AdaMS: Adaptive Mountain Silhouette Extraction from Images

Daniel Braun, Michael Singhof(✉), and Stefan Conrad

Institut für Informatik , Heinrich-Heine-Universität Düsseldorf,
Universitätsstr. 1, 40225 Düsseldorf, Germany
{braun,singhof,conrad}@cs.uni-duesseldorf.de

Abstract. Modern image sharing platforms such as instagram or flickr support an easy publication of photos to the internet, thus leading to great numbers of available photos. However, many of the images are not properly tagged so that there is no notion of what they are showing.

For the example of mountain recognition it is advisable to create reference silhouettes from digital elevation maps. Those are matched with the silhouette extracted from a given image in order to recognise the mountain. It is therefore necessary to obtain a very precise silhouette from the query image.

In this paper, we present AdaMS, an adaptive grid segmentation algorithm, that extracts the silhouette from an image. By the help of an artefact detection method, we find erroneous parts in the silhouette and show how our algorithm uses this information to recalculate the silhouette in the surroundings of the error. We also show that our method yields good results by evaluating our approach on an existing data set of mountain images.

1 Introduction

In times of social media services and a high spread of smart phones, the importance of sharing our experiences with other people rises as part of our today's life. As a result of this, every day the number of publicly accessible photos increases significantly, which can be observed in image sharing platforms like Instagram where users share about 80 million [7] new photos per day. Unfortunately, the majority of these images is not properly tagged and therefore we have no notion of what they are showing, which makes the search for images with specific objects as motif difficult. This leads to a rising need for efficient and precise algorithms for automatic object recognition in images, so that a subsequent tagging, without the need for time consuming human interactions, is possible. Therefore, the significance of this research field increases, what concludes in a high amount of innovations and advances in this area in the last decades.

Our work focuses on the automated landmark recognition for the example of mountain recognition. This task is still challenging, especially because of the problems that are a consequence of the motif itself. These are, for instance, volatile weather conditions, the snowline in combination with clouds,

P. Perner (Ed.): MLDM 2016, LNAI 9729, pp. 98–112, 2016.
DOI: 10.1007/978-3-319-41920-6_8

or vegetation that hides the mountain. As a result, the extraction of meaningful features to describe a mountain, which should be recognized in other images, gets more complex. In addition to this, the appearance of the mountain depends strongly on the viewpoint of the camera, leading to an enormous amount of different feature descriptions for just one mountain to recognize. Therefore, one common method to identify a mountain in an image is to match the silhouette of the mountain with known silhouettes, which for example have been extracted out of a digital elevation map of the mountains to compare with. With the growing spread of devices that have the capability to tag an image with GPS coordinates of the camera position, like smartphones and cameras with an attached GPS unit, this task becomes simpler. This is due to the fact that, with the known position, an algorithm has just to check the surroundings of this location and therefore match the extracted silhouette only against a low number of mountain candidates in this area. However, there are still many images without this advantage, especially in older image collections, so that an algorithm, which can handle images without GPS data, is still a valuable aim. On the downside, we have to check every mountain on the earth, which makes a highly precise silhouette of the mountain in the query image inevitable.

In this work we introduce AdaMS, an adaptive segmentation algorithm, for the purpose of extracting a mountain silhouette out of an image, that tries to reduce the amount of artefacts through segmentation errors or obstacles in the resulting silhouette. For this, we first use an outlier detection algorithm to identify possible artefacts and afterwards classify the encountered outliers to choose the right removal method. If the outlier is classified as a segmentation error, we locally recalculate the segmentation to eliminate it. We therefore use a grid based approach for the segmentation, to get the opportunity to locally modify the segmentation parameters. If, on the contrary, the artefact is caused by an obstacle and therefore the segmentation recalculation would achieve no further improvement, we eliminate this artefact by simply cutting it off.

This paper is structured as follows: In the next section we discuss related work. Afterwards, we introduce our algorithm and give a detailed explanation of its different parts. In chapter four we evaluate the algorithm, before we summarise our work in the last chapter, where we furthermore outline our future work.

2 Related Work

Baatz et al. [2] were the first to target the task of large-scale geo-localisation. For the silhouette extraction they propose an approach which is based on unary data costs for the belonging of a pixel to the sky segment. This approach is combined with a possible user interference, where the user can mark sky or ground pixels for better results. For 49% of the images in their dataset, which they have collected during their research, this user intervention was needed. Thankfully, they have published this dataset, so that it can be used for evaluating our algorithm. For a large scale approach, a fully automated segmentation would be preferable, because every user interference is a bottleneck.

