Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

A modular toolkit for visual tracking performance evaluation

Luka Čehovin Zajc

Faculty of Computer and Information Science, University of Ljubljana, Slovenia

A R T I C L E   I N F O

Article history:
Received 23 April 2020
Accepted 6 November 2020

Keywords:
Computer vision
Visual tracking
Performance evaluation
Experiment result analysis

A B S T R A C T

We present a modular software package for conducting single-target visual object tracking experiments and analyzing results. Our software supports many of the common usage patterns in visual tracking evaluation out of the box, but is also modular and allows various extensions. Users are able to integrate existing implementations of visual tracking algorithms with little additional effort using a standardized and flexible communication protocol. The software has been the technical backbone of the VOT Challenge initiative for many years and has grown and evolved with the competitions that it supported. We present its current state and the capabilities of the package and conclude with some plans for future development.

Code metadata

Current code version	7.0.2
Permanent link to code/repository used of this code version	https://github.com/ElsevierSoftwareX/SOFTX_2020_180
Legal Code License	GNU GPL v3
Code versioning system used	Git
Software code languages, tools, and services used	Matlab / Octave
Compilation requirements, operating environments & dependencies	Windows, Linux, OSX
Link to developer documentation/manual	http://docs.votchallenge.net/
Support email for questions	luka.cehovin@fri.uni-lj.si

1. Introduction

Objective performance measuring is a crucial component in most engineering disciplines, designing new visual tracking algorithms is no exception. In comparison to other computer vision research domains, visual tracking performance evaluation scenarios are one of the more complex ones. Evaluation can focus on different usage scenarios which have different rules and desired qualities. It is especially challenging to perform evaluation when the temporal component is considered (on-the-fly reinitialization, real-time evaluation, influence of parameters).

Because of this, large-scale comparisons using a consistent methodology were in the past a challenging task that was limited in many ways. Authors of these evaluations were left with adapting or even implementing trackers which typically resulted in standalone survey papers [1–4]. Recently, more focus has been directed towards designing consistent evaluation methodologies [4–6]. Wu et al. [3] presented a benchmark dataset and provided a simple set of scripts that implement only a single off-line evaluation strategy. Modifications of the source code are required to add new trackers. In [4] authors propose only a new methodology, but do not release any source code, which makes its adoption difficult.

Since then, numerous benchmarks have been presented for different visual tracking scenarios, their contributions ranging from addressing new challenging scenarios, e.g. long-term tracking [7–9] or application-specific tracking [10], or simply making consistently larger datasets [9,11,12], primarily aimed at the new generation of data-hungry deep-learning tracking algorithms. Most of these benchmarks are supported by some kind of software, however, as we show in Table 1, the design of the software is very narrow, either because it is tightly connected to the dataset or evaluation methodology [8,10,12] or in that it limits the integration of tracking algorithms in some way (e.g. programming language, experiment methodology) [9].

E-mail address: luka.cehovin@fri.uni-lj.si.

Table 1
Comparison of the most known tools for single-target visual tracking evaluation in terms of flexibility. The following attributes are presented: Language — in which programming language is the tool written in, Trackers — what is the mechanism for tracker integration, Datasets — what kind of datasets are supported without major modification of the source code, Experiments — what kind of experiments are supported, Active — has the source-code for the tool been updated in the last year.

	Language	Trackers	Datasets	Experiments	Active
OTB [3]	Matlab	File	no	unsupervised	no
UAV [10]	Matlab	File	no	unsupervised	no
OxUvA [7]	Python	File	no	unsupervised	yes
GoT-10k [9]	Python	Python API	yes[1]	many[2]	yes
LaSOT [12]	Matlab	File	no	unsupervised	no
TrackingNet [11]	Python	N/A[3]	no	unsupervised	yes
ours	Matlab	TraX protocol	yes	many	yes

[1]Experiments are hard-coded for the use with corresponding datasets.
[2]More experiment types are available only for the VOT datasets, other datasets are limited to unsupervised use-case.
[3]The tool only takes care of downloading the data, not running trackers.

