



ÉCOLE POLYTECHNIQUE  
FÉDÉRALE DE LAUSANNE

SEMESTER PROJECT REPORT

# Tag Your Memory with Colors: Semi-automatic Recoloring on Interested Objects via Hypergraph Learning

Student **Alexandre Chappuis** <alexandre.chappuis@epfl.ch>

Supervised by **Feng Yang** <feng.yang@epfl.ch>  
Audiovisual Communications Laboratory (LCAV)  
**Li Pu** <li.pu@epfl.ch>  
Laboratory of Artificial Intelligence (LIA)

Abstract A new image segmentation technique via hypergraph learning is presented, along with an Android application providing semi-automatic recoloring on a region of interest. Details of both theory and implementation are given in this report.

January 12, 2012

## Contents

<b>I</b>	<b>Introduction</b>	<b>3</b>
<b>II</b>	<b>Image segmentation basics</b>	<b>4</b>
1	Overview . . . . .	4
2	Existing work . . . . .	4
2.1	Kmeans . . . . .	4
2.2	Meanshift mode seeking . . . . .	5
2.3	Normalized cut . . . . .	6
2.4	Contour detection and hierarchical segmentation . . . . .	7
2.5	Ensemble methods . . . . .	7
2.6	Summary and examples . . . . .	7
<b>III</b>	<b>Segmentation via hypergraph learning</b>	<b>10</b>
3	Overview . . . . .	10
4	Hypergraph creation . . . . .	10
5	Hypergraph cut . . . . .	12
6	Summary and examples . . . . .	12
7	Matlab code . . . . .	14
8	Evaluation . . . . .	15
<b>IV</b>	<b>Android application</b>	<b>18</b>
9	Specification . . . . .	19
9.1	Overview . . . . .	19
9.2	Application . . . . .	19
9.3	Server . . . . .	21
10	Implementation details . . . . .	21
10.1	Activities . . . . .	22
10.2	Server and Matlab interaction . . . . .	24
11	Examples . . . . .	25
12	Design choices, limitations and possible extensions . . . . .	25
<b>V</b>	<b>Conclusion</b>	<b>28</b>
<b>VI</b>	<b>Future work</b>	<b>28</b>

## List of Algorithms

1	Kmeans clustering . . . . .	5
2	Meanshift . . . . .	6
3	Normalized cut . . . . .	7
4	Contour and hierarchical segmentation . . . . .	7
5	Segmentation via hypergraph learning . . . . .	13

## List of Figures

1	Example of simple segmentations . . . . .	8
2	Example of superpixels . . . . .	11
3	Example of hypergraph . . . . .	11
4	Example of segmentation merging and hypergraph segmentation . . . . .	13
5	Matlab GUI . . . . .	15
6	Overview of the recoloring process . . . . .	20
7	Home and recoloring screens . . . . .	20
8	Java classes of the Android application . . . . .	22
9	Protocol used between the Android phone and the Java server . . . . .	23
10	The colorpicker used to choose a color. . . . .	24
11	Example of expansion and color change . . . . .	26

## List of Tables

1	Summary of representative segmentation algorithms . . . . .	9
2	Summary of the fields in a hypergraph structure . . . . .	14
3	Evaluation results for simple methods . . . . .	18
4	Evaluation results 1 for hypergraph method . . . . .	18
5	Evaluation results 2 for hypergraph method . . . . .	19

## Part I. Introduction

Image segmentation is the first step of image analysis, aiming at grouping image pixels into meaningful segments. Thousands of clustering algorithms have been proposed over the last decades, with recent significant progress. In this project, a new ensemble approach using the concepts of superpixels and hypergraphs is investigated. Existing segmentations are combined to form superpixels, resulting in a hypergraph that can be cut into a custom number of segments. The first goal of this project was not developing a new segmentation algorithm which beats all existing approaches, but to demonstrate the effectiveness of hypergraph learning techniques in image segmentation.

Applications of image segmentation are omnipresent in today's world. With the recent growth of pocket devices markets and their embedded digital cameras, creating cool applications that modify pictures has become a child's play. By using recoloring applications, one can for example create funny images where some interested region is colorful and everything else is grayed out. As of January 2012, this procedure has to be done manually in all existing applications. Being able to recolor one image in a more intelligent way on smartphones was the second goal of this project. With the knowledge of segmentations of a picture and by using the new hypergraph approach, an Android application providing semi-automatic recoloring on region of interest has been created. People can consequently tap on an object in order to recolor it entirely.

In the first part of this report, the goals of image segmentation as well as some representative image segmentation techniques are carefully explained. Then, the new segmentation method is developed along with some examples and its evaluation. In the final part, the implementation of the Android application is detailed.

