Annotating Digital Images of Insects

Automatic Generation of RDF files using Computer Vision Methods

Project paper of seminar

SEMANTIC MULTIMEDIA TECHNOLOGIES

Summer semester 2015

Hasso Plattner Institute

University of Potsdam

presented by

Leander Neiß
Friedrich Horschig
Clemens Frahnow
Sten Aechtner

31th August 2015

Abstract

In this paper we present the automated annotation tool for insect images BugTracker. Using computer vision tools, BugTracker analyses digital images of insect boxes from museums. It locates occurring insects and creates annotations in form of an RDF file, which contain the position as well as the species of the found individuals. We present the methods Edge and Contour Detection and Template Matching as possible algorithms for segmentation. A combination of both is used in our tool. For the classification of the insects we use the information provided by the QR-codes, which can be found in the images as well.

Contents

1 Introduction

The *Museum für Naturkunde*, a natural history museum in Berlin, Germany, features an insect collection consisting of several million insect specimen. This collection is digitalized in the context of the museum's EoS project. Typically, the resulting images show multiple insect boxes filled with insect specimen of the same species. On average, such an image is about 100 MB in size and contains about 50 specimen.
In this paper, we present BugTracker, a tool developed to annotate the museum's insect collection images with information about the location of each single specimen in the image and what type of insect it is. This should support the museum's work with those rather large images in order to use and further process them.
The paper presents an approach based on template matching that allows the automatic annotation of insect images with semantic information. The algorithm itself is a multi-staged approach consisting of four main subtasks:

Contour detection to automatically generate template candidates and extract one of them as the actual template

Template matching to identify the location of all insect specimen in the image and calculate their bounding box

QR code analysis to identify qr codes, extract a species identifier, and lookup the biological information associated with it from a reference data set

Annotation of the image by creating the output rdf file

This work is created in the context of the *Semantic Multimedia Technologies* seminar during the summer semester 2015 at the Hasso Plattner Institute, University of Potsdam, Germany.
The remainder of this paper is structured as follows. Section 2 highlights related work. The motivation for this work is presented in section 3 and detailed information about the implementation are described in the next section. The results are evaluated in section 6. In the last section we draw a conclusion and discuss it.

2 Related Work

2.1 Semi-automated Insect Analyzer

The Natural History Museum of London launched a project to achieve the goal of digitalizing their insect photographs. The resulting tool is called “Inselect” [Gro14] and provides assistance for semi-automatic annotation of insects. During the time of this writing, their approach uses edge detection from the library OpenCV (details about this library follow in the next subsection). A part of annotation and segmentation is done by the user. They provide a possibility to scan the present QR codes for annotation proposals. There is a good coverage of tests but their test set consist of one image. Apparently, an evaluation of efficiency of their actual algorithm was not suitable so the projects are very hard to compare.

2.2 Approaches based on Machine-learning

During the last years, there was great process in image analysis. A lot of them are based on clustering algorithms which are very well suitable for either known numbers or sizes/shapes of objects [Pap92]. Others use learning algorithms which are usually trained with a large set of human-reviewed data to extract the most probable segmentations. One recent example for a rising technology is the use of neural networks [TMJ$^+$10].
During Section 4, we will explain why it would be very hard for our specific use case to gather a large training set.

2.3 Current state of Computer Vision

Luckily, there are very sophisticated approaches of static image analysis based on different features of a single image. The most famous, free and open-source collection of such computer vision algorithm is OpenCV [Bra00]. This library contains very advanced methods like the “SURF” algorithm as well as basic steps for image processing for threshold-based contour detection algorithms.
One particularly interesting part of this library implements a template matching algorithm. It uses a predefined template that is applied on different positions to an image to find similar regions. The comparison of those regions to the template image can base on different factors like oriented gradients or distribution in luminosity/color.

