IEEE Second International Conference on Data Stream Mining & Processing
August 21-25, 2018, Lviv, Ukraine

Software for Visual Insect Tracking Based on F-transform Pattern Matching

Petr Hurtik
IRAFM, CEIT4I University of Ostrava Ostrava, Czech Republic petr.hurtik@osu.cz

David Čížž
Department of informatics and computers University of Ostrava
Ostrava, Czech Republic
davidciz95@gmail.com

Oto Kaláb
Department of Biology and Ecology University of Ostrava
Ostrava, Czech Republic
kalab.oto@gmail.com

David Musiolek
Department of Biology and Ecology University of Ostrava
Ostrava, Czech Republic
david.musiolek@osu.cz

Petr Kočárek
Department of Biology and Ecology University of Ostrava
Ostrava, Czech Republic
petr.kocarek@osu.cz

Martin Tomis
Department of Telecommunications VSB-TU Ostrava
Ostrava, Czech Republic
martin.tomis@osu.cz

***Abstract*—We introduce a problem of tracking small animals, especially insects. To solve this problem, we focus on visual tracking in recorded movies, propose our pattern tracking mechanism based on F-transform, and implement a user-friendly software to handle the movies. The tracking core is compared with five state-of-the-art tracking algorithms: KCF, MIL, TLD, Boosting and MedianFlow from processing time and algorithm failure rate point of views. Based on the results computed from 1000 movie frames, we observed that the proposed F-transform tracking core is the fastest and the most reliable method.**

***Index Terms*—Gryllus Assimilis, insect tracking, visual tracking, F-transform, pattern matching, 4k movie**

I. Introduction

In Zoology field, transmitters placed onto animals are used to track animal paths and therefore help to understand the behavior of the tracked object. In the case of big animals such as wolfs, tigers etc., there are many publications dealing with such tracking, see e.g., [1] and [2]. In those cases, transmitters are small enough to be placed on such big animals without affecting their behavior or ability to move. Problems ensue when biologists want to track insects because the transmitters weight and size can affect the insect's behavior and therefore invalidate the research. In our work, we investigate the impact of various transmitters on insects, namely on a field cricket, *Gryllus assimilis* (Orthoptera), in order to define a safe weight an insect can carry without it influencing its movement. To formulate such statement, we need to record statistically big enough quantity of insects with various transmitters and track their movement. Because of the recorded data size, it is not possible to track them manually, therefore we have to design automated tracking software. The design of such application is the topic and the main goal of this paper. The paper structure is following: at first, we briefly describe visual tracking process in Section II and then we recall state-of-the-art software and algorithms for pattern tracking in Section III. Our own pattern tracking algorithm is described in Section IV. The detailed description of the considered insect tracking task with its obstacles, application design and benchmarking are the aims of Section V. Finally, conclusions are formulated in Section VI.

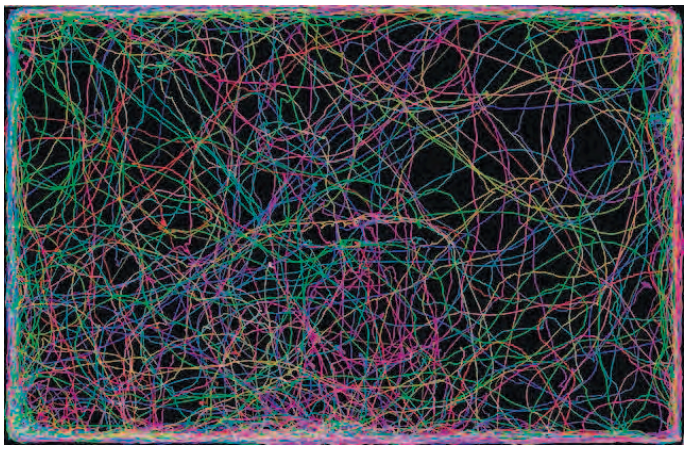

Fig. 1. Illustration of the task goal: tracked movement of insects in an arena.