## Part II. Image segmentation basics

### 1 Overview

Image segmentation is a fundamental topic of digital image and video processing. It seeks to partition an image into  $K$  non-overlapping regions, called segments, whose union is the entire image. Many application in computer vision, digital entertainment and pattern recognition rely on this challenging and critical step. Segmenting images has been widely studied and thousands of methods have been invented since the beginning of the computer era. (See [1] for a brief survey of this research in the last 40 years). Although some exciting results have emerged recently, there is no unique solution to that problem, mostly because of the broad range of applications and the difficulty in having an accurate segmentation. What's more, the process is often computationally expensive.

As segmenting an image strongly depends on the usage scenario, many approaches are possible while designing an algorithm. Survey [2] gives insight on some important principles; it distinguishes notably supervised from unsupervised methods. In the former, user interaction is required whereas in the latter, *there is no contextual knowledge assumption regarding to the object being segmented*. This difference will play a significant role in the evaluation of the algorithms.

Five representative methods are explained in the following section: the "K-means clustering algorithm" (abbreviated later as "Kmeans"), the Meanshift mode seeking algorithm, the Normalized Cut, the contour detection and hierarchical based segmentation and finally some words about ensemble methods. They will serve as a basis for the new method proposed in part III.

### 2 Existing work

#### 2.1 Kmeans

Kmeans is a well-known algorithm used to partition data into  $K$  clusters.  $K$  random or uniform seeds are initially chosen. The algorithm subsequently alternates between two steps: an assignment step where each point is associated to its closest mean, and an update step during which every mean is recomputed according to the previous assignment. It converges when the  $K$  means no longer change. At the end, the within cluster sum of squares is minimized. The pseudo-code is listed in algorithm 1. Note that in order to speed up the operations, the squared root operator is not used while computing the Euclidean distance. This does not affect the final result.

This algorithm is pretty fast and generic, *i.e.* it works for many kinds of data, for examples images represented in RGB or Lab colorspace. The drawback is that it's quite sensitive to outliers and works well only for spherical clusters. In practice, I have downloaded a Matlab Kmeans implementation from [tool1], that is quicker than Matlab's built-in Kmeans, and I have extended it with some ideas found in [3]. Basically, two dimensions are added to the image in order to have spatial coherence, one for the  $x$  coordinate and one for the  $y$  coordinate, and a new parameter, the minimum region area, is used in a post-processing

**Algorithm 1** Kmeans clustering

---

Start with K initial means  $\mu_k, k = 1..K$   
**repeat**  
  **for all** datapoint  $x_n, n = 1..N$  **do**  
    1. Compute the squared Euclidean distance from the  $k^{th}$  mean:  
 $dist_k(x_n) = \|x_n - \mu_k\|^2$   
    2. Assign each data point  $x_n$  to its closest mean  $\mu_k$ , i.e assign  $x_n$  to class  $q_k$  if  $dist_k(x_n) \leq dist_l(x_n) \forall l \neq k$   
    3. Replace each mean with the mean of the data points assigned to the corresponding class  
  **end for**  
**until** no further change occurs

---

step so that regions that are too small to represent a segment are removed. The vertical and horizontal sensitivities parameters,  $s_x$  and  $s_y$ , scale respectively the two added dimensions. One can therefore give more importance to vertical or horizontal elements in an image.

## 2.2 Meanshift mode seeking

The meanshift method [4] finds modes or local maximum of a density function in the feature space. The modes of the image are located at the zeros of its gradient. Given  $n$  data points  $x_i, i = 1..n$ , a density function  $f$ , a window of radius  $h$  represented by a kernel function <sup>1</sup>  $G$  and a  $d$ -dimensional space  $R^d$ , the expression of the gradient is given by the following equation (See [4] for detailed explanations) :

$$\nabla \hat{f}_{h,K}(\mathbf{x}) = \underbrace{\frac{2c_{K,d}}{nh^{d+2}} \left[ \sum_{i=1}^n g\left(\left\|\frac{\mathbf{x}-x_i}{h}\right\|^2\right) \right]}_{KDE} \underbrace{\left[ \frac{\sum_{i=1}^n x_i g\left(\left\|\frac{\mathbf{x}-x_i}{h}\right\|^2\right)}{\sum_{i=1}^n g\left(\left\|\frac{\mathbf{x}-x_i}{h}\right\|^2\right)} - \mathbf{x} \right]}_{\text{Meanshift vector } m_{h,G}(\mathbf{x})}$$

The first term, the kernel density estimator (KDE), is proportional to the density estimate at  $x$  computed with the kernel  $G(x) = c_{g,d}(\|x\|^2)$  and the second term is the mean shift vector. This vector points always toward the direction of the maximum increase in the density, thus leading to stationary points, the modes of the image. The overall procedure is listed in algorithm 2.

