Automatic Areas of Interest Detector for Mobile Eye Trackers

Konstantin Ryabinin[1,2], Svetlana Alexeeva[1] and Tatiana Petrova[1]

[1]Saint Petersburg State University, University Embankment, 7/9, Saint Petersburg, 199034, Russia
[2]Perm State University, Bukireva Str., 15, Perm, 614068, Russia

Abstract

The paper deals with automatic areas of interest detection in video streams derived from mobile eye trackers. Defining such areas on a visual stimulus viewed by an informant is an important step in setting up any eye-tracking-based experiment. If the informant's field of view is stationary, areas of interest can be selected manually, but when we use mobile eye trackers, the field of view is usually constantly changing, so automation is badly needed. We propose using computer vision algorithms to automatically locate the given 2D stimulus template in a video stream and construct the homography transform that can map the undistorted stimulus template to the video frame coordinate system. In parallel to this, the segmentation of a stimulus template into the areas of interest is performed, and the areas of interest are mapped to the video frame. The considered stimuli are texts typed in specific fonts and the interest areas are individual words in these texts. Optical character recognition leveraged by the Tesseract engine is used for segmentation. The text location relies on a combination of Scale-Invariant Feature Transform and Fast Library for Approximate Nearest Neighbors. The homography is constructed using Random Sample Consensus. All the algorithms are implemented based on the OpenCV library as microservices within the SciVi ontology-driven platform that provides high-level tools to compose pipelines using a data-flow-based visual programming paradigm. The proposed pipeline was tested on real eye tracking data and proved to be efficient and robust.

Keywords

Eye Gaze Tracking, Area of Interest, Video Segmentation, Image Template Detection, OpenCV, Scientific Visualization.

1. Introduction

Eye tracking is a modern technology widely used in psycholinguistics, cognitive psychology, vision research, marketing, usability, etc. as a window to cognitive processing in action. The core concepts of eye tracking are fixations and saccades. The former represents eye stops when the new information is obtained and the latter are very quick jumps (e. g. 20–50 ms in reading) that shift the eyes to the new part of the visual object [1]. Due to saccadic suppression [2] the visual input is received only during fixations. The influential eye-mind hypothesis formulated by Just and Carpenter [3] states that "there is no appreciable lag between what is fixated and what is processed". This means that fixation durations on visual objects (e. g. words in a text being read) directly reflect the amount of cognitive efforts required to process them. More recently

GraphiCon 2022: 32nd International Conference on Computer Graphics and Vision, September 19–22, 2022, Ryazan State Radio Engineering University named after V.F. Utkin, Ryazan, Russia
kostya.ryabinin@gmail.com (K. Ryabinin); s.alekseeva@spbu.ru (S. Alexeeva); t.e.petrowa@spbu.ru (T. Petrova)
0000-0002-8353-7641 (K. Ryabinin); 0000-0002-8540-6178 (S. Alexeeva); 0000-0002-6711-5385 (T. Petrova)

the theory was refined [4]: it was pointed out that fixation durations show not only moment-to-moment processing of currently fixated visual objects but also preprocessing of visual input in parafovea, the part of the visual field that extends out to 5° on either side of fixation. Therefore, it is extremely important to analyze separate parts of a complex visual object (e. g. individual words in a text) to get a deeper understanding of what factors of fixated/non-fixated objects influence cognitive processing.

Technically, a so-called interest area report that comes with most eye tracker software (e. g. Eyelink, Tobii, Eyegaze, etc.) is usually carried out. To generate such a report, a researcher first has to divide manually or automatically a visual stimulus into areas of interest (AOIs, e. g. words in a text). When dividing manually, a researcher usually just draws several rectangles or other shapes in the respective software around objects of interest [5]. This is a more or less easy task for stationary eye trackers, which use a computer screen to show visual stimuli since their software has information about stimuli location on the screen, and the informant does not move their body and head during the experiment. However, it is not the case for mobile eye trackers [6, 7]. Mobile eye trackers are glasses-like devices, which have cameras facing the informant's eyes (for tracking the informant's gaze) and field-of-view camera facing forward (for recording what the informant sees). The reasons are the following.

