TREEO
Image-based DBH measurement

Technical report for project financed by Fairventures Worldwide FVW gGmbH.
Supervisor: Tomas Vitek

Bianca Palackova, Tereza Sýkorová, Martin Kolář, Vítězslav Beran

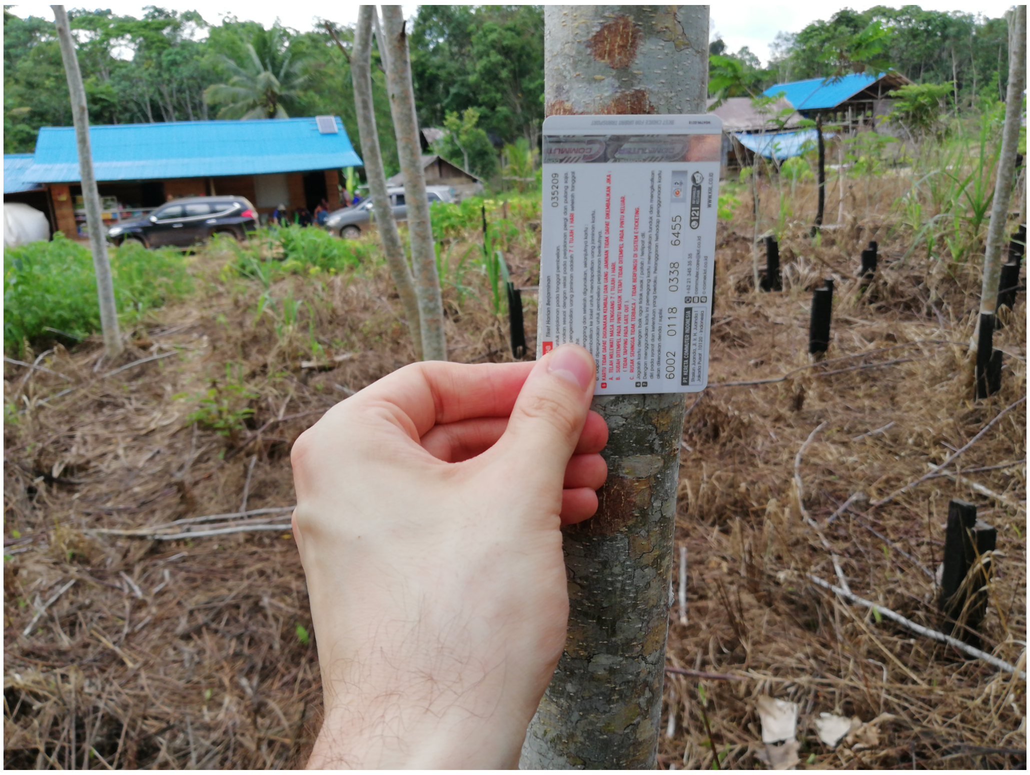
Faculty of Information Technology
Brno University of Technology
2021

Abstract

This document describes the computer vision method for DBH measurement based on pictures collected in an outdoor environment designed by Brno University of Technology.

The DBH (Diameter at breast height) refers to tree diameter measured at 4.5 feet above the ground. The method is based on detection and localization of the predefined card placed on the tree-trunk. The work includes and the report contains:

- design of the computer vision methods for card detection and localization, and tree-trunk segmentation,
- analysis of the methods reflecting the practically maximal achievable accuracy and also the stability of the method (e.g. ROC),
- monitoring the computational efficiency of designed methods reflecting the requirement of running the solution also on old-type smartphones (like Galaxy J1, but also actual types),
- documentation of the C++ library implementation with API defined together with FVW team.



Example of the image with the tree-trunk and the known card to process DBH measurement.

Content

Card Detection

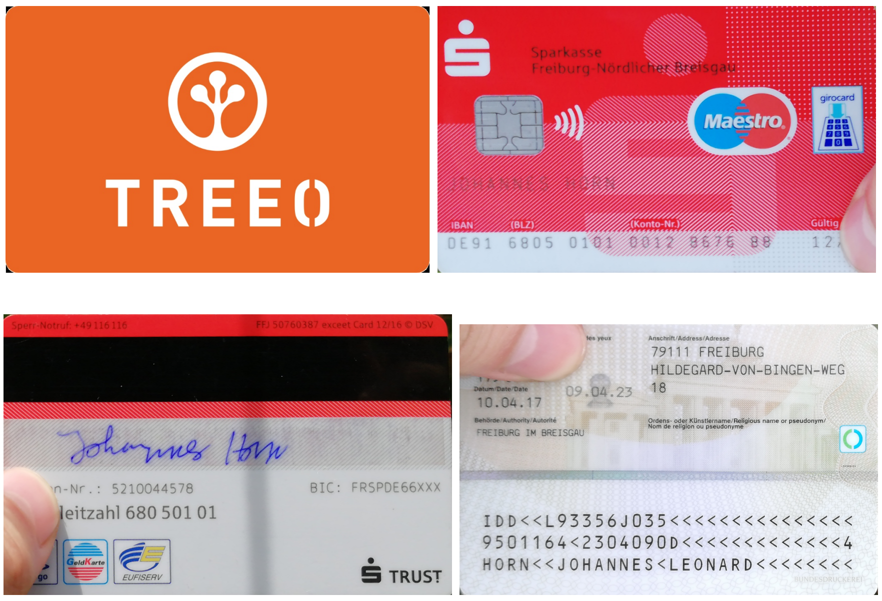
The first step of the image-based DBH (Diameter at breast height) measurement based on a priory known card is a detection of the card in the image. The method is designed with the expectation that the image contains the tree-trunk with the known card overlaid over it. Having the example images of the card, the method is trained to create the model of the card that is used to detect and localize the card in the image. The detection and localization precision is evaluated and followed by a couple of possible extensions to improve the accuracy.

Detection and localization

For card localization, we use a feature detector and then we try to find a homography between detected keypoints from image and card model. With our approach we are able to successfully detect most of the Treeo card (new orange card) if it is not covered, image is not blurry or card is in high angle. We have worse results in raw treeo dataset, it is mainly because of cards with small text like driving licenses and ID cards.

In the future, we want to improve this algorithm by precise localization of card edges and try other feature detectors, which could be faster. Detection in some area of image (50% of image in the middle of this image) could also speed up the algorithm. We also plan to improve the card's model, which should increase the percentage of localized cards.

The datasets contain images with various card examples. For the first experiments, the two cards are used to evaluate the method: card01 and card12.



Card examples for detection evaluation: card01 (top left), card14 (top right), card15(bottom left), card09(bottom right).

The detection stability (precision, recall) and computational cost (time of detection) is evaluated on various experiments:

- DetCard01
    - Card model: card01,
    - Data used:
        - 70 images (27 from Tomáš Vítek, 20 raw_treeo_uncleaned, 23 our dataset from Brno)
        - 20 raw_treeo_uncleaned - 10 cards without card and 10 with different card
        - The rest of 50 cards contain card01.
    - Method parameters:
        - Resize: 1000px width
        - SIFT: full-scale
        - Score: 0.9% of inliers
- DetCard14 and DetCard9
    - Card model: card01, card09
    - Data used:
        - 400 images with the card09, card14, card15, card16 from raw_treeo_dataset_uncleaned, folders b/1/4ref, c/1/4ref, c/1/5ref, c/1/6ref
        - 32 images of card09
        - 175 images of card14
        - 154 images of card15
        - 136 images of card16
    - Method parameters:
        - Resize: 1000px width
        - SIFT: full-scale

The detection card experiments are evaluated on the computer with the following setup: Intel Core i7-9750H CPU, 2.60GHz, RAM 16 GB

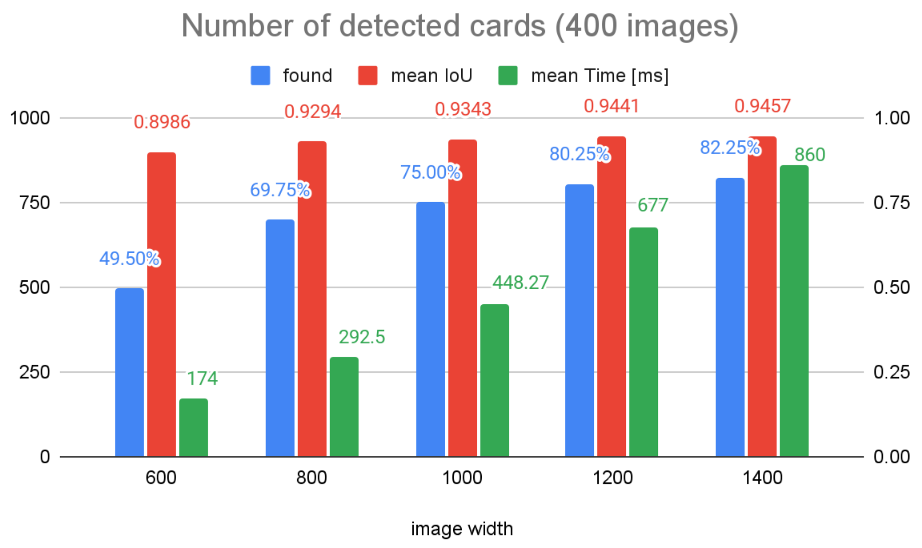
Average time to detect card: 0.46s

	Precision	Recall	Acc
DetCard01	1.00	0.94	0.95
DetCard14	1.00	0.49	0.82
DetCard09	0.95	0.59	0.97
DetCard15	1.00	0.94	0.98
DetCard16	0.99	0.97	0.99

Results of the detection card experiments.

Precision provides a number of how many detected cards were actually detected correctly. High precision mean that we do not detect card where we should not or we do not detect different type of card (for example Card14 in image with Card09).

Recall tells us what proportion of all cards that should be found were identified correctly. We can see that Card14 and Card09 were detected only in half of the cases. This could be caused because of small letters on cards and missing large features which could be found in the image.



