Visual Object Tracking Challenge (VOT2013) Evaluation Kit

Matej Kristan, Luka Čehovin

July 12, 2013

Contents

1 The challenge

The goal of the VOT2013 challenge is to reliably track a single moving object in realistic scenes subject to various common conditions. The VOT2013 challenge provides an evaluation kit and the dataset for performing objective evaluation of the trackers. The authors attending the VOT2013 challenge are required to download:

- The VOT2013 evaluation kit.
- The VOT2013 dataset.

They have to integrate their tracker into the VOT2013 evaluation kit, which will automatically perform a standardized experiment on their tracker. The results will then be evaluated under a common methodology proposed by the VOT2013 challenge. The kit automatically generates the results in a zip file as well as in the form of a table. The table should be added to the *"The VOT2013 Challenge Results and Short Description Page"* document describing the tracker. This document, along with the results zip file should be submitted through the VOT2013 submission page. Note that in order to attend the VOT2013 challenge, the submission of the results zip file and the *"The VOT2013 Challenge Results and Short Description Page"* document (a template is available here) is mandatory. The authors may also choose to submit the binaries of their tracker and/or the source code. The binaries will be used by the VOT2013 committee to verify the submitted results. If the authors chose, the binaries will be deleted by the VOT2013 committee after the completion of the verification. For more details on the participation, please refer to the VOT2013 participation page.

In the remainder of this document we first make a note on the causality property that is expected in the participating trackers. Then we briefly describe the dataset in Section 2, the VOT2013 performance measures in Section 3, and we describe the experiments in Section 4. Sections 5, 6 and 7 are dedicated to describe the details of the VOT2013 evaluation kit.

1.1 A note on causality

The VOT2013 challenge is aimed at single-object trackers. The trackers participating in the challenge must be causal and must provide a complete reinitialization at frame t. 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.

1.2 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's 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.

1.3 A note on the output

The tracker is required to produce a single bounding box per frame in the sequence. For a particular format of the output, please refer to Section 6.1. If a tracker does not detect a target in a particular frame, then it is up to the tracker to decide how to predict its bounding box. 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. 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 3). 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 Dataset

The tracking evaluation will be conducted on a dataset provided by the VOT2013 challenge. Our main criteria for dataset selection was that the dataset should sufficiently well represent various realistic scenes and conditions,

Table 1: An overview of the experimental sequences.

Name	number of frames	Description	Notes
bicycle [8]	271	bike, occlusion, scale change	
bolt [10]	350	body, articulation, scale change	
car	374	car	Based on *PETS* 2001 challenge data
cup [3]	303	cluttered background	sequence was shortened for the challenge
david [6]	770	head, illumination and scale change	
diving [4]	231	body, articulated, rotation	
face [1]	415	head, occlusion	sequence was shortened for the challenge
gymnastics [7]	207	body, articulated, scale change	
hand [7]	244	hand, articulated	
iceskater [9]	500	body, articulated	sequence was shortened for the challenge
juice [3]	404	box	
jump [2]	228	bike, scale change	
singer [5]	351	body, illumination and scale change	
sunshade	172	head, illumination change	
torus [7]	264	interesting geometry	
woman [1]	597	body, occlusion, scale change	

while it should not contain too many sequences to keep the time required to conduct an experimental evaluation for an average tracker reasonably low. The dataset was prepared as follows. We have collected a large pool of the sequences that have been used by various authors in the tracking community. Each sequence was annotated by several attributes and subset of sequences (sixteen) was selected from this pool such that the various visual attributes such as occlusion, illumination changes, etc., were still represented well within the selection. We present the basic characteristics of the selected sequences in Table 1 and show some examples in Figure 1.

The relevant target 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 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 snugly fits the target. But in some cases, (e.g., gymnastics) where an elongated target is rotating significantly, the bounding box contains a larger portion of the background at some frames.

3 Performance measures

The tracking performance in the VOT2013 challenge will be primarily measured by two performance measures: (i) accuracy and (ii) robustness. In accuracy we are interested in how well the bounding box predicted by the tracker overlaps with the 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.

3.1 Accuracy

We define the tracker's accuracy at time-step t as the overlap between the tracker's predicted bounding box A_t^T and the ground truth bounding box A_t^G:

$$\Phi(\Lambda^G, \Lambda^T) = \{\phi_t\}_{t=1}^N \ , \ \phi_t = \frac{A_t^G \cap A_t^T}{A_t^G \cup A_t^T}. \tag{1}$$

A nice property of the overlap measure is that it accounts for both position and size of the predicted and ground-truth bounding boxes simultaneously, and does not result in arbitrary large errors at tracking failures. In fact, once the tracker drifts to the background, the measure becomes zero, regardless of how far from the target the tracker is currently located. The overlap measure is summarized over an entire sequence by an *average overlap* over the *valid frames*.

