Università degli Studi di Padova

FACOLTÀ DI INGEGNERIA
Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di laurea magistrale

Color and depth based image segmentation using a game-theoretic approach

Candidato:
Martina Favaro
Matricola 681484

Relatore:
Prof. Pietro Zanuttigh

Correlatore:
Prof. Andrea Albarelli

Anno Accademico 2011–2012

Contents

List of Tables

List of Figures

Abstract

In this thesis a new game theoretic approach to image segmentation is proposed. It is an attempt to give a contribution to a new interesting research area in image processing, which tries to boost image segmentation combining information about appareance (e.g. color) and information about spatial arrangement.

The proposed algorithm firstly partition the image into small subsets of pixels, in order to reduce computational complexity of the subsequent phases. Two different distance measures between each pair of pixels subsets are then computed, one regarding color information and one based on spatial-geometric information. A similarity measure between each pair of pixel subset is then computed, exploiting both color and spatial data. Finally, pixels subsets are modeled into an evolutionary game in order to group similar pixels into meaningful segments.

After a brief review of image segmentation approaches, the proposed algorithm is described and different experimental tests are carried up to evaluate its segmentation performance.

Chapter 1

Introduction

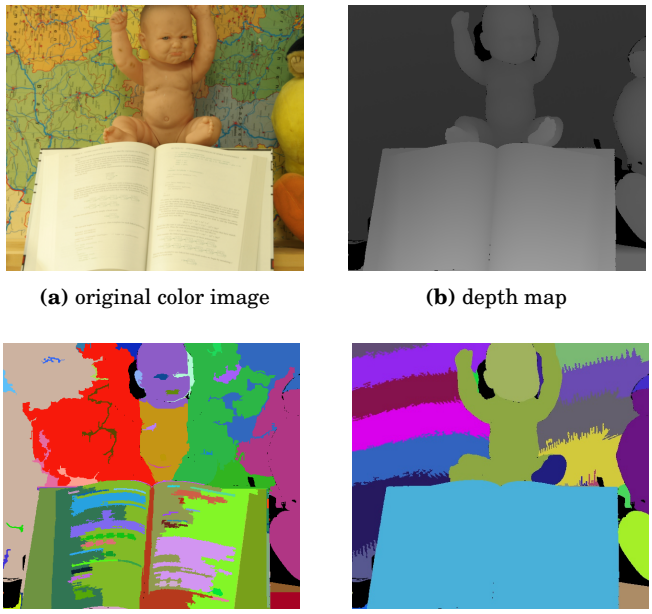
In computer vision, segmentation refers to the process of partitioning a digital image into its constituent regions or objects. That is, it partitions an image into distinct regions that are meant to correlate strongly with objects or features of interest in the image. Image segmentation usually serves as the pre-processing before image pattern recognition, image feature extraction and image compression, or more generally, it is the first stage in any attempt to analyze or interpret an image automatically. The success or failure of subsequent image processing task is often a direct consequence of the success or failure of segmentation.

The relevance of segmentation task is precisely the reason why, since its formulation in the '70, hundread of different techniques have been proposed, mainly focusing on intensity-based and color image segmentation.

Nevertheless, if the goal is to separate objects in the image with respect to a semantic meaning, image segmentation based only on color information can fail to distinguish physical distinct object. For example, Figure 1.1 shows that information provided by the color image is not sufficient to discern the baby doll from the background.

The advent of relatively cheap techniques capable of acquiring or computing a 3D reconstruction of real world scenes may help to overcome this weakness, providing useful information about depth and position of the objects in the view. There are two many group of methods, passive methods or active methods.

Passive methods use only the information coming from two or more standard cameras to estimate depth values. Among them stereo vision systems (for example [3] and [4]), that exploit the localization of corresponding locations in the different images, are perhaps the most widely used. Stereo vision systems have been greatly improved in the last years but they can not work on untextured regions and the most effective methods are also very computational time consuming.



(a) original color image

(b) depth map

(c) segmentation based on color information

(d) segmentation based on depth information

Figure 1.1: Color vs depth segmentation of Baby2 image of Middlebury Stereo Dataset [2]

Active methods, eg. structured light and Time of Flight (ToF) sensors, interfere radiometrically with the reconstructed object. In ToF sensors a laser is used to emit a pulse of light and the amount of time before the reflected light is seen by a detector is timed; since the speed of light is known, the round-trip time determines the travel distance of the light. Structured-light scanners (such as Microsoft Kinect) project a pattern on the subject, then look at the deformation of the pattern on the subject and uses a technique similar to triangulation to calculate the distance of every point in the pattern. Such methods can obtain better results than passive stereo vision systems, but they are also usually more expensive.

Depth based segmentation is usally more robust and can lead to better segmentation performance. In fact, abrupt changes in depth values usually correspond to object boundaries, while this is not always true in color images, for example in a textured region. On the other hand, image segmentation based only on depth information sometimes fails due to depth image noise, and discard much useful appearance information. For example, despite its advantages, depth data produced by typical real-time stereo implementations is much noisier than the gray-scale or color imagery from which it is computed, and contains many pixels whose values have low confidence due to stereo matching ambiguity at textureless regions or

depth discontinuities.

Considering this, in the last few years a new approach to image segmentation has risen, which uses information about depth and geometric structure of the scene to enhance color image segmentation.

Harville and Robinson [5] use stereo data to augment local appearance features extracted from the images. In [6] and [7] the authors introduce a parallel segmentation of depth map and color image and then combine the two produced segmentations by merging color region that are related to the same depth region. In [8], Bleiweiss and Werman use Mean Shift over a 6D vector that fuses color and depth data obtained with a ZCam, while in [9] a similar approach is proposed using k-means clustering algorithm instead of mean-shift. Using such different sources of information makes unavoidable to deal with their different nature from both a physical and a semantic point of view. In [8] color and depth data are weighted with respect to their estimated reliability and in [9] a weighting costant for 3D position components is used.

The offline algorithm proposed in this thesis uses a different approach, where pairwise non linear similarities between macropixels are computed taking into account both color and depth information; a game theoretic approach is then employed to make them play in an evolutionary game [10] until stable segmentation emerges.

The thesis is structured as follows: chapter 2 will describe the most common general purpose approaches to image segmentation; in chapter 3 the principles of the proposed algorithm will be desribed while Appendix A will delve into implementation details; finally, results of experimental test will be shown in chapter 4.

Chapter 2

Image segmentation

Image segmentation is a task of fundamental importance in digital image analysis and it is a first step in many computer vision methods. It is the process that partitions a digital image into disjoint, nonoverlapping regions, each of which normally corresponds to something that humans can easily separate and view as an individual object. Unlike human vision, where image segmentation takes place without effort, computers have no means of intelligently recognising objects, therefore digital processing requires that we laboriously isolate the objects by breaking up the image into regions, one for each object [11, 12, 13]. The main goal of image segmentation is domain independent partitioning of an image into a set of disjoint regions that are visually different one from the other but internally homogeneous and meaningful with respect to some characteristics or computed properties, such as grey level, colour, texture, depth, motion, etc. aiming at simplify and/or change the representation of an image into something that is more meaningful with respect to a particular application and easier to analyze.

Commonly considered applications of segmentation include region-based image and video description, indexing and retrieval, video summarization, interactive region-based annotation schemes, detection of objects that can serve as cues for event recognition, region-based coding, etc.[14]

The idea of segmentation has its roots in work by the Gestalt psychologists, who studied the preferencies exhibited by human beings in grouping or organizing sets of shapes arranged in the visual field. Gestalt principles dictate certain grouping preferences based on features such as proximity, similarity, and continuity. [12]

Image segmentation is usually approached from one of two different but complementary perspectives: discontinuity and similarity. In the first category, the approach is to partition an image based on abrupt changes in intensity, such as edges in an image. The principal approches in the second category are based on

partitioning an image into regions that are similar according to a set of predefined criteria. Thresholding, region growing and region splitting and merging are examples of methods in this category. [15]

2.1 Detection of Discontinuities

Techniques belonging to this category are usually applied to grayscale digital image and try to segment it detecting discontinuities such as points, lines or, more frequently, edges. In general, discontinuities correspond to those points in an image where gray level changes sharply: such sharp changes usually occur at object boundaries. The most common way to look for discontinuities is to run a mask through the image; this procedure involves computing the of products of the coefficients with the gray levels contained in the region encompassed by the mask. Figure 2.1 presents some detection masks used to detect points (a), horizontal lines (b) or vertical lines (c).

-1	-1	-1
-1	8	-1
-1	-1	-1

(a) Point

-1	-1	-1
2	2	2
-1	-1	-1

(b) Horizontal line

-1	2	-1
-1	2	-1
-1	2	-1

(c) Vertical line

Figure 2.1: Examples of detection mask

Although point and line detection certinly are important in any discussion on segmentation, edge detection is by far the most common approach for detecting meaningful discontinuities in gray level.

There are many derivative operators designed for 2-D edge detection, most of which can be categorized as gradient-based or Laplacian-based methods. The gradient-based methods detect the edges by looking for the maximum in the first derivative of the image. The Laplacian-based methods search for zero-crossings in the second derivative of the image to find edges.

Gradient

First-order derivatives of a digital image are based on various approximations of the 2-D gradient. The gradient of an image $f(x, y)$ at location (x, y) is defined as the *vector*

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

Most operators perform a 2-D spatial gradient measurement using convolution with a pair of horizontal and vertical derivative kernels, i.e. each pixel in the image is convolved with two kernels, one estimating the gradient in the x direction and the other in th y direction. The most widely used derivative-based kernels for edge detection are the Roberts operators (Figure 2.2a), the Sobel opertors (Figure 2.2b) and the Prewitt operators (Figure 2.2c).

-1	0	0	-1
0	1	1	0

(a) Roberts

-1	-2	-1	-1	0	1
0	0	0	-2	0	2
1	2	1	-1	0	1

(b) Sobel

-1	-1	-1	-1	0	1
0	0	0	-1	0	1
1	1	1	-1	0	1

(c) Prewitt

Figure 2.2: Derivative-based operators

Laplacian

The Laplacian of a 2-D function $f(x, y)$ is a second-order derivative defined as

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

and can be implemented by any of the convolution kernels in Figure 2.3.

0	-1	0	-1	-1	-1	1	-2	1
-1	4	-1	-1	8	-1	-2	4	-2
0	-1	0	-1	-1	-1	1	-2	1

Figure 2.3: Examples of laplacian mask

The Laplacian has the advantage that it is an isotropic measure of the second derivative. The edge magnitude is independent of the orientation and can be

obtained by convolving the image with only one kernel. The presence of noise, however, imposes a requirement for a smoothing operation prior to using the Laplacian. Usually, a Gaussian filter is chosen for this purpose. Since convolution is associative, we can combine the Gaussian and Laplacian into a single Laplacian of Gaussian (LoG) kernel.

Canny Edge Detector

Generally, edge detection based on the aforementioned derivative-based operators is sensitive to noise. This is because computing the derivatives in the spatial domain corresponds to high-pass filtering in the frequency domain, thereby accentuating the noise. Furthermore, edge points determined by a simple thresholding of the edge map (e.g., the gradient magnitude image) is error-prone, since it assumes all the pixels above the threshold are on edges. When the threshold is low, more edge points will be detected, and the results become increasingly susceptible to noise. On the other hand, when the threshold is high, subtle edge points may be missed. These problems are addressed by the Canny edge detector, which uses an alternative way to look for and track local maxima in the edge map. The Canny operator is a multistage edge-detection algorithm. The image is first smoothed by convolving with a Gaussian kernel. Then a first-derivative operator (usually the Sobel operator) is applied to the smoothed image to obtain the spatial gradient measurements, and the pixels with gradient magnitudes that form local maxima in the gradient direction are determined. Because local gradient maxima produce ridges in the edge map, the algorithm then performs the so-called nonmaximum suppression by tracking along the top of these ridges and setting to zero all pixels that are not on the ridge top. The tracking process uses a dual-threshold mechanism, known as thresholding with hysteresis, to determine valid edge points and eliminate noise. The process starts at a point on a ridge higher than the upper threshold. Tracking then proceeds in both directions out from that point until the point on the ridge falls below the lower threshold. The underlying assumption is that important edges are along continuous paths in the image. The dual-threshold mechanism allows one to follow a faint section of a given edge and to discard those noisy pixels that do not form paths but nonetheless produce large gradient magnitudes. The result is a binary image where each pixel is labeled as either an edge point or a nonedge point. [16]

2.2 Region-Based Segmentation

Region segmentation methods partition an image by grouping similar pixels together into identified regions. Image content within a region should be uniform and homogeneous with respect to certain attributes, such as intensity, rate of change in intensity, color, and texture. Regions are important in interpreting an image because they typically correspond to objects or parts of objects in a scene. Most of the segmentation techniques belong to this category: region-growing, split-and-merge, clustering approach, threshold, etc... just to name a few.

Basic Formulation

Let R represent the entire image region; segmentation process partitions R into n subregions $R_1,\ R_2,\ ..., R_n$ such that

(a) $\bigcup_{i=1}^{n} R_i = R$

(b) R_i is a connected region, $i = 1, 2, ..., n$

(c) $R_i \cap R_j = \emptyset$ for all i and j, $i \neq j$

(d) $P(R_i) =$ TRUE for $i = 1, 2, ..., n$

(e) $P(R_i \cup R_j) =$ FALSE for $i \neq j$

where $P(R_i)$ is a logical predicate defined over the points in the set R_i. Condition (a) indicates that the segmentation must be complete; that is every pixel must be in a region. Condition (b) requires that points in a region must be connected, even though this condition is not considered by every segmentation algorithm. Condition (c) indicates that the regions must be disjoint. Condition (d) deals with the properties that must be satisfied by the pixel in a segmented region. Finally condition (e) indicates that regions R_i and R_j are different in the sense of predicate P. Item 5 of this definition can be modified to apply only to adjacent regions, as non-bordering regions may well have the same properties.

2.2.1 Thresholding

The simplest method of image segmentation is called thresholding method; this method is based on a threshold value to turn a gray-scale image into a binary image, labeling each pixel as background or foreground and it is particularly useful for scenes containing solid objects resting on a contrasting background. Thresholding can also be generalized to multivariate classification operations, in

which the threshold becomes a multidimensional discriminant function classifying pixels based on several image properties.

