utility Classes**

**EyetrackerErrors.cs**

Static class to convert error codes to human readable string explanations. Errors rarely happen. I have yet to encounter one.

**EyetrackerEvent.cs**

Wrapper class for when you want to store GazeDataItem and SFDFixation objects in the same data structure.

**Emdat C++ Project**

Originally ATUAV-RT was meant to be written in Python so that it could easily make use of EMDAT for feature generation. The Tobii SDK however doesn’t work in Windows XP (the RC 1 version at least, Tobii says this should be fixed for RC 2 which is due out end of summer 2012) so C# was chosen instead. To use Python code from C# I wanted to use IronPython but that depended on C# 4.0 which requires Visual Studio 2010. So I ended up writing my own connector in C++/CLI.

Note that since the names are all very similar: EMDAT is the python toolkit, Emdat is the C++/CLI project connecting C# and Python and emdat.py is a python module for ATUAV-RT that uses EMDAT.

The Emdat C++/CLI project (in the C++ subdirectory of the ATUAV\_RT) makes available to C# the python module **emdat.py**. This module is not part of the actual EMDAT library but instead handles all interaction with it. This minimizes the required amount of C++ code.

emdat.py has a single function that takes as input 4 strings (segment ID, gaze data points in Tobii export format, fixations in Tobii export format, AOI definitions in ATUAV format) and return a string of feature value pairs (each line is of the format feature=value).

There was some difficulty in first compiling the Emdat because the debug version of python is not included in the default install. To resolve pyconfig.h was hacked in the following way: <http://tech.groups.yahoo.com/group/OpenCV/message/73972>

**TODO**

- emdat.py fails to import EMDAT library

- Conversion of GazeDataItems and SFDFixations into Tobii export format in EmdatProcessor.cs needs to be verified.

- EmdatProcessor.cs needs to do something with the generated features once they return from EMDAT.

**Tobii Documentation Resources**

- Tobii Studio

<http://www.tobii.com/Global/Analysis/Downloads/User_Manuals_and_Guides/Tobii_Studio1.X_UserManual.pdf>

- Tobii SDK

<http://www.tobii.com/Global/Analysis/Downloads/User_Manuals_and_Guides/Tobii%20SDK%203.0%20Release%20Candidate%201%20Developers%20Guide.pdf>