First, mobile eye trackers often have cameras of lower quality than stationary ones. This results in poorly legible recorded scene images. Second, the working area or the part of the visual field where visual stimuli could be located is predetermined in advance in stationary eye trackers, whereas even static objects of interest recorded by a mobile eye tracker could appear in different parts of successive images due to head movements. For example, by text reading, let the word "cat" be printed in the last paragraph of one-page text that is required to be read by an informant. At the beginning of a reading, this word is located at the bottom of the visual field. But approaching the end of the text, an informant lowers their head to read the last paragraph. The eye tracker's field-of-view camera also lowers, and as a result, the word "cat" appears at the top of the visual field, and all the previous text is cut off. Finally, the images recorded by a mobile eye tracker are much noisier than those that came from stationary ones. When the screen is used, visual stimuli are usually placed on a white/gray plane therefore everything that does not correspond to background color is a visual object of interest with high probability. When a mobile eye tracker is used, it is difficult to differentiate between foreground and background objects. All mentioned above makes the task of automatic object detection in images recorded by mobile eye trackers to be very complicated. Moreover, the manual AOIs generation (by drawing some shape) is not always provided by manufacturers.

In our experiments at Saint-Petersburg State University, we use the mobile eye tracker PupilCore by PupilLabs (https://pupil-labs.com/products/core/). PupilLabs produces an eye tracking platform that consists of open source software and wearable eye tracker glasses. Corresponding visualization software (PupilPlayer) allows to export all the fixations for the full recording or the part of it. It is not possible just to draw AOIs inside the recordings; instead, ArUco markers [8] should be placed in the AOIs corners to allow the software to generate the corresponding AOIs bounding boxes automatically. Then one could export eye movement data for these AOIs. The drawbacks of such a procedure are the following. First, the visual stimulus becomes less natural, and "strange stickers in the corners" may distract the informant's attention. Second, conceptually it is not an option to place the markers inside a big AOI to

maintain the AOI hierarchy (e.g. words in the texts or magazine pictures). These markers just disrupt the integrity of the complex visual objects. Automatic AOIs detection algorithms for mobile eye tracker data could overcome these drawbacks and should be explored more thoroughly. This is the goal of our present study.

In this paper, we propose a new pipeline for automatic detection of texts and words inside the texts on video streams obtained from the mobile eye tracker PupilCore by PupilLabs.

To test the proposed pipeline, we used the data from the following eye-tracking-based experiment. 72 adolescents read printed versions of five texts (text examples and their detailed description are provided in an Open Science Repository project, https://osf.io/4m59b/). Each text occupied no more than one page. The font factor was manipulated: texts were presented in Verdana, Roboto, Times New Roman, and LexiaD, a Cyrillic font created for people with reading problems [9]. The sampling frequency of eye tracking was 200 Hz. The recording was performed in binocular mode with a narrow-angle lens for the field-of-view camera (1080p: $88^\circ \times 54^\circ$). Before reading each text, participants underwent an equipment calibration using an on-screen marker and a 3D view display in the Pupil Capture 2.6.19 app. The on-screen marker was presented on the laptop screen ($14''$ 1980×1080, 309×175 mm). The screen was positioned slightly below eye level, the distance from the eyes to the screen was between 600 and 700 mm.

The AOIs detection pipeline is implemented within the SciVi microservice platform (https:/scivi.tools/). This pipeline has a modular structure allowing fine-tuning of each step for a particular AOIs detection task.

2. Key Contributions

In this work, we propose a set of computer vision and scientific visualization tools to automate the detection of AOIs by given templates in video streams derived from mobile eye trackers. The following key results can be highlighted:

1. The robust method to segment the video into AOIs preserving fine details and maintaining the AOIs hierarchy (e.g. detecting the text and individual words in this text).
2. The set of server-side SciVi platform microservices to construct the templates for text-based AOIs, load video streams, and automatically detect the templates in loaded streams.
3. The corresponding client-side SciVi platform microservice to monitor the detection process and assess the detection quality.

The above tools allow us to compose robust AOI detection pipelines using the high-level visual programming environment of the SciVi platform and thereby automate the video segmentation for subsequent analysis of eye gaze tracks recorded by mobile eye trackers.

3. Related Work

An analytical review of the literature shows that automation of AOIs detection is badly needed for eye-tracking-based research. Brône et al. "argue for the integration of object recognition algorithms from vision engineering, such as invariant region matching techniques, in gaze analysis software" [10]. Later, De Beugher et al. followed this idea by implementing the

computer-vision-based software for the automatic detection of objects, faces, and persons in the recordings derived from mobile eye trackers [11].