Note that all frames are not valid for computation of the accuracy measure. In fact, the overlaps in the frames right after initialization are biased toward higher overlaps since the (noise-free) initialization starts at maximum overlap and it takes a few frames of the burn-in period for the performance to become unbiased by the initialization. In a preliminary study we have determined by a large-scale experiment that the burn-in period is approximately ten frames. This means that ten frames after initialization will be labeled as invalid for accuracy computation.

Figure 1: An overview of dataset sequences. From the top-left: *bicycle*, *bolt*, *car*, *cup*, *david*, *diving*, *face*, *gymnastics*, *hand*, *iceskater*, *juice*, *jump*, *singer*, *sunshade*, *torus*, and *woman*. For each sequence we show three images.

3.2 Robustness

The tracker's robustness in a given sequence will be evaluated by the failure rate. The failure rate measure casts the tracking problem as a supervised system in which an operator reinitializes the tracker once it fails. An equivalent of the number of required manual interventions per sequence is recorded and used as a comparative score. This measure also reflects the tracker's performance in a real-world situation in which the human operator supervises the tracker and corrects its errors. A failure is detected once the overlap (1) measure drops to the value zero. Due to simplicity of the experimental kit, many failures will result in longer evaluation time. To address this issue, the tracker will not be reinitialized immediately after the failure, but a few frames later. In our preliminary study we have determined that dropping five frames does not hamper the evaluation of the robustness.

4 Experiments

The challenge includes the following three experiments:

- Experiment 1: This experiment runs a tracker on all sequences in the dataset from Section 2 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 order of ten percent of the ground truth bounding box size.
- Experiment 3: This experiment performs the Experiment 1 on all sequences with the color images changed to grayscale.

In Experiment 2 there will be randomness in the initialization of the trackers. To make the experiment reproducible, we will generate in advance for each frame in the sequence several possible random initialization boxes (one per experiment repetition). This is because the tracker can fail at any frame and there will be multiple runs of the experiment. The perturbations on position and size will be sampled uniformly from $[-0.1, 0.1]$ distortion of the ground truth bounding box.

Trackers that do not use the color information can be also run only on Experiment 3 and the same results will be assumed also for Experiment 1.

All the experiments will be automatically performed by the evaluation kit from Section 5. 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 and 3 multiple times for the deterministic trackers. In this case, the computation time for the deterministic trackers can be lowered for Experiments 1 and 2. The evaluation kit from Section 5 aims to address this.

5 Overview and usage instructions

This section describes the structure of the VOT evaluation kit (also called the VOT toolkit) code and its basic usage. Note that in order to to perform an evaluation you must have your tracker adapted to behave in a specified way that is described in Section 6. For a detailed overview of the toolkit structure move to Section 7.

5.1 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. In the current toolkit the only type of supported annotation is a bounding-box, described as a series of four values denoting the left-top corner of the rectangle as well as its width and height.
- *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 CSV format.
- *Tracker run* - A single execution of a tracker on an entire sequence or its subset. The tracker gets input information in terms of a list of images and an initial position of the object and produces a sequence of positions in the following images.
- *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. noise).
- *Region overlap* - Overlap distance between two regions, as described in Section 3.

5.2 Set up the toolkit

A quick way to get the VOT2013 evaluation up and running requires the steps below. The more comprehensive steps are also described in details in the following subsections.

1. Download the toolkit code and place it to an empty *toolkit directory*. See Subsection 5.3 for details.
2. Compile one of the sample trackers in the `examples` subdirectory of the toolkit (in the case of the Matlab example no compilation is necessary).
3. Create a *working directory*. A working directory is a directory where the toolkit stores sequences, results and cached data.

4. Configure the toolkit by creating `configuration.m` file in the *toolkit directory*. See Subsection 5.6 for details.
5. Open Matlab or Octave environment and move to *toolkit directory*.
6. Test the toolkit by executing `do_test`. See Subsection 6.5 for details.
7. Run the evaluation by executing `do_experiments`. See Subsection 5.7 for details.

5.3 Downloading

A snapshot of the toolkit code is available at the VOT2013 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, a link to the sequence bundle is also provided on the VOT2013 website.

