ABSTRACT

Nowadays system’s user experience can be considerably improved by tracking the gaze of the user since an explicit input can be transformed into implicit or extended, which encourages the user focus to be on the desired task rather than the interaction with the system.

Gaze tracking systems are usually implemented on stationary processing computers because heavy processing is needed. With the emergence of mobile and wearable devices a new possibility for gaze tracking has appeared alongside. However, implementing a gaze tracking system on such devices implies new challenges not currently present on stationary ones, such as mobility and limited available resources.

This paper presents EyeDroid, a low cost mobile video-based gaze tracking system designed to be used with a USB head mounted camera. Unlike other gaze tracking systems, EyeDroid performs all its processing workload in a mobile device and sends the resulting coordinates of the incoming video streaming to a network client. For this reason, a stationary processing server is not needed, which encourages mobility when used along wearable and mobile devices.

Author Keywords

EyeDroid, Gaze tracking, Android, OpenCV, JLPF

ACM Classification Keywords

Human-computer interaction: Ubiquitous and mobile computing; Computer graphics: Image manipulation

1 INTRODUCTION

Due to the emergence of everyday life wearable and mobile devices, the need to develop unobtrusive systems has become more significant, therefore a redefinition of traditional ways on which users interact with computers is required. Such redefinition needs to provide more natural and intuitive interfaces to interact with this kind of devices and move from explicit to implicit input sources. In order to overcome this need, a device should be aware of the environment and use the surrounding context information. [3]

User body gestures can be potentially used as a valuable input system source due to it’s high naturalness, specially eye movements which also are completely unobtrusive.

Eye tracking has been studied widely in the past years as an unobtrusive alternative to typical Human-Computer Interaction (HCI) and applied to different fields, such as assisting technologies and augmented reality, between others. By extending or replacing a system input for a gaze tracking interface, new possibilities arise to improve the users experience. For instance, a camera that tracks the gaze of the user could be used to recognize eye movements and use them to perform certain actions, such as controlling a wearable device in a more natural way, allowing the user to focus on other activities while the device can be in the periphery of the user’s attention.

As mentioned above, eye tracking could be seen as the solution for the HCI challenges raised by the emergence of mobile and wearable devices, but current gaze tracking systems do not provide the required features to fully overcome these problems and are usually expensive. For this reason, implementing a low cost eye tracking system capable to be used along with such devices implies new challenges.

The first challenge present while using mobile and wearable devices is the need of developing low resource consuming algorithms that can be executed fast enough and have an acceptable accuracy level. Gaze tracking requires image analysis techniques that are computationally intensive and therefore it is difficult for wearable and mobile devices to perform this kind of processing on the device itself due to the lack of optimal resources, such as limited battery life, processing power and network capabilities. For this reason, efficient algorithms able to perform the required image analysis are needed in order to avoid delegating computational workload to an external computer, as is usually done in other gaze tracking systems.

A second challenge is given as a consequence of the previous one. When doing cyber foraging, mobility becomes a problem in the way that an external static computer capable of performing heavy processing has to be connected to the device. For this reason, the mobility of wearable and mobile devices is limited by the external computer connection range.
EyeDroid is an Android platform low cost mobile gaze tracking system designed to be used with an infrared USB 2.0 head mounted camera. EyeDroid receives video streaming from the user’s eye as input, process it and sends the resulting 2-axis coordinates to a networked client.

The remainder of this paper is organized as follows: Section 2 summarizes previous studies on eye tracking and mobile image processing. Section 3 describes the methodology followed during the whole process. The proposed system is introduced in section 4, evaluated in section 5 and discussed in section 6. Finally, the paper’s conclusions are summarized in section 7.

2 RELATED WORK

Gaze tracking

This section briefly reviews previous work on gaze tracking.