In the simplest implementation of thresholding, the value of the threshold gray level is held constant throughout the image (Figure 2.4). If the background gray level is reasonably constant over the image and if the objects all have approximately equal contrast above the background, then the gray-level histogram is bimodal, and a fixed global threshold usually works well, provided that the threshold, T, is properly selected, usually analyzing the gray-level histogram of the image (Figure 2.5).

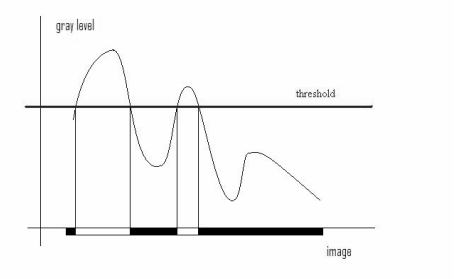


Figure 2.4: Threshold function

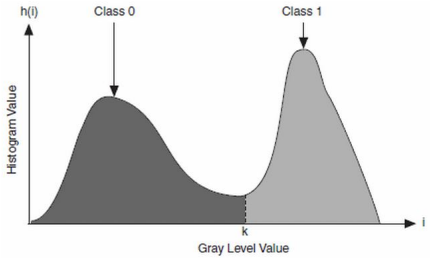


Figure 2.5: Gray-level histogram

Often, due to uneven illumination and other factors, the background gray level and the contrast between the objects and the background often vary within the image. In such cases, global thresholding is unlikely to produce satisfactory results, since a threshold that works well in one area of the image might work poorly in other areas. To cope with this variation, one can use an adaptive, or variable, threshold that is a slowly varying function of position in the image, i.e. it depends also on the spatial coordinates x and y. This type of threshold function is usually defined dynamic threshold or adaptive threshold. One approach to adaptive thresholding is to partition an $N \times N$ image into nonoverlapping blocks of $n \times n$ pixels each $(n < N)$, analyze gray-level histograms of each block, and then form a thresholding surface for the entire image by interpolating the resulting threshold values determined from the blocks.

2.2.2 Region-Growing

The fundamental limitation of histogram-based region segmentation methods, such as thresholding, is that the histograms describe only the distribution of gray levels without providing any spatial information. Region growing is an approach that exploits spatial context by grouping adjacent pixels or small regions together into larger regions. The basic approach is to start with with a set of "seed" points

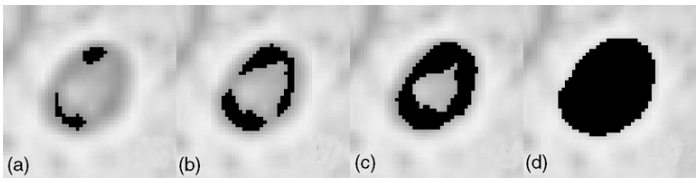


Figure 2.6: Example of region growing

and from these grow regions by appending to each seed those neighboring pixels that have properties similar to the seed, such as specific ranges of gray level, color or depth (see Figure 2.6). These criteria are local in nature and do not take into account the "history" of region growth. Therefore many region-growing alorithms uses additional criteria that increase their power such as the concept of size, likeness between a candidate pixel and the pixels grown so far, and the shape of the region being growing. The use of these types of descriptors is based on the assumption that a model of expected results is at least partially available.

2.2.3 Split-and-Merge

Opposite to the "bottom-up" approach of region growing, region splitting is a "top-down" operation. The basic idea of region splitting is to break the image into disjoint regions within which the pixels have similar properties.

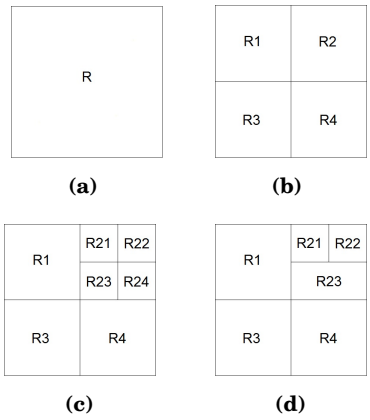


Figure 2.7: Example of region splting and merging

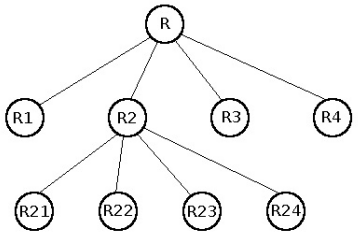


Figure 2.8: Quadtree corresponding to 2.7c

Region splitting usually starts with the whole image as a single initial region (Figure 2.7a). It first examines the region to decide if all pixels contained in it satisfy certain homogeneity criteria of image properties; if the criterion is met, then the region is considered homogeneous and hence left unmodified in the image.

Otherwise the region is split into subregions, and each of the subregions, in turn, is considered for further splitting (Figure 2.7b and Figure 2.7c). This recursive process continues until no further splitting occurs. This splitting technique has a convenient representation in the form of a so-called *quadtree* (Figure 2.8), a tree in which nodes have exactly four descendants, whose root corresponds to the entire image. After region splitting, the resulting segmentation may contain neighboring regions that have identical or similar image properties. Hence a merging process is used after each split to compare adjacent regions and merge them if necessary (Figure 2.7d)[17].

2.2.4 Clustering

Data clustering or cluster analysis is the organization of a collection of unlabeled data (usually represented as a vector of measurements, or a point in a multidimensional space) into clusters based on similarity. Intuitively, data within a valid cluster are more similar to each other than they are to a pattern belonging to a different cluster [18]. By considering pixels and their related features as unlabeled data points, it is possible to apply clustering alghoritms to image segmentation, thus exploiting the wide research in this field.

Besides the clustering algorithm itself, various image segmentation techniques based on clustering differ primarily on how the feature space is modeled. Color image segmentation often use a 3-dimensional space, corresponding to a color space, such RGB, CIELab or HSL, thus segmenting pixels depending only on color information. As thresholding, this approach may lead to unconnected components, breaking the rule (b) (see section 2.2), because it does not consider the continuity law. To avoid this, spatial information may be added to pixel description, leading to a 5-dimensional feature space (RGBXY) or even more if also motion or texture information are included. A drawback of this kind of feature spaces is the definition of the distance metric used by the clustering algorithms, that must take into account the different nature of this information.

Once feature space and distance metric have been fixed, a general purpose clustering algorithm may be applyed; the most frequent ones are k-means and mean shift [19].

K-means clustering

K-means clustering [20, 21] is a method of cluster analysis which aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean.

The algorithm is composed of the following steps:

1. Place K points into the space represented by the objects that are being clustered. These points represent initial group centroids.
2. Assign each object to the group that has the closest centroid.
3. When all objects have been assigned, recalculate the positions of the K centroids.
4. Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

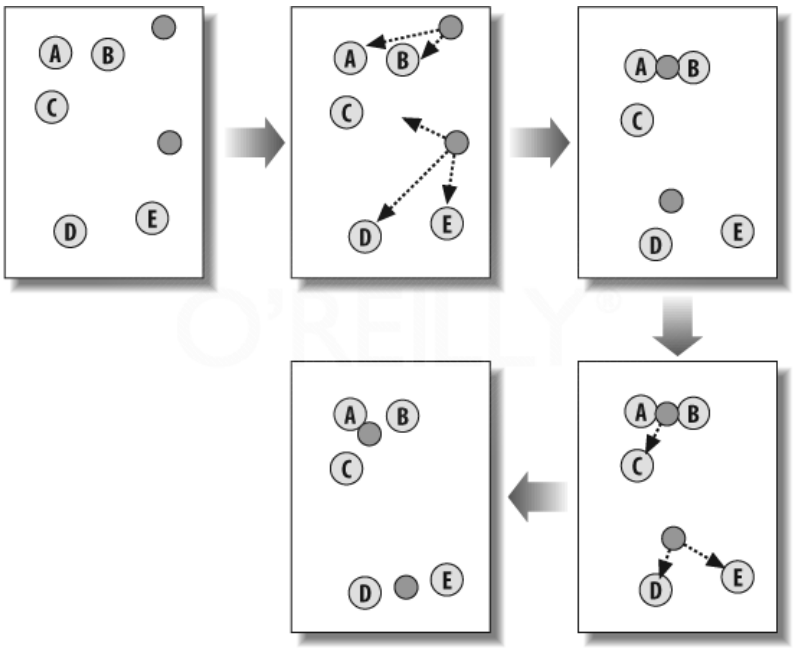


Figure 2.9: Example of k-means clustering

Figure 2.9 shows an example of k-means clustering with two clusters and five items. In the first frame, the two centroids (shown as dark circles) are placed randomly. Frame 2 shows that each of the items is assigned to the nearest centroid; in this case, A and B are assigned to the top centroid and C, D, and E are assigned to the bottom centroid. In the third frame, each centroid has been moved to the average location of the items that were assigned to it. When the assignments are calculated again, it turns out that C is now closer to the top centroid, while D and E remain closest to the bottom one. Thus, the final result is reached with A, B, and C in one cluster, and D and E in the other.

Although it can be proved that the procedure will always terminate, the k-means algorithm does not necessarily find the most optimal configuration. The

algorithm is also significantly sensitive to the initial randomly selected cluster centres. The k-means algorithm can be run multiple times to reduce this effect.

Mean shift

Mean shift was first proposed by Fukunaga and Hostetler [22], later adapted by Cheng [23] for the purpose of image analysis and extended by Comaniciu and Meer to low-level vision problems, including segmentation [24].

Mean shift clustering algorithm is a simple, nonparametric technique for estimation of the density gradient and it is based upon determining local modes in the joint spatio-feature space of an image, and clustering nearby pixels to these modes.

Mean shift uses a kernel function $K(x_i - x)$ to determine the weight of nearby points for re-estimation of the mean. Typically the Gaussian kernel on the distance to the current estimate is used, i.e. $K(x - x_i) = e^{\|x_i - x\|^2}$. Assuming $g(x) = -K'(x)$, the mean shift is calculated as

$$m(x) = \frac{\sum_{i=1}^{n} g\left(\frac{x-x_i}{h}\right) x_i}{\sum_{i=1}^{n} g\left(\frac{x-x_i}{h}\right)} - x$$

where h is the bandwith parameter. Using a gaussian kernel, the mean shift is

$$m(x) = \frac{\sum_{i=1}^{n} exp\left(-\frac{1}{2}\left\|\frac{x-x_i}{h}\right\|^2\right) x_i}{\sum_{i=1}^{n} exp\left(-\frac{1}{2}\left\|\frac{x-x_i}{h}\right\|^2\right)} - x$$

The segmentation algorithm proceeds as follows:

1. Associate a feature vector x_i to each pixel p_i of the image.
2. For each feature vector x_i, perform the *mean-shift procedure*:
   - Set $y_1 = x_i$.
   - Repeat $y_{i+1} = y_i + m(y_i)$ until convergence, i.e. while $|y_{i+1} - y_i| > \tau_1$. See Figure 2.10 as an example
   - Set x_i to the converged value of y.
3. Identify clusters of feature vectors by grouping all converged points that are closer than a prescribed threshold τ_2.
4. Assign labels to clusters.

Typically the spatial domain and the range domain are different in nature, so it is often desirable to employ separate bandwidth parameters for different

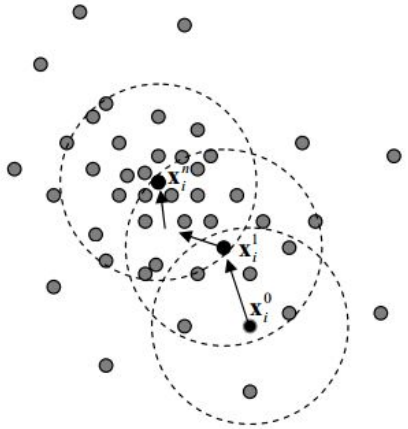


Figure 2.10: Mean shift procedure. Starting at data point x_i^0 run the mean shift procedure to find the stationary points of the density function. Superscripts denote the mean shift iteration while the dotted circles denote the density estimation windows.

components of the feature vector; usually a bandwith h_s for spatial components and h_c for color components.

One of the most important difference is that K-means makes two broad assumptions - the number of clusters is already known and the clusters are shaped spherically (or elliptically). Mean shift , being a non parametric algorithm, does not assume anything about number of clusters, because the number of modes gives the number of clusters. Also, since it is based on density estimation, it can handle arbitrarily shaped clusters. On the other hand K-means is fast and has a time complexity $O(knT)$ where k is the number of clusters, n is the number of points and T is the number of iterations, while classic mean shift is computationally expensive with a time complexity $O(Tn^2)$.

2.2.5 Graph Based

In graph based image segmentation methods, the image is modeled as a weighted, undirected graph $G = (V, E)$, where each node in V is associated with a pixel or a group of pixels of the image and edges in E connect certain pairs of neighbooring pixels. The weight $w(u, v)$ associated with the edge $(u, v) \in E$ describes the affinity (or the dissimilarity) between the two vertices u and v. The segmentation problem is then solved by partitioning the corresponding graph G, using efficient tools from graph theory, such that each partition is considered as an object segment in the image.

Among graph based algorithms, Normalized Cut by Shi and Malik [25] and

Efficient Graph-Based Image Segmentation by Felzenszwalb and Huttenlocher [26] are worth to be mentioned due to their effectiveness and low computational complexity.

Normalized Cut

The Normalized Cut algorithm was presented as an improvement of a previous graph based groupin algorithm, min cut [27]. A graph $G = (V, E)$ whose edge weights represent the similarity between linked vertices can be partitioned into two disjoint sets by removing edges connecting the two parts. Min cut algorithm computes the degree of dissimilarity between sets A and B as the total weigth of the edges that have been removed, which is called cut:

$$cut(A, B) = \sum_{u \in A,\ v \in B} w(u, v) \tag{2.1}$$

The optimal bipartitioning of a graph is the one that minimizes the cut value and it can be efficiently found using min-cut/max-flow algorithms. The minumum criteria favors cutting small sets of isolated nodes in the graph, as shown in Figure 2.11. This is a result from the definition of the cut cost in (2.1), which increases with the number of edges going across the two partitioned parts.