To sum up, an image has many attraction basins with respect to some features. Finding the centers of gravity of those basins is like climbing up hills until the peaks are reached. It's also worth noting that convergence is always guaranteed and that a linear feature space, such as the Lab or Luv colorspace, is required for accurate results.

---

<sup>1</sup> A function  $K : X \rightarrow R$  is said to be a kernel if there exists a profile,  $k : [0, \infty] \rightarrow R$  s.t.

1.  $K(x) = k(\|x\|^2)$
2.  $k$  is non-negative and non-increasing
3.  $k$  is piecewise continuous and  $\int_0^\infty k(r)dr < \infty$

**Algorithm 2** Meanshift

- 
1. Find features (color, texture, *etc.*)
  2. Initialize windows at individual feature points
  - repeat**
    - A. Compute the meanshift vector  $m_h(x^t)$  for each window
    - B. Translate each window by its meanshift vector :  $x^{t+1} = x^t + m_h(x^t)$
  - until** convergence
  4. Merge windows that end up near the same peak or mode
  5. (Optional) Eliminate spatial regions containing less than M pixels
- 

The EDISON Matlab interface [tool2] has been used to segment images with the meanshift procedure. There are three parameters to specify :  $h_s, h_r$  and  $M$ , the minimum region area. The spatial bandwidth  $h_s$  defines the radius of the window in the spatial domain (x,y), whereas the range bandwidth  $h_r$  define its radius in the color domain.

Meanshift is clearly robust to outliers and doesn't constrain the shape of the clusters. What's more, a variable number of modes is found. However, this method is computationally expensive.

### 2.3 Normalized cut

Images can be seen as a graph with a node for every pixel, and an edge between any (sufficiently close) pairs of pixels. Each edge is weighted by a similarity function measured between two nodes. The normalized cut algorithm [5] treats segmentation as a spectral clustering problem and extends the idea of traditional graph cut<sup>2</sup>. Usual graph cuts tends to cut off very small, isolated component, whereas there is no such problem with the normalized cut approach. An image is first mapped to a weighted graph with affinity matrix  $W$ , whose elements are defined by a similarity measure between pixels. The goal is to find the normalized cut of this graph, defined as

$$Ncut(A, B) = \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)} \text{ with } assoc(A, V) = \sum_{u \in A, t \in V} w(u, t).$$

The main idea is to minimize the dissociation between groups and maximize the association within each group. The solution is given by the eigenvector corresponding to the  $2^{nd}$  smallest eigenvalue (by the Rayleigh quotient criterion) of the following equation:

$(D - W)y = \lambda Dy$  where  $W = (W_{ij})$  is the weight matrix and  $D$  is the  $N \times N$  degree matrix with the vector  $d$  on its diagonal,  $d(i) = \sum_j w(i, j)$ .

The pseudocode is given in algorithm 3.

In order to use this method, I've downloaded its Matlab code from [tool3]. Only one parameter is needed: K, the desired number of regions. Although the normalized cut is much better than traditional graph cuts, its high complexity ( $O(n^3)$ ) is an important drawback for large and complex images.

---

<sup>2</sup> A graph cut is defined as  $cut(A, B) = \sum_{i \in A, j \in B} w(i, j)$  where  $w(i, j)$  represent a weight between nodes  $i$  and  $j$ . Minimizing this cut value leads to an ideal bi-partitioning of the graph.

---

**Algorithm 3** Normalized cut

---

1. Set up a weighted graph  $G = (V, E)$  with an edge between each pair of pixels. Choose a similarity measure and compute matrices  $W$  and  $D$
  2. Solve  $(D - W)x = \lambda Dx \equiv D^{-1/2}(D - W)D^{-1/2}x = \lambda x$  for eigenvectors with the smallest eigenvalues
  3. Use the eigenvector with the second smallest eigenvalue to bipartite the graph
  4. Recurse if needed !
- 

## 2.4 Contour detection and hierarchical segmentation

Accurate contour detection is closely related to image segmentation. The recently published method [6] presents a state-of-the-art contour detection algorithm, the *gPb* contour detector, that can be used to partition an image into hierarchical regions. Two steps are needed: first, an Oriented Watershed Transform (OWT) is used to construct a set of initial regions, then an Ultrametric Contour Map (UCM) is constructed from the boundaries of these regions. The goal is to form a set of closed contours by estimating the probability that a contour is significant in an image. A hierarchy of regions is finally built by using a greedy graph-based region merging algorithm. For detailed explanations of each of these steps, please refer to [6]. High-level pseudocode is given in algorithm 4.

