ECL: a C++ library for dual-Purkinje Image eye trackers

Maarten Demeyer, Jeroen Clarysse, and Peter De Graef

Maarten Demeyer

University of Leuven (K.U. Leuven)

Laboratory of Experimental Psychology

Tiensestraat 102, bus 3711

B-3000 Leuven

Belgium

maarten.demeyer@psy.kuleuven.be

Abstract

Here we present the Eyetracker Code Library (ECL), a C++ class library designed to enable the use of analog dual-Purkinje Image (DPI) eye trackers on modern MS Windows platforms. The central function of ECL is to fetch eyetracker samples from an analog-to-digital conversion device, and process them into a screen coordinate and an eye status in real-time (e.g., ‘in saccade’). Automatic calibration and drift correction routines are present. In addition, ECL offers functionality to define relevant regions on the experimental screen, render a live control screen, and write the eye movement data to a file after each trial. Following a modular object-oriented design, all input and output functionality can be flexibly replaced by different devices, rendering libraries or file formats.

1. Introduction

The dual-Purkinje Image eye tracker (DPI; Crane & Steele, 1985) has enjoyed decades of popularity as a tool in eye movement research, and continues to occupy a valuable niche in the market. There is however no official software package available to use DPI eye trackers in conjunction with the modern computing devices that most researchers program and run their experiments on. In this report, we describe the Eyetracker Code Library (ECL) developed at our lab, improving upon our existing software for the DOS platform (Van Rensbergen & De Troy, 1993; Van Diepen, 1998). The goal here is to provide the reader with a clear overview of the general design of these libraries, rather than to provide a full documentation of all ECL functions.

ECL is aimed specifically at computers running on a Microsoft Windows platform, and uses the C++ programming language. Features include the acquisition and calibration of the analog DPI eye position signals, drift correction, on-line saccade detection, support for a control monitor, output to a custom file format, and a modular object-oriented design. ECL can therefore be adapted and extended flexibly to fit the researcher’s specific hardware configuration and scientific goals.

The current report though assumes the following standard hardware setup. To a Windows XP computer is connected a BNC-2111 connector block and a PCI-6221 analog-to-digital conversion card manufactured by National Instruments. This Data AcQuisition (DAQ) device reads in the four analog lines of the DPI eye tracker: horizontal position, vertical position, the blink signal, and the track signal. Both position signals are in the range of -5V to +5V, whereas the binary blink and track signals are either at 0V or +5V. Optionally, a photosensitive cell can be connected to the fifth DAQ channel to register local luminance signals at the experimental screen. This allow accurate post-hoc synchronization between screen commands and eye movement data. The experimental screen is controlled by an internal graphical card compatible with DirectX 8.1 or higher. The subject has two separate manual response buttons available, each connected to a pin of the PC’s parallel port. The experimenter uses a standard keyboard, and receives visual feedback from a control screen connected to a second DirectX-capable graphical device on the PC.

2. Architecture

2.1 General

ECL is built around five main classes. The central class is **ecl_process**, processing the raw input signals into pixel coordinates on the experimental screen and an eye status (e.g., 'in saccade'). ecl_process automates the use of the auxiliary class **ecl_daq**, which fetches the digitized DPI signals from the DAQ device. ecl_process also handles the use of the **ecl_output** auxiliary class, which writes the processed samples to a file on the hard disk. **ecl_monitor** objects contain functionality to display visual stimulation on either an experimental or a control screen, and **ecl_response** objects are used to collect manual responses from either the subject or the experimenter. ecl_process uses the latter two classes for its automated calibration routines, and to render a control screen. However, both classes can in addition be used to program the remainder of the experiment.