Eye tracking using a video camera has been extensively studied in the literature, particularly in the field of Human-Computer Interaction (HCI). Depending on the available technology, two basic types of eye tracking systems can be distinguished, electro-oculography and video-based. Electro-oculography can be less unobtrusive because small electrodes can be positioned around the eye of the user, but specialized hardware is needed. In contrast, video based techniques can be used along with regular low-cost cameras that can be placed either close to user’s eye for remote recording or head mounted.

A wide variety of video-based products have been released to the market, such as the case of open source Haytham project [5] stationary gaze tracking system. Haytham project provides both remote and head mounted eye tracking software. The technique used on this software to detect the pupil is based on predicting a region of Interest (ROI), applying image filters, removing outliers and blob detection.

Similar to haytham project, Sewell [7] presented a methodology for real-time eye gaze tracking using standard web-cam without the need for hardware modification. Additionally, it describes the methodology used for pupil detection, which rely first on user’s face detection, eye region image cropping and finally, pupil detection. Because a regular web-cam image resolution and/or appropriate lighting to find the pupil might be inadequate, an extra computation to determine the gaze using neural networks was used. This approach resulted to be very accurate compared to other gaze tracking systems but extensive calibration and heavy computation was needed. For this reason, in order to overcome the challenges on mobile and wearable devices, a similar pupil detection technique to Haytham project was used along some optimizations for EyeDroid implementation, which resource usage is lower. OpenCV library was used in both Haytham project and Neural network approach for image processing.

Other eye tracking systems have been developed, such as MobiGaze [6] and the EyeTribe [8] (commercial product). MobiGaze research project tried to provide a new way of interaction with the mobile device by building a remote eye tracker. By attaching a stereo camera on the device it was able to extract the position of the gaze and use this information as input to the device. In a similar way, the EyeTribe is a two infrared camera based external device that can be connected to mobile devices for remote gaze tracking.

Though its intrusiveness, head mounted eye trackers provide higher accuracy than remote trackers and can be applied to mobile use cases, such as the case of Kibitzer [1], a wearable gaze-based urban exploration system. Kibitzer used computer vision techniques to determine the gaze of the user. It suggests the usage of a head mounted camera in a bike helmet, along with an Android mobile device and backpack held laptop. First, the camera sends the captured image to the processing laptop via USB cable, afterwards the computer sends the gaze data and send it to the mobile client through a socket-based API. In a similar way, the openEyes eye tracker [4] is proposed. Their solution provides both a low-cost head mounted camera and a set of open-source software tools to support eye tracking. OpenEyes was intended to be mobile, therefore the processing unit was carried on a backpack. However, in both head mounted scenarios unobtrusiveness level is low due to the size of the processing units carried in the back of the user.

Mobile image processing

3 Background & Research Method

The project followed an iterative incremental process with five iterations were new functionality was built on top of the previous iteration.

The first iteration consisted of the overall design of the system architecture. This design included only the processing core of the system that was implemented as a platform independent external library to encourage future portability.

In the second iteration the processing core was implemented and tested based on the design decisions that were taken in the first iteration. At the end of this iteration the processing library was imported to the Android platform to be used by the gaze tracking algorithm. A first prototype was build.

As part of the third iteration, the image processing algorithm was implemented using the Android native support for C++ to improve the system performance and added to the core. Further optimization was performed once it was deployed in the Android platform.

During the forth iteration, the final prototype was built and a simple GUI was built in order to improve the user interaction.

The fifth and final iteration consisted on algorithm configuration tuning and evaluation.

4 THE EYEDROID GAZE TRACKER

EyeDroid is an Android platform based low cost eye tracker which is intended to be used along with a head mounted USB infrared camera. EyeDroid, offers a truly mobile solution as all the required equipment for the system can be carried by the user. Unlike other gaze tracking systems, all the image
processing is done on the smartphone itself without the need of delegating the task to an external processing server. The input to the system consists of real time video of one of the user’s eyes provided by a connected infrared camera that is directly connected to the Android Device. The output of the system can be send to any TCP/IP client that can consume the produced gaze coordinates.

**Design decisions**

Below, the most important decisions during design and implementation are presented.

