# Visual Object Tracking Challenge (VOT2014) Evaluation Kit

Matej Kristan, Luka Čehovin, Tomaš Vojir, Georg Nebehay

May 27, 2014

## Contents

# 1 How to participate in the VOT2014 challenge

The task addressed in the VOT2014 challenge is to reliably track a single moving object in realistic scenes subject to various common conditions.

The participants of VOT2014 challenge are required to:

1. Download the The VOT2014 evaluation kit.
2. Download the The VOT2014 dataset.
3. Integrate their tracker with the VOT2014 toolkit (see Section 6) and run the experiments (Section 5).
4. Submit the formatted results via the VOT2014 submission page.

The results will then be evaluated under a common VOT2014 methodology. The kit automatically generates the results in a zip file (Section 8.5) as well as in the form of a table. The table should be added to the *"The VOT2014 Challenge Results and Short Description Page"* document (a template is available here), which should also provide a short description of the tracker. This year we have standardized the content of the document to facilitate inclusion in the results paper. The document, along with the results zip file should be submitted through the VOT2014 submission page.

Note that in order to attend the VOT2014 challenge, the submission of the results zip file and the *"The VOT2013 Challenge Results and Short Description Page"* document is mandatory. The participants may also choose to submit the binaries of their tracker and/or the source code. The binaries will be used by the VOT committee to verify the submitted results. If the authors chose, the binaries will be deleted by the VOT2014 committee after the completion of the verification.

The VOT2014 also solicitates full-length papers describing trackers tested on VOT2014 to be considered for publication at the VOT2014 workshop at ECCV2014. For further details on the participation, please refer to the VOT2014 participation page.

## 1.1 Becoming a coauthor of the VOT2014 results paper

Like in VOT2013, the results of participating trackers will be published at the workshop organized in conjunction with the VOT2014 challenge. This year, the VOT2014 workshop will take place in conjunction with the ECCV2014. All participants who conduct all the experiments from Section 5, perform reasonably well compared to the baseline tracker and submit the valid results along with the short tracker description document in time, will become coauthors of the results paper published at the workshop. The baseline tracker that a submitted tracker has to outperform to gain co-authorship will be a simple non-adaptive normalized cross correlation tracker that will be implemented and provided by the VOT2014 committee.

## 1.2 Going beyond VOT2013

The major differences between the VOT2013 and the VOT2014 are as follows:

- A new fully-annotated dataset including new sequences is offered to the tracking community.
- Object annotations will be in form of rotated bounding boxes rather than just axis-aligned as used in the VOT2013.
- An improved evaluation kit is made available that performs the experiments faster than VOT2013 kit and uses a powerful, but simple communication protocol between the kit and trackers.

## 1.3 Document overview

The remainder of this document is structured as follows. In Section 2 we overview the class of trackers considered, the required output and we make a note about the tracker parameters. Section 1.1 states the requirements of becoming a co-author of the VOT2014 results paper. Section 3 overviews the VOT2014 dataset, Section 4 makes note on the performance measures, while Section 5 overviews the VOT2014 experiments. Section 6 provides a quickstart for VOT2014 evaluation kit integration. Further details on integration and communication protocols are provided in Sections 7 and 8.

# 2 The class of trackers considered

The VOT2014 challenge considers single-camera, single-object, short-term, model-free, causal trackers. The model-free property means that a single training example is provided to the tracker by a bounding-box at the time of s(re)initialization. Causality requires that the output of the tracker depends only on the frames from the initialization up to the current frame. Use of any information from future frames is not allowed. The complete reinitialization of the tracker at time-step $t$ starts tracking on a frame $t$ of a video sequence as if the sequence started at that frame. After reinitialization, use of information from frames 1 up to $t-1$, e.g. learnt dynamics or appearance, is not allowed. A complete reinitializaion is required to cleanly recover from the failure.

## 2.1 The tracker output

The VOT2013 challenge required the tracker to produce a single axis-aligned bounding box per frame in the sequence. The VOT2014 challenge goes a step further, by allowing a rotated bounding box per each frame (see Figure 1). Note that a tracker may still output the axis-aligned bounding box if it chooses. For a particular format of the output, please refer to Section 7.