An important feature of the auxiliary input and output classes is their polymorphic nature. As base classes, they declare abstract functionality that is to be implemented through the definition of child classes. For instance, the ecl_response base class declares the is_pressed() function, but does not implement it. Its child classes ecl_response_keyboard and ecl_response_mouse however do provide implementations specific to these respective devices. Since all ECL functions define their arguments as a base class type, the definition of input and output devices becomes very flexible. To continue the example, the calibration routine needs to record an ecl_response button press, but will not discriminate whether the response object provided to it is of the keyboard or of the mouse subtype. Thus, ECL experiments can be flexibly ported between various hardware setups, often by changing only a single line of code. This applies equally to DAQ devices, response devices, monitors, and output file formats. Moreover, currently unsupported hardware devices, stimulus presentation libraries and file formats can easily be integrated into ECL through defining new child classes.

By default, ECL runs at a temporal resolution of 1000 Hz, basing itself on the DAQ device hardware clock for timing. If the DAQ device does not have such a clock, it can choose to employ the PC's CPU clock, accessed through the Windows API function QueryPerformanceCounter(). ECL also contains a placeholder LUT system for graphical operations. Though currently mapping directly onto standard Windows

RGB-values, this allows more advanced visual stimulation routines to be implemented in the future.

A final aim of ECL is to provide function calls that are intuitive even to the non-expert programmer.

2.2 The auxiliary input/output classes

2.2.1 ecl_daq

Raw, uncalibrated eye position data are collected using an ecl_daq object. By default this analog-to-digital conversion is handled in an unbuffered manner. That is, only the latest eye position information is used every time ECL collects a new sample, and time-outs of the sampling operation are treated as missing data. The emphasis of ECL is therefore on real-time eye movement measurements, to allow for accurate gaze-contingent display changes. However, on a modern computer no time-outs should occur. A buffered sampling call is also available, allowing the retrieval of data corresponding to a specific timestamp within the internal buffer of the DAQ device.

In practice, the user will most often only need to declare and initialize the ecl_daq object, before passing its address on to the central ecl_process object, which will then use its functionality automatically. Currently two DAQ devices are implemented into child classes. First, ecl_daq_ni utilizes the NI-DAQmx libraries to support the National Instruments setup described in the Introduction. Second, ecl_daq_vsg allows the use of the DAQ capabilities of the popular CRS ViSaGe stimulus generator. In addition, ecl_daq_mouse can be used for simulation purposes, whereas ecl_daq_recording allows the experimenter to replay a recorded DAQ stream as if it represented live data.

2.2.2 ecl_monitor

ecl_monitor defines functionality to perform basic visual operations onto a computer screen, including the drawing of text, simple shapes, single pixel values, and bitmap images. Complex drawing surfaces can be preloaded before a trial to avoid interference with the DPI data collection. In order to make graphical operations effective a **present()** call must be made, drawing the contents of video

memory onto the screen. These present() calls can be auto-throttled. Finally, each ecl_monitor object contains information on the monitor's size, distance, resolution, and refresh rate, to be used in various ECL calculations.

At least one monitor needs to be provided to the ecl_process object to perform the ECL calibration routines. An optional second monitor can be passed on to give the experimenter automated visual feedback during the experiment. If the required visual stimulation is relatively simple, ecl_monitor objects could also be used to perform all graphical operations in the experiment. If not, the same graphical device encapsulated within an ecl_monitor child class can still be addressed directly to perform more advanced presentation routines.

Currently implemented child classes utilize either Microsoft DirectX (ecl_monitor_dx) or the CRS ViSaGe stimulus generator (ecl_monitor_vsg).

2.2.3 ecl_response

The ecl_response class defines functionality to collect a manual response, both from continuous dimensions and binary button presses. ecl_process requires these response objects to perform the ECL calibration routine. Currently supported are a standard keyboard and mouse, response boxes connected to the PC's parallel port, response boxes connected to the ViSaGe's digital input port, and the axes and buttons of a DirectX-enabled joystick. Standard devices such as a keyboard have their buttons defined by default. In custom devices the experimental application will need to specify how to address each button.

