Foliage detection project

Iuliia Pliushch[1]*, Dennis Soukup[2]*, Wolfgang Stammer[3]*

Abstract
This is the summary of the foliage detection project. The aim is to construct the context model of foliage detection on the ground by means of a drone with a camera, create artificial data with blender scripting on its basis, design an algorithmic pipeline for foliage detection on those artificial images and analyze its performance by relating it to the parameters of the model.

Keywords
contextual models, blender scripting, foliage detection

**Goethe University, Frankfurt am Main*
[1] ipliush@stud.uni-frankfurt.de
[2] s4811888@stud.uni-frankfurt.de
[3] s5401424@stud.uni-frankfurt.de

Contents

Introduction

This project focuses on creating a context model of foliage generation and distribution in a private property, generating graphical data based on this model and training a machine learning system to recognize foliage in the generated images. The project thus covers the subject of task specific context and generative modelling, systems design and finally systems engineering. The inspiration for this project was driven by the software engineering class. While the *software* engineering class emphasizes the practical side of the project (flying the drone, gathering real data with it and detecting foliage on its basis), the *systems* engineering class emphasizes the context model creation that should be a preceeding step for the software engineering. The relationship between real and simulated data is more profoundly considered in [**?**].
We begin our project by precisely defining the task and context model parameters. We next create graphical models of the world in order to generate realistic graphical images using blender scripting. Here it is important to note that the images are realistic not based on all world parameters, but only those we assume to be important for foliage generation and distribution and which we assume to lead to important features for algorithmic detection. We then consider which parameters of the context model provide sufficient amount of information about foliage in order to construct an algorithmic pipeline to create classified images (classified into foliage and non-foliage).

1. Context model

1.1 Enlistment of context parameters

In this section we describe the specific task of our system, the context model parameters and graphical models.
General setup: The case that we are exploring here is that of a drone flying over some private property which contains a house, a yard and trees.

Our context model (see figure 1) will contain three sets of parameters:

- the parameters of the assignment (task) that is to be solved
- the parameters of the drone (most importantly parameters of the camera)
- the paramters of the (generative) world model

1. Task: Foliage detection:
   (a) 2D foliage detection: Detection on the basis of only **one** picture
   (b) 3D reconstruction: Use sequences of 2D images from the flying camera to reconstruct the 3D surface and detect foliage there. Here, camera trajectory is imporant.
   (c) *What*: Tree and bush foliage
   (d) *Where*: Detection of fallen leaves on the ground
   (e) *When*: Real time or not, offline or online
   (f) Success criterions:
      i. Detection step (our): Percentage of correctly as belonging to foliage classified pixels. Alternatively one can also take the trade-off between performance and processing time.
      ii. Overall: The ground is cleared from fallen leaves (foliage).
2. Camera:
   (a) Parameters: pixel resolution, type of lens and focus, saturation, zoom factor, motion blur, lighting type, lens staining (e.g., dependent on weather)
   (b) Flying trajectory
3. Scene:
   (a) Objects:
      i. **Ground**
         A. Geometric form (if square, its width and breadth; if circle, its radius)
         B. Surface type (e.g., gras vs. asphalt) and properties (leaf-like color or not)
      ii. House (we do not consider variation of this object type)
      iii. **Tree**
         A. Object properties: tree height, diameter of the tree trunk, diameter and amount of branches, amount and type of leaves, color (different interpolations of colors, except for blue) etc.
         B. Object location distributions:
            - single distribution, e.g. Gaussian
            - multiple distribution: e.g. trees near the boundaries of the property and right near the house
      iv. **Foliage** (leaves on the ground)
         A. Object properties: size, geometric form (e.g., curvature of the leaf boundary and curvature of the leaf plane) and structure (leaf skeleton type, symmetry), shadow-rich areas of high contrast [**?**], randomness of edges, varying color (yellow-brown-red)
         B. Texture properties: reflection and rough texture
         C. Object location distributions: e.g. Gaussian
   (b) Conditions:
      i. Weather: sunny, rainy, foggy, wind
      ii. Time of the day: day, night, sunset, sunrise
      iii. Season (time of the year): light intensity, snow

Figure 1. Context model parameters

Below we will briefly summarize some of the parameters and properties stated in figure 1.

Task: The task is to detect (and remove) foliage that has fallen from the trees, in other words, fallen leaves. We will restrict our summary and analysis to the subtask of fallen leaves detection, not removal. As input for this subtask, one can take either a single image or a sequence of images generated at certain time intervals at different angles. Our success criterion is the percentage of correctly classified pixels.

Drone/Camera: The drone contains a camera with preset parameters and can change its flying trajectory.

World: Ground properties: We consider a plane with different textures and colors, for example a combination of grass and asphalt.

World: Tree properties: Several properties of trees vary depending on the time of the year, for example the amount of leaves, their color etc. Others depend on the tree type, for example the diameter of the tree trunk.

World: Tree location: Tree location may stem from different combinations of distributions. Natural tree location can be approximated with a Gaussian distribution, but man-made gardens may also contain regularities, such as planting the tree at the outer boundary of the property and near the house.

World: Leaf properties: Leaf properties may vary depending on the tree type, e.g. size, structure, geometric form. Others may depend on the time of the year, e.g. color. Groups of leaves build together shadow-rich areas. Leaves are symmetric and contain a periodic skeleton.

World: Leaf location: Leaves can fall off the trees and, hence, distribute over the property such that right below the tree the amount of leaves from that tree will be greater than

further away. One possibility is to assume that leaves are Gaussian distributed with respect to the trees that they fall off. This may be influenced by weather conditions, such as wind which would cause scattering of the leaves if strong enough.