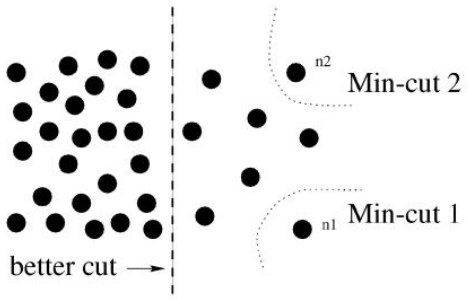


Figure 2.11: A case where minimum cut gives bad partitioning

To avoid this bias, normalized cut algorithms uses a different measure of dissimilarity between two groups by calculating the cut cost as a fraction of the total edge connections to all the nodes in the graph:

$$Ncut(A, B) = \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)}$$

where $assoc(A, V) = \sum_{u \in A,\ t \in V} w(u, t)$ is the total connection from nodes in A to all nodes in the graph.

Altought minimizing normalized cut exactly is NP-complete, it can be shown that find an approximation of the min cut is equivalent to solve the generalized

eigenvalue system $(\mathbf{D}-\mathbf{W})\mathbf{x} = \lambda\mathbf{D}\mathbf{x}$ with the second smallest eigenvalue ([] section 2). In this formula, $\mathbf{x}$ is an $N = |V|$ dimensional indicator vector, $x_i = 1$ if node i is in A and -1 otherwise; $\mathbf{D}$ is an $N \times N$ diagonal matrix with $\mathbf{d}(i) = \sum_j w(i,j)$; $\mathbf{W}$ is an $N \times N$ symmetrical matrix with $W(i,j) = w(i,j)$.

The grouping stage consists therefore of 4 steps:

1. Measuring the similarity of points pair-wisely
2. Applying eigendecomposition to get the eigenvector with the second smallest eigenvalue
3. Using the eigenvector to bipartition the graph
4. Deciding if further partition is necessary.

Effiecient Graph-Based Image Segmentation

The Efficient Graph-Based algorithm is based on minimum spanning tree (MST) and adaptively adjusts the segmentation criterion, therefore it can preserve detail in low-variability image regions while ignoring detail in high-variability regions.

Starting considering each vector as a component, it merges similar components using a predicate which is based on measuring the dissimilarity between elements along the boundary of the two components relative to a measure of the dissimilarity among neighboring elements within each of the two components. The *internal difference* of a component $C \subseteq V$ is defined as the largest weight in the minimum spanning tree of the component, $MST(C,E)$. That is,

$$Int(C) = \max_{e \in MST(C,E)} w(e)$$

The *differerence between* two components $C_1, C_2 \subseteq V$ is defined as the minimum weight edge connetting the two components. That is,

$$Dif(C_1,C_2) = \min_{v_1 \in C_1,\ v_j \in C_2,\ (v_i,v_j) \in E} w((v_i,w_j))$$

The region comparison predicate evaluates if there is evidence for a boundary between a pair or components by checking if the difference between the components, $Dif(C_1,C_2)$, is large relative to the internal difference within at least one of the components, $Int(C1)$ and $Int(C2)$. The pairwise comparison predicate is then defined as

$$D(C_1,C_2) = \begin{cases} true & \text{if } Dif(C_1,C_2) > MInt(C_1,C_2) \\ false & \text{otherwise} \end{cases}$$

where the minimum internal difference $MInt$ is defined as

$$MInt(C_1, C_2) = min(Int(C_1 + \tau(C_1), Int(C_2) + \tau(C_2))$$

τ is a threshold function based on the size of the component, $\tau(C) = k/|C|$ where $|C|$ denotes the size of C and k is a constant parameter. Using this threshold function makes harder to create small components because for small components a stronger evidence for a boundary is required.

The algorithm can be summarized as follows:

1. Sort E into $\pi = (o_1, \ldots, o_m)$, by non-decreasing edge weight.
2. Start with a segmentation S^0 where each vertex v_i is in its own component.
3. Repeat step 3 for $q = 1, \ldots, m$.
4. Construct S^q given S^{q-1} as follows. Let v_i and v_j denoting the vertices connected by the q-th edge in the ordering, i.e. $o_q = (v_i, v_j)$. If v_i and v_j are in disjoint components of S^{q-1} and $w(o_q)$ is small compared to the internal difference of both those components, then merge the two components otherwise do nothing. More formally, let C_i^{q-1} be the component of S^{q-1} containg v_i and S_j^{q-1} the component containing v_j. If $C_i^{q-1} \neq C_j^{q-1}$ and $w(o_q) \leq MInt(C_i^{q-1}, C_j^{q-1})$ then S^q is obtained from S^{q-1} by merging C_i^{q-1} and C_j^{q-1}. Otherwise $S^q = S^{q-1}$.
5. Return $S = S^m$.

Chapter 3

Proposed algorithm

The proposed algorithm uses information about geometry provided by range images to enhance color images segmentation. It can be mainly partitioned in three steps according to the pipeline shown in Figure 3.1:

1. The first step consist of a preprocessing stage which groups pixel into "superpixels" or "macropixels", local subsets of pixels with similar color and position. This is necessary to limit the computational complexity in the subsequent phase because the high number of pixels in images even at moderate resolutions would make later steps intractable.

2. The second step produces a compatibility matrix between each pair of macropixels using both photochromatic information provided by a color image and geometric information provided by the range image.

3. The last step groups macropixels into segments based on the compatibility matrix computed in the previous step. It uses game-theoretic evolutionary games provided by the DIAS - Dipartimento di scienze ambientali, informatica e statistica - of Università Ca' Foscari of Venice, which suggest which macropixels must be merged to create the segments of the image.

3.1 First phase: oversegmentation

Macropixels are created by using already implemented and well-known segmentation algorithms tuning the relative parameters to obtain an oversegmentation. The Efficient Graph Based (EG) technique proposed by Felzenszwalb and Huttenlocker in [26] and presented in section 2.2.5 on page 17 has been chosen due to the good results and the low complexity.

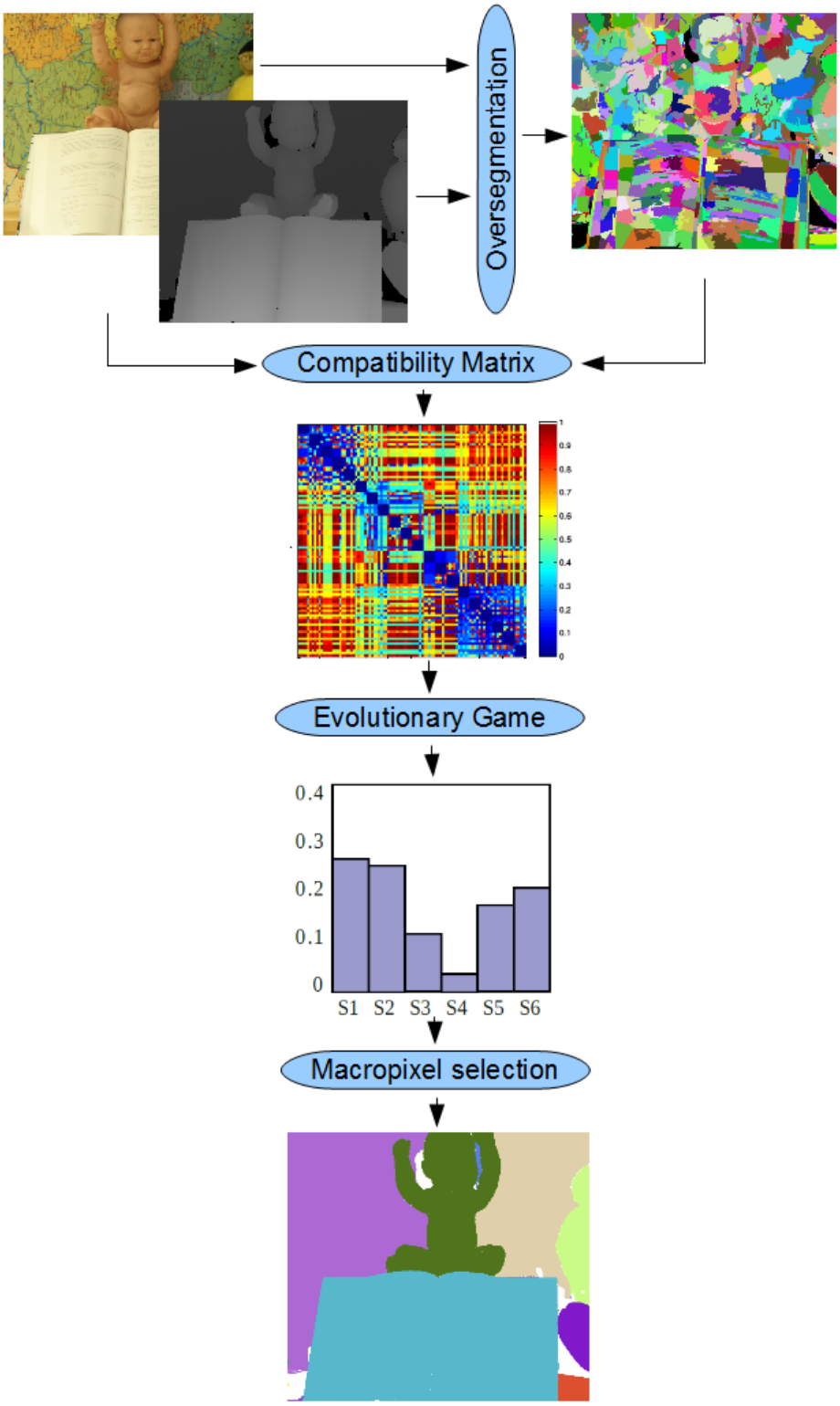


Figure 3.1: Pipeline of the proposed algorithm

However, this procedure has shown to be insufficient: as shown in Figure 3.2a, some produced macropixels contained pixels of different objects whenever close objects share similar color. This forced to include an additional image process by implementing a hierical photochromic-geometric oversegmentation: the procedure performs first a color-based segmentation using EG algorithm, subsequently for each produced macropixel, sample mean and standard deviation are calculated with respect to the depth of the pixels. If standard deviation exceeds a threshold σ_t the macropixel is further subdivided applying k-means with $k = 2$. The subsegments are recursively examined and splitted until standard deviation meets threshold (Figure 3.2b). This splitting procedure does not ensure region continuity, therefore 8-connettivity is finally forced by grouping neighbouring pixels (Figure 3.2c).

(a) **(b)** **(c)**

Figure 3.2: Oversegmentation detail at different stages: (a) wrong macropixel produced by color-based segmentation, (b) subsegments after splitting using kmeans with $k = 2$, (c) final correct macropixels

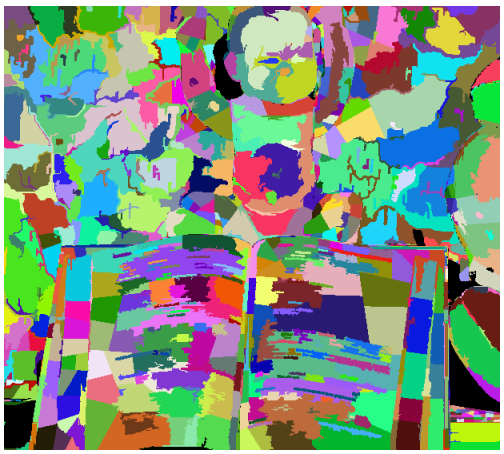


Figure 3.3: Output of first phase

The result of first phase is shown in Figure 3.3.

3.2 Second phase: compatibility computation

In order to calculate the compatibility between two macropixels, color distance and geometric/spatial distance are indipendently computed. The compatibility measure is then obtained by combining the two distances.

The color distance is simple defined as the distance between the average colors of macropixels on the UV plane of the YUV color space. The luminance component is not considered to make the measure insensitive to illumination variations. The average color of each macropixel is computed averaging the chroma components of the pixels that belong to it. Given two macropixels m_i and m_j respectively with chroma coordinates $[u_i\ v_i]$ and $[u_j\ v_j]$, the color distance is then computed as

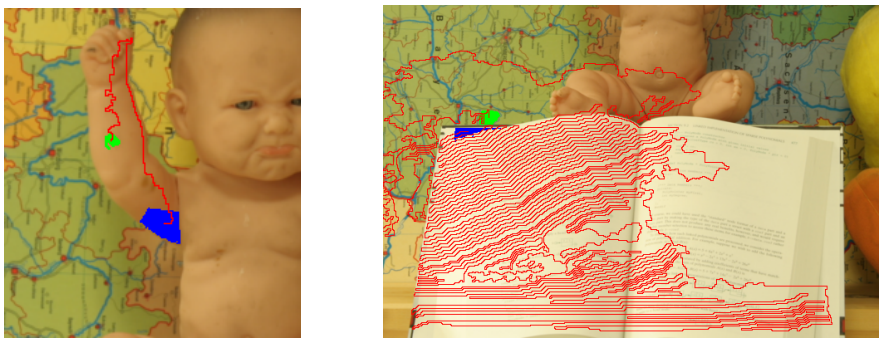
$$d_c(m_i, m_j) = \sqrt{(u_i - u_j)^2 + (v_i - v_j)^2}$$

The definition of the geometric distance is instead more elaborate. It is related to the path between macropixels in a 4-conntected graph representing the image where each node corresponds to a pixel and is connected to the 4 adjacent pixels, while the weigth of the edges is related to the distance between the corresponding point of the range image. If two macropixels belong to the same uninterrupted surface, it can be expexted that there is a path connecting the macropixels that contains only small jumps, because it rings around abrupt discontinuities. In order to find such a path, a modified version of the Dijkstra algorithm has been implemented where the most convinient step is not the one that shortens the total trip to the destination, but the one that perfoms the smallest drop over depth values. The path obtained through this algorithm is therefore one of the paths with the "smallest maximum drop" and it is generally different from the shortest path, as shown in Figure 3.4. In particular, in Figure 3.4b it can be observed how the route between the two macropixels goes through the baby to minimize the maximum jump.

The algorithm is performed once for each macropixel, finding a path from its centroid to all other pixels. For every other macropixel, the path p to its centroid is selected and the geometric distance between them is set to the maximum jump of the path, i.e.

$$d_z(m_i, m_j) = \begin{cases} \min_{p \in P_{m_i \to m_j}} \max_{(k,l) \in p} w(k,l) \\ \infty \quad \text{if } P_{m_i \to m_j} = \emptyset \end{cases}$$

where $P_{m_i \to m_j}$ represents the set of all the paths from m_i to m_j. The distance is automatically set to infinity if there is no path connecting the macropixels. The meaning of this defition of d_z is that there is no way to get from the centroid of m_i to the centroid of m_j without a drop of at least d_z.



