PyEcosampling Whitepaper: Ecological Sampling of Gaze Shift in Python

1nd Renato Avellar Nobre
Computer Science Department
Università degli Studi di Milano
Milano, Italy
0000-0003-3360-2844

***Abstract*—Modeling visual attention, particularly stimulus-driven, saliency-based attention, has been a very active research area over the past 25 years [1]. This report considers a Python implementation (PyEcosampling) of the ecological sampling of gaze shift by Boccignone textitet al. [2]. In Boccignone's original work, he considers the problem of the variability of visual scan paths produced by human observers. His work tries to mimic a fundamental property of the eye guidance mechanism where we choose to look next at any given moment in time is not entirely deterministic, nor is it completely random [3]. This report's primary focus is to achieve the most reproducibility from the original work, providing a reliable, well-structured python library and comprehensive documentation while maintaining performance. Hopefully, providing the PyEcosampling as a complete python package will motivate other researchers to develop more work in this direction. The central intuition behind this report is to be a white paper for the python software and provide an introductory notion for the field of gaze sampling.**

***Index Terms*—python, probabilistic programming, eye movements, foraging, Lévy flight, salience, α-stable processes, visual attention**

I. Introduction

In this report, we consider a Python implementation of the ecological sampling of gaze shift by Boccignone *et al.*. [2]. Our primary focus is to achieve the most reproducibility from the original work, providing a reliable, well-structured python library and comprehensive documentation while hopefully improving performance.

In Boccignone's original work [2], from here on referred to as Ecosampling [1], he considers the problem of the variability of visual scan-paths produced by human observers. His work tries to mimic a fundamental property of the eye guidance mechanism: where we choose to look next at any given moment in time, it is not entirely deterministic, nor is it completely random [3]. The core of his work strategy relies on using a mixture of α-stable motions modulated by the scene's complexity. This strategy allows exploiting the characteristics of long-tailed distributions of gaze shift lengths for analyzing dynamic scenes. The work modules the scan-paths as a Langevin-like stochastic differential equation [2] whose noise is sampled from the distributions to the composition of Lévy flights random walks [2]. Thus it can treat different types of eye movement (smooth pursuit, saccades, fixational movements) within the same framework.

Appart from Boccignone's [2], multiple other works techniques were also employed in the Ecosampling code and thus ported to the python version. Specially, Boccignone *et al.* [4] previous study on gaze sampling which builded the foundation for the Ecosampling, and Seo's [5], [6] method for feature and salience map computing, which MATLAB's code was also converted for PyEcosampling implementation. Notwithstanding, Halır *et al.* [7] concepts are also used as means for the proto-objects ellipse fittings, and Shiner *et al.* [8], Lopez-Ruiz *et al.* [9], and Feldman *et al.* [10] concepts are used for spatial configuration complexity of Interest Points (IPs) computation.

Considering that this report is only an implementation of Ecosampling's previous work and does not derive new scientific advancements to the field, the main contributions of this report lie in the following:

- A light-weighted python version of MATLAB Ecosampling code;
- A python implementation of Seo *et. al* "Static and space-time visual saliency detection by self-resemblance" [6];
- A comprehensive web-based documentation of the work.

Initial tests on the PyEcological system demonstrated that the python version is comparable in run-time execution to the original MATLAB version. The PyEcological system achieves a 2.223 spf (seconds per frame) against a 2.175 spf for the original version, comparable results considering that Python is slower overall. Notwithstanding, the python version provides comprehensive web-based documentation, and to the best of my knowledge, the first implementation of Seo *et. al* [6] in Python,

The remainder of this work is organized as follows. Section II mentions related works and other works used to develop the proposed system. Section III describes the python implementation of Ecosampling, with all its modules described and detailed. Section IV explains the execution process in the application, develops its documentation, and demonstrates some essential run-time evaluations compared to the original MATLAB implementation. Finally, Section V concludes the work by summarizing the application and suggesting opportunities for future development.

[1]Boccignone's named its approach ecological sampling because of the results reported in the foraging literature, where similar distributions characterize the moment-to-moment relocation of many animal species between and within food patches.

II. Related Work

Modeling visual attention, particularly stimulus-driven, saliency-based attention, has been a very active research area over the past 25 years [1]. Besides all the works already aforementioned, which were considered for implementing PyEcosampling, it is worth noting some additional recent advancements in the gaze sampling literature [1], [11], [12].

