Metadata-Driven Eye Tracking for Real-Time Applications

Yasith Jayawardana
yasith@cs.odu.edu
Dept. of Computer Science
Old Dominion University
Norfolk, VA, USA

Gavindya Jayawardena
gavindya@cs.odu.edu
Dept. of Computer Science
Old Dominion University
Norfolk, VA, USA

Andrew T. Duchowski
duchowski@clemson.edu
School of Computing
Clemson University
Clemson, SC, USA

Sampath Jayarathna
sampath@cs.odu.edu
Dept. of Computer Science
Old Dominion University
Norfolk, VA, USA

ABSTRACT

When conducting eye tracking studies, having a mechanism to collect data, build workflows, and validate results in a FAIR (i.e., *findable*, *accessible*, *interoperable*, and *reusable*) manner, facilitates automation. Given the vast landscape of vendor-specific eye tracking software, adopting FAIR metadata standards for the eye tracking domain is one step towards this. In this paper, we propose an approach to simplify the creation, execution, and validation of eye tracking studies through metadata. Using a metadata format that we developed, we first describe two eye trackers, and two datasets collected using them. Next, we use this metadata to simulate real-time data collection by replaying each dataset. From this replayed data, we analyze eye movements in real-time, and synthesize eye movement data from analytics in real-time. Based on our results, we discuss the utility of metadata in real-time eye tracking studies, and how this idea can be generalized into other applications.

CCS CONCEPTS

• **Applied computing → Document metadata**; • **Information systems → Data streaming**; *Temporal data.*

KEYWORDS

Metadata, Eye Tracking, Stream Processing, Replay, Simulation

ACM Reference Format:
Yasith Jayawardana, Gavindya Jayawardena, Andrew T. Duchowski, and Sampath Jayarathna. 2021. Metadata-Driven Eye Tracking for Real-Time Applications. In *ACM Symposium on Document Engineering 2021 (DocEng '21), August 24–27, 2021, Limerick, Ireland.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3469096.3474935

1 INTRODUCTION

Eye tracking is the process of measuring where a person is looking at over time, and thereby capturing the sequence in which their gaze shifts from one position to another [14]. An eye tracker is a device that samples one's gaze position over time, and optionally, their pupil diameter. Both measurements are known indicators of one's visual attention [10]. In eye movement analysis, eye tracking data is typically grouped into sequences of higher-order events such as *fixations* (i.e., the centering of gaze around a certain region) and *saccades* (i.e., a rapid shift of gaze from one region to another). These events are akin to how we perceive the visual world; our gaze is directed onto objects through rapid eyeball movements (saccade), and remains on them (fixation) until they are perceived.

Modern eye trackers come in variants such as screen-based, wearable, and webcam-based. Devices such as *Tobii Pro Fusion* and *EyeLink 1000 Plus* are screen-based, while devices such as *Tobii Pro Glasses*, *Pupil Core*, *EyeLink II*, and *HTC Vive Pro Eye* are wearable. On the other hand, software such as *WebGazer*, *GazeRecorder*, and *XLabsGaze* are webcam-based. While eye tracker manufacturers provide software suites such as *Tobii Pro Lab*, *EyeLink Data Viewer*, and *Pupil*, they are designed to work exclusively on specific devices, and uses non-standard file formats such as *EDF* (SR Research) and *PLOF* (Tobii) by default. As a solution to this vendor lock-in, third-party research software such as *iMotions* support multiple types of eye trackers and biosensors such as EEG and fMRI. These softwares, however, are proprietary, which makes it challenging to share/reuse applications built using them. **Hence, eye tracking data and workflows would benefit from adapting standard file formats such as CSV/JSON, with FAIR [17] metadata.**

Data-intensive research, such as eye movement analysis, typically involve developing, executing, and validating a series of data manipulation and visualization steps (i.e., a workflow) that lead to a desired outcome. A scientific workflow (SWF) system is a form of a workflow management system designed for this specific need [13]. They enable users to manage the execution of complex data computations and parameter-driven simulations, and simplifies the process of testing, refining, and comparing experimental setups [6]. While a plethora of SWF systems exist [6, 13], SWF systems such as *KNIME*, *Kepler*, and *Node-RED* allow users to build workflows through visual programming [1]. This appeals to non-programmers, such as scientists with no programming background, as it abstracts away the complexity of underlying programming [12]. Moreover, SWF systems such as *Node-RED* allow users to build stream-based workflows for real-time applications. **With eye trackers increasingly being used in real-time applications such as gaming, adapting a stream-based visual programming SWF system such as Node-RED would expedite the building of real-time eye movement analysis workflows for a larger population of users.**