II. Pattern tracking

Visual pattern tracking (tracing) [3] can be viewed as a special case of pattern matching technique [4]. Generally, pattern matching is searching and checking given object(s) (image where an insect is captured, in our case) for a presence of given patterns (images) in order to find and mark the patterns locations (if any) within the given objects. The matching can be exact or approximate. In the exact one, the pattern appears in its exact form while in the approximate one, it is allowed some freedom between the pattern and its found match. Formally, we expect computer two-dimensional image $f : D \to L$, $D = \{1, 2, \ldots, W\} \times \{1, 2, \ldots, H\}$

978-1-5386-2874-4/18/ $31.00 ©2018 European Union

and $L = \{0, 1, \dots 2^n - 1\}^c$, where W and H is width and height respectively, n denotes considered bit depth (amount of intensities, standardly $n = 8$) and c denotes number of color channels, i.e., $c = 1$ for a grayscale image and $c = 3$ for, e.g., RGB color model. Such image function is a space (usually called as a database) where we search pattern for. As a pattern, we consider image $f' : D' \to L$, which is a proper inclusion of f and $|D'| \leq |D|$ holds. The goal is to browse f and find inside it a sub-area f_s such $DistDist(f_s, f_p)$, where $Dist$ is a metric is minimal for all possible considered sub-areas.

The pattern tracking instead of one image f consider a database of functions $\mathbf{f} = \{f_1, f_3, \dots, f_t\}$, where the image functions are captured in various time moments. As a typical representative, we can mention a movie from a camera decomposed into particular images, called frames. The point is that a frame has not be matched fully in all frames. Because of the images in a database are supposed to be time-ordered and we know previous pattern location, we can search for the pattern in a Δ-neighborhood of the previous pattern location in an actual frame, where Δ is maximal considered pattern movement between frames. Further, because the previous pattern locations are known, we can determine pattern trajectory and approximate it in future frames in order to create a more precise match or to continue with tracking even if the pattern is covered by another object.

III. Existing software and algorithms

In this paper, we will describe own made software for insect tracking, which will be benchmarked. In order to compare the software, we will describe current existing solutions and algorithms in this section.

A. Existing solutions in the form of software

During solving the problem, we searched for a full software dealing with the insect tracking problem at first. We omitted old, unsupported software and handled for different applications. The list of investigated applications is following.

Ctrax[1] is a software specialized in tracking walking flies. Unfortunately, we were not able to read the movie by the application even if it was converted from mp4 into avi file as is stated in the application manual.

Bio-Records[2] is aimed on general insect tracking. We were not able to use the application because it is not available for Windows operating system. The reason why we mention it is it is known in zoology field and because its web page and youtube presentation movies are impressing. On the other hand, even in their short demonstration movie[3] there are visible a lot of cases when insect tracking fails.

Noldus animal tracker[4] At least by the software price and its presentation, it is one-of-the-best software for animal tracking. We tried to obtain free trial version, but we were not successful in a negotiation process with the company.

SwisTrack[5] more than a complex tracking software, it is a package of basic graphical operations which can be stylized by a user into full algorithm serving as a tracker. We faced the same problem as in the case of Ctrax - the software was unable to open our movie, it shows "codec missing" even if we have had installed required codecs according to the reference manual.

Winanalyze[6] is a software for a general tracking such as movement tracking etc. Our experience is the software critically crashed when a movie was opened.

As it is obvious, no one of the investigated software can be used to solve our task. The general problem is the applications are designed for one particular task without enough generality which is caused, e.g., by supporting minimum input file types.

B. Existing algorithms

To be our current state overview complete, we investigated also state-of-the-art pattern tracking algorithms which we used as a core in our implemented application described in Section V-B. The list of tested algorithms is following.

Adaptive color attributes tracker (KCF) [5]: Danelljan et al. propose to improve CSK tracker [6] which compares two bag-of-words of patterns, where words are shapes and colors. The improvement relies on adding more extracted color features. The benefit of the algorithm is it runs in real-time for a reasonable-big (small) movie resolution.

Tracking-Learning-Detection (TLD) [7]: the approach idea is not to accumulate error between initialized pattern and actual one in long-term tracking. This approach is similar to the idea of our algorithm and it realizes three steps: tracking, position improving and pattern re-learning. Pattern position is improved by performing pattern matching while pattern-relearning use actual detected area as the pattern when a distance between the pattern and the searched pattern is big enough.

Online Multiple Instance Learning (MIL) [8]: the algorithm works with a set of patches around the selected pattern, placed into bag-of-words in order to work as a weak classifier [9].