Borji *et al.* [1] developed a comprehensive survey of the state-of-art in the field. Their work reviewed concepts of attention implemented in nearly 65 different models from a computation perspective while comparing approaches, capabilities, and shortcomings critically. In particular, they define 13 criteria derived from behavioral and computational studies for qualitative comparison of attention models. Those criteria include bottom-up vs. top-down, spatial vs. spatio-temporal, overt vs. covert attention, space-based vs. object-based, features, stimuli and task type, evaluation measures, and datasets. Even though this review was published before the Ecosampling work, we can fit it into their criteria. Ecosampling is a bottom-up, spatio-temporal model for overt attention to a dynamic natural stimulus considering both space and object-based interest points derived from 3D LARK features. Above all, Ecosampling is one of the few works that consider the process of where to look next not completely deterministic, but neither completely random.

Apart from the review, more recent work is "On Gaze Deployment to Audio-Visual Cues of Social Interactions" from Boccignone *et al.* [11]. Their article develops a computational model for the deployment of gaze within a conversational scene. Their principle is that attention supports our urge to forage on social cues, and under such, we spend the majority of time scanning people and spotting the speaker. Therefore, gaze dynamics is derived by reformulating attention deployment as a stochastic foraging problem. Results show that the simulated scan paths exhibit similar eye movements of human observers watching and listening to conversational clips in a free-viewing condition [11]. Their work is also a great case on the process of modeling visual attention considering both deterministic and stochastic factors. Hopefully, providing the PyEcosampling as a complete python package with code that was only available for MATLAB will motivate other researchers to develop more work in this direction.

It is worth noting that the main intuition behind this report is to be a white paper and introductory notion for the field of gaze sampling. If more details are needed for the general concepts of the field, refer to [1], [13], [14]. For details on the feature and salience method used in this implementation, read [5], [6]. For more information on the complexity methods of landscape evaluation read [8]–[10], and more information on proto-objects ellipse fittings read [7]. Last but not least, for the complete formal description and motivating reasons for the implemented software, read the original ecological sampling paper by Boccignone *et al.* [2].

III. Implementation

This section describes the PyEcosampling, a python version of the original MATLAB implementation of the Ecosampling work. Indeed PyEcosampling's software architecture was devised to quickly adapt to other gaze sampling algorithms or multiple internal mechanisms. The code should be easily adapted to change, e.g., the salience map generation or the feature map algorithm, as the user desires. Additionally, a configuration file with multiple parameters for experiment execution is provided to support the execution of experiments with ecological sampling. Finally, for experienced users, a comprehensive in-code and website documentation is provided, thus allowing a better understanding of the system and a deeper understanding of its functionalities.

A. Overview

Figure 1 demonstrates the library classes' interaction in creating a focus of attention for a single frame. It is observable that the process of creating a field of attention starts with the current frame (but also it's previous and future). The FrameProcessor class is responsible for loading those frames and computing a foveated version with a gaussian filter. The foveated image is passed for the FeatureMap class, which uses the backend class Seo's SelfRessemblance [5], [6] to calculate the 3D LARK descriptors. Further, the same backend supports the SalienceMap class, which inputs the LARK features and outputs a static and space-time visual saliency. The ProtoParameters class can compute a patch map and identify proto-objects given the salience map. Following the PyEcosampling IPSampler class, the salience map or the proto-objects can be used to sample interest points. Those interest points are represented as a histogram of their empirical distributions, which is used for the Complexity class to calculate the landscape order, disorder, and complexity values. With such values, the ActionSampler can sample the appropriate gaze motion for the frame. Finally, the information from proto-objects, interest points, complexity, and the action sampled are used to compute the appropriate gaze relocation using a Langevin-like stochastic differential equation, whose noise source is sampled from a mixture of alpha-stable distributions. Additionally, a statistics helper file encapsulate interface functions to draw samples from multiple distributions, making it easy to change the desired backend library.

PyEcosampling was implemented using the Python 3 programming language with aid from its multiple libraries. Specifically, multiple image processing functions were supported with either *Skimage* [15] and *OpenCV* [16]. OpenCV (Open Source Computer Vision Library), originally developed by Intel, is an open-source library for developing applications in the area of Computer Vision. OpenCV has an Image Processing module with more than 350 Computer Vision algorithms, such as Image filters, camera calibration, object recognition, structural analysis, and others [16]. However, despite OpenCV's powerful features, it lacks comprehensibility due to its verbose syntax and non-language-specific documentation. For this reason, wherever possible, PyEcosampling resorts to *Skimage*, which aims to be the reference library for scientific image analysis in Python [15].