Conditions Light conditions can be influenced by the weather and season of the scene, e.g. the light intensity and angle of the sun. Other weather conditions may influence some properties of leaves listed above, e.g. long dry intervals may create a crumbling effect of the leaves, rain may create reflection of the light from the leaves.

1.2 Leaf modelling

Figure 2. Scene objects

Modelling the scene objects would need to start with choosing the *location* and *properties* of the ground, then doing the same for the house, road, trees and, lastly, leaves (see figure 2). The location and properties of the scene object that is constructed first, constrain all the other objects generated later. This is because, given the constraint of where the house is located and how big it is (a property of the house), the trees could not be at the same position as the house.

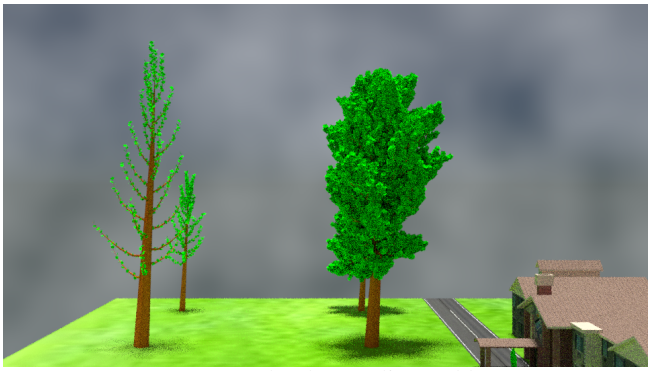


Figure 3. Blender Sapling Add-on

1.2.1 Foliage distribution on the ground

Let us start modelling the *location* and the *attributes* of fallen leaves, because they are the most important scene object, in the light of the foliage recognition task. For tree modelling, an already existing Sapling Add-on of Blender can be used, see 3. As for the distribution of the location of the leaves, a *spatial stochastic point process* can be used. It may be described such that the ground is dissected into subparts (e.g. a grid), for each of which a probability of falling leaves can be assigned according to a certain distribution. In the case of leaves, a bivariate (2D) mixture of Gaussians (see 4) builds a good starting point for describing such a distribution.

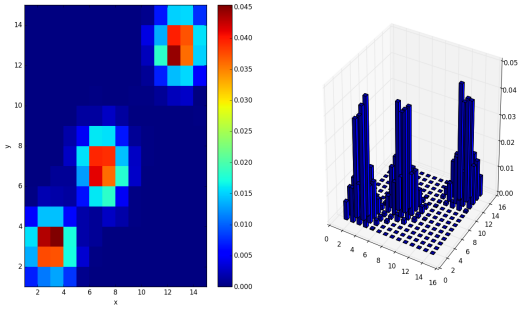


Figure 4. Bivariate mixture of Gaussians with three trees: Plot of fallen leaf samples for given δ x and δ y

Given the position of trees, the distribution of foliage on the ground would be a (modified) bivariate (2D) mixture of Gaussians with the *means* of those Gaussians being determined by the locations of the trees and the *(co-)variances* by the weather conditions (e.g. high variance when it is windy) and topological constrains (e.g., leaves gathering near fences or outer walls of houses), see figure 6. The modification of this probability distribution is needed, because though leaves are more often found under the tree, than far away from it, the proximity to the tree is bounded by the tree trunk radius. In other words, there will be no fallen leaves *inside* the tree. Figure 5 shows one such weighting possibility by means of a sinus: a_i, as well as the *variance* of the Gaussian component are varied such that one can model, respectively, different sizes of the tree trunk and different amounts of leaves that one tree has.

The amount of leaves that are sampled from the distribution of a single tree depends on the initial amount of leaves on the tree, conditioned on the tree type, and the time of the year.

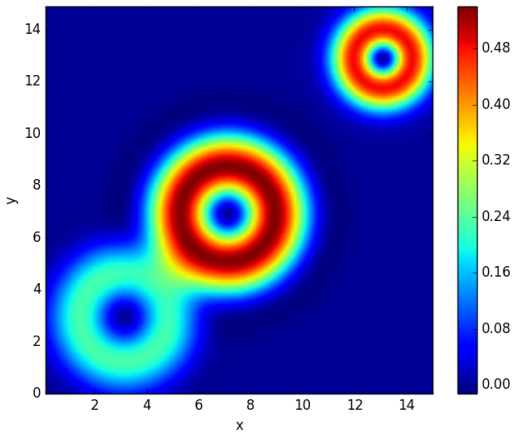


Figure 5. Bivariate mixture of Gaussians, where each component is weighted by $a_i * k * sin((mean_i - x)^2 + (mean_i - y)^2)$ with $k = 0.2$

