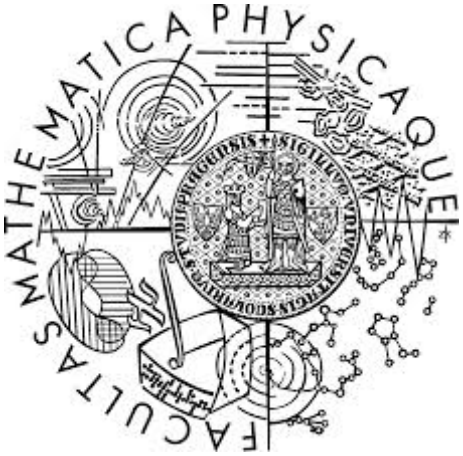
Charles University in Prague

Faculty of Mathematics and Physics

MASTER THESIS



Bc. Lukáš Tomasy

Leaf recognition for smartphones

Department of Software Engineering

Supervisor of the master thesis: doc. RNDr. Tomáš Skopal Ph.D.

Study programme: Informatics

Specialization: software systems

Prague 2014

I would like to thank to my supervisor doc. RNDr. Tomáš Skopal Ph.D. for his valuable notes during thesis software development and for borrowing his own literature, which was very helpful for my purposes. I would like to thank also to my family and especially to my girlfriend for psychical support during thesis development.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague, 5.12.2014 .....

Název práce: Rozpoznávání listů v chytrých telefonech

Autor: Bc. Lukáš Tomasy

Katedra: Katedra softwarového inženýrství

Vedoucí diplomové práce: doc. RNDr. Tomáš Skopal Ph.D.

Abstrakt: Dnešní mobilní telefony disponují dostatečně kvalitním fotoaparátem k pořizování fotografií přímo v terénu a taky mobilním připojením pro odesílání dat na analýzu na server. Cílem diplomové práce je proto vytvořit mobilní aplikaci určenou pro rozpoznávání listů stromů. Práce bude zkoumat různé algoritmy pro efektivní rozpoznávání listů stromů a jejich kombinace. Pro tento účel vznikne klient-server aplikace (jedna z podob klientské aplikace bude aplikace pro smartphone), která bude schopná s využitím zkoumaných algoritmů list rozeznat porovnáváním se vzory v databázi na serveru. Práce navíc bude zkoumat různé podpůrné algoritmy zejména z oblasti extrakce tvaru z fotografie a reprezentace deskriptorů pro možnost algoritmického porovnání tvaru se vzory v databázi.

Klíčová slova: List, rozpoznávaní, chytrý telefon

Title: Leaf recognition for smartphones

Author: Bc. Lukáš Tomasy

Department: Department of Software Engineering

Supervisor: doc. RNDr. Tomáš Skopal Ph.D.

Abstract: Today's smartphones are capable of taking pretty good quality photos and are able to connect to the internet via mobile network. Therefore, the aim of this thesis is to develop mobile application designed to leaf recognition. In the thesis we will discuss various algorithms suitable for efficient leaf recognition and their combinations. The client-server application (one of the looks of client applications would be a smartphone app) will be developed for the respective purpose. The application using discussed algorithms would be able to recognize the leaf samples by comparing them against the database patterns. We will also discuss various support algorithms mainly from the area of shape extraction from photography and from the area of leaf descriptors representation and comparison.

Keywords: Leaf, recognition, smartphone

Contents

Introduction

We search for information every day. Mostly we type the words to various search engines such as Google or Bing to get the desired result. We are looking for things by their names. But imagine the opposite. What if we want to find the words (names, descriptions, ...) linked to the thing we have. And **the thing** can be almost anything. It doesn't have to be tangible thing such as ball or shoe. It can be music, smell or even taste.

Not all things can be described properly (from previously mentioned especially the smell or the taste). In this thesis I will try to describe shapes, better said 2D shapes. And even better said, not any 2D shape, but the shapes of leaves of different tree species.

Tree species recognition according to their leaves could be helpful in real life situations such as elementary school biology class. For example, during the practical lesson in forest or park children could use the smartphone application which will be one of the outputs of thesis to identify various leaves.

Computers however cannot simply recognize the leaf just by a quick glance. Therefore, in next chapters I will propose a solution how could be images of trees recognized by computer.

1. Analysis

1.1 Human leaf recognition

How can we say "this is a maple leaf"? To determine the tree specie by its leaf we use various factors such as the shape of the leaf, its color, the size or even vein routes inside the leaf. Then we try to match these factors with something we have seen before, something similar and something we can name.

1.2 Machine leaf recognition

Computers are much more simple than human brain. The input for the computer is a picture, which is two dimensional array of pixels. First thing we can notice is that we cannot simply distinguish the size of the leaf. The distance between the lens of a camera and a photographed leaf can vary. Even a small leaf can be photographed from a small distance and in the pixel size it can be as big as a bigger leaf comparing the first one. See figure 1.1 for an example. Of course, determining the object size from the photo image isn't impossible. We can use for example focal length, but no all cameras provide such information. Next possible option is to use some sort of virtual ruler, but this approach won't be as simple as take a picture and go.

What about the color of the leaf? Is it somehow possible to determine some information from the leaf color? Actually yes, we can determine the season, but unfortunately this is unimportant to our problem. The same leaf changes its color. First in spring and during summer, it is mostly green. Later in autumn it switches its color to yellow, red or even brown. In figure 1.2 you can see the same leaf specie in different colors.

As for leaf veins, this would be probably the most difficult task to extract. They can be unreadable from the picture, in most cases only partially readable. Veins can be interesting for our purposes, but in most cases we won't be able to collect enough information to determine the tree species correctly.

The conclusion from this paragraph is, that the best way how to compare two leaves would be by their shapes.

Figure 1.1: Leaf size comparison photo. Maple leaf photo compared to Ficus benjamina. You can notice that Ficus appears even bigger, but its real size is much smaller.

Figure 1.2: Leaf color comparison.Two maple leaves in different colors.

1.3 Shape extraction proposal

In order to compare leaves we firstly need to extract the shape from the image. The first prerequisite is that the background would be of the **same bright color** (white is the best - A4/A3 paper sheet). This condition is required by many kinds of shape detection algorithms. Disunity of background may lead into wrong shape detection and to the failure of whole process of leaf recognition.

The next prerequisite to successfully extract the shape is the image itself. The

content of the image must be scan-like. It means the leaf must be sharp enough and must be the only one object in the picture. Also the picture must be large enough (picture size larger than 2 MPx should be enough).

Prerequisites had been said we can discuss shape extraction itself.

1.3.1 Background removal

For purpose to extract the shape (whole leaf in our case) properly we have to decide what is the shape (foreground) and what is the background. This is very important step and should not be underestimated.

The whole background is uninterested for our purpose, therefore, we have to get rid of it. The vital part is the foreground, which carries the information about the leaf. Possible methods of background removal will be discussed in chapter 1.4.

1.3.2 Foreground processing

Once we have extracted the foreground, we can start to process a shape extraction itself. We need a computer representation of a shape in order to continue in leaf recognition.

Unfortunately, leaf shapes can have more forms. One is a single blade on a petiole, next multiple blades (also called leaflets) on a petiole (you can see the difference in Figure 1.3). These two forms definitely aren't the only ones but are the most common to find in nature. Due to higher complexity of a second form of leaf (and all others not mentioned), this thesis will only consider single blade on a petiole as a valid leaf. However other leaf forms could be recognized as well, but it would cost much more effort. A few of possible methods of shape processing will be introduced in chapter 1.4.

The output of the image processing algorithm will be describing the shape of the leaf for the computer, so let's call it **the leaf descriptor**.

Figure 1.3: A leaf consisted of one blade on a petiole on the left and a leaf consisted of multiple blades (leaflets) on a petiole on the right.

1.4 Image processing

One very necessary step before descriptor extraction is to preprocess the image with a picture of a leaf. In this thesis I was investigating multiple algorithms and their combinations. The way I was successfully able to extract a useful descriptor is following:

- **Color normalization** - as we have discussed earlier the color of a leaf is not significant, we can convert out image into shades of one color, specifically of gray color.

- **Distinguish the background and the foreground** - if we have a picture with shades of grey color, we can find a threshold degree of shade which will makes the difference between the foreground and the background. For this purpose I implemented two algorithms regarding this problem, *Otsu segmentation algorithm* and *Huang threshold algorithm*. Both of them will be described in more details in the next chapters of this thesis.

- **Shape extraction** - Once we have a foreground available we need to extract its shape. Basically what we need to do, did every one of us in the kindergarten. We were tracing the boundaries of the object by a pencil, or at least I did that. For this purpose I implemented two algorithms, *Boundary tracing algorithm* and *Edge detection algorithm*. They will be discussed deeply later in this thesis.

- **Descriptor extraction** - At this point, we have all we need to extract the descriptor. Descriptors themselves are the main point of study of this thesis and will get more space to further investigation later in this work.

1.4.1 Color normalization

In the core a bitmap image is the matrix of pixels. And each pixel carries an information about the color. And the color consists of 4 elements. Red, green, blue and alpha channel. Since alpha channel sets the opacity of the color, we do need to adjust this value. We don't want the leaf to be transparent at all. Significant elements are reg, green and blue channel, since these are the tree basic colors and specific ratio of amount of them can form other colors. Each of these channels in bitmap fits into 8 bits, so the number in range 0 to 255. Here are some examples of common colors in RGB notation: Black = RGB(0,0,0), white = RGB(255,255,255) and red for instance is RBG(255,0,0).

Grey color is specific in one thing. It has all RGB channels set to same value. How to transform random color pixel into grayscale pixel then? It is pretty simple actually. We do an arithmetic average of RGB channels, and then we set red, green and blue channel to that value. We do that for each pixel in the matrix (in bitmap picture) and we have successfully converted color picture into grayscale one.

Figure 1.4: Grayscale vs. color leaf.

1.4.2 Huang threshold algorithm

Let X denote an image set of size MxN pixels with L levels of gray and x_{mn} is the gray level of pixel (m, n) in X. Let $\mu_X(x_{mn})$ denote the membership value which represents the degree of possessing a certain property by the (m, n) pixel in X; that is, a fuzzy subset of the image set X is a mapping μ from X to interval [0,1]. In the notation of fuzzy set, the image set can be written as

$$X = \{(x_{m,n}, \mu_X(x_{mn}))\},$$

where $0 \leq \mu_X(x_{mn}) \leq 1$, $m = 0, 1, ..., M-1$ and $n = 0, 1, ..., N-1$. Here, the membership function $\mu_X(x_{mn})$ can be viewed as characteristic function that represents the fuzziness of a (m, n) pixel in X. For the purpose of image thresholding, each pixel in the image should possess a close relationship with its belonging region (either the foreground or the background). Hence, the membership value of (m, n) pixel in X can be defined by using the relationship between the pixel and its belonging region.