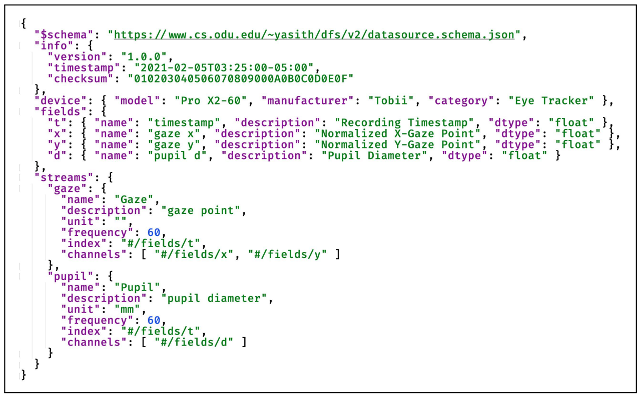
In this paper, we describe eye trackers and eye movement data using FAIR metadata, and attempt to build reusable, reproducible, and verifiable stream-processing workflows using them. Based on our Dataset File System (*DFS*) [7, 8] metadata format, we first generate metadata for two eye trackers, and two eye tracking datasets collected from them. Next, we use this metadata to replay the datasets akin to real-time data collection. Following this, we stream the replayed eye movement data into a workflow built visually using

DocEng '21, August 24–27, 2021, Limerick, Ireland
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8596-1/21/08…$15.00
https://doi.org/10.1145/3469096.3474935

Node-RED, and into a workflow built natively on *Python*/*OpenGL*. Using these workflows, we perform real-time eye movement analysis, synthesize eye movement data in real-time, and verify the consistency of their outputs. Based on our results, we discuss the utility of this approach for real-time eye tracking applications, and how it could be generalized into other types of temporal data.

2 EXPERIMENTAL SETUP

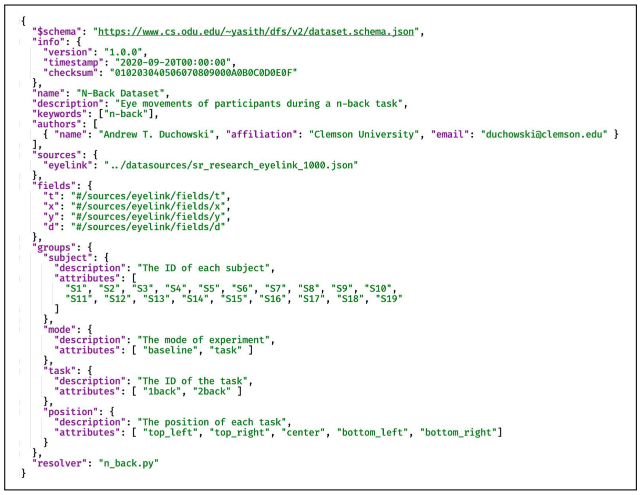
Data: We use two of our pre-collected datasets: *ADHD-SIN* [9] and *N-BACK* [4], each providing the gaze position and pupil diameter of subjects engaging in cognitive tasks. The *N-BACK* dataset was acquired using a *SR Research EyeLink 1000* eye tracker (1000 Hz), while the *ADHD-SIN* dataset was acquired using a *Tobii Pro X2-60* eye tracker (60 Hz). We pre-process each dataset to provide normalized gaze position (x,y) and pupil diameter (d) readings over time (t). Any missing values were filled via linear interpolation, backward-fill, and forward-fill, in order.



```
{
  "$schema": "https://www.cs.odu.edu/~yasith/dfs/v2/datasource.schema.json",
  "info": {
    "version": "1.0.0",
    "timestamp": "2021-02-05T03:25:00-05:00",
    "checksum": "0102030405060708090000A0B0C0D0E0F"
  },
  "device": { "model": "Pro X2-60", "manufacturer": "Tobii", "category": "Eye Tracker" },
  "fields": {
    "t": { "name": "timestamp", "description": "Recording Timestamp", "dtype": "float" },
    "x": { "name": "gaze x", "description": "Normalized X-Gaze Point", "dtype": "float" },
    "y": { "name": "gaze y", "description": "Normalized Y-Gaze Point", "dtype": "float" },
    "d": { "name": "pupil d", "description": "Pupil Diameter", "dtype": "float" }
  },
  "streams": {
    "gaze": {
      "name": "Gaze",
      "description": "gaze point",
      "unit": "",
      "frequency": 60,
      "index": "#/fields/t",
      "channels": [ "#/fields/x", "#/fields/y" ]
    },
    "pupil": {
      "name": "Pupil",
      "description": "pupil diameter",
      "unit": "mm",
      "frequency": 60,
      "index": "#/fields/t",
      "channels": [ "#/fields/d" ]
    }
  }
}
```