Real-Time Tracking via On-line Boosting [10]: the algorithm is based on the original on-line AdaBoost [11], i.e., it uses a lot of weak classifiers in order to establish a strong one. The benefit of the algorithm is that, at first, it can adapt itself to pattern changes online and, at second, the performance in comparison with the original work is improved because author designed so-called "global weak classifier" which allows updating all selectors in time.

Median Flow [12]: authors use Lucas-Kanade tracker [13] and improve it by tracking trajectory. In opposite to standard algorithms which process ascending frames, Median Flow tracks both ascending and descending frames, search for irregularities in searched locations given by Lucas-Kanade tracker and filter them out.

[1] http://ctrax.sourceforge.net/
[2] http://www.bio-tracking.org
[3] http://www.youtube.com/watch?v=T5W0iplroSg
[4] http://www.noldus.com/animal-behavior-research
[5] https://en.wikibooks.org/wiki/SwisTrack
[6] https://winanalyze.com

IV. F-TRANSFORM PATTERN TRACKING

In this section, we will briefly describe how our tracking algorithm works. It is based on our general pattern matching algorithm [4] and because its speed is superior [14], we are able to follow the idea "tracking by matching" without significant loosing of performance and achieving high precision.

The algorithm is based on the idea of transforming images from their domain into a reduced domain using F-transform [15]. We can define the direct F-transform of a discrete function f of two variables defined on $[1, W] \times [1, H]$. Let $f : P \to \mathbb{R}$ where $P = \{(i, j)|i = 1, ..., W; j = 1, ..., H\}$ and let $\{A_1, \ldots, A_m\} \times \{B_1, \ldots, B_n\}$ establish a fuzzy partition of $[1, W] \times [1, H]$ such that $m < W$, $n < H$ and $\forall k, l\ \exists i, j;\ A_k(i)B_l(j) > 0$. Then the direct F-transform of f w. r. t. the chosen partition is a matrix $\mathbf{F_{mn}}[f] = (F[f]_{kl})$, $k = 1, \ldots, m$, $l = 1, \ldots, n$, of F-transform components where the components are defined for all $k = 1, \ldots, m$, $l = 1, \ldots, n$ and $(i, j) \in P$ as follows

$$F[f]_{kl} = c_{kl}(f \otimes C_{kl}),$$

where $\otimes$ is a convolution, $C_{kl} = A_k B_l$, and

$$c_{kl} = \left(\sum_{i=1}^{W} \sum_{j=1}^{H} C_{kl}(i, j) \right)^{-1}.$$

Let us remark that in the following algorithm, we use the direct F-transform with respect to the h-uniform fuzzy partition of $[1, W] \times [1, H]$. The parameter h influences the number of basic functions in the fuzzy partition and thus also the size of the matrix $\mathbf{F_{mn}}[f] = (F[f]_{kl})$ representing the original function f. Specifically, the larger h the lesser components $F[f]_{kl}$ and thus, the bigger reduction of the original function. The algorithm consists of two phases. Both of them use the same-valued user-defined parameter h. The first one aims to pattern preparing as:

1) Take pattern f_p given on $[1, W_p] \times [1, H_p]$.
2) For f_p create two fuzzy partitions with respect to the same parameter h. The first one notated as $<1>$ is created on $[1, W_p] \times [1, H_p]$, the second one $<2>$ is on $[h/2, W_p] \times [h/2, H_p]$.
3) Compute the F-transform components for f_p, with respect to both fuzzy partitions, i.e., for f_p, we obtain two representations given by matrices $\mathbf{F}[f_p]<1>$ and $\mathbf{F}[f_p]<2>$, of the F-transform components.

The second part realizes the pattern matching as follows:

1) Take a particular movie frame (the actual image) and represent it by a discrete function f_D given on $[1, W_D] \times [1, H_D]$.
2) Create a fuzzy partition of $[1, W_D] \times [1, H_D]$ with respect to the same parameter h given in the first part.
3) Compute the F-transform components of f_D with respect to the fuzzy partition and obtain the matrix $\mathbf{F}[f_D]$.
4) Compare the matrices $\mathbf{F}[f_p]<j>$, $j = 1, 2$, with $\mathbf{F}[f_D]$ by sliding windows comparison by computing distances between components $Dist_i(\mathbf{F}[f_p]<1>, \mathbf{F}[f_D])$ and $Dist(\mathbf{F}[f_p]<2>, \mathbf{F}[f_D])$, and remember the particular position of f_p in f_D where $Dist = Dist(\mathbf{F}[f_p]<1>, \mathbf{F}[f_D]) + Dist(\mathbf{F}[f_p]<2>, \mathbf{F}[f_D])$ is the smallest one.
5) Take the position $\mathbf{p} = \{p_x, p_y\}$ of f_p in f_D.