Such an image segmentation is in general a significant technique in many object extraction and recognition tasks. Therefore the research in this field advanced and led to a great number of proposed methods, such as the watershed transform [6], region based algorithms [10] or image clustering [12]. In [10] the authors present a seeded region growing algorithm based on different probability maps calculated for the whole image. Their method relies on homogeneous data and therefore the right probability map for a good segmentation result has to be chosen manually, just as the seed pixel, which determines the region of interest. In contrast to that, our goal is a fully automated parameter determination for the subsequent silhouette extraction. Perner [13] suggests to use case-based-reasoning (CBR) to choose the right parameters for a segmentation task, so that the parameters of the nearest case for an input image are used for the segmentation. In the work of Frucci, Perner and Sanniti di Baja [6], for example, such a CBR system is used for a modified watershed transform algorithm, which reduces the problem of over-segmentation significantly. In comparison to that, we try to choose the parameters locally in the image to automatically reduce the error in respect to the extracted silhouette.

The authors of [12] suggest a k-means clustering for segmentation and introduce an estimation method for the right number of clusters. Therefore, they extract the edges out of the image, by using phase congruency, and merge them as long as the distance of the average colour is below a given threshold. The number of different edges then serves as number of clusters for the subsequent clustering step. For multi-object segmentation both methods can be advisable, but in the case of silhouette extraction we have only two possible classes, namely sky and ground, and therefore the segmentation process has to yield two resulting segments. So, a subsequent region merging step would be necessary to extract the silhouette, but it seems beneficial to use an algorithm, which already takes the requirement of a binarisation for the segmentation step into account and therefore can optimise the result regarding this condition.

This binarisation is, for example, widely used for tasks like character recognition in document images or general foreground/background separation. In the last decades many solutions where presented, like using support vector machines [17], graph cut with Gaussian mixture models [14], global [9,11] and local [16] thresholding, or Markov random fields in combination with the Dempster-Shafer theory [5]. For both tasks, global threshold based algorithms, like Otsu's method [11] or the algorithm from Kittler and Illingworth [9], are often successfully used to partition the image into two groups of pixels [15]. A related technique is proposed in [17]. Their algorithm first clusters the pixels with fuzzy C-means and thereafter the resulting membership values of randomly chosen pixels are used to train a SVM, which serves as a kind of separating threshold, for classifying the remaining pixels. But as a result of the given conditions in mountainous regions an image thresholding algorithm based only on the brightness or colour values of the whole image is not advisable. This is due to the common inhomogeneity of the brightness values, be it the sky, for example through weather conditions, or the ground in general, and a possible fluent passage between the sky and the ground segment. Both lead to a high variation in illumination and contrast which can result in a non-optimal threshold for finding the exact silhouette of the mountain. To overcome similar problems, local thresholding techniques have been proposed. Roman et al. [16], for example, present a local thresholding, which calculates a threshold for a pixel relying on the mean and the mean deviation in a local window around it, by using the integral sum for efficient computation.

With the target of only extracting the border between mountain and sky and the prior knowledge that the sky will be a portion of the upper part of the image, we use a seed growing algorithm with local constraints to find the initial silhouette of the mountain. Having the other target of an unsupervised silhouette extraction process, solutions like GrabCut [14], in which the user gives first a bounding box to locate the foreground object and then possibly interferes to optimise the result, respectively the technique proposed from Chen et al. [5], which needs the user to mark some foreground/background pixels before the segmentation starts, are not feasible. Even so, the GrabCut algorithm can be used for an automated segmentation, if we have some prior knowledge over the background of the image.

Another way to find the mountain silhouette is to analyse all edges in the image in respect to their plausibility to be part of it. The authors of [8] extract the edges in the images with the Canny algorithm [4] and use different filters to reduce the candidate edges. Afterwards they use a measure to find the silhouette with the highest probability to be the skyline. By choosing only one resulting silhouette, this suffers from edges with low contrast, which leads to a fragmented skyline and therefore only a part of the skyline will be found. Ahmad et al. [1] use both, edge-less and egde-based approaches, to find the horizon line as shortest path in a classification map, which values represent the horizon-ness of each pixel, with dynamic programming. A related technique is presented in [3], where the authors use an edge map to find the right terrain alignment by matching this against an edge map created with a digital elevation map. Contrary to our target, this method needs the knowledge of the viewpoint location, through GPS data, and they do not extract the skyline particularly.