- Implement an independent processing framework as basis (JLPF). The reason to create it independently from the image processing process and the Android platform was that this component could be implemented and tested separately of the main system, build the actual gaze tracking system on top and provide portability for future implementations.

- Use Pipes and filters architectural framework. The reason behind this decision was that the image processing algorithm was not known in advance and so there was the need for a design that allowed flexibility and experimentation on the algorithm itself to improve performance. This approach allowed the gaze tracking algorithm to be decomposed, ran and tested under different scheduling policies.

- Enabling parallel execution of the processing algorithm. Since performance was a key issue for the system, there would be a performance penalty if the algorithm was implemented to run in a single thread, specially when regular video frame ratio had to be processed. Additionally, configurable scheduling policies were allowed to improve performance and balance workload between threads.

- Passive consumer-producer pattern on architecture pipes. In order lower resource consumption, different passive implementations of the architectural pipes were done to fit a variety of scheduling policies. The producer-consumer pattern simplified workload management by decoupling filters that may produce or consume data at different or variable rates.\[2\]

- Algorithm composition. The steps on which the algorithm was decomposed and the parallel execution policies were decided based on experimentation. Once the prototype was build, both the algorithm itself and its scheduling policy were tuned up until the most efficient configuration was found.

- Most recently computed frame for region of interest (ROI) prediction. The algorithm uses feedback from the most recently processed frame to predict the region of interest (ROI) around the eye instead of the immediate frame before. During sequential execution, feedback from the previous frame can be used once it is processed to predict the current ROI, but it produces an execution bottleneck because a frame needs to wait until the one before has finished to proceed. By using the feedback from the most recently computed coordinates, this problem can be solved. As described below in the subsection Image processing algorithm, the feedback from frame N does not necessarily affect the frame N+1 because the gap between frames feedback can be of more than one. This issue was considered as acceptable since the maximum difference between two frames can be configured by using constant size pipes.

When executing in parallel, even though each step could potentially run on a different thread, there is no deterministic execution of the individual steps which can lead to erroneous feedback.

- Android NDK usage. Android NDK was used instead of the regular Android SDK because of its performance boost.

- Generic use of input and output to the processing core. This decision made it easy for the processing core to work with different kind of inputs and outputs. For instance, it is transparent to the core whether frames are provided by the network, a file or a camera.

**Hardware**

In the current implementation of the EyeDroid gaze tracker the only hardware requirements are an Android mobile device and a head mounted USB 2.0 infrared camera connected directly to the phone. This minimal and cheap hardware requirements make the system more usable and attractive to the users.

**Software**

Java Lightweight Processing Framework

According to the design decisions, pipes and filters design pattern (or pipeline) was used as the main architectural framework. Since the algorithm to be used was not known in advance and variable parametrization was needed to test different algorithm configurations in order to be optimized, this design provided flexibility enough to experiment with different parameters to customize the processing steps required to perform gaze tracking.
The Java Lightweight Processing Framework (JLPF) was built as an external library in the first iteration of the development process with the notion in mind that it should be platform independent and be able to perform any kind of processing and not just image processing. The idea behind the design was to decouple as much as possible the whole algorithm and its scheduling execution policy. Since the target platform is Android running on a mobile device performance was a key issue. This design allowed for a fully configurable algorithm in terms of decomposable steps and how these steps should be scheduled for execution on the available processing resources, instead of a monolithic algorithm that would perform poorly. Finally, in order to divide the algorithm in steps of equal execution time, the composite pattern was implemented to allow composition of individual steps.

**Image processing algorithm**

Since performance is important due to the lack of available resources (compared to a stationary gaze tracking system), an important decision was to use the Android NDK support for C++ instead of the regular Android SDK for java. This decision allowed the algorithm code to run directly on the processing resources and access system libraries directly, unlike Java which would run on a virtual machine. Moreover, this allowed for independent development and testing of the actual processing algorithm that was later added to the main Android application.