Let $h(g)$ denote the number of occurrences of the gray level g in the image X. Given a certain threshold value t, the average gray level of the background μ_0 and the average gray level of the foreground μ_1 can be obtained as follows:

$$\mu_0 = \frac{\sum_{g=0}^{t} gh(g)}{\sum_{g=0}^{t} h(g)}$$

and

$$\mu_1 = \frac{\sum_{g=t+1}^{L-1} gh(g)}{\sum_{g=t+1}^{L-1} h(g)}.$$

The average gray levels, μ_0 and μ_1, can be considered as the target values for the background and the foreground (the leaf) for the given threshold value t. The relationship between pixel (m, n) in X and its belonging region should be by intuition depend on the difference of its gray level and target value of its belonging region.Thus, let the relationship posses the property that the smaller the absolute difference between gray level of a pixel and its corresponding target value is, the larger membership value the pixel has. Therefore, the membership function which evaluates the above relationship for a (m, n) pixel can be defines as:

$$\mu_X(x_{mn}) = \frac{1}{1+|x_{mn}-\mu_0|/C} \text{ if } x_{mn} \leq t$$
$$\mu_X(x_{mn}) = \frac{1}{1+|x_{mn}-\mu_1|/C} \text{ if } x_{mn} > t,$$

where C is a constant value such as that $0.5 \leq \mu_X(x_{mn}) \leq 1$. For given threshold t, any pixel any input in the threshold image should belong either to the foreground or the background. Hence, it is expected that the membership value of any pixel should be no less than 0.5. The membership function $\mu_X(x_{mn})$ really reflects the relationship of a pixel and its belonging region.

The measure of a fuzziness usually indicates the degree of a fuzziness set. It is a function,

$$f : A \to \mathtt{R},$$

which gives the fuzzy set A a value to represent the degree of fuzziness of A. Several methods of measuring the fuzziness have been proposed, in this text we are going to introduce one commonly used method called Shannon's entropy function. The entropy of set A (by DeLuca and Termini) is defined as follows:

$$E(A) = \frac{1}{nln2} \sum_i S(\mu_A(x_i))$$

with the Shannon's function

$$S(\mu_A(x_i)) = -\mu_A(x_i) ln[\mu_A(x_i)] - [1 - \mu_A(x_i)] ln[1 - \mu_A(x_i)].$$

Extending to the 2-D image plane, the entropy of an image X is expressed as

$$E(X) = \frac{1}{MNln2} \sum_m \sum_n S(\mu_X(x_{mn})), \text{ with } m = 0, 1, ..., M-1 \text{ and } n = 0, 1, ..., N-1.$$

Using the gray levels histogram, the previous equation can be further revised as

$$E(X) = \frac{1}{MNln2} \sum_g S(\mu_X(g)) h(g),\ g = 0, 1, ..., L-1.$$

$E(X)$ value is the threshold value, which should make the difference between the background and the foreground. [8][6]

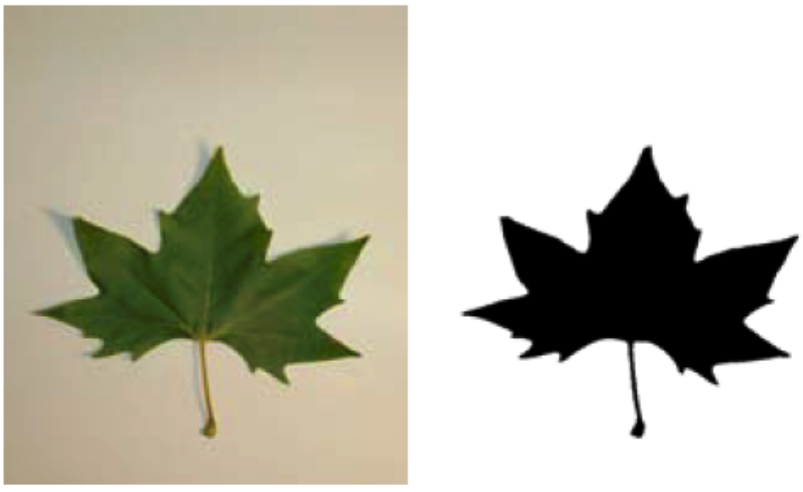


Figure 1.5: Example of Huang threshold algorithm.

1.4.3 Otsu segmentation algorithm

In Otsu's method we exhaustively search for the threshold that minimizes the intra-class variance (the variance within the class), defined as a weighted sum of variances of the two classes:

$$\sigma_\omega^2(t) = \omega_1(t)\sigma_1^2(t) + \omega_2(t)\sigma_2^2(t)$$

Weights ω_i are the probabilities of the two classes separated by a threshold t and σ_i^2 variances of these classes. Otsu shows that minimizing the intra-class variance is the same as maximizing inter-class variance:

$$\sigma_b^2 = \sigma^2 - \sigma_\omega^2(t) = \omega_1(t)\omega_2(t)[\mu_1(t) - \mu_2(t)]^2$$

which is expressed in terms of class probabilities ω_i and class meansμ_i. The class probability $\omega_1(t)$ is computed from the histogram as t:

$$\omega_1(t) = \sum_0^t pi$$

while the class mean $\mu_1(t)$ is:

$$\mu_1(t) = [\sum_0^t p(i)x(i)]/\omega_i$$

where $x(i)$ is the value at the center of the i^{th} histogram bin. Similarly, you can compute $\omega_2(t)$ and μ_2 on the right-hand side of the histogram for bins greater than t. The class probabilities and class means can be computed iteratively. This idea yields an effective algorithm: [7]

1. Compute histogram and probabilities of each intensity level
2. Set up initial $\omega_i(0)$ and $\mu_i(0)$
3. Step through all possible thresholds $t = 1...maximumintensity$
    1. Update ω_i and μ_i
    2. Compute $\sigma_b^2(t)$
4. Desired threshold corresponds to the maximum $\sigma_b^2(t)$
5. We can compute two maxims (and two corresponding thresholds). $\sigma_{b1}^2(t)$ s the greater max and $\sigma_{b2}^2(t)$ is the greater or equal maximum.
6. Desired threshold = $\frac{threshol_1 + threshold_2}{2}$

Figure 1.6: Example of Otsu segmentation algorithm.

1.4.4 Boundary tracing algorithm

One of the two implemented methods serving for shape (or boundary) extraction is simple boundary tracing algorithm. Algorithm is simple:

1. Search the image from the top left corner until the first pixel of thresholded object is found. Mark this pixel P_0. This pixel has the minimum columns value of the group of pixels having minimum row value. Pixel P_0 is a starting pixel of a region border. Define a variable dir which stores the direction of the previous move along the border from the previous border element to the current border element. Assign $dir = 7$. See figure 1.7 for direction numbering.

2. Search 3x3 neighborhood of the current pixel in an anticlockwise direction, beginning the neighborhood search in the pixel positioned in the direction $(dir + 7) mod 8$ if dir is even (figure 1.9) or $(dir + 6) mod 8$ if dir is odd (figure 1.8). The first pixel found with the same value as the current pixel is a new boundary element P_n. Update dir value.

3. If the current boundary element P_n is equal to the second border element P_1, and if the previous border element P_{n-1} is equal to P_0, stop. Otherwise repeat step (2).

4. The detected inner border is represented by pixels $P_0..P_{n-2}$. [9]

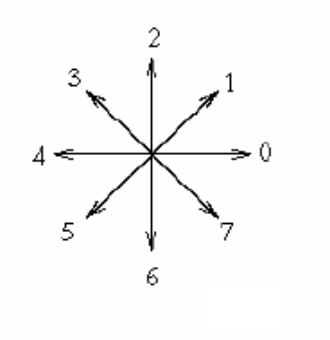


Figure 1.7: Boundary tracing direction numbering.

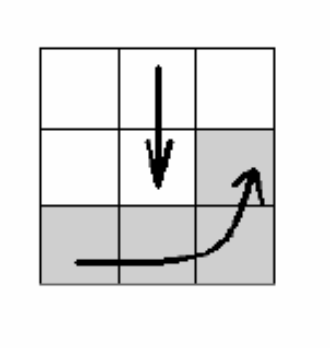


Figure 1.8: Boundary tracing explanation.

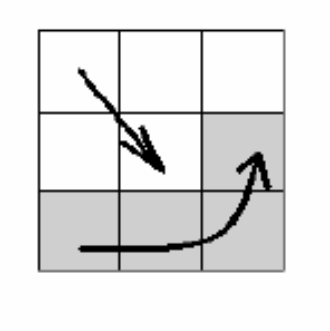


Figure 1.9: Boundary tracing explanation.

The advantage of this algorithm is his speed. For each of n boundary points you have to investigate at most 8 surrounding pixels. That give us at most $8 * n$ operations of comparison. Therefore, this algorithm can run in $O(n)$ time. The disadvantage is that it requires perfectly separated background from the picture, in another words the threshold algorithm filter as many background pixels as possible and leaves leaf pixels intact. For the scenario of not entirely precise part of background removal we can use another approach - edge detection. This method will be described in next section.

1.4.5 Edge detection

My implementation of edge detection algorithm [11] is based on the Sobel operator [10]. The Sobel operator performs a 2-D spatial gradient measurement on an image and so emphasizes regions of high spatial frequency that corresponds to edges. Typically it is used to find the approximate absolute gradient magnitude at each point in an input grayscale image. In theory at least, the operator consists of a pair of 3x3 convolution kernels as shown in Figure 1.10. One kernel is simply the other rotated by 90 °.

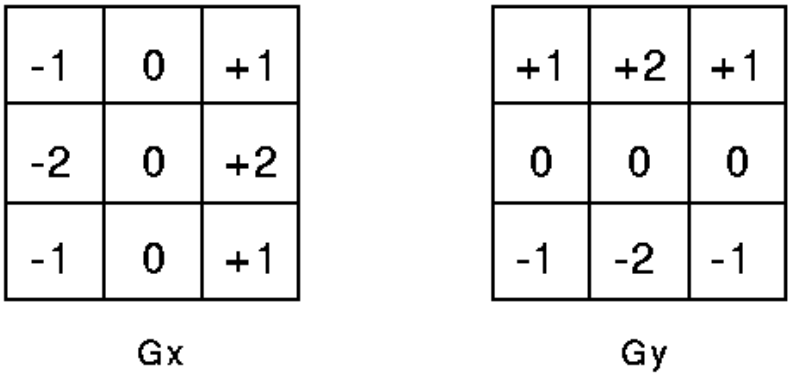


Figure 1.10: Sobel convolution kernels.

These kernels are designed to respond maximally to edges running vertically and horizontally relative to the pixel grid, one kernel for each of the two perpendicular orientations. The kernels can be applied separately to the input image, to produce separate measurements of the gradient component in each orientation (call these Gx and Gy). Subsequently, these can then be combined together to find the absolute magnitude of the gradient at each point and the orientation of that gradient. The gradient magnitude is given by: $|G| = \sqrt{G_x^2 + G_y^2}$. Now we only need to divide pixels into two groups, first groups are points which does belong to some edge, second the points which do not belong to any edge. Points with gradient magnitude lower then chosen threshold value do belongs to second group, other pixels to the first group. For leaf edge detection suited threshold value of 128 the best.

Obviously edge detection algorithm takes more time than boundary tracing, but can successfully retrieve at least partial silhouette of the leaf.

1.5 Descriptor comparison

To finish a recognition, we have to perform one last step. We have to find similar descriptors from the knowledge base to the extracted one. Knowledge base can be potentially a large set of descriptors, therefore, the comparison must be as fast as possible. To achieve this goal, we can perform comparison on L1 distance (also know as Taxicab distance). Taxicab distance for two vectors p, q is defined as $d_1(p,q) = \sum_{i=1}^{n} |p_i - q_i|$. This adds a prerequisite to the descriptors' structure. The descriptor must be a vector, or at least a structure that can be easily converted into the vector.