Such proposed algorithm realizes pattern matching. In order to realize pattern tracking, we propose following upgrades:

1) Instead of full frame domain $[1, W_D] \times [1, H_D]$, we consider small sub-area $[p_{x,t-1} - \Delta, p_{x,t-1} + \Delta] \times [p_{y,t-1} - \Delta, p_{y,t-1} + \Delta]$, where $t - 1$ denotes previous location, i.e., we use previous known pattern location and search in only its Δ-neighborhood. This restriction improves processing speed because searching space is reduced and also increases success rate from the same reason.
2) Let S is a spatial distance between pattern and its projection in searched sub-area and T_U is an threshold. When $S > T_U$, tracking is stopped. Such situation means pattern cannot be found.
3) When $S > T_L$, where T_L is a threshold and $T_L < T_U$ holds, algorithm replaces original pattern f_p by image of its located projection in actual searched sub-area. This pattern updating helps to handle pattern visual modification such as rotation.

V. EXPERIMENTS

In this section, we will formulate the exact conditions of our experiment at first, then design an application and finally, we will benchmark various tracking cores used inside the application.

A. Experiment setting and obstacles

In the experiment, we build a glass-side arena with dimensions 1200×800mm. On the floor of the arena, a flat black cotton fabric is placed. The arena is recorded by a 4k camera, i.e., it is recorded with a resolution of 3840×2160px in 24 frames per second. Further, we recorded simultaneously 20 pieces of insect at one time, where each one insect has placed a paper with a label on its back in order to unambiguously distinguish between them. As the label, we propose one set of characters {Z, X, B, O, M, H, 4, K, V, +} in two colors: green and red. Because of the experiment has to be recorded during a night, the green and red colors are UV-reactive and over the arena is turned on a UV-light bulb. A part of a recorded movie is illustrated in Figure 2. Without any deeper exploration, the problem seems to be trivial to solve - a black background without any significant noise where highly-visible objects are moving.

Going into details, several problems appear. Even though a 4k movie is recorded, the insects and therefore the characters are really small - their size is approx 25×15 pixels so the possible information which can be extracted from such pattern is highly limited which may result in decreased success rate. The extracted patterns with all possible characters are visualized in Figure 3. Moreover, the labels are placed onto live insects which are moving in two dimensions, jump and

Fig. 2. Illustrative screenshot of a recorded movie

also rotate around its axes. From the reason, the same labeled pattern varies in time in its rotation, size and light reflection ability. An example is shown in Figure 4 where ten patterns with the same label "+" were arbitrarily extracted from one single movie. It is obvious that so big variety can lead to decreasing success rate when several insects are very close to each other and therefore a possible swap of different detected patterns may appear.

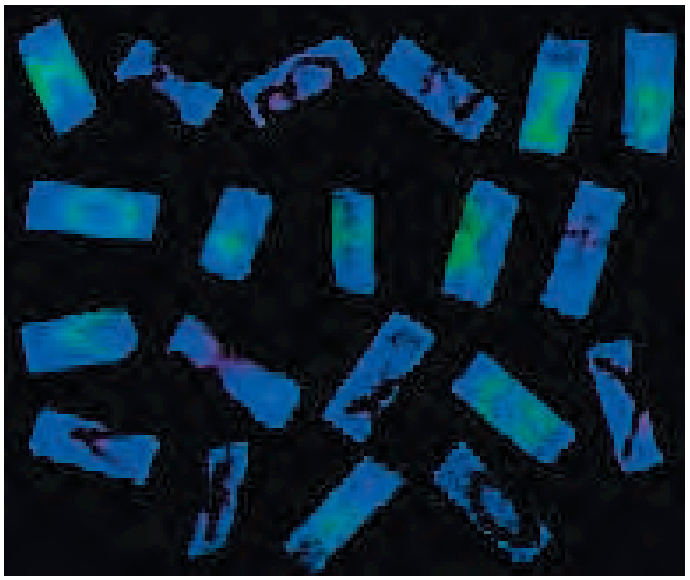

Fig. 3. Extracted objects - patterns from a single movie frame.