Moving apart from image processing libraries, PyEcosampling also relies on *SciPy* [17], and *Numpy* [18]. SciPy

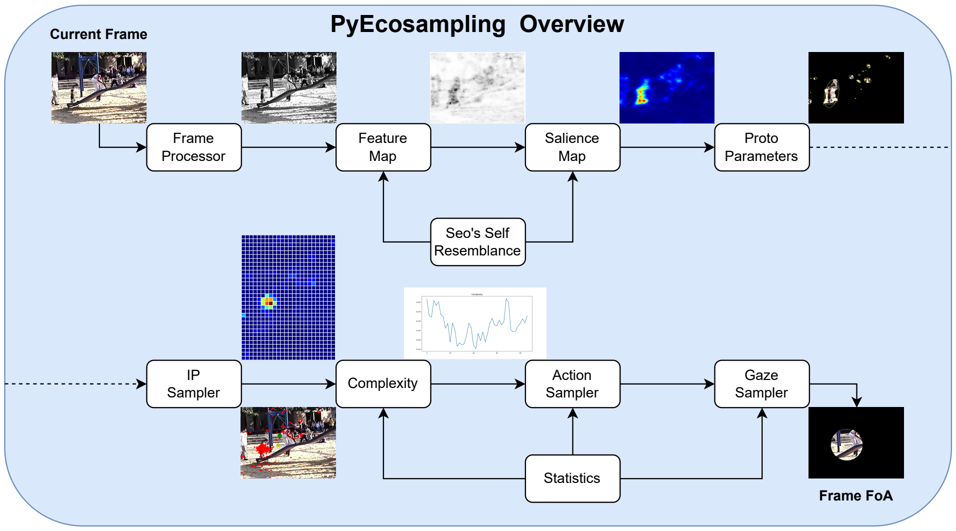

Fig. 1: Diagram of the PyEcosampling system

is an excellent toolset for scientific work which provides fundamental algorithms for scientific computing in Python. Its use made possible, as an example, the implementation of KDTrees and sampling from multiple distributions[2], including the Levy Stable distributions [17]. Numpy allowed for most of the computation, providing fast and versatile vectorization, indexing, and broadcasting concepts, that are *de-facto* standards of array computing today. Finally, *matplotlib* [19] the famous plotting library for Python, was used for visualization and saving purposes.

Notwithstanding, several codes previously in MATLAB provided by other researchers on topics related to gaze sampling were also translated to Python. Specially, on Seo *et. al* work [6] their 3D-LARK and space-time self-resemblance methods were ported to PyEcosampling.

B. FrameProcessor Class

Starting from the process of gaze sampling, there is the FrameProcessor class. Its primary purpose is to provide functions for opening and transforming the image frames. Significantly, the FrameProcessor class has functions to create the window size frames from images in the local directory, apply a foveated filtering to a frame, and reduce and resize a given frame.

Figure 2 is an example of how a frame is loaded in the frame processor. For the rest of the report, the same frame will be used for additional plots. In the way the PyEcosampling is designed, the current frame is read and the previous and the future frame, creating a three-time-step frame. Additionally, these images are converted to grayscale for the rest of the experiment.

[2]Another approach for this would be to use PyMC a probabilistic programming library for Python that allows users to build Bayesian models with a simple Python API and fit them using Markov chain Monte Carlo (MCMC) methods. However, PyMC is heavyweight and not very suitable for this problem. Even though one could argue that the PyMC would be a simple approach for estimating the alpha-stable parameters of the entailed distributions, its functionalities are not comprehensive enough. Constructing a probabilistic distribution in PyMC requires the probability density function to be written analytic, which is not possible for general stable distribution. Therefore, PyMC was left aside in exchange for a lightweight implementation using SciPy.

Fig. 2: Frame Processor visualization of a frame.

Once the three time-step is created, the next step is to apply a digital image processing technique denominated foveated imaging [2]. The foveated imaging apply a Gaussian filter in the previous focus of attention, thus focusing the amount of detail in the current gaze location and reducing the resolution of surrounding areas [2]. Figure 3 shows the foveated visualization for such with the current frame.