MacInnes et al. propose a method "for mapping gaze data from an egocentric coordinate system (i. e. the wearable eye-tracker) to a fixed reference coordinate system (i. e. a target stimulus in the environment)" [6]. The method relies on the robust computer vision algorithms implemented in the OpenCV library [12], such as Scale-Invariant Feature Transform (SIFT) [13], Fast Library for Approximate Nearest Neighbors (FLANN) [14], and Random Sample Consensus (RANSAC) [15]. This method "allows researchers to study aggregate viewing behavior on a 2D planar target stimulus without restricting the mobility of participants" [6].

Wolf et al. introduce the computational gaze-object mapping method "that automatically maps gaze data onto respective AOIs" [16]. This method relies on the region-based convolutional neural network (R-CNN) that ensures precise object masking. Although this method is highly-functional, it only detects an entire object and cannot split it into finer details. Therefore, it cannot maintain hierarchical AOIs. To overcome this limitation, Batliner et al. improved this approach to support the mapping of the user's gaze on dynamic AOIs [17]. The aim of this research group was to study the usability of tangible screen-based user interfaces, like the ones in modern medical equipment. First, the AOI bounding box is detected in the video frame (in the mentioned work, just a single top-level AOI is considered) using state-of-the-art computer vision algorithms. Second, the gaze is mapped from the video coordinate system into the AOI coordinate system. Third, the extracted AOI is matched against the set of predefined templates to find out the appropriate one and to allow studying the mapped gaze fixations on the level of subordinate AOIs. In this way, a two-level hierarchy of AOIs is maintained. To keep track of the top-level AOI, Lucas-Kanade optical flow [18] is utilized. The main limitation of the approach of Batliner et al. is the computational burden requiring cloud computing to be involved.

Callemein et al. propose a tool to support the eye-tracking-based study of human communication [7]. Using the YOLOv2 real-time object detection model and the OpenPose detection library for the body, face, hands, and foot estimation, this research group implemented the software for robust automatic detection of human faces and hands in the mobile eye tracker videos.

Kurzhals et al. propose to create AOIs based on the gaze coordinates without any a priori information [19]. The idea is to extract small regions around each fixation point, then cluster these regions and define the result clusters as AOIs. To maintain this process, a special high-level graphical editor is created allowing the researchers to assess and correct the clustering results if necessary.

As stated by Hessels et al., despite many attempts to make the AOIs detection fully automatic, there is still no universal approach; each well-known solution has its certain limitations [20]. These limitations are due to the underlying computer vision techniques, which are multifarious but have no "silver bullet" to handle all possible use cases at once.

This is why we decided to choose a microservice-based approach for creating a modular AOIs detection pipeline, that allows to fine-tune, reuse, and replace the individual steps according to the specifics of a particular detection task.

4. Background

While collecting, preparing, and analyzing the data within our Digital Humanities (DH) research, we often rely on SciVi visual analytics platform [21]. This platform has a microservice architecture with microservices' behavior and communication driven by ontologies.

SciVi is a Web application; its server is written in Python using Flask and its client is based on JavaScript, HTML5, and CSS3. SciVi supports microservices, which run on the client (written in any browser-compatible programming language, including JavaScript, TypeScript, GLSL, and WebAssembly), on the server (usually written in Python or organized as pre-compiled binary libraries with Python-compatible interface), or on the special external hardware (for that case, appropriate firmware and middleware are generated automatically [22, 23]).

The involved ontology engineering principles enable a declarative way of describing the functionality, graphical user interfaces (GUIs), and communication protocols of individual microservices, which simplifies their extension and reuse. This in turn allows an efficient adaptation of the platform to solving the new classes of visual analytics tasks. SciVi provides four levels of this adaptation:

1. **System level.** This level is available for system programmers and assumes an extension of the built-in ontology reasoner to introduce fundamentally new features to the platform's core. Changes on this level are very rare. They are needed if SciVi is being adapted to solving problems far beyond visual analytics.
2. **Application level.** This level is available for application programmers who develop new microservices for solving new problems within the common SciVi paradigm.
3. **Knowledge base level.** This level is available for the SciVi administrators who can change the underlying ontologies to build so-called workbenches: sets of available microservices suitable for solving problems of a particular class (for example, problems related to DH [21], Smart Museum [24], Internet of Things [25], Human-Computer Interaction [26], etc.).
4. **Data flow level.** This level is available for end users who declare the pipelines of microservices for solving particular tasks in a given application domain. For this, the users utilize a high-level visual programming tool based on data flow diagrams (DFDs) that helps define the data processing pipeline in a form of an operators' chain. The involved microservices are represented as DFD nodes (data processing operators) and links denote data transfer. The composing of DFD requires no special IT hard skills from the user: SciVi automatically takes care of execution order, load balancing, and needed data transfer protocols, allowing the user to concentrate on the data processing semantics only.