5.4 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 was extensively tested under Windows, Linux and OSX, however, it is difficult 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.

5.5 Directory structure

The code and the documents are organized in the following directory structure:

- `documentation` - Contains documentation files
- `evaluation` - Contains the evaluation toolkit
- `examples` - Contains example source code in various languages
- `templates` - Contains templates for writing a supplementary description of the tracker

The evaluation source code resides in the `evaluation` directory and is structured into several subdirectories.

5.6 Basic configuration

In order to set up the toolkit, you have to copy the `configuration_template.m` to `configuration.m` and edit it to set the required variables.

Working directory: The first variable that has to be set is an absolute path to a working directory. It is recommended that the directory is empty before the first use, however, it can be used for multiple trackers later on. For more information regarding the directory structure consult Section 7.

```
track_properties.directory = '<TODO: set a working directory>';
```

Tracker: The evaluated tracker has to be configured as well. This is done by setting an unique tracker name (an abbreviated version, used to determine the tracker at the VOT2013 on-line result repository) as well as the exact executable command (consult the Section 6 for details).

```
tracker_identifier = '<TODO: set a tracker identifier>';

tracker_command = '<TODO: set a tracker executable command>';
```

It is recommended that you test the evaluation with one of the example trackers, contained in the `examples` directory. To configure the ''static" C tracker, compile `static.c` to executable using a C compiler (on Linux you can use `build.sh` script) and then configure the `tracker_identifier` and `tracker_command` variables appropriately as seen in the example below (using Unix path syntax):

```
tracker_identifier = 'c_example';
tracker_command = '<toolkit directory>/examples/c/static';
```

For the Matlab example, the process does not require compilation, however, the command specification is a bit more complex:

```
tracker_identifier = 'matlab_example';
tracker_command = '<matlab executable> -nodesktop -nosplash -r "addpath(''<toolkit directory>/examples/↩
    matlab''); wrapper"';
```

For more instructions on how to run Matlab trackers (also some specifics for the Windows platform) please check the `integration.md` document.

5.7 Perform tracker evaluation

The entire evaluation is currently performed by executing the `do_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 to the VOT2013 website.

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 Section 7.

5.8 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. Alternatively, the author will have to manually generate the zip file containing the results.

In order to publish the results to the VOT2013 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 VOT2013 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. Note that this data is not required for a valid submission and can be deleted from the file if you deem it necessary.

5.9 Tips and tricks

- *Parallel execution* - A full evaluation can be extremely time-consuming. Due to simplicity, 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 Section 7.
- *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 `track_properties.cache` parameter in your `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.

5.10 Additional configuration

There are several non-essential parameters available in `configuration.m` file.

- `track_properties.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.
- `track_properties.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.
- `track_properties.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 VOT2013 on-line result repository.

6 Integration of trackers

This section describes the process of tracker integration into the VOT2013 evaluation kit 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.

6.1 Basic idea

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.

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 sequence of object's bounding-boxes that correspond to the appropriate frames in the input image sequence. The bounding-box sequence is encoded as a comma-separated list of four values per frame:

```
<left_1>,<top_1>,<width_1>,<height_1>
<left_2>,<top_2>,<width_2>,<height_2>
<left_3>,<top_3>,<width_3>,<height_3>
<left_4>,<top_4>,<width_4>,<height_4>
...
```

Note that each line corresponds to a single frame and only a single bounding box should be outputted by the tracker per each frame. Please not that the tracker is required to produce the bounding box for each frame and it is up to 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.

6.2 Binary trackers

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 VOT2013 toolkit with utility functions that can be copied to ones code to ease the adaptation process.