---

**Algorithm 4** Contour and hierarchical segmentation

---

1. Use the *gPb* contour detector
  2. Create an Oriented Watershed Transform of the contour
  3. Estimate the importance of the contours by using an Ultrametric Contour Map
  4. Create a hierarchical tree and use a region merging algorithm to eventually create segments
- 

Matlab code for this segmentation algorithm can be downloaded from [tool4]. Although providing excellent results in an unsupervised manner, this method is too slow for a real-time usage.

## 2.5 Ensemble methods

Another type of method called ensemble methods tries to combine results from simple methods in order to improve the accuracy of a segmentation. For example, the Maximum Weight Independent Set (MWIS) [7] problem can be used in order to intelligently combine segmentations. This approach will be used in the new segmentation algorithm explained in the next part of this report.

## 2.6 Summary and examples

As an example, figure 1 displays an image segmented with each of the above-mentioned method. Each segmentation is different from the others. Evaluating them is a hard task, and more thoughts will be presented in section 8.

The methods described in the previous subsections represent only a tiny subset of all existing work about segmentation. One can't say that one method





Fig. 1: Example of segmentations. On top: the original image. Then, from left to right: Kmeans, Meanshift, Normalized cut and Contour/Hierarchical segmentations

Method	Parameters	Advantages	Drawbacks
Kmeans	$K, s_x, s_y, M$	Fast, generic	Assumes spherical clusters, sensitive to outliers, initialization (choose k)
Meanshift	$h_s, h_r, M$	Robust to outliers, finds variable number of modes, does not constrain the shape of clusters	Sensitive to initialization, computationally expensive
Normalized cut	$K$	Generic, works with many different features	High memory usage, segments tend to have the same size
Contour detection and hierarchical segmentation	None	No parameter needed, excellent segmentation accuracy	Very high time complexity
Hybrid	Various	Better accuracy, freedom in combining segmentations	Slower than a simple method

Tab. 1: Summary of representative segmentation algorithms

is clearly better than the other, but still some positive and negative points can be drawn. A short summary is listed in table 1 for a better overview of their parameters, advantages and drawbacks.

## Part III. Segmentation via hypergraph learning

### 3 Overview

The new segmentation proposed can be qualified as an ensemble method. It first combines existing segmentations (at least two) in order to create superpixels. A hypergraph is then constructed from both the information present in the superpixels and the original segmentation. Finding a good segmentation is finally achieved by cutting the hypergraph into a maximum of  $K$  regions.

The idea of this new method is similar to what is found in [7], except that hypergraphs are used instead of traditional graphs. What's more, cutting a hypergraph is slightly different than cutting a simple graph and provide some advantages.

In the following subsections, details of both theory and Matlab implementation will be given. After some important definitions are given, the hypergraph creation and cut are explained. Finally, the algorithm is summarized, examples are given and the new method is evaluated versus the simple methods.

### 4 Hypergraph creation

#### Definitions

**Definition 1.** A *superpixel* is a set of contiguous pixels.

**Definition 2.** Let  $V$  be a finite set of vertices and  $E$  a family of subsets of  $V$  such that  $\cup_{e \in E} e = V$ .  $G = (V, E, \omega)$  is called a **hypergraph**, with the vertex set  $V$  and the hyperedge set  $E$ , and each hyperedge  $e$  is assigned a positive weight  $\omega(e)$  (Definition given by [8])

A set of superpixels can therefore be viewed as an over-segmented image. The notion of superpixels comes from [9] and an example is shown on figure 2. A hypergraph is a generalization of a simple graph, allowing more than 2 vertices for an edge. An example of hypergraph along with its incident matrix is displayed in figure 3. The incident matrix links hyperedges (rows) with vertices (columns). Note that there might be more than two ones in each row.

In the next paragraph, the link between original segmentations, superpixels and the hypergraph is made.