1.6 The result

The result of comparison against the knowledge base is a list of descriptors which matches the given one the most. What does the match in this case mean? Obviously two descriptors p, q are similar if $d_1(p,q)$ isn't too high. For that purpose we need a some sort of ceiling. Once the distance $d_1(p,q)$ crosses this value, we would be able to say, the vectors are not similar anymore. Formally $d_1(p,q) < thresh$, where *thresh* is a threshold value determining whether vectors p, q are similar or not. We will discuss possible methods to pick up the *threshold* value later in this thesis.

The resulting vectors have to be paired with appropriate tree names, as we want to return tree species names instead of descriptors. Also we have to pick only distinct tree names, otherwise the same tree name can appear more then once in the result (more descriptors can be associated with the same tree). The good idea would be also to have the list of vectors sorted by L_1 distance. This way we can express the probability of the match. Tree species higher in the result would be more probable right answer then the ones on the bottom of the result. Good practice would be also not to return more than only a first few items from the resulting answer list (3-5 tree names), if the answer contains more possible tree species that could match the leaf provided.

1.7 Existing applications

Truth to be tell, there are not many similar applications available for smartphones or as a web application. I have found only two possible applications which should do exactly the same as this project. One is called Leafsnap (`https://itunes.apple.com/us/app/leafsnap/id430649829?mt=8`) and other one Plan-

tifier (https://itunes.apple.com/us/app/plantifier/id524938919?mt=8). Both applications are available for iPhone.

However Plantifier app does not use any advanced algorithms to recognize the leaves, but uses the community behind the app. Therefore the search is not instant, you just have to wait until someone recognizes your leaf. Since humans recognize the leaves themselves, you are not limited only for leaf recognition. Unfortunately I was unable to complete the required registration for some reason, therefore no results for me from this app.

On the other hand Leafsnap should do exactly the same as my application. Workflow is very similar as well, in first step you take a picture (or upload one from the library of photos), second step should return a result. The project seems to be dead for some time, since the application is not optimized for iPhone 5 and higher and is giving no results at all (waiting for the result seems to be infinite). What surprised me was size of the team around this project - approximately 50 people from 3 universities.

2. Descriptors

For purposes of this thesis I have implemented two different descriptor types. Search for result concurrently runs in databases of both descriptor types and the results are merged into one to increase the probability of recognition.

The two types (*Centroid based descriptor and DFH*) of implemented descriptors will be described in further sections of this chapter.

2.1 Centroid based descriptor

The leaf defined in the beginning of this paper is a 2-D shape. Another observation is that the center of gravity usually lies inside that shape. Unfortunately it is not an easy task to identify the point of gravity only from a picture. What we can do, is to identify another very similar point - **centroid**.

Centroid is by definition a geometrical center of the shape. This point may as well be the point of gravity, but in most cases it will be not. We rely on a fact, that most leaves are of convex shapes. The property of the centroid is that it always lies inside of the convex object. The example of centroid of a convex shape can be found on Figure 2.1

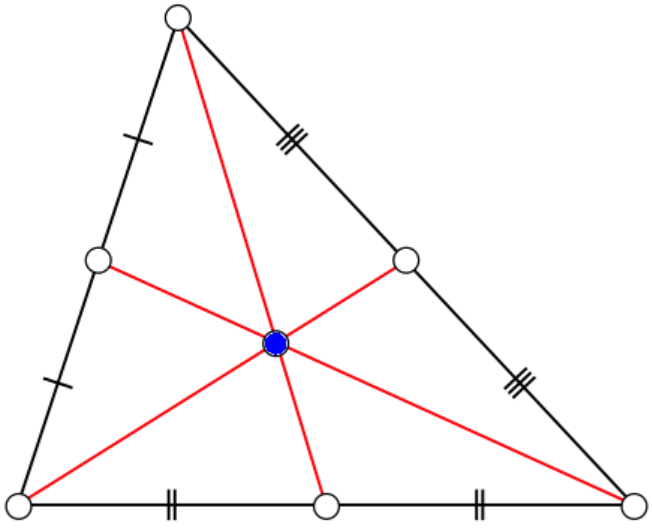


Figure 2.1: Centroid (blue point) of the triangle.

Centroid can be computed from points of the boundary of the shape as following: The cenotrid of a finite set of k points $X_1, X_2, ...X_k \in \mathbb{R}$ is $C = \frac{X_1+X_2+...+X_k}{k}$. This point minimizes the sum of squared Euclidean distances between itself and

each point in the set.

Obviously the shape can have many (hundreds or thousands) points in its contour. This could lead to significant performance loss. However, since we assume that the leaf is a convex object, we can encapsulate the leaf shape with its convex hull. Then we decrease the set of points only to the points in corners of polygon which creates the convex hull of leaf shape. Centroid of that polygon would be with high probability moved from the position of centroid of leaf shape. This won't cause us any problems, because we were not looking for the centroid of leaf shape exactly. We only need a referential point as close to the center of gravity as we can very quickly compute. That was our goal here.

Now when we have our referential point, we can lead direct lines from that point to the boundary of the shape. To achieve enough of granularity I have chosen to lead exactly 160 lines from centroid point, where each two of the lines forms the same angle. The respective approach allows us to cover the whole area of the shape. As a result we have 160 lines and each of them has certain length. In another words we have a vector in 160 dimensional space. Due to the fact that two leaves which in the reality are of different sizes can appear the same size on the picture (mentioned in chapter 1 in Figure 1.1), the vector has to be normalized to ensure that the leaf size has no effect on descriptor. On Figure 2.2 you can see lines from centroid point of leaf shape (or perhaps of the convex hull polygon) to the boundaries of the leaf. This descriptor is inspired by Eamonn Keogh's work. [12][13][14]

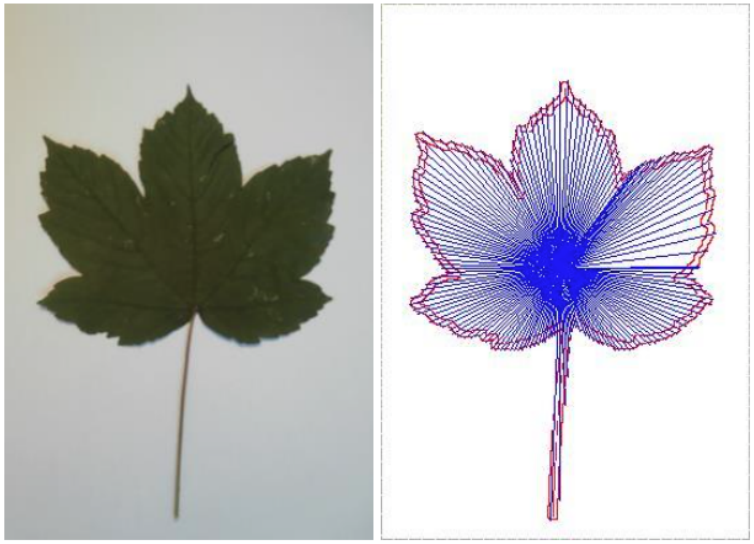


Figure 2.2: Centroid based descriptor.

Normalized vectors can be easily compared in L1 metric as described in 1.6.

2.2 Centroid based descriptor adjustments

This descriptor method relies on discrete lines from one point. That comes to an advantage when we realize that in some circumstances we can omit a few of the lines and we still can get expected results. Therefore, image preprocessing part includes **edge detection algorithm** (1.4.5) rather than **boundary tracing algorithm** (1.4.4). Edge detection can return edges with narrower or wider gaps, especially when the original picture is not taken in very good quality. When that happens, the line from centroid point will cross the boundary and stops basically at the edge of the picture itself. These lines can be easily detected and removed (or replaced with the average line, which can be computed from the rest of the lines).

Current implementation of this descriptor is non rotation invariant. Since the descriptor is a vector of values, it can be easily rotated (values shifting) to ensure the rotation invariance. However, I haven't implemented this feature due to possible high performance impact.

2.3 DFH - Directional Fragment Histogram

DFH is a boundary based approach of leaf identification. This descriptor, Directional Fragment Histogram [5], is an estimated probability density (via histogram) of a random vector $(\alpha, \mathbb{P})$ where $\alpha \in [0, 2\pi]$ is the gradient angle of a pixel and $\mathbb{P} \in [0, 1]$ is the frequency amplitude in local histogram of variable α computed from segments of the contour. This descriptor summarizes the local information provided by all local histograms and additionally integrates some general information of the contour shape.

Since we present a boundary based approach, the extraction of boundary is the first required step. To accomplish this task, we use one of the threshold algorithms to separate background and foreground of the picture. We talk about either Otsu segmentation algorithm (1.4.3) or Huang threshold algorithm (1.4.2). Once we have obtained the binary image, we must extract the whole shape. Boundary tracing algorithm (1.4.4) is the most suitable for this task, since Edge detection (1.4.5) can detect edges with gaps among them. We denote by P the contour of the leaf. It is represented by n discrete points: $P = \{(x_i, y_i) | 0 \leq i \leq n-1\}$

We consider that any contour is composed of succession of elementary components. For our purposes, the elementary component is considered to be a pixel. To each pixel of the contour we associate a directional information by means of two variables, θ the angle of the gradient computed from the original image, and θ', the numerical derivative of the variable θ, computed on the contour. The variable θ is absolute. It corresponds to the gradient angle on the original image. These angles are projected into a quantified $[0, 2\pi]$ interval. The outline will then be represented by a string composed of directions resulting from this quantified space. The second variable θ' is relative. It gives information about slopes of the gradient functions. In the case of leaf shapes, this corresponds mainly to extrema of leaf margin. All segments of size s are constructed considering that each segment is composed of s successive pixels from the contour. For example, given a contour P composed of n pixels, i.e., $P = (c_1, c_2, ..., c_n)$. The first two segments of size $s = 4$ are $S_1 = (c_1, c_2, c_3, c_4)$ and $S_1 = (c_2, c_3, c_4, c_5)$.

As the contour is closed, the total number of all possible segments is equal to n, the size of P. The size s of the segment will be chosen to be proportional to the length of the global outline of the contour considered in order to ensure the scale invariance. The proportion of the segment size is denoted $p_s = s/n$. For each segment $S_i, i = 0, ..., n-1$ and for each variable (θ or θ'), we identify groups of pixels in the segment S_i that have the same qualified value of θ or θ'. Such groups are called directional fragments. The variables θ and θ' are from interval $[0, 2\pi]$ (360°). The interval $[0, 2\pi$ is partitioned into M bins with the same length. For each variable, θ and θ', we compute its local (relative to the segment S_i) histogram. Hence we obtain n local histogram for variable θ and n histograms for variable θ'.

We will now summarize this local information (provided by the above $2n$ histograms) by means of a (somehow) double histogram, which also integrates moreover some global information of the leaf. This is done as follows. We consider that the quantified values of θ or θ' (on M values) are realizations of a random variable of direction, denoted α, taking its value in the interval $[0, 2\pi]$. We also consider that the amplitude of frequencies in the $2n$ histogram are realizations of a random variable, denoted $\mathbb{P}$, taking its values in the interval $(0, 1]$, which we quantify into J uniform bins. The value zero is excluded. We then compute 2-D histogram of the random vector $(\alpha, \mathbb{P})$ from the above $2n$ local directional histograms, using M uniform bins of $[0, 2\pi]$ for the random variable α and J uniform bins for the random variable $\mathbb{P}$. This 2-D double histogram will be the descriptor called DFH descriptor. At scale s it is a two-dimensional array of

values that contains MxJ bins of $[0, 2\pi]x(0, 2\pi]$. For illustration see Figure 2.3, which describes the extraction procedure of the fragment histogram considering $M = 8$ different directions (for α) and $J = 4$ different fraction ranges (for $\mathbb{P}$).