2.2.4 ecl_output

The ecl_output class declares functionality to save the full eye position and eye status data to a file. These data are temporarily stored in a RAM memory buffer, and written to the hard disk when data collection ends for that trial. The buffering is implemented in the ecl_output base class, but the saving to a file functionality must be implemented in a child class. Typically, the user does not interact with output objects directly during data collection. The recommended procedure is to initialize an output object and register it with the central ecl_process object using the attach_output_file() call. ecl_process will then automatically add any new data

samples to the output buffer, including the status of binary key-press and stimulus onset flags.

Currently ECL supports two file formats: a raw dump of the output buffer to a text file (ecl_output_raw), and the p-file format used in previous versions of our DPI software (ecl_output_pfile). For the latter, custom analysis and viewing tools are available. Of course, the experimenter is free to define a new data format in a new child class, as long as all the information it requires is contained within the general ecl_output buffer.

2.3 The core class: ecl_process

2.3.1 Overview

The central task of ECL is to process the raw eye coordinates delivered by an ecl_daq object into meaningful pixel coordinates on the experimental screen. To this end, ecl_process will establish a correspondence between both coordinate systems through a calibration procedure. It will then apply this transformation in real-time as it samples new data from the DPI. Moreover, ecl_process will perform an on-line analysis on a five-sample buffer of recent eye position data to determine the current status of the eye. Both the eye position and the eye status can be accessed at any time when running an experiment. If during data registration an ecl_output object was attached to the ecl_process object, the data can be written to a file at the end of the trial.

2.3.2 Configuration

ecl_process is configured through supplying an **ecl_config** object at the time of initialization. The parameters of a configuration object can only be altered by loading a properly formatted plain text file. For instance, this allows the experimenter to fine-tune the saccade detection algorithm to his or her specific scientific requirements. The ecl_config object supplied to ecl_process will also contain the initial calibration setup. Since ecl_process can export its current configuration to an ecl_config file, this allows the experimenter to carry calibration results across multiple executions of the experimental application.

2.3.3 Calibration

Before processing of eye position data can begin, a calibration routine must be run, or valid calibration results must be present in the ecl_config object supplied to ecl_process. It will utilize the subject and control monitors specified during the initialization of ecl_process.

After calling the **calibrate()** function, the subject is first presented with a central fixation dot, allowing the experimenter to set up the mechanical parts of the DPI device. Feedback on the current state of the DPI analog signals is simultaneously present on the control screen. Once the DPI is properly set up, the calibration points are shown sequentially on the subject screen. The subject is instructed to fixate each point as it appears. When the relevant ecl_response button has been pressed, specified as an argument to the calibrate() function, a number of samples before and after the button press are averaged to obtain the raw data measurement for this calibration point. By default, ecl_config defines these values to equal 150 and 50, respectively. Once all calibration points have been recorded, a linear fit is applied to the horizontal and the vertical raw data separately. The experimenter can evaluate the quality of the fit both visually and numerically on the control monitor. Insufficiently accurate measurements can be re-sampled. When the calibration routine has completed successfully, the application can retrieve the eye position in pixel coordinates on the experimental screen. The calibration can be run again at any time during the experiment.

The drift_correction() function allows immediate adjustment of the constant part of the linear fits to the calibration results, using only one calibration point. The experimenter is required to specify the relevant pixel position on the subject screen as an argument. The calibration point must be drawn manually by the experimental application; for instance, the fixation cross shown at the start of a trial could be used as the drift correction point. Drift correction is only permitted within a limited spatial range specified in ecl_config. More serious deviations from the previously established calibration will need to be corrected through a full calibration routine.

2.3.4 Sampling

The user must explicitly enable active data collection by calling **start_sampling()**. The output buffer will now be created, and the DAQ device relevant to ecl_process will start to continuously sample its channels. To prevent time-outs from occurring

when real-time data collection is required, the **sample()** function must now be called at least once every millisecond. Each call to this function will sample all DAQ channels, process these raw samples into an eye position and an eye status, and store them to any attached output buffer. If a photocell is connected to the fifth DAQ channel, its status will also be processed and saved. More frequent calls will return without consequence; that is, data sampling is auto-throttled at 1000Hz. Fetching and processing a sample will also update the internal clock of ecl_process, which is based on the DAQ clock.

