FacetrackerCLM

Usage and Examples of the Face Tracking- and Facial Gesture detection plugin

UPMC

SEVENTH FRAMEWORK PROGRAMME

European Commission
Information Society and Media

Version History

Version	Date	Changed	Author(s)
1.0	14/12/12	Draft but probably the final version ☺	Andrea Carbone

AsTeRICS – Assistive Technology Rapid Integration & Construction Set
Grant Agreement No.247730
ICT-2009.7.2 - Accessible and Assistive ICT
Small or medium-scale focused research project

Disclaimer

Table of Content

1 Introduction

The FacetrackerCLM plugin is part of the *Computer Vision* category of the *Sensors* family of AsTeRICS components. Its purpose is similar to the FacetrackerLK plugin, which detects and tracks coordinates of a human face in a live video stream usually captured from a web camera. This information can be used in AsTeRICS models for mouse control, selection of cells of an on-screen keyboard (usually an OSKA grid) or other types of control. In contrast to the FacetrackerLK plugin which tracks estimated nose- and chin positions and has low CPU performance requirements, the FacetrackerCLM is able to track more comprehensive features of a user's face as described in this document. As a remote system, it has the main advantage that no devices or sensors have to be mounted on the subject's body, which may increase the comfort for the users, especially in long-term use.

2 Using FacetrackerCLM

The FacetrackerCLM working principle is based on the detection and tracking of the face features by means of Active Shape Models. The available information from the tracker can be used to associate head movements and face gestures to mouse movements and interaction events. As a recap, the facetracker CLM software returns the best fit of a deformable face model over the current image.

A deformable model in case of faces is a sparse collection of distinguishable facial landmarks. The face is a *non-rigid* structure thus the fitting of a generic model to the current instance requires determining both global deformation parameters (rotation, scale and translation) and the set of local deformations that minimize the mismatch with the current facial pose and expression.

All outputs from the facetrackerCLM plugin are in fact computed on the basis of the explicit image coordinates of the landmarks and the parameters of the fitting algorithms. Overall there are two classes of outputs from this plugin:

- Measures are related to the change of pose of the head;
- Events associated to specific configurations of the facial landmarks, in other words expressions (i.e. eyes closed or opened).

2.1 Setup and resources needed

FacetrackerCLM needs only a webcamera connected to the system to work. Normally the best solution is to position the camera so that it can take a clean image of the face of the subject while she/he is looking at the monitor.

Some internal functions need also a number of external resources whose default relative path corresponds to the "*ARE/data/facetrackerCLM/EyeStateModels"* folder. Those resources will be automatically loaded at the execution of the plugin. The user does not need to perform any explicit management of those resources except in the case of a specific file which contains the information useful to correctly classify the state of the state of the eyes. The instructions necessary to manage those files will be explained in Section 3.

2.2 Parameters/Ports

Let's look in particular at the values exported by the plugin. Outputs can be put in two categories, those related to the head pose and those related to facial gestures.

- **Output Port Description**:

  Roll, Pitch and Yaw [double]: Angles of the current head pose.

  PosX, PosY [double]: Relative displacement in terms of image coordinates (2D) of the face centre (approximately corresponding to the nose).

  Scale [double]: Relative change of the scale of the face.

  EyeLeft, EyeRight [integer]: The eye state (Opened=1, Closed=0).

- **Event Listener** Description:

  reset: At an incoming event, the face tracker is re-calibrated (a new face recognition is performed to re-align the model). This is useful if the correct face alignment gets lost during operation of the tracking.

  showCameraSettings:
  An incoming event displays the settings window for the camera device, where parameters like image brightness or contrast can be adjusted

  setReferencePose:
  An incoming event sets the reference pose of the CLM algorithm.

- **Event Trigger** Description
  EyebrowsRaised: This event is sent out as the eye brows are raised.