3 AdaMS Extraction

As we stated in the first chapter, the use of digital elevation maps to identify the mountain in a query image is beneficial. But without the use of prior knowledge, like GPS data or user pre-selection of possible locations, our algorithm has to check every mountain in the database, which means, that we have to compare the silhouette of the query image with many, through view point relocation, possible silhouettes of these mountains. Therefore we need a highly accurate query silhouette to achieve a good precision in the subsequent matching process. To achieve the desired high accuracy, our AdaMS (Adaptive Mountain Silhouette) extraction algorithm identifies possible artefacts in the extracted silhouette and tries to eliminate them through a refined segmentation, if it is a

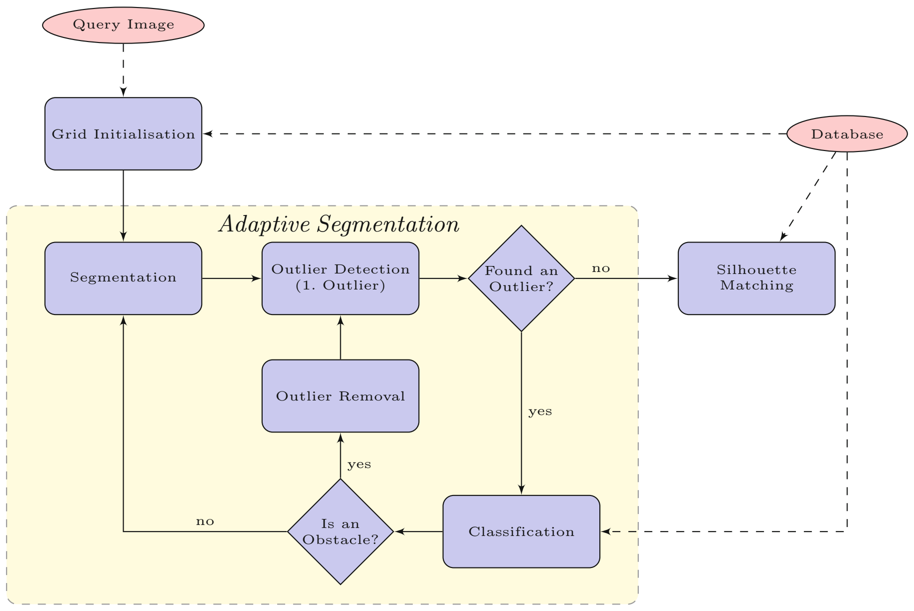

Fig. 1. Flow diagram of AdaMS.

segmentation error, respectively a straight forward removal policy, if the artefact is caused by an obstacle. These different artefact types are described in the following subsection, where we additionally point out some causes for these artefacts. Afterwards, we define the grid which will be used for the segmentation process. Finally, we introduce in the following subsections the five steps of our silhouette extraction algorithm. As can be seen in Figure 1 these steps are grid initialisation, image segmentation, silhouette extraction, outlier detection, and silhouette refinement. The latter one differentiates between outlier removal and recalculation of the segmentation for some parts of the silhouette.

The grid initialisation overlays the image with a grid with predefined grid element spacing. In the segmentation step, in which we use some pixel marked as sky of the image as seed to let the sky segment grow, we finally extract the transition between sky and ground segment as initial silhouette. This silhouette is passed to the outlier detection step, to find all anomalies in its structure which are then classified. For every segmentation error the algorithm tries to recalculate the segmentation in a local area around the artefact.

3.1 Artefact Definition

For the silhouette extraction we have to identify the pixels in the image which are part of the sky. Therefore we have a binary segmentation task to solve, which suffers from possible error sources with the result of possible artefacts in the extracted silhouette. But as a whole we can condense these into the following three main sources: The first one is a general problem with images, where several problems, like noise pollution, background clutter, blur, strong illumination variations, or a reduced contrast of the skyline through over-/underexposure, can occur. As result of such an error, the extracted silhouette can have a saw-tooth-shaped appearance. The second one is a consequence of the motif itself, because of highly volatile weather conditions, clouds or mist can hide parts of the mountain silhouette completely or reduce the contrast of it significantly, especially in combination with snow on mountain peaks, so that a separation of the sky from the ground gets difficult. The last main error source is a result of the natural environment of the mountains. Therefore we have to deal with different possible obstacles, like trees or other vegetation, humans or buildings in the foreground, or cables from a ski-lift, in the image, which partly cover the mountain. Altogether, this problems make the segmentation and therefore the extraction of the real mountain silhouette error prone.

The disparity of these errors leads to another problem, because an algorithm which solves one of it may not be ideal for the other ones. An artefact, which is a result of a low contrast at the mountain silhouette, may be solvable through a modification of the segmentation parameters. But if an obstacle leads to an erroneous silhouette, the segmentation step has worked properly, so that another parameter set will not lead to any improvement. Therefore, if we identify an artefact in the extracted silhouette, we have to classify it, so that we can choose the right method to get rid of it.

3.2 Grid

For the binary segmentation of the image we use a seed based region growing algorithm. This means that the algorithm chooses some start points as sky and then adds every neighbouring pixel that fulfils certain conditions.

For this, we define a grid G as follows:

Definition 1. *Given an image I with the width w_I and height h_I. Then the grid G with step size d_G consists of the pixels*

$$p_{i,j} = (\frac{d_G}{2} + id_G, \frac{d_G}{2} + jd_G)$$

with $0 \leq i < \frac{(w_I - \frac{d_G}{2})}{d_G}$ *and* $0 \leq j < \frac{(h_I - \frac{d_G}{2})}{d_G}$.