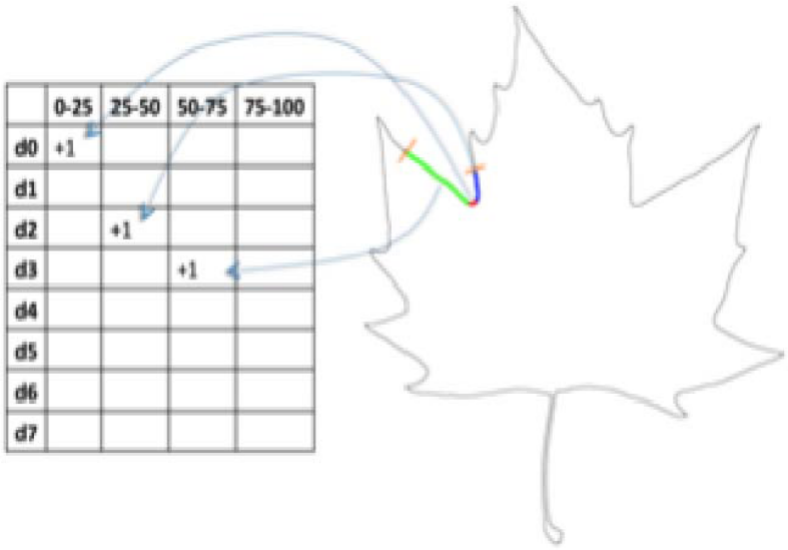


Figure 2.3: Extraction of the directional fragment histogram.

Note that this descriptor depends on three parameters M, J and p_s. In order to determine the optimal values for these parameters, a large dataset of leaves is required.

To achieve a realtime search we us simple L1 distance to compare the DFH descriptors of the leaves. I expect that the DFH will be able to distinguish between leaves, if the parameters M, J and p_s are well chosen. In fact, consider two leaves l_1 and l_2. Denote $(\alpha_1, \mathbb{P}_1)$ and $(\alpha_2, \mathbb{P}_2)$ the associated random variables, and f_1 and f_2 theirs probability densities. Then the L1 distance between f_1 and f_2, denoted $d(f_1, f_2)$, is non-negative, and it is close to zero if the leaves l_1 and l_2 come from the same species.

2.4 DFH adjustments

The DFH descriptor as described in previous section is not rotation invariant. However, the solution to make this descriptor rotation invariant is not that difficult. On Figure 2.3 you can notice that the descriptor is a 2-D array. The first dimension is a distribution into J directional fragment groups (in the example into 4 groups). The second dimension is a direction distribution (all possible directions of directional fragments). If we rotate the image, the fragments lengths stay the same. However, their directions change. Therefore, we can simulate the

rotated image by rotating the rows (according to the example on Figure 2.3) of the histogram 2-D array. This feature is not implemented due to possible high performance impact on the system.

2.5 Region based descriptor

Region based descriptor is described in this thesis primarily to show also other possibilities of shape description. It will not be implemented in the attached application, because other descriptors described in this thesis (**??**) are better suited for shapes similar to leaves description.

This descriptor takes into account all pixels constituting the shape, that is both the boundary and interior pixels (background). It is therefore applicable to objects consisting of a single connected region or multiple regions, possibly with holes. The descriptor works by 'decomposing' the shape into a number of orthogonal 2-D basis function (complex valued), defines by Angular Radial Transformation (ART). The normalized and quantized magnitudes of coefficients are used to describe the shape. [1]

2.5.1 ART Transform

ART is the orthogonal unitary transform defined on a unit disk that consists of the complete orthogonal sinusoidal basis functions in polar coordinates. The ART coefficients are defined by:

$$F_{nm} = \langle V_{nm}(\rho,\theta), f(\rho,\theta)\rangle = \int_0^{2\pi}\int_0^1 V^*_{nm}(\rho,\theta) f(\rho,\theta)\rho d\rho d\theta$$

where F_{nm} is an ART coefficient of order n and m, $f(\rho,\theta)$ is and image function in polar coordinates and $V_{nm}(\rho,\theta)$ is the ART basis function that are separable along the angular and radial directions, that is,

$$V_{nm}(\rho,\theta) = A_m(\theta)R_n(\rho)$$

In order to achieve rotation invariance, an exponential function is used for the angular basis function,

$$A_m(\theta) = \tfrac{1}{2\pi} exp(jm\theta)$$

The radial basis function is defined by cosine function,

$$R_n(\rho) = 1, n = 0$$
$$R_n(\rho) = 2cos(\pi n\rho), n \neq 0$$

Real parts and imaginary parts of ART basis are shown in Figures 2.4, 2.5. [1]

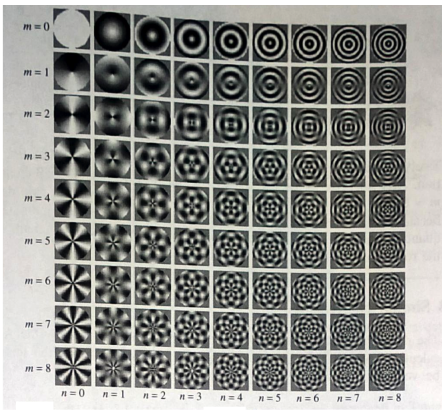


Figure 2.4: Real parts of ART basis functions.

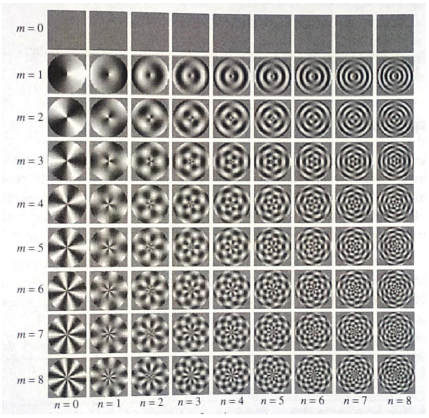


Figure 2.5: Imaginary parts of ART basis functions.

2.5.2 Descriptor representation

The ART descriptor is defined as a set of normalized magnitudes of complex ART coefficients. Rotational incariance is obtained by using the magnitude of the coefficients. Twelve angular and three radial functions are used ($n < 3, m < 12$). For scale normalization, ART coefficients are divided by the magnitude of ART coefficient of order $n = 0$, $m = 0$. The ART coefficient of order of $n = 0, m = 0$ is not used as a descriptor element, because it is constant after scale normalization. [1]

2.5.3 Similarity measure

The distance between two shapes described by ART descriptor can be calculated as L1 distance as follows:

$$D(a, b) = \sum_i |M_a[i] - M_b[i]|$$

Here a represents the image in the database and b query image and M is the array of ART descriptor values. [1]

3. Architecture

3.1 Who is going to use this software

The purpose of this application is to recognize leaves of various tree species. It would make sense to use this software right in the nature, around the trees. Fortunately, current age provides us with powerful enough hand-held devices with internet connectivity, the devices we call smartphones nowadays. I believe that software of this kind can be used among the children during the unconventional biology class right in the forest or in city park.

People with their devices can also provide data. In another words, system can learn from users. The idea is, that people can provide more descriptors which will be used to improve positive search rate.

3.2 Client-server architecture

The best way how to ensure the latest data are available is to provide it online. That means shared knowledge base for all users (clients) on one place (server). The server does not have to be necessarily on one physical place, but for the clients it can appear that way.

All business logic would be encapsulated in server. This approach would allow easily create thin client application for various (not only smartphone) platforms. Of course, sufficient API must be available. Part of this thesis will include client application for currently the most popular smartphone platform - Android. Web application will be provided as a part of this thesis as well.

3.3 Data storage

Ideal database would be the one with possibility of indexing of descriptors. From the previous chapter 1.5 we know descriptor would be some sort of vector. Another great feature would be programability of database to be able to compare descriptors directly on database machine. This feature could potentially save a lot IO operations (transfer a lot of data from DB to program).

3.4 API

As we discussed in 3.2, sufficient API (Application Programming Interface) is required in order to easily develop various thin clients. API should be providing all necessary functionality including leaf recognition, database extension through user input data and probably also user account and infrastructure functionality.

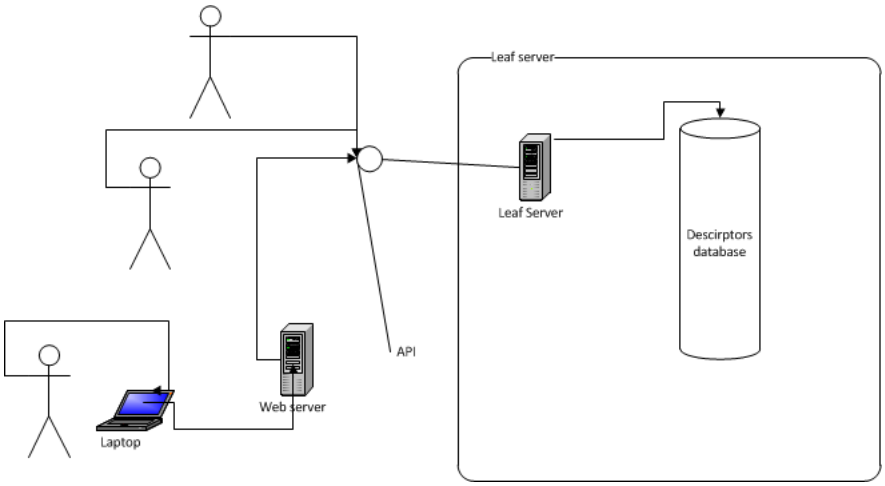


Figure 3.1: Possible architecture of application.

4. Implementation

4.1 Platform

The development environment I feel the strongest is Microsoft .NET 4.0 platform [2]. It provides the whole stack of technologies to develop application described in chapter 3. Application server and other server related features will be written in C# .NET language. This is the language I have the most experiences with and I find it suitable for building even enterprise application, which will be more than enough for our needs. The application will be installed on Microsoft Windows Server 2008R2, which provides necessary environment to run .NET and .NET applications. As I own one server with this system, it will be used to host the application itself.

The most suitable data storage would probably be database server directly from Microsoft as well. I have chosen SQL Server 2008 in Express edition which is for free even for commercial use. SQL Server 2008 allows to develop advanced functionality over relation-based operation in T-SQL language or even in .NET itself. This comes very handy, because we can encapsulate storage of arrays and operations over arrays (mainly computation of L1 distance) directly to the database engine. This is the feature we were looking for in 3.3. Unfortunately, indexing of CLR User Defined Types is not supported in SQL Server 2008. [3]

Express version of SQL Server 2008 has one limitation with great impact. It only uses 1 processor core even if the system provides more of them. That is the penalty for free version of SQL Server. This can cause performance issues while executing concurrent leaf recognition algorithms. Therefore, I decided to run Raven DB document database [4] side by side with MS SQL Server 2008. It can be also interesting to compare performance of those two database engines, but that is not the aim of this thesis.

API would be provided as web service created in WCF (Windows Communication Foundation) framework which is a part of .NET 4.0. These web services can be easily hosted on Microsoft IIS (Internet Information Services) web server. ISS web server is available for free in every edition of Windows Server 2008R2. WCF web services can be easily consumed by any other application, since the communication happens over HTTP protocol and request and response messages are SOAP XML messages. Also WSDL document which describes the service

methods is always available. There are also many tools on the internet to generate WCF client wrappers for various languages (i.e. Metro package for Java language, SVCutil for C# or VB.NET, etc.).