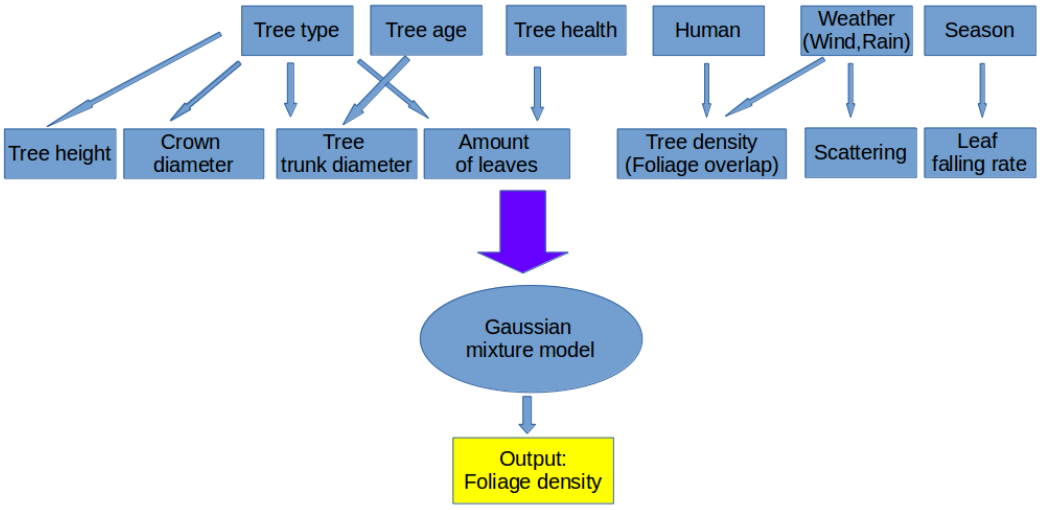


Figure 6. Main parameters contributing to the Bayesian model of foliage location

1.2.2 Foliage and leaf geometry

Depending on foliage density, leaves may be scattered or clustered, see 7. Therefore, it is necessary to distinguish between the geometry of a single leaf and clustered leaves.

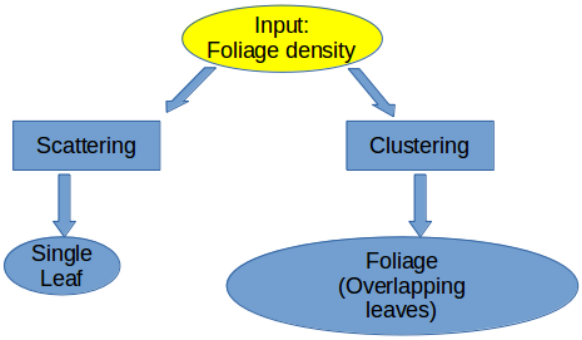


Figure 7. Foliage geometry as a result of either scattering or clustering

Regarding leaf properties, one can distinguish between their *geometry*, as well as *texture* properties. Since our task is to detect several leaves, the fact that fallen leaves may *overlap* will also come into play. In [**?**], the distinctive shape of leaves is used as a sole recognition cue for categorizing them (establishing their type). This is because their color or venation pattern are too variable. Leaf color varies depending on the season and weather conditions. The venation pattern is visible only when the resolution of the camera is good, see 8.

The geometrical and textural properties of a single leaf are described in figure 8. The geometrical and textural properties of foliage include:

- random edges/high variance of edge directions
- minimum edge density
- a shadow-rich area

Given that there are at least two ways, in which one might represent foliage: either as a *curved surface*, or as a *point cloud*, we have, in agreement with [**?**], decided to use the second option, because leaves may consist of smaller components or clusters (e.g. pine needles), such that the curvature would be difficult to compute. From the point cloud, the leaf contour as an "array of contiguous point locations" can be computed and the scale-dependent leaf curvature can be subsequently extracted, see [**?**].

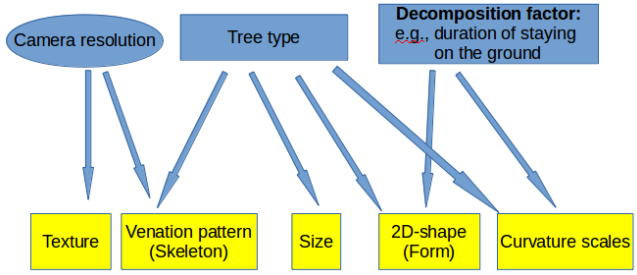


Figure 8. Main parameters contributing to the Bayesian model of leaf geometry and texture

1.2.3 Foliage photometry

Photometry is influenced by motion blur (1), illumination (2) and specular reflection (3), while foliage color distribution depends on the latter two, see 9. Importantly, the leaf color changes with the time of the year (season). Finally, weather can increase or decrease the amount of specular reflection.

2. Realization of the model in Blender

Though our graphical models are already a simplification, we further simplify them for our foliage recognition task. This is because we assume that some parameters are more important than others for foliage recognition. We hypothesize that important leaf detection features are leaf form/geometry, leaf location (namely on the ground and not on the trees), color distinction with respect to background, sufficient amount of random edges and shadow-rich areas. Thus, we vary only

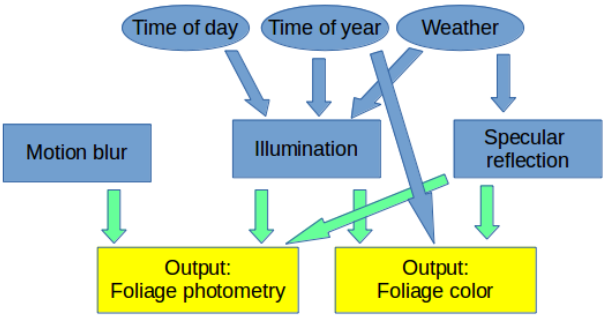


Figure 9. Main parameters contributing to the Bayesian model of foliage photometry

those parameters in our graphical models that are most important for the mentioned detection features. Here we mainly focused on the parameters determining the construction and location of individual leaves. Such other parameters of our model as, for example, "tree health" and "tree age" have been ignored. As for "tree type", we have introduced randomization into some tree parameters, as the number of branches and the trunk diameter. Nevertheless, those randomized parameters do not necessarily coincide with a certain real tree type.

2.1 Simplification of graphical model parameters

Following parameters of our graphical models we have further simplified or excluded:

Tree location: Instead of modelling a complex arranging of trees on the property, as a first step we use only fixed positions.

Tree type: As described above we randomly produce trees by randomly varying the parameters of the sapling add-on add_tree() function. Thus every tree looks different and has individual parameter values, but these do not correspond to real tree types.

Tree age, Tree health, Season: These parameters were ignored entirely in the blender scripting.