Additionally, the FrameProcessor class reduces the frame to the desired size and resizes it back to the original dimension. Those functions are beneficial in the creation of the feature map, which, for computation purposes, requires a reduced size image, while visualization requires re-scale back to the original format.

C. Salience by Self-Resemblance

After we have the foveated frame computed, the next step is to create a feature map and use the feature map to create a salience map with the features found. Therefore, PyEcosampling, just like the traditional Ecosampling, relies on the work

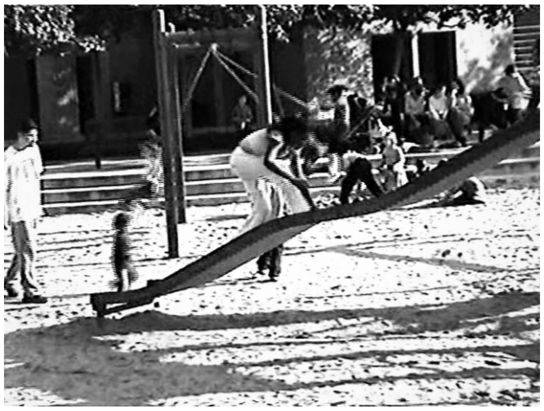
Fig. 3: Foveated Imaging of the current frame using a Gaussian filter.

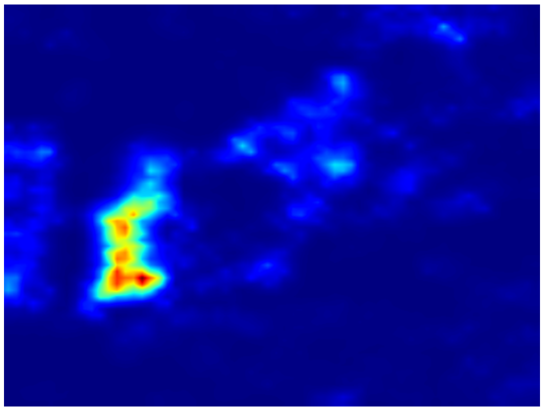
Fig. 4: Salience Map with the 3D self-resemblance spatial temporal saliency method [6].

by Seo *et al.* [6]. Their work entitled: "Static and space-time visual saliency detection by self-resemblance" presents a unified framework for static and space-time saliency detection [6]. Their method is a bottom-up approach and computes local descriptors from the given image, which measure the likeness of a pixel to its surroundings. Visual saliency is then computed using the said "self-resemblance" measure, resulting in a saliency map where each pixel indicates the statistical likelihood of saliency of a feature matrix given its surrounding feature matrices. Therefore, Seo's method provides comparable performance while handling static and space-time saliency detection [2].

Therefore, PyEcological sampling not only provides the python version for Boccignone's Ecosampling [2] but also Seo's self-resemblance salience method [6]. This is implemented in the python version as a backend library, aiming to deploy flexibility in the software for other future saliency methods.

1) FeatureMap Class: First, PyEcosampling relies on its FeatureMap class to generate the features to generate the salience. This class is designed to be a wrapper to call backend feature computing methods and should be extended if future methods are to be implemented. In the PyEcosampling version, the FeatureMap class calls Seo's 3D-LARK description features to reproduce the static and space-time visual saliency by self-resemblance.

In the original paper, Seo's work relies on a space-time local steering kernel (3D LSK). The key idea behind LSK is to obtain the space-time local structure of images by analyzing radiometric differences based on estimated gradients and determine the shape and size of a canonical kernel [6]. However, on a recent version of the MATLAB code, they implemented a 3D locally adaptive regression kernel (3D LARK), which was developed in a future work [5]. The 3D-LARK descriptor measures a self-similarity based on the "signal-induced distance" between a center pixel and surrounding pixels in a local neighborhood. Since the 3D LARK does not involve quantization in the computation of descriptors, it conveys more rich and discriminative power than other descriptors [5].

2) SalienceMap Class: After the LARK feature map calculation, the Self-Resemblance backend is ready to create the salience map, and it does so through the template class SalienceMap. In SalienceMap, just as in FeatureMap, the methodology backend to create the map is called and should be extended for future methods.