One of the most established efforts in visual tracking performance evaluation is the VOT Challenge initiative [13]. The initiative started in 2013 with the aim of consolidating the evaluation methodology in single object visual tracking by organizing an annual challenge and a workshop based on the results of a challenge. From the technical point of view, the core of the challenge is software, which we are presenting in this article. The software, to which we will refer to as the *VOT toolkit* or the *toolkit*, can consistently perform challenge experiments in diverse environments and can scale with the growing demands of the research community. Even though the VOT toolkit has been used extensively by many researchers, its design and the wide scope of its applicability have not been discussed in any prior work. This is the main reason of this article. In the following chapters we will describe the problems that the toolkit is addressing, its modular structure, and some diverse use-cases.

2. Problems and background

From the very beginning, the main goal of the VOT toolkit design was to make the challenge of single-target model-free visual tracking evaluation faster and more consistent. It was also clear that the evaluation methodology in the field of visual tracking is not fixed and will likely get more ambitious and sophisticated so the toolkit was designed with growth in mind.

The other problems that the toolkit had to accommodate were the diverse development environments used by the researchers in the field of visual tracking. At the time of its initial versions, most researchers used Matlab to write their own algorithms with some rare exceptions of trackers implemented in C/C++. Nowadays Python is dominating the field. To address this diversity, a language independent way of communication had to be developed. The communication also had to be interactive (the toolkit has to be able to act on information for every given frame) and support multi-modal sequences to allow more sophisticated tracking evaluation scenarios. This standard protocol is known as the *TraX protocol* and is presented more in details in [14]. The toolkit can be used directly with the protocol, but language-specific wrappers are also provided that make its adoption easier and hide some less frequently used functionality (run-time parameters, in-memory images, etc.).

3. Architecture

The toolkit is a set of Matlab (Octave compatible) scripts and some C++ code at performance critical algorithms. It has minimal dependencies, only image processing toolbox. The scripts are structured in several modules (directories) that operate with the most important data structures.

3.1. Tracker

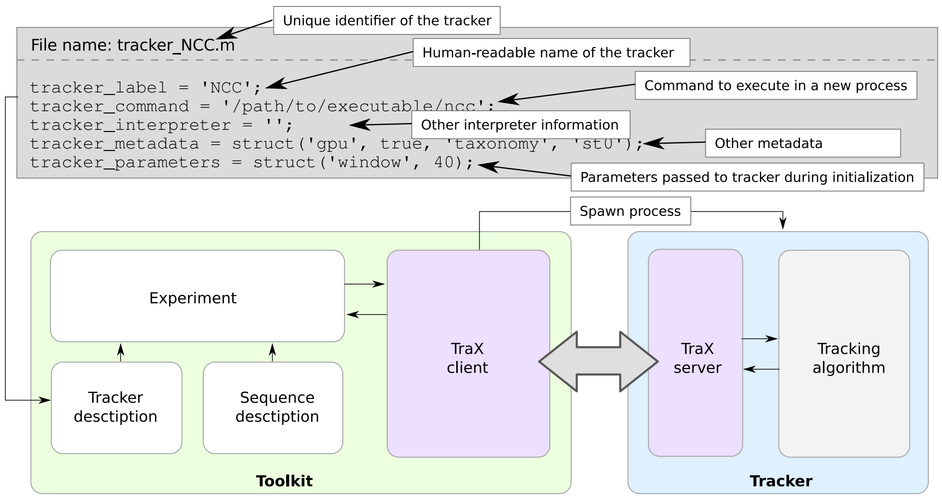
A tracker entity denotes a specific visual tracking algorithm together with its configuration parameters. Multiple trackers can be evaluated on a dataset and compared through their results. Trackers are denoted by their unique identifier, they also have a set of other metadata assigned to them through a tracker description file. The main data, defined in this file is a corresponding executable or a script containing an entry point to a tracking algorithm, but it can also contain data about the interpreter, required paths for linking and other information useful during the evaluation or analysis. An example of a tracker description file is presented in Fig. 1.