For purposes of this thesis I will develop also Android application as a one possible client using API provided by server. Android application will require at least Android in version 2.3.3 due to references packages for consuming WCF services. I believe, this is reasonable requirement, since current version of Android system on the market is 4.4.2 (first half of year 2014) and more recent versions will follow.

For test and administration purposes as well as for real usage of application from PC web application will be introduced too. This application will be communicating with server also via the same API as mobile clients. Web based application will contain some admin based features which will not be publicly available to other mobile clients. I aim mainly to some sort of approval process of learning from users. The descriptor provided by user can be defective, and therefore, it can damage our knowledge base or the provided descriptor can be intentionally assigned to another tree species which will result into damaging the knowledge base as well.

4.2 Database structure

The relational database can be divided into two parts. The first one handles user data and user access and rights. The second part holds data about tree species and descriptors itself. I have also introduced CLR type *Descriptor* which encapsulates the vector of floating point numbers. This vector is an internal representation of *Centroid based descriptor* which will be described later in this thesis. With *Descriptor* type I also implemented method which can compare two objects of type *Descriptor* and as a result is a distance of underlying vectors in L1 metric. I have also used concurrent RavenDB database which is not relation based, therefore, data tables inherit the structure from the objects stored inside. This database is used to store *DFH descriptors* data. Again, this descriptor will be described later.

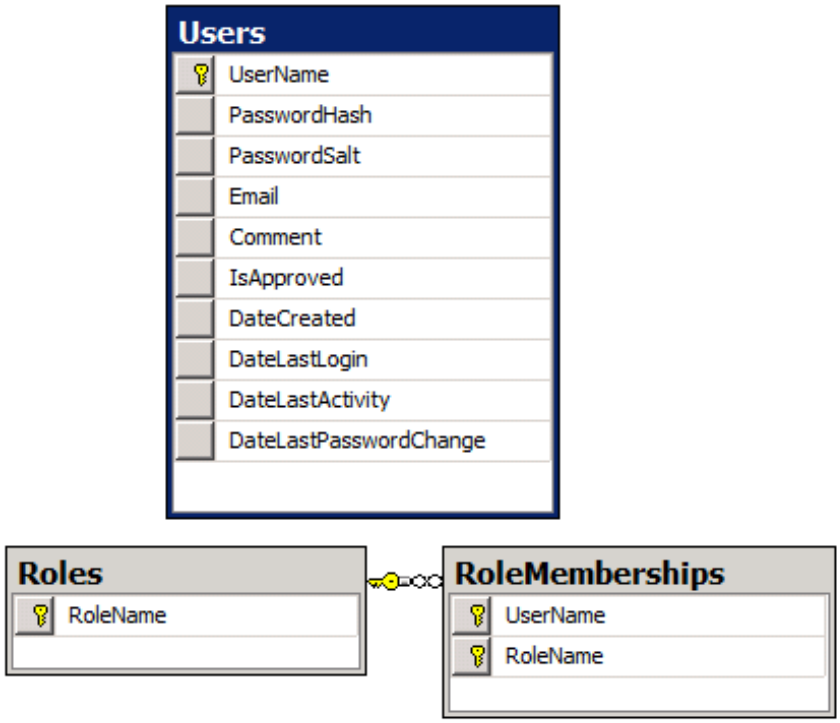


Figure 4.1: User data tables.

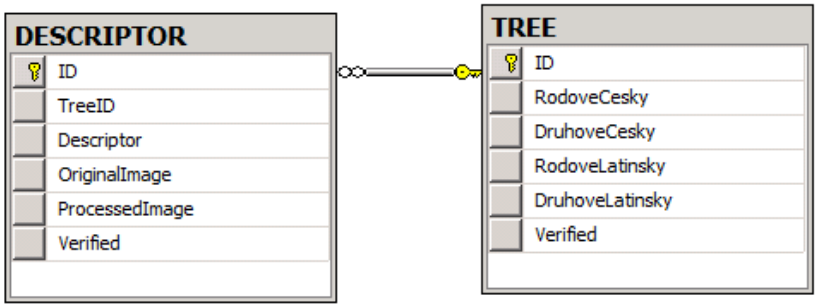


Figure 4.2: Descriptor data tables.

4.3 Public API

The API for mobile client is as easy as possible. It contains only 4 methods and their signature is following:

- **Tree[] Recognize(string picture, int noAnswers)**
- **Tree[] GetTrees()**

- **int AddTree(string czRodove, string czDruhove, string ltRodove, string ltDruhove)**
- **string Learn(int treeID, string picture)**

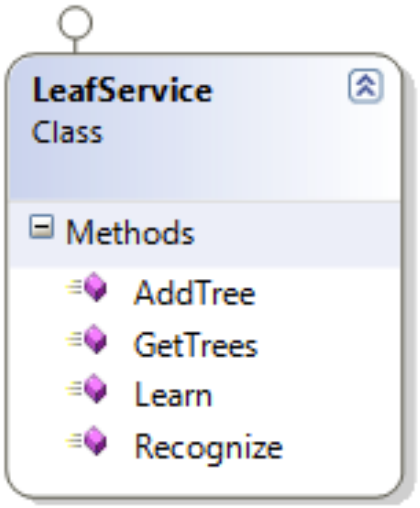


Figure 4.3: Leaf API diagram.

4.3.1 Tree object

Tree object is very simple data transfer object carrying significant information about resulting tree species. Here is the list of properties encapsulated in Tree class:

- **int ID** - unique identifier of tree species inside the system.
- **RodoveCzech** - Czech genus
- **DruhoveCzech** - Czech species
- **RodoveLatin** - Latin genus
- **DruhoveLatin** - Latin species
- **Confidence** - number from interval [0;1] which indicates how confident was the system with the result tree.

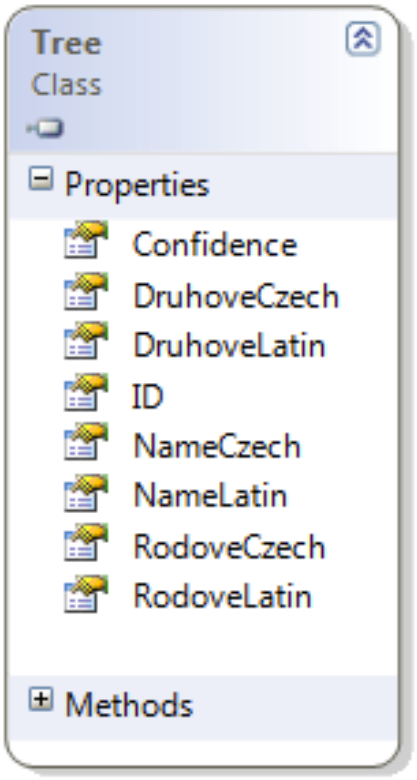


Figure 4.4: Tree class diagram.

4.3.2 Method Recognize

Method Recognize tries to recognize the leaf provided on the photo. The photo is passed in string parameter **picture** encoded as Base64. Encoding an byte array which the originally is much more universal in cross platform communication. The second parameter specifies the number of max results in case there are more results available then we want to receive. If the parameter value is zero or is negative, default value applies (max 3 results per request).
The result of this method is an array of Tree objects with the most tree species names with the match to the provided tree (if any).

4.3.3 Method GetTrees

This method simply returns all tree species names in the database. This method is particularly important when we need to show a list of all possible tree names to the user.

4.3.4 Method AddTree

Via this method the tree species database can be enlarged. Method has 4 mandatory parameters, two describe genus and species in Czech and next two the same in Latin. As a successful result of this operation a new tree species information will be added to the system.

4.3.5 Method Learn

Method Learn has two parameters. The first one is an unique identifier of a tree species, the second is a picture of a leaf from that tree. Picture is encoded in Base64 encoding for the reasons described in 4.3.2. Obviously, the user can select wrong tree species for given leaf picture. There may be more reasons why, but as a precaution another interface will provide the approval process of new descriptors. You can read more about that interface in 4.4

The result of this operation can be nothing (empty string) or Base64 encoded images of visualized descriptors. They can be handy to check whether descriptors were extracted properly or not. Especially when no results are found in the system.

4.4 Descriptor verification API

The API for verification of provided descriptors contains only 3 methods. Their signature is following:

- **UnverifiedDescriptor[] GetUnverified(DescriptorType descriptorType)**
- **bool VerifiyDescriptor(string descriptorId, DescriptorType descriptorType)**
- **bool DenyDescriptor(string descriptorId, DescriptorType descriptorType)**

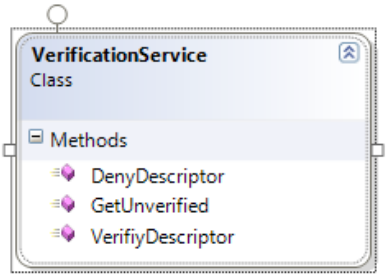


Figure 4.5: Verification API diagram.

4.4.1 DescriptorType enum

DescriptorType is an enumeration which contains supported descriptor types by the system.

4.4.2 UverifiedDescriptor class

A person responsible for verifying the descriptors needs certain information about the descriptor such as descriptor unique identifier through the system, Czech and Latin genus and species inserted by another user (who was teaching the system new descriptor), type of the descriptor and of course source image and processed image (the last step of descriptor extraction that can be easily visualized). All respective information is encapsulated inside UnverifiedDescriptor class.

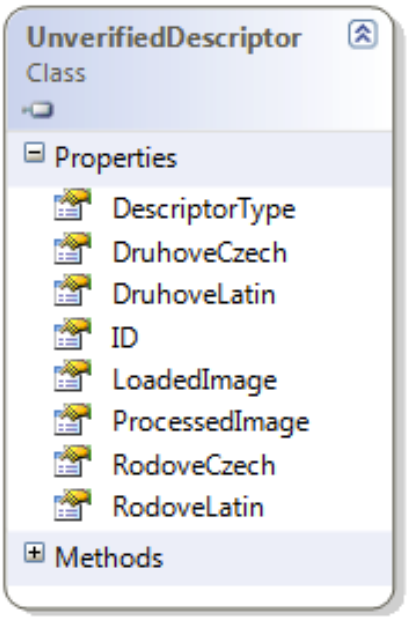


Figure 4.6: UnverifiedDescriptor class diagram.

4.4.3 GetUnverified method

The method GetUnverified gets only one parameter of type DescriptorType which signals the method to return unverified descriptors of specified type. The result of this method is an array of 10 items at most. This provides kind of buffering consisting of sending always up to 10 oldest unverified descriptors of given type in the database.

4.4.4 VerifyDescriptor method

The parameters of this method are unique identifier of the descriptor in the system and its type. These two parameters are all the system needs to verify the descriptor as such. The result of this operation is true of false in dependency of operation success.

4.4.5 DenyDescriptor method

DenyDescriptor method gets the same parameters as VerifyDescriptor method. Instead of descriptor approval, in current implementation the descriptor, which we want to deny is deleted from database. This method is useful, when descriptor was not generated properly due to e.g. low quality photo of a leaf.

5. Results

5.1 Descriptors comparison

The following tables show the experimental results for both descriptors, DFH and Centroid based lines made by author if this thesis. The leaf photos were taken on daylight on white paper sheet as a background with flash turned off. The insufficient knowledge base can have negative impact on final results as well as on future use of this application.