Figure 1: DFS Metadata for the *Tobii Pro X2-60* Datasource

Metadata: We first create metadata for the two eye tracking devices, using the *DFS data-source*[1] schema (see Figure 1). It provides attributes to describe data streams, such as frequency and channels.



```
{
  "$schema": "https://www.cs.odu.edu/~yasith/dfs/v2/dataset.schema.json",
  "info": {
    "version": "1.0.0",
    "timestamp": "2020-09-20T00:00:00",
    "checksum": "0102030405060708090000A0B0C0D0E0F"
  },
  "name": "N-Back Dataset",
  "description": "Eye movements of participants during a n-back task",
  "keywords": ["n-back"],
  "authors": [
    { "name": "Andrew T. Duchowski", "affiliation": "Clemson University", "email": "duchowski@clemson.edu" }
  ],
  "sources": {
    "eyelink": "../datasources/sr_research_eyelink_1000.json"
  },
  "fields": {
    "t": "#/sources/eyelink/fields/t",
    "x": "#/sources/eyelink/fields/x",
    "y": "#/sources/eyelink/fields/y",
    "d": "#/sources/eyelink/fields/d"
  },
  "groups": {
    "subject": {
      "description": "The ID of each subject",
      "attributes": [
        "S1", "S2", "S3", "S4", "S5", "S6", "S7", "S8", "S9", "S10",
        "S11", "S12", "S13", "S14", "S15", "S16", "S17", "S18", "S19"
      ]
    },
    "mode": {
      "description": "The mode of experiment",
      "attributes": [ "baseline", "task" ]
    },
    "task": {
      "description": "The ID of the task",
      "attributes": [ "1back", "2back" ]
    },
    "position": {
      "description": "The position of each task",
      "attributes": [ "top_left", "top_right", "center", "bottom_left", "bottom_right"]
    }
  },
  "resolver": "n_back.py"
}
```

Figure 2: DFS Metadata for the *N-BACK* Dataset

[1]https://github.com/nirdslab/streaminghub/tree/master/dfs/datasource.schema.json

Next, we create metadata for the two datasets, using the *DFS data-set*[2] schema (see Figure 2). It provides attributes to describe the ownership, identification, provenance, and groups (i.e., viewpoints) of a dataset, and to reference a *resolver* script that maps stream-wise and/or group-wise queries into data. In this study, both datasets were grouped *subject*-wise and *task*-wise to facilitate such queries.

Replay: Here, we implement *resolver* scripts (in Python) for each dataset, and invoke them to replay data. When a resolver is invoked, *outlets* (i.e., data streams) are created for the requested streams and groups of the dataset, and they transition into an awaiting state. Once an *inlet* (i.e., subscriber) connects, the outlet transitions into a streaming state, begins to replay the data, and continues doing so until the inlet disconnects, or until all data has been replayed.

3 METHODOLOGY

Next, we use the aforementioned metadata and resolvers to replay data akin to real-time data collection, and test them via four tasks.

(1) Detect fixations and saccades from gaze streams
(2) Synthesize gaze data from fixation and saccade analytics
(3) Chain a workflow to run on *synthetic* gaze streams
(4) Calculate a pupillary measure (LHIPA [4]) from pupil streams

In Task 1, we detect fixations and saccades from replayed gaze streams. In Task 2, we generate synthetic gaze streams from real-time fixation and saccade analytics. In Task 3, we feed the synthetic gaze streams as input to another workflow running in parallel. In Task 4, we calculate LHIPA from replayed pupil streams. The goal of these tasks is to test the FAIR-ness of our data replay setup.