By design the tracker is always contained in a separate executable which is run by the toolkit when needed in a new temporary directory. The main reason for this is stability of the evaluation process and equal handling of all algorithms regardless of the programming language that they were implemented in. The toolkit and a tracker communicate with each other using the TraX protocol [14]. The communication flow is regulated by the type of the experiment that is conducted. This approach also has downsides, the main one is that the debugging of the tracker is more difficult. This is mitigated by the toolkit by keeping the entire output of the tracker in a log file if the tracking session has been interrupted by an error. The log files are kept in the workspace (see Section 3.4).

3.2. Sequence

A sequence is a list of images with manually annotated ground-truth positions of the object used for evaluation. The toolkit supports *one file per frame* sequences, which is a standard in visual tracking evaluation. The filenames of the images can follow an arbitrary numeric pattern that is defined in sequence metadata file and defaults to `%08d.jpg`. The metadata contains other information about the sequence, like frame rate and other useful metadata. Additionally, each sequence can have more than one channel. With this the toolkit can support different modalities and even multi-modal tracking scenarios, e.g. infra red (IR), RGB-T (visible light and thermal), RGB-D (visible light and depth information).

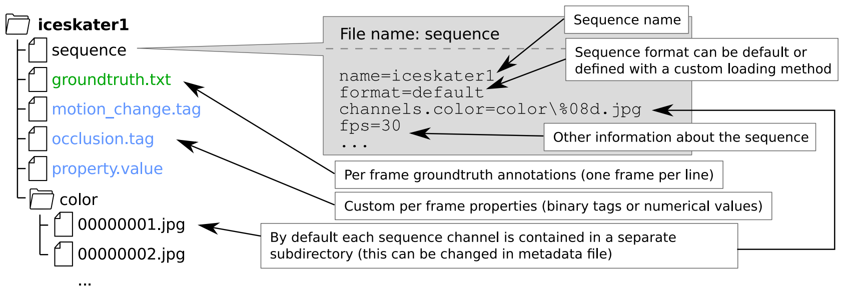
Multiple sequences are combined in a dataset. Several generic dataset were assembled for various VOT challenges, those can be downloaded automatically by the toolkit from VOT servers. Custom datasets can also be assembled by any researcher from various sources with minimum amount of effort, sometimes simply by adding a metadata file that describes its structure and properties. An example of sequence description file is shown in Fig. 2.



```
tracker_label = 'NCC';
tracker_command = '/path/to/executable/ncc';
tracker_interpreter = '';
tracker_metadata = struct('gpu', true, 'taxonomy', 'st0');
tracker_parameters = struct('window', 40);
```

Unique identifier of the tracker
File name: tracker_NCC.m
Human-readable name of the tracker
Command to execute in a new process
Other interpreter information
Other metadata
Parameters passed to tracker during initialization
Spawn process
Experiment
Tracker desctiption
Sequence desctiption
TraX client
Toolkit
TraX server
Tracking algorithm
Tracker

Fig. 1. Tracker description example and an illustration of tracker execution.



```
name=iceskater1
format=default
channels.color=color\%08d.jpg
fps=30
...
```

iceskater1
sequence
groundtruth.txt
motion_change.tag
occlusion.tag
property.value
color
00000001.jpg
00000002.jpg
...
File name: sequence
Sequence name
Sequence format can be default or defined with a custom loading method
Other information about the sequence
Per frame groundtruth annotations (one frame per line)
Custom per frame properties (binary tags or numerical values)
By default each sequence channel is contained in a separate subdirectory (this can be changed in metadata file)

Fig. 2. Directory structure for a single sequence and a structure of a sequence metadata file.

3.3. *Experiment*

An experiment defines a tracking scenario that a tracker is exposed to on a given sequence. This includes any sequence modifications (e.g. image noise, initialization error, frame skipping) as well as control of the tracking flow (e.g. supervised or real-time tracking). Currently, the toolkit supports three types of experiments: unsupervised (run a tracker on a sequence without any intervention), supervised (run a tracker and reinitialize it if it drifts away from the target), and real-time (takes into account the speed of execution, skips frames if a tracker is too slow). More experiments can be added by implementing a well defined interface.