Our approach uses the foveated 3D LARK feature map to create the Seo et al. static and space-time self-resemblance salience. Overall, their method had a state-of-the-art performance on commonly used human eye fixation data. Thus, it was a good choice for the Ecosampling implementation, and it was worth translating their work to PyEcosampling. Figure 4 shows the salience map obtained for the first frame of our python implementation.

D. ProtoParameters Class

After the salience map is generated, we can look for proto-objects in the map. The ProtoParameter class is responsible for creating a proto-object map and extracting proto-parameters. In the first step, the class generates the raw patch map by the threshold of the normalized salience map to achieve a 95% significance level for deciding whether the given saliency values are in the extreme tails. After that, only a subset of the best patches ranked through their size is selected. With those selected patches, ProtoParameters class can sample the proto proto-objects by computing the patch boundaries and fitting ellipses in the region. The technique used in the Ecosampling and hence this python version derives from Halır *et al.* [7] numerically stable direct least square fitting of ellipses. Figure 5 demonstrate the resulting patch map and its proto-objects surrounded by their corresponding ellipses.

It is worth mentioning that the proto-parameters step of gaze sampling is not mandatory. If the user desires not to execute it, the user can opt to skip it and only sample interest points from the landscape.

E. IPSampler Class

Now that we have proto-objects, the system can sample interest points. In PyEcosampling, the IPSampler class is responsible for sampling those points. Overall, the class controls the sampling of interest points. If there are no proto-objects, it

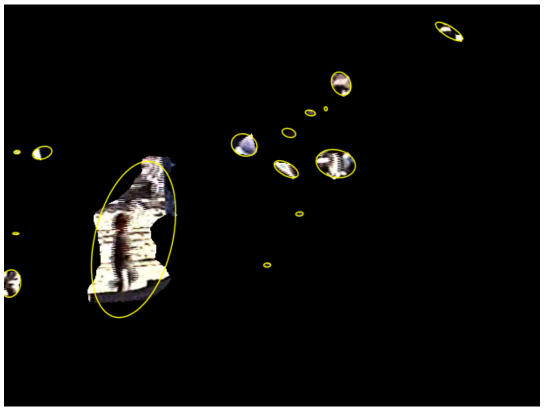
Fig. 5: Patch map and its proto-objects surrounded by their corresponding ellipses.

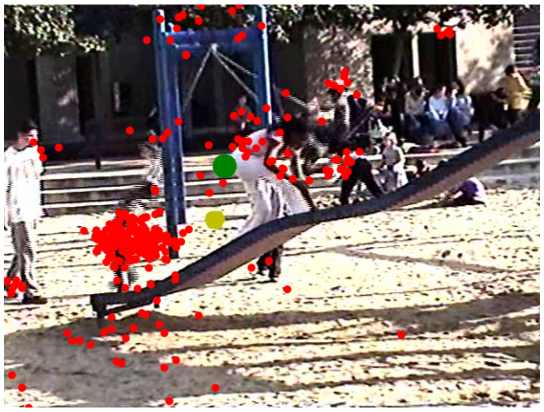
Fig. 6: Interest points (red dots) in the current frame. Yellow and green dots represents both all candidates and the maximum focus of attention.

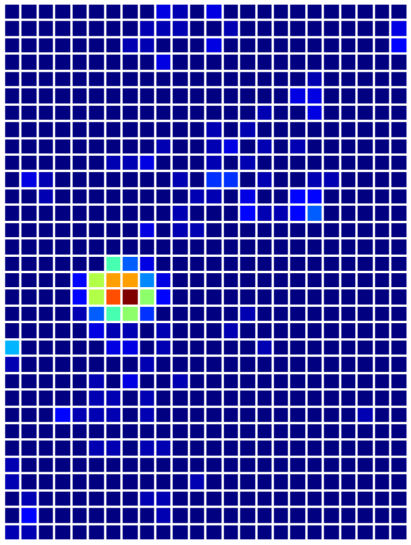
Fig. 7: Empirical Distribution for sampled IPs.

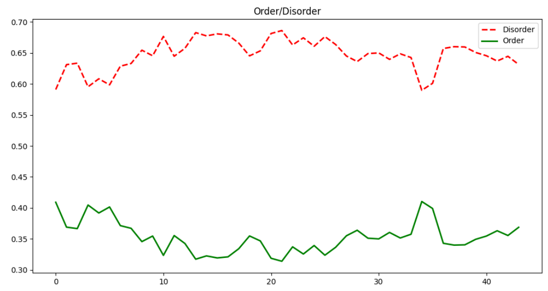

Fig. 8: Order and Disorder values for each frame on the end of the experiment.

will sample only from the landscape. If there are proto-objects, the user can choose to sample from both or just one.