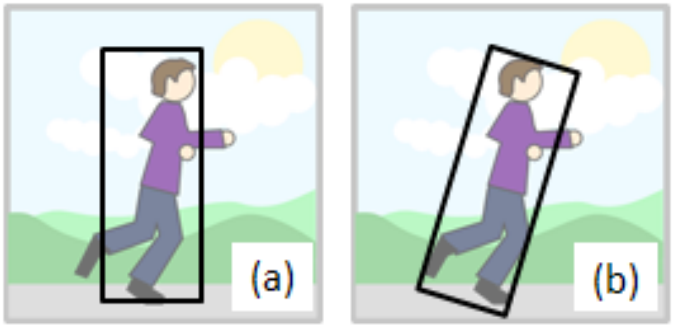


Figure 1: The axis-aligned bounding box (a) and the rotated bounding box (b).

The tracker should always output a bounding box. Some trackers may internally detect a loss of target and it is up to the tracker to decide on output bounding box in that case. The most naive output in such situation, for example, would be setting the bounding box to the size of the entire image with its center corresponding to the center of the image. An alternative is to apply a better prediction using a dynamic model. In any case, we are measuring the accuracy and robustness of the tracker (see Section 4). The naive choice will be reflected as a reasonably robust, but very low in accuracy. Using a better prediction model might lead to a better accuracy but also to a lower robustness if completely losing the target.

## 2.2 A note on parameters

Participants are expected to submit a single set of results per tracker. Participants who have investigated several trackers may submit one result per tracker. Changes in tracker parameters do not constitute a different tracker. The tracker is required to run with fixed parameters on all experiments. Note that the method may internally change specific parameters, but these have to be set automatically by the tracker, e.g., from the image size and the initial size of the bounding box, and should not be set by detecting a specific test sequence and then selecting the parameters that were hand-tuned to this sequence.

# 3 Dataset

The tracking evaluation will be conducted on a dataset provided by the VOT2014 challenge. Our main criteria for dataset selection was that various realistic scenes and conditions are sufficiently well represent by the dataset. We avoided including too many sequences to keep the time required to conduct an experimental evaluation for an average tracker reasonably low. Note, however, that since the evaluation kit for VOT2014 is significantly more advanced than that of VOT2013, we were able to increase the number of sequences compared to VOT2013.

The dataset was prepared as follows. We have collected a large pool of the sequences (394 sequences containing in total of 139548 frames) that have been used by various authors in the tracking community, we have added

the recently published ALOV dataset [1], Online Object Tracking Benchmark [2] and additional, so far unpublished, sequences. Each sequence was annotated by several attributes and a subset of sequences (25) was selected from this pool such that the various visual attributes (occlusion, illumination changes, etc.) were still represented well within the selection. The images-exemplars from the selected sequences are shown in Figure 2.