- **Properties**:
  cameraSelection [string] (combobox selection):
  Using this property, the utilized camera can be chosen. Possible values range from “first camera” to “fifth camera”. If only one camera is available in the system, “first camera” shall be chosen.

  cameraResolution [string] (combobox selection):
  This selection box provides several standard camera resolutions. Changing the resolution affects accuracy and performance (CPU load of the runtime system). Provided selections include “320x240”, “640x480. If the selected resolution cannot be delivered by the image acquisition device, the next matching resolution is chosen by

the plugin.

cameraDisplayUpdate [integer]:
This property allows to select the update rate for the camera display in milliseconds. If “0” milliseconds is chosen, no window for the live-video will be displayed. If “100” is chosen, the live image window will be updated 10 times a second. Please note that this property does not influence the frame rate of the camera nor the processing interval for new camera frames, only the display in the GUI is adjusted.
Camera Index [integer]: choose which camera connected to the system should be used.

modelFileName [string]:
file name of a classification data file for the blink detector. The blink detector needs to load a trained classifier (model) for recognising the state of the eyes. The model is generated by a set of routines that are introduced in the following section. Thus it is possible to learn different models in different lighting conditions and use the one that better performs in a given circumstance. A preloaded, generic model file will be used if no model filename is given – this generic model is present in the *ARE/data/facetrackerCLM/EyeStateModels* directory in the folder of the ARE runtime.

3 Eye State – Acquisition and Training of the classifier

In order to discriminate the state of the eyes we start from the natural hypothesis that the appearance of each eye can represent only two different states: **Opened** or **Closed**. This allows us to interpret the problem as a supervised binary classification and use machine-learning tools to learn how to associate each sample to its most likely class.

There are two main steps that should be followed in order to customize the eye state detector used in the plugin thus overriding the default classifier parameters that could not be tuned to the actual lighting conditions and/or eye appearance characteristics. The steps and the corresponding software tools will be introduced in the following sub-sections:

1. Recording and storing the samples.
2. Training the classifier.

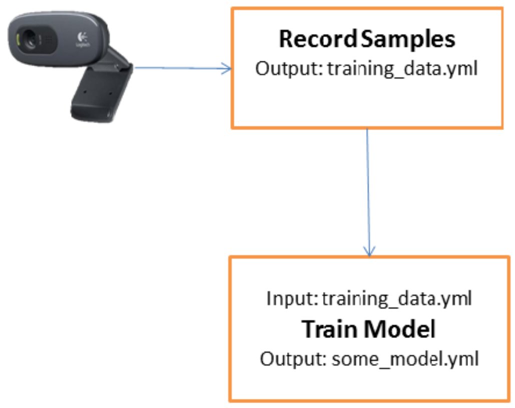


Figure 1

3.1 Recording Samples

The blink detector is included in the facetrackerCLM plugin but requires a model description file as input. This file is produced by a set of routines that takes as input a collection of samples for both classes to detect: opened and closed eyes.

The samples recorder is called *EyeStateRecord.exe* and manages the training dataset, you can find it in the folder *ARE/tools/EyesStateTrainer*. The implemented operations allow creating, updating and modifying the training set that consists of a set of labelled samples {Ei, Li} where Ei is the i-th cropped eye sample and Li is the associated label: **1** if Ei is an instance of the *'Closed'* class, **0** otherwise (catches all other appearances). The output dataset is a YAML file (an XML-like markup format) that is the official technology used by OpenCV to serialize/deserialize matrices and data structures.

3.1.1 Command line parameters

At the moment, the executable needs a number of command line parameters, here some examples of invocations:

```
EyesStateRecord --create  subject1.yml --camera 0 --width 25

EyesStateRecord --append  subject1.yml --camera 0 --width 25 --resample 250

EyesStateRecord --append  subject1.yml --camera 0 --width 25 --resolution 640

The parameters in details:
```

- **create/append** followed by the name of the file with 'yml' as extension.
  - create will create a new log.
  - append will add new samples to some an repertoire of samples.
- **camera** *idx:* where idx is the integer index of the camera we'd like to use
- **width** *w:* it's the width in pixel of the normalised patch of each eye (stay around 25/30 pixels).

- **resample** *num***:** when using several times *append* the dataset file will grow quite a lot. If we want to acquire new samples but keep only a subset of them, then it's possible to specify how many eye-pairs we want to save. Samples are randomly sampled at the end of the session.
- **Resolution** *num:* only two options can be set: 320 and 640, the former will set the captured images to 320x240 the latter to 640x480.