Sampling points from the proto-objects correspond to drawing samples from a multivariate distribution, which will eventually generate clusters of interest points, one cluster for each patch. On the other hand, sampling points from the landscape involve sampling IPs directly from points weighted according to their salience. For each sample, set a scale by drawing from a uniform distribution over a given configuration scale [2]. Thus being able to generate IPs both from the proto-objects and directly from the salience map. Figure 6 show the obtainable results. It is worth noting that the IPs are the red dots in the picture, while the yellow and green dots represent all candidates and the maximum focus of attention.

Once all the interest points are set, we can represent the IPs as a histogram of the spatio-temporal configuration of the landscape, as presented in Figure 7. Ecosampling bases this idea on the landscape entropy determined by dispersion or concentration of food items or preys [2]. Therefore, generalizing this approach, Ecosampling can capture the time-varying configurational complexity of interest points within the landscape.

F. *Complexity Class*

Considering the generated empirical distribution of proto-objects and landscape's interest points, the next step is to compute its spatial configuration complexity. The main reason for computing the spatial complexity is the action sampling process. Low complexity scenarios usually lead to longer flights, while at the edge of order or disorder, more complex and mixed behaviors take place [2].

In the case of a time-varying visual landscape, a crowded scene with many people moving represents a disordered system (high entropy, low order) as opposed to a static scene where no events take place (low entropy, high order) [2]. Figure 8 represents the corresponding values of order and disorder through the execution of an experiment, where the x-axis represents the frame number.

Overall the Complexity class implements three different complexity calculations, Shiner-Davison-Landsberg (SDL) complexity [8], Lòpez-Ruiz, Mancini, and Calbet complexity [9], and Feldman and Crutchfield's complexity [10]. The user is free to choose the measurement they desire on the execution. Figures 8 and 9, were generated using the (SDL) complexity.

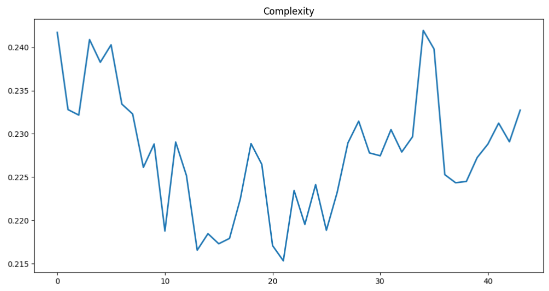

Fig. 9: Complexity values on the end of the experiment.

G. *ActionSelector Class*

Boccignone's Ecosampling defines three possible events that will eventually be identified to provide the gist of the spatio-temporal habitat: ordered dynamics, edge dynamics, and disordered dynamics, each biasing the process toward a specific gaze shift behavior. The ActionSelector class is responsible for reading the order, disorder, and complexity values, updating the Dirichlet hyperparameters, and sampling the appropriate gaze action, corresponding to the α, β, δ, and γ values of a α-stable distribution.

Summing up, the action sampling step corresponds to i) compute the complexity of the landscape as a function of sampled interest points; ii) update accordingly the Dirichlet hyperparameters; iii) sample the gaze-shift action from a multinomial distribution using the resulting Dirichlet π probability [2].

H. *GazeSampler Class*

With the proto-objects, the IPs and their empirical distribution, and the gaze action sampled, the GazeSampler class can now calculate the final Focus of Attention. The class computes the actual gaze relocation by using a Langevin-like stochastic differential equation, whose noise source is sampled from a mixture of α-stable distributions.

A Langevin equation is a stochastic differential equation describing how a system evolves when subjected to a combination of deterministic and stochastic forces [13]. The main idea of using the Langevin equation is its application to simulate Brownian motion, which is a well-known Lévy process [13]. More specifically, the Langevin equation in Ecosampling simulates a Lévy flight, in which the step lengths have a Lévy distribution [2]. A visual system producing Lévy flights implements a more efficient strategy of shifting gaze in a random visual environment than any strategy employing a typical scale in gaze shift magnitudes [2]. Indeed, the heavy-tailed distributions of gaze shift amplitudes are close to those characterizing the foraging behavior of many animal species [2].