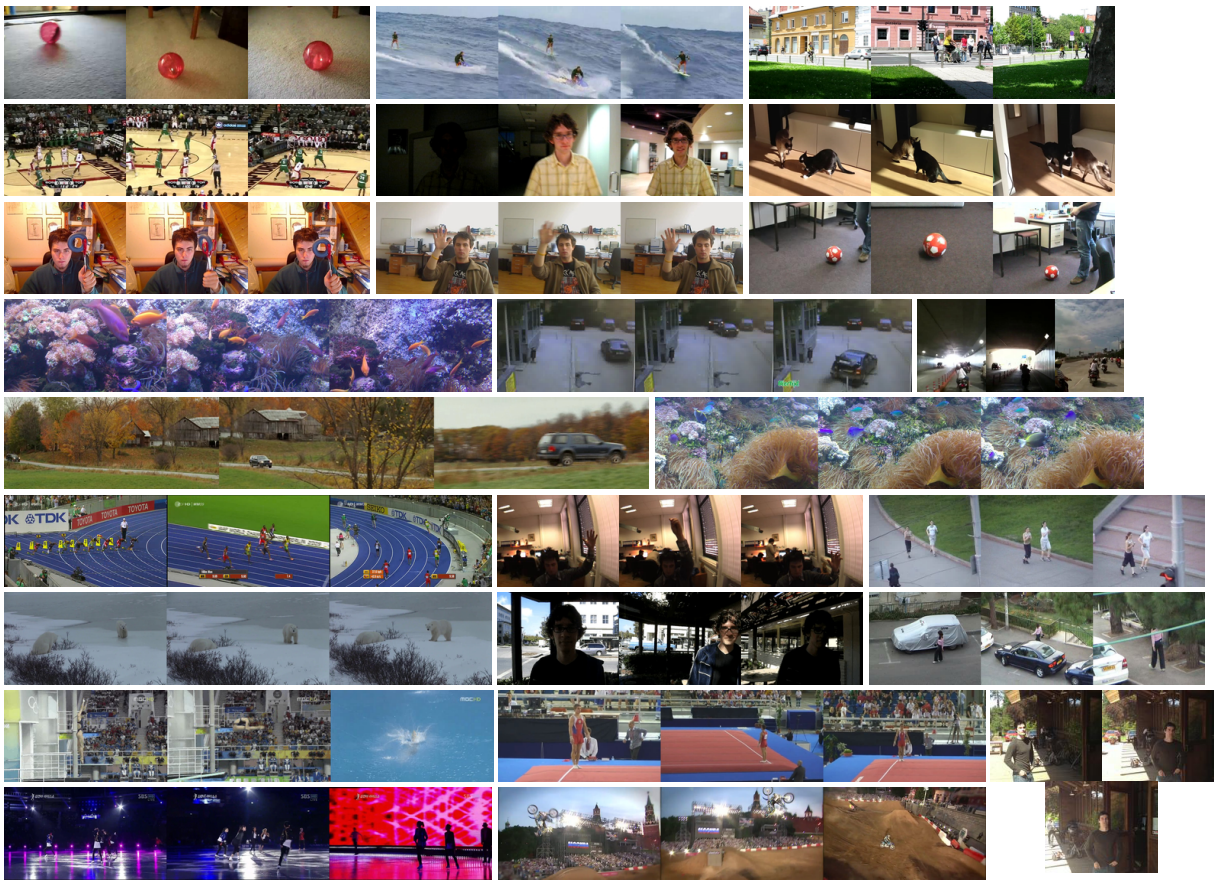

Figure 2: Examples of VOT2014 dataset sequences. From the top-left: *01-Light_video00023*, *03-Specularity_video00003*, *bicycle*, *basketball*, *david*, *fernando*, *torus*, *hand2*, *Vid_A_ball*, *fish4*, *drunk2*, *01-Light_video00022*, *CarScale*, *fish3*, *bolt*, *hand*, *Jogging*, *10-LowContrast_video00013*, *trellis*, *woman*, *diving*, *gymnastics*, *sunshade*, *skating1*, *motocross1*. For each sequence we show three images.

### 3.1 Target annotation

The relevant object in each sequence is manually annotated by placing a bounding box over the object in each frame. Since the sequences were obtained from the pool of existing tracking sequences, most have come along with the axis-aligned bounding boxes. This means that the bounding boxes were annotated by various authors. It is difficult to specify a common rule that guided the annotators. It appears that the bounding boxes were placed such that large percentage of pixels within the bounding box (at least $> 60\%$) belonged to the target. In most cases, this percentage is quite high since the upright bounding box fits the target well. Some sequences contained elongated targets that were deforming, which made axis-aligned bounding box a rather poor approximation of the target location in the image – in some extreme cases, the bounding box contained a significant portion of the background as well. To address this, we have manually re-annotated these sequences with a rotated bounding box. The guideline for annotation with a rotated bounding box was reducing the number of background pixels (to approximately less than $30\%$). For an example of re-annotated frame, pleases see Figure 3.

## 4 Performance measures

The tracking performance in the VOT2014 challenge will be primarily measured by two performance measures: (i) accuracy and (ii) robustness as defined in [3, 4]. In accuracy we are interested in how well the (rotated)

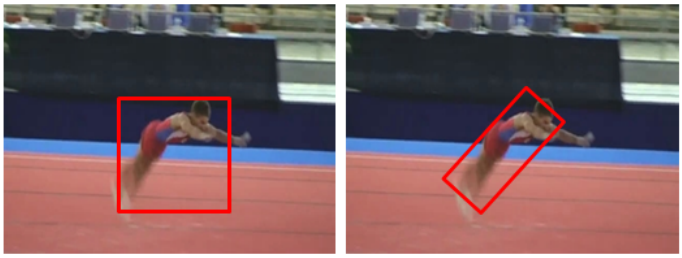

Figure 3: Example of an axis-aligned bounding box annotation (a) and re-annotation for the VOT2014 dataset (b).

bounding box predicted by the tracker overlaps with the (rotated) ground truth bounding box. On the other hand, in the robustness we are primarily interested in how many times the tracker loses the target during tracking. We will use the methodology developed in VOT2013 challenge [4] to compare the trackers. Apart from accuracy and robustness, we will also measure the speed of trackers. The evaluation kit will attempt to address the fact that speed largely depends on hardware, however, since we cannot control for programming languages and implementation skills, these measurements will only be used for reference.