This results in a total number of

$$n = \lceil \frac{(w_I - \frac{d_G}{2})}{d_G} \rceil \cdot \lceil \frac{(h_I - \frac{d_G}{2})}{d_G} \rceil$$

grid points g_i with $i \in \{1, \ldots, n\}$ defining n grid cells C overlaying the image.

3.3 Segmentation

For the region growing algorithm we have to specify some seed points, where the algorithm starts to add connected points, if they satisfy a given condition. For this task we can use the assumption, that the sky will be localised at the top of the image. Because of this, we choose the highest pixel row in the image, mark it as sky and finally use this set of pixels as initial sky segment.

Starting at this seed points, the algorithm then adds successively every 8-connected pixel to the sky segment, if for the pixel holds

$$|B_{p_{(x,y)}} - mean^r_{p_{(x,y)}}| < \gamma \cdot \sqrt{V_G}$$

with $B_{p_{(x,y)}}$ the brightness of the pixel $p_{(x,y)}$, $mean^r_{p_{(x,y)}}$ as the mean of the brightness in a neighbourhood of the pixel with the radius of r and γ as a scaling factor. When the algorithm cannot add any further point, the binary segmentation ends. Now the silhouette extraction could start, but there are two drawbacks at this point. First, it is possible, that for instance clouds have been marked as ground and therefore the sky segment is not homogeneous. To get rid of such additional ground patches, we eliminate every patch which has no direct connection to the right or the left border of the image, resulting in only two uniform segments for the sky respectively ground. Finally, through small segmentation errors at the border between sky and ground, the transition can have small parts with the width of one pixel, which would result in a jagged silhouette. To remove this parts, we use erosion as a morphological operation. Therefore, a square with the width w_{struct} centred around each pixel acts as structuring element. Having the class $C_{p_{(x,y)}} \in \{sky, ground\}$ we count every pixel in the hereby defined neighbourhood with the same class and switch $C_{p_{(x,y)}}$ if the ratio of this pixels is below the threshold ζ. After this cleaning step we end with a smoothed border between the two parts of the image.

At this point, the algorithm starts the silhouette extraction by finding a ground pixel in the first pixel column on the left side of the image. If it finds no appropriate point, the algorithm will check the next columns until a ground pixel is found or the right side of the image has been reached. Afterwards it tracks the transition between sky and ground and adds every pixel on the upper border of the ground as vertex $v = (x, y)$, with x and y the pixel coordinates in the image, of the polygonal chain S which constitutes the silhouette of the mountain.

As last step of the extraction, our algorithm eliminates every vertex for which it holds that it is positioned on the line between its predecessor and its ancestor. In this case, the point carries no further information regarding the extracted silhouette and therefore it can be safely removed.

3.4 Outlier Detection

As stated before, the silhouette S extracted in the previous step can contain artefacts and therefore the algorithm now tries to identify the erroneous parts of it. For that, we first convert the silhouette to a representation which is independent from the absolute position of each vertex in the query image. This is necessary, because otherwise equal parts of the different silhouettes could be different through varying positions, which makes pattern recognition difficult. Therefore we define the following representation that is based on relative positions of the vertices and similar to polar coordinates with complex numbers.

Definition 2. *A* relative silhouette $RS = (v_1, \dots, v_n)$, $n > 0$, *is a polygonal chain with* $v_i = (l_i, a_i)$ *for all* $1 \leq i \leq n$, *where* $l_i > 0$ *is the length of a line segment and* $a_i \in (-180°, 180°]$ *is the angle relative to the* x*-axis.*

Using this representation of the silhouette we now compare parts of a relative silhouette to reference data R, free of outliers, by computing histograms on the silhouettes and computing a distance between the histograms. These histograms contain the same two dimensions as the relative silhouette, namely segment length and angle relative to x-axis. In the following, by H_R we denote the reference histogram, i.e. the histogram computed from the reference data.

Definition 3. *Given a relative silhouette* RS, *then* $H_{RS}(s, l)$ *denotes the histogram consisting of the points* $v_s, \dots, v_{s+l-1}$ *of* RS *and* H_{RS} *denotes the histogram over all vertices of* RS.

Now, for a silhouette $RS = (v_1, \dots, v_n)$ of a query image we compute the histograms $H_{RS}(1, l)$ to $H_{RS}(n - l, l)$ and their respective distances to the reference histogram $d_i = \text{dist}(H_{RS}(i, l), H_R)$ via a sliding window approach. Then the anomaly score $an(v_j)$ of a vertex v_j is computed as the average of the set of distances $\{d_i | i \leq j \wedge j < i + l\}$, i.e. the distances of histograms that are computed over parts of the silhouette that contain v_j.