The next problem is processing speed. Because of the experiment design, 27 movies with the length of 600 seconds per movie were recorded. It results in $7.7 \cdot 10^6$ pattern positions to be tracked in 4k movie resolution. Just for illustration, if we consider general pattern tracking algorithm working in real-time (e.g., 30fps) for a full HD video, it means it would take 288 hours to process all the movies in our experiment setting excluding time for movie decoding/coding, handling failures etc.

The last obstacle is insect's behavior. Standardly, tracking algorithms include way, how a trajectory can be estimated

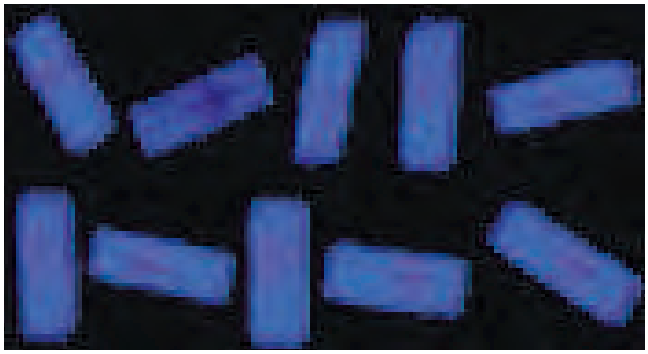

Fig. 4. Various rotations and light reflections of character "+".

when an object is lost - it can be done by, e.g., Kalman filter [16] and its derivatives. In our case, an insect can be lost for a while because one insect can cross another one by moving over its back. Unfortunately, Kalman filter etc. cannot be used for such tracking because an insect does not have a nice, smooth trajectory but its movement is *chaotic*. It can change direction randomly as same as speed - a non-moving insect can jump, i.e., it can achieve a massive acceleration between captured frames.

B. Application design

When handling a large number of movies, fast and reliable tracking core is useless without a user-friendly application interface. Therefore we implemented such interface using multi-platform Qt framework[7] connected with C++ coding language. The application GUI is visualized in Figure 5 and offer following functionalities.

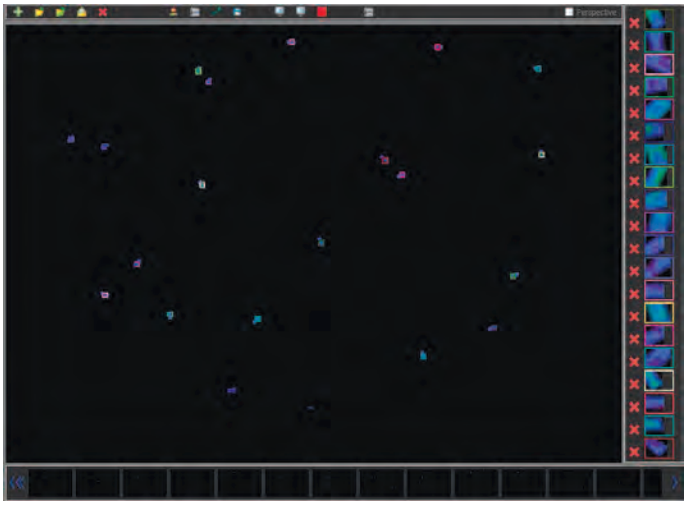

Fig. 5. GUI of the application for insect tracking.

Create project: a user can define project name and select the movie to be processed. After that, the movie is decoded into particular frames. Because FFmpeg[8] is used for decoding, we

[7]http://www.qt.io
[8]https://www.ffmpeg.org/

can support a huge variety of movie formats and codecs. At the same time as the movie is decoded, frames thumbnails are created and saved to hard disk too. This step is time exhaustive (especially in the case of our 4k movie), but it is realized only once for the whole lifetime of the project. After that, the project can be opened using *Open project* button in a short time. When the project is loaded, a user can see the actual frame in the center of the window, and several actual thumbnails in the bottom. The thumbnails can be easily browsed using left/right buttons and after the click to a particular thumbnail, the big image is updated. In the big central image, a user can select (by a mouse drawing) patterns to be tracked. The selected patterns are visualized by a unique color and visualized in magnified form on the right side of the application window. If a pattern is not selected in a proper way, a user can click to the *delete* button and select it again.