# 5 Experiments

The challenge includes the following two experiments:

- Experiment 1: This experiment runs a tracker on all sequences in the dataset from Section 3 by initializing it on the ground truth bounding boxes.
- Experiment 2: This experiment performs Experiment 1, but initialize with noisy bounding box. By noisy bounding box, we mean a randomly perturbed bounding box, where the perturbation is in the order of ten percent of the ground truth bounding box size.

All the experiments will be automatically performed by the evaluation kit from Section 6. A tracker will be run on each sequence multiple times to obtain a better statistic on the performance. In particular, each experiment will be repeated 15 times by each tracker. Note that it does not make sense to perform Experiment 1 multiple times for the deterministic trackers. In this case, the evaluation kit (Section 6) will automatically detect whether the tracker is deterministic and will reduce the number of repetitions accordingly.

# 6 VOT2014 evaluation kit – overview and instructions

This part of the document describes the structure of the VOT2014 evaluation kit (also called the VOT2014 toolkit) code and its basic usage. Note that in order to perform an evaluation you must have your tracker adapted to behave in a specified way that is described in Section 7. For a detailed overview of the structure check out Section 8.

## 6.1 Terminology

For better understanding we begin with a definition of important terms:

- *Tracker* - An executable or a script containing a tracking algorithm that is evaluated.
- *Sequence* - A single sequence of images with manually annotated ground-truth positions of the object used for evaluation.
- *Annotation* - Manual or automatic description of visual position of the object.
- *Trajectory* - A sequence of annotations that describes the motion of the object in the sequence. In the toolkit it is saved to a file in a text format.
- *Tracker run* - A single execution of a tracker on an entire sequence or its subset.
- *Trial* - If a tracker fails during tracking it is reinitialized after the first point of failure.
- *Repetition* - To properly address the potential stochastic nature of the algorithm, several trials are performed on each sequence.
- *Experiment* - Evaluation of a tracker on a set of sequences in specific conditions. Specific experiment may change the original sequences to simulate some specific kind of circumstances (e.g. image noise, initialization error).

- *Evaluation* - Performing a set of experiments on a specific tracker.
- *Region overlap* - Overlap distance between two regions.

## 6.2 Platform support

The toolkit is written in a pure Matlab language with a strong emphasis on Octave compatibility as well as support for multiple versions of Matlab (at the moment it was tested with versions from 2011 to 2013).

The code should work on Windows, Linux and OSX, however, it is hard to verify this on all versions and system configurations. If you therefore find that there are some issues on your computer setup, submit a bug report at Github as soon as possible.

## 6.3 Downloading resources

A snapshot of the toolkit code is available at the VOT website, however, it can also be downloaded from the official Github repository, where the toolkit is being collaboratively developed.

By default the sequences are automatically downloaded by the toolkit, however, if you have to manually retrieve them check the link to the bundle on the VOT challenge website.

## 6.4 Set up the toolkit

A quick way to get the VOT evaluation up and running requires the steps below. The more comprehensive steps are also described in details in the following subsections.

1. Make sure that you have all the prerequisites (e.g. Matlab or Octave) and that your MEX build environment is set up.
2. Download the toolkit code and place it to an empty *toolkit directory*. Add at least the root directory of the toolkit to the Matlab/Octave search path.
3. Compile one of the sample trackers in the `tracker/examples` subdirectory of the toolkit (in the case of the Matlab example no compilation is necessary).
4. Create a *working directory* by creating a directory, moving to it in Matlab/Octave shell and executing `vot_initialize`. A working directory is a directory where the toolkit stores the sequences, results and cached data.
5. Configure the workspace by configuring the generated files in working directory (e.g. the `configuration.m` file). Configure your tracker by setting the generated tracker configuration file. For more information about that check the Integration chapter.
6. Test the tracker integration by executing `run_test` script. The first time that the you initialize the environment the script will take a while to execute because it will download the sequence data and compile the native functions.
7. Run the evaluation by executing `run_experiments`.

## 6.5 Perform tracker evaluation

The entire evaluation is currently performed by executing the generated `run_experiments` script in Matlab or Octave. This script performs all the experiments, collects and analyzes result data and creates a result package that is ready for submission.

The time required for the evaluation depends on the speed of your tracker. Because of the rigorous testing methodology the entire evaluation can take up to several days. For a more detailed explanation of the testing procedure consult the Internals guide.