Based on the mean μ and the standard deviation σ for the single vertices' distribution of anomaly scores, we introduce two thresholds τ_{in} and τ_{out} similar to the double threshold approach in [4]. These lead to two kinds of anomalies.

Definition 4. *Let* $RS = (v_1, \dots, v_n)$ *be the silhouette of an image with corresponding anomaly scores* $an(v_i)$ *for the vertex* v_i, *reference outlier score distribution mean* μ *and standard deviation* σ *and the two thresholds* $0 < \tau_{out} < \tau_{in}$.

Then we call v_i *a* weak anomaly *if*

$$an(v_i) \geq \mu + \tau_{out} \cdot \sigma$$

and a strong anomaly *if*

$$an(v_i) \geq \mu + \tau_{in} \cdot \sigma.$$

Note, that a strong anomaly always is a weak anomaly, too. The second thing to remember here, is, that for ease of understanding, we do *not* use "anomaly" and "outlier" synonymously. With "anomaly" we refer to a single vertex in a polygonal chain as in the definition above. In contrast to this, with "outlier" we mean a part of a polygonal chain that consists of anomalies as the following definition states.

Definition 5. *Let $l > 0$, and RS, $an(v_i)$, μ, σ, τ_{in} and τ_{out} as in definition 4. We call $o = (v_i, \ldots, v_j)$ an* l-outlier *if the following is true:*

1. *For all v_k, $i \leq k \leq j$, it holds that v_k is a weak anomaly.*
2. *There exist $m, n \in \{i, \ldots, j\}$ such that $n - m \geq l$ and for all v_k, $m \leq k \leq n$, it holds that v_k is a strong anomaly.*

An outlier $o = (v_i, \ldots, v_j)$ is called a maximum l-outlier *if and only if neither $(v_{i-1}, \ldots, v_j)$ nor $(v_i, \ldots, v_{j+1})$ are l-outliers.*

In the outlier detection step, only maximum l-outliers are of interest, so, whenever we find l consecutive strong anomalies we expand the outlier as long as there are adjacent weak anomalies to that outliers. Given a silhouette RS, we are thus able to find all outliers in linear time relative to the length of the silhouette, since anomaly score computation can be accomplished in linear time as well.

3.5 Outlier Classification

As discussed in section 3.1, different types of outliers exist. For the classification process we use four classes of outliers. All kinds of obstacles are summarised in the class *obstacle*. For segmentation errors, we introduce two classes. If parts of the mountain are recognised as sky, i.e. the silhouette is too far downwards in the image, these outliers are classified as *segUp*. On the other hand, if parts of the sky are recognised as mountain and the silhouette is too far upwards, outliers are classified in *segDown*. Finally, we introduce a class for false positives that contains parts of the silhouette, that get marked as outliers but are not outliers, so we end up with a set of outlier classes

$$OC = \{obstacle, segUp, segDown, falsePos\}.$$

For the outlier classification itself, we choose a weighted k-nearest neighbour approach, using a manually tagged dataset of 80 outliers, 20 of each class, as reference data for the classification. Assuming this module gets an outlier o from the detection module, we compute the distance of its histogram H_o to all reference outlier histograms H_i^{ref} with $1 \leq i \leq 80$, resulting in a set of k nearest outliers, each saved as tuple of type and distance $\{(t_i, d_i)\}$. Finally, for every type $type_j \in OC$ we compute $s_j = \sum_{t_i = type_j} w_i$ and declare o to be of the type with maximum s_j. Here, the weighting w_i for the ith tuple is given by $w_i = \frac{1}{d_i + \varepsilon}$ with ε the smallest positive double in the programming language, since distances can become 0. Having classified the outlier, there are three possible ways to further process it. First, the outlier can be a false positive, which results in deleting it by marking this part of the silhouette as non-outlier. If it is classified as a real outlier, it is, in the case of an obstacle, passed to the artefact elimination tool, described in section 3.6 or, if it is a segmentation error, passed to the segmentation module introduced in section 3.7 for a local refinement.

3.6 Artefact Elimination

In the case of an obstacle hiding the mountain a re-segmentation would be pointless, because the first segmentation has worked properly and a less conservative segmentation can result in a jagged mountain silhouette if the algorithm marks too many pixel as sky. Furthermore, the obstacle hides the background completely, leaving no further information of the mountain contour which could be used for a preliminary processing of this part of the image. Therefore, a good way to handle such an outlier is to cut it off and replace it by a straight line, since, if the width of the outlier is small enough, the mountain will most likely have no greater curves at this place. But even for a greater obstacle such a replacement will be in most cases more natural for the mountain than the obstacle itself. Given the outlier o of length n, with v_1 and v_n as start respectively end vertex, our algorithm removes the obstacle in the following manner: First, we construct the line connecting both endpoints $l = \overline{v_1 v_n}$ and calculate the distance $d(v_i, l)$ for every vertex v_i, resulting in a list of distances D. Then, beginning at the start vertex v_1 and with a threshold parameter δ, the algorithm searches a vertex for which holds

$$d\left(v_i, l\right) > \delta \wedge \forall v_j \left(j < i \wedge d\left(v_j, l\right) \leq \delta\right).$$