Amount of leaves: The parameter "Amount of leaves", similarly as "Tree type" we have modelled by randomly varying the tree parameters in the add_tree() function. That is we do not vary this conditioned on the season.

Leaf falling rate: This is modelled as a gaussian distribution without considering the time of year.

2.2 Leaf geometry

In order to keep the geometry and form of leaves as realistic as possible we have chosen to model leaf geometry by using photographs of five different kind of leaves, loading them in Blender and changing their color. This way, we can benefit from the realistic leaf form and texture. Furthermore, the geometric object to which we add the leaf texture is a grid. We randomly add noise to grid points such that our leaf is not a pure 2D plane, but a randomly *bent* one, see 10.

This way not only bending, but also crumbling of leaves can be realized, if the noise degree is sufficiently high. We

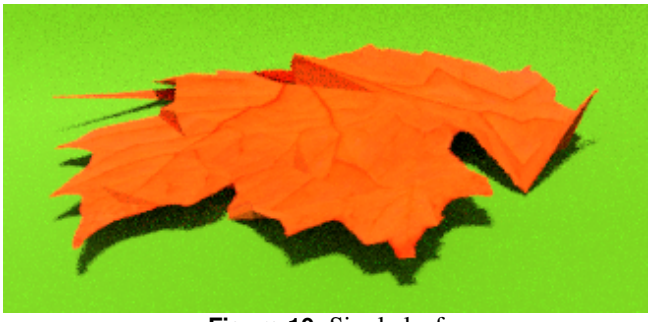


Figure 10. Single leaf

assign to each tree a leaf type, so that the distribution of leaves looks realistic.

2.3 Leaf distribution

We have simplified the distribution we have described previously such that for each tree, we sample from a multivariate gaussian with the tree location as our mean, but instead of the sinusoidal function to push the leaves out of the tree trunk we just sample x,y positions outside of the trunk diameter and within the property boundaries. Scattering of the leaves, for example by wind, we model by varying the variance and covariance of the multivariate distribution.

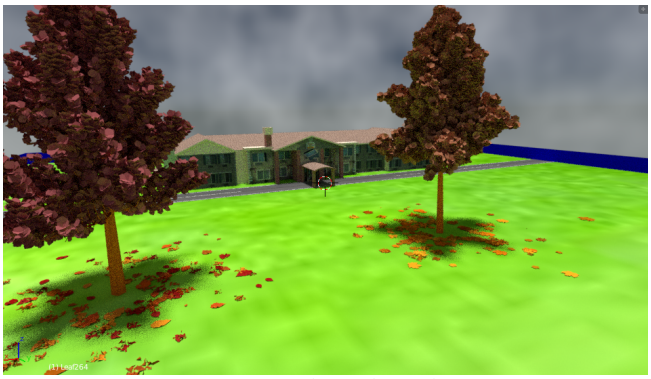


Figure 11. Coloured tree leaves

2.4 Tree geometry

The tree parameters that we manipulate during tree generation with the Sapling Add-on are:

1. tree trunk diameter
2. tree height
3. number of branches
4. number and color of fallen leaves

We model the parameters above by means of normal and uniform distribution.

Our assumption is that for our task it is not important that leaves on the trees and on the ground have the same geometric form and texture. We use the predefined geometric leaf shape from Sapling for the trees, but use a realistic form for the leaves on the ground.

1. **scene geometry**
    - background: *medium entropy*
    - ground plane: *low entropy*
2. **scene material**: *medium - high entropy*
3. **leaf geometry**
    - size: *low entropy*
    - color: *low - medium entropy*
    - form: *low - medium entropy*
    - texture: *medium entropy*
4. **leaf material**: *medium entropy*
5. **environment**
    - illumination: *high entropy*
    - weather: *high entropy*
6. **camera geometry**
    - position: *low - medium entropy*
    - internal parameters: *low entropy*
7. **camera photometry**
    - noise model: *low entropy*
    - transfer function: *low entropy*

Figure 12. Hypothesized information content of the context variables for the detection task described using entropy.

2.5 Color adjustment

We adjust the color of the leaves on the tree to the color of the leaves on the ground, see 11. For this, we randomly sample from a several fixed rgb values and slightly vary these for every leaf and tree.

3. Algorithmic pipeline

The next step in our project is to design a pipeline of filters and image transformations that can sufficiently well detect the foliage in our data sets. Again the task is a classification task and the performance metric is the number of correctly labeled pixels using binary values of whether a pixel belongs to foliage or not.

We began by trying out different texture, edge and color filters. However, in order to design an appropriate pipeline for our special task we went back to our context model and the variables that influence our scenes. We analysed these concerning how much information they give to the task. In other words we tried to figure out which of these variables our classifier should be invariant to and which give an ample amount of information to confine the feature space an image lives in.

We thus categorized the scene variables into the categories scene geometry, scene material, leaf geometry, leaf material, camera geometry and camera photometry. These categories were further subdivided and finally given the labels low, medium or high entropy, representing how much information the variable gives to the classification task. These results can be seen in figure 12.

From this table we can summarize that the background, scene material and environment variables (i.e. illumination and weather) have a high entropy. For example what the house on the property looks like is entirely independent of the foliage classification. Our classifier should also, as another example, be able to detect foliage no matter the type or intensity of illumination. Thus our classifier should be invariant to the variables of these three mentioned categories. This can be achieved by transforming input into a feature space in which highly varying parameters are invariant.

From the table we can, however, also summarize those variables with hypothesized low entropy. In comparison to those with high entropy we can try to use information from these variables to aid our classifier. As an example knowing that an object lies on the ground plane gives us quite a lot of information on whether that object might be a leaf or not. Altogether the categories and variables with low entropy can be described as sensor, leaf size, leaf color, leaf form and ground plane, where camera geometry and photometry variables have been summarized into the category sensor.