3 Motivation

4 Concepts

4.1 Edge Detection

Edge Detection is a part of segmentation in image processing. It is used to isolate some areas of an image from others, such as shapes in the foreground from the background. An edge is determined by the difference of its adjacent brightness value. The points where the image brightness has discontinuities are represented as curved line segments, the *edges*. A threshold image of the original image is computed to determine whether two brightness values are classified as a discontinuity. Using a global, fixed threshold would classify everything as an edge, that has one adjacent brightness above, and one adjacent brightness below a specific, predefined value. As second possibility there is an adaptive threshold. It sets the threshold adaptively for each area in the image, using the mean brightness of that area.

4.2 Template Matching

Template Matching in image processing basically is a method to extract parts of an image that match with a chosen template based on a comparison function. The algorithm takes all possible template sized sub-images and uses the comparison function to yield a value for the similarity to the template. Based on this value, the best match can be found, or giving a threshold value x, all results with a higher or lower value than x can be gained. There are multiple comparison functions like pixel-wise comparison or a simple perfect-match function.
The most relevant functions used in this paper are Correlation Coefficient and Histogram of Oriented Gradients which are introduced in the following sections.

4.2.1 Correlation Coefficient

The Correlation Coefficient Function (short "CCOEFF") first calculates the mean of all pixel values of the template as well as the mean of the pixel values in the selected sub-image. By iterating over all pixels in the template, two sub-values are calculated per iteration: The first one is the signed distance of the pixel value in the template to the mean value of the template. The other one is the signed distance of the value in the sub-image at the corresponding pixel position to the mean value of the sub-image. Both sub-values are multiplied in each iteration and the resulting values of all iterations are summed up. To have a normed value for the comparison, the result is divided by the highest possible value. Since the used parameters for the function are the distances to the mean, the function calculates a value independent of the absolute values.
The resulting formula for this function can be seen here:

$$ccoeff = \frac{\sum_{x,y}(T(x,y) \times I(x',y'))}{\sqrt{\sum_{x,y}(T(x,y)^2 \times \sum_{x,y} I(x',y')^2)}}$$

Where T(x,y) is the pixel value difference from the mean in the template at position x, y and I(x', y') the corresponding pixel value difference from the mean in the sub-image.

4.2.2 Histogram of Oriented Gradients

Going away from the pure pixel values, the Histogram of Oriented Gradients function (short "HOG") is based on gradient directions. It counts the number of gradient orientations in each (self defined) section of the sub-image and compares the result with that from the template. [DT05].

4.3 Machine Learning

4.4 Comparing Effectiveness of Algorithms

To have an objective measurement of the effectiveness of changes in the algorithm, we will consider two established factors for data quality. They are based on a gold standard [Ver92]. The gold standard will be a manually selected and annotated subset of the given data. All algorithms will be tested against the same set of data and compared against the standard. Some data will be rightfully found (true positives), some will be wrongfully found (false positives) and some insects won't be found even if they should have been (false negatives)
The first factor is the recall that measures how many relevant results were retrieved. This is achieved by dividing the true positives by all relevant results (true positives and true negatives).
The second factor is precision of the results how many of the found results were relevant. This is achieved by dividing the the true positives by all positives (true and negative).
To raise the meaning of each factor, they are usually combined in a harmonic mean called F-Measure. The resulting value can only be maximized by maximizing both values. A very low value in one of the factors will result in an overall low factor. The F-Measure is calculated as follows [Pow11]:

$$F - Measure = 2 * \frac{Precision * Recall}{Precision + Recall}$$

5 Implementation

We implemented BugTracker as commandline-based Python tool. It takes an image as input, and produces an RDF file with annotations of the found insects. Template

5.1 Contour Detection

As a basic approach, we use Contour Detection to find insects in a given image. The input image converted to gray-scale is used to detect edges as mentioned in Section 4. We use an adaptive threshold to create a binary image, which is then transformed by a morphological operation from OpenCV, in order to close gaps in areas of interest. On this transformed binary image, an OpenCV contour detection function is applied, which groups connected edges into one contour (Figure 1). Those contours are a first approach on finding something on an image. However, the threshold and edge detection algorithm can not possibly know, what we are looking for. The found contours can be anything that appears to stand out from the background, not only insects. This method is a good approach to just find anything on the image, but not precise enough for our needs.