In words, this is the first vertex with a distance exceeding the threshold and consecutive predecessors with distances smaller than δ. If on the one hand no vertex is found, every outlier vertex lies near the connecting line l and therefore the chance for an error, be it through the detection or classification step, rises. Furthermore, with a δ small enough, the impact on the silhouette precision should be negligible, so that the algorithm marks this part as non-outlier. If on the other hand a vertex is found, the algorithm searches such a point in respect to the end vertex and though considering all successors, returning the two vertices v_{low} and v_{high}, whose connecting line $\overline{v_{low} v_{high}}$ is used to remove the obstacle.

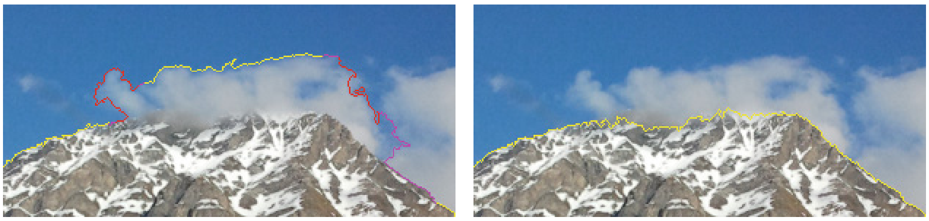
Fig. 2. An example of the local refinement. Left: The initial silhouette with the marked outliers. Right: The resulting silhouette after the local refinement terminated.

3.7 Local Refinement

Segmentation errors can occur when the segmentation is too conservative, which implies that the scaling factor γ is too low, and therefore clouds are not properly marked as sky (*segDown*) or when the contrast of the transition between sky and ground is low, so that the segmentation algorithm starts to add ground pixels to the sky segment (*segUp*), which is a result of a γ too high for this part of the image. With the knowledge of the outlier's class, a recalculation with a changed γ may solve the problem, as long as this refinement is localised around the outlier, because a changed value for the whole image could make the result at other parts of the silhouette even worse. Therefore a local refinement for the outlier removal is the target. For that purpose, we reuse the grid G defined in definition 1. Having the outlier o, the set C_o^{alter} of grid cells containing parts of the outlier and C_o^{update}, the union of C_o^{alter} and all directly connected grid cells, we define the local refinement step as follows.

Given the alteration rate θ, α as the predefined alteration value and the grid cells $C \in C_o^{alter}$ we first compute the new scale factor

$$\gamma_C^* = \alpha \cdot \theta + \gamma_C.$$

Here, the sign of α and therefore the direction of the alteration of γ_C is determined by the class of the segmentation error as follows:

$$\alpha = \begin{cases} 1, & \text{if } o \text{ has been classified as } segDown \\ -1 & \text{else.} \end{cases}$$

Afterwards, we recalculate the segmentation, using

$$|B_{p_{(x,y)}} - mean^r_{p_{(x,y)}}| < \gamma_C^* \cdot \sqrt{V_G},$$

for every point $p_(x,y)$ which lies in a grid cell $C \in C_o^{update}$. The resulting silhouette is then extracted and passed back to the outlier detection module. This will be repeated until there are no outliers classified as segmentation errors left or I^{MAX} iterations have been executed. The latter one will result in the use of the originally extracted silhouette by additionally marking it as unsure silhouette. An example for the result after some refinement steps can be seen in Figure 2.

3.8 Global Refinement

If the ratio of detected segmentation errors to the length of the silhouette exceeds a predefined threshold θ, we recalculate the whole segmentation. This is due to the assumption, that such a high outlier ratio is a sign for a messed up segmentation through a wrong initial γ for the whole image. Therefore, we alter each grid point value and restart the segmentation on the whole image. For the selection of the right γ we use a grid search, using a predefined set of possible values, and choose the resulting silhouette with the lowest outlier ratio as start silhouette for the possibly needed local refinement.

4 Evaluation

In the following we evaluate the preciseness of our silhouette extraction approach on the Switzerland dataset from [2]. This dataset consists of 203 images

Table 1. Comparison of extracted silhouettes to ground truth. Values are given in pixels. RG1 and RG2: Region growing with a threshold of 1 respectively 2.