An experiment stack is a set of configured experiments. A stack is used to denote high-level evaluation configuration tied to a specific challenge, but custom experiment stacks can also be created easily. An example of stack configuration is shown in Fig. 3.

3.4. *Workspace*

A toolkit workspace is a directory that contains a local copy of a dataset and configuration that ties the dataset to a specific experiment stack. The main idea is that the evaluation is not performed in the same directory where the source code for the toolkit is which is the case with many of the other evaluation frameworks. A workspace stores raw tracking results in a structured hierarchy of files. These results can be used for later analysis or can be packaged and submitted for external comparison (e.g. VOT challenge).

Although the nature of the toolkit is very extendable a few entry points that represent the most frequent use-cases have been made more accessible as executable scripts automatically generated in each workspace during its creation. These include testing tracker integration (`run_test`), running the experiment stack (`run_experiments`), performing analysis (`run_analysis`), and packaging results (`run_pack`) since the main motivation for creating the toolkit is to support organization of VOT Challenges.

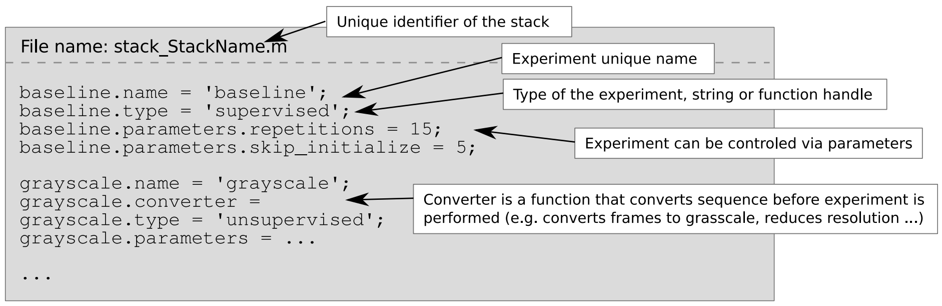
3.5. *Analysis and reports*

The final two modules of the toolkit are dedicated to the analysis of raw results and generation of reports based on the analysis. The toolkit supports various performance measures, e.g. OTB overlap threshold curve [3] and more complex measures used in VOT Challenge, e.g. accuracy-robustness (AR) [6] analysis and expected average overlap (EAO) [15].

The analysis module is used by report generation module that contains functions that generate high-level reports involving multiple experiments on multiple trackers and a large dataset. Examples from such report are shown in Fig. 4. The reports support generation of plots and tables, all of them are available in a set of HTML documents available for browsing. Additionally export to EPS or raw Matlab figures is also supported for plots and tables can be exported to comma separated values (CSV).

4. Examples of use-cases

In this section we briefly present a few use-cases that we have considered in the design of the toolkit. Note that these are not the only possible scenarios, but still show the great potential of the toolkit beyond VOT Challenges.