Graph *Number of detected cards* shows experiments with various sizes of input image. Experiments were done with 400 images from dataset raw_treeo_dataset_uncleaned with cards Card09, Card14, Card15 and Card16. IoU represents Intersection over Union of detected card and ground truth label. With larger image size, algorithm is able to find more cards, but with increasing precision also increase computing time.

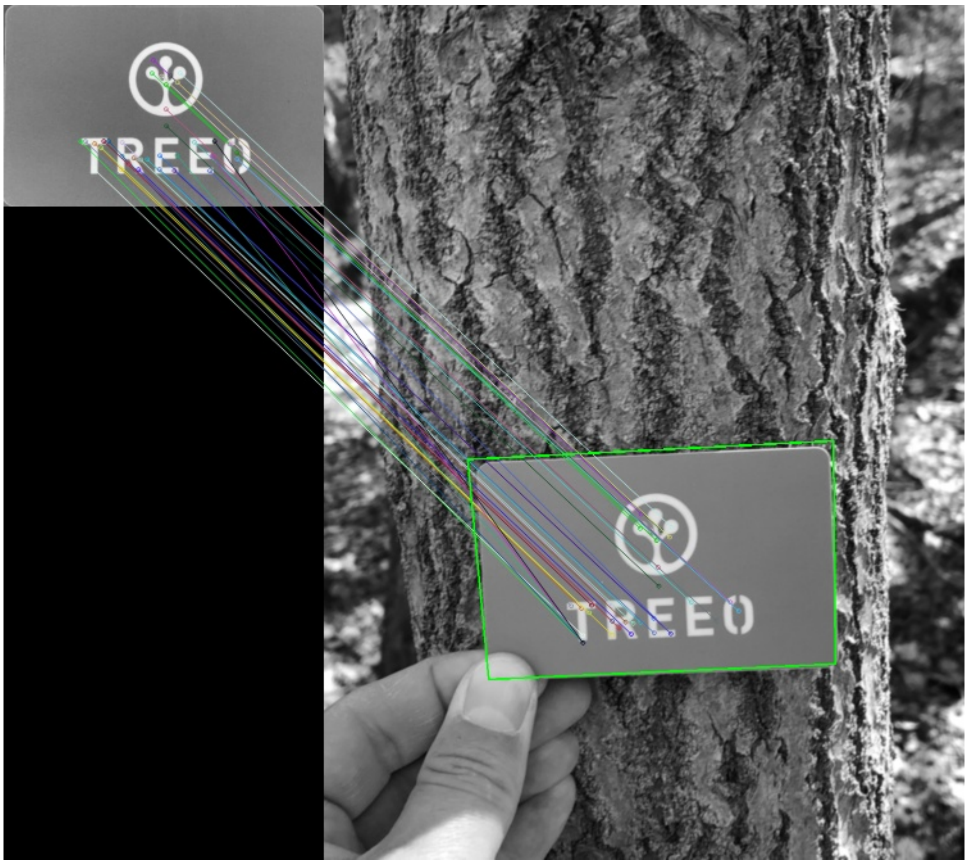
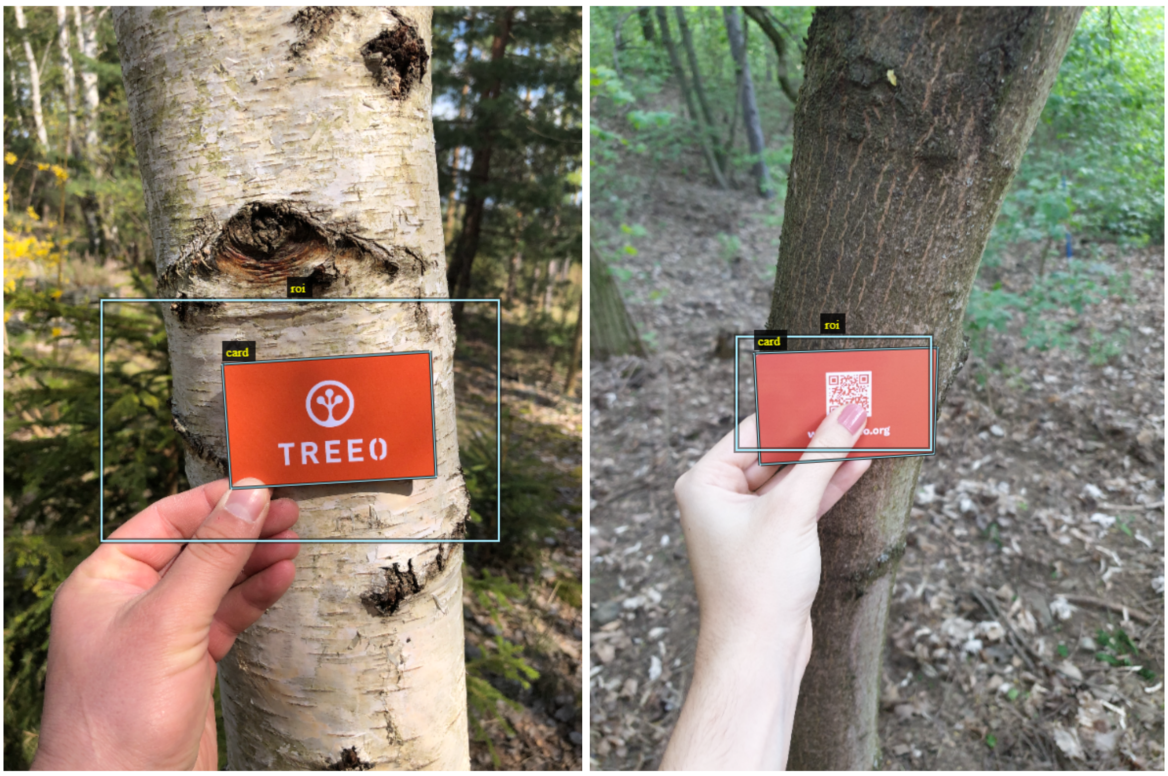


Illustration of the card detection method.

Localization accuracy improvements

In this part, experiments with other feature detectors and refinement of card edges with 2 methods will be described. Experiments were done with a dataset which contains 200 images with both sides of the new Treeo card. In these images, cards were labeled manually with the tool VGG Image Annotator. We also labeled a region around the card, which should simulate a region (ROI) where the card should be present and user should place it here during photo taking. This dataset also contains cards which are blurred, partially covered or taken in high angle. Examples from this dataset can be seen in the following images.



Examples from testing dataset. Correctly taken image (left) and partially covered card (right).

Experiments with ORB and AKAZE

We tried to replace the SIFT detector with ORB and AKAZE in order to reduce time and increase precision. Results are in the following table, where accuracy means how many cards were detected and IoU (Intersection over union) means how precise these detections were. In these experiments ROI wasn't used, which means our algorithm was looking for the card in the whole image.

detector	accuracy	IoU	time [ms]
SIFT	87.5 %	0.9536	436.41
ORB	54.0 %	0.9317	389.69
AKAZE	87.0 %	0.9541	824.31

As we can see in the table, best results were achieved with the SIFT detector. This detector was able to find a card in 87.5 % of images (175 of 200 images). AKAZE provided similar results, but with a longer time. The ORB detector was slightly faster but with much worse results.

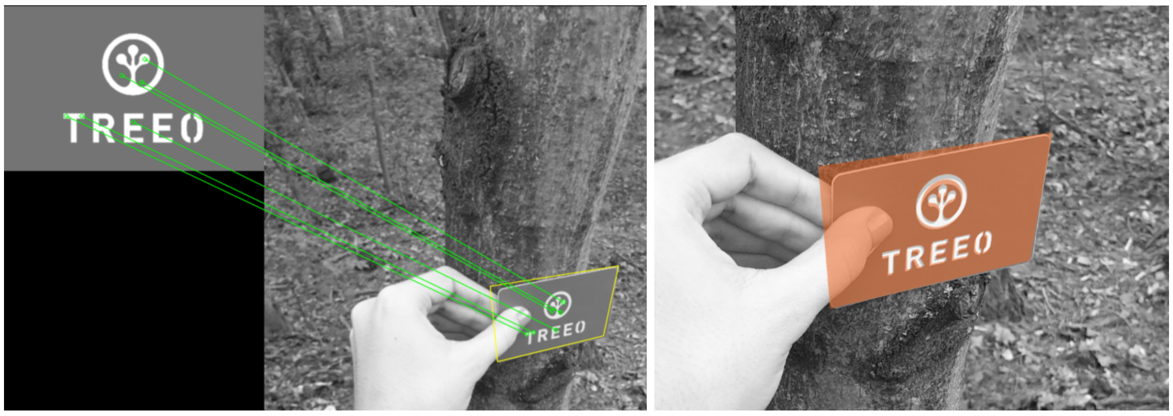
Experiment with ROI

This experiment was done in order to find out how much better card detection will be, if the user will be instructed to place the card in some predefined region on the screen of his smartphone. In the table below, we can see that computing time is rapidly lower. Accuracy has better results too. In the case where ROI was used, card was found in 193 images, which is 15 more than in the case with the whole image. Mean IoU is also slightly higher.

Where card is detected	accuracy	IoU	time [ms]
whole image	87.5 %	0.954	436.41
ROI	96.5 %	0.968	49.15

Experiments with Subpixel local registration

In this method, card is localised with SIFT at first. As we can see in pictures below, SIFT is not accurate enough in some cases and cards will not overlap properly. Our first refinement method finds corners in both cards with subpixel accuracy. These “good features” are used for local registration using Optical flow algorithm. With this algorithm we try to find better overlap for detected card.



First-step card detection with SIFT.