That variables of the sensor have low entropy essentially means that there is low amount of noise or variation due to camera settings etc. and that the objects portrayed over several images are portrayed as they are. This can be used as a baseline assumption that in fact the images correctly capture the scene with consistency across all images and few artifacts.

The other four variables previously listed with low entropy we can now use for building a pipeline. As a side remark these four variables are very close to the important leaf detection features described in section 2. Further noting that leaf size has a lower entropy than leaf color or leaf form we have designed a rough analysis pipeline depicted in figure 13.

3.1 Initial pipeline version

We began building our pipeline by examining ground plane detection algorithms, leaf color filters and leaf form filters in form of edge and corner detection. The results of this can be found below.

3.1.1 Ground plane detection

Our ground plane detection algorithm accepts an image pair (a destination and source image) as input. We apply local feature detection on both images via SURF. The detected features are matched with Flann based matching such that if enough good matches are found the destination image is warped to match the source image (homography). Afterwards the source image is subtracted from the warped image.
To turn the resulting grayscale image into a binary image the following filters are applied: median blur $\rightarrow$ erosion $\rightarrow$ binary threshold $\rightarrow$ dilation $\rightarrow$ canny edge detection. Median blur and erosion are applied to get rid of noise grain. To enhance the color value of certain pixels we used a binary threshold of 50. Dilation is applied in order to cancel out the effect of erosion. And finally the canny edge detection is applied to

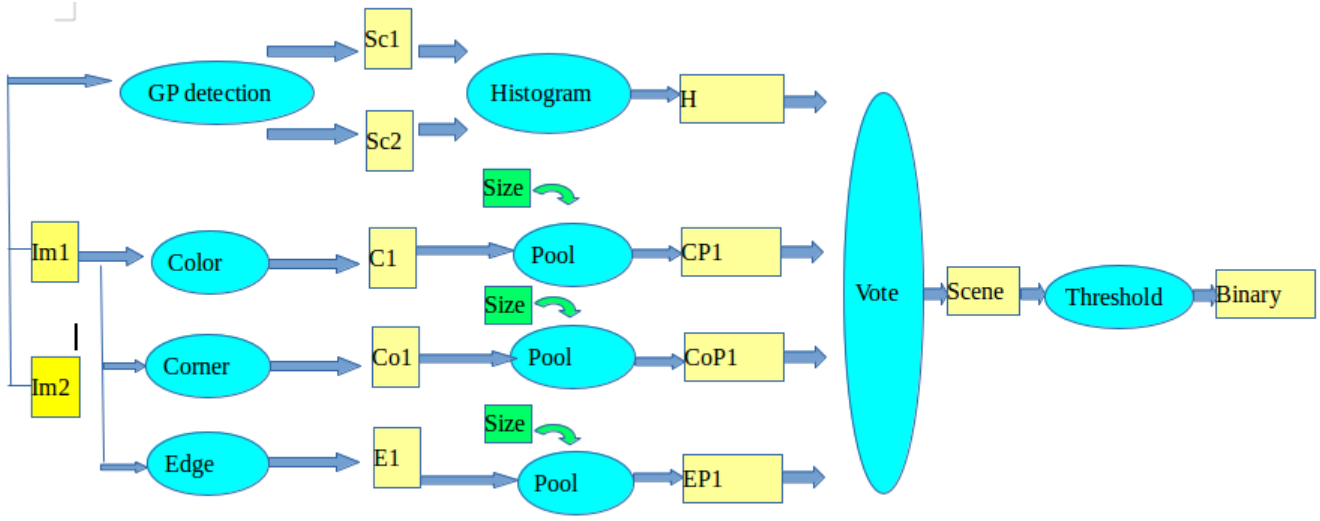


Figure 13. Rough pipeline

achieve better visualization of the result. Example figures for the filter pipeline can be seen in figure 17g.
The resulting edges are mapped onto the color source image to visualize the result. An example image for the general algorithm is in figure 17.

3.1.2 Leaf color detection

The color detection procedure is as follows: first, the rgb values of the image are normalized by transposition into the chromaticity space ($\frac{r}{r+g+b}$). Then, on the assumption that leaves on the ground are redder, than greenish or blue, the median filter is applied to the normalized r values. This filter acts as an averager: the pixel value of the most values around it is chosen. The size of the mask should correspond to the size of the leaves here. This filter is used in order to close the gap inside the leaves, with other words, to smooth the foliage area. A threshold can then be set and all pixels above it can be labelled as foliage. As one can see in figure 18, the results are reasonable, but sensitive to the magnitude of the threshold. Thus, it might be difficult to choose a threshold that will work equally good on all images.

3.1.3 Leaf form analysis

We begin by sliding a rectangular window of the approximate size of a leaf over the original grayscale image. On these so called tiles, we perform a FAST corner edge detection, a canny edge detection, a histogram of gradients and an edge orientation detection. The motivation behind using these filters was the assumption that foliage shows a large amount of different edge orientations and corners. Furthermore we assumed that foliage has a large amount of edges on a small surface area.
The results of this step can be found in figure 19 for an example tile in which the filters worked sufficiently well, however many images did not result in such clear leaf edges. However here it can also be seen that also the background material has many different edge orientations (figure 19e), if not more than a single leaf does. The corner detection in figure 19c finds more corners in the noisy background material than in the leaf structure. Canny edge filters (figure 20f) generate interesting results, however it is not clear how even in this good example the resulting feature map can be used in combination with the results of the previously expressed steps.
Finally better performance was achieved with FAST corner