Task 1: Detect Fixations and Saccades

Here, we use *StreamingHub*[3] to build and execute eye movement analysis workflows (see Figure 3). *StreamingHub* is a framework that we developed upon DFS and Node-RED to build stream-processing workflows using visual programming. It has four components:

(1) **Dataset File System (DFS)**[4] – Collection of JSON schemas for describing data-sources, data-analytics, and data-sets.
(2) **Data Mux** – Bridge between connected sensors, datasets, and data streams, which leverages DFS to automate data streaming. Implemented using *LabStreamingLayer* [11] and *WebSockets*.
(3) **Workflow Designer** – Front-end based on *Node-RED* to design stream-processing workflows using visual programming.
(4) **Operations Dashboard** – Web interface based on *Node-RED* to monitor workflows, generate interactive visualizations, and control data streams via search, subscribe, and seek operations.

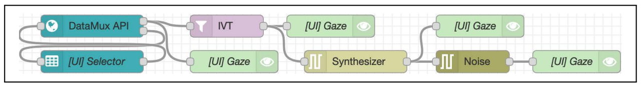


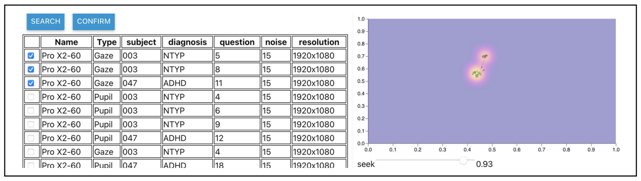
Figure 3: Eye Movement Analysis Workflow

Figure 3 illustrates an eye movement analysis workflow designed on StreamingHub. Here, the *IVT* node classifies data points as fixations or saccades using the IVT algorithm [15]. The *Synthesizer* node

[2]https://github.com/nirdslab/streaminghub/tree/master/dfs/dataset.schema.json
[3]https://github.com/nirdslab/streaminghub
[4]DFS Version 2.0 (https://github.com/nirdslab/streaminghub/tree/master/dfs)

generates synthetic gaze streams from fixation data (see Task 3). The *Noise* node adds Gaussian noise into data. The *[UI] Gaze* node uses Vega [16] to display gaze points in a scatter plot, connect consecutive gaze points with lines, overlay a heat map of the distribution of gaze, and provide a seek bar to navigate through time.

Next, we deploy this workflow and use its operations dashboard (See Figure 4) to replay and visualize data. Here, we first search for available streams, and select three of them for replay. Once this selection is confirmed, the workflow subscribes to the selected streams and begins to receive their data. Whenever a new sample is received, the interactive visualization is updated with that sample.



	Name	Type	subject	diagnosis	question	noise	resolution
✓	Pro X2-60	Gaze	003	NTYP	5	15	1920x1080
✓	Pro X2-60	Gaze	003	NTYP	8	15	1920x1080
✓	Pro X2-60	Gaze	047	ADHD	11	15	1920x1080
	Pro X2-60	Pupil	003	NTYP	4	15	1920x1080
	Pro X2-60	Pupil	003	NTYP	6	15	1920x1080
	Pro X2-60	Pupil	003	NTYP	9	15	1920x1080
	Pro X2-60	Pupil	047	ADHD	12	15	1920x1080
	Pro X2-60	Gaze	003	NTYP	4	15	1920x1080
	Pro X2-60	Pupil	047	ADHD	18	15	1920x1080

Figure 4: Operations Dashboard of Workflow: Available Streams (left), Selected Streams (ticks), and Visualization (right).

Task 2: Synthesize Gaze Data

The idea of eye movement synthesis is to control and direct a simulated point of gaze towards pre-defined areas of interest [5]. It is particularly useful for generating test cases to evaluate eye movement workflows (e.g., by synthesizing data for specific viewing patterns), and to evaluate the robustness of workflows (e.g., by synthesizing data with varying levels of recording noise). For this task, we select three parametric models of micro-saccadic jitter [2, 3] that we previously developed to synthesize gaze data from fixations, and adapt them for real-time eye movement synthesis.