## 6.6 View and submit the results

At the end of the evaluation the toolkit generates a ZIP file containing all the raw results and an overview HTML document of calculated result scores. Note that this requires having ZIP utilities installed on your computer. We strongly encourage you to submit the automatically generated archive so that we can use a script to process a pre-defined structure of the data. Alternatively, the author will have to manually generate the zip file containing the results.

In order to publish the results to the VOT on-line result repository, the ZIP file has to be submitted to the repository maintainers along with a short description of the tracker (consult the on-line instructions at the VOT website).

**Note on meta-data acquisition**: In order to get some technical statistics that will help us improve the evaluation methodology and the toolkit itself the generated ZIP file contains certain information regarding the software and hardware specifications of the computer that the experiments were performed on. These information range from the information about the operating system type to Matlab/Octave version. All the aggregated information about your computer can be verified prior to submission by checking the `manifest.txt` file in the corresponding ZIP file.

### 6.7 Tips and tricks

- *Parallel execution* - A full evaluation can be extremely time-consuming. At the moment the toolkit does not support parallel execution, however, it is possible to execute parts of the evaluation in parallel on multiple computers or on a single multi-core computer with a bit of manual work. Two parallelization strategies are described in the Internals chapter.
- *Clearing results* - By default the toolkit caches already calculated trials, so that the evaluation can be stopped at any time and resumed later with as little lost data as possible. If you, however, wish to clear the already stored data there are two options. The first one is to disable caching parameter (using `set_global_variable↩ ('cache', 0)`) in your workspace `configuration.m`. The second option is to delete the appropriate directory in the results subdirectory of your working directory. This subdirectory has the same name as your tracker, so make sure that you do not delete the wrong one if you are evaluating multiple trackers.
- *Measuring execution time* - One of the aspects of trackers performance that is being measured in the evaluation is also execution time. Of course this depends on the hardware, however, some conclusions can also be drawn from these estimates. To make them as reliable as possible it is advisable to perform experiments on a single computer or multiple computers with same the hardware specifications. It is also advisable that the computer is not being used extensively during the evaluation. At the end of the experiments the toolkit will also perform several simple benchmarks that will be included in the submission archive. These results will help us generate more reliable computational performance comparisons.

### 6.8 Additional configuration

There are several non-essential parameters available that can be set in the `configuration.m` file.

- `debug` sets the debug output option (disabled by default). Using this option it is possible to get additional information regarding the progress of the evaluation, however, its usage is mainly indented for development purposes.
- `cache` sets the result caching (enabled by default). By disabling this option all the results are generated every time the evaluation is run instead of preserving results for trials that were already successfully executed.
- `pack` sets the result packaging (enabled by default). If enabled, the results of the evaluation are packed into a ZIP archive at the end of the evaluation. This file is ready to be submitted to the VOT on-line result repository.

## 7 Tracker integration

This file describes the process of tracker integration into the VOT toolkit evaluation environment. When designing the integration interface the main goal was that the adaptation of a tracker should be as simple as possible for as many programming languages and operating systems as possible and that the separation between the evaluation logic and the tracker should be very clear.

With the current version of the toolkit there are now two ways of integrating a tracker to the VOT toolkit evaluation environment. There is the old way of communication using files and the new way of using the TraX protocol. The recommended and default way of integration with 2014 challenge is the TraX protocol, however, it requires some extra work. The benefit is faster execution of experiments and more flexibility.

### 7.1 TraX protocol

Tracking eXchange protocol is a simple protocol that enables easier evaluation of computer vision tracking algorithms. The basic idea is that a tracker communicates with the evaluation software using a set of textual commands

over the standard input/output streams of each process.

Integration of TraX protocol into a C/C++ or Matlab tracker is quite simple as there are examples available in the repository of the reference implementation.

VOT toolkit is now tightly connected to the TraX reference implementation as it shares some native code with it. For the execution the toolkit uses a `traxclient` tool to execute a tracker. The tool acts as a wrapper that communicates with the tracker and reinitializes it when required. While the TraX source code is automatically downloaded by the toolkit (to the `trax` subdirectory in the toolkit source), the client executable as well as a native TraX library have to be compiled manually using CMake and an appropriate compiler for your platform (we are working on automating this process, however, there are more urgent things that we have to do before). Once the client is compiled you have to specify the path to the executable in your workspace by adding the following line to the `configuration.m`:

```
set_global_variable('trax_client', '<TODO: full path to the executable>');
```

## 7.2 File protocol

A tracker is written as a GUI-less executable or a script that receives its input data from a known location on the hard-drive and outputs its result to a known location on the hard-drive and then terminates. The limitation of this approach is that only bounding boxes may be used while the TraX protocol also supports more complex region descriptions.