(a) **(b)**

Figure 3.4: Smallest maximum drop paths

Four different measures of distance between pixels have been implemented: given two adjenct pixels A and B with 3D coordinates respectively (x_A, y_A, z_A) and (x_B, y_B, z_B) their distance (which corresponds to the weigth of the edge connecting the corresponding nodes) can be computed as

1. 3D euclidean distance $d_{euclid}(A, B) = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2 + (z_a - z_b)^2}$
2. displacement over z-coordinates $d_{delta-z}(A, B) = |z_a - z_b|$
3. ratio between z-component of the distance and the overall distance

$$d_{norm}(A, B) = \frac{|z_a - z_b|}{\sqrt{(x_a - x_b)^2 + (y_a - y_b)^2 + (z_a - z_b)^2}}$$

4. angle between the direction of the jump and the plane of the image

$$d_{angle}(A, B) = \frac{\arcsin(d_{norm}(A, B))}{\pi/2}$$

   $d_{angle} = 1$ means that the jump is orthogonal plane of the image, while $d_{angle} = 0$ means that the jump is parallel to the plane.

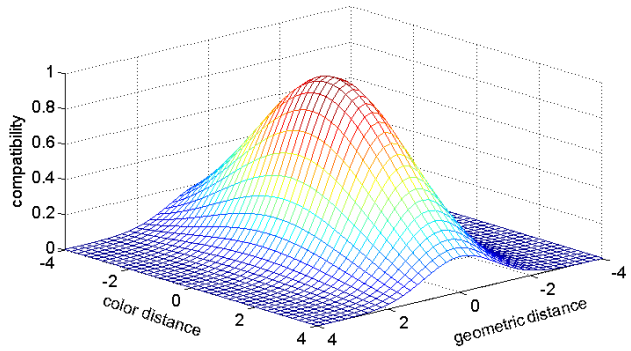
Measures 1 and 2 have thus the same unit of measurement of the range image, while 3 and 4 are dimensioneless and limited in the interval $[0, 1]$. In addiction, measure 3 and 4 can be considered normalized measure, as they are not affected by the distance from the image plane and are thus expected to avoid bias toward foreground objects.

Once both distances have been computed, the compatibility matrix is calculated by mixing them. Each distance is modeled as a gaussian indipendent event with zero mean and standard deviation σ_z and σ_c respectively. The compatibility

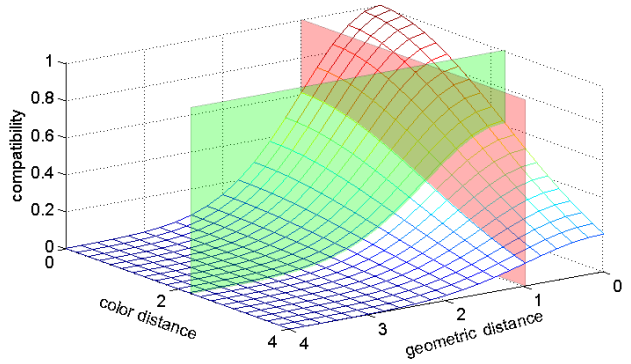
measure results therefore in a bivariate gaussian function as shown in Figure 3.5 and the elements of the compatibility matrix Π are computed as

$$\pi_{i,j} = \pi(m_i, m_j) = e^{-\frac{1}{2}\left(\frac{d_z(m_i,m_j)^2}{\sigma_z^2} + \frac{d_c(m_i,m_j)^2}{\sigma_c^2}\right)} \tag{3.1}$$

The parameters σ_z and σ_c are used to set the selectivity of compatibility measure.



(a)



(b)

Figure 3.5: (a) Bivariate gaussian model of compatibility measure with $\sigma_z^2 = 1$ and $\sigma_c^2 = 5$. (b) Bivariate gaussian for non-negative distance values. Red plane represent $\sigma_z^2 = 1$, green plane represents $\sigma_c^2 = 5$

3.3 Third phase: clustering

The task of third phase is to group macropixels into segments which hopefully correspond to image objects. It uses a C++ library developed and provided by the Department of Computer Science of Università Ca' Foscari of Venice, which has

already been applied to different fields of computer vision, especially matching and grouping [28, 29, 30]. It provides an Evolutionary Game-Theory framework [10], an application of Game-Theory to evolving population. Originated in the early 40's, Game Theory was an attempt to formalize a system characterized by the actions of entities with competing objectives, which is thus hard to characterized with a single objective function. According to this view, the emphasis shifts from the search of a local optimum to the definition of equilibria between opposing forces, providing an abstract theoretically-founded framework to model complex interactions. In this setting multiple player have at their disposal a set of strategies and their goal is to maximize a payoff that depends on the strategies adopted by the other players. Evolutionay Game-Theory differs from standard Game-Theory in that each player is not supposed to behave rationally or have a complete knowledge of the details of the game, but he acts according to a pre-programmed pure strategy.

Specifically, the library provides a simmetric two player game, i.e. game between two players that have the same set of available strategies and that receive the same payoff when playing against the same strategy. More formally, let $O = \{1, \dots, n\}$ be a set of available strategies (*pure strategies* in the language of Game-Theory) and $C = (c_{ij})$ be a matrix specifying the payoffs, then an individual playing strategy i against someone playing strategy j will receive a payoff c_{ij}.

The amount of population that plays each strategy at a given time is expressed through the probability distribution $\mathbf{x} = (x_1, \dots, x_n)^T$ called *mixed strategy* with $\mathbf{x} \in \Delta^n = \{\mathbf{x} \in \mathbb{R}^n : \forall i \; x_i \geq 0, \sum_{i=1}^n x_i = 1\}$.

The *support* $\sigma(\mathbf{x})$ of a mixed strategy $\mathbf{x}$ is defined as the set of elements chosen with non-zero probability: $\sigma(\mathbf{x}) = \{i \in O | x_i > 0\}$. The expected payoff received by a player choosing element i when playing against a player adopting a mixed strategy $\mathbf{x}$ is $(C\mathbf{x})_i = \sum_j c_{ij} x_j$, hence the expected payoff received by adopting the mixed strategy $\mathbf{y}$ against $\mathbf{x}$ is $\mathbf{y}^T C \mathbf{x}$. The *best replies* against mixed strategy $\mathbf{x}$ is the set of mixed strategies

$$\beta(\mathbf{x}) = \left\{ \mathbf{y} \in \Delta | \mathbf{x}^T C \mathbf{x} = \max_{\mathbf{z}} (\mathbf{z}^T C \mathbf{x}) \right\}$$

A mixed strategy $\mathbf{x}$ is said to be a *Nash equilibrium* if it is the best reply to itself, i.e. $\forall \mathbf{y} \in \Delta, \mathbf{x}^T C x \geq \mathbf{y}^T C \mathbf{x}$. This implies that $\forall i \in \sigma(\mathbf{x}) \; (C\mathbf{x})_i = \mathbf{x}^T C \mathbf{x}$, that is, the payoff of every strategy in the support of $\mathbf{x}$ is constant. The idea underpenning the concept of Nash equilibrium is that a rational player will consider a strategy viable only if no player has an incentive to deviate from it.

Finally, $\mathbf{x}$ is called an *evolutionary stable strategy* (ESS) if it is a Nash equilibrium and $\forall \mathbf{y} \in \Delta, \; \mathbf{x}^T C \mathbf{x} = \mathbf{y}^T C \mathbf{x} \Rightarrow \mathbf{x}^T C \mathbf{y} > \mathbf{y} C \mathbf{y}$. This condition guarantees that any deviation from the stable strategies does not pay, i.e. ESS's are strategies such that any small deviation from them will lead to an inferior payoff. The search for a

stable state is performed by simulating the evolution of a natural selection process. Specifically, the dynamics choosen are the replicator dynamics [31], a well-known formalization of the selection process governed by the following equation

$$x_i(t+1) = x_i(t)\frac{(C\mathbf{x}(t))_i}{\mathbf{x}(t)^T C\mathbf{x}(t)}$$

where x_i is the i-th element of the population and C the payoff matrix.

Π

	m1	m2	m3	m4	m5	m6
m1	0	1	0.1	0.1	0.7	0.9
m2	1	0	0	0.1	0.7	0.9
m3	0.1	0	0	0	0.6	0.4
m4	0.1	0.1	0	0	0	0.1
m5	0.7	0.7	0.6	0	0	0
m6	0.9	0.9	0.4	0.1	0	0

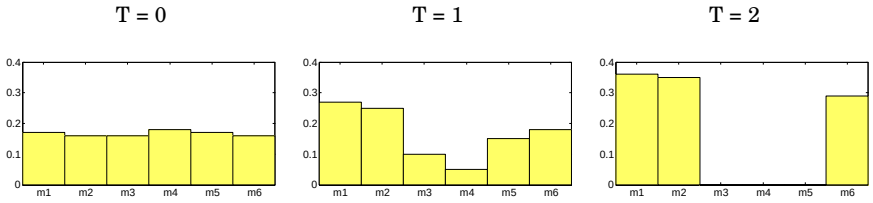


Figure 3.6: Example of evolution: given a payoff matrix Π , the population is initially (T = 0) set to the barycenter of the simplex; as expected, after just one iteration of the replicator dynamics (T = 1) the most consistent strategies (*m1* and *m2*) obtain a clear advantage. Finally, after ten iteration (T = 10) strategies *m1*, *m2* and *m6* have prevailed and other strategies have no more support.

In the contest of the proposed algorithm, each *pure strategy* represents a macropixel and the payoff of strategies i and j corresponds to the compability between macropixel m_i and macropixel m_j, i.e. the compatibility matrix $\Pi = (\pi(i,j))$ computed during the previous phase is used as payoff matrix. The evolutionary game is then started and when the population reaches an equilibrium, all the non-extincted pure strategies (i.e. $\sigma(\mathbf{x})$) are considered selected by the game and thus the associated macropixels are merged into a single segment. After each game the selected macropixel are removed from the population and the selection process is iterated until all the segments have been merged.

This procedure does not ensure connectivity of the generated segments; in fact, the produced segments are sometimes made up of several unconnected components (as in Figure 3.7f) especially at later iterations, when best segments have already

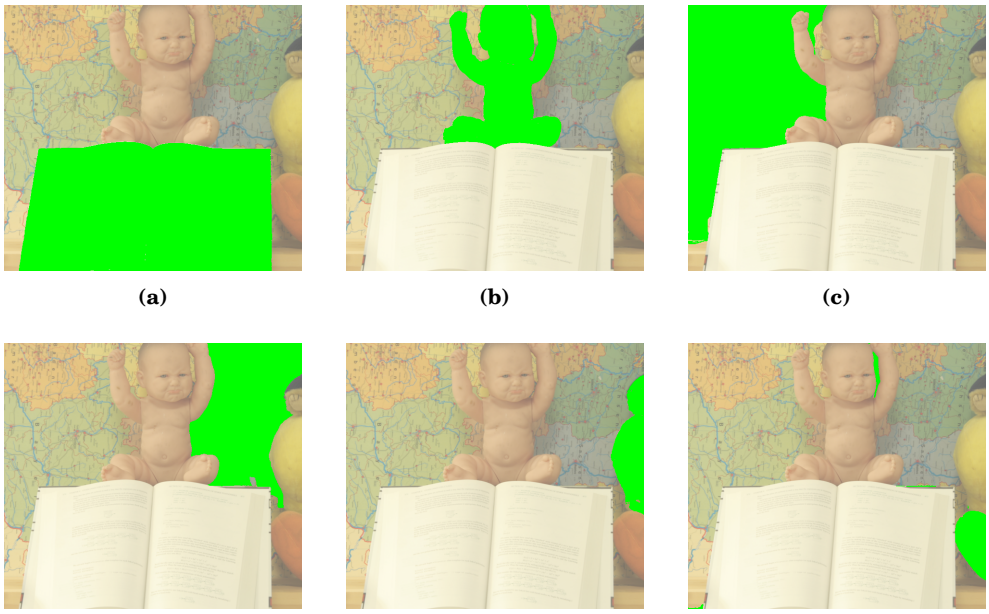


Figure 3.7: Sequence of segments produced by each iteration of evolutionary game

been created. The last step of proposed algorithm has then the task to check the connettivity of segments and eventually split unconnected components, as done at the end of first phase (section 3.1).

An example of final segmented image is shown in Figure 3.8.

Figure 3.8: Example of segmented image produced by the proposed algorithm

Chapter 4

Experimental results

According to taxonomy proposed in [32], the evaluation method used can be classified as groundtruth based, i.e. it attemps to measure the difference between the machine segmentation result and an expected ideal segmentation, specified manually.

Three batches of tests have been carried out: the first one performs a research in the parameters space in order to find the best configuration; the second one compares the resulting segmentation when the groundtruth depth map is replaced with depth maps obtained through stereo algorithms; the last one compares the results of the proposed algorithm with other segmentation algorithms.

4.1 Image dataset

All the test have been performed using images part of Middlebury's 2006 Stereo dataset [2], a large set of stereo pairs associated with a ground truth disparity map, obtained through structured light technique [33]. The reasons why these images have been preferred is the high quality and accurate disparity maps and the limited number of objects in the scenes, which limits the number of segments to be generated. A subset of 7 scenes has been selected to be used in tests, and for each subject a manual segmentation was performed and used as groundtruth. The scenes selected are shown in Figure 4.1; following Middlebury naming, they are: Baby1, Baby2, Bowling1, Bowling2, Lampshade2, Midd1, Midd2.

Disparity maps contain occluded points, i.e. points whose disparity is unknown, which are labeled as black pixels both in second and third column images in Figure 4.1; due to this lack of information, they are not considered by the segmentation alghoritm.