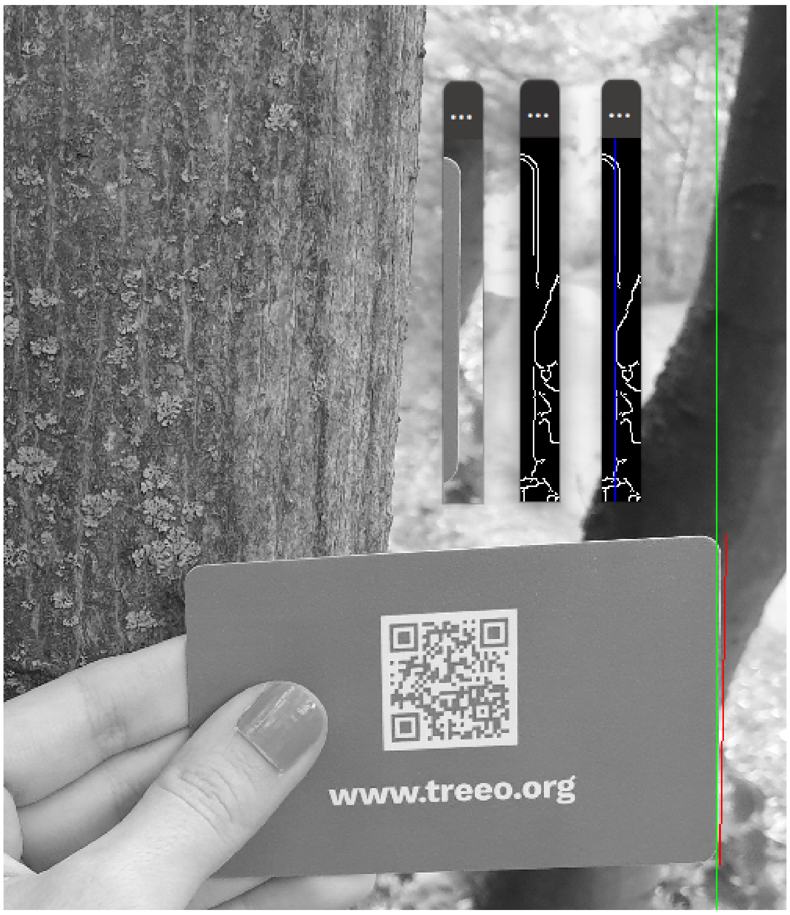
Experiments with Card borders

Like in the first method, the card is localized first by SIFT. Then a small region around each card edge is cut out. Canny edge detection and Hough line transform is done in each region in order to find a better card edge.

Both methods found images where they were able to improve card detection, but also images where IoU dropped. In the end, mean IoU and accuracy wasn’t improved with none of the methods mentioned. So the best solution remains the SIFT detector on ROI without any refinement.

	accuracy	IoU	time [ms]
Without refinement	96.5 %	0.968	49.15
Optical flow	94.5 %	0.964	49.11
Hough lines	95.5 %	0.963	52.02

Table with results of 2 refinement methods.



Example of refinement with Hough lines. Red line is the original edge found by SIFT, the green line is refined edge with our method. Neighborhood of this edge can be seen in the first cut out region, canny edge detection in second and line found by Hough transformation in the third region (blue line which is also green in the whole image).

Tree Detection

The second step of the image-based DBH (Diameter at breast height) measurement is to detect the tree-trunk. Having the card detected and localized, the method is designed to detect the borders of the tree-trunk and avoid the false borders coming from the scene clutter.

Tree-trunk segmentation and localization

Tree detection is based on image segmentation and then estimation of lines that represent the trunk of the tree. The lines are then used to calculate the diameter of the tree. The algorithm assumes that the card is positioned so that its center is approximately in the middle of the trunk. As a result, a smaller part of the image (above or below the card) can be used for segmentation, which makes the calculation faster.

The algorithm is able to successfully detect most tree trunks in the tested dataset from Indonesia. In good lighting conditions with high accuracy.

Conditions for good results:

- the center of the card in the middle of the tree,
- uniform lighting, without shadows and overexposure,
- color difference between background and tree (green background is best),
- straight tree (vertical, not rotated in the image),
- avoid complex/challenging backgrounds - dark color similar to a tree, other trees in the background, wooden stick near the tree,
- tree trunk with a smooth surface works better.

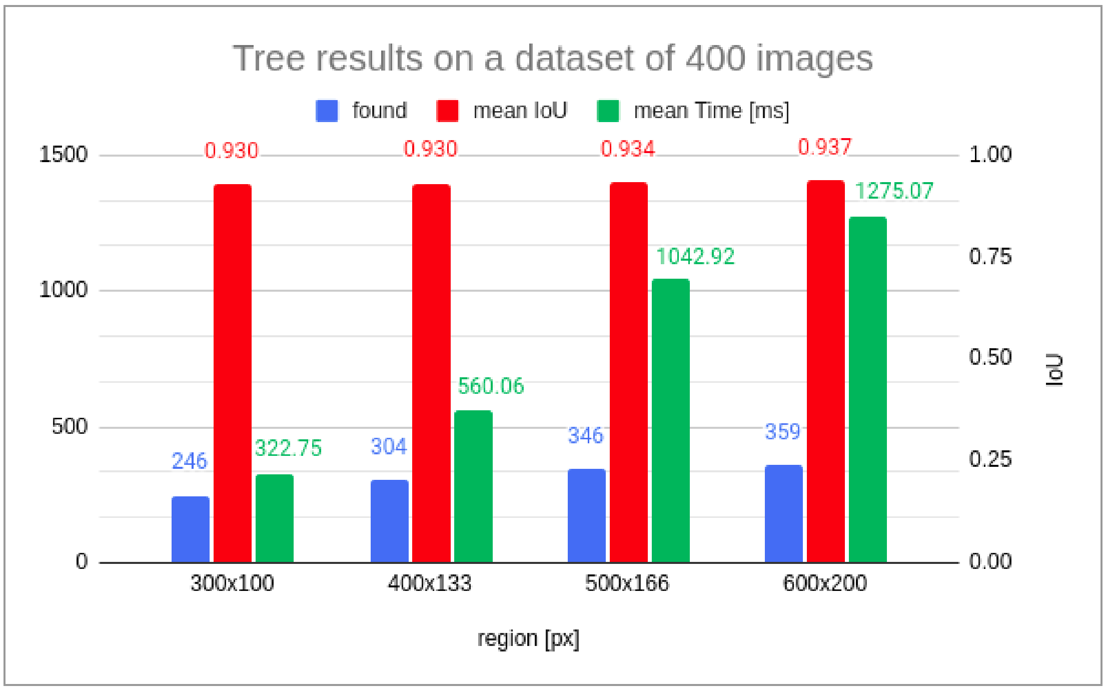
The detection stability and computational cost (time of detection) is evaluated on various experiments:

- 400 images from dataset raw_treeo_dataset_uncleaned, folders b/1/4ref, c/1/4ref, c/1/5ref, c/1/6ref
- Method parameters:
    - Different ROI (region of interest) size - image is resized to width 600px and then an area of height = ⅓ width is selected for tree detection
    - Card points loaded from json groundtruth

ROI [px]	Found trees	mean IoU	mean Time [ms]
600x200	359	0.937	1275.07
500x166	346	0.934	1042.92
400x133	304	0.930	560.06
300x100	246	0.930	322.75

The experiments are evaluated on a computer with the following settings: CPU Intel Core i3-6100U, 2.30GHz, RAM 16GB.

The graph cut algorithm works better on larger images, but is relatively time consuming. The accuracy (IoU - intersection over union) is around 93% but the number of found trees is smaller in smaller images. If suitable conditions are provided (lighting, background), graphcut can work on small images.



Graph presents the overview of accuracy and time-complexity of graph-cut method.

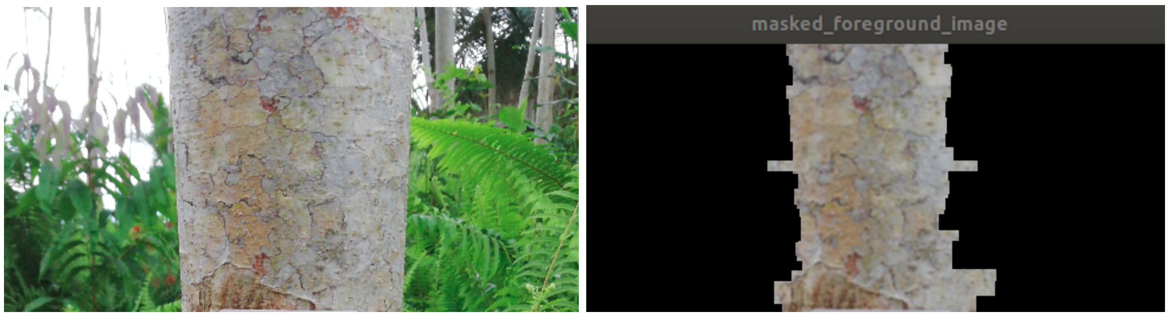
The algorithm works best on trees that are approximately the width of a card. In the selected dataset, the trees were mostly smaller (sometimes only 1-2cm). The main problems with detection occur when there are other trees in the background, bad lighting conditions or challenging backgrounds.

Segmentation optimization

In order to enhance and speed up tree trunk detection, we experimented with different image segmentation approaches and also tried to improve existing GrabCut detection.

