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

31st 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 *Erschließung objektreicher Spezialsammlungen* (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.
This paper presents BugTracker – a tool developed to annotate the museum's images of its insect collection with information about the location of each single specimen in the image and its biological classification. This supports the museum's work with those rather large images enabling further use and processing of them. The main focus of the project can be found in the field of image processing to extract these extra information. The approach presented in this paper is based on template matching and contour detection and allows the automatic annotation of insect specimen on images with semantic information. The algorithm itself consists of multiple stages and 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. In Section 6, the results are evaluated and future work discussed.

2 Related Work

2.1 Semi-automated Insect Analyzer

The *Natural History Museum* in 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. Currently, 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 and many projects 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 6, 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 tools and methods as well as basic steps for image processing for threshold-based contour detection algorithms.
One particularly interesting part of this library for this project 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

The main idea of the BugTracker project is to support the digitalization of an insect collection by creating an automated tool that allows to annotate thousands of images. Existing tools and algorithms are explored in order to reuse them, and eventually combined for the best results. The four aspects in detail:

Support digitalization The EoS project of the *Museum für Naturkunde* in Berlin is aimed at digitalizing the museum's insect collection. The BugTracker tool allows to support this process by providing additional information about the data such as the position of a single insect in an image.
Further, the QR codes in each image provide information about the taxonomic classification of those insects. So far, the included species of each image are known but annotating each insect and linking it to QR code information offers more details about it.

Automatic Annotation Process The EoS project resulted in a few thousand images of insects. In order to support annotating this amount of images, it is important to develop a tool that completely automates the annotation process. This includes everything from reading an image, processing the image and its meta data, up to creating the rdf annotations.

Reuse Existing Tools and Algorithms The development of the BugTracker tool was part of a university project over the span of one semester. In turn, to create a well-suited tool in this amount of time, it is important to explore and reuse already existing tools and algorithms instead of implementing everything from scratch.

Combined Algorithm Single algorithms lead to great results for some images. Unfortunately, the same algorithms fail for others kinds of images. As single algorithms were not able to lead to satisfying results, different image processing algorithms were used for automatic template extraction and the following template matching. This combined approach is supposed to leverage the advantages of more than one algorithm to yield better results.

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 (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 (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

An approach for the classification of image segments is located in the Machine Learning field. A well-trained Support Vector Machine (SVM) [HCL03] can decide whether an image (e.g. a segment of an insect box) is an actual insect (and also which species), or something else like a label, or dust in the box. A big training data set is required in order for the SVM to be precise.

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], a reliable set of data. 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.

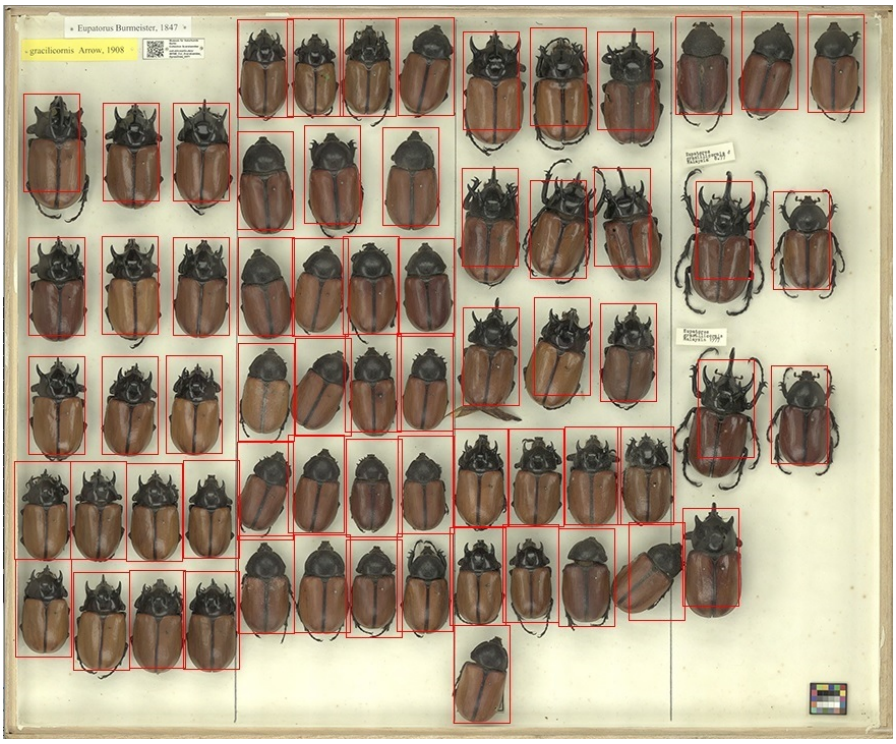

Figure 5: The bottom right frame has gone by analyzing the histogram

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 method *matchTemplate* provided by OpenCV yields a score when matching a template against another image. The higher the score, the better the match. 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 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 an 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 *Museum für Naturkunde* in 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.
Such information can be found in QR codes as described in the following section.