The GazeSampler class starts by getting the possible attractors for the FoA. It can compute one or multiple gaze attractors. If a landscape of proto-objects is given, then their centers are used; otherwise, attractors are determined through the IPs sampled from saliency. The attractors are the deterministic aspect of the Langevin-like SDE, and the final FoA is determined by stepping the differential equation. Figure 10 shows a resulting Focus of Attention. If the Langevin fails to return a valid FoA, the system can resort to a perturbed argmax solution or even keep the previous one.

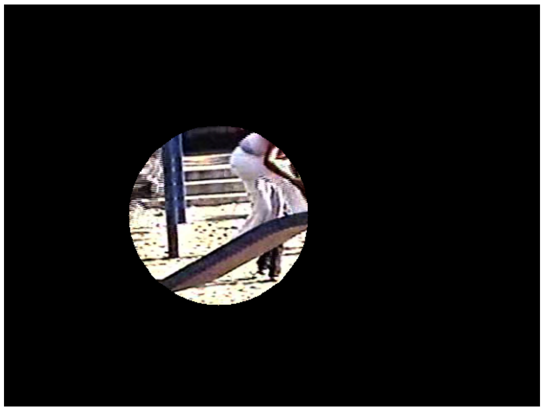

Fig. 10: Final focus of attention for the current frame.

IV. Execution and Run-time Evaluation

Execution of the PyEcosampling library relies on Python (which has been tested on ¿=3.8) and Tkinter, a GUI backend for matplotlib. Additionally, regarding the operational system. PyEcosampling was tested on macOS Monterey (version 12.4) and Ubuntu 20. To install the Tkinter dependency, you can rely on either apt-get (on Ubuntu) or brew (on macOS). The package name is python3-tk.

After the dependencies are correctly installed, all that is left to do is install the software. To create the software library and run the demos, given that your Python is in the correct version and Tkinter is installed, clone the repository[3], enter the project root folder, and install the project requirement using:

- pip install -r requirements.txt

A sample sequence for demo purposes is provided with the source code in the *data* directory. To add your clips, add a folder in the same directory. The folder you add must have the video as a sequence of ordered frames. To direct the application execution for your data folder, change the configurations in the *ecosampling/config.py* folder.

Ecosampling might be used as an application stand-alone or as an API to write your gaze sampling script. To run the application as it is, you can run the following command on the root of the project:

- python ecosampling

This should be enough to generate all the processes of gaze sampling with interactive visualization and data saving for your application. If you want to play around, change parameters, or try with your personalized data set, try changing the *ecosampling/config.py* file. Different gaze shifting behaviors can be obtained by playing with parameters in the configuration script.

If you want an in-depth understanding of how the project is organized or the functioning of the API, read the documentation provided as a Read the Docs website. To open

[3]Available at https://github.com/phuselab/INMCA_Nobre

the web documentation, open the file on the directory: *docs/build/index.html*. This documentation was developed knowing that if the software is to be published as a pip package, the documentation is ready to be uploaded to Read the Docs.

A. Performance Evaluation

Ecosampling's original system was implemented in plain MATLAB code with no specific optimizations and executed on a 2.8 GHz Intel Core 2 Duo processor, 4 GB RAM, under Mac OS X 10.5.81. The original paper calculates the execution time of the experiment in an spf measurement (seconds per frame, frame size 640×480 pixels). Regarding actual performance under such a setting, the average elapsed time for the whole processing amounts to 2.175 spf. This value is divided into 0.044 spf for the foveated frame, 1.155 spf for feature computation, 0.846 spf for salience, 0.106 spf for obtaining patches, 0.021 spf for sampling interest points, 0.001 spf to evaluate the complexity, and eventually 0,002 spf for sampling the new point of gaze.

For the PyEcosampling computing on the python version, the time library was used, which is worth mentioning, that calculates the real-time, not instruction time (including time used by other programs), therefore influenced by computer load. The tests were conducted in a macOS Monterey 12.4, with 2,9 GHz Dual-Core Intel Core i5 and 8 GB 2133 MHz LPDDR3 RAM. The average elapsed time for the whole processing amounts to 2.223 spf, a little higher than the MATLAB version. This value is divided into 0.0190 spf for the foveated frame, 1.925 spf for feature computation, 0.150 spf for salience, 0.030 spf for obtaining patches, 0.02 spf for sampling interest points, 0.0000066 spf to evaluate the complexity, and eventually 0.010 spf for sampling the new point of gaze.