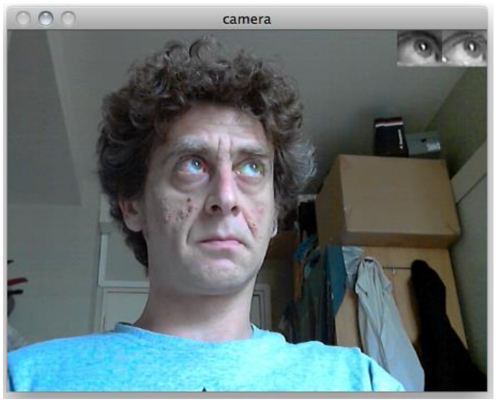


Figure 2 -– In the upper right corner the current detected eyes, normalised in scale and rotation.

3.1.2 Runtime Usage:

The program needs some interaction by a secondary user to correctly associate labels to samples according to the current state of both eyes. Normally the software is in an *idle* state; this means that it doesn't save any information but just shows the current image and the cropped images of both eyes normalised and corrected in scale and rotation.

The interaction takes place through the keyboard:

- Press **'o'** (the letter) to acquire **'OPENED** eyes' samples. Press again 'o' or **SPACEBAR** to stop.
- Press **'c'** to acquire **'CLOSED** eyes' samples. Press 'c' again or **SPACEBAR** to stop acquisition.
- Press **'r'** (as 'reset') if the facetracking status is not satisfying (the visible mesh is not correctly aligned to the face).
- Finally press '**ESC'** to save and quit the recording.

Figure 3 shows two examples of the acquisition window during both phases. As a suggestion consider acquiring multiple short sequences (few seconds) for both states alternating opened and closed eyes states in order to avoid the unavoidable discomfort in holding the eyes opened or closed for longer times.

Although there are no specific recommendations about the light settings of the environment, it is advisable to avoid high contrast situations in which the face receives lights mainly from one side. In this case, the facetracker is likely to be fooled very often thus affecting the quality and reliability of the cropped eyes samples. The ideal situation could be represented by a uniformly distributed illumination.

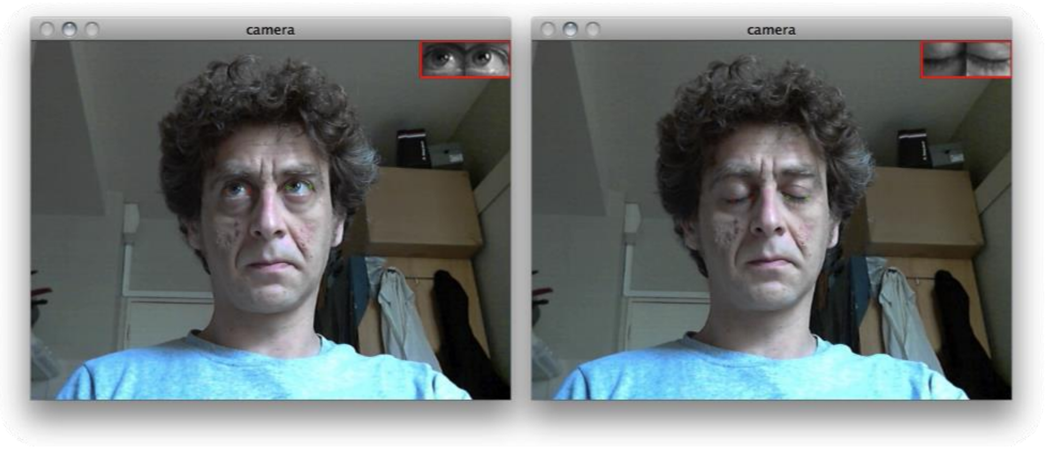


Figure 3 - During the acquisition of the training set. Open (left) and closed (right) eyes.