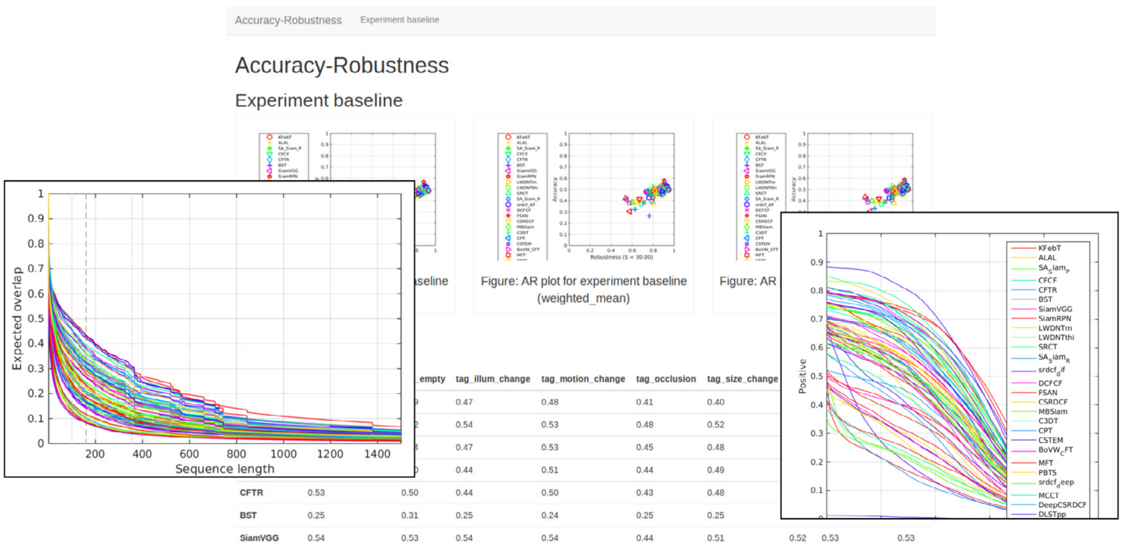
```
File name: stack_StackName.m
baseline.name = 'baseline';
baseline.type = 'supervised';
baseline.parameters.repetitions = 15;
baseline.parameters.skip_initialize = 5;

grayscale.name = 'grayscale';
grayscale.converter =
grayscale.type = 'unsupervised';
grayscale.parameters = ...

...
```

Unique identifier of the stack
Experiment unique name
Type of the experiment, string or function handle
Experiment can be controled via parameters
Converter is a function that converts sequence before experiment is performed (e.g. converts frames to grasscale, reduces resolution ...)

Fig. 3. Example of stack configuration with two experiments.



		_empty	tag_illum_change	tag_motion_change	tag_occlusion	tag_size_change			
			0.47	0.48	0.41	0.40			
			0.54	0.53	0.48	0.52			
			0.47	0.53	0.45	0.48			
			0.44	0.51	0.44	0.49			
CFTR	0.53	0.50	0.44	0.50	0.43	0.48			
BST	0.25	0.31	0.25	0.24	0.25	0.25			
SiamVGG	0.54	0.53	0.54	0.54	0.44	0.51	0.52	0.53	0.53

Fig. 4. A few examples of a generated HTML report.

- *VOT Challenge participation* is historically the core use case. The user creates a workspace for one of the pre-defined experiment stacks available with the toolkit that correspond to the open challenges. User integrates a tracker, tests it, then performs experiments. After experiments are finished, the results can be packaged and submitted to the organizers for examination. A detailed tutorial on how to do this is available in the online toolkit documentation.
- *Tracker comparison for a specialized dataset*. In case finding the best tracking algorithm for a specific scenario is desired, the toolkit supports manual creation of custom dataset. Then, a set of trackers can be evaluated with specific experiments and compared to one another.
- *Tracker development* can be facilitated by evaluating multiple alternative modifications easily. Multiple tracker instances are registered in the toolkit and compared on a given sequence dataset or against existing relevant trackers.
- *Algorithm behavior analysis* enables in-depth study of an individual tracking algorithm over a set of input parameters. In this case, multiple tracker instances are registered with the toolkit, all of them using the same code base, but different initialization parameters. These parameters can also be passed to a tracker during initialization using custom arguments of the TraX protocol [14]. A set of experiments is performed and results analyzed.
- *New experiments* can be supported through the modular design of experiment function. An example of this is presented in [16], where a dynamic sequence generation based on changing parameters was integrated into the evaluation pipeline.

5. Conclusions

In this paper we have presented the VOT toolkit, a framework for evaluating the performance of single-target visual object tracking algorithms. We have illustrated how it is used to perform diverse set of experiments used in VOT challenges and noted how it can be used to perform even other types of tracking experiments. Because of its flexibility, the toolkit was able to withstand the test of time and endured seven years of growth and evolution, it supports different types of experiments, different annotation formats and even multi-modal sequence datasets.

The toolkit is written as a set of Matlab scripts, which made a lot of sense when the initial version was created. Since the programming trends in the research community have changed in favor of other languages, especially Python. Our future goal is therefore to bring the toolkit closer to the new generation of researchers, potentially rewriting it in another language following the same design concepts and maintaining compatibility of the persistent data. Additionally, the toolkit will also grow with new editions of VOT Challenge which are expected to push the boundaries of single-target visual tracking performance evaluation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work has been in part funded by the ARRS project Z2-1866.