DFH:

Tree species	**Good hits**	**False hits**	**Total**	**Success rate**
Betula pendula	7	3	10	70 %
Fagus silvatica	9	2	11	81.8 %
Quercus robus	9	3	12	75 %
Quercus rubra	7	2	9	77.8 %
Ficus benjamina	6	1	7	85.7 %
Acer platanoides	14	3	17	82.4 %
Aesculus hippocastanum	6	4	10	60 %
Tilia cordata	9	1	10	90 %
Corylus avellana	6	3	9	66.7 %
Salix alba	10	2	12	83.3 %
Sum	83	24	105	**79 %**

Centroid based lines:

Tree species	**Good hits**	**False hits**	**Total**	**Success rate**
Betula pendula	8	2	10	80 %
Fagus silvatica	6	4	11	54.5 %
Quercus robus	8	4	12	66.7 %
Quercus rubra	8	1	9	88.9 %
Ficus benjamina	7	0	7	100 %
Acer platanoides	14	3	17	82.4 %
Aesculus hippocastanum	7	3	10	70 %
Tilia cordata	7	3	10	70 %
Corylus avellana	6	3	9	66.7 %
Salix alba	8	4	12	66.7 %
Sum	78	27	105	**74.3 %**

Based on the results, we can distinguish an exact winner. The descriptors achieve more less the same results, in average around 75-80% of success rate. But the advantage of combination of these two algorithms is much more accurate result. If one algorithm fails to identify the leaf, second one may succeed and the merged result may present right answer after all.

5.2 Real environment test

Experimental results show the potential of methods used to identify leaves. However the experience from real life usage may be very different. Therefore the experiment from real life environment shouldn't be missing in this thesis.

In order to start the experiment I sent instructions how to obtain photos of various leaves to approximately a two dozens of my friends. They could choose to use either mobile client or web based client. Each mobile client and each unique session in web client was given random GUID (global unique identification number) which made distinguishing between devices possible.

My first discovery was that sadly only less than a half of friends really wanted to participate on this experiment. Even less of them really participated.

5.2.1 Gathered data analysis

When a significant set of data is gathered, we can start the analysis. The following table shows the result of the experiment:

Tree species	**Good hits**	**False hits**	**Total**	**Success rate**
Betula pendula	10	1	11	90.9 %
Fagus silvatica	8	1	9	88.9 %
Quercus robus	11	3	14	78.6 %
Acer platanoides	19	9	28	67.9 %
Tilia cordata	10	2	12	83.3 %
Corylus avellana	1	2	3	33.3 %
Salix alba	0	1	1	0 %
Sum	59	19	78	**75.6 %**

The table shows us that leaves of only 7 tree species have been looked up during this experiment by all the participants. The leaf Salix alba has been submitted only once with one false result. In other hand the most desired leaf to submit was a maple leaf (Acer platanoides). This table contains only those leaves where the

descriptor was properly extracted, then the descriptor evaluation could produce the valid result. Success rate is very similar to our experimental results, where the emphasis was laid to obtain as best pictures as possible.

5.2.2 Invalid data problem

Unfortunately a lot of gathered data(more than 50%) returned false hits or no results at all. During the data investigation I have discovered various problems.

Photos produced in real life environment were very often of bad quality of did not meet the requirements of the application (white background, specific leaf orientation, ...) and therefore the recognition failed practically at the first phase - leaf shape extraction. In the following figures we can see the most common violations of prerequisites by users:

Figure 5.1: Bad orientation of the leaf.

Figure 5.2: Bad orientation of the leaf.

Figure 5.3: Not suitable background for the leaf.

Another common picture issue was related to the edge of the background. The quality could be pretty decent, event the leaf orientation could be right, but if there is part of another object (usually at the edge of the photography, where the white paper as the background wasn't enough to cover the whole frame of the photo) the recognition will fail, because the shape of the leaf wouldn't be recognized properly. See figure 5.4.

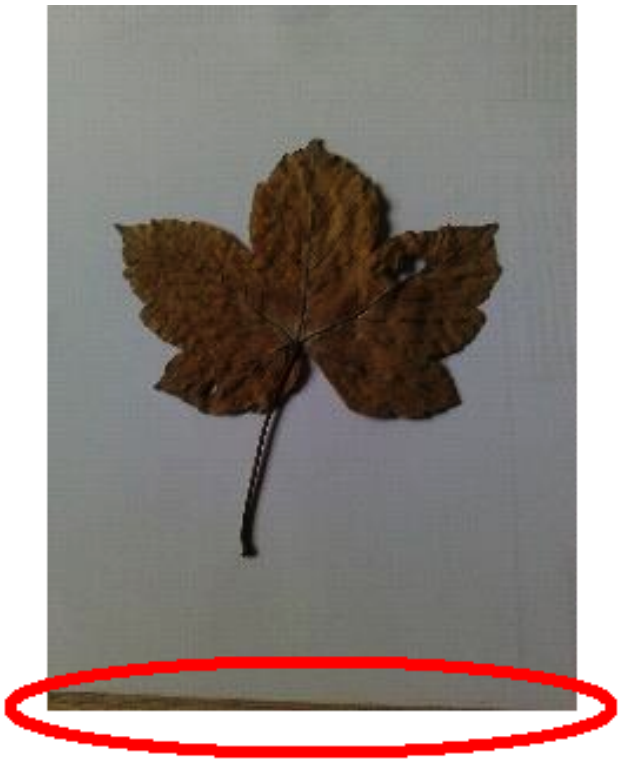


Figure 5.4: White background does not cover whole frame.

Sometimes the quality of the photography is the problem. We can see picture artifacts in some photos which are ultimately recognized as objects and therefore the recognition will most likely fail. One of the artifact investigated in the data could be seen in the figure 5.5

Figure 5.5: Unwanted artifacts in the photo.

Users also made some complains about the gyroscope in Android app camera. When they were pointing on the leaf the phone was almost in horizontal position. Gyroscope therefore somehow decided that the orientation of the phone changed and the resulting picture is in landscape format instead of intended portrait orientation. Current version Android application uses build-in support to access the camera which does not allow to control gyroscope directly. Another approach needs to be developed. Figure 5.6 shows how the leaf photo is being rotated when Android phone decides to change the phone orientation.

Figure 5.6: Photo rotated by wrong gyroscope action.

One participant of the test even took photo of his/her rabbit instead of a leaf. Rules violation of course, but there was something interesting in this attempt. It has returned some results of possible tree species of the "leaf". Figure 5.7 show the photo of the rabbit and the centroid lines descriptor which was generated. It seems very strange and we would say it shouldn't return any valid result. But centroid line descriptor is normalized to achieve size invariability. Most of the lines are about the same length, very similarly as in leaves of circular shapes. The top result said that leaf comes from Fagus sylvatica (European beech) and the leaf of beech is a bit circular.

Of course, this is an issue which needs to be resolved. False positive hit as bad as wrong hit or no result at all.

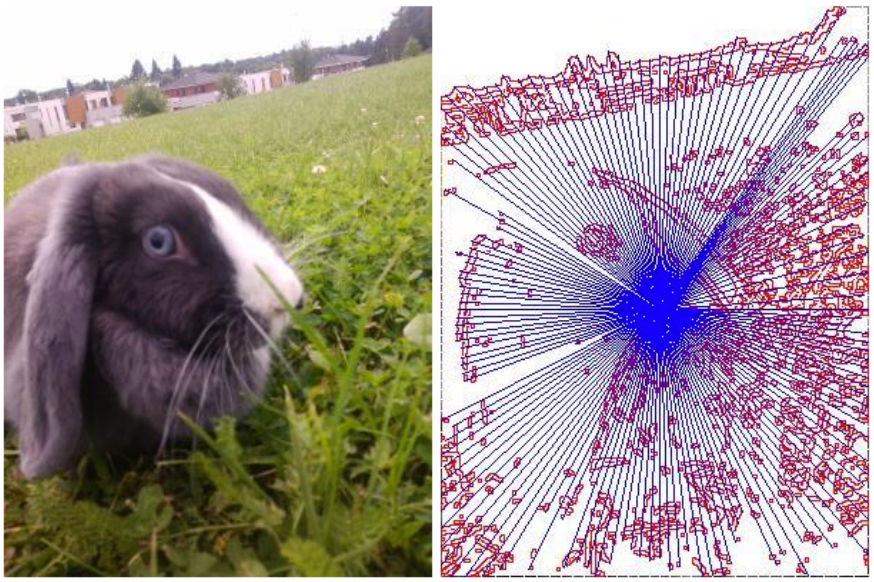


Figure 5.7: Photo of the rabbit and appropriate descriptor.

5.3 Search log

Every attempt for a leaf recognition is saved into the database to keep track of all records. Search log is very simple and consists only from two database tables $SEARCH_LOG$ and $SEARCHED_IMAGE$, with simple one-to-many relationship between them. Table structure is shown in Figure 5.8. To extract and process these data I developed a simple tool, log reader which shows all stored information for a search result in human readable form. Example of a one entry of search log can be seen in Figure 5.9

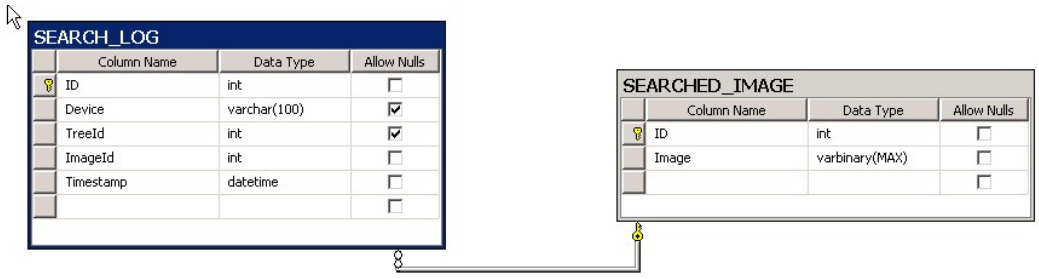


Figure 5.8: Database structure used for storing information about leaf recognition attempts.

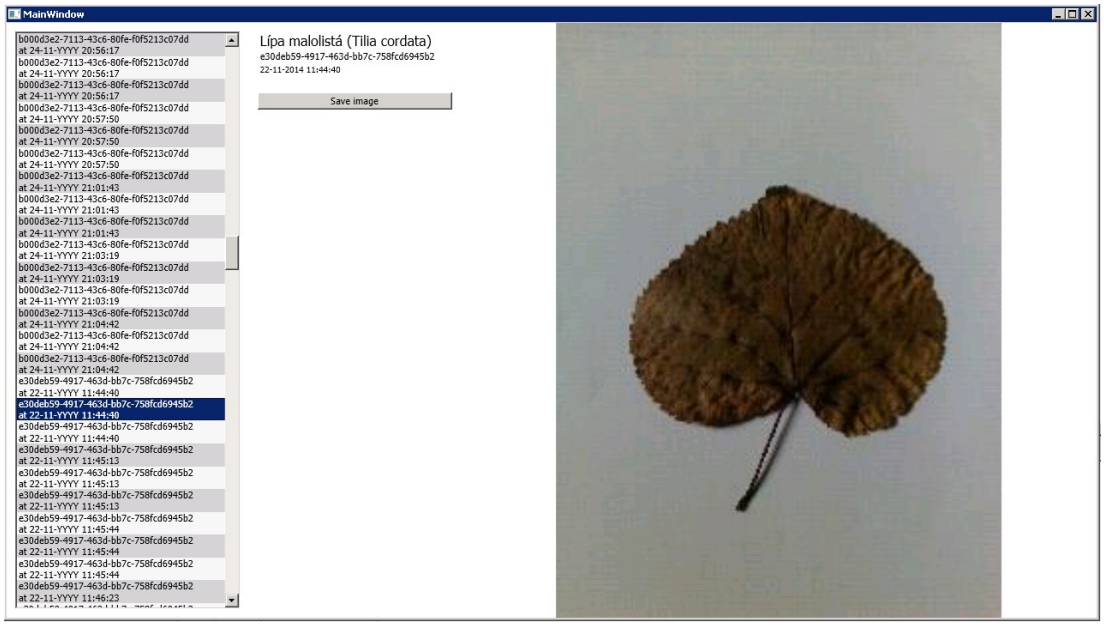


Figure 5.9: Search log reader example.