GrabCut variations

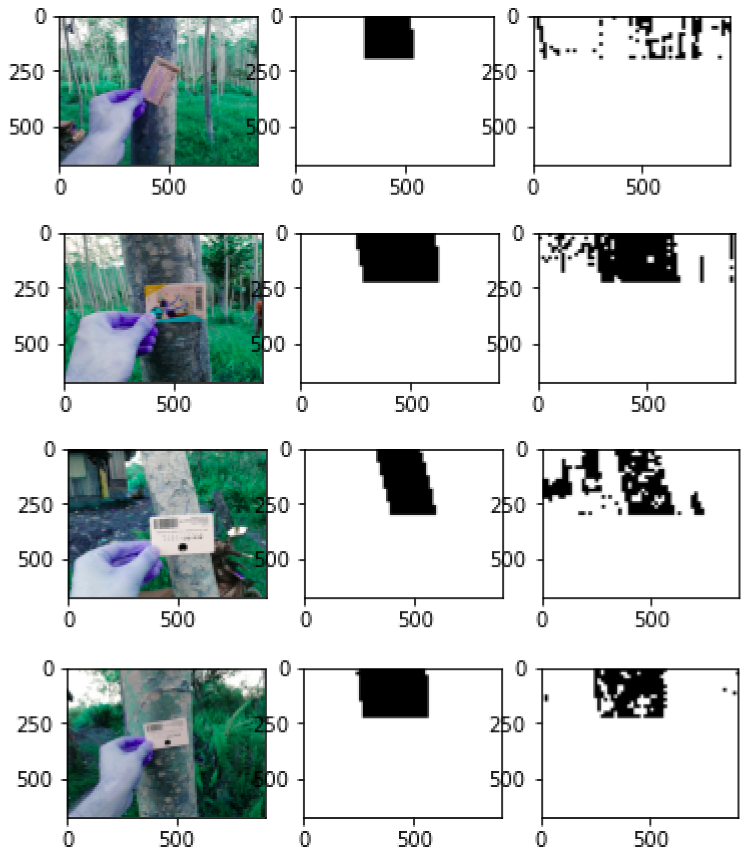
In an effort to speed up the algorithm, experiments were performed with different input image sizes. We tried to use GrabCut line by line and merge individual segmentations into one mask because we expected it to be faster. However, the result of segmentation was significantly worse and the acceleration of the algorithm was only slight (a few milliseconds). An example of segmentation is shown in the figure below.



GraphCut line-by-line optimization segmentation example.

Image segmentation using SVM

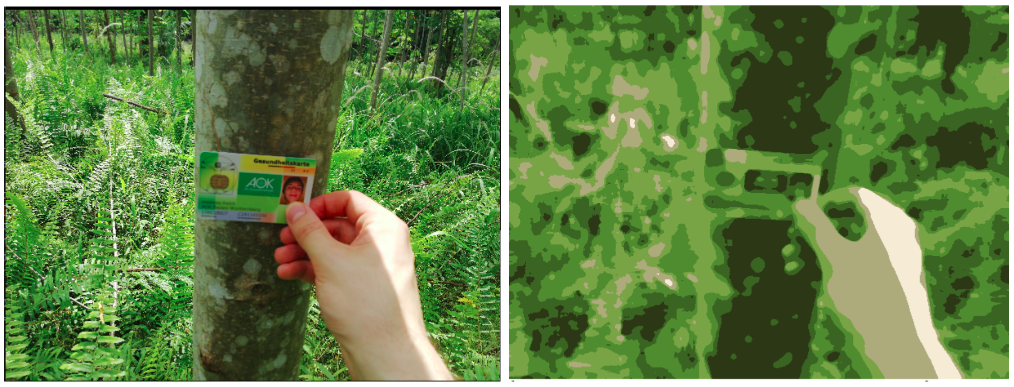
We started segmenting the image using SVM (support-vector machines) with one set of images from the raw_treeo_dataset_uncleaned, where there are similar trees (similar texture and color), and therefore the algorithm could have good results. Each image was divided into small squares from which input data for learning were calculated. We tried different sizes of input image, squares, differently calculated data (color histograms). Only the part of the picture where there is no hand with the card was used, which brought improvements, but the results were still not sufficient. Another machine learning method or different input data would probably be needed for improvement.



Left - input image, middle - input data mask (groundtruth), right - segmentation output

Color-based image segmentation

We have started experiments with color-based image segmentation algorithms such as k-means clustering, but further research would be appropriate in this area.


Left - original image, right - k-means clustering output

As a result, the GrabCut algorithm was used in the library for segmentation, because all of the tested approaches had significantly worse results at this stage. Further research in color-based segmentation and machine learning is needed to replace GrabCut with a faster, at least as accurate method.

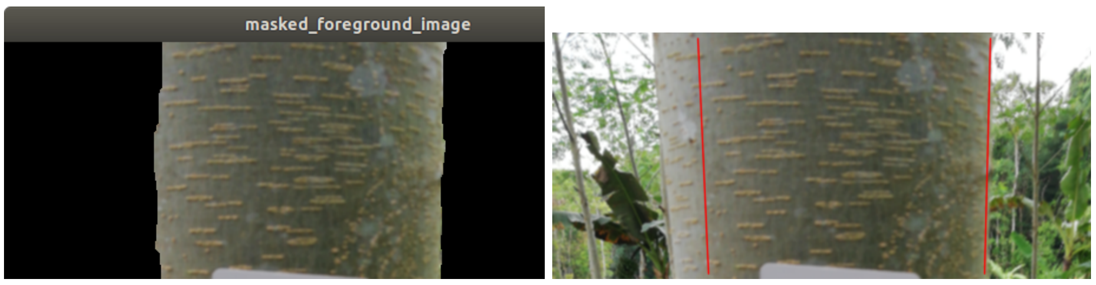
Tree-trunk border optimization

The tree detection algorithm consists of two main parts - image segmentation using the GrabCut algorithm and estimation of lines that represent the trunk of the tree from a binary mask created by segmentation.

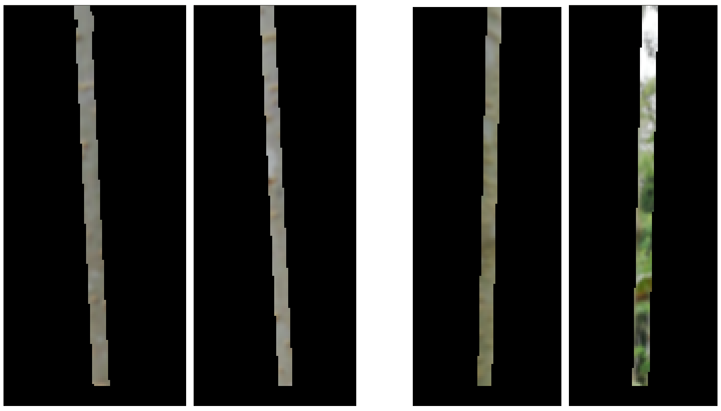
Image segmentation is performed on a 600 pixel wide image for faster computation. Only the area above the card with a height of 200 pixels is used. The segmentation output is then used to estimate the lines of the tree, but the segmentation is not always accurate. In this section, we therefore focused on evaluating whether the found lines correspond to the edges of the tree.

First of all, the lines in the image must not intersect. Next, the angle between the lines is calculated. We assume that the trunk of the tree is straight and too large an angle probably means wrong detection.

When checking the lines, we assume that the tree differs significantly in color and texture from its background. Therefore, a narrow area to the left and right of the line is selected and the average color and color variation are calculated in them. If the line fits well on the edge of the tree, there should be a large color difference in the areas. If the colors and variations are too similar, the line is evaluated as incorrect.

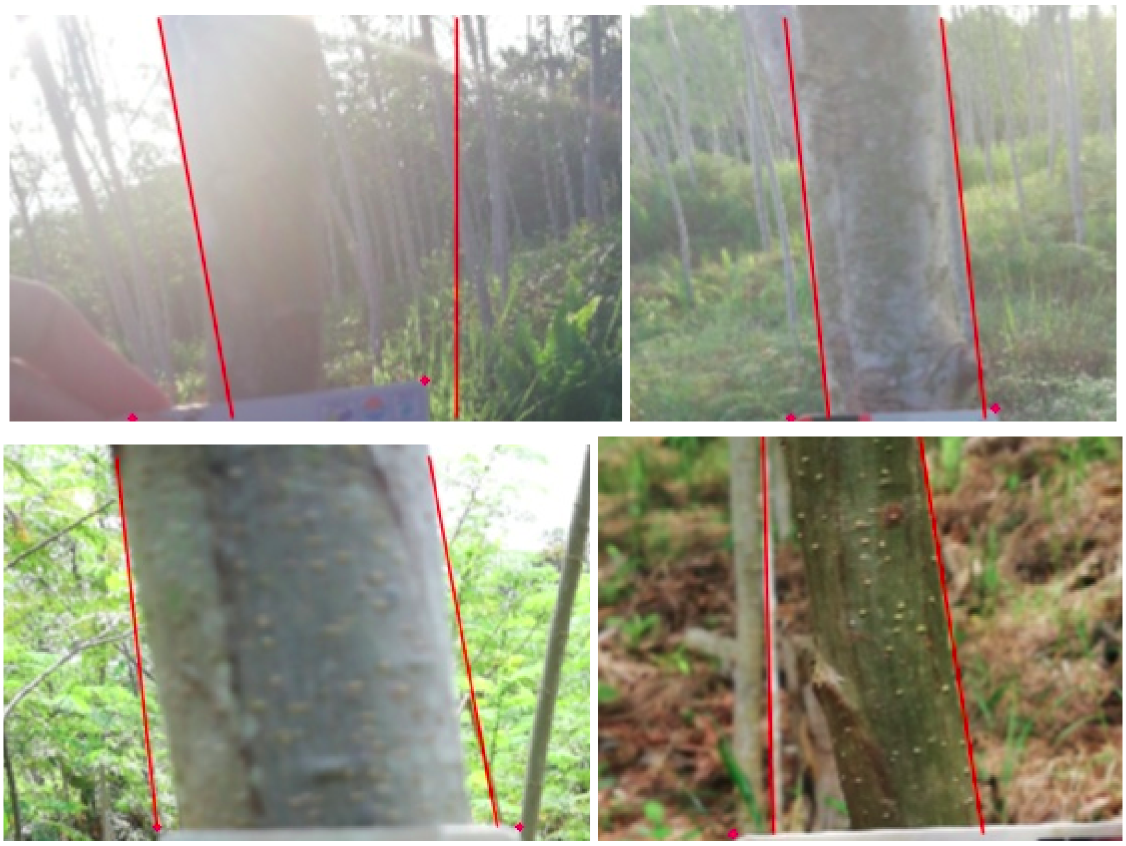


Example of wrong segmentation on left side.