#### Merging simple algorithms

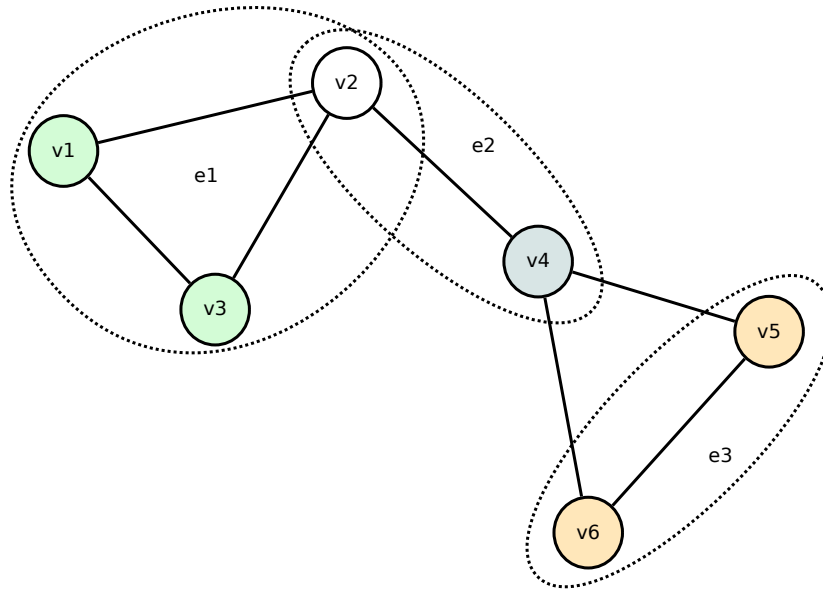
The first step toward hypergraph creation is to intersect at least two segmentation results, say for example the Kmeans  $S_1$  and the Meanshift  $S_2$  segmentations. If more than two segmentations are given, say  $S_1 \dots S_M$ , then the process can be continued in an iterative way. Let the final segmentation be  $I$ , given by

$$I = (((S_1 \cap S_2) \cap S_3) \cap S_4) \dots \cap S_M$$

$I$  can be viewed as a set of superpixels. Each superpixel is then represented as one vertex in a hypergraph. Hyperedges are subsequently defined as the link



Fig. 2: Example of image divided into superpixels (*Image courtesy of [9]*)



$$H = \begin{matrix} & v1 & v2 & v3 & v4 & v5 & v6 \\ \begin{matrix} e1 \\ e2 \\ e3 \end{matrix} & \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \end{matrix}$$

Fig. 3: Example of hypergraph with 6 vertices and 3 hyperedges, along with its incidentMatrix  $H$ .

between the segments in each original segmentation and their corresponding superpixels. That is to say, if  $R$  is the set of all segments of a segmentation  $S_i$ , then for each segment  $R_j, j = 1..N$ , there will be one hyperedge representing the set of superpixels covering that particular segment. After all hyperedges are computed, the incident matrix of the hypergraph is known.

Hyperedge weights are computed at the time each single segmentation is processed. The weight should represent the quality of a segment. Many measures exist [10], such as the color error, the entropy of a segment or the (squared) intensity error. The weights of the hypergraph will be used by the hypergraph cut algorithm in order to segment the hypergraph in  $K$  regions. One could easily scale the weights coming from each segmentation, thus giving more importance to one method or another.

## 5 Hypergraph cut

Partitioning an hypergraph into  $K$  regions can be solved via a matrix eigenvalue problem. A new efficient formulation preserving the invariance property <sup>3</sup> has been proposed by Li Pu. This work is to be published soon, so only a brief explanation can be given in this report.

The clustering algorithm first gets an embedding of the vertices in the hypergraph, then uses Kmeans to partition them. Let  $\mathbf{W}$  be a matrix with the hyperedges weights on its diagonal,  $\mathbf{A}$  be the adjacency matrix of the hyperedges <sup>4</sup> and  $\mathbf{D}$  be the diagonal matrix of column sums of  $\mathbf{WA}$ . An ideal partitioning is found by using the eigenvectors corresponding to the  $k$  smallest eigenvalues of the following matrix

$$P_0 = \begin{bmatrix} 2D & -W \\ -WA & 2W \end{bmatrix}$$

By ignoring the smallest eigenvector, one can multiply the transposed incident matrix of the hypergraph with the eigenvector matrix. Running kmeans would then give an ideal partitioning and getting the final segmentation is finally achieved by mapping each superpixel to its cluster.

The main advantages of using hypergraphs instead of graphs are that the resulting cut does not depend on the size of the clusters and that unbalanced clusters are avoided.

## 6 Summary and examples

The procedure leading to a segmentation is summarized in algorithm 5. As an example, consider figure 4. First, 4 different segmentations are computed and combined into superpixels. Each color represent one superpixel. A hypergraph cut with  $K = 3$  is eventually made, leading to a nice segmentation result with 3 regions.

<sup>3</sup> The invariance property of a clustering objective function holds whenever the size of hyperedges are not taken into account, but only the connectivity of the hypergraph and the hyperedges weights.