6. User guide

6.1 Android application

First thing to say, Android application is not yet available on Google Play store. It is only distributed as an attachment to this thesis in form of APK package. The application is localized only in Czech language in current version. In future versions, other language mutations will come.

6.1.1 Installation

The installation of the application is pretty simple. Since it is not located at Google Play Store, the only way how to install the application is to copy it to the device, locate it in device's file manager and tap the APK package file to install it. You may be noticed, that you are trying to install an application from untrusted source, but Android should let you install the application anyway. If not, please check the support internet pages to enable installation of application from other sources than Google Play Store.

6.1.2 Usage

The application is very straightforward to use. First screen you encounter is a welcome screen with a few advices how to properly take a photo of a leaf. You can see the screen on Figure 6.1.

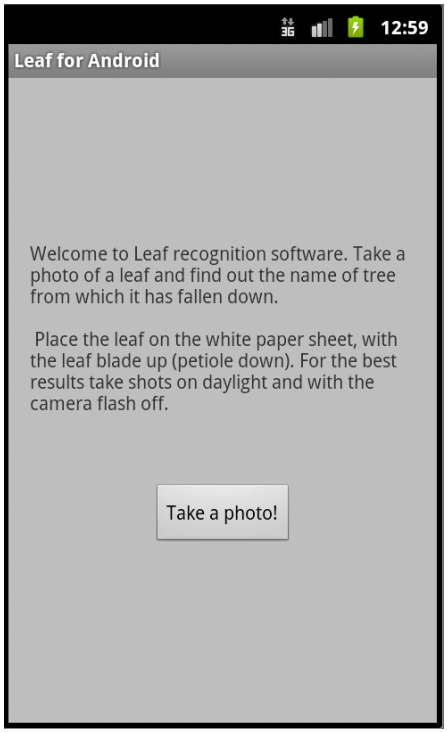


Figure 6.1: Welcome screen.

There is only one button on welcome screen, which says "Take a photo!" ("Vyfotit!" in Czech localization). By clicking on this button, you will be forwarded to standard Android camera application, which allows us to take a picture of a leaf and afterwards it passes the result of the camera back to the Leaf application. Figure 6.2 shows how Android camera application looks like in Android 2.3

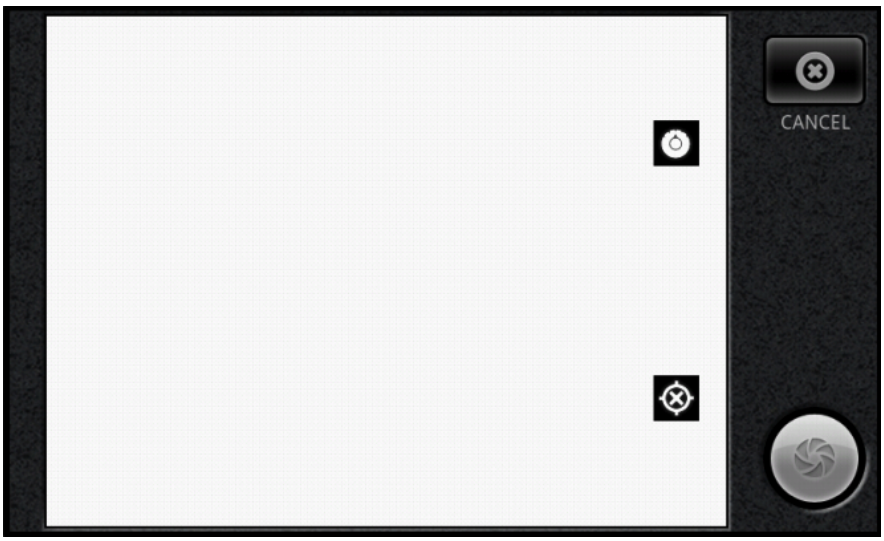


Figure 6.2: Default Android camera application in Android version 2.3.

After the picture is taken, it is immediately processed by the application and subsequently sent to the Leaf server to be recognized. Recognition takes a few seconds and after the process is finished and the result is ready, it is sent back to

the client application, in our case the Android application. Result screen shows the picture taken and the result which is an information about the most probable tree species of the leaf we were processing. On the Figure 6.3 you can see an example of a result screen.

Figure 6.3: Example of a result screen.

As we can see on Figure 6.3, the picture of Android logo is definitely not Ficus benjamina. We are sure, that system has failed to recognize the leaf correctly. In addition, assume that we are able to exactly classify the right tree species for provided leaf. In such case, we can press "Teach it" button ("Naučit" in Czech localization) and then we can teach the system the correct answer. The teaching is very simple, we just pick the right value drop down list with various tree species. The system cannot learn new tree species values through the Android application, but this feature will be implemented in future version of this Android application. The example of the learning screen can be found on Figure 6.4

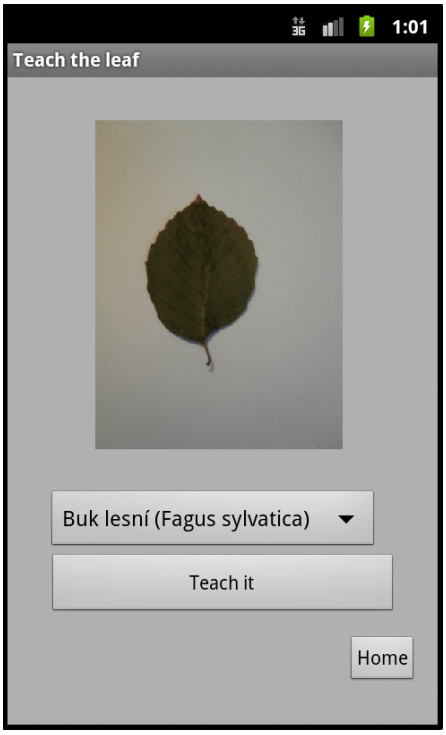


Figure 6.4: Example of a screen, where you can teach the system correct tree species.

6.2 Web application

Web application is again as simple as possible and can be found on http://wholoooo.cz. The application works in two modes: **anonymous** and **logged** in mode. Anonymous user can only get the leaf recognized, and teach the system, logged user can also verify descriptors. Verification is necessary to avoid the degeneration of the knowledge base and also to protect the system from intentional misleading. The welcome screen allows to upload the picture and by pressing "Recognize it" button, system attempts to recognize the leaf (see Figure 6.5)

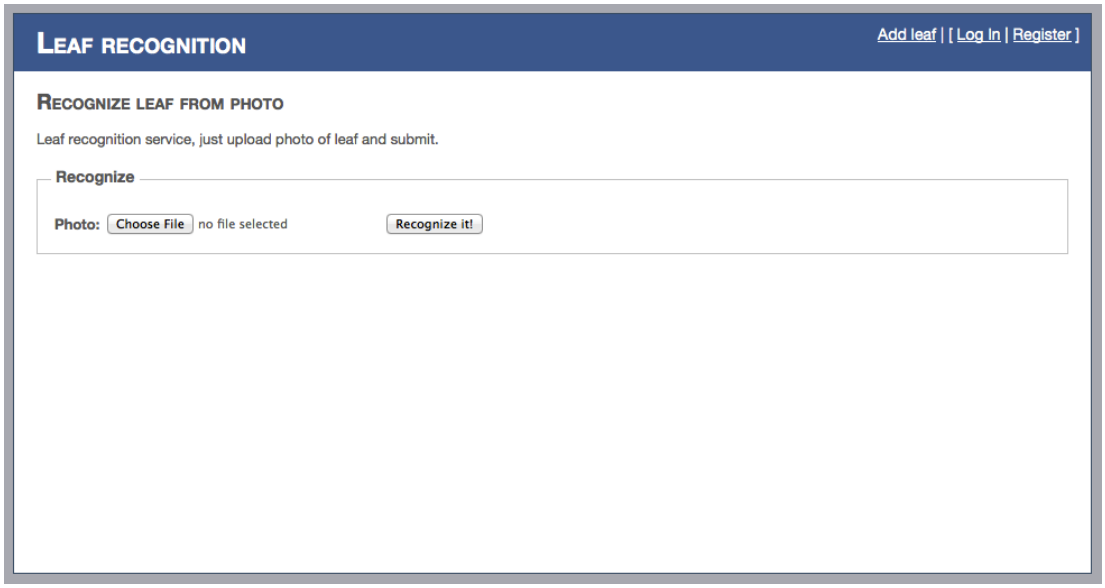


Figure 6.5: Web application home page.

After a few seconds a result screen should appear with the most probable tree species which should correspond with the leaf provided on the picture. The Figure 6.6 shows the possible result screen after the result has been displayed to the user. The provided picture is displayed as well on the right side of the screen.

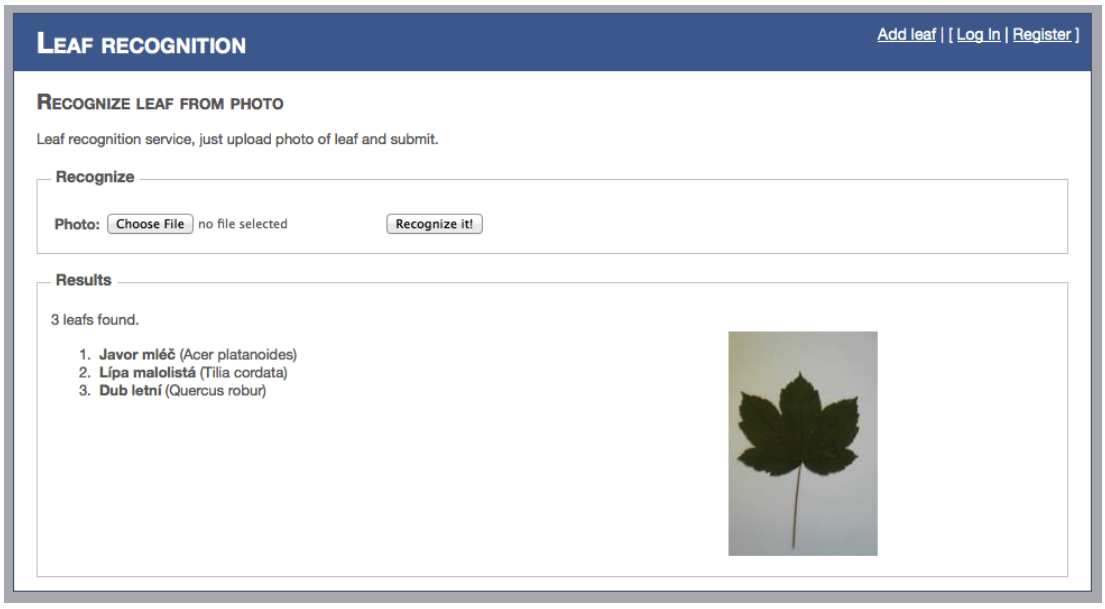


Figure 6.6: Web application result page.