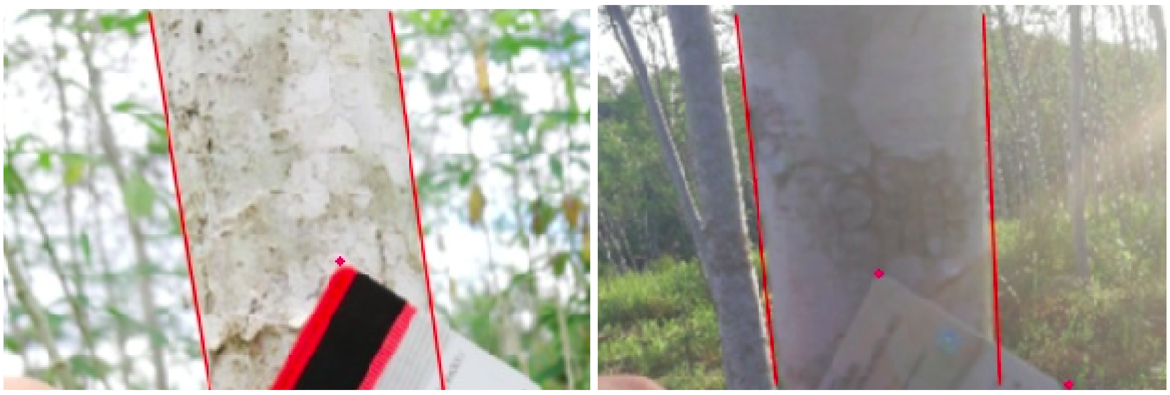


Left: left and right side of the left line - similar colors=wrong detection
Right: left and right side of the right line - different colors=OK

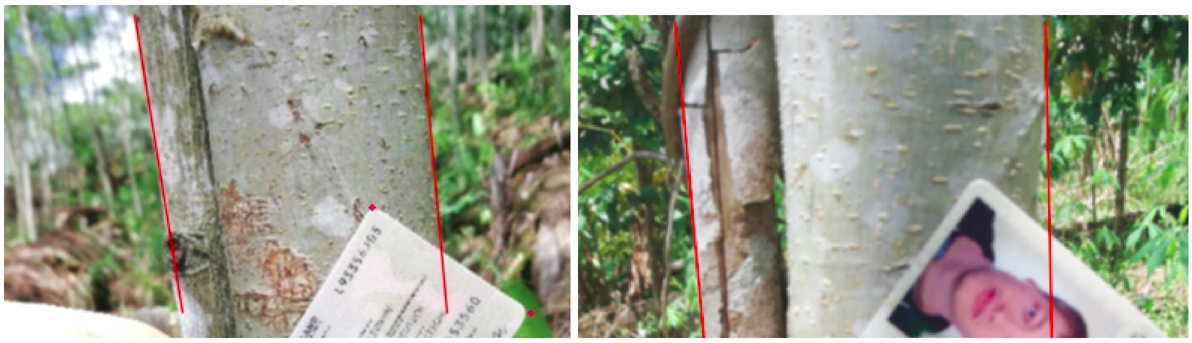
Thanks to these checks, we can eliminate wrongly found lines and thus improve detection. As a result, the algorithm finds fewer trees from the dataset than the original version, but with higher accuracy, because it eliminates images with poor quality (poor lighting, trees in the background). The conditions for checking lines can be easily changed or removed if needed in the future.

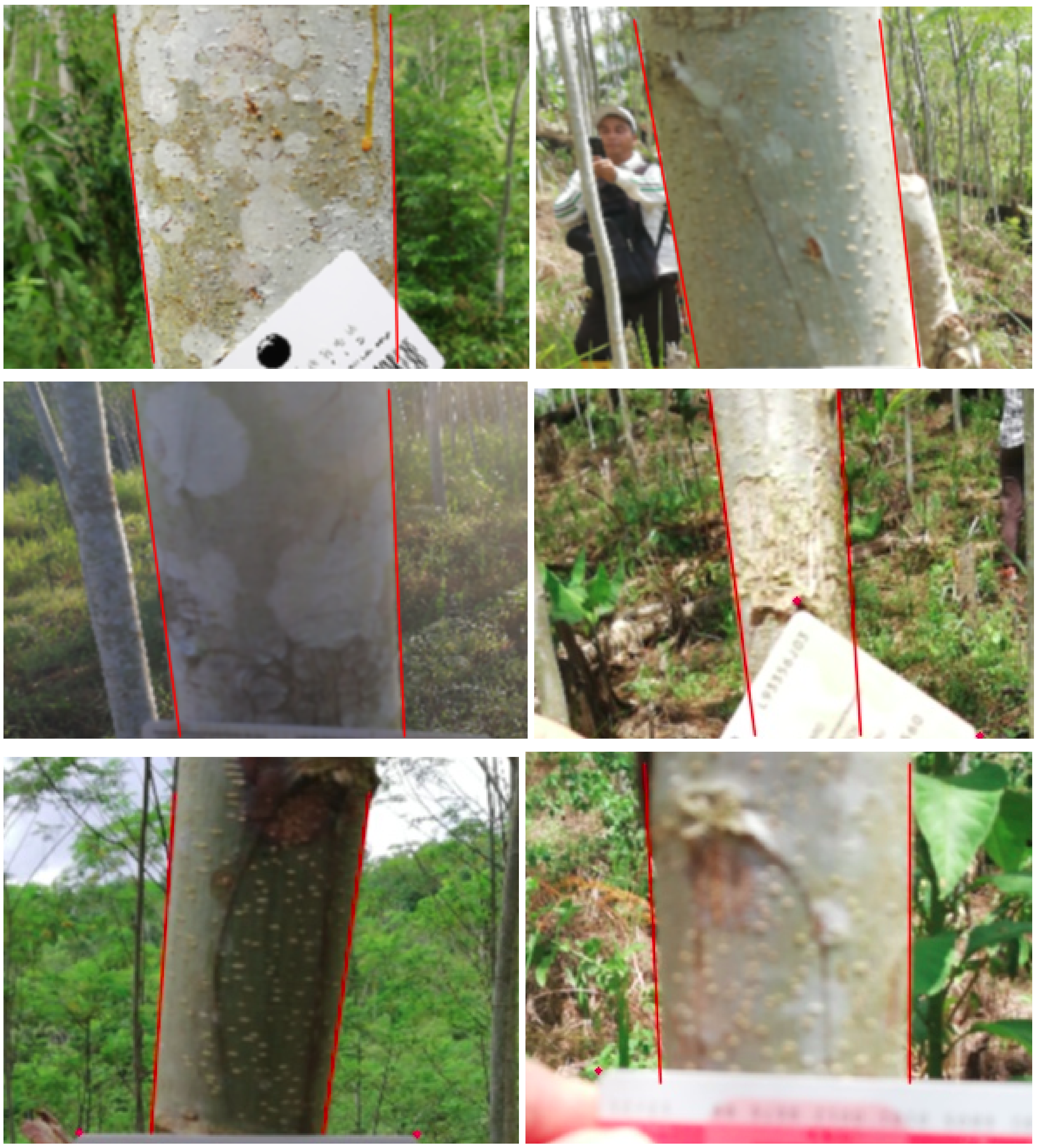


Examples of wrong segmentation (original version of the algorithm), but the lines are in the end evaluated as incorrect (new version with line check) and a new photo will need to be taken.


Despite a good segmentation, these lines are also evaluated as wrong (in the new version of algorithm) because of the small difference in background color, and a new photo will need to be taken.

We also tried to solve the problem if there are two trees next to each other in the photo. But when detecting the edges between the lines, it was not possible to distinguish whether it was an edge between two trees (a tree and a stick) or a texture of the tree.


Examples of wrong detection (evaluated as good lines)



Examples of good detection

version	*mean IoU*	*min IoU*	*std. dev. IoU*	*found trees*
original	0.931	0.326	0.097	356
new	0.955	0.605	0.044	280

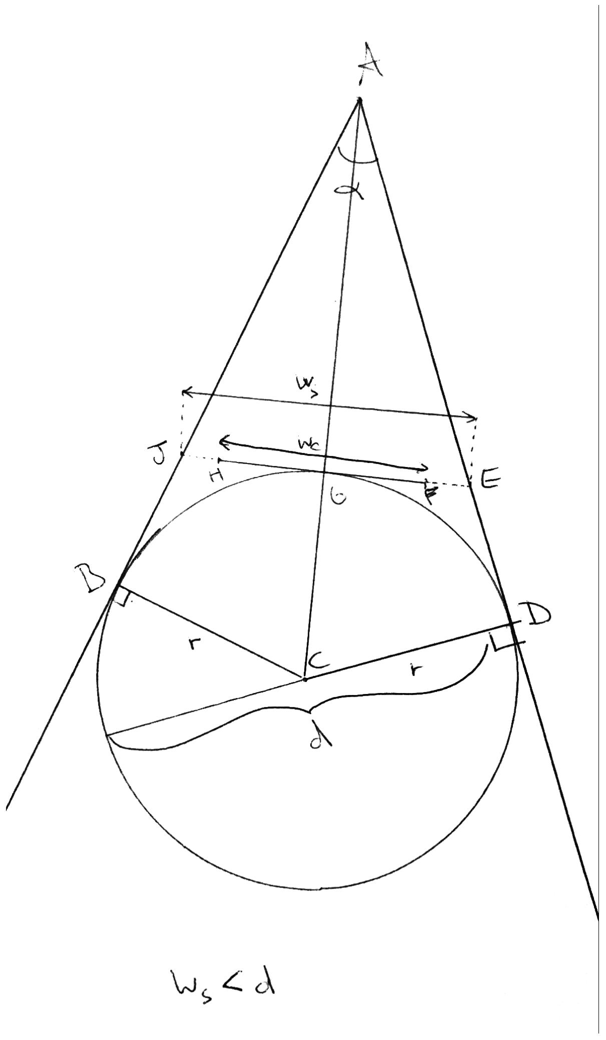
Results on 400 images. There are a lot of pictures in this dataset with poor lighting conditions, two trees next to each other, etc.

version	*mean IoU*	*min IoU*	*std. dev. IoU*	*found trees*
original	0.958	0.551	0.059	264
new	0.967	0.700	0.038	242

Results on 280 images mostly good quality

Diameter Computation

Diameter computation from detected tree edge lines and card edge lines is performed by taking into account the perspective, field of view, and absolute dimensions of the card. This is performed in two steps: linear conversion of pixel count to width, and perspective adjustment.