In this work, we introduce new microservices on the application level (level 2) for solving a new class of tasks related to the automatic segmentation of video streams. After that, we enriched the SciVi knowledge base (level 3) to allow end users to utilize these microservices on the data flow level (level 4). To test the new microservices, we composed a DFD to solve a particular video segmentation task. Below, the newly introduced features are described in detail.

5. AOIs Detection Problem

The mobile eye gaze tracker is a glasses-like device with two cameras E_1 and E_2 facing the human's eyes (responsible for optical gaze tracking), and one field-of-view camera C facing forwards (responsible for recording nearly the same that the human sees). Two records are the tracker output: video stream V from the camera C and eye gaze track T derived from the cameras E_1 and E_2. T is an array of timestamped samples of eye gaze coordinates expressed in the plain 2D Cartesian coordinate system of V. It is implied that there is a known location $\{x_t, y_t\}$ of human gaze within the video stream frame $V(t)$ for each timestamp t in the eye-tracking-based experiment.

These data are enough to build a heatmap-like visualization for preliminary analysis of areas, which attract the informant's attention. However, for deeper analysis, meaningful segmentation of $V(t)$ into the AOIs is needed. Such segmentation allows revealing the patterns of information processing by performing the data mining of the eye gaze tracks.

It must be noted, that in our experiments, an informant looks at the static visual stimulus I (hereafter also denoted as "template image") while the gaze track is being recorded. For stationary trackers, where the informant's head is fixed opposite to I, the segmentation can be performed manually, because $V(t) = \text{const}$. For mobile trackers, each $V(t)$ is unique, because the informant can freely move their head while watching I, so automated segmentation is badly needed.

For this kind of automation, we suggest the following algorithm:

1. Perform a one-time segmentation S of I into the set of AOIs $A = \{a_1, ..., a_n\} : A = S(I)$. Each AOI $a_i, i = \overline{1,n}$ is represented as $a_i = \{l_i, p_i^1, p_i^2, ..., p_i^m\}$, where l_i is a text label of corresponding AOI and $p_i^j, j = \overline{1,m}$ are the points of AOI bounding polygon expressed in the plain 2D Cartesian coordinate system of I.
2. For each t:
   a) Locate I in $V(t)$ using a detector $D : I' = D(I, V(t))$.
   b) If the locating is successful:
      i. Calculate a homography transformation H such as $I' = H(I)$.
      ii. For each a_i from A:
         A. For each p_i^j from a_i: calculate $q_i^j = H(p_i^j)$.
         B. Compose a new set $a_i' = \{l_i, q_i^1, q_i^2, ..., q_i^m\}$.
      iii. Compose a new set $A' = \{a_1', ..., a_n'\}$.

As a result, for each $V(t)$ that contains I, a set of AOIs A' will be created, and the bounding polygon of each AOI a_i' from this set will be expressed in the coordinate system of $V(t)$. This in turn enables matching the eye gaze points $\{x_t, y_t\}$ against these AOIs during the subsequent analysis.

The main advantages of the above mentioned algorithm over segmenting each $V(t)$ individually are higher robustness, sustainability, and efficiency: the template image I has no distortions like perspective, barrel, pincushion, defocus, chromatic aberration, etc., which are typical for video footages. This is why the quality of automatic segmentation should be much higher for the template image than for the video frame for almost any segmentation algorithm.

Since step (1) of the above algorithm is performed just once, the initial segmentation S can potentially be a manual operation. But in our case we are about to retrieve the bounding boxes of individual words in the textual stimuli, therefore S is performed automatically using the optical character recognition (OCR) technique.

We implemented the above algorithm using the visual programming capabilities of the SciVi platform. SciVi provides a high-level GUI for fine-tuning the corresponding data preparation and detection steps. This allows the users without deep computer vision knowledge to adapt this algorithm for the new types of template images and new video streams.