3.2 Training the model

This is a crucial routine since it computes the parameters for the PCA projection and the SVM classifier (described in D4.6a). The generic call format is the following:

```
EyesStateTrain --input subject1.yml --eigenratio 0.8 --samples 250 --model
subject1_model.yml
```

- **--input** [string]: the name (with extension) of the file containing the dataset we intend to use.
- **--eigenratio** [double] : this is the parameter that influences the number of eigenvectors of the reduced dimensionality subspace (values between 0.8/0.9 will work fine).
- **--samples** [integer]: how many pairs for each category to use in learning the SVM classifier (so, in this case there will be 4*N total samples). When absent, all available samples will be used to compute the PCA projections and train the SVM classifier. The number of samples will be in any case checked against the real number of available samples.
- **--model** [string]: the name of the model

4 Using the Model (Testing and Inclusion in the plugin)

In the EyesStateTrainer directory (*ARE/tools/EyesStateTrainer*) you will find all output files, both the samples log and the trained models. At this point it is possible to quickly test the performance of the classifier and/or make it visible to the FacetrackerCLM plugin.

A first check can be done directly using a custom executable which loads the newly created model description and lets you verify if the detection results are satisfying. In the next section you'll find the instructions for the command line usage of the program.

4.1 Detection Test

Open the windows shell (commandline window) and move to the EyesState tools directory in the folder *ARE/tools/EyesStateTrainer* where you'll find the executable **EyeStateDetector.exe**. An example of invocation:

```
EyeStateDetector.exe --model subject1_model.yml --camera 0
```

--model [string]: the name of the model to test.

--camera [int]**:** the camera index, optional parameter, 0 by default.

Press **ESC** to quit the application.

4.2 Use the model in the plugin

In this section we'll do a full recap of all the steps introduced up until now. We will record a set of samples and then we will use them to train a detector that we will use in the FacetrackerCLM plugin. The first step is to open a windows shell (commandline window). In Windows 7 it is sufficient to type and enter the command "cmd" in the search bar in the Start Menu as in Figure 4.

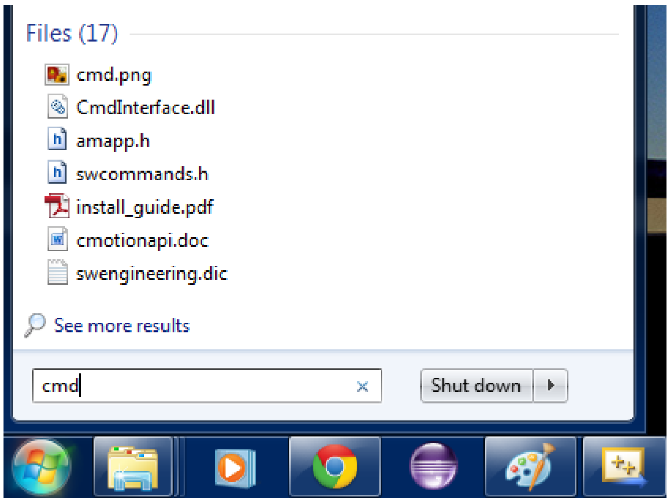
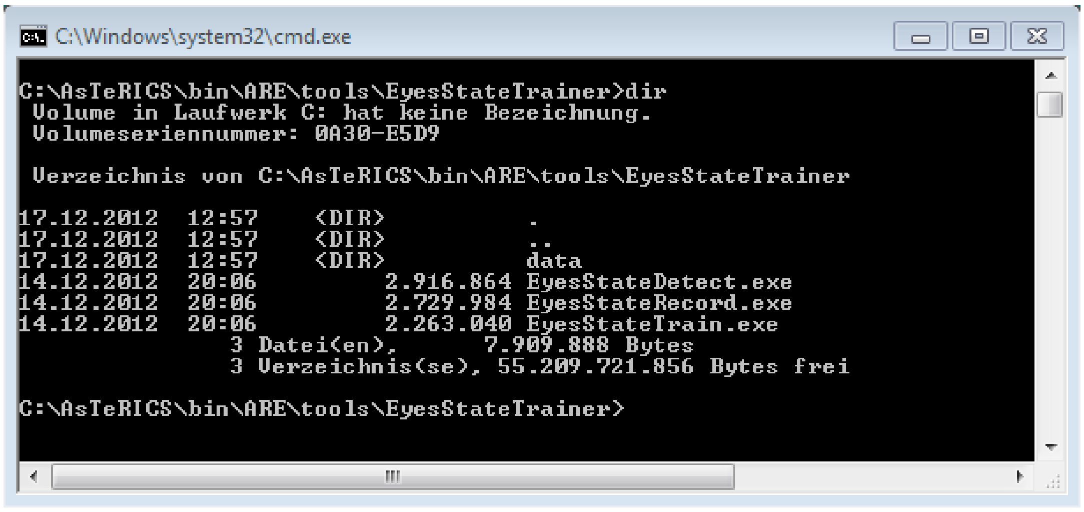