- **Pink** Noise ($1/f^{\alpha}$) – [$G = 0.85, \alpha = 0.4, f = f_s$]
- **White** Noise (w/Butterworth LPF) – [$d = 4, f = f_s, l = 1.5915$]
- **Zero** Noise (i.e., noise-free)

Here, G is the gain, α is the exponent, f is the noise frequency, d is the degree, l is the cutoff frequency, and f_s is the eye tracker sampling frequency. We refer to them as **IDEAL** streams. Next, we duplicate the IDEAL streams and add 2D Gaussian noise with $\mu = (0, 0)$ and $\sigma \approx (0.00002, 0.00004)$ into them, to mimic recording noise. We refer to them as **NOISY** streams. All parameter values were taken from the original work [2, 3]. Thus, for our *three* models, we synthesize *two* data streams each, totaling *six* data streams.

Algorithm 1 Eye Movement Synthesis Workflow

```
 1: model $pink, $white, $zero
 2: istream $ivt
 3: ostream $pI, $wI, $zI, $pN, $wN, $zN
 4: async while (ivt) stream ⟶ $v
 5:     synthesize ($pink, $v) ⟶ (pI)
 6:     synthesize ($white, $v) ⟶ (wI)
 7:     synthesize ($zero, $v) ⟶ (zI)
 8: end while
 9: async while (pI) stream ⟶ $v
10:     noise ($v) ⟶ (pN)
11: end while
12: async while (wI) stream ⟶ $v
13:     noise ($v) ⟶ (wN)
14: end while
15: async while (zI) stream ⟶ $v
16:     noise ($v) ⟶ (zN)
17: end while
```

Task 3: Chaining Workflows

In this task, we test whether the output from one workflow can be consumed by another workflow, to effectively build chains of workflows. First, we create a replica of the original workflow used in the previous tasks, and run it in parallel. Next, we use its operations dashboard to subscribe to gaze streams, with one difference: instead of subscribing to the replayed gaze streams (as in the original workflow), we subscribe to the synthetic gaze streams generated by the original workflow. By doing so, we check whether the outputs of a workflow (in our case, the synthetic gaze streams) can be discovered using the same mechanism as discovering replayed gaze streams.

Task 4: Calculate Real-Time LHIPA

Here, we tested whether our data replay setup can be used outside of SWF systems for real-time analysis. We first developed an eye movement analysis workflow (using *Python*/*OpenGL*) to calculate the Low/High Index of Pupillary Activity (LHIPA) [4] from pupil streams in real-time. It estimates cognitive load from the frequency of pupil diameter oscillation. Next we implemented a stream listener (using *Python*/*PyLSL*[5]) to discover the *outlets* (in our case, the pupil streams) currently available on the network and spawn *inlets* (i.e., subscribers) to receive incoming data and metadata. The stream listener repeatedly polls the outlets from the inlets with a timeout of 0 s, to obtain chunks of data available for immediate pickup. If/when the data stream is lost, a *LostError* is raised by PyLSL, upon which the the stream listener is closed.

Following this, we reused the metadata created earlier for the *SR Research EyeLink 1000* eye tracker and the N-BACK dataset, along with the resolver implemented earlier for the N-BACK dataset, to spawn outlets that replay pupil diameter readings of each subject during each task. As specified in the eye tracker metadata, these outlets replay pupil diameter readings at a frequency of 1000 Hz. Then we subscribed to the replayed pupil streams through the stream listener. Any incoming data chunks were passed into the eye movement analysis workflow to calculate the real-time LHIPA, and the processed pupil diameter signal and the threshold used to calculate LHIPA were visualized via OpenGL (see Figure 5).

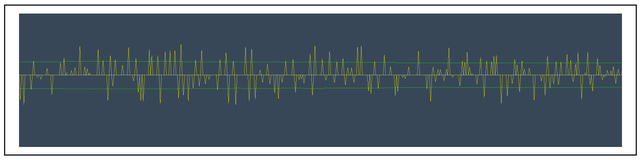


Figure 5: OpenGL plot of the processed pupil diameter signal (yellow) and the threshold used to calculate LHIPA (green).