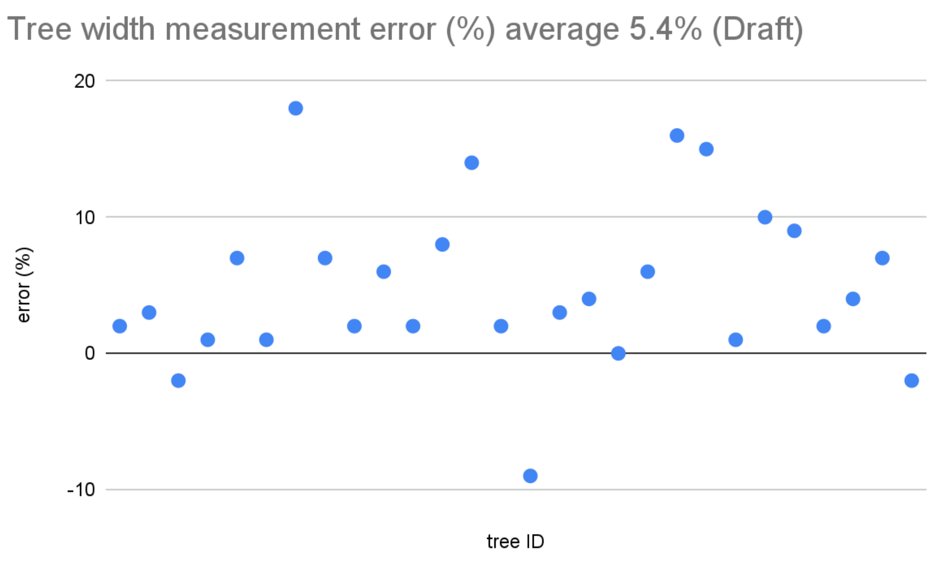
Scheme of the diameter computation - top cut views of the tree-trunk.

As shown here, tree width **d** is computed from perceived tree width $\mathbf{w_s}$. By assuming that the field of view is 81.5°, as is typical of mobile cameras, all parameters are fixed and the conversion can be performed as:

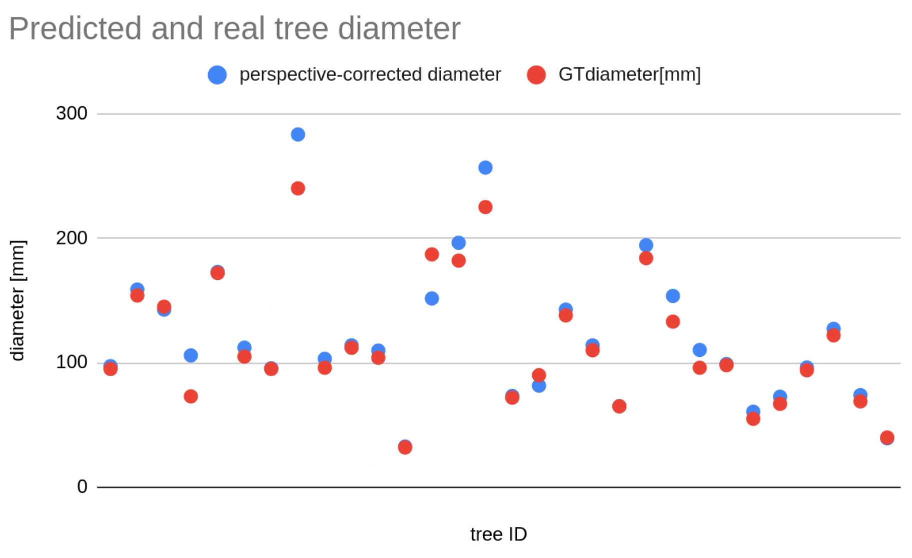
$$d = \frac{w_s}{\sqrt{w_s^2/y^2 + 1} \times \left(1 - \frac{w_s}{2y\sqrt{w_s^2/y^2 + 1}}\right)}$$

Qualitative analysis

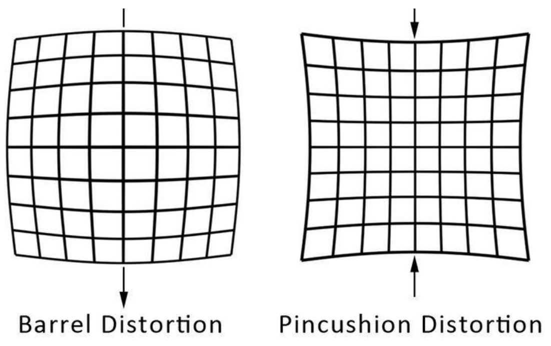
The method above significantly reduces the error of tree width measurement. Before adjustment, the error on the evaluation dataset is 18.5%, and applying this correction results in an average error of 5.4%, corresponding to a reduction of error by 70%. See the following graph for a graphical analysis of errors across the evaluation dataset:



This graph also demonstrates that certain trees can still exhibit an error of 19% in diameter measurements. As shown in the graph below, the largest errors are exhibited by trees with a very large diameter of over 200mm.



These significantly larger errors for wide trees are probably caused by non-linear edge aberrations of the smartphone lens, which are not addressed by this approach. Future work should focus on addressing these distortions, which are notable for smartphone lenses.



Example of lens distortions exhibited by smartphone lenses.

Additional corrections

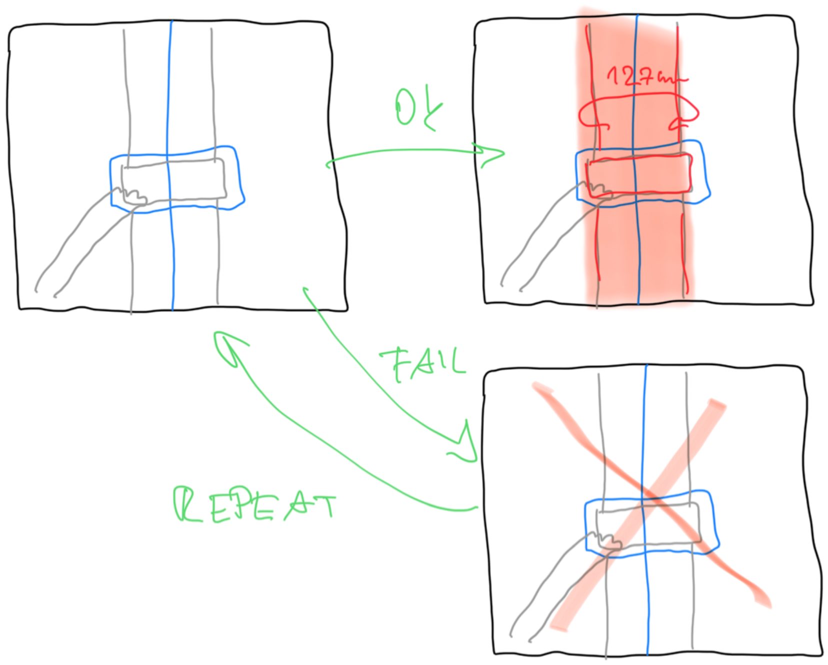
This approach yielded the greatest reduction in error with respect to ground truth measurements. However, correction taking into account the lens field of view (FoV) was also attempted. This was approached by retrieving the FoV parameter from the image EXIF information, allowing the computation with correction of total scene width.

This approach did not yield a further improvement, probably caused by non-linear pincushion distortions of the mobile lens, which is more pronounced on the edges, while the FoV was assumed to affect the scene linearly.

It was also attempted to recover the specific lens distortion information across popular smartphone camera lenses, and this proved unfruitful. Numerous lenses are available, but not all, and not in a consistent format. Therefore, we believe that the correction method presented here is the most robust, and present the maximum practically achievable accuracy of measurement with farmers' smartphones.

Human-in-the-loop approach

In order to improve the robustness of the entire solution, there was also a discussion with experts from the contracting team about possible modifications to the GUI of the mobile application. These were mainly graphic navigation elements that would guide the user to take a picture of the tree and card with the most suitable location and quality.



Mobile app UI design for better card&tree photo taking.

Mobile App UI might help to navigate the user to take the tree picture properly and as close as possible to the ideal position. Below is the proposal of guidance layout when picture grabbing. Showing lines representing the tree center and card region might support the user to align the card and the tree accordingly.

The resulting library and its API take into account the possibilities of these visual navigation elements. The specific location and use of these navigation elements can then be set using the API and used in the calculation of detection, location and measurement.

Treeo-Measurement-Library API

The resulting code is implemented in C++, for easy integration into Android apps as native builds. The code is available in the private repository https://github.com/DCGM/stromy

Structure

Code is structured as five header files with five source files, as follows:

```
CardDetection.h/cpp

ObjectDetector.h/cpp ← main object for integration

SaveBinarySIFTmodel.h/cpp

TreeDetection.h/cpp

TreeDiameter.h/cpp
```

Build

Requires OpenCV 4.5.0. To build as a standalone executable, run the following commands:

```
mkdir results

mkdir build

cd build

cmake ..

make

./treeProject path/to/tree_image
```

Integration

The function to call is `measureTree()` in `ObjectDetector`. For more information, see documentation in the comments.