References

[1] Wang Q, Chen F, Xu W, Yang M-H. An experimental comparison of online object-tracking algorithms. In: SPIE Optical Engineering+ Applications. San Diego; 2011, http://dx.doi.org/10.1117/12.895965, p. 81381A–81381A–11. http://adsabs.harvard.edu//abs/2011SPIE.8138E.56W.

[2] Pang Y, Ling H. Finding the best from the second bests – inhibiting subjective bias in evaluation of visual tracking algorithms. In: IEEE International Conference on Computer Vision. 2013, p. 2784–91. http://dx.doi.org/10.1109/ICCV.2013.346.

[3] Wu Y, Lim J, Yang M-h. Online object tracking: A Benchmark. In: IEEE computer society conference on computer vision and pattern recognition. 2013, p. 2411–2418.

[4] Smeulders AWM, Chu DM, Cucchiara R, Calderara S, Dehghan A, Shah M. Visual tracking: an experimental survey. IEEE Trans Pattern Anal Mach Intell 2013;36(7):1442–68. http://dx.doi.org/10.1109/TPAMI.2013.230.

[5] Wu H, Sankaranarayanan AC, Chellappa R. Online empirical evaluation of tracking algorithms. IEEE Trans Pattern Anal Mach Intell 2010;32(8):1443–58. http://dx.doi.org/10.1109/TPAMI.2009.135.

[6] Čehovin L, Leonardis A, Kristan M. Visual object tracking performance measures revisited. IEEE Trans Image Process 2016;25(3):1261–74.

[7] Valmadre J, Bertinetto L, Henriques JF, Tao R, Vedaldi A, Smeulders AW, Torr PH, Gavves E. Long-term tracking in the wild: A benchmark. In: Proceedings of the european conference on computer vision (ECCV). 2018, p. 670–685.

[8] Moudgil A, Gandhi V. Long-term visual object tracking benchmark. In: Asian Conference on Computer Vision. Springer; 2018, p. 629–45.

[9] Huang L, Zhao X, Huang K. GOT-10k: A large high-diversity benchmark for generic object tracking in the wild. arXiv preprint arXiv:1810.11981. 2018.

[10] Mueller M, Smith N, Ghanem B. A benchmark and simulator for UAV tracking. In: European Conference on Computer Vision. Springer; 2016, p. 445–61.

[11] Muller M, Bibi A, Giancola S, Alsubaihi S, Ghanem B. TrackingNet: A large-scale dataset and benchmark for object tracking in the wild. In: The European Conference on Computer Vision (ECCV). 2018.

[12] Fan H, Lin L, Yang F, Chu P, Deng G, Yu S, Bai H, Xu Y, Liao C, Ling H. Lasot: A high-quality benchmark for large-scale single object tracking. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2019, p. 5374–5383.

[13] Kristan M, Matas J, Leonardis A, Vojir T, Pflugfelder R, Fernandez G, Nebehay G, Porikli F, Čehovin L. A novel performance evaluation methodology for single-target trackers. IEEE Trans Pattern Anal Mach Intell 2016. http://dx.doi.org/10.1109/TPAMI.2016.2516982.

[14] Čehovin L. Trax: The visual tracking exchange protocol and library. Neurocomputing 2017;260:5–8. http://dx.doi.org/10.1016/j.neucom.2017.02.036.

[15] Kristan M, Matas J, Leonardis A, Felsberg M, Čehovin Zajc L, Fernandez G, Vojir T, Häger G, Nebehay G, Pflugfelder R, et al. The Visual Object Tracking VOT2015 challenge results. In: Visual Object Tracking Workshop 2015 At ICCV2015. 2015.

[16] Čehovin L, Lukežič A, Leonardis A, Kristan M. Beyond standard benchmarks: Parameterizing performance evaluation in visual object tracking. In: The IEEE International Conference on Computer Vision (ICCV). 2017.