Every time the tracker is run, the toolkit generates a new temporary directory and populates it with the input data. The tracker is then executed in this (working) directory.

**Input**: There are two input files available in the working directory:

- `images.txt` - A text file containing a new-line separated list of absolute file paths. Each line therefore determines an image file on the hard-drive. Images are in JPEG format. Note that no assumptions should be made regarding the names of the files. The only legal order of the images is the one provided in the file.
- `region.txt` - A text file containing four comma-separated values that denote the left,top coordinate of the initial object bounding-box as well as its width and height.

**Output**: In order to consider a tracker execution as successful the tracker has to produce a `output.txt` file before it exits. This file should contain a per-line list of object's regions that correspond to the appropriate frames in the input image sequence. Each region is encoded as a comma-separated list. There are two region formats available, a rectangle (four values) and a polygon (even number of six or more values), for details about both please check out the Internals chapter.

Note that each line corresponds to a single frame and only a single region should be outputted by the tracker for each frame. Also note that the tracker is required to produce a valid region for each frame and it is up to the tracker to decide how to predict that frame in case it does not detect any target. The most naive output in such situation, for example, would be setting the bounding box to the size of the entire image with its center corresponding to the center of the image.

For trackers that are compiled into a binary executable form languages, such as C or C++ the integration is simple. The tracker should be able to read and write text files as described in the previous section. Several examples of such trackers are provided with the toolkit (in the `tracker/examples` directory together with some utility code that can be copied to ones code to ease the adaptation process.

## 7.3 Tracker identifier and configuration file

Each tracker has a unique identifier, i.e. a string that is used to identify that particular tracker within the evaluation system. Because of the simplicity there are certain limitations to the type of characters that can be used in the identifier. Only English alphabet characters (lower-case and upper-case, but preferred lower-case), digits, and dash are allowed.

The properties of each tracker are defined in a configuration script that is named `tracker_<tracker_identifier>.m`. Within this script the important properties of the tracker are set:

- `tracker_command` - The executable command that is used to invoke the tracker program. Contains the full path to the executable and optional input arguments.

- `tracker_label` - An optional string that is used to identify the tracker in visualizations. There are no restrictions to the format of the tracker label, but please try to keep it similar to the tracker identifier. If no value is given, the identifier is used instead.
- `tracker_linkpath` - An optional array of additional search-paths to be set before executing the tracker.
- `tracker_trax` - An optional logical flag. If the variable exists and is true then the tracker is using the TraX protocol and will be executed as such.

## 7.4 Binary trackers

To register a binary tracker in the environment, simply set the `tracker_command` variable value in the `tracker_<tracker_identifier>.m` to the full absolute path to the executable (optionally together with required parameters if the tracker requires some).

**Linking problems**: In some cases the executable requires access to some additional libraries, found in non-standard directories. Matlab overrides the default linking path environmental variable, which can cause linking problems in some cases. For this we have introduced a `tracker_linkpath` variable. This variable should be a cell-array of all directories that should be included in the linking path. An example below adds two custom directories to the library path list in Linux:

```
tracker_linkpath = {'/usr/lib64/qt4/', '/usr/lib64/opencv/'};
```

## 7.5 Matlab trackers

Matlab-based trackers are a bit more tricky to integrate as the scripts are typically run in an integrated development environment. In order to integrate a Matlab tracker into the evaluation, a wrapper function has to be created. This function will usually read the input files, process it and write the output data. In case of the old integration approach is it is important that the `exit` command is called at the end in order to terminate Matlab interpreter completely. This is very important as the toolkit waits for the tracker executable to stop before it continues with the evaluation of the generated results. Another issue that has to be addressed is the user-issued termination. When a `Ctrl+C` command is issued during the `system` call the command is forwarded to the child process. Because of this the child Matlab will break the execution and return to interactive mode. In order to tell Matlab to quit in this case we can use the onCleanup function which also addresses the normal termination scenario:

```
function wrapper()
    cleanup = onCleanup(@() exit() ); % Tell Matlab to exit once the function exits
    ... tracking code ...
```

For an example of integration please check out the Matlab tracker example in the `examples` directory. Note that this precautions are not needed in the new system as the TraX client takes care of termination (it can also stop a tracker process if a timeout is reached).