<sup>4</sup> An element  $(i, j)$  is set to 1 if two different hyperedges  $i$  and  $j$  share at least a common vertex

**Algorithm 5** Segmentation via hypergraph learning

---

Given  $N$  segmentations  $S = S_1 \dots S_M$   
 Compute the intersection of all segmentations  
 $I \leftarrow (((S_1 \cap S_2) \cap S_3) \cap S_4) \dots \cap S_M$   
 Initialize *incidentMatrix* with zeros  
**for all** segmentation  $S_i$  in  $S$  **do**  
   Let  $R$  be the set of all segments in segmentation  $S_i$   
   **for all** segment  $R_j, j = 1 \dots N$  in  $R$  **do**  
     Add one hyperedge (= line) to the *incidentMatrix* with the value 1 in  
     column  $k$  if superpixel  $k$  is part of segment  $R_j$   
     Assign the weight of segment  $R_j$  to this new hyperedge  
   **end for**  
**end for**  
 Cut the hypergraph into  $K$  regions  
 Recover the image segmentation from the hypergraph cut

---

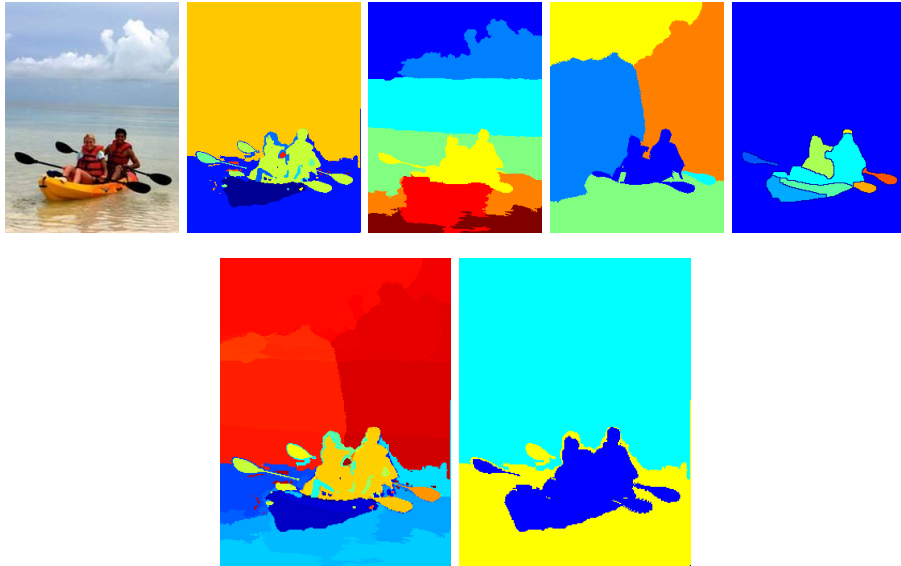


Fig. 4: Example of segmentation merging and hypergraph segmentation. On top, the original image with its segmentations (Kmeans, Meanshift, Normalized cut and Contour/Hierarchical segmentations). On the bottom, the superpixels and the resulting segmentation via hypergraph cut.

Field Name	Description
pixelMapping	The superpixels coming from the intersection of all segmentations
incidentMatrix	The incident matrix of the hypergraph
edgeWeights	The list of all hyperedge weights
trueSegments	The ground truth segmentation(s) if existing
vertexNames	An array containing all vertices names
edgeNames	An array containing all edge names
isSimpleGraph	Logical value, set to 1 for simple graphs, 0 for hypergraphs
graphName	The name of the hypergraph
notes	Some optional notes

Tab. 2: Summary of the fields in a hypergraph structure

## 7 Matlab code

Some important details about the Matlab implementation are presented here. For more information, refer to the comments in the code.

### Simple segmentations

As written in part II, the basis Matlab code for simple methods has been downloaded from [tool1, tool2, tool3, tool4]. The code can be found in the `codes_others` directory. As already mentioned, I've extended some methods and I've created functions that share a common format for each of these methods. The Matlab code can be found in the folder `codes/seg`. This folder also contains a function `assign_weights` that assigns a weight to each original segment. The user can choose between three measures: the entropy of a region, the intensity error or the squared intensity error. From experience, the squared intensity error is the one that works best.

### Hypergraph creation and cut

The code for segmentation merging, hypergraph creation and cut is located in the `codes/hypergraph` directory.

A hypergraph is saved as a structure containing many field. Along with the incident matrix, the hyperedge weights and the superpixels, some other useful fields are saved to a hypergraph structure. For instance, if a ground-truth segmentation exists, it is saved to the `trueSegments` field. Table 2 lists all fields of such a structure.