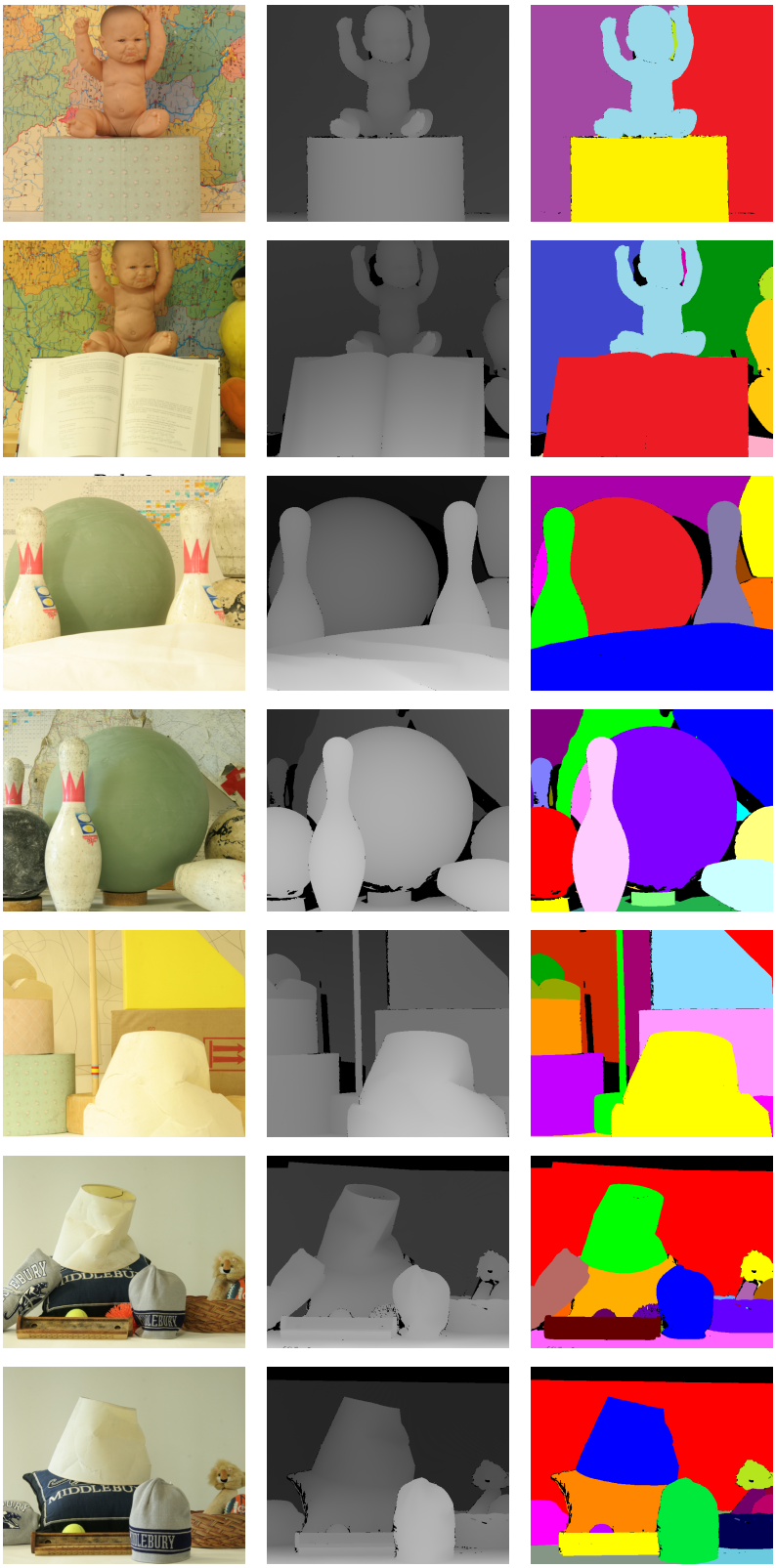


Figure 4.1: Image dataset: first column contains left color image; second column contains left disparity map provided as ground-truth by Middlebury; third column contains manual segmentation

4.2 Evaluation metrics

Following [34] and [35] six different evaluation measures have been implemented and computed for each automatic segmentation. All indexes express a measure of dissimilarity between the automatic segmentation and the manual ground-truth and can be easily computed using a contingency table, also called confusion matrix or association matrix. Considering two segmentation $S_1 = \{R_1^1, R_1^2, \ldots, R_1^k\}$ and $S_2 = \{R_2^1, R_2^2, \ldots, R_2^l\}$ of the same image of n pixels, contingency table is a $k \times l$ matrix whose ijth element m_{ij} represents the number of points in the intersection of R_i^1 of S_1 and R_j^2 of S_2, that is $m_{ij} = |R_i^1 \cap R_j^2|$.

4.2.1 Hamming distance

The first metric uses Hamming Distance to measure the overall region-based difference between segmentation results and groundtruth [36]. Given the segmentations S_1 and S_2 the *Directional Hamming Distance* is defined as

$$D_H(S_1 \Rightarrow S_2) = \sum_i |R_2^i \backslash R_1^{i_t}|$$

where $\backslash$ is the set difference operator, $|x|$ is the number of pixel in set x and $i_t = \max_k |R_2^i \cap R_1^k|$, that is $R_1^{i_t}$ is the best corresponding segment in S_1 of segment $R_2^i \in S_2$.

Directional Hamming Distance can be used to define the missing rate R_m and the false alarm rate R_f as

$$R_m(S_1, S_2) = \frac{D_H(S_1 \Rightarrow S_2}{n}) \qquad R_f(S_1, S_2) = \frac{D_H(S_2 \Rightarrow S_1)}{n}$$

The *Hamming Distance* is then defined as the average of missing rate and false alarm rate:

$$HD(S_1, S_2) = \frac{1}{2}(R_m(S_1, S_2) + R_f(S_1, S_2)) = \frac{1}{2n}(D_H(S_1 \Rightarrow S_2) + D_H(S_2 \Rightarrow S_1))$$

4.2.2 Consistency Error: GCE and LCE

The second metric, Consistency Error, tries to account for nested, hierarchical similarities and differences in segmentations [37]. Based on the theory that human's perceptual organization imposes a hierarchical tree structure on objects, consistency error metric does not penalize differences in hierarchical granularity, i.e. it is tolerant to refinement.

Let $R(S, p_i)$ be the set of pixels corresponding to the region in segmentation S that contains the pixel p_i. Then, the *local refinement error* associated with p_i is

$$E(S_1, S_2) = \frac{|R(S_1, p_i \backslash R(S_2, p_i)|}{|R(S_1, p_i|}$$

Given the refinement error for each pixel, two metrics are defined for the entire image, *Global Consistency Error* (GCE) and *Local Consistency Error* (LCE), as follows:

$$GCE(S_1, S_2) = \frac{1}{n} \min \left\{ \sum_{\text{all pixels } p_i} E(S_1, S_2, p_i), \sum_{\text{all pixels } p_i} E(S_2, S_1, p_i) \} \right\}$$

$$LCE(S_1, S_2) = \frac{1}{n} \sum_{\text{all pixels } p_i} \min \{ \{ E(S_1, S_2, p_i), E(S_2, S_1, p_i) \} \}$$

The difference between them is that GCE forces all local refinement to be in the same direction, while LCE allows refinement in different directions in different regions of the image. Both measures are tolerant to refinement: in the extreme case, a segmentation containing a single region or a segmentation consisting of regions of a single pixels, GCE = LCE = 0.

4.2.3 Clustering indexes: Rand, Fowlkes and Jaccard

As stated in subsection 2.2.4, image segmentation problem can be modeled as a clustering problem by considering the pixels of the image as the set of objects to be grouped and then considering each cluster produced as a segment. In the same way, the difference between two segmentation can be quantified using metrics proper of clustering evaluation: the last three metrics implemented are borrowed from cluster analysis.

Given two clusterings C_1 and C_2 and a set $O = o_1, \ldots, o_n$ of objects, all pairs of objects $(o_i, o_j) \in O \times O,\ i \neq j$ are considered. A pair (o_i, o_J) falls into one of the four categories:

N11: in the same cluster under both C_1 and C_2

N00: in different clusters under both C_1 and C_2

N10: in the same cluster under C_1 but not C_2

N01: in the same cluster under C_2 but not C_1

The number of pairs in each category can be efficiently calculated using the contingency table:

$$N_{11} = \frac{1}{2} \left[\sum_i \sum_j m_{ij}^2 - n \right],$$

$$N_{00} = \frac{1}{2} \left[n^2 - \sum_{c_i \in C_1} |c_i|^2 - \sum_{c_j \in C_2} |c_j|^2 + \sum_i \sum_j m_{ij}^2 \right],$$

$$N_{10} = \frac{1}{2}\left[\sum_{c_i \in C_1} |c_i|^2 \sum_i \sum_j m_{ij}^2\right],$$

$$N_{01} = \frac{1}{2}\left[\sum_{c_j \in C_2} |c_j|^2 - \sum_i \sum_j m_{ij}^2\right].$$

The proposed metrics are:

Rand [38]

$$\mathcal{R}(C_1, C_2) = 1 - \frac{N_{11} + N_{00}}{n(n-1)/2}$$

Fowlkes [39]

$$\mathcal{F}(C_1, C_2) = 1 - \sqrt{W_1(C_1, C_2), W_2(C_1, C_2)}$$

$$\text{where} \quad W_1(C_1, C_2) = \frac{N_{11}}{\sum_{c_i \in C_1} |c_i|(|c_i|-1)/2}$$

$$\text{and} \quad W_1(C_1, C_2) = \frac{N_{11}}{\sum_{c_j \in C_2} |c_j|(|c_j|-1)/2}$$

Jaccard

$$\mathcal{J}(C_1, C_2) = 1 - \frac{N_{11}}{N_{11} + N_{10} + N_{01}}$$

It is easy to see that these three indices are all distance measures with a value domain $[0, 1]$. The value is zero if and only if the two clusterings are the same except for possibly assigning different names to the individual clusters, or listing the clusters in different order.

4.3 Parameter tuning

The proposed algorithm exposes seven parameters, four in the oversegmentation phase (section 3.1) and three in the compatibility computation phase (section 3.2):

- σ_{EG} : width of the gaussian filter used in the preprocess of EG algorithm to smooth the image and remove noise;
- k : parameter of EG algorithm which controls how coarsely or finely an image is segmented (see threshold function τ on page 17);
- min : minimum area of macropixel, forced in postprocess step of EG;
- σ_t : threshold of standard deviation of depth values inside a macropixel;
- distance type : the formula used to compute the distance between adjacent pixels (euclid, delta, norm or angle);

- σ_z : standard deviation of geometric distance (see Equation 3.1 on page 24);
- σ_c : standard deviation of color distance;

Parameters of first phase have been empirical set to $\sigma_{EG} = 0.5,\ k = 500,\ min = 20$ and $\sigma_t = 3$. These values showed to provide a good quality oversegmented image, whose macropixels do not overlap between different objects/belong to only one object, without producing too many macropixels (by way of comparison, segmentation presented in Figure 1.1c used $k = 3000$ and $min = 200$). The average number of macropixels is 1316, which has been considered a good tradeoff between accuracy and computational complexity. Distance type, σ_z and σ_c have been used as variable parameters instead.

For each of the four distances, many couples of σ_z and σ_c values have been tested in order to find the configuration which minimizes the evaluation metrics presented in section 4.2; in particular, segmentation quality has been assesed depending firstly on Hamming Distance and Jaccard Index.

Each configuration has been tested on the 7 scenes of the image dataset; the resulting segmentations have been evaluated and the average over the 7 scenes has been calculated. Figure 4.2 summarizes average results of Hamming Distance (first column) and Jaccard Index (second column).

All plots present similar structure, a large blue (i.e. good) area for mid values of σ_z and high values of σ_c, surrounded by a yellow/red (i.e. less good) area. This means that all distances have similar reaction to variation of parameters: if σ_z falls into the optimal interval, which varies slightly among different distances, and σ_c is greater than a threshold, segmentation quality remains almost stable. Specifically, in this image dataset, information about depth is more relevant than color information; in fact, even the extreme case $\sigma_C = \infty$ (which corresponds to not consider color information) leads to good quality segmentations, provided that σ_z is in the optimal interval. In Figure 4.3 the qualitative effects of σ_z and σ_c are shown. When the two parameters are both too low (case e) the grouping phase becomes too selective and a lot of erroneous clusters are produced, providing an oversegmentation; on the other hand, if they are both too high (case c) the process is unable to separate each region, producing an undersegmentation. If σ_c is low and σ_z is high (case d), the image is oversegemented with respect to color values (see, for example, the map details on the background); on the other hand, if σ_c is high but σ_z is low (case a), the oversegmentation depends on depth values (such as the foreground book). Finally, when parameters are choosen from the optimal blue region (case b), the resulting segmentation is really close to the ground truth.