6. AOIs Detection Pipeline in SciVi

The DFD pipeline for automatic detection of AOIs in a set of video streams is shown in Fig. 1. Each DFD node denotes a corresponding data processing operator that is implemented as an individual microservice.

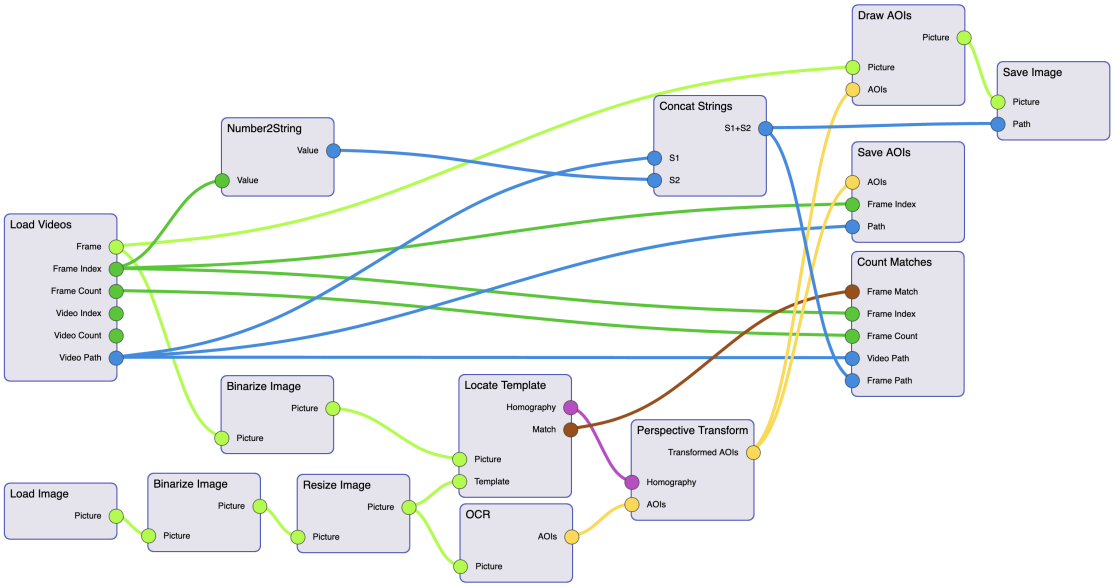

Figure 1: SciVi DFD of interest areas detection pipeline.

The pipeline execution starts with the “Load Videos” and “Load Image” operators. They both act as data sources. “Load Videos” is responsible for loading video streams. The corresponding microservice is written in Python and runs on the SciVi server side. It has corresponding settings (not shown in the DFD, because they appear in a special side panel in SciVi GUI), which allow the user to choose a set of video files to be loaded. The files should be located on the server because normally they are quite large (hundreds of Megabytes), so it is inefficient to transfer them over the network. The loading is leveraged by OpenCV that takes care of video decoding and splitting the stream into frames for subsequent analysis. “Load Videos” performs an implicit iteration over the array of videos, and for each video, over its frames. For each

iteration, the operator's outputs take the corresponding values: frame image ("Frame", denoted as $V(t)$ in Section 5), frame index in a stream ("Frame Index"), number of frames in the stream ("Frame Count"), index of the current video stream in the array ("Video Index"), length of this array ("Video Count"), and file name of the current video stream ("Video Path"). The subsequent operators are executed for each iteration taking these values as inputs. The outputs "Video Index" and "Video Count" could be used to maintain a progress bar of the entire operation, but in the present case, they are not connected to anything because the progress is estimated elsewhere (see the description of "Count Matches" operator below).

Parallel to the video loading, the template image is loaded using the "Load Image" server-side operator that relies on Python-based OpenCV API. The template is an image that should be located in the video stream (this image is denoted as I in Section 5). Normally it is the graphical or textual stimulus that has been used in the eye tracking experiment. In our case, it contains a text with particular formatting.

Then, both frame image and template image are binarized by the "Binarize Image" server-side operator that uses adaptive thresholding from OpenCV (https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html). Next, the template image is downsampled using the "Resize Image" server-side operator. The downsampling ratio is empirical and can be set via the operator's GUI. We revealed that this step dramatically affects the subsequent template location quality. The task is to make the template image as close as possible to the size of its representation in the video frame.