4 EVALUATION

Using RMSE, we compare the six synthetic gaze streams with original data, both globally (i.e., all data) and group-wise (i.e., data within a group). The goal here is to check the similarity of their data at different levels. Table 1 reports the global RMSE between synthetic and original data. Both datasets exhibited a similar RMSE for *IDEAL* and *NOISY* streams across all models. This can be attributed to the low standard deviation (σ) used to generate the Gaussian noise.

[5]https://pypi.org/project/pylsl/1.10.4/

Overall, the RMSE was lower for *N-BACK* than *ADHD-SIN*. This may be due to different sampling frequencies that they were collected/replayed at (N-BACK: 1000 Hz, ADHD-SIN: 60 Hz). For both datasets, the RMSE increased as *Zero* < *White* < *Pink*.

Table 1: Global RMSE: Synthetic vs Original Data

MDL	N-BACK		ADHD-SIN	
	IDEAL	NOISY	IDEAL	NOISY
Pink	0.062640	0.062640	0.115373	0.115373
White	0.062596	0.062596	0.115244	0.115244
Zero	0.062593	0.062593	0.115226	0.115226

Table 2 reports the group-wise RMSE between synthetic and original data. The *N-BACK* dataset was evaluated on two *MODE* items and two *TASK* items, while the *ADHD-SIN* dataset was evaluated on four *NOISE* items. Here too, both datasets exhibited similar RMSE for *IDEAL* and *NOISY* streams, with a lower RMSE overall for *N-BACK* than *ADHD-SIN*. For the N-BACK dataset, the RMSE increased as *Zero* < *White* < *Pink*. For the ADHD-SIN dataset, however, this order was inconsistent between noise levels. By visually inspecting gaze data from each stream, we found that the naturalness of simulation increases as *Zero* < *White* < *Pink*, which was consistent with the original work [2, 3].

Table 2: Group-wise RMSE: Synthetic vs Original Data

GRP	ITEM	MDL	RMSE		SAMPLES
			IDEAL	NOISY	
N-BACK (MODE)	Baseline	Pink	0.072336	0.072336	5,582,188
		White	0.072283	0.072283	5,582,170
		Zero	0.072277	0.072277	5,582,160
	Task	Pink	0.059648	0.059648	19,965,068
		White	0.059607	0.059607	19,965,082
		Zero	0.059604	0.059604	19,965,075
N-BACK (TASK)	1-back	Pink	0.063064	0.063064	12,581,319
		White	0.063023	0.063023	12,581,289
		Zero	0.063018	0.063018	12,581,298
	2-back	Pink	0.062225	0.062225	12,965,937
		White	0.062179	0.062180	12,965,963
		Zero	0.062177	0.062177	12,965,937
ADHD-SIN (NOISE)	0dB	Pink	0.102673	0.102673	50,052
		White	0.102225	0.102225	50,051
		Zero	0.102297	0.102297	50,051
	5dB	Pink	0.117201	0.117201	41,969
		White	0.117311	0.117311	41,970
		Zero	0.117228	0.117228	41,970
	10dB	Pink	0.112958	0.112958	40,240
		White	0.112778	0.112778	40,241
		Zero	0.112737	0.112737	40,242
	15dB	Pink	0.121464	0.121463	33,555
		White	0.121521	0.121521	33,555
		Zero	0.121467	0.121467	33,555

5 DISCUSSION

One foreseeable application of metadata-driven data replay and synthesis is to test experimental setups, both pre and post-collection. For pre-collection, one could simply stream in either random data or synthesized test data to verify that workflows run as expected. For post-collection, one could replay data through a workflow and verify that experimental results match. Overall, this provides an ecosystem to build robust, error-tolerant, and most importantly, reproducible real-time applications. Moreover, one could synthesize data from analytics akin to data augmentation.

In this study, our primary focus was on conceptual design and evaluation, and not on performance. In the future, we plan to integrate workflow engines like *Flink* or *Kinesis* to improve scalability and performance.

6 CONCLUSION

Using two eye tracking datasets, we demonstrated how FAIR metadata can simplify the creation, execution, and validation of eye tracking studies for real-time applications. From our experimental setup, we were able to a) replay datasets as data streams, b) generate synthetic gaze streams from replayed data, c) stream data into eye movement analysis workflows, and d) perform exploratory data analysis through stream control and visualization. We tested our setup on eye movement analysis workflows running on different technologies, both in serial and in parallel, and demonstrated that it works as expected. When comparing the synthetic gaze streams to the replayed data streams, we observed a low RMSE in data from the 1000 Hz eye tracker than the 60 Hz eye tracker.