The selected patterns can be stored using *save* button and also loaded back by *load* button. When the patterns are selected, a user can click on *process* button. With that, tracking is started, i.e., one-by-one frames are processed, stored into a hard disc and visualized in the application GUI. A user can also see actual progress in percents. The tracking process stops when one of the three following conditions is reached: 1) all patterns are processed; 2) user clicked on *stop* button; 3) application detects a possible error in tracking. In all the three cases, a user can click to *Show patterns* to see couples consisting of patterns with the same id from the first and the actual frame. This is necessary for easy comparison if some patterns have not been swapped. Also, by clicking on *Render movie*, a movie where all tracked positions are marked by the uniques colors is rendered using FFmpeg. When a user thinks that all fit, he can store positions and sizes of tracked patterns. Finally, there is button *show path* to visualize the complete paths of all patterns in one single image - the illustration of the output is shown in Figure 2.

C. Benchmark

To create fair conditions for our proposed F-transform (FT) based algorithm and other benchmarked ones, we use the same application functionalities as is described in Subsection V-B for all the algorithms, the only one thing which differs is which algorithm inside is called for tracking - if FT, KCF, MIL etc. To test the five state-of-the-art algorithms, we use OpenCV[9] framework, where all the algorithms are implemented because, in this framework, we can be sure they are well coded. To be convinced, we tested one real-life movie where an object was set to be tracked and based on that, we can confirm that all the five algorithms work great.

From the qualitative point of view, we measured two variables - processing speed and failure rate. The processing speed is measured as the time needed for determining one single insect position; as the time needed for processing whole frame including twenty pieces of insect; and to be complete, as the estimated time to process all 27 movies. The three measurements are dependent, we use them just to illustrate the scale of the problem. The times are measured on MacBook Air 2013 notebook and include time needed for loading images from a hard drive and storing them back. The measured times are shown in Table II. As we supposed, the fastest one is FT because it works over reduced space and realizes only one simple strong classifier which is by the principle faster than one strong classifier consisting of many weak ones. The second one is KCF - according to the original publication [5], KCF is able to process up to 100fps when one single pattern is tracked in a standard-resolution movie. Note, even if insect per second tracking seems to be quite similar to FT tracking algorithm, considering all 27 movies the difference is not negligible 121 hours of processing time. For the same reason, the rest of methods are unacceptable from the processing time point of view.

TABLE I
PROCESSING SPEED

Algorithm	Insects per sec	Frames per sec	27 movies [h]
F-transform	8.7	0.43	251
KCF	5.9	0.29	372
Boosting	3.5	0.17	635
Median Flow	3.5	0.17	635
MIL	1.0	0.05	2160
TLD	0.4	0.02	5400

The second measured variable, the failure rate is a percentage ratio how often is it necessary to stop the application (or the application has stopped itself) and manually repair the bad location. We designed test set which includes 1000 frames (i.e., we track 20000 positions in total), which were extracted from one randomly chosen movie and are time-sorted. The results are shown in Table II, where the best one - FT - achieved $15\times$ lower failure rate than the runner-up - MIL. But in the case of MIL tracker, we have to point out there is one drawback. Even though the failure rate is pretty low, we cannot speak about precise detection, because the algorithm marks a location with a certain tolerance which results to oscillating between frames. According to that, the measured path differs from the true one. The measured length varies from 7 % to 53 % to be longer than the truth is. Such performance is not usable for further statistics processing.

TABLE II
ALGORITHM RELIABILITY

Algorithm name	Failure rate [%]
F-transform	0.005
MIL	0.075
Median flow	0.085
Boosting	> 1
TLD	> 1
KCF	N/A

The runner-up in processing speed benchmark, KCF tracker does not work in our experiment - when an insect is moving, KCF is remaining to marks the same area where the insect was originally located and it does not follow its trajectory. We

[9]http://docs.opencv.org/3.1.0/d0/d0a/classcv_1_1Tracker.html

tested the implementation on another real-life movie, where we observed it works when a big-enough area is marked to be tracked. Therefore, we tested not to mark the only insect, but also its surrounding in order to obtain bigger area. Unfortunately, such setting is not suitable because the algorithm fails when another insect came into the same area, which occurs very often.