### Graphical user interface and recoloring

A Matlab GUI (Graphical User Interface) has been created in order to play with simple algorithms and with the hypergraph method. Its organisation is the following (see figure 5 : on the left-hand side, each simple method can be used according to its parameters. The original image and its ground truth segmentation (if existing) are displayed on the top right corner. The rest of the right-hand side is dedicated for the hypergraph method and for the evaluation against a ground-truth image : the superpixels are first displayed, followed

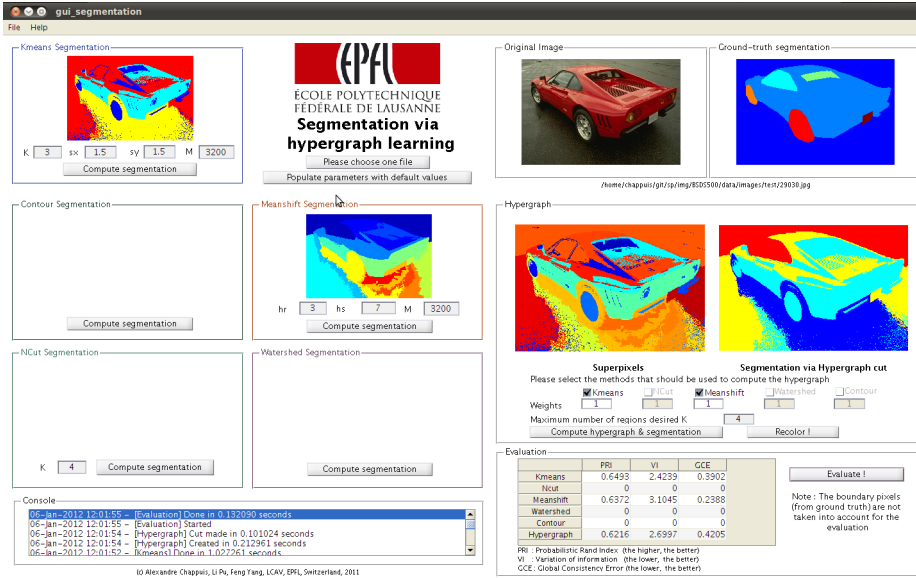


Fig. 5: Matlab GUI

by the segmentation resulting from the hypergraph cut. In order to use the new method, the user must select at least two methods with the checkboxes and indicate the desired maximum number of regions for the hypergraph cut. He/She can also assign a weight for each method, before the intersection is done. What's more, a console is located at the bottom left corner of the GUI. Error messages as well as feedback is written to the console for each event.

Some menus are proposed to the user, so that he/she can save the images or the hypergraph to an output directory. The help menu is intended to give some hints about the parameters needed for each method.

An interactive recoloring process is also available for the segmentation results via hypergraph cut. The user has to press the "Recolor!" button, select the regions he/she wants to recolor and eventually choose a new color. The recolored image would then be displayed in a pop-up window.

Last but not least, the GUI can be started by calling the `gui_segmentation` function from the `codes/gui` directory.

## 8 Evaluation

Evaluating a segmentation algorithm is not as simple as it might appear. Firstly, the notion of segmentation is subjective. Not only human beings perceive images and scenes in different ways, but applications have different usages and requirements. Hence, there is no way to say that one perception is the right one. Some evaluation indices have been thought to overcome this problem, *i.e.* they evaluate a segmentation against one or multiple ground-truth images, that have been labelled by different human subjects.

Secondly, many methods have some input parameters that must be chosen. Evaluating supervised algorithms with the entire space of parameters is infeasible.



ble, even with greatest processing power. That's why only a reasonable subset of parameter values must be chosen.

Before evaluating the algorithm, a choice had to be made regarding the dataset to use. Two well-known datasets are the BSD500 [tool4] and the Voc2011 [11] datasets. The former contains 5 ground-truths per image and tackles only the problem of segmentation, whereas the latter contains only 1 ground-truth per image and targets more problems, for example object classification. Both datasets are available within the created Matlab GUI. However, batch evaluation has only been carried out on the BSD500 dataset.

In the following, the first subsection tries to define what is a good segmentation. Three evaluation measures are then presented: the probabilistic Rand Index (PRI), the Variation of Information (VI) and the global consistency error (GCE). In the final subsection, each simple algorithm along with the new hypergraph approach are evaluated against the BSD500 test images.