To register a binary tracker in the environment, simply set the `tracker_command` variable value in the `configuration.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 in the `configuration.m`. 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/'};
```

6.3 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, but more importantly it should call `exit` command 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()
    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.

When specifying the `tracker_command` variable in the 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> [-wait] -nodesktop -nosplash -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.

6.4 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 (that is `images.txt` and `region.txt`).

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.

6.5 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 `do_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).

7 Details about the toolkit

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

7.1 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.

7.2 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 is repeated from the point of the failure (plus additional offset frames). A tracker failure is declared on the first frame where there is no overlap between the ground-truth and the predicted region (the tracker fails if the overlap between the ground-truth and the predicted region is 0).

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. Let's assume that we have a stochastic tracker that requires 0.5 seconds to process a single frame. The average length of a sequence is 350 frames. The tracker performs 15 trials on each sequence, and there are 16 sequences in the dataset. We pessimistically assume that a tracker fails uniformly 8 times throughout the sequence, which approximately amounts to equivalent of 5 re-runs over entire sequence. Based on these assumptions a rule-of-thumb estimate of the time required to perform a single experiment is more than two days. The entire evaluation contains three experiments which results in roughly 7 days. The function `do_test()` provides an option for estimating the processing time for the entire evaluation based on a single run on one sequence.

7.3 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 back to 2-3 days. 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 three instances of the interpreter (Matlab or Octave). Disable result package creation (using `track_properties.pack`). In each of the interactive shells set a variable `selected_experiment` to one of the values from 1 to 3. Then execute the evaluation by calling `do_experiments`. After all three experiments are done, re-enable result package creation, clear the variables, and call `do_experiments` again in one of the interactive shells.

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 re-enable result package creation, clear the variables, and call `do_experiments` again.

Parallelize by dataset partitioning: Another option to speed up the evaluation is to form dataset partitioning. This option requires more manual work, but also offers better options of parallelization.

By default the toolkit assembles a list of sequences by reading the `list.txt` file in the `sequences` directory. This file contains a list of all the sequences in the evaluation. To split the execution into multiple parts prepare files `list1.txt`, `list2.txt` ... `listN.txt` in the `sequences` directory and copy appropriate subsets of sequences from the `list.txt` into each of them. Then open N interpreters (one one or more computers) and set the variable `selected_sequences` to `list<I>.txt` where `I` = 1 ... N on each of the environments before running the `do_experiments` function.

Other instructions (disabling package creation and result merging for multi-computer setup) are the same as with the parallelization by experiment.

7.4 Trajectory format

The tracker has to output a single bounding box per frame. The final output of the tracker (the final combined bounding-box trajectory) is encoded as a comma-separated list of four values per frame:

```
<left_1>,<top_1>,<width_1>,<height_1>
<left_2>,<top_2>,<width_2>,<height_2>
<left_3>,<top_3>,<width_3>,<height_3>
<left_4>,<top_4>,<width_4>,<height_4>
...
```

This stored sequence describes the entire tracking trial process with failures and re-initializations encoded between regular frames in a special format. All irregular frame states have `left`, `top` and `width` values set to `NaN`. The `height` negative values define the type of the special frame:

- Initialization of the tracker: `height = -1`.
- Failure of the tracker: `height = -2`.
- Undefined state due to frame skipping: `height = 0`.

7.5 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 VOT2013 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.

8 Enquiries, Question and Comments

If you have any further enquiries, question, or comments, please refer to the "General enquiries" link (under the News tab) on the VOT2013 homepage.

References

[1] A. Adam, E. Rivlin, and I. Shimshoni. Robust Fragments-based Tracking using the Integral Histogram. In *CVPR*, volume 1, pages 798–805. IEEE Computer Society, June 2006.

[2] M. Godec, P. M. Roth, and H. Bischof. Hough-based tracking of non-rigid objects. In *ICCV*, pages 81–88. IEEE, Nov. 2011.

[3] D. A. Klein, D. Schulz, S. Frintrop, and A. B. Cremers. Adaptive Real-Time Video-Tracking for Arbitrary Objects. In *IROS*, pages 772–777, 2010.

[4] J. Kwon and K. Lee. Tracking of a non-rigid object via patch-based dynamic appearance modeling and adaptive basin hopping monte carlo sampling. In *CVPR*, pages 1208–1215, 2009.

[5] J. Kwon and K. M. Lee. Visual tracking decomposition. In *CVPR*, pages 1269–1276. IEEE, June 2010.

[6] D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang. Incremental Learning for Robust Visual Tracking. *IJCV*, 77(1-3):125–141, May 2008.

[7] L. Čehovin, M. Kristan, and A. Leonardis. An adaptive coupled-layer visual model for robust visual tracking. In *ICCV*, Barcelona, 2011.

[8] L. Čehovin, M. Kristan, and A. Leonardis. Robust Visual Tracking using an Adaptive Coupled-layer Visual Model. *TPAMI*, 35(4):941–953, Apr. 2013.

[9] T. Vojíř, J. Noskova, and J. Matas. Robust Scale-Adaptive Mean-Shift for Tracking. In J.-K. Kämäräinen and M. Koskela, editors, *Image Analysis, Lecture Notes in Computer Science*, pages 652–663. Springer Berlin Heidelberg, 2013.

[10] M.-H. Yang, H. Lu, and F. Yang. Superpixel tracking. In *ICCV*, pages 1323–1330. IEEE, Nov. 2011.