Table 4.1 contains the three best configuration for each distance type with respect to Hamming Distance (a) and/or Jaccard Index (b). The best result for

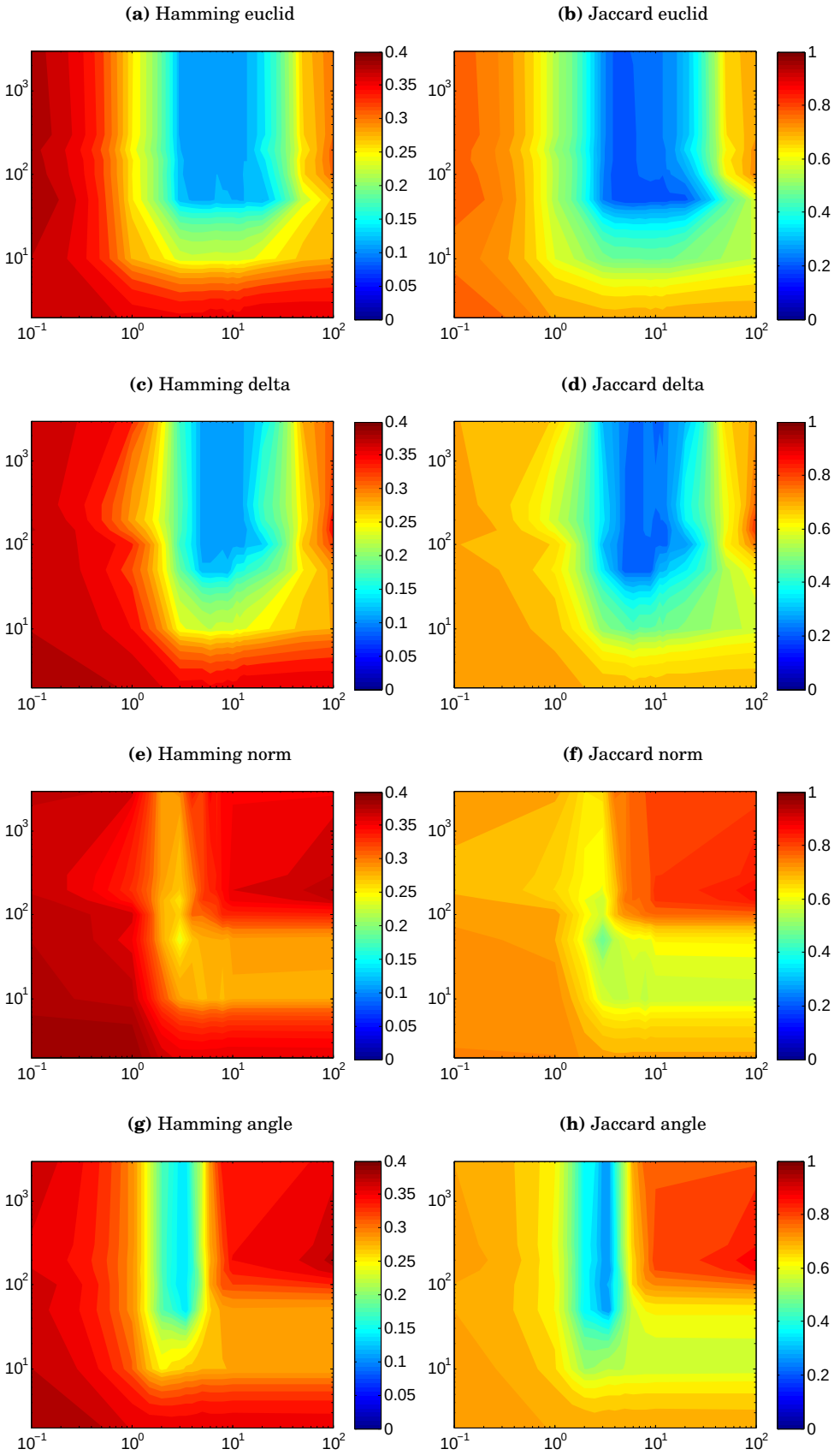


Figure 4.2: Exploration of parameter space σ_z (x-axis) and σ_c (y-axis)

each distance is represented also in Figure 4.4 (average best result).

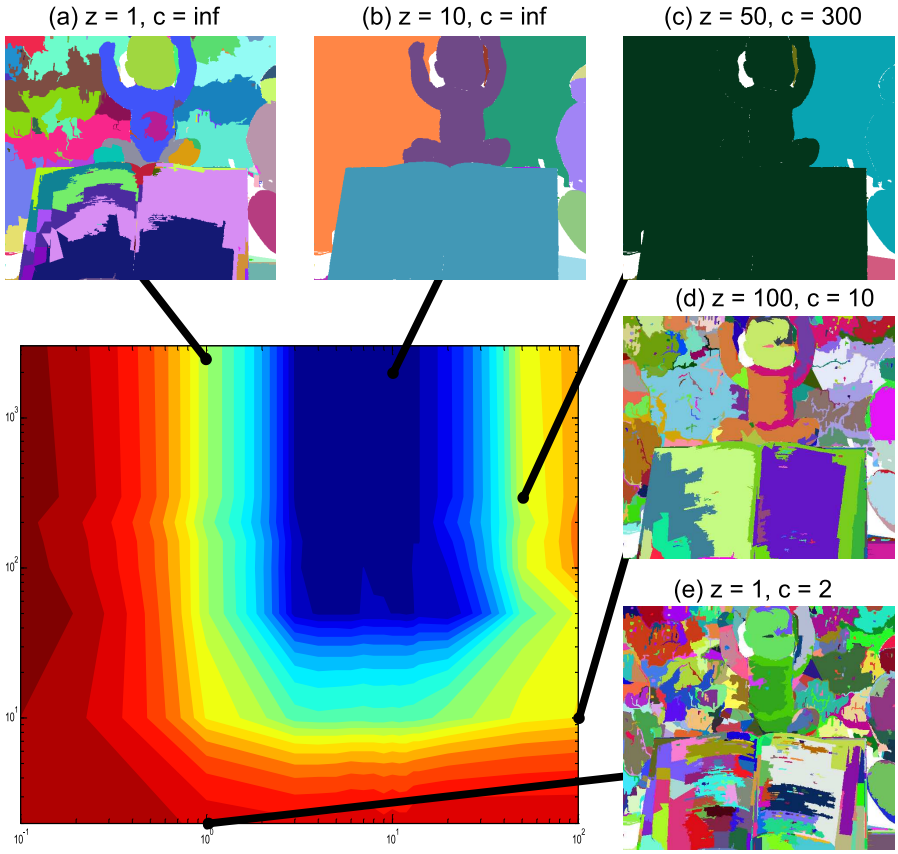
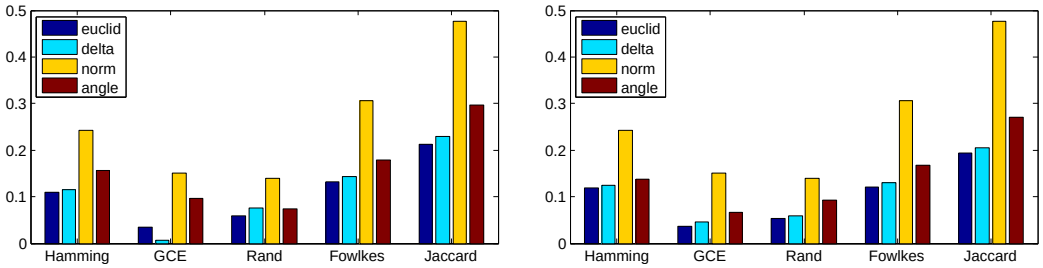


Figure 4.3: Qualitative effects of σ_z and σ_c



(a) according to Hamming Distance

(b) according to Jaccard Index

Figure 4.4: Comparison of best results for each distance type

Contrary to expectations, non-normalized measures - euclid and delta - achieve better results than the promising angle distance. This is probably due to the fact that the choosen images have similar structure (objects of different scenes are at

(a) Hamming Distance

distance type	σ_z	σ_c	Hamming	GCE	LCE	Rand	Fowlkes	Jaccard
euclid	4	inf	0.10991	0.03512	0	0.05958	0.13123	0.21222
euclid	4	200	0.10997	0.03540	0	0.05961	0.13135	0.21240
euclid	4	300	0.10997	0.03540	0	0.05960	0.13135	0.21240
delta	12	inf	0.11481	0.00662	0	0.07603	0.14354	0.22900
delta	6	200	0.11531	0.00831	0	0.07400	0.14170	0.22701
delta	6	150	0.11545	0.00943	0	0.07322	0.14138	0.22637
norm	3	50	0.24336	0.15026	0	0.13888	0.30689	0.47670
norm	3	150	0.25682	0.09227	0	0.30538	0.38622	0.57282
norm	3	100	0.26824	0.11125	0	0.26364	0.38875	0.58022
angle	3.5	50	0.13824	0.06693	0	0.09334	0.16830	0.27086
angle	3.5	150	0.14102	0.04002	0	0.08306	0.16942	0.27865
angle	3	100	0.14215	0.05278	0	0.07824	0.17233	0.28585

(b) Jaccard Index

distance type	σ_z	σ_c	Hamming	GCE	LCE	Rand	Fowlkes	Jaccard
euclid	6	50	0.11949	0.03631	0	0.05341	0.12111	0.19383
euclid	8	50	0.12443	0.04593	0	0.05688	0.12561	0.19870
euclid	7	50	0.12510	0.04577	0	0.05685	0.12561	0.19879
delta	9	50	0.12430	0.04597	0	0.05857	0.12978	0.20590
delta	7	50	0.12649	0.04609	0	0.05952	0.131720	0.21000
delta	8	50	0.12674	0.04584	0	0.05967	0.13184	0.21019
norm	3	50	0.24336	0.15026	0	0.13888	0.30689	0.47670
norm	8	10	0.27503	0.15027	0	0.13888	0.30690	0.47669
norm	5	10	0.27529	0.03966	0	0.12886	0.36710	0.56609
angle	3.5	50	0.13824	0.06693	0	0.09334	0.16830	0.27086
angle	3.5	150	0.14102	0.04002	0	0.08306	0.16942	0.27865
angle	3.5	200	0.14250	0.04325	0	0.08476	0.17250	0.28254

Table 4.1: Best configuration with respect to Hamming Distance (a) and/or Jaccard Index (b)

the same distance from camera), therefore the normalization introduced by the angle measure is unnecessary, while, at the same time, parameters of euclid and delta measures are probably overfitted to the image dataset.

4.4 Segmentation in association with stereo algorithms

The aim of the second batch of experiments is to examine the effect of replacing the ground-truth disparity map associated with each scene of the image dataset with a disparity map generated through a dense stereo algorithm. For each scene Graph Cut and Semi-Global Block Matching stereo algorithm have been performed and the disparity maps obtained have been used as inputs of segmentation algorithms.

In order to properly evaluate the resulting segmentations, both stereo algorithms output and evaluation process have been slightly modified. Firstly, every pixel labeled as occluded point in the ground truth (see third column of Figure 4.1) has been set occluded also in the disparity maps produced by stereo algorithm. This means that the resulting set of occluded points is the union of occluded points in groundtruth and occluded points in stereo output. If the stereo algorithm produced erroneous occluded points, these have been treated as occluded points by segmentation algorithm, while in the evaluation process occluded points of groundtruth are not considered and erroneous occluded points are treated as an additional cluster by evaluation process, resulting therefore in a penalty.

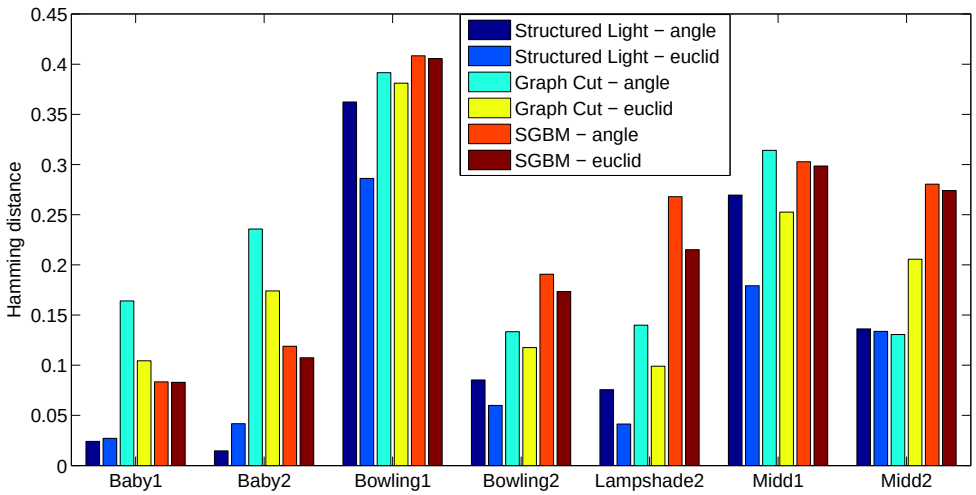
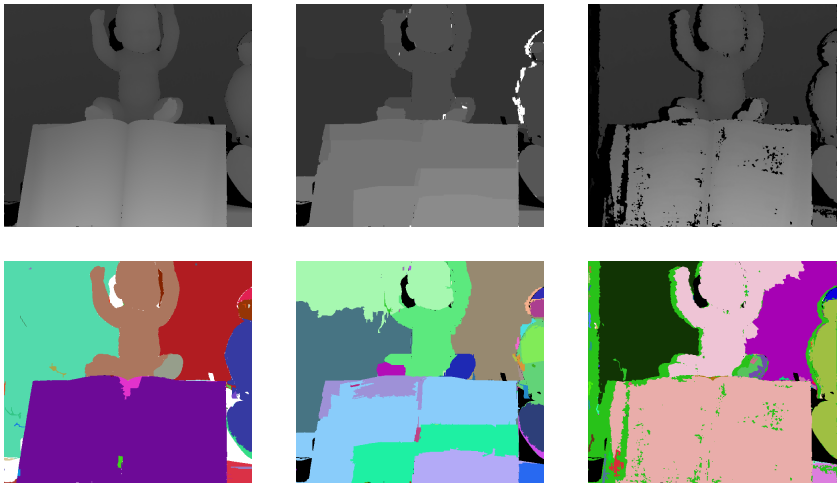


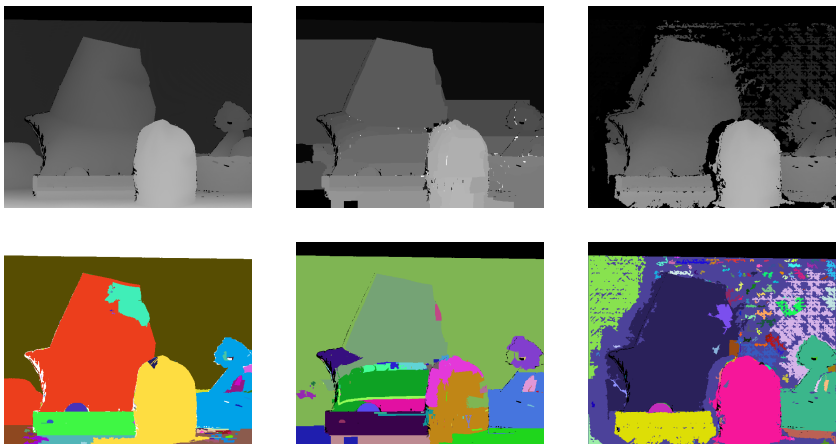
Figure 4.5: Comparison of 3D data sources

Figure 4.5 shows the segmentation quality obtained using angle or euclid distance with optimal values of σ_z and σ_c ($\sigma_z = 3.5, \sigma_c = 50$ and $\sigma_z = 4, \sigma_c = \inf$ respectively) and three different 3D data sources: structured ligth (ground-truth), Graph-Cut Stereo and SGBM Stereo. Figures 4.6 and 4.7 show qualitative effects of different 3D data source in case of Baby2 and Midd2 scenes.



Structured light Graph Cut SGBM

Figure 4.6: Three disparity maps and their corresponding segmentation - Baby2



Structured light Graph Cut SGBM

Figure 4.7: Three disparity maps and their corresponding segmentation - Midd2

It can be seen that neither Graph Cut nor SGBM prevails on the other: in fact, there are almost half cases in which SGBM lead to a better segmentation (such as

Baby2), and the residual in which Graph-Cut is better (such as Midd2). This is due to fact that, in scenes containing large untextured areas, a global method like GC is able to better discriminate depth regions. On the other hand, in scenes with lots of textures, the depth map produced may be smoother than local methods, hence hendering the segmentation.