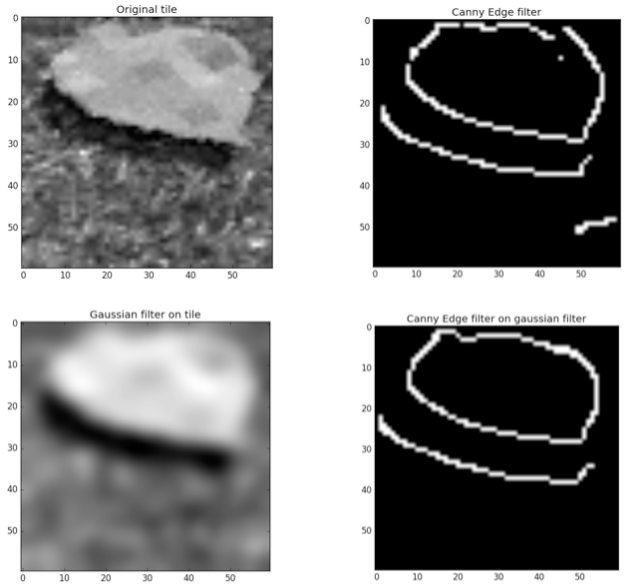


Figure 14. Final Edge filter example.

and canny edge filters, if a gaussian filter was first applied to the image. This way some of the noise in the ground texture could be smoothed and reduced. Also small noise in the leaf structure itself could be reduced such that more complete leaf contours were detected. An example can be seen in figure 14 where starting from the top left to bottom right one can see the original gray scale tile, the canny edge filter results on

this tile, the gaussian smoothed tile and finally the edge filter results on the gaussian smoothed tile.

4. Performance analysis

In order to analyse the performance of our pipeline we semantically label the test images, by setting all pixels to certain fixed RGB value corresponding to foliage objects, ground objects and the rest. This can be seen in figure 15. This ground truth image we can use to compare how many pixels in our resulting image are correctly labelled as foliage.

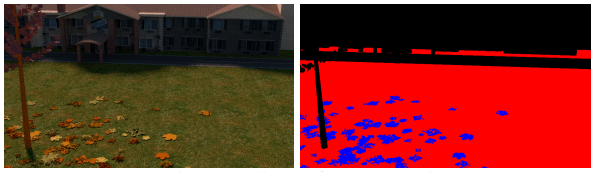


Figure 15. Semantic labelling of example image used to produce the ground truth labelling.

5. Results

From our pipeline, illustrated in figure 13, we receive four different feature maps from the modules ground plane detection, color transformation, edge and corner detection. These feature maps we then linearly combine to a single feature map which is thresholded to get a binary classification of foliage vs non-foliage. This binary image we finally compare to the binary ground truth image described above. Exemplary results of our computations can be seen in figure 20.

As can be seen in figure 20d, using all four feature maps did not achieve good results. We noticed that the ground texture (stones) has far too many edges and corners on the scale of the foliage that this could be used for discrimination. Additionally although stated above that the leaf form has low entropy for our task, this in fact is not true given that we allow strong crumbling of the ground leaves, resulting in partly very random leaf forms. Also through the crumbling effect leaves can partly be split thus also changing the size of the leaf.

However using only color transformations and ground plane detection achieved greatly improved results. On the particular image used in figure 20 and a threshold of -0.4 our pipeline correctly detected about 50% of foliage pixels. We created a dataset with five different ground textures (two grass textures and three sand/stone textures), 50 pictures a texture. Our results can be seen in table 16.

We computed four different scores to evaluate our results: accuracy, false positive and false negative rate, as well as $\frac{\text{true positives}}{\text{num of foliage pixels}}$. We also included the number of images, for which the ground detection could not be computed because of the low number of matches. As can be seen, the true positive rate is quite low, but surprisingly consistent over different textures. The false positive rate is very low, but the false negative rate is very high, which means that the color threshold is probably set too high.

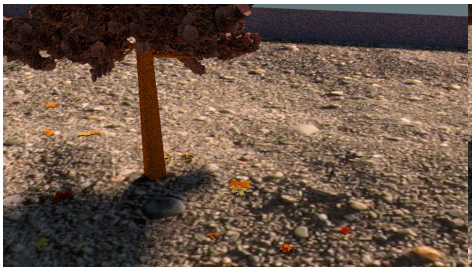
Texture	Size of sample set	Accuracy	FPR	FNR	TP / # of foliage pixels
dirt.jpg	50	0.95	0.02	0.52	0.48
gras_02.jpg	50	0.96	0.02	0.55	0.45
gras_04.jpg	49	0.95	0.02	0.55	0.45
rocks.jpg	50	0.95	0.02	0.56	0.44
sand.jpg	49	0.95	0.02	0.52	0.48

Figure 16. Final results

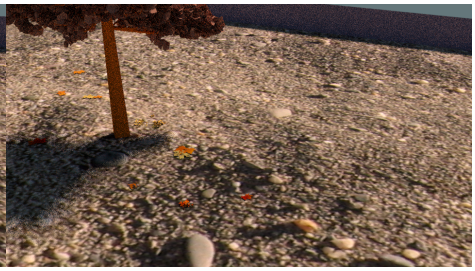
6. Discussion

The color invariant property of foliage could be handled more easily than ground or leaf edge/corner detection. Particularly, bringing the results of edge/corner detection into such a form such that they can be compared to color detection results, was more difficult than expected.

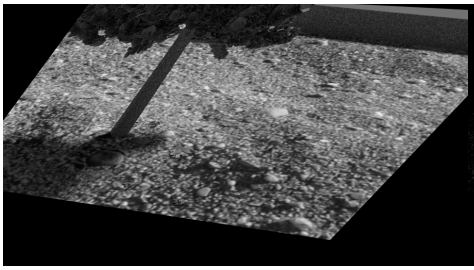
For future work and improvements it would be worthwhile to improve the ground plan detection. This does not perform as well as it could, such that still many leaves in trees are considered foliage. Also to achieve better results a future task could be to create more complex leaf form filters. Finally, as mentioned above we could try adjusting the final threshold in order to increase the true positive rate.