ACKNOWLEDGEMENTS

This work was supported in part by NSF CAREER-2045523. Any opinions, findings and conclusions or recommendations expressed in this material are the author(s) and do not necessarily reflect those of the sponsors.

REFERENCES

[1] Margaret M Burnett and David W McIntyre. 1995. Visual programming. *COMPUTER-LOS ALAMITOS-* 28 (1995), 14–14.

[2] Andrew Duchowski, Sophie Jörg, Aubrey Lawson, Takumi Bolte, Lech Świrski, and Krzysztof Krejtz. 2015. Eye movement synthesis with 1/f pink noise. In *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games*. 47–56.

[3] Andrew T Duchowski, Sophie Jörg, Tyler N Allen, Ioannis Giannopoulos, and Krzysztof Krejtz. 2016. Eye movement synthesis. In *Proceedings of the ninth biennial ACM symposium on eye tracking research & applications*. 147–154.

[4] Andrew T Duchowski, Krzysztof Krejtz, Nina A Gehrer, Tanya Bafna, and Per Bækgaard. 2020. The Low/High Index of Pupillary Activity. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–12.

[5] MFP Gillies and Neil A Dodgson. 2002. Eye movements and attention for behavioural animation. *The Journal of Visualization and Computer Animation* 13, 5 (2002), 287–300.

[6] Carole Goble, Sarah Cohen-Boulakia, Stian Soiland-Reyes, Daniel Garijo, Yolanda Gil, Michael R Crusoe, Kristian Peters, and Daniel Schober. 2020. FAIR computational workflows. *Data Intelligence* 2, 1-2 (2020), 108–121.

[7] Yasith Jayawardana and Sampath Jayarathna. 2019. DFS: a dataset file system for data discovering users. In *2019 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*. IEEE, 355–356.

[8] Yasith Jayawardana and Sampath Jayarathna. 2020. Streaming Analytics and Workflow Automation for DFS. In *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries in 2020*. 513–514.

[9] Gavindya Jayawardena, Anne Michalek, Andrew Duchowski, and Sampath Jayarathna. 2020. Pilot study of audiovisual speech-in-noise (sin) performance of young adults with adhd. In *ACM Symposium on Eye Tracking Research and Applications*. 1–5.

[10] Siddhartha Joshi, Yin Li, Rishi M Kalwani, and Joshua I Gold. 2016. Relationships between pupil diameter and neuronal activity in the locus coeruleus, colliculi, and cingulate cortex. *Neuron* 89, 1 (2016), 221–234.

[11] C. Kothe. 2014. Lab streaming layer (LSL). *https://github.com/sccn/labstreaminglayer* 26 (2014), 2015.

[12] Mohammad Amin Kuhail, Shahbano Farooq, Rawad Hammad, and Mohammed Bahja. 2021. Characterizing Visual Programming Approaches for End-User Developers: A Systematic Review. *IEEE Access* (2021).

[13] Chee Sun Liew, Malcolm P Atkinson, Michelle Galea, Tan Fong Ang, Paul Martin, and Jano I Van Hemert. 2016. Scientific workflows: moving across paradigms. *ACM Computing Surveys (CSUR)* 49, 4 (2016), 1–39.

[14] Alex Poole and Linden J Ball. 2006. Eye tracking in HCI and usability research. In *Encyclopedia of human computer interaction*. IGI Global, 211–219.

[15] Dario D Salvucci and Joseph H Goldberg. 2000. Identifying fixations and saccades in eye-tracking protocols. In *Proceedings of the 2000 symposium on Eye tracking research & applications*. 71–78.

[16] Arvind Satyanarayan, Kanit Wongsuphasawat, and Jeffrey Heer. 2014. Declarative Interaction Design for Data Visualization. In *ACM User Interface Software & Technology (UIST)*. http://idl.cs.washington.edu/papers/reactive-vega

[17] Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, et al. 2016. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific data* 3 (2016), 160018.