Overall, in the python version, the feature computation process, interest point sampling, and gaze sampling are higher than the MATLAB, but only the feature computation was of significant impact, 1.925 to 1.155. This is probably due to the $\theta(W^2T)$ time complexity for the feature calculation, where W is the LARK window size and T LARK time-step size. Even though both implementations have the same $\theta(W^2T)$ time complexity, Python loops are usually about 10x slower than MATLAB's, thus giving this difference.

V. Conclusion and Future Work

This paper described the implementation of Ecosampling as a python software denominated PyEcosampling. The newly devised class-based infrastructure aims to provide an easy-to-use framework API. Considering that Ecosampling is developed in Python, with complete documentation, use case examples, and even porting the Seo's self-resemblance work, hopefully, this framework will impact other researchers to investigate the process of the gaze of a mixture of deterministic and stochastic processes.

Results obtained in PyEcosampling, regarding execution are very similar to the MATLAB version, which is a great overall result considering that Python is not a fast language. Future work opportunities rellies on reducing the execution time of Seo's feature processing techniques, possibly implementing a vectorized approach. Additionally, if this software is to be disclosed publicly, implementing command line execution and publishing the software in the Python Package Index (PyPI) could be of great interest.

References

[1] A. Borji and L. Itti, "State-of-the-art in visual attention modeling," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 1, pp. 185–207, 2012.

[2] G. Boccignone and M. Ferraro, "Ecological sampling of gaze shifts," *IEEE transactions on cybernetics*, vol. 44, no. 2, pp. 266–279, 2013.

[3] R. L. Canosa, "Real-world vision: Selective perception and task," *ACM Transactions on Applied Perception (TAP)*, vol. 6, no. 2, pp. 1–34, 2009.

[4] G. Boccignone and M. Ferraro, "The active sampling of gaze-shifts," in *International Conference on Image Analysis and Processing*. Springer, 2011, pp. 187–196.

[5] H. J. Seo and P. Milanfar, "Face verification using the lark representation," *IEEE Transactions on Information Forensics and Security*, vol. 6, no. 4, pp. 1275–1286, 2011.

[6] ——, "Static and space-time visual saliency detection by self-resemblance," *Journal of vision*, vol. 9, no. 12, pp. 15–15, 2009.

[7] R. Halır and J. Flusser, "Numerically stable direct least squares fitting of ellipses," in *Proc. 6th International Conference in Central Europe on Computer Graphics and Visualization. WSCG*, vol. 98. Citeseer, 1998, pp. 125–132.

[8] J. S. Shiner, M. Davison, and P. T. Landsberg, "Simple measure for complexity," *Physical review E*, vol. 59, no. 2, p. 1459, 1999.

[9] R. Lopez-Ruiz, H. L. Mancini, and X. Calbet, "A statistical measure of complexity," *Physics letters A*, vol. 209, no. 5-6, pp. 321–326, 1995.

[10] D. P. Feldman and J. P. Crutchfield, "Measures of statistical complexity: Why?" *Physics Letters A*, vol. 238, no. 4-5, pp. 244–252, 1998.

[11] G. Boccignone, V. Cuculo, A. D'Amelio, G. Grossi, and R. Lanzarotti, "On gaze deployment to audio-visual cues of social interactions," *IEEE Access*, vol. 8, pp. 161 630–161 654, 2020.

[12] L. Itti, C. Koch, and E. Niebur, "A model of saliency-based visual attention for rapid scene analysis," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 20, no. 11, pp. 1254–1259, 1998.

[13] G. Boccignone, *Advanced statistical methods for eye movement analysis and modeling: a gentle introduction*, 10 2019.

[14] A. C. Schütz, D. I. Braun, and K. R. Gegenfurtner, "Eye movements and perception: A selective review," *Journal of vision*, vol. 11, no. 5, pp. 9–9, 2011.

[15] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors, "scikit-image: image processing in Python," *PeerJ*, vol. 2, p. e453, 6 2014. [Online]. Available: https://doi.org/10.7717/peerj.453

[16] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[17] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[18] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[19] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.