The downsampled template image is passed to two other server-side operators: "Locate Template" and "OCR". The "Locate Template" operator chains under the hood three algorithms implemented in OpenCV. First, it utilizes the SIFT detector [13] to find a set of features in the "Template" and "Picture" (video frame) images. Second, it uses FLANN [14] and Lowe's ratio test [13] to find the best match between these two sets. If there are more than 50 matching features, the template is considered to be located, and "True" is assigned to the "Match" Boolean output. Otherwise, "False" is assigned. Third, regardless of the matching features number, homography transformation is calculated using the RANSAC [15] algorithm. The homography matrix is assigned to the "Homography" output.

The "OCR" operator performs optical character recognition to find the bounding boxes of all the words in the text. This operator is leveraged by Tesseract open source OCR engine (https://github.com/tesseract-ocr/tesseract) using Long Short-Term Memory (LSTM) artificial neural network [27] to recognize the characters and language-specific dictionaries to recognize whole words. Although this method is quite robust, it can be improved if the default dictionary is substituted by the actual text being processed. This is possible in our case as the text is not random but represents a part of an actual eye-tracking-based experiment protocol. The result of the "OCR" operator is a set of AOIs, each one having a label (the recognized word) and four points of bounding box corners.

The next pipeline step is the "Perspective Transform" server-side operator. It takes a homography matrix and a set of AOIs detected in the template image and transforms the bound boxes from the template image space to the video frame space. To solve this, a built-in OpenCV perspective transform function is used.

According to the main algorithm, "Perspective Transform" is the last step. Its result is transmitted to the "Save AOIs" server-side operator that stores the transformed AOIs in a CSV

file. Each entry has the following format:
`Frame Index, "Word", X1, Y1, X2, Y2, X3, Y3, X4, Y4`
where $X_i, Y_i, i = \overline{1,4}$ are the bounding box corners coordinates enumerated in counter-clockwise order, expressed in the video frame coordinate system.

The file name corresponds to the name of the processed video stream (obtained through the "Path" input). For each frame, the number of entries generated is equal to the number of words in the text from the template image.

The above-mentioned operators solve the AOIs detection problem. However, to monitor the detection process, additional operators are introduced. To store the visual representation of AOIs, the "Draw AOIs" server-side operator draws bounding polygons of AOIs in the video frame using OpenCV image manipulation API. The resulting image is passed to the "Save Image" server-side operator that stores it in a file with a name specified in the "Path" input. This input takes concatenation ("Concat Strings" server-side operator) of the video path and video frame index (converted to a string using the "Number2String" server-side operator).

Last but not least, the "Count Matches" client-side operator allows the user to monitor the overall progress of the pipeline execution and assess the detection quality. This is the only operator in this DFD that runs in the browser. For each video stream, a labeled sequence chart is drawn in real time (see Fig. 2). Green bars indicate the frames, for which the "Frame Match" value is "True", and red bars stand for "False". This allows the user to see if there are any template location failures in the video stream. Clicking in the sequence chart opens up the pop-up with the loaded frame image that is requested from the server by the "Frame Path" cached in the "Count Matches" operator for each frame. Using this feature, the user can visually assess the AOIs detection quality for any place in any processed video stream.

Figure 2: Sequence chart rendered in SciVi displaying the successful and unsuccessful attempts to locate template image in video streams (processing of the videos "Informant-1-1" and "Informant-1-2" are completed; processing of the video "Informant-1-3" is in progress).

7. Discussion

As for the time of writing the paper, we tested the above pipeline on 3 different text-based visual stimuli, each one with 3 different video streams. The OCR worked correctly for all the template images used (this was expected because the template images have no distortions and are in fact perfect text representations). The template location accuracy is nearly 90%, which is a fairly good result. It must be noted, that the locating failures at the beginning of the videos (see Fig. 2) are expected and correct: the text was presented to the informant a couple of seconds after the start of the recording. The individual failures in the middle are not a big problem: the

single-frame gaps can be filled by interpolating the AOIs bounding polygons in the neighboring frames (thus, such post-processing is a matter of future work).

The example of the template image is shown in Fig. 3 (left). The AOIs detection result for a single frame (as saved by the "Save Image" DFD operator) is shown in Fig. 3 (middle). The SIFT/FLANN-based template location for text-containing images appears to be very robust because the graphical representation of a text has many salient features. The corner case is shown in Fig. 3 (right). As it can be seen, the detection still works correctly even if the informant bows their head pretty much.