The current eye position can then be accessed at any time using the **get_eye_x()** and **get_eye_y()** functions. Similarly, the current eye status can be retrieved. A saccade is detected using the algorithm described by Van Diepen (1998), on the basis of the velocity profile contained within the five-element cyclical buffer storing the latest eye positions obtained. A safe saccade requires that two additional criteria are met: A stable velocity over the entire cyclical buffer, and a minimum saccade duration. An oscillation is the absence of a saccade when ecl_process is not in blink, loss of track or time-out, but no safe fixation is detected either. Safe fixations are detected when the deviation from the mean across the five-sample cyclical buffer does not exceed a criterion value. A false lock occurs when the eye position is outside the calibrated area, augmented with a tolerance value. While false lock measurements are in principle valid, they are problematic because they determine pixel coordinates on the basis of extrapolation from the calibration measurements, rather than interpolation. The opposite of a false lock is a good saccade, fixation or oscillation. Blink and track are direct conversions of the analog DPI signals. If the subject is blinking, or the DPI machine has lost track, no further eye state can be determined. Time-out occurs when the latency between two successive samples is larger than a time-out limit (typically 1 ms). This usually means that the PC is being overburdened with calculations, either within the application or in background processes. Common causes include random network traffic or performing demanding graphical operations during data sampling. The exact criteria used in determining the eye status can be set in ecl_config. When storing the data as p-files, all of these eye status calculations can be redone, cleaned of artifacts or summarized, using the P-REPORT tool.

When data collection is no longer required, **stop_sampling()** must be called. This will reset the internal clock, and stop sample collection on the DAQ device.

2.3.5 Synchronization

If a photocell is connected to the fifth DAQ channel, ecl_process will compare its current value to a criterion value defined in ecl_config, enabling or disabling a binary flag that is stored in the output buffer. The recommended procedure is then to stimulate the photocell with a local luminance change on the experimental screen at time-critical points, such as the onset of a stimulus. Since the theoretical onset time (that is, the time at which the stimulus onset signal was sent to the graphical processing unit) can also be stored as a binary flag in the output buffer, an estimate can be made of the exact time lag between the DAQ device and the monitor device on every trial.

2.3.6 Regions

ECL allows the definition of 'regions' on the screen. Currently rectangles, ellipses, and sectors of a circle are implemented. The **ecl_region** base class can be inherited by new, user-defined region types, should the default types not suffice. Region objects can be added to the central ecl_process object. ecl_process can then easily determine inside which region of the screen the eye is currently located, if any. The boundaries of regions are for this purpose augmented by a tolerance value defined in ecl_config. A second purpose of regions is to provide the experimenter with a schematic overview of the current trial on the control screen.

2.3.7 Experimenter feedback

The experimental application has full low-level control over drawing operations on both screens. However, through the **ctrl_update_screen()** function it is also possible to automate the visual feedback given to the experimenter on the control monitor. Each time this function is called, ecl_process will redraw the current eye position, the current eye status, the regions added to ecl_process color-coded by whether the eye is consider to be inside them, and the contents of a text buffer to which the experimental application can add custom messages. Typically, this function will during active data collection be put in a loop together with the sample() function. Its update frequency is by default auto-throttled to 20 Hz.