Parameter	GrabCut	Otsu	RG1	RG2	AdaMS
Avg. initial distance					18.80
Avg. final distance	26.03	118.74	27.61	49.31	15.75
Median final distance	2.58	97.44	1.65	1.81	1.43

and corresponding silhouettes as extracted by the authors. We use these silhouettes as ground truth here, since they have been refined manually. However, in some cases there are differences between the approaches. For example, in the ground truth silhouettes, most obstacles have not been removed. Therefore, in our approach, we skip the obstacle removal step in order to get comparable results. We use GrabCut [14], Otsu [11] and a region growing algorithm with a predefined threshold for the maximal allowed divergence in the brightness values of 4-connected pixels in the sky segment, for a comparison of our algorithm with other segmentation methods. GrabCut and the region growing algorithm need some given background pixels and therefore we choose, equally to our approach, the upper row as background. For the latter one, we furthermore choose the values 1 (RG1) and 2 (RG2) as threshold for the maximal brightness difference.

For comparing two silhouettes we use the following distance measure:

Definition 6. *Let S_1 and S_2 be the two silhouettes with n_1 respectively n_2 pixels and $pd(p_i^1, S_2)$ be the pixel distance from the point p_i^1 of S_1 to S_2 defined by*

$$pd(p_i^1, S_2) = \min_{1 \leq j \leq n_2} \{|y_i^1 - y_j^2| : x_i^1 = x_j^2\}.$$

Then the distance between S_1 and S_2 is given by

$$\max(\sum_{i=1}^{n_1} pd(p_i^1, S_2), \sum_{i=1}^{n_2} pd(p_i^2, S_1)).$$

Table 1 shows some aggregated results for the comparison. Average initial distance gives the average distance of the silhouettes after the first iteration, i.e. with the initially computed silhouette. Therefore, only our approach has such a value, because for the other algorithms there exists no refinement step. Average final distance gives the distance values after all iterations of the refinement step have been completed. Median final distance is the median of the final distances. As mentioned before, the fact that Otsu finds the threshold by considering the whole image seems to be a great drawback for this task and this can be seen in the results. With an average error of 118.74 pixel per vertex and the fact that for six images the results are too fragmented to extract the silhouette properly, without choosing the bottom pixel row as silhouette candidate, Otsu yields the worst results. On the other side, the region growing algorithm yields relatively good results for a threshold of 1, except for 2 images for which the silhouette extraction fails due to an extremely low contrast. This shows, that the difference

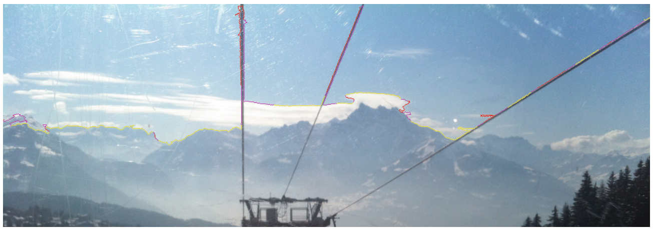

Fig. 3. Huge distance between extracted silhouette and ground truth because we do not cut out obstacles.

in the illumination in the sky segment is very low in many images of the dataset. But increasing the threshold leads to worse results, because the contrast in most images is low and therefore the sky segment grows far to wide, so that, additionally to the higher average distance, the silhouette extraction fails for 19 images when we choose 2 as threshold. This is in contrast to AdaMS and GrabCut. For both, the silhouette extraction has found proper silhouette candidates for all 203 images of the dataset.

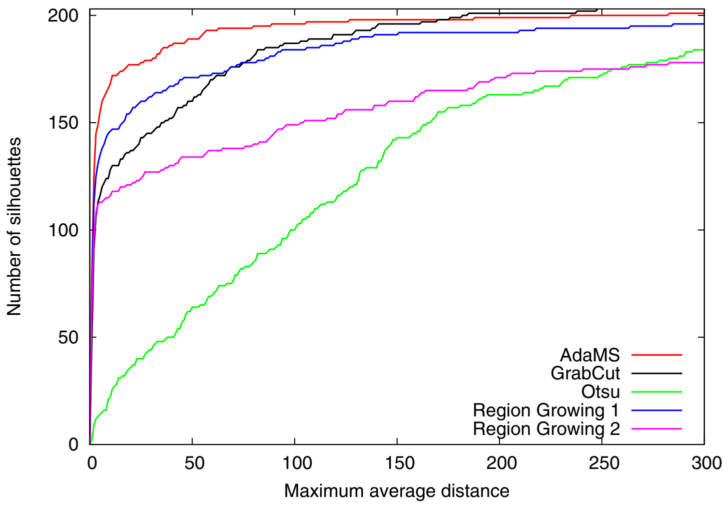

Fig. 4. The cumulative distances of the different algorithms.

The results of GrabCut are nearly equal to RG1 in respect to the overall average distance. But as can be seen in the median value, the region growing algorithm works better on most of the images and furthermore our adaptive approach yields far better results, even for the initial segmentation. In Figure 4 we visualise this by the cumulative sum of the distances. As can be seen, the presented approach can find a good silhouette in significantly more images than the other algorithms. On the other side, GrabCut seems to be a bit more stable in respect to the distance for completely wrong segmentation results, but average distances over 10 pixels are nonetheless not very helpful for a subsequent matching.