When specifying the `tracker_command` variable in the tracker configuration file please note that the wrapper script file is not the one being executed but only a parameter to the Matlab executable. The actual command therefore looks like this:

```
tracker_command = '<TODO: path to Matlab executable> -nodesktop -nosplash [-wait] -r <TODO: name of the wrapper script>';
```

*Windows specific parameters*: The parameter `-wait` forces Matlab to wait for the script that was executed to finish before returning control to the toolkit and is required when running Matlab trackers on Windows.

It is important that all the directories containing required Matlab scripts are contained in the MATLAB path when the evaluation is run. Also note that any unhandled exception thrown in the script will result in Matlab breaking to interactive mode and that this will prevent the evaluation from continuing. It is therefore advised that all exceptions are handled explicitly so that the wrapper script always terminates the interpreter.

## 7.6 Integration rules

To make the tracker evaluation fair we list several rules that you should be aware of:

- *Stochastic processes* - Many trackers use pseudo-random sampling at certain parts of the algorithm. To properly evaluate such trackers the random seed should not be fixed to a certain value. The best way to

ensure this is to initialize seed with a different value every time, for example using current time. In C this is done by calling `srandom(time(NULL))` at the beginning of the program, while one way of doing this in Matlab is by calling:

```
RandStream.setGlobalStream(RandStream('mt19937ar', 'Seed', sum(clock)));
```

- *Image stream* - The tracking scenario specifies input images as a stream. Therefore the tracker should always only access images in the specified order and not skip ahead.
- *Tracker parameters* - The tracker is supposed to be executed with the same set of parameters on all the sequences. Any effort to determine the parameter values that were pre-tuned to a specific challenge sequence from the given images is prohibited.
- *Resources access* - The tracker program should only access the files in the directory that it is executed in.

While we cannot enforce these guidelines in the current toolkit, the adherence of these rules is mandatory. Any violation is considered as cheating and could result in disqualification from the challenge.

### 7.7 Testing integration

It is not recommended to immediately run the entire evaluation without testing the integration on a simpler task. For this the toolkit provides the `vot_test` function that provides an interactive environment for testing your tracker on various sequences.

Using this environment you can verify the correct interpretation of input and output data (at the moment the interactive visualization only works in Matlab) as well as estimate the entire evaluation time based on several runs of the tracker on various sequences (run the tracker on several sequences, then select the option to display required estimate).

## 8 Details about the toolkit

This document contains a detailed description of the several internal mechanisms of the evaluation toolkit.

### 8.1 Working directory

The working directory is automatically populated with several subdirectories:

- `sequences/` - This subdirectory contains sequence data. By default the testing sequence dataset will be automatically downloaded the first time the evaluation starts in a specific working directory. Alternatively you can download the entire dataset manually and extract it to this directory. Each sequence is contained in a separate directory with the following structure:
  - `<sequence name>/` - A directory that contains a sequence.
    - `groundtruth.txt` - Annotations file.
    - 00000001.jpg
    - 00000002.jpg
    - 00000003.jpg
    - ...
- `results/` - Results of the tracking for multiple trackers.
  - `<tracker identifier>/` - All results for a specific tracker.
    - `<experiment name>/` - All results for a specific experiment.
    - `<sequence name>/` - All results for a sequence.
      - `<sequence name>_<iteration>.txt` - Result data for iteration.
- `cache/` - Cached data that can be deleted and can be generated on demand. An example of this are gray-scale sequences that are generated from their color originals on demand.

## 8.2 Evaluation execution

By default the entire evaluation is performed sequentially as described by the following pseudo-code:

```
tracker t
for experiment e:
    for sequence s:
        repeat r times (if tracker is stochastic):
            perform trial for (t, s)
        end
    end
end
```

Each trial contains one or more executions of the tracker. The idea is that if the tracker fails during tracking the execution (the failure criterion can be experiment dependent) it is repeated from the point of the failure (plus additional offset frames if specified).

In the case of stochastic trackers, each sequence is evaluated multiple times. If the tracker produces identical trajectories two times in a row, the tracker is considered deterministic and further iterations are omitted. It is therefore important that the stochastic nature of a tracker is appropriately addressed (proper random seed initialization).

Because of the thorough methodology, the entire execution time can be quite long. The function `vot_test` provides an option for estimating the processing time for the entire evaluation based on a single run on one sequence. Using the new integration approach the execution time has been reduced especially for trackers that have to be reinitialized a lot.