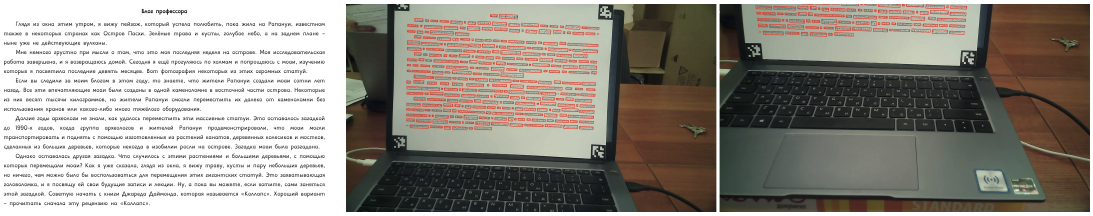

Figure 3: Template image of one of the experimental texts (left), the result of AOIs detection in SciVi (middle), The result of AOIs detection in SciVi for the corner case of almost fully obscured template (right).

8. Conclusion

In this work, we propose a pipeline for automatic detection of AOIs in the video streams derived from the field-of-view cameras of mobile eye trackers. This pipeline is based on the SciVi ontology-driven microservice-based platform that provides high-level tools to maintain its modularity and extensibility.

Currently, we adopted tools to work with text-based visual stimuli, detecting the given text (top-level AOI) in the video streams and segmenting individual words (subordinate AOIs). The proposed AOIs detection pipeline proved its efficiency on the real data derived from the mobile eye tracker PupilCore. Although currently it has been used with text-based stimuli only, thanks to the modularity it can be extended to other types of stimuli as well by introducing new segmentation algorithms instead of OCR.

For the future work, we plan to mitigate the mentioned limitations by adopting other template localization techniques and implementing AOIs inter-frame interpolation.

Acknowledgments

The study has been conducted within the research project "Text processing in L1 and L2: Experimental study with eye-tracking, visual analytics and virtual reality technologies" supported by the research grant No. ID92566385 from Saint Petersburg State University.

References

[1] K. Rayner, The 35th Sir Frederick Bartlett Lecture: Eye Movements and Attention in Reading, Scene Perception, and Visual Search, Quarterly Journal of Experimental Psychology 62 (2009) 1457–1506. doi:10.1080/17470210902816461.

[2] E. Matin, Saccadic Suppression: A Review and an Analysis, Psychological Bulletin 81 (1974) 899–917. doi:10.1037/h0037368.

[3] M. A. Just, P. A. Carpenter, A Theory of Reading: From Eye Fixations to Comprehension, Psychological Review 87 (1980) 329–354. doi:10.1037/0033-295X.87.4.329.

[4] K. Rayner, Eye Movements in Reading and Information Processing: 20 Years of Research, Psychological Bulletin 124 (1998) 372–422. doi:10.1037/0033-2909.124.3.372.

[5] T. E. Petrova, E. I. Riekhakaynen, V. S. Bratash, An Eye-Tracking Study of Sketch Processing: Evidence From Russian, Frontiers in Psychology 11 (2020). doi:10.3389/fpsyg.2020.00297.

[6] J. J. Macinnes, S. Iqbal, J. Pearson, E. N. Johnson, Wearable Eye-Tracking for Research: Automated Dynamic Gaze Mapping and Accuracy/Precision Comparisons Across Devices, bioRxiv (2018). doi:10.1101/299925.

[7] T. Callemein, K. Van Beeck, G. Brône, T. Goedemé, Automated analysis of eye-tracker-based human-human interaction studies, Lecture Notes in Electrical Engineering 514 (2019) 499–509. doi:10.1007/978-981-13-1056-0_50.

[8] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, M. J. Marín-Jiménez, Automatic Generation and Detection of Highly Reliable Fiducial Markers under Occlusion, Pattern Recognition 47 (2014) 2280–2292. doi:10.1016/j.patcog.2014.01.005.

[9] S. Alexeeva, A. Dobrego, V. Zubov, Towards the first dyslexic font in russian, in: Workshop on Linguistic and Neurocognitive Resources (LiNCr2020), 2020, pp. 9–14.

[10] G. Brône, B. Oben, T. Goedemé, Towards a More Effective Method for Analyzing Mobile Eye-Tracking Data: Integrating Gaze Data with Object Recognition Algorithms, in: Proceedings of the 1st International Workshop on Pervasive Eye Tracking & Mobile Eye-Based Interaction, 2011, pp. 53–56. doi:10.1145/2029956.2029971.