5.4.1 QR Code Analysis

Typically, each image contains multiple insect boxes with their own QR code that encodes a species specific URI. We can link this URI to further information, which is basically the taxonomic classification of a bug, such as a its order, genus, and species.
To search QR codes in the images, we use the *ZBar* library [2]. This Python library allows to obtain various bar codes from images and supports multiple types of bar codes. It is also used by the *Inselect* project to search and decode QR codes. *ZBar* locates and decodes QR codes, which returns the species URI in our case.
Once we have this URI, we can lookup the associated taxonomic data from a CSV file provided by the *Museum für Naturkunde*.
The mapping from a found bug to its QR code proved to be more difficult than expected because of varying box layouts. As there was no algorithm that matched everything correctly, we alternated the annotation approach to specify the taxonomic information that is common to all bugs. The alternative would be an imperfect matching algorithm

[1] http://rs.tdwg.org/dwc/
[2] http://zbar.sourceforge.net/

that provides more detailed information for some bugs but also introduces errors. We favor first approach because we agreed it is easier to add more detailed information than fixing an unknown number of introduced errors at a later point.

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, the "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 collection 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 actual results, their effect on our process and how we could optimize the benchmarking process will be discussed in section 6.

6 Evaluation and Discussion

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

Running the benchmark on our test set yields the following results:

Recall 0.82

Precision 0.80

F-Measure 0.81

The used method consists of the automated template extraction by contour detection, followed by a template matching with the extracted template.
In our test set we have images of well-shaped insects, like butterflies, and different bugs, but also images of border cases, like insects with transparent wings, stick insects, or insects which are missing some parts of their body. Those border cases cause a worsening of our benchmark results. Using a slightly smaller test set, where we only use the common cases, we can achieve much better results:

Recall 0.90

Precision 0.97

F-Measure 0.94

6.3 Discussion

When executing the benchmark, there is an option to enable a mode to see the evaluated image. By showing all false and true positives and true negatives it is possible to see what causes the resulting numbers. The main problem for not having 100% lies in the diversity of the insects: Butterflies of the same class often have differing patterns on their wings. Also differences in color and size are serious, especially between male and female ones of the same species. Another problem is the condition of the insects: Some

animals have missing parts like wings or legs or they differ in the folding of their wings. Furthermore sometimes the insects are aligned horizontally and sometimes vertically in one box, so that the template won't find all insects of one alignment. The first box in Figure 6 shows such a high diversity like missing wings.
The condition of the box where the insects are in is another influence on the result. Sometimes the background of the images has dirt on it, that makes it hard to find a difference to the insects, especially if they have transparent parts like in the second box of Figure 6

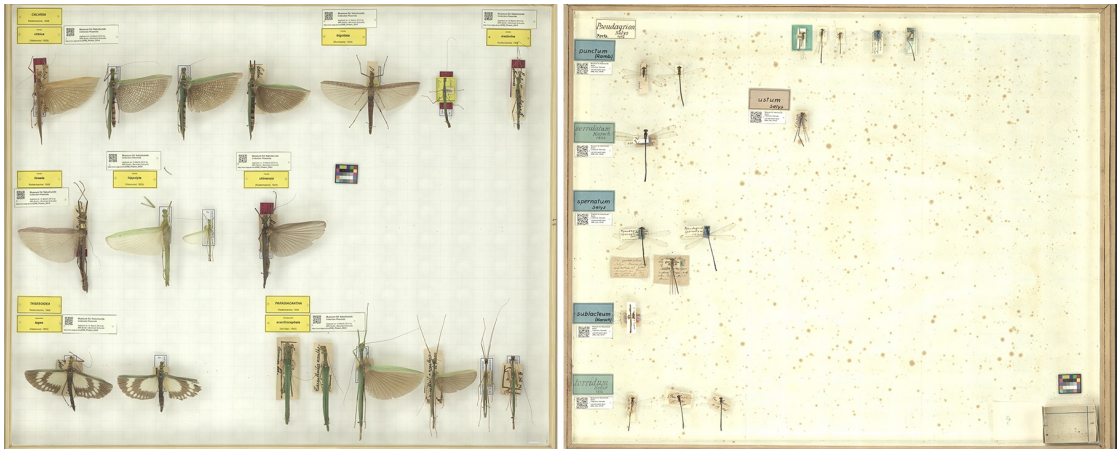

Figure 6: High diversity in one box and dirt in the other one

All these points make it hard to have a result close to 100%. But an F-measure of 94% is still a very good result and even 81% with a lot of edge cases is pretty good.

6.4 Machine Learning

As described in Section 4, using an SVM is a Machine Learning approach on classification of images. In the process of developing our tool, we also implemented a SVM, to see if it fits our purposes. We did not use it in our final method, since a really big training data set is required, which we did not have at that time. But given proper training data, an SVM would be a valid classification method.

6.5 Future Work

Since we can get the classification of the specimen in a picture out of the QR-code, it should be possible to improve the template extraction. If there is an online-service for getting images of insects by their name, the result of the automated template extraction could be verified on a much more precise level.
Another idea is to have a feedback for the user to show him critical findings or pictures with no matches at all. Furthermore before processing all images, the software could just generate all templates and show suspect ones to the user to verify if it is a good template or just a label or QR-Code.

In a separate preprocessing step it can be useful to remove all labels and QR-codes from the image before looking for a template. We spend a lot of time at this, but found no satisfying results without deleting true positives.
As mentioned before, with the help of a large trainings set, Machine Learning could be an interesting method to verify findings or to improve the segmentation. Such training sets could be gathered from other sources, where an annotation already has been done.

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/`

[HCL03] Hsu, Chih-Wei ; Chang, Chih-Chung ; Lin, Chih-Jen: A Practical Guide to Support Vector Classification. (2003). `http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf`

[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