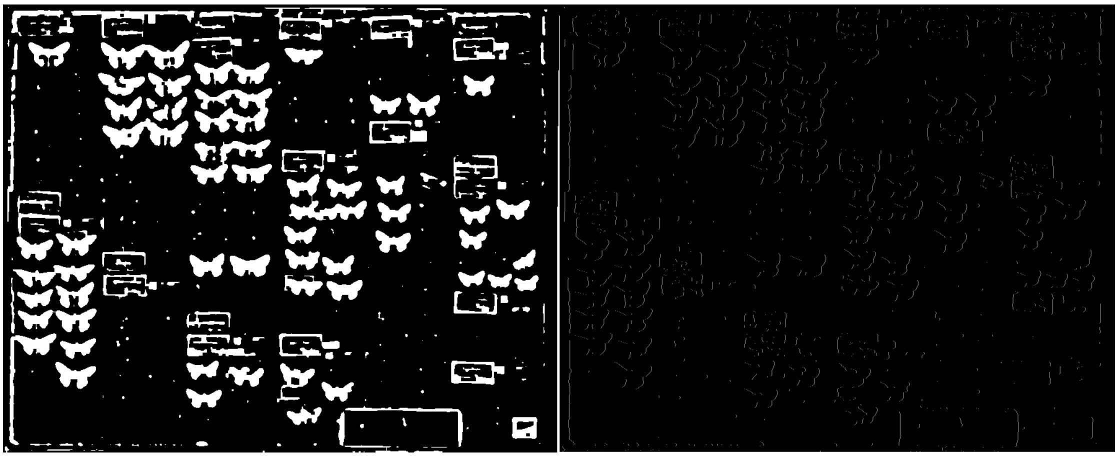

Figure 1: Binary image after using an adaptive threshold and morphological closing (left). Contour image created from binary image (right).

5.2 Template Matching

To provide the algorithm an imagination of how the shapes which it is supposed to find look like, it is necessary to give it a sample shape to be geared on. A possibility for that is using a template matching algorithm. OpenCV provides this functionality, so the only thing to do was to define a comparison function. By testing it turned out that CCOEFF gave the best results. As templates cropped images from the insect library were used like in Figure 2.

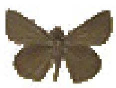

Figure 2: A template for butterflies

The result of that OpenCV algorithm was a list of all sub-images with their corresponding match values describing how good they match the template. The part of the image where we took the template from, of course always had a value of 1, which is the best possible value. All other values are between -1 and 1, where a higher value means, that it matches the template better. The next step is to set a threshold, defining from which value on sub-images are taken as matching. A threshold of 0.41 worked best for us to avoid too many false positives and true negatives.
Using this algorithm a result like in Figure 3 is generated.