Figure 4

Afterwards we need to change the working directory to the folder containing the software tools we want to execute (as in Figure 5):



```
C:\AsTeRICS\bin\ARE\tools\EyesStateTrainer>dir
 Volume in Laufwerk C: hat keine Bezeichnung.
 Volumeseriennummer: 0A30-E5D9

 Verzeichnis von C:\AsTeRICS\bin\ARE\tools\EyesStateTrainer

17.12.2012  12:57    <DIR>          .
17.12.2012  12:57    <DIR>          ..
17.12.2012  12:57    <DIR>          data
14.12.2012  20:06         2.916.864 EyesStateDetect.exe
14.12.2012  20:06         2.729.984 EyesStateRecord.exe
14.12.2012  20:06         2.263.040 EyesStateTrain.exe
               3 Datei(en),      7.909.888 Bytes
               3 Verzeichnis(se), 55.209.721.856 Bytes frei

C:\AsTeRICS\bin\ARE\tools\EyesStateTrainer>
```

Figure 5

Now we type the command at the prompt:

```
EyeStateRecord.exe --create testdata.yml --camera 0 --resolution 320 --width 25
```

This command will open a window showing the streamed images from the connected camera. As explained in the above sections, we are now able to record images of our opened or closed eyes by pressing the keys “o” and “c” on the keyboard.

As a suggestion it is advisable to alternate the recording of both modalities (open and close) each for a short period of time (few seconds).

As soon as we are satisfied with the recorded samples, we can quit the application by pressing the ESC key, and finally pass to the training phase.

Of course we can continue to record new samples whenever we think it’s time to update the dataset or build a new dataset to be used for a specific lighting condition. Don’t forget to use the option `--append` instead of `--create` when adding samples to an already existing dataset.

Now let’s consider the training phase:

```
EyeStateTrain.exe --input testdata.yml --samples 250 --model testmodel.yml --eigenratio 0.8
```

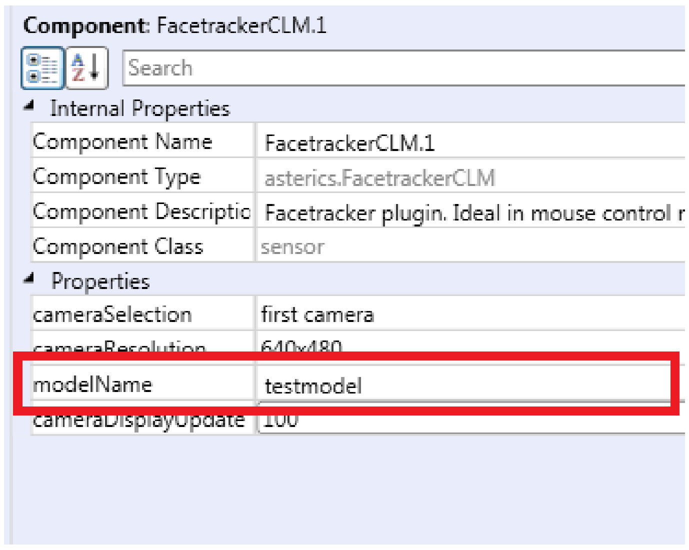
So we are using the previously recorded samples stored in testdata.yml. The `--eigenratio` parameter decides implicitly the dimension of the feature space that will be finally used in the training phase. It is also somewhat loosely tied to the degree of denoising applied to the original images captured (whereas lower values give higher degree smoothing).

The outcome of this instruction is a file (*testmodel.yml*) describing the parameters of the SVM classifier that is the only information that we need at runtime to correctly classify the state of both eyes.

The `--samples` option simply tells how many samples should be used during the training phase. The final number of samples will be four times the number entered (2 states x 2 eyes x # of samples).