For the image processing the OpenCV library was used. Below are listed the individual steps of the actual algorithm, how they were composed and scheduled in order to optimize the algorithm execution. It should be noted that each frame passes through all the steps in the exact same order as it was originally provided.

1. **Eye Region of interest (ROI).** The first time a frame is received, a constant ROI is defined in the center of the image, covering the whole user’s eye. This region is later used to look for the existence of the user pupil on a smaller image than the original one in order to minimize the processing overhead. Other gaze tracking systems, such as Haytham project, uses immediate previous frame computed coordinates to define new ROI’s to increase accuracy. Following this approach, a sequential algorithm execution is needed because each frame depends on the previous one in order to be processed. This paper now proposes a simple technique to estimate subsequent ROI’s that allows the parallelization of the algorithm. Once a gaze is found in subsequent steps of the filter, the ROI is reduced and moved to the most recently computed pupil coordinates. This technique was adopted based on the assumption that if the algorithm execution is fast enough, the ROI of the current frame being processed is very similar to the most recently computed one. In case that previous coordinates has not been computed yet, the default constant ROI us used. Because each frame is now independent from a previous one, parallelization of the algorithm steps can be done. Finally, if the pupil is not found then the default constant ROI is used until an appearance is detected again.

![Figure 2. EyeDroid software architecture](image)

2. **RGB to Grey conversion.** The second step of the algorithm converts the original image ROI into gray scale. This makes the frame even smaller as there is one byte per pixel instead of three.

3. **Erode-dilation.** Erode and dilation is performed both before and after the threshold step. This step is used in order to smooth the corners in the image.

4. **Thresholding.** The exact type of the thresholding used was binary inverted. The result of this operation was a new image where the most dark parts of the original image were converted to black while the most bright parts of the image were converted to white. This way the pupil is represented now as a black blob in the image while removing any unnecessary information in the rest of the image.

5. **Erode-dilation.** This step is performed in case the thresholding step detected other dark blobs in the image except the pupil. By using erode in the output image any small dark blobs are shrunked until they disappeared. Dilation was used to bring the pupil blob back to its original size. This step was necessary in order to remove blob outliers.

![Figure 3. ROI prediction technique](image)
6. Blob detection. In this step the actual pupil blob detection is performed. After each frame is processed by all the previous steps the result is a white image with black blobs in it. This makes easier for the detection method to find blobs. The method that was used in this step is the HoughCircles from the OpenCV library.

7. Gaze detection. Finally, because the previous step can detect many blobs, the one that is closest to the center of the image is taken as the pupil and the 2-axis coordinates are computed. This information is used later as feedback to the first step in order to compute the ROI on subsequent frames based on the location of the pupil detected in the this step.

In order to fit all these steps in the most efficient way into filters and work along the JLPF library, two composite filters were used. The first one contained the steps 1-5 and the second one the steps 6 and 7. This composition was chosen after computing the execution time of each step.

A result comparison of each processing step using the proposed ROI prediction technique against a fixed ROI can be seen in figures 4 and 5. In figure 5 it can be seen that the ROI is smaller, meaning that in a previous frame the pupil was detected. For this reason, the ROI was moved and the image around the eye was cropped to improve confidence on later detections. In figure 5 the default constant ROI is used which means that in the previous frame the pupil was not detected.

![Figure 4. Processing steps with dynamic ROI prediction](image)

![Figure 5. Processing steps with constant ROI](image)

Input/Output

Since the core was decoupled from the input and output implementations, the system could be used for example as remote server that receives the video streaming from a remote source, performs the processing and sends the result to a networked client. In the current implementation, the available input sources can be given from a networked source, the cameras installed on the device itself or a USB camera connected to the phone. Input video streaming frames are initially converted to a resolution of 640x480 px. As available output, resulting coordinates can be send to a TCP/IP connected client. Due to the modifiability the architectural design provides, future input sources and destination outputs could be added.

5 EVALUATION

6 DISCUSSION

7 CONCLUSION

8 ACKNOWLEDGEMENTS
REFERENCES