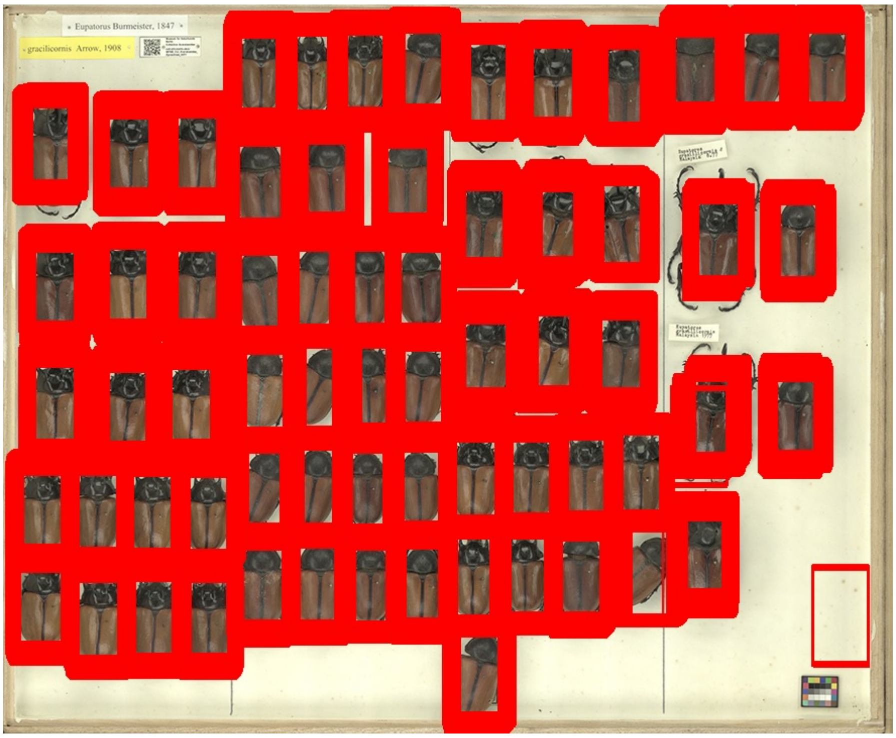

Figure 3: Multiple matches per insect

As you can see the wide borders of the frames are the result of multiple overlapping matches, that are showing the same insect. To group such corresponding matches together to one match, an overlay threshold of 30% was defined to tell the algorithm when two frames are put together. The same image as in Figure 3 now with frame grouping in Figure 4

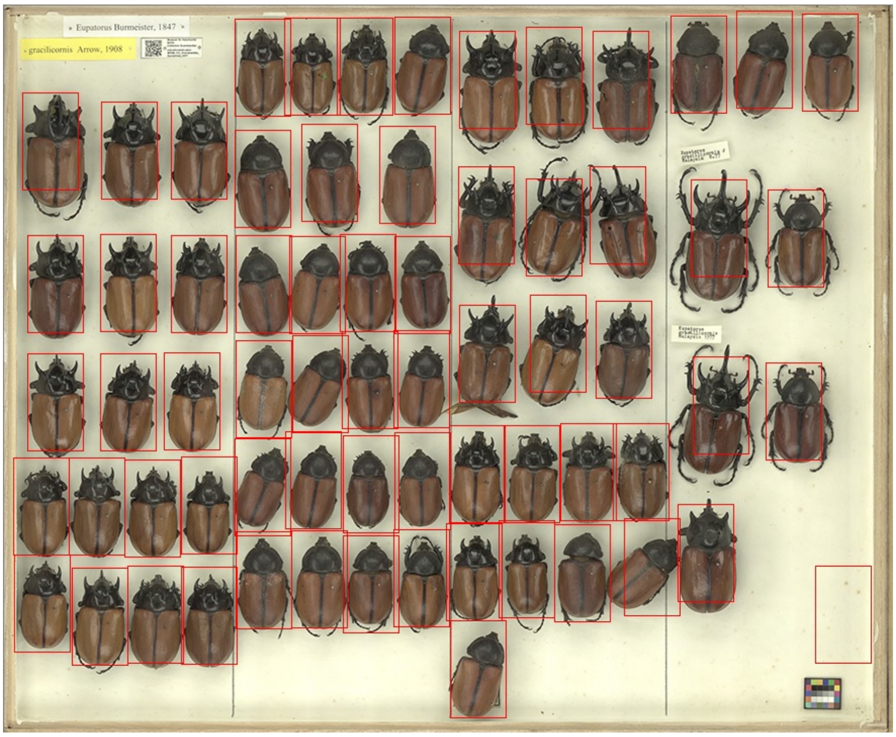

Figure 4: Only one frame per insect after grouping corresponding ones together

In Figure 4 you can see the results with the given algorithms so far. As you can see there are many false positives left, which have to be discarded. This is done by a second comparison using Histogram of Oriented Gradients. If the matching value is below 0.95, the result is discarded. Another problem is dirt and other unexpected noise on the image, that sometimes matches the template pretty well as a false positive. An example for that is the false positive in Figure 4 on the bottom right. The soft shadow in the background describes a shape almost like one of the bugs. To get rid of those results, the histogram of the whole image is generated to get the background color. All matches now have to have an average color that has a certain difference to the background color to be taken as a match. A final result can be seen in Figure 5.