Now we are ready to use this brand new model in the FacetrackerCLM plugin but first we need to move the ‘testmodel.yml’ file to its proper location where it can be found by the plugin: into the folder *ARE/data/facetrackerCLM/EyeStateModels/*

Now let’s open the ACS GUI and set the **modelName** property as in Figure 6 with the usual recommendation to skip its file extension.



Component: FacetrackerCLM.1

Search

Internal Properties

Component Name	FacetrackerCLM.1
Component Type	asterics.FacetrackerCLM
Component Descriptio	Facetracker plugin. Ideal in mouse control r
Component Class	sensor

Properties

cameraSelection	first camera
cameraResolution	640x480
modelName	testmodel
cameraDisplayUpdate	100

Figure 6

Now the FacetrackerCLM plugin will load and use the recommended trained model to discriminate the state of eyes.

5 One Full example

In **Figure 7** it is depicted the full model named *Cammouse_CLM_test_demo.acs* that is available in the “*ACS\models\componentTests\sensors*” folder. It implements a mouse control model driven by the movement of the head and explicit left and right clicking event associated to the (prolonged) blink of the left and right eye respectively.

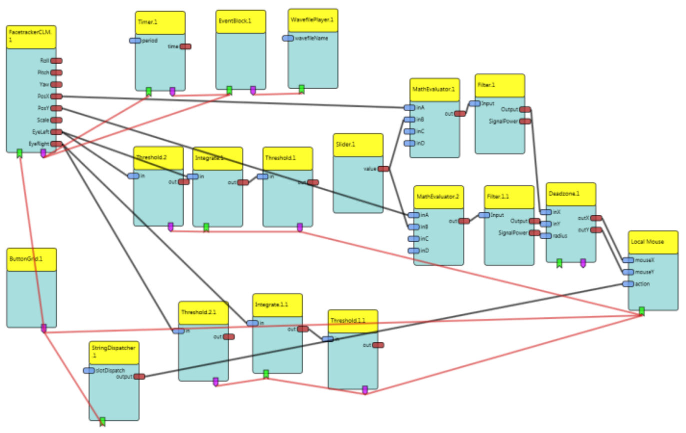


Figure 7 Cammouse_CLM_test_demo.acs

In Figure 8 the highlighted modules are responsible for triggering a succession of closed left eye state (the left in this case). In other words those 3 blocks act as a counter. Each time the eye is classified as open the counter (the integrator) is reset to 0. This is a way to filter out spurious false detection and short voluntary eye blinks.

.