4.5 Comparison with other segmentation algorithms

In the third batch of tests, results of the proposed algorithm are compared with other image segmentation algorithms. Specifically, the considered algorithms are:

RGB Kmeans on color data

RGBXY Kmeans on union of color and 2D spatial data

XYZ Kmeans on 3D spatial data

DalMutto Kmeans on 6D feature space LABXYZ combining color and 3D data proposed in [9]

Werman Mean shift on a 6D feature space XYRGBD proposed in [8].

Algorithms which combine color and spatial data use a weighting constant to tune the contribution and the relevance of color and geometric information. Figure 4.8 summarizes the results achieved by each algorithm on different images of the dataset, while Table 4.2 shows the average results of each segmentation algorithm.

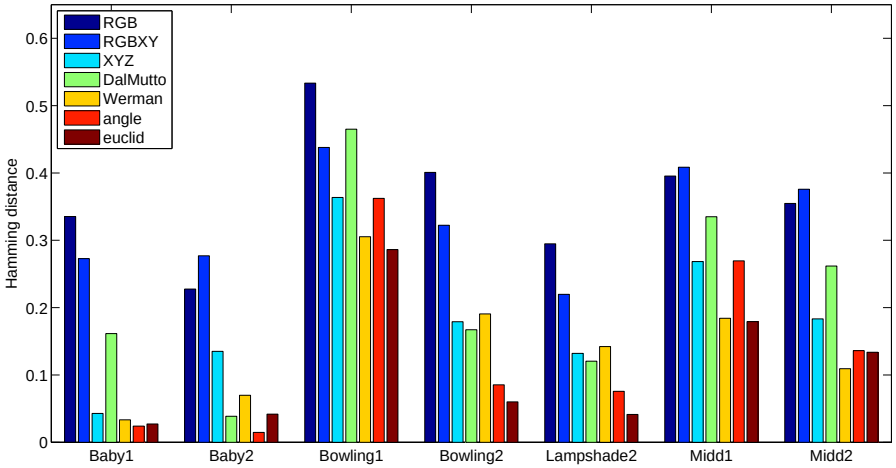
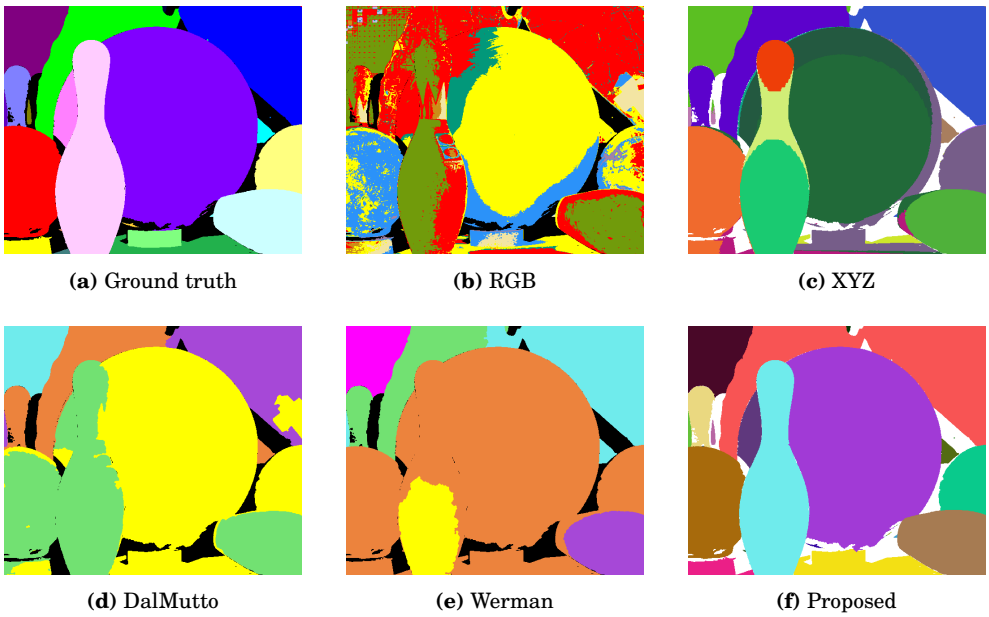


Figure 4.8: Comparison of different segmentation algorithms

The three basic kmeans algorithms (RGB, RGBXY and XYZ) have been been tested with many different values of k and only the best result has been considered in the statistic. Nevertheless, they all tend to performs badly with respect to other methods, as expected. Overall, the proposed algorithm obtains very good results in most situations. Finally in Figure 4.9 some qualitative results are shown.

algorithm	Hamming	Rand	Fowlkes	Jaccard
RGB	0.36319	0.19043	0.43195	0.59879
RGBXY	0.33061	0.15499	0.40427	0.58648
XYZ	0.18635	0.07415	0.18506	0.30276
DalMutto	0.22136	0.13409	0.32237	0.46917
Werman	0.14784	0.09600	0.19057	0.46917
angle	0.13824	0.09334	0.16830	0.27086
euclid	0.10991	0.05958	0.13123	0.21221

Table 4.2: Comparison of different segmentation algorithms



(a) Ground truth **(b)** RGB **(c)** XYZ

(d) DalMutto **(e)** Werman **(f)** Proposed

Figure 4.9: Comparison of qualitative results of different segmentation algorithms

Chapter 5

Conclusions

In this thesis a new offline segmentation algorithm based on evolutionary game has been proposed. Experimental tests that have been carried out showed that the proposed approach achieves overall good results and that it performs at same level of other recent methods reaching in some cases even a higher quantitative quality. Considering that it is a first implementation and that it has quite wide margin for improvement, it can be considered a promising approach. On the other hand, experiments on segmentation in association with a stereo algorithm showed that the proposed algorithm is quite sensible to depth map quality: as the depth map gets more inaccurate, the poorer are the segmentation results, especially in the presence of many occluded points.

The main amendable aspects and the directions for future development that have been identified are:

- The segmentation algorithm should be tested against other image dataset, in order to confirm or controvert consideration of section 4.3 on parameters configuration. It is necessary to verify if euclid parameters are overfitted and, at the same time, check if the normalization introduced by angle distance make it possible to use the same parameters over different image dataset.
- It could be interesting to implement and test other distance measures, considering that the proposed color distance is not sufficient to encapsulate the whole photocromic information.
- It is necessary to implement a technique to treat also occluded points, at least small areas that can be included in neighbouring regions. This is particularly relevant when the segmentation algorithm is used in association with stereo algorithms, which produce noisy depth data and a considerable number of occluded points.

- Code can be optimezid, as computational speed was not the main target, even though some measures for reducing time complexity have already been implemented. Nevertheless, average execution time of the segmentation algorithm is about 30-40 seconds on a high performance server and about 4-5 minutes on a user-level system. For example, the whole second phase presents a high grade of parallelization, considering that the computation of distance and similarity regarding each pair of macropixels is indipendent from other couples. In the current implementation, each iteration of the modiefied Dijkstra process is performed by a different thread, thus exploiting the computational potential of multi-process systems, but it could be further improved using GPU parallelization.

Appendix A

Implementation details

The program has been developed in C# language using Visual Studio 2010 Professional. Besides standard libraries, Emgu libraries [40], a C# wrapper to OpenCV functions [41] have been extensively used too.

The program is made up of one principal class - SegmentImage - which contains most of the code and four auxiliar classes: Vertex, Macropixel, Priority Queue and CalculateScore. An auxiliary structure has been defined too, Point3D: it represents a range point and stores the three coordinates $x,\ y,\ z$ as `double` values.

Vertex.cs

Instances of Vertex class represent the nodes of a 4-connected graph corresponding to the image. Fields of each Vertex object are:

- `idRow` and `idColumn`: values that identify the vertex using the row and column coordinates of the corresponding pixel in the image
- `id`: value that identifies the vertex as if the image were a row-vector instead of a matrix; `id` = `idRow` * offset + `idColumn`
- `AdjacentVertices`: list of connected vertices and corresponding distances, stored as `Dictionary<Vertex, double>`.

In addiction, `Vertex` class has a static property `offset`, which corresponds to the image width and it is used to transform `idRow` and `idColumn` into `id` and vice-versa.

Macropixel.cs

Instances of Macropixel class represent the subsets of pixels introduced in section 3.1. Fields of each Macropixel object are:

- `id`: identification number of the macropixel; it can be provided as external parameter of constructor, or determined using an incremental counter
- `Pixels`: set of pixels belonging to the macropixel, stored as `List<Point>`
- `Centroid`: a `Point3D` which stores the 3D coordinates of the centroid of the macropixel, calculated as the average of 3D coordinates of range points corresponding to pixels of the instance
- `iCentroid` and `jCentroid`: values that store the coordinates of the pixel belonging to the macropixel whose corresponding range point is the closest one to `Centroid`
- `uColor` and `vColor`: U and V components of the average color of the macropixel.

In addiction, Macropixel class has a static property, `nextIndex`, which is used as internal counter to generate next index values, and a static method `resetIndex`, which set counter back to 0 without making it accessible.

Macropixel instances are used also to represent final segments produced by segmentation algorithm; in this case, only `id` and `Pixels` variables are initialized.

CalculateScore.cs

CalculateScore class encapsulate the evaluation process section 4.2; it stores two lists of segments, one produced by the segmentation algorithm and one provided as ground-truth (manual segmentation).

Fields of each CalculateScore object are:

- `automatic`: set of segments produced by segmentation algorithm; it is stored as a `List<Macropixel>`
- `manual`: segmentation provided as ground-truth; it is stored as `List<Macropixel>`
- `intersectionCountMatrix`: contingency table, which stores the cardinality of the intersection between each pair of automatic-segment and manual-segment
- `numberOfPixels`: number of pixels of the image to be considered, i.e. `width * height - occludedPixels`

Public methods are:

- `loadManual(string path)`: it loads an image corresponding to manual segmentation, then it turns the image into a `List<Macropixel>` and stores the list in `manual` field

- `setAutomatic(string path)`: it loads an image representing automatic segmentation, then it turns the image it into a `List<Macropixel>` and it stores it in `automatic` field
- `setAutomatic(List<Macropixel> segments)`: it copies the list provided in parameter `segments` in `automatic` field
- `calculateIntersectionCountMatrix`: it computes the contingency table. Two different implementation have been created:
  - naive sequential implementation:

```
1 for (int i = 0; i < manual.Count; i++)
2  for (int j = 0; j < automatic.Count; j++)
3   intersectionCountMatrix[i, j] = manual[i].Pixels.Intersect
        (automatic[j].Pixels).Count();
```

  - parallel implementation: each thread deal with a manual-segment

```
1 Parallel.For(0, manual.Count - 1, i =>
2  {for (int j = 0; j < automatic.Count; j++)
3     intersectionCountMatrix[i, j] = manual[i].Pixels.
          Intersect(automatic[j].Pixels).Count();});
```

- `calculateHamming`: it calculates Hamming Distance presented in subsection 4.2.1
- `calculateCE`: it calcualtes GCE and LCE indexes presented in subsection 4.2.2
- `calculateMetrics`: it calculates Rand Index, Fowlkes Index and Jaccard Index presented in subsection 4.2.3

PriorityQueue.cs

PriorityQueue class implements a priority queue using a min-binary heap data structure and it used to perform the modiefied Dijkstra algorithm.

A priority queue is an abstract data type similar to a queue data structure, but, additionaly, each element is associated with a priority. Priority queue are specifically designed to quickly:

- add an element to the queue with an associated priority
- remove the element from the queue that has the highest (or lowest) priority, and return it

A binary heap is a binary tree with two additional constraints:

- The shape property: the tree is a complete binary tree; that is, all levels of the tree, except possibly the last one (deepest) are fully filled, and, if the last level of the tree is not complete, the nodes of that level are filled from left to right.

- The heap property: each node priority is lower than or equal to each of its children priority.

Because binary heap is a complete binary tree, it can be stored compactly in a linear array, without using pointers considering that the parent and children of each node can be found by arithmetic on array indices: if the tree root is at index 1, then element at index i has children at $[2i]$ and $[2i + 1]$, and parent at $[floor(i/2)]$.

Specifically, the items contained in the priority queue are Vertex object and their priority is the distance of the Vertex from its predecessor in the path generated by Dijkstra algorithm.

The operations supported by PriorityQueue class are:

- `Add(item, priority)`: `item` is inserted as last item of heap structure, according to shape property; in order to restore heap property `heapify-up` procedure is called:

```
1 Compare the added element with its parent; if they are in the
    correct order, stop.
2 If not, swap the element with its parent and return to the
    previous step.
```

- `RemoveMin`: the element with the smallest priority, which corresponds to root element, is removed. The root element is swapped with the last item of heap structure and then it is removed; in order to restore heap property `heapify-down` procedure is called:

```
1 Compare the new root with its children; if they are in the
    correct order, stop.
2 If not, swap the element with its smaller children and return to
     the previous step.
```

- `DecreaseKey(item, newPriority)`: if `item` is in the heap structure and its current priority is greater than `newPriority`, its priority is set to `newPriority` and `heapify-up` procedure is called.

SegmentImage.cs

SegmentImage is the project principal class and it encapsulate almost the entire segmentation algorithm. Its main fields are:

- `colorImage`: it stores the input color image
- `disparityMap`: it stores the input disparity map
- `rangeImage`: matrix of `Point3D` objects which represent the range image computed from the disparity map; element at (i, j) contains the 3D coordinates corresponding to pixel at row i and column j
- `segments`: it stores the result of oversegmentation phase as a `List<Macropixel>`
- `graph`: matrix of `Vertex` object, which represents the 4-connected graph corresponding to the image; element at (i, j) represents the node corresponding to pixel at row i and column j
- `geometricMatrix`: matrix whose element at (i, j) corresponds to the geometric distance between Macropixel i and Macropixel j
- `colorMatrix`: matrix whose element at (i, j) represents the color distance between Macropixel i and Macropixel j
- `compatibilityMatrix`: matrix whose element at (i, j) represents the compatibility between Macropixel i and Macropixel j as defined in section 3.2
- `components`: result of segmentation algorithm as `List<Macropixel>`

Methods of SegmentImage class can be split into three main categories, according to the phase of segmentation algorithm they pertain to.