Bark-based tree classification

A feature which would be useful for the computation of standing wood value is the automated detection of tree types, and since the images to be taken are of the tree near its base, we have also explored the topic of tree bark classification. Can we detect the different types of trees in Indonesia and Uganda from single images?

Known approaches and Datasets

The most successful available methods[1] rely on two important assets: a deep neural network and a tree databank. The selected network is the widely used ResNet[2], with experiments performed here on the small ResNet-18, which has 18 residual layers. Because the deep network used for this task contains a total of 11 million parameters, it is not fit to be evaluated on low-end devices, and requires a GPU to perform evaluations under a second.

Furthermore, it requires a vast dataset to train. The *BarkNet 1.0* dataset created at Université Laval in Canada contains 23 thousand distinct images of tree bark, for 23 different tree varieties. In order to train the network, these images are augmented by random cropping, geometry transformations, changes in color, and addition of noise to create sufficient variation to train the classifier. The published classification accuracy of this classifier is 97.81%, when evaluated through majority voting.

The dataset used here was composed of BarkNet 1.0 and images from FairVentures:

source	Species	Common name
BarkNet 1.0	Abies balsamea	Balsam fir
BarkNet 1.0	Acer platanoides	Norway maple
BarkNet 1.0	Acer rubrum	Red maple
BarkNet 1.0	Acer saccharum	Sugar maple
BarkNet 1.0	Betula alleghaniensis	Yellow birch
BarkNet 1.0	Betula papyrifera	White birch
BarkNet 1.0	Fagus grandifolia	American beech
BarkNet 1.0	Fraxinus americana	White ash
BarkNet 1.0	Larix laricina	Tamarack
BarkNet 1.0	Ostrya virginiana	American hophornbeam
BarkNet 1.0	Picea abies	Norway spruce
BarkNet 1.0	Picea glauca	White spruce
BarkNet 1.0	Picea mariana	Black spruce
BarkNet 1.0	Picea rubens	Red spruce
BarkNet 1.0	Pinus rigida	Pitch pine

[1] Carpentier, Mathieu, Philippe Giguere, and Jonathan Gaudreault. "Tree species identification from bark images using convolutional neural networks." *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018.
[2] He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

BarkNet 1.0	Pinus resinosa	Red pine
BarkNet 1.0	Pinus strobus	Eastern white pine
BarkNet 1.0	Populus grandidentata	Big-tooth aspen
BarkNet 1.0	Populus tremuloides	Quaking aspen
BarkNet 1.0	Quercus rubra	Northern red oak
BarkNet 1.0	Thuja occidentalis	Northern white cedar
BarkNet 1.0	Tsuga canadensis	Eastern Hemlock
BarkNet 1.0	Ulmus americana	American elm
FairVentures	Paraserianthes falcataria	Sengon

Older approaches have significantly lower accuracies, but require significantly less processing power, thanks to their use of Wavelets and SIFTs[3].

Adaptation to new varieties

We hypothesised that the same accuracy could be reached on tree varieties seen in Indonesia. Therefore, the BarkNet 1.0 dataset of tree varieties present in Quebec, Canada, was extended with 35 Sengon trees photographed in Indonesia. These 35 were additionally split into 25 for training and 10 for evaluation.

Training and evaluating the Resnet18 architecture on these yielded the following results: A classification accuracy of 94% for all classes, and 90% on Sengon trees in particular (9 out of 10). The lower accuracy across all classes in comparison to the published results is explained by the lack of use of majority voting, and different training conditions. However, the high accuracy on Sengon trees has low statistical significance, because there were only ten in the evaluation dataset.

Furthermore, classification inaccuracies for such a small dataset may be caused by different factors. For example, by analysing the EXIF data it was discovered that the incorrectly classified image came from a different camera than the others. Similarly, lighting conditions are likely to affect these results, but these are controlled for well in the larger *BarkNet 1.0* dataset.

In conclusion, it may be stated that tree identification from bark is possible for trees captured on smartphones. However, a significantly larger dataset will be required, of around 1000 trees for each class. For this number of training samples, we expect an accuracy between 98% and 99%.

It may be of interest that an exploration of the literature has also yielded methods for *Tree Re-Identification from bark images*[4]*.* That is, images of the same tree taken months apart could be used to detect whether the same tree is being measured, to provide per-tree growth estimates.

[3] Fiel, Stefan, and Robert Sablatnig. *Automated identification of tree species from images of the bark, leaves or needles*. na, 2010.
[4] Robert, Martin, Patrick Dallaire, and Philippe Giguère. "Tree bark re-identification using a deep-learning feature descriptor." *2020 17th Conference on Computer and Robot Vision (CRV)*. IEEE, 2020.

Conclusion

The aim of the project commissioned by Fairventures Worldwide FVW gGmbH was to design and test computer vision methods for image-based DBH measurement respecting the requirement for effective functionality also on older mobile devices.

The measurement procedure was divided into three steps: detection of a known card in the image, detection of the edges of a tree trunk, and calculation of the diameter of a tree trunk. For each of the steps, particular methods were designed concerning the low-computation cost requirement and memory limitations. The methods were statistically evaluated, fine-tuned and the results presented and discussed. In addition, the individual methods have been optimized and include additional options to manually balance the accuracy or computational costs when integrated into the particular application. The resulting implementation creates the designed library containing the final methods. Part of the work was also the close cooperation on the integration of the library into the mobile application environment developed by the client's development team.

The research also included a study and the possibility of adapting modern machine learning techniques for recognizing the type of tree from photography. A suitable existing published procedure was used as a basis for the solution and extended with new data from Indonesia. The results show the applicability of this approach when more relevant data is provided.

Part of the solution was also the realization of new annotations on parts of the provided dataset by the client, including the acquisition of additional datasets and their annotations.

All additional approaches, extended experiments, and results are part of this documentation.

Example of card and tree-trunk detection.

Appendix

Smartphone minimal requirements

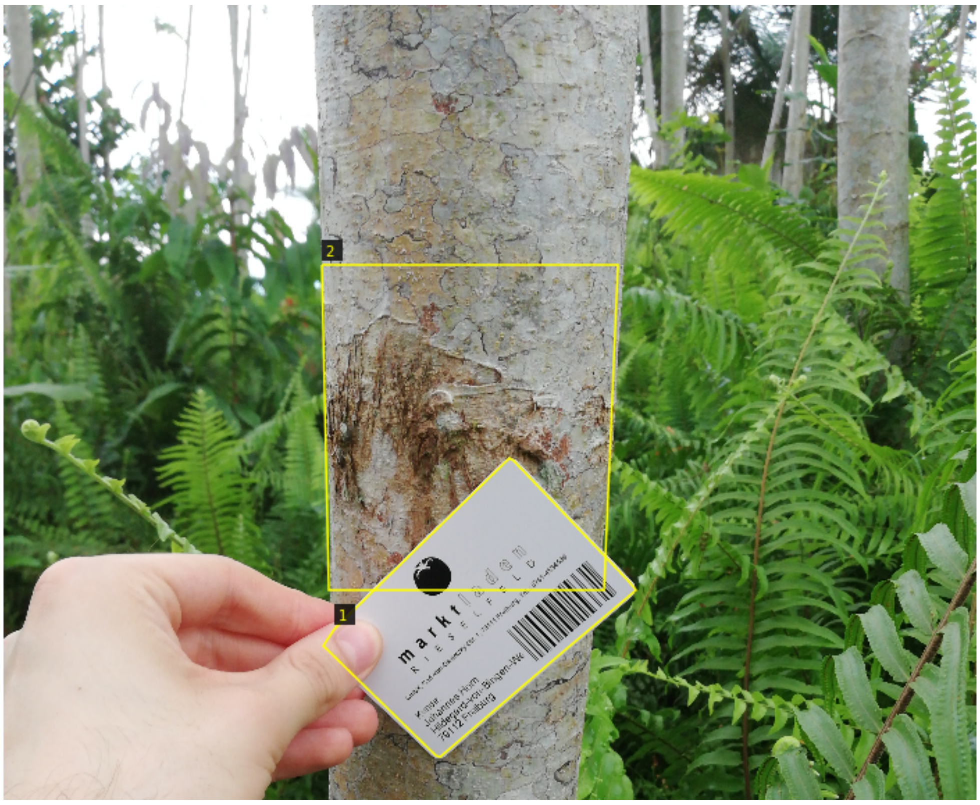
Runtime environment:

- Android: version 5.1 and higher

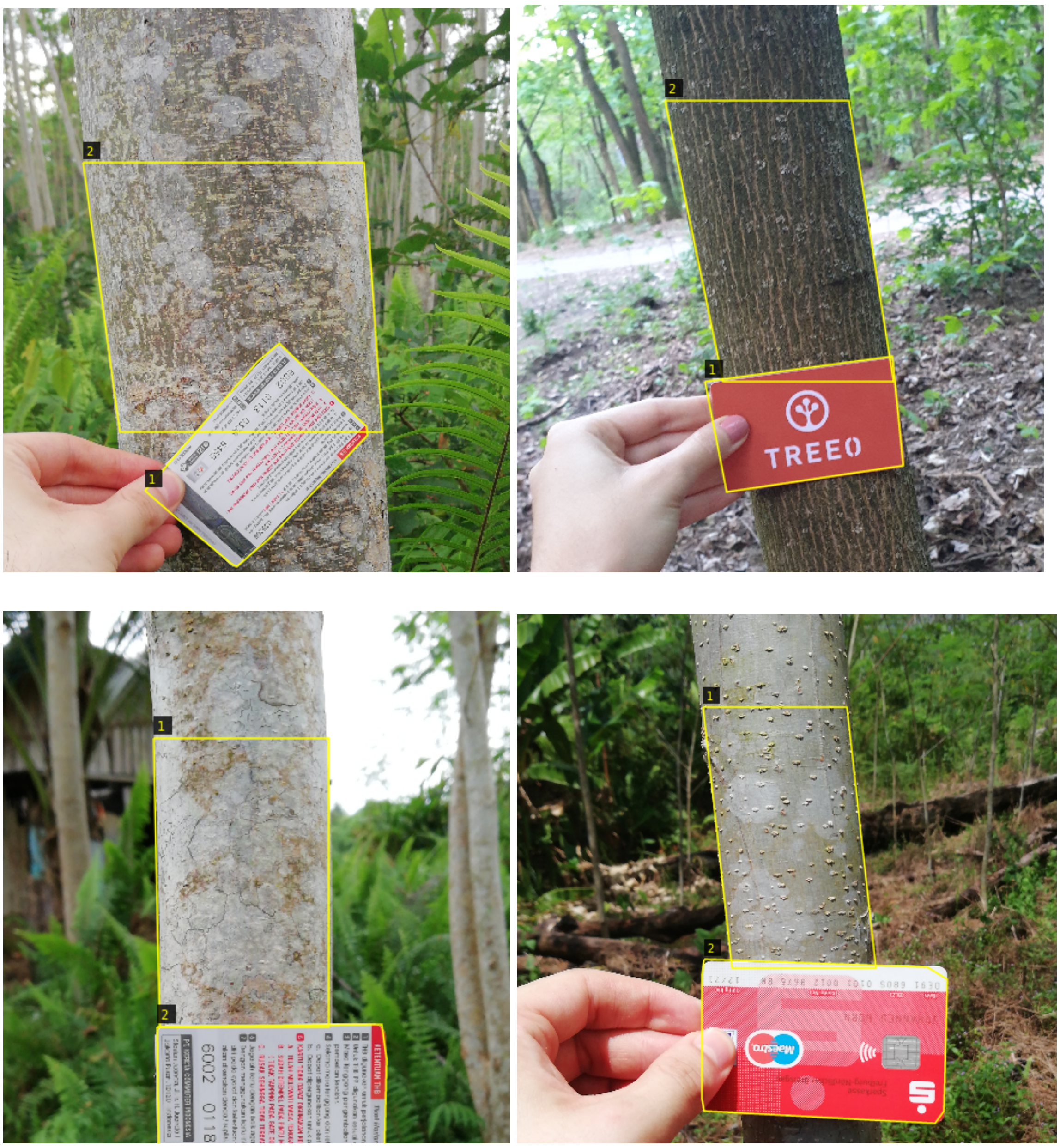
- 1 GB RAM

- 5 MP camera

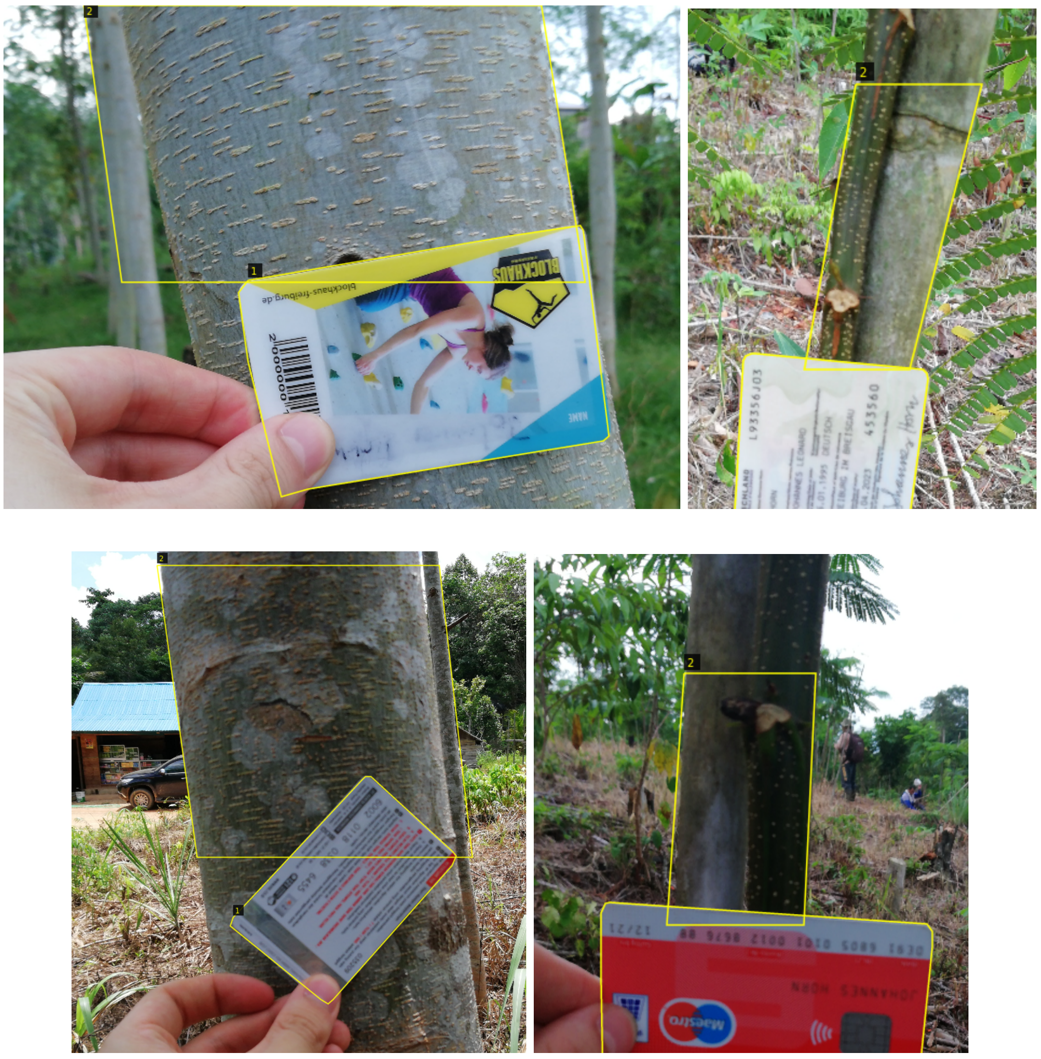
Gallery of example results



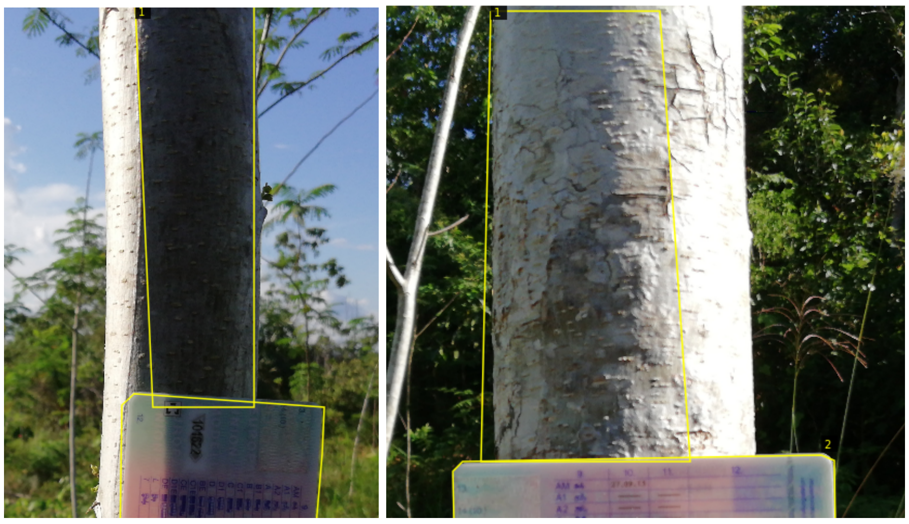
Correct detection, proper conditions.



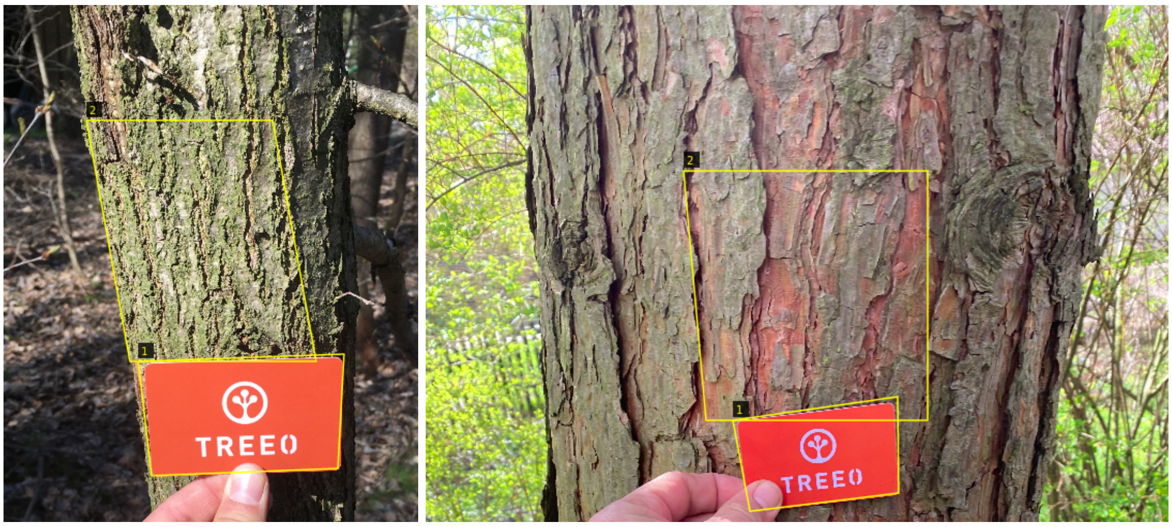
Correct detection, proper conditions.



Tree(s) (or wooden stick) too close to the trunk or on the background



Lighting conditions, sharp highly-contrast shadows



High-color distribution on the bark