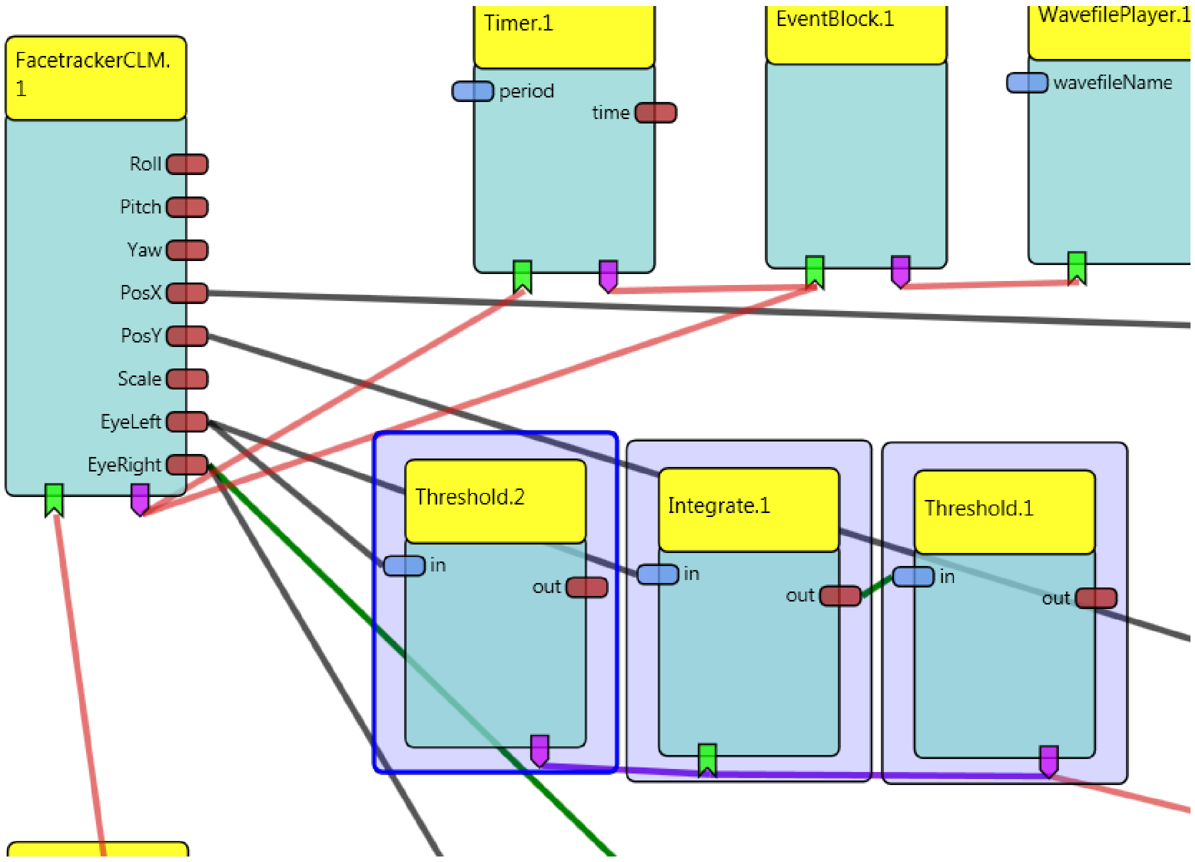


Figure 8 Detail of the blocks that trigger a sequence of N sequential detection of the closed left eye.

The GUI elements (Figure 9) that shows in the ARE window implement a set of several other mouse functions and also condition the runtime execution of the plugin.

The first two buttons from the left are responsible to Enable/Disable the control of the mouse through the model. The following three buttons instead set the ‘next click’ to the corresponding function (drag, right click, double click). The last two buttons let the user reinitialise the face detection in case the model has drifted away from the subject’s face. The last one opens the camera settings window.

The last important GUI element is the slider. It basically serves as an amplifier of the PosX and PosY outputs of the plugin (it is worth to remember that those outputs are a relative displacement of the nose between two consecutive frames).

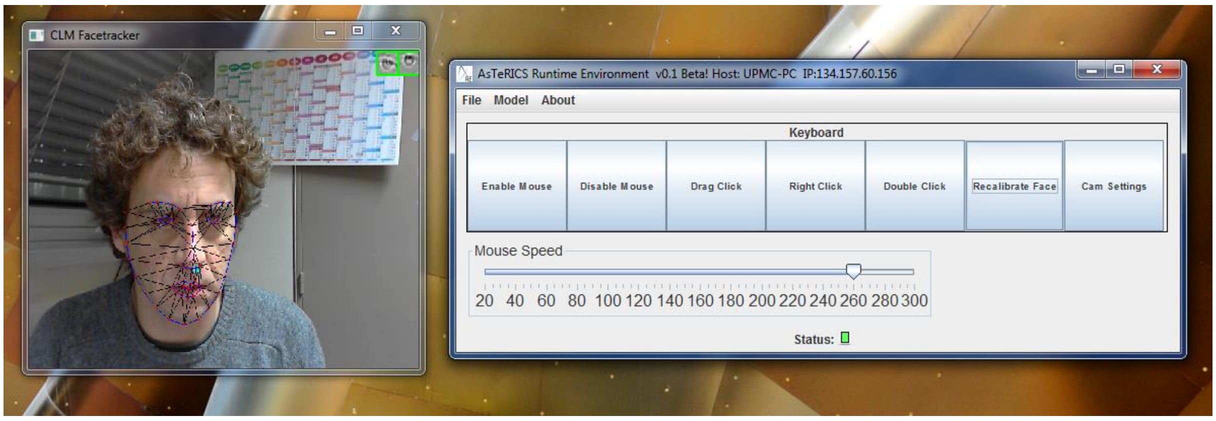


Figure 9 FacetrackerCLM in action.