More options became available after logging into the system. The new account can be created after clicking on "Register" link on the top right corner of the home screen. After filling the registration form you are eligible to log in to the system by clicking on "Log In" link also on the top right corner of the home screen.

Since we are logged in, more options became available. First we can try to add a new descriptor to the database (teach the system new leaf). To do that, we have to click on "Add leaf" link on the top menu. The screen contains two significant fields. One is a path to the file which contains the leaf, we want the system to learn it. Second field is a drop down list with tree species the system knows. If the valid tree species are missing, we can easily add them clicking the option –**New tree**– in the bottom of the drop down list. Once we have these fields properly filled, we can press "Save leaf" button and the system tries to extract and save the descriptors. The result of this operation appears after a few seconds on the screen. The result is only for visual confirmation that the descriptors were created successfully. Tree images appear on the screen. First one is the provided image, second one is the last visual phase of Centroid based descriptor extraction, third one is the last visual phase of DFH descriptor extraction. You can observe the result of learning process on the Figure 6.7

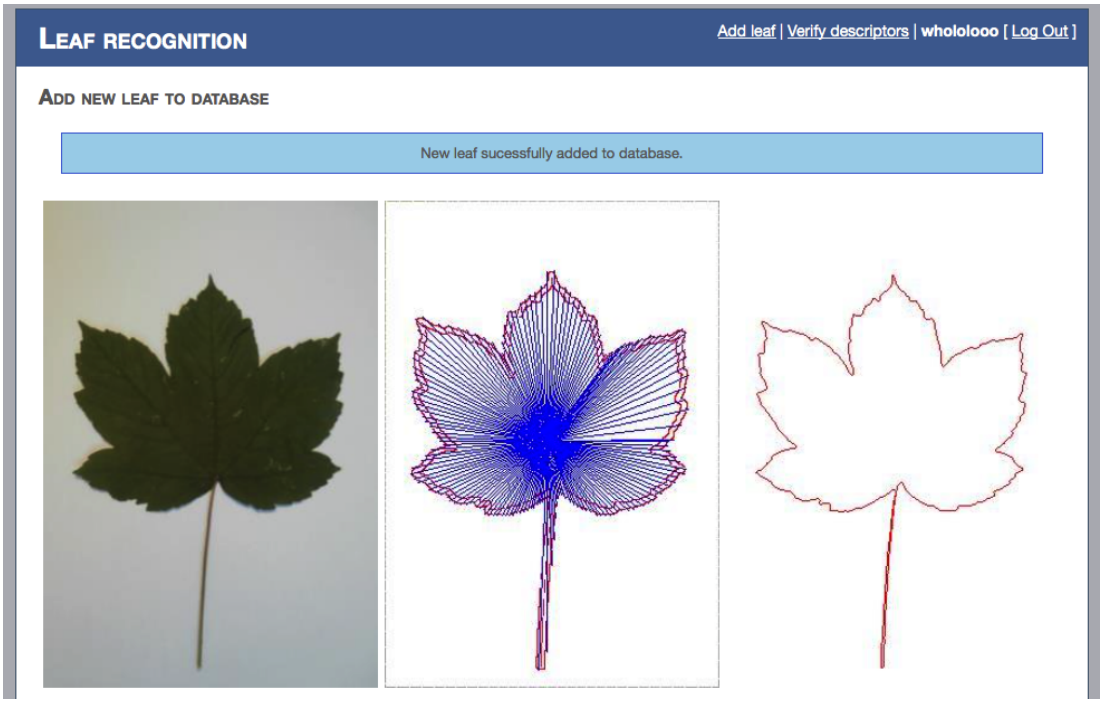


Figure 6.7: Web application learn process result page.

The system does not use newly created descriptors automatically. For the security reasons as described previously they have to be verified first. As logged in user, we have an option on the top menu to step to the descriptors verification section by clicking the "Verify descriptors" link. This screen allows us to verify or deny certain descriptors previously learned by the system. By the drop down list on the left side you can switch among available descriptors. Then they are pairs of pictures, one is an original picture, second one the final step of the image processing before the extraction of the descriptor itself. Below that pair

of pictures there are simply two buttons "Accept" and "Deny" for appropriate action. If the descriptor is generated from defected image, it should be denied for the sake of the knowledge base health. The example of the verification screen can be found on Figure 6.8

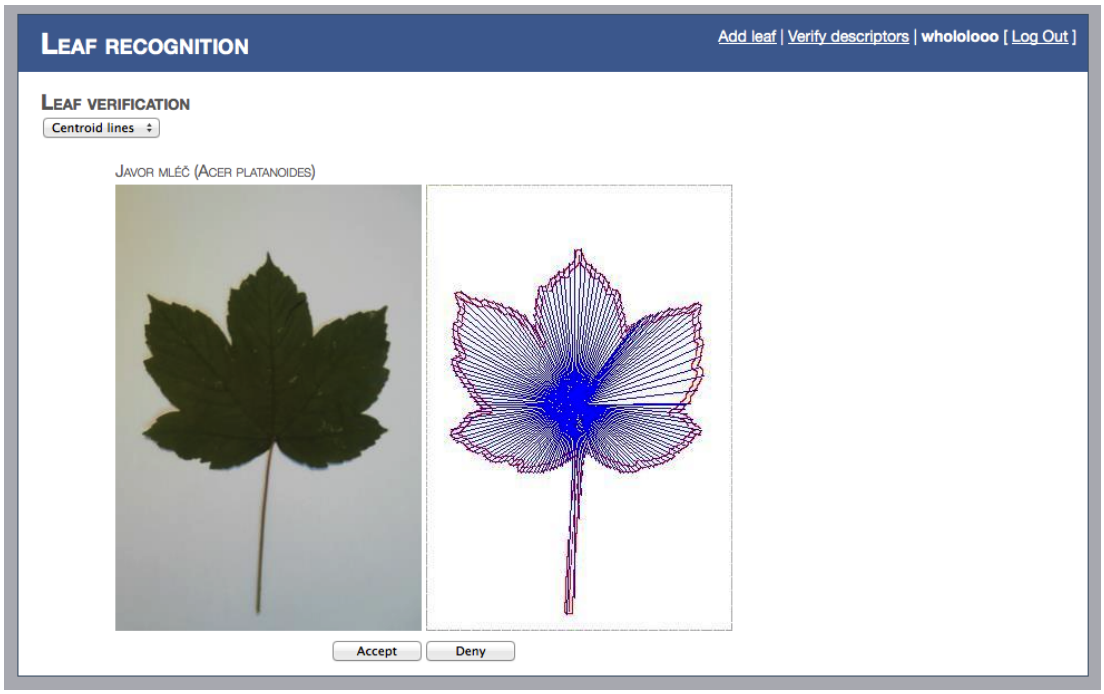


Figure 6.8: Web application descriptors verification page.

Conclusion

In this thesis I successfully implemented a few methods of leaf recognition. The implemented methods are as different in core as possible. By this diversity I wanted to achieve even more exact results in leaf recognition. However, the implemented methods of extraction of descriptor (DFH and Centroid based method) shows very similar success rate in average. At least, the results differ a bit by comparing the exact tree species, e.g. Fagus silvatica shows better results by using DFH algorithm. On the other hand, Centroid based algorithm recognized all Ficus benjamina samples correctly during the comparison test.

Unfortunately, present implementation is very sensitive to picture quality. The system requires at least scan-like quality picture, otherwise current processing method cannot correctly extract the shape of the leaf in the foreground. This is a huge problem of this solution and definitely there is a possibility of improvement.

Another thing I would like to implement in future version of this project is a rotation invariance of descriptors. Current state strictly desires that the leaf in the picture is oriented with the petiole down. Otherwise the descriptor produced by the system would probably match no other descriptor in the descriptor base, or potentially can match the descriptor of another tree species.

Android application is very basic in current state and isn't very fancy. It also misses some features that web application implements. I'm personally not satisfied also with built in camera application. My plan is to replace it with custom one, where the user could see in preview the leaf contour as the extension of the reality. Also there could be some sort of slider to adjust the threshold value for Otsu/Huang threshold algorithms. Although, the Android system is the most popular smartphone system, iOS and Windows Phone devices also represent a significant market share and applications for these platform should be also developed in the near future.

I believe this project has a real potential, however, there still a lot of areas for improvement. Hopefully, I would be able to contribute to the development after the diploma thesis defense and the project would live for many more years.

Bibliography

[1] Manjunath B. S., Salembier Philippe, Sikora Thomass. *Introduction to MPEG-7: multimedia content description interface*. Wiley, 2002. ISBN 9780471486787.

[2] Troelse Andrew. *Pro C# 2010 and the .NET 4 Platform*. Fifth edition. Apress, 2010.

[3] Ben-gan Itzik. *Microsoft SQL Server 2008 T-SQL Fundamentals*. Microsoft Press, 2008.

[4] Tannir, Khaled. *RavenDB 2.X Beginner's Guide*. Packt Publishing, 2013.

[5] Yahiaoui Itheri, Mzoughi Olfa, Bouemaa Nozha. *Leaf shape descriptor for tree species identification*. IEEE International Conference on Multimedia and Expo, 2012.

[6] A. DeLuca, S. Termini. *A definition of a non-probabilistic entropy in the setting of fuzzy set theory*. Int. Control 20, 301-312, 1972.

[7] N. Otsu. *A threshold selection method from gray level histogram*. IEEE Trans. Syst. Man Cybern. 8, 62-66, 1978.

[8] L.K. Huang, M.J.J. Wang. *Image thresholding by minimizing the measures of fuzziness*. Pattern recognition, 41–51, 1995.

[9] Wagenknecht, G. 2007. *A Contour Tracing and Coding Algorithm for Generating 2D Contour Codes from 3D Classified Objects*. Elsevier Science Inc., New York, NY, USA, 1294-1306.

[10] Sun, Jing. 2012. *Image Edge Detection Based on Relative Degree of Grey Incidence and Sobel Operator*. Springer-Verlag, 762-768.

[11] J. Canny. *A computational approach to edge detection.*, IEEE Trans. Pattern Analysis and Machine Intelligence, vol 8, pages 679-714, 1986.

[12] Li Wei, Eamonn Keogh and Xiaopeng Xi. 2006. *SAXually Explicit Images: Finding Unusual Shapes*. In Proceedings of the Sixth International Conference on Data Mining (ICDM '06). IEEE Computer Society, Washington, DC, USA, 711-720.

[13] Dragomir Yankov and Eamonn Keogh. 2006. *Manifold Clustering of Shapes*. IEEE Computer Society, Washington, DC, USA, 1167-1171.

[14] Ken Ueno, Xiaopeng Xi, Eamonn Keogh, Dah-Jye Lee. 2006. *Anytime Classification Using the Nearest Neighbor Algorithm with Applications to Stream Mining* IEEE Computer Society, Washington, DC, USA, 623-632.

Attachments

1. **Compact Disk**
   The part of this thesis is a Compact Disk with the source codes of the server and client applications as well as the digital generated documentation. The digital version of this text is also the part of the CD-ROM content.