## 8.3 Parallelism

To speed up the execution, the evaluation can be parallelized. Due to simplicity, the toolkit does not support parallel execution explicitly, however, it is possible to execute individual experiments in parallel on multiple computers or on a single multi-core computer with a bit of manual work, thus reducing the evaluation time. The other, more complicated option is to separate the execution by sequence dataset partitioning.

**Parallelize by experiment**: To separate execution of the evaluation on a single multi-core computer simply run multiple instances of the interpreter (Matlab or Octave). For each instance write a separate `run_experiment_<N>.m` script file where only one experiment is given to the `vot_experiments` function. Disable result package creation (using `set_global_variable('pack', 0)`). Then execute the evaluation in each shell by calling `run_experiment_<N>`. After all three experiments are done, re-enable result package creation, clear the variables, and call the script that runs all the experiments and creates a results package in one of the interactive shells (as the results are already cached, this will take almost no time).

To separate execution on multiple computers more manual work is needed. On each computer configure the toolkit, run an interpreter (Matlab or Octave), and proceed in similar manner than with the multi-core computer. Note that by default sequences will be downloaded on each computer, which can be avoided by copying the initialized workspace from one computer to the rest. The results have to be manually merged on a single computer by copying the result data for each experiment in the `results` directory. Only then run a script that performs packaging. Because the system also collects some hardware performance information it is not desired to run the tracker on computers with different hardware configuration as this will corrupt the performance measurements.

**Parallelize by dataset partitioning**: Another option to speed up the evaluation is to form dataset partitioning. The process is similar to the one above, we are just splitting the set of sequences this time. Other instructions (disabling package creation and result merging for multi-computer setup) are the same as with the parallelization by experiment.

## 8.4 Trajectory format

The stored output of the tracker (the final combined trajectory) is encoded as text file where a region for each frame is encoded as comma-separated list. Currently there are three types of region formats that are supported by the system.

- **Rectangle** - Specified by four values: `left`, `top`, `width`, and `height`.
- **Polygon** - Specified by even number of at least six values that define points in the polygon (`x` and `y` coordinates).
- **Special**: This stored sequence describes the entire tracking trial process with failures and re-initializations encoded between regular frames in a special format that is specified by a single value. This value can have

a special meaning. Initialization of the tracker is denoted by `1`, failure of the tracker is denoted by `2` and undefined state (e.g. due to frame skipping) is denoted by `0`.

### 8.5 Results bundle

When the evaluation is complete the data is bundled in a zip file that can be used to submit your results to the VOT Challenge website. The zip file contains raw tracking results from the `results` directory together with some tracker and platform meta-data. The structure of the data is the same as in the `results` directory (contains the folder for a tracker as well as the generated `performance.txt` file).

## References


[1] A. W. M. Smeulders, D. M. Chu, R. Cucchiara, S. Calderara, A. Dehghan, M. Shah, Visual tracking: an experimental survey, IEEE Trans. Pattern Anal. Mach. Intell.

[2] Y. Wu, J. Lim, M.-H. Yang, Online object tracking: A benchmark, in: Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on, 2013, pp. 2411–2418. doi:10.1109/CVPR.2013.312.

[3] L. Čehovin, M. Kristan, A. Leonardis, Is my new tracker really better than yours?, in: IEEE Winter Conference on Applications of Computer Vision, WACV2013, 2014.

[4] M. Kristan, R. Pflugfelder, A. Leonardis, J. Matas, F. Porikli, L. Čehovin, G. Nebehay, G. Fernandez, T. Vojir, A. Gatt, A. Khajenezhad, A. Salahledin, A. Soltani-Farani, A. Zarezade, A. Petrosino, A. Milton, B. Bozorgtabar, B. Li, C. S. Chan, C. Heng, D. Ward, D. Kearney, D. Monekosso, H. C. Karaimer, H. R. Rabiee, J. Zhu, J. Gao, J. Xiao, J. Zhang, J. Xing, K. Huang, K. Lebeda, L. Cao, M. E. Maresca, M. K. Lim, M. E. Helw, M. Felsberg, P. Remagnino, R. Bowden, R. Goecke, R. Stolkin, S. Y. Lim, S. Maher, S. Poullot, S. Wong, S. Satoh, W. Chen, W. Hu, X. Zhang, Y. Li, Z. Niu, The visual object tracking vot2013 challenge results, in: ICCV2013 Workshops, VOT2013, 2013, pp. 98–111.