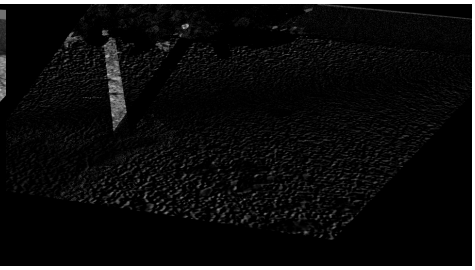
(a) Source image



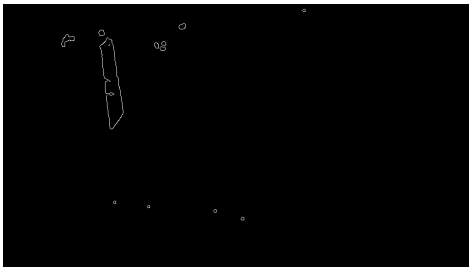
(b) Destination image



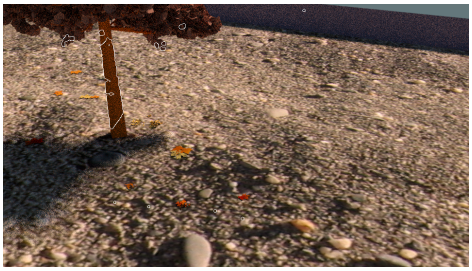
(c) Warped image



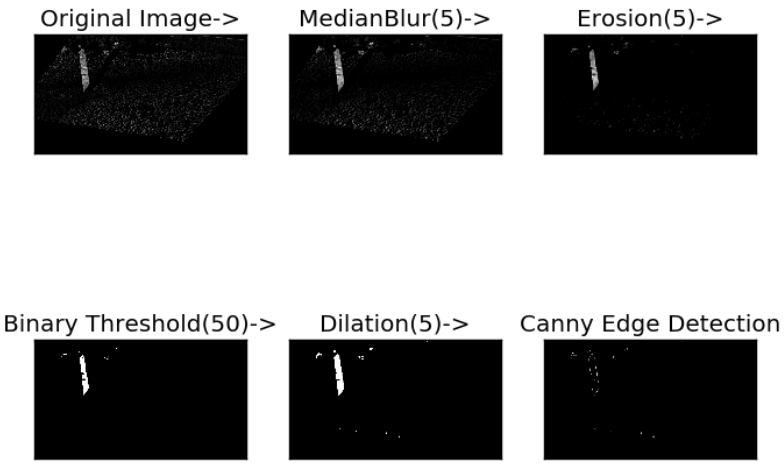
(d) Subtracted image



(e) Resulting image from filter pipeline



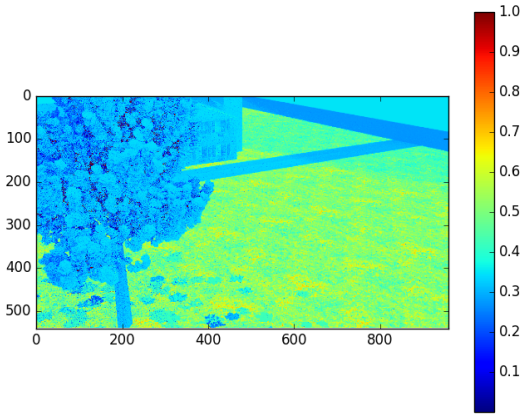
(f) Resulting image overlaid with Destination image



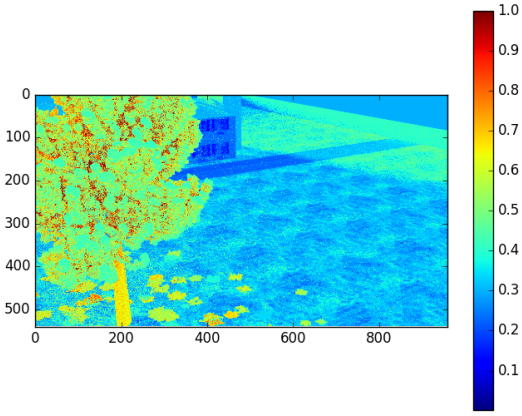
(g) Filters applied to ground plane detection results.

Figure 17. Overview of ground plane detection algorithm.