[11] S. De Beugher, G. Brône, T. Goedemé, Automatic Analysis of In-The-Wild Mobile Eye-Tracking Experiments Using Object, Face and Person Detection, in: 2014 International Conference on Computer Vision Theory and Applications (VISAPP), 2014, pp. 625–633.

[12] G. Bradski, A. Kaehler, Learning OpenCV: Computer Vision with the OpenCV Library, O'Reilly Media, Inc., 2008.

[13] D. G. Lowe, Distinctive Image Features from Scale-Invariant Keypoints, International Journal of Computer Vision 60 (2004) 91–110. doi:10.1023/B:VISI.0000029664.99615.94.

[14] M. Muja, D. G. Lowe, Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration, in: VISAPP 2009 - Proceedings of the Fourth International Conference on Computer Vision Theory and Applications, 2009.

[15] M. A. Fischler, R. C. Bolles, Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography, Communications of the ACM 24 (1981) 381–395. doi:10.1145/358669.358692.

[16] J. Wolf, S. Hess, D. Bachmann, Q. Lohmeyer, M. Meboldt, Automating Areas of Interest

Analysis in Mobile Eye Tracking Experiments Based on Machine Learning, Journal of Eye Movement Research 11 (2018). doi:doi.org/10.16910/jemr.11.6.6.

[17] M. Batliner, S. Hess, C. Ehrlich-Adám, Q. Lohmeyer, M. Meboldt, Automated Areas of Interest Analysis for Usability Studies of Tangible Screen-Based User Interfaces Using Mobile Eye Tracking, Artificial Intelligence for Engineering Design, Analysis and Manufacturing 34 (2020) 505–514. doi:10.1017/S0890060420000372.

[18] B. D. Lucas, T. Kanade, An Iterative Image Registration Technique with an Application to Stereo Vision, in: IJCAI'81: Proceedings of the 7th international joint conference on Artificial intelligence, volume 2, 1981, pp. 674–679.

[19] K. Kurzhals, M. Hlawatsch, C. Seeger, D. Weiskopf, Visual Analytics for Mobile Eye Tracking, IEEE Transactions on Visualization and Computer Graphics 23 (2017) 301–310. doi:10.1109/TVCG.2016.2598695.

[20] R. S. Hessels, J. S. Benjamins, T. H. W. Cornelissen, I. T. C. Hooge, A Validation of Automatically-Generated Areas-of-Interest in Videos of a Face for Eye-Tracking Research, Frontiers in Psychology 9 (2018). doi:10.3389/fpsyg.2018.01367.

[21] K. Ryabinin, K. Belousov, S. Chuprina, Novel Circular Graph Capabilities for Comprehensive Visual Analytics of Interconnected Data in Digital Humanities, Scientific Visualization 12 (2020) 56–70. doi:10.26583/sv.12.4.06.

[22] K. Ryabinin, S. Chuprina, Ontology-Driven Edge Computing, Lecture Notes in Computer Science 12143 (2020) 312–325. doi:10.1007/978-3-030-50436-6_23.

[23] K. Ryabinin, S. Chuprina, I. Labutin, Tackling IoT Interoperability Problems with Ontology-Driven Smart Approach, Lecture Notes in Networks and Systems 342 (2021) 77–91. doi:10.1007/978-3-030-89477-1_9.

[24] K. Ryabinin, M. Kolesnik, Adaptive Scientific Visualization Tools for a Smart Paleontological Museum, Programming and Computer Software 45 (2019) 180–186. doi:10.1134/S0361768819040066.

[25] K. Ryabinin, S. Chuprina, M. Kolesnik, Calibration and Monitoring of IoT Devices by Means of Embedded Scientific Visualization Tools, Lecture Notes in Computer Science 10861 (2018) 655–668. doi:10.1007/978-3-319-93701-4_52.

[26] K. Ryabinin, S. Chuprina, K. Belousov, Ontology-Driven Automation of IoT-Based Human-Machine Interfaces Development, Lecture Notes in Computer Science 11540 (2019) 110–124. doi:10.1007/978-3-030-22750-0_9.

[27] S. Hochreiter, J. Schmidhuber, Long Short-Term Memory, Neural Computation 9 (1997) 1735–1780. doi:10.1162/neco.1997.9.8.1735.