Even so, the average initial distance of our algorithm seems to be relatively large with an error of nearly 19 pixels as is the average final distance with a distance of more than 16 pixels. If we compare this, however, to the median final distance, which reaches a good value with 1.43 pixels, it becomes clear, that the high average number is due to only a few images causing problems to our approach. Mostly this is due to omitting the obstacle removal step, as Figure 3 illustrates. It can be seen here, that our approach marks the ropeway's cables as outliers, however, we do not remove those and hence get huge distances in some images.

In comparison to a fixed threshold, our approach regards the global and local appearance of the image to find a better measure for the membership of a pixel to the sky, which leads to a better silhouette for the images with low contrast. In conclusion, we get a great improvement with the precision by the adaptive refinement presented here. Our experiments show that the adaptive refinement improves the quality of the extracted silhouette by 16%.

5 Summary and Future Work

In this work we have proposed a baseline system for an adaptive segmentation approach that is focused on the precise extraction of mountain silhouettes from images of mountains. Our results show that our adaptive solution has great advantages over our initial segmentation and comparable algorithms.

In the future, we aim to make the seed point selection for the grid initialisation more precise by using different feature descriptors combined with a classification step to find pixels with a high probability to be part of the sky, so that we can drop the assumption, that the upper row will be part of the sky. We also want to improve the outlier detection and classification by considering further information such as edge strength and the usage of more than one reference histogram for the description of normal mountain silhouettes.

Furthermore we are currently working on a data set for mountain recognition with ground truth notation that we aim to make publicly available.

References

1. Ahmad, T., Bebis, G., Nicolescu, M., Nefian, A., Fong, T.: Fusion of edge-less and edge-based approaches for horizon line detection. In: 6th IEEE International Conference on Information, Intelligence, Systems and Applications (IISA 2015), Corfu, Greece, July 6–8, 2015. IEEE (2015)

2. Baatz, G., Saurer, O., Köser, K., Pollefeys, M.: Large scale visual geo-localization of images in mountainous terrain. In: Fitzgibbon, A., Lazebnik, S., Perona, P., Sato, Y., Schmid, C. (eds.) ECCV 2012, Part II. LNCS, vol. 7573, pp. 517–530. Springer, Heidelberg (2012)
3. Baboud, L., Čadík, M., Eisemann, E., Seidel, H.P.: Automatic photo-to-terrain alignment for the annotation of mountain pictures. In: 2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE (2011)
4. Canny, J.: A Computational Approach to Edge Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence **PAMI-8**(6) (November 1986)
5. Chen, Y., Cremers, A.B., Cao, Z.: Interactive color image segmentation via iterative evidential labeling. Information Fusion **20** (2014)
6. Frucci, M., Perner, P., Sanniti Di Baja, G.: Case-based-reasoning for image segmentation. International Journal of Pattern Recognition and Artificial Intelligence **22**(05) (2008)
7. Instagram (accessed January 1, 2016). https://instagram.com/press/
8. Kim, B.J., Shin, J.J., Nam, H.J., Kim, J.S.: Skyline extraction using a multistage edge filtering. World Academy of Science, Engineering and Technology **55** (2011)
9. Kittler, J., Illingworth, J.: Minimum error thresholding. Pattern Recognition **19**(1) (1986)
10. Mancas, M., Gosselin, B., Macq, B.: Segmentation using a region-growing thresholding. In: Electronic Imaging 2005. International Society for Optics and Photonics (2005)
11. Otsu, N.: A threshold selection method from gray-level histograms. IEEE Transactions on Systems, Man and Cybernetics **9**(1) (1979)
12. Patil, R., Jondhale, K.: Edge based technique to estimate number of clusters in k-means color image segmentation. In: 2010 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT), vol. 2 (2010)
13. Perner, P.: An architecture for a CBR image segmentation system. Engineering Applications of Artificial Intelligence **12**(6) (1999)
14. Rother, C., Kolmogorov, V., Blake, A.: "GrabCut": Interactive foreground extraction using iterated graph cuts. In: ACM Transactions on Graphics (SIGGRAPH), vol. 23(3) (2004)
15. Sezgin, M., Sankur, B.: Survey over image thresholding techniques and quantitative performance evaluation. Journal of Electronic Imaging **13**(1) (2004)
16. Singh, T.R., Roy, S., Singh, O.I., Sinam, T., Singh, K.M.: A new local adaptive thresholding technique in binarization. IJCSI International Journal of Computer Science Issues **8**(6) (2011)
17. Wang, X.Y., Zhang, X.J., Yang, H.Y., Bu, J.: A pixel-based color image segmentation using support vector machine and fuzzy -means. Neural Networks **33** (2012)