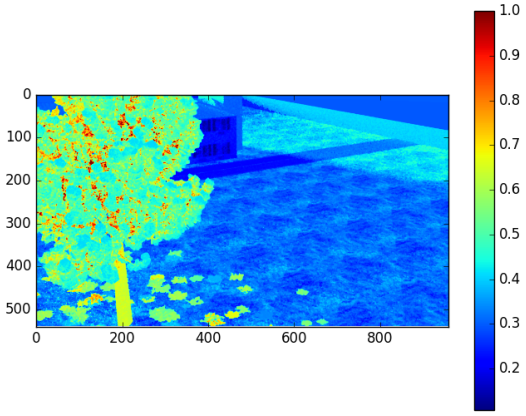
(a) Original image



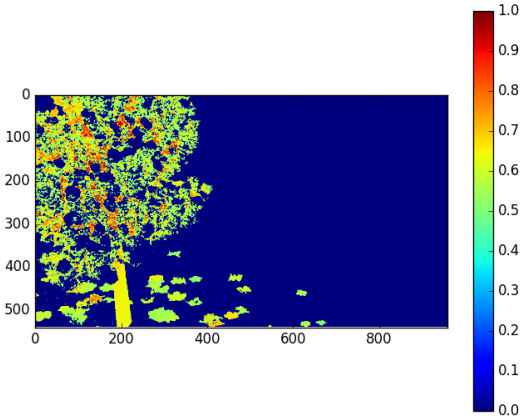
(b) Normalized g values



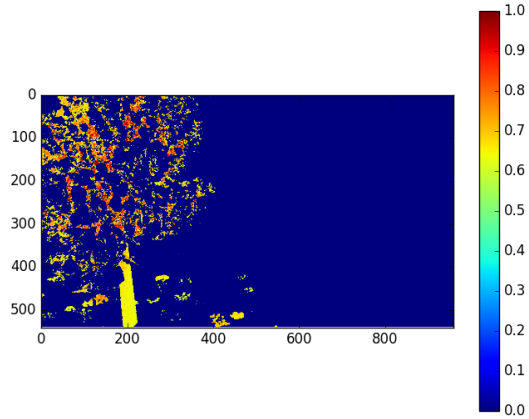
(c) Normalized r values



(d) Median filter

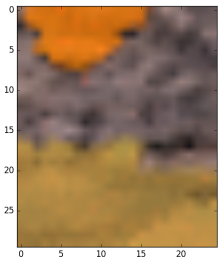


(e) Resulting image threshold 0.5

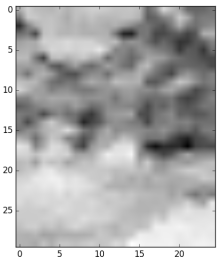


(f) Resulting image threshold 0.6

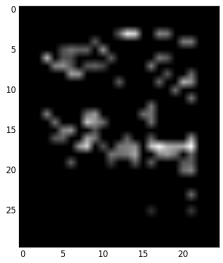
Figure 18. Overview of foliage color filtering.



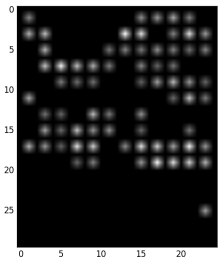
(a) Color tile



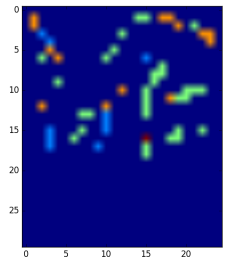
(b) Grayscale tile



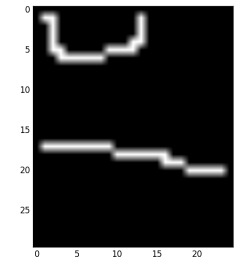
(c) FAST Corner filter tile



(d) Histogram of Gradients tile

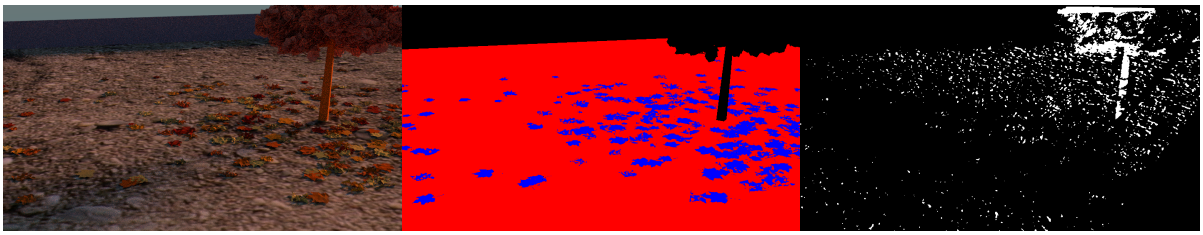


(e) Maximal edge orientation tile



(f) Canny Edge Tile

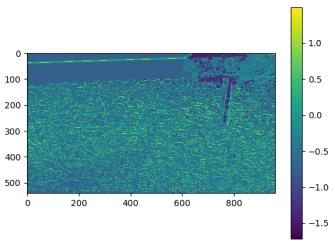
Figure 19. Overview of edge and corner filters on exemplary leaf sized tile. The color coding in subfigure 19e is: dark blue:= horizontal edge, light blue:= vertical edge, green:= 45° edge, orange:= 135° edge.



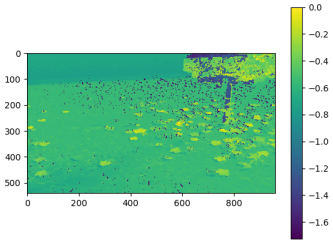
(a) Orig. Image

(b) GT

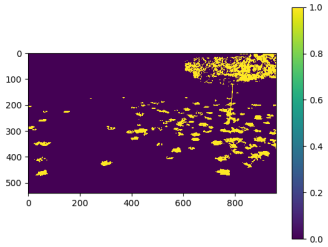
(c) Ground detection



(d) inverted GD + color + edges + corner



(e) inverted GD + color



(f) thresholded, thr=-0.4

Figure 20. Example for the combination of filters

References

[1] V S R Veeravasarapu, Rudra Narayan Hota, Constantin Rothkopf, and Ramesh Visvanathan. Model validation for vision systems via graphics simulation. *arXiv*, preprint:1–15, 2015.

[2] David Van Hamme, Peter Veelaert, Wilfried Philips, Kristof Teelen, Niels Stevens, and Bart Vermeersch. Foliage recognition based on local edge information. In J. Blanc-Talon et al., editor, *Advanced Concepts for Intelligent Vision Systems*, pages 838–849. Springer Verlag, Berlin Heidelberg, 2008.

[3] Neeraj Kumar, Peter N. Belhumeur, Arijit Biswas, David W. Jacobs, W. John Kress, Ida Lopez, and João V. B. Soares. Leafsnap: A computer vision system for automatic plant species identification. In *The 12th European Conference on Computer Vision (ECCV)*, October 2012.