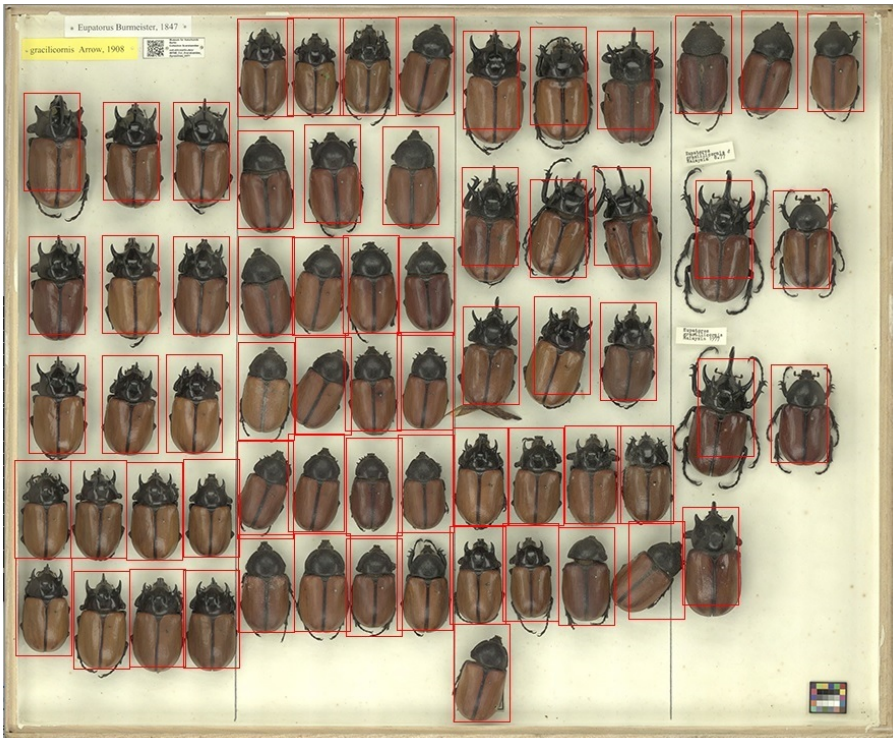

Figure 5: The bottom right frame has gone by analyzing the histogram

5.3 Automated Template Extraction

Until this point, the templates were provided manually by cropping images. Since the annotation has to run fully automated, the template extraction also has to be done by the algorithm. The Edge Detection algorithm is reused here to provide a list of potential templates. This list is still full of false positives that have to be discarded. In multiple steps, the following contours are thrown away:

1. Very small contours, that consist of less than 50 points.
2. Contours that appear to be a label, if identified correctly. This is done by a template matching against some sample labels which were manually cropped out. Since labels look quite similar in each image, there is a chance that with this step those contours are discarded.
3. 5% of the number of contours are removed which have the highest height, 5% which have the lowest height, 5% that have the highest width, and 5% that have the lowest width. This removes contours that have caught for example noise on the side of an image, which resembles e.g. the edge of the insect box (very long but thin contour).
4. Contours with an extreme small or big area.

After these steps, we have reduced the number of found contours that are potential templates for our Template Matching. From the current set of potential templates, we extract one, which is representative for the insect species in that image. This is done by apply template matching to all of our potential templates against each other. The OpenCV matchTemplate function yields a score when matching a template against another image. The higher the score, the better the match. So when matching each potential template against each other, we sum up the scores that are yielded, so that in the end the potential template with the highest score is our actual extracted template. This of course has the premise, that after reducing the set of potential templates, there are more candidates which are actual insects, than other stuff, like labels, or just dirt on the box. That is most of the time the case, but to not completely rely on that premise, we add multiple previously cropped out samples to the template matching process to improve the score of the templates that are actual insects. We also add negative samples, such as labels, which should reduce the score of unwanted templates. After this process, we choose the potential template with the highest match score as our extracted template for our Template Matching step.