## What is a good segmentation?

Defining what is a good segmentation is not trivial at all; one has first to know what is a good segment. Haralick gave already in 1985 some important rules to follow [12] :

1. Regions of an image segmentation should be uniform and homogeneous with respect to some characteristic such as gray tone or texture.
2. Region interiors should be simple and without many small holes.
3. Adjacent regions of a segmentation should have significantly different values with respect to the characteristic on which they are uniform.
4. Boundaries of each segment should be simple, not ragged, and must be spatially accurate.

Since then, much progress has been made toward objective evaluation, see for example [10] for a survey of unsupervised methods and [13] for a discussion about a fundamental question, "What is a good segment?". The measures described below are intended to compare one segmentation against a ground-truth. In order to be more objective, the evaluation can easily be averaged over a set of ground-truths for each image.

## Probabilistic Rand Index

**Definition 3.** *The **Probabilistic Rand Index (PRI)** [14] counts the fraction of pairs of pixels whose labellings are consistent between the computed segmentation and the ground truth, averaging across multiple ground truth segmentations to account for scale variation in human perception. (source : [tool5])*

Once normalized, the value of the PRI lies in the interval  $[0, 1]$ . The closer to 1, the better will be the segmentation.

## Variation of Information

**Definition 4.** *The **Variation of Information** (VI) metric [15] defines the distance between two segmentations as the average conditional entropy of one segmentation given the other, and thus roughly measures the amount of randomness in one segmentation which cannot be explained by the other. (source : [tool5])*

As for the GCE, the closer the VI score to 0, the better !

## Global Consistency Error

**Definition 5.** *The **Global Consistency Error** (GCE) [16] measures the extent to which one segmentation can be viewed as a refinement of the other. Segmentations which are related in this manner are considered to be consistent, since they could represent the same natural image segmented at different scales. (source : [tool5])*

The closer the GCE score will be to 0, the better should be the segmentation.

## Results

Code from [tool5] has been used to compute the PRI, GCE and VI indices. The results displayed in this subsection are averaged over 5 ground-truths for each image, and over the 200 images located in the `test` directory of the BSD500 dataset. The processing time PTIME has also been considered : it's the time in seconds taken by a 3Ghz Quad-Core AMD machine to compute a segmentation.

Table 3 displays the results for the simple methods. The Meanshift method seems to be the best one, both in terms of PRI and processing time. Kmeans is also fast, especially when the number of regions  $K$  is low. Its quality is not too bad when  $K = 4$ . The contour detection and hierachical segmentation method (abbreviated as *contour*) beats all others regarding the GCE and VI, but is really slow. Finally the Normalized cut lies in the middle range. For all methods, parameters have been arbitrarily fixed with reasonable values. Assigning one set of fixed parameters per image (or even better, deducing them from the image) has not been investigated.

Table 4 lists the results for the new hypergraph method with fixed parameter `hypergraph_K = 7`. Many experiments have been carried out, by combining various methods. In this table, the time added to generate the hypergraph, do the cut and provide the segmentation is summarized in the **Added PTIME** column. In all cases, the overhead is very low. However, we don't get significant improvements by using this new method with the chosen parameters.

I've also tried to set the `hypergraph_K` in a more dynamic way, for example by taking the minimum, maximum or the average number of segments in the combined segmentations. By using the ceiling of the average, both the PRI and GCE are slightly improved for some methods, but it's still not better than existing segmentation algorithms.

Results with the average are listed in table 5.

Due to a lack of time, the evaluation could not be done with other set of parameters and every possible combination. Nevertheless, the functions to process the sets of images and evaluate them are ready to be used in the `codes/evaluation` directory. Although the averages are not really encouraging, the new method worked better than all others for some images. Perhaps some more adaptive parameters could be used for both simple method parameters and the hypergraph  $K$  parameter, cutting the hypergraph into  $K$  regions.

In spite of those relatively "bad" results, a practical and functional application has been created with this new approach : an Android application providing semi-recoloring on region of interest. Details are given in the following part.

Method	PRI	VI	GCE	PTIME
Kmeans4	0.7523	2.4924	0.3209	1.7458
Kmeans10	0.6418	2.7090	0.3191	10.1133
Ncut10	0.7429	2.6013	0.3136	101.5119
Meanshift	0.7817	2.4802	0.2203	1.3102
Contour	0.7604	1.6350	0.1511	239.6692

Tab. 3: Evaluation results for simple methods. Kmeans $X$  stands for Kmeans with  $K = X$ ,  $s_x = 1.5$ ,  $s_y = 1.5$ ,  $M = 3200$ . Ncut $X$  stands for the normalized cut with  $K = X$ . Meanshift has been computed with fixed parameters  