Oversegmentation phase

- `oversegment(`σ`, k ,min,` σ_t`)`: this method controls the whole oversegmentation phase; it constists of following steps
  - convert color image to .ppm format
  - invoke external procedure to perform Efficient Graph segmentation via a System.Diagnostic.Process object. Its input arguments are σ, k, min, path of .ppm input image, path of output image; it produces as output a .ppm image corresponding to oversegmented image, in which each segment is distinguished by a different random color.

– trasnform oversegmented image in a temporary list of macropixel, removing at the same time occluded pixels

```
Dictionary<Rgb, Macropixel> dict = new Dictionary<Rgb,
    Macropixel>();
Macropixel occludedPoints = new Macropixel(-1);
for (int i = 0; i < overSegmentedImage.Height; i++)
  for (int j = 0; j < overSegmentedImage.Width; j++){
    if (disparityMap.Data[i, j, 0] == 0)
       occludedPoints.Add(new Point(i, j));
    else {
      Macropixel mp;
      Rgb color = overSegmentedImage[i, j];
      if (!dict.TryGetValue(color, out mp))
      {
        mp = new Macropixel();
        dict.Add(color, mp);
      }
      mp.Pixels.Add(new System.Drawing.Point(i, j));
    }
  }
tempSegments = dict.Values.ToList<Macropixel>();
```

– check depth homogeneity of each temporary segment, invoking CheckDepth method

– check connectivity of each temporary segment, using CheckConnectivity method

```
foreach tempSegment
 List<Point> newList;
 bool done = false;
  while (!done){
   done = checkConnectivity(tempSegment.Pixels,out newList);
   Macropixel newMacroPixel = new Macropixel();
   newMacroPixel.Pixels = newList;
   segments.Add(newMacroPixel);
  }
```

- `CheckDepth(macropixel)`: this method checks depth homogeneity of `macropixel`

```
deviation ← standard deviation of disparity values
if (deviation > σ_t)
 split macropixel into two subsegments using k-means with k = 2
 add subsegments to temporary segments
```

```
5 end if
```

- `CheckConnectivity(pixelsIn, out pixelsOut)`: given a set of pixels as inputs, it returns true if and only if pixels in pixelsIn form a single connect component. Starting from a pixel p, it tries to expand the region through 8-connected neighbooring pixels until there are no more reachable pixels. At the end of the procedure, pixelsOut will contain the pixels that form a connected component, while pixelsIn will eventually contain pixels that are not connected to p. CheckConnectivity method is used also in the last step of third phase

```
1  pick a pixel p ∈ pixelsIn
2  Queue Q ← ∅
3  Enqueue(Q, p)
4  while (Q ≠ ∅){
5   x ← dequeue(Q)
6   add(pixelsOut, x)
7   foreach pixel ∈ neighboor(x)
8    if pixel ∈ pixelsIn
9     enqueue(Q, pixel)
10    remove(pixelsIn, pixel)
11   end if
12 end while
```

Compatibility computation phase

- `CalculateColorMatrix`: it computes the pairwise color distance.

```
1  foreach (Macropixel mp in segments)
2    calculateAverageColor(mp);
3  colorDistanceMatrix = new float[segments.Count, segments.Count];
4  for (int i = 0; i < segments.Count; i++)
5   for (int j = i + 1; j < segments.Count; j++)
6   {
7   double distance = 0;
8   distance += Math.Pow(segments[j].UComponent - segments[i].
        UComponent, 2);
9   distance += Math.Pow(segments[j].VComponent - segments[i].
        VComponent, 2);
10  colorDistanceMatrix[i, j] = colorDistanceMatrix[j, i] = (float)
        Math.Sqrt(distance);
11  }
```

- `buildRangeImage`: given a disparity map, it computes the corresponding range image, using following formulae
  - X = [column - (width/2)] * baseline / disparity
  - Y = [row - (height/2)] * baseline / disparity
  - Z = focalLength * baseline / disparity
- `InitializeGraph`: it initializes `graph` variable, creating a `Vertex` object for each non-occluded pixel.
- `CreateEdges(distance type)`: it completes the graph, computing the weigth of edges connecing neighbooring pixels. It uses the formula corresponding to the distance type as defined in section 3.2.
- `FindCentroid(macropixel)`: it computes the 3D coordinates of the centroid of `macropixel` as the average of 3D coordinates of every range point whose correspoinding pixel belongs to `macropixel`. Then it searches the pixel whose corresponding range point closer to centroid.
- `CalculateGeometricMatrix`: it computes the pairwise geometric distance between macropixels.

```
1  Parallel.For(0, segments.Count - 1, i =>
2   {
3    double[] saltoMax = modifiedDijkstra(segments[i].iCentroid,
         segments[i].jCentroid);
4    for (int j = i + 1; j < segments.Count; j++)
5     {
6      float value = 0;
7      int id = segments[j].iCentroid * Vertex.offset + segments[j
           ].jCentroid;
8      if (saltoMax[id] != -1)
9       value = (float)saltoMax[id];
10     else
11      value = float.PositiveInfinity;
12     spatialDistanceMatrix[i, j] = spatialDistanceMatrix[j, i] =
           value;
13    }
14   spatialDistanceMatrix[i, i] = 0;
15 });
```

- `ModifiedDijkstra(row,column)`: it implements the modified Dijkstra algorithm introduced in section 3.2 to compute the "smallest maximum drop"

path from `(row, column)` pixel to every other pixel. Let `S` represents the set of already processed pixels, `frontier` the set of pixels that are not in `S` but that are connected to a pixel in `S`, `T` the set of the rest of image pixels and `maxDrop` the vector whose i-th element is the maximum drop of the path connecting (row,column) and i-th pixel. `Frontier` is implemented as a PriorityQueue.

```
1  maxDrop(p) ← ∞ for each pixel ≠ (row,column)
2  S ← ∅
3  frontier ← {(row,column)}
4  while frontier ≠ ∅
5   x ← removeMin(frontier)
6   foreach p ∈ neighboor(x)
7    if p ∈ T
8     remove(T,p)
9     maxDrop(p) ← max{w(x,p),maxDrop(x)}
10    insert(frontier,p,maxDrop(p))
11   end if
12   else if p ∈ frontier
13    if max{w(x,p), maxDrop(x)} < maxDrop(p)
14     decreaseKey(p, max{w(x,p), maxDrop(x)})
15    end if
16   end if
17  end foreach
18 end while
```

- `CalculateCompatibility(`σ_z, σ_c`)` computes the pairwise compatibility between macropixels using the bivariate gaussian function expressed in Equation 3.1

Clustering phase

- `segment`: this method controls the third phase of segmentation algorithm. It keeps track of the macropixels that have not already been grouped by storing their indexes in a List called segmentsLeftToCluster. It starts a new evolutionary game as described in section 3.3 until all macropixels are grouped. At each iteration, it builds up a payoff matrix using the compatibility values of unprocessed macropixels and it provides the matrix as input parameter of evolutionary game library, which returns as output a population vector whose ith element corresponds to the likelyhood of ith macropixel to belong to the cluster at equilibrium situation. Macropixels whose population is at least 10% of the maximum likelyhood are selected, merged and removed from

the list of unprocessed macropixels. Then, the connectivity of the produced segment is checked using the `checkConnectivity` method previously described. The segment (or the segments) produced is then inserted into the `components` list.

Bibliography

[1] F. Bergamasco, A. Albarelli, A. Torsello, M. Favaro, and P. Zanuttigh, "Pairwise similarities for scene segmentation combining color and depth data," in *21st International Conference on Pattern Recognition (ICPR 2012)*, November 11-15, 2012. (submitted).

[2] "Middlebury stereo dataset." `http://vision.middlebury.edu/stereo/data/scenes2006/`.

[3] V. Kolmogorov and R. Zabih, "Computing visual correspondence with occlusions via graph cuts," in *IEEE International Conference on Computer Vision (ICCV)*, pp. 508–515, 2001.

[4] H. Hirschmüller, "Stereo processing by semiglobal matching and mutual information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 2, pp. 328–341, 2008.

[5] F. Tang, M. Harville, H. Tao, and I. Robinson, "Fusion of local appearance with stereo depth for object tracking," in *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on*, pp. 1 –8, june 2008.

[6] K. Ntalianis, A. Doulamis, N. Doulamis, and S. Kollias, "Unsupervised segmentation of stereoscopic video objects: investigation of two depth-based approaches," in *Digital Signal Processing, 2002. DSP 2002. 2002 14th International Conference on*, vol. 2, pp. 693 – 696 vol.2, 2002.

[7] P. An, C. Lu, and Z. Zhang, "Object segmentation using stereo images," in *Communications, Circuits and Systems, 2004. ICCCAS 2004. 2004 International Conference on*, vol. 1, pp. 534 – 538 Vol.1, june 2004.

[8] A. Bleiweiss and M. Werman, "Fusing time-of-flight depth and color for real-time segmentation and tracking," in *Dyn3D*, pp. 58–69, 2009.

[9] C. D. Mutto, P. Zanuttigh, G. M. Cortelazzo, and S. Mattoccia, "Scene segmentation assisted by stereo vision," in *3DIMPVT*, pp. 57–64, 2011.

[10] J. W. Weibull, *Evolutionary Game Theory*, vol. 1 of *MIT Press Books*. The MIT Press, June 1997.

[11] Q. Wu and C. K. R., "Chapter 9 - image segmentation," in *Microscope Image Processing*, pp. 159 – 194, Burlington: Academic Press, 2008.

[12] D. H. Ballard and C. M. Brown, *Computer Vision*. Englewood Cliffs, NJ: Prentice-Hall, 1982.

[13] A. Blasiak, "A comparison of image segmentation methods," Master's thesis, Middlebury College, 2007.

[14] "Segmentation of images and video." `http://encyclopedia.jrank.org/articles/pages/6890/Segmentation-of-Images-and-Video.html`.

[15] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, ch. 10. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2nd ed., 2001.

[16] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 8, pp. 679–698, June 1986.

[17] S. L. Horowitz and T. Pavlidis, "Picture Segmentation by a directed split-and-merge procedure," *Proceedings of the 2nd International Joint Conference on Pattern Recognition, Copenhagen, Denmark*, pp. 424–433, 1974.

[18] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Comput. Surv.*, vol. 31, pp. 264–323, Sept. 1999.

[19] D. Comaniciu, "Image segmentation using clustering with saddle point detection," in *Image Processing. 2002. Proceedings. 2002 International Conference on*, vol. 3, pp. III–297 – III–300 vol.3, 2002.

[20] J. B. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability (Berkeley, California, 1965/66)*, pp. 281–297, Berkeley, California: Univiversity of California Press, 1967.

[21] W. D. Fisher, "On grouping for maximum homogeneity," *Journal of the American Statistical Association*, vol. 53, no. 284, pp. 789–798, 1958.

[22] K. Fukunaga and L. Hostetler, "The estimation of the gradient of a density function, with applications in pattern recognition," *IEEE Transactions on Information Theory*, vol. 21, no. 1, pp. 32–40, 1975.

[23] Y. Cheng, "Mean shift, mode seeking, and clustering," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 17, pp. 790 –799, aug 1995.

[24] D. Comaniciu and P. Meer, "Mean shift: a robust approach toward feature space analysis," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, pp. 603 –619, may 2002.

[25] J. Shi and J. Malik, "Normalized cuts and image segmentation," in *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pp. 731 –737, jun 1997.

[26] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *International Journal of Computer Vision*, vol. 59, pp. 167–181, Sept. 2004.

[27] Z. Wu and R. Leahy, "An optimal graph theoretic approach to data clustering: theory and its application to image segmentation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 15, pp. 1101 –1113, nov 1993.

[28] A. Albarelli, S. Rota Bulo, A. Torsello, and M. Pelillo, "Matching as a non-cooperative game," in *Computer Vision, 2009 IEEE 12th International Conference on*, pp. 1319 –1326, 29 2009-oct. 2 2009.

[29] A. Albarelli, E. Rodolà, and A. Torsello, "Imposing semi-local geometric constraints for accurate correspondences selection in structure from motion: A game-theoretic perspective," *International Journal of Computer Vision*, vol. 97, no. 1, pp. 36–53, 2012.

[30] A. Torsello, S. Bulo, and M. Pelillo, "Grouping with asymmetric affinities: A game-theoretic perspective," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 1, pp. 292 – 299, june 2006.

[31] P. D. Taylor and L. B. Jonker, "Evolutionary stable strategies and game dynamics," *Mathematical Biosciences*, vol. 40, no. 1 - 2, pp. 145 – 156, 1978.

[32] X. Jiang, "Performance evaluation of image segmentation algorithms," in *Handbook of Pattern Recognition and Computer Vision* (C. H. Chen and P. S. P. Wang, eds.), pp. 525–542, World Scientific, 2005.

[33] D. Scharstein and R. Szeliski, "High-accuracy stereo depth maps using structured light," in *IEEE computer society conference on Computer vision and Pattern Recognition*, CVPR 2003, (Madison, Wisconsin), pp. 195–202, 2003.

[34] X. Chen, A. Golovinskiy, and T. Funkhouser, "A benchmark for 3D mesh segmentation," *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 28, Aug. 2009.

[35] X. Jiang, C. Marti, C. Irniger, and H. Bunke, "Distance measures for image segmentation evaluation," *EURASIP Journal on Applied Signal Processing*, vol. 2006, pp. 1–10, 2006.

[36] Q. Huang and B. Dom, "Quantitative methods of evaluating image segmentation," in *Proceedings of the 1995 International Conference on Image Processing (Vol. 3)-Volume 3 - Volume 3*, ICIP '95, (Washington, DC, USA), pp. 3053–, IEEE Computer Society, 1995.

[37] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, vol. 2, pp. 416 –423 vol.2, 2001.

[38] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical Association*, vol. 66, no. 336, pp. pp. 846–850, 1971.

[39] E. B. Fowlkes and C. L. Mallows, "A method for comparing two hierarchical clusterings," *Journal of the American Statistical Association*, vol. 78, no. 383, pp. pp. 553–569, 1983.

[40] "EmguCV: cross platform .Net wrapper to OpenCV." `http://www.emgu.com/wiki`.

[41] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.