In the case of TLD and Boosting, we tested only several frames - the algorithms failed for some of the twenty insects in almost all frames so we stopped the processing - it is not in human abilities to stop the algorithms almost every frame, fix the error and continue for the 1000 test images.

VI. Conclusion

In the paper, we have presented a real solved biologics problem - visual insect tracking. Because existing tested applications realizing tracking did not produce useful outputs, we design own user-friendly application allowing us to select patterns (insects) and which tracks them frame-by-frame. To realize the tracking, we developed and described own tracking core based on F-transform algorithm which serves as a strong classifier and follows idea "tracking by matching". This approach has been compared with five existing state-of-the-art algorithms (KCF, MIL, Boosting, TLD, MedianFlow) with the result that FT based algorithm is the fastest one and in the same time with the lowest failure rate. A part of a processed movie has been coded as a movie and uploaded into youtu.be/PYydVG6gjE0.

Acknowledgment

This research was supported by the project "LQ1602 IT4Innovations excellence in science".

We would like to thanks our colleague Michal Burda for his advice about UV light usage.

References

[1] R. Kays, M. C. Crofoot, W. Jetz, and M. Wikelski, "Terrestrial animal tracking as an eye on life and planet," *Science*, vol. 348, no. 6240, p. aaa2478, 2015.

[2] M. Wikelski, R. W. Kays, N. J. Kasdin, K. Thorup, J. A. Smith, and G. W. Swenson, "Going wild: what a global small-animal tracking system could do for experimental biologists," *Journal of Experimental Biology*, vol. 210, no. 2, pp. 181–186, 2007.

[3] A. W. Smeulders, D. M. Chu, R. Cucchiara, S. Calderara, A. Dehghan, and M. Shah, "Visual tracking: An experimental survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 7, pp. 1442–1468, 2014.

[4] P. Hurtik and P. Števuliáková, "Pattern matching: overview, benchmark and comparison with f-transform general matching algorithm," *Soft Computing*, vol. 21, no. 13, pp. 3525–3536, 2017.

[5] M. Danelljan, F. Shahbaz Khan, M. Felsberg, and J. Van de Weijer, "Adaptive color attributes for real-time visual tracking," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Columbus, Ohio, USA, June 24-27, 2014*. IEEE Computer Society, 2014, pp. 1090–1097.

[6] F. S. Khan, J. Van de Weijer, and M. Vanrell, "Modulating shape features by color attention for object recognition," *International Journal of Computer Vision*, vol. 98, no. 1, pp. 49–64, 2012.

[7] Z. Kalal, K. Mikolajczyk, and J. Matas, "Tracking-learning-detection," *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, no. 7, pp. 1409–1422, 2012.

[8] B. Babenko, M.-H. Yang, and S. Belongie, "Visual tracking with online multiple instance learning," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 983–990.

[9] C. Ji and S. Ma, "Combinations of weak classifiers," in *Advances in Neural Information Processing Systems*, 1997, pp. 494–500.

[10] H. Grabner, M. Grabner, and H. Bischof, "Real-time tracking via on-line boosting." in *Bmvc*, vol. 1, no. 5, 2006, p. 6.

[11] H. Grabner and H. Bischof, "On-line boosting and vision," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 1. Ieee, 2006, pp. 260–267.

[12] Z. Kalal, K. Mikolajczyk, and J. Matas, "Forward-backward error: Automatic detection of tracking failures," in *Pattern recognition (ICPR), 2010 20th international conference on*. IEEE, 2010, pp. 2756–2759.

[13] B. D. Lucas, T. Kanade *et al.*, "An iterative image registration technique with an application to stereo vision," *IJCAI*, vol. 81, p. 674–679, 1981.

[14] P. Hurtik, P. Hodáková, and I. Perfilieva, "Approximate pattern matching algorithm," in *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*. Springer, 2016, pp. 577–587.

[15] I. Perfilieva, "Fuzzy transforms: Theory and applications," *Fuzzy sets and systems*, vol. 157, no. 8, pp. 993–1023, 2006.

[16] D.-J. Jwo and S.-H. Wang, "Adaptive fuzzy strong tracking extended kalman filtering for gps navigation," *IEEE Sensors Journal*, vol. 7, no. 5, pp. 778–789, 2007.