5.4 Annotations

The simplest annotation is just an RDF-triple locating a bug on an image and describing it as Organism. The definition for an Organism and all properties we use to describe are part of the Darwin Core [1] Standard of describing living organisms.
We can extract further information from a CSV file that was provided by the Natural History Museum of Berlin. It contains a general overview of all species in a photograph. This includes dates and information about the family.
Whenever a new bug is found, it is added to the collection of bugs that will be written as RDF file at the end. It is possible to annotate every bug with additional, specific properties (apart from those that apply to every insect in the box).
Such information can be found in QR codes as described in the next section.

5.4.1 QR Code Analysis

5.5 Integrating Benchmarking

Due to the availability of each algorithm as single step in the pipeline, it was easy to execute them in the exact same environment with the exact same parameters. Different from the usual pipeline, the step of selecting the examples was automated and the step of writing out RDF files was replaced by analyzing the discovered bugs.
To have a reliable set of data (a "gold standard"), we created annotations for some photos manually. The photos were mainly taken randomly. In addition, we chose some photos which we found hard to analyze (due to transparent wings, difficult shapes or different sizes).
The actual gold standard was not a set of RDF files but a collections of comma-separated values. These CSV files were easier to create (with a helping tool we wrote to manually annotate the bugs) and easier to analyze than RDF documents with same contents. The

[1] http://rs.tdwg.org/dwc/

actual results, their effect on our process and how we could optimize the benchmarking process will be discussed in chapter 7.

6 Evaluation

6.1 Benchmarks

To evaluate our tool, we created a benchmark set containing several images where we manually marked the insects. The result of one benchmark run is expressed as *Recall*, *Precision*, and *F-Measure* [Pow11]. Although F-Measure is a very common way to measure the quality of results, we consider changing to another method that puts a higher weight on the quantity of results. Usually, it would be easier to refine existing results than to find additional insects. Also, when looking for false positives, it is possible that true positives are found because the bounding box did not fit correctly. However, it is more useful to see an image of multiple insects or a cropped insect instead of no result at all.

The test set is crucial for the accuracy of the benchmark. It is always good to have a large test set to improve the accuracy of the benchmark. Therefore continuous extending of the test set is needed.

6.2 Results

The results show nothing!!

7 Conclusion and Discussion

7.1 Optimizing the Template

7.2 Machine Learning?

7.3 Sources for Meta Information

Sten? where from? (QR, provided file, crawling webpages)
how could it benefit US? (get templates from meta data)

References

[Bra00] Bradski, G.: The OpenCV Library. In: *Dr. Dobb's Journal of Software Tools* (2000)

[DT05] Dalal, Navneet ; Triggs, Bill: Histograms of Oriented Gradients for Human Detection. In: *Computer Vision and Pattern Recognition* (2005)

[Gro14] Group, Natural History Museum's Biodiversity I.: Inselect Automated segmentation of scanned specimen images. (2014). `http://naturalhistorymuseum.github.io/inselect/`

[Pap92] Pappas, T.N.: An adaptive clustering algorithm for image segmentation. In: *Signal Processing, IEEE Transactions on* 40 (1992), Apr, Nr. 4, S. 901–914. `http://dx.doi.org/10.1109/78.127962`. – DOI 10.1109/78.127962. – ISSN 1053–587X

[Pow11] Powers, David M W.: Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation. (2011)

[TMJ+10] Turaga, Srinivas C. ; Murray, Joseph F. ; Jain, Viren ; Roth, Fabian ; Helmstaedter, Moritz ; Briggman, Kevin ; Denk, Winfried ; Seung, H S.: Convolutional networks can learn to generate affinity graphs for image segmentation. In: *Neural Computation* 22 (2010), Nr. 2, S. 511–538

[Ver92] Versi, E.: "Gold standard" is an appropriate term. (1992), S. 187