4 Example

This example program executes a simplified version of a single experimental trial as reported in Demeyer, De Graef, Wagemans, and Verfaillie (2010). It should not be used as a template for a real experiment, since it does not contain all the necessary checks on possible errors or irregularities; it was conceived merely to illustrate how ECL works. The subject is instructed to fixate on a dot to the left of the centre of the screen, and saccade towards a circular shape to his right as soon as it appears. During the saccade, the shape is displaced in the vertical direction and changes into a square. The subject is to respond whether the displacement occurred in the upward or downward direction. All data is then saved to a file. Meanwhile, the experimenter receives feedback on a control screen.

```
// Declaration of ECL objects
ecl_daq_ni                  daq;
ecl_output_pfile            pfile;
ecl_monitor_dx              subj_mon;
ecl_monitor_dx              ctrl_mon;
ecl_response_keyboard       keyb;
ecl_response_pport          breakers(0x378);
ecl_config                  cfg;
ecl_process                 DPI;

ecl_region_circle           regf(250,300,20);       // Fixation region
ecl_region_circle           regs1(550,300,80);      // Stimulus region 1
ecl_region_circle           regs2(550,310,80);      // Stimulus region 2


// Initialization and setup of auxiliary ECL objects
daq.init(4);

subj_mon.init(hInstance, "SUBJECT SCREEN", 2, 800, 600, 200, 24, 400, 300, 1000);
// DirectX monitor #2, 800x600 pixels, 200Hz, 24bit, 400x300 mm, 1000mm distance
subj_mon.set_bg_color(RGB(128,128,128));
subj_mon.set_fg_color(RGB(200,200,200));
ctrl_mon.init(hInstance, "CONTROL SCREEN", 1, 800, 600, 85, 24, 200, 150, 500);

keyb.init();
breakers.init();
breakers.add_button("BRK_LEFT",0);          // On pin 0
breakers.add_button("BRK_RIGHT",1);         // On pin 1

pfile.set_fname("mydata");
pfile.create_buffer(5000);                          // Allocate a 5 second buffer

cfg.load("ecl_default_cfg.txt");


// Initialization and setup of the main ecl_process object
DPI.init(cfg, &daq, &keyb, &breakers, &subj_mon, &ctrl_mon);
DPI.add_region(&regf, "REG_FIX");
```

```
DPI.add_region(&regs1, "REG_STIM_1");
DPI.add_region(&regs2, "REG_STIM_2");
DPI.ctrl_disable_region("REG_STIM_2");                  // Do not draw this region yet
DPI.add_stim_flag(2);                                   // Stim flags for the output file
DPI.add_key_flag(&breakers, "BRK_LEFT");                // Keys flags for the output file
DPI.add_key_flag(&breakers, "BRK_RIGHT");


// Execute the calibration routine
DPI.calibrate();


// Preload both stimuli (circle and square)
long s1_h = subj_mon.create_preloaded_image(80, 80);    // 80x80 pixel drawing surface
long s2_h = subj_mon.create_preloaded_image(80, 80);
subj_mon.draw_oval(s1_h,40,40,10,10,-1,4,-1);
// Preloaded circle, centered on (40,40), 10 px radius, 4 px frame, default colors
subj_mon.draw_rect(s2_h,40,40,20,20,-1,4,-1);
// Preloaded square, centered on (40,40), 20 px wide, 4 px frame, default colors


// Draw the fixation point without preloading
subj_mon.draw_oval(0,250,300,2,2);
// Filled circle, no preload, centered on (250,300), 2 px radius, default colors
subj_mon.present();


// Start sampling
DPI.start_sampling();


// Apply drift correction when a button is pressed
bool done = false;
do
{
        DPI.sample();                                   // Collect and process a sample
        DPI.ctrl_update_screen();               // Update the control screen in video memory
        ctrl_mon.present();                     // Show the updated control screen

        if(breakers.is_pressed("BRK_RIGHT") || breakers.is_pressed("BRK_LEFT"))
                if(DPI.drift_correction(250,300))       // Subject is looking at (250,300)
                        done = true;

        if(keyb.is_pressed("KB_ESCAPE"))                // Allow recalibration
                DPI.calibrate();

} while (!done);


// Start saving the data from now on
DPI.attach_output_file(pfile);


// Main loop
long start_time = DPI.get_sampling_time();
long resp = 0, s2_time = 0, stim_on = 0;

done = false;
while(!done)
{
        DPI.sample();
        DPI.ctrl_update_screen();
        ctrl_mon.present();
```

```
	// Fixation period of 500 ms + minimum 150 ms saccadic reaction time
	if (DPI.get_sampling_time() < start_time + 500 + 150)
	{
		if (!DPI.on_region("REG_FIX"))
		{
			DPI.stop_sampling();
			DPI.add_to_msgbuf("ERROR: Participant did not keep fixation");
			// Display a message on the control screen
			done = true;
		}

		if (DPI.get_sampling_time() > start_time + 500 && !stim_on)
		{
			subj_mon.blit_preloaded_image(s1_h, 550, 300);
			// Preloaded circle is copied to (550,300)
			subj_mon.present();
			DPI.stim_flag_on(1);
			// Enable the first stimulus flag in the output buffer
			stim_on = 1;
		}
	}

	// Then require a saccade, and switch the stimulus when it occurs
	if (DPI.get_sampling_time() >= start_time + 500 + 150 && stim_on == 1)
	{
		if (DPI.is_in_saccade())
		{
			subj_mon.blit_preloaded_image(s2_h, 550, 305);
			// Preloaded square is copied to (550,305)
			subj_mon.present();
			s2_time = DPI.get_sampling_time();
			DPI.stim_flag_off(1); DPI.stim_flag_on(2);
			// Now show the stimulus 2 region on the control screen
			DPI.ctrl_disable_region("REG_STIM_1");
			DPI.ctrl_enable_region("REG_STIM_2");
			stim_on = 2;
		}
	}

	// Abort if no saccade was made in time
	if (DPI.get_sampling_time() >= start_time + 500 + 400 && stim_on == 1)
	{
		DPI.stop_sampling();
		DPI.add_to_msgbuf("ERROR: Participant did not saccade in time");
		done = true;
	}

	// Show the second stimulus for 100 ms
	if (DPI.get_sampling_time() > s2_time + 100 && stim_on == 2)
	{
		subj_mon.clear_screen();
		subj_mon.present();
		DPI.stop_sampling();
		DPI.stim_flag_off(2);

		// Collect response
		if (breakers.is_pressed("BRK_LEFT"))
			resp = 1; done = true;

		if (breakers.is_pressed("BRK_RIGHT"))
			resp = 2; done = true;
	}
}
```

```
DPI.stop_sampling();


// If a valid response was obtained, save the data
if (resp)
        DPI.save_output_file();
DPI.detach_output_file();


// Show the response to the control screen
char msg[50]; sprintf(msg, "Response given: %d", resp);
DPI.add_to_msgbuf(msg);


// Refresh the control monitor one last time
DPI.ctrl_update_screen();
ctrl_mon.wait_for_present();
ctrl_mon.present();
```

Acknowledgments

This research was supported by the Concerted Research Effort Convention GOA of the Research Fund K.U. Leuven (GOA/2005/03-TBA) granted to Géry d'Ydewalle, Karl Verfaillie, and Johan Wagemans, by the European Community through GazeCom project IST-C-033816 to Karl Verfaillie and Peter De Graef, and by a Methusalem grant to Johan Wagemans (METH/08/02). Maarten Demeyer is a postdoctoral research fellow of the Research Fund K.U. Leuven (PDMK/10/061).

References

Demeyer, M., De Graef, P., Wagemans, J., & Verfaillie, K. (2010). Object form discontinuity facilitates displacement discrimination across saccades. *Journal of Vision, 10(6):17*, 1-14.

Crane, H. D., & Steele, C. M. (1985). Generation-V dual-Purkinje-Image eyetracker. *Applied Optics, 24*, 527–537.

Van Diepen, P. (1998). *New data-acquisition software for the Leuven dual-PC controlled Purkinje eye-tracking system* (Psychological Reports No. 246). Leuven, Belgium: Laboratory of Experimental Psychology, University of Leuven.

Van Rensbergen, J., & De Troy, A. (1993). *A reference guide for the Leuven dual-PC controlled Purkinje eyetracking system* (Psychological Reports No. 145). Leuven, Belgium: Laboratory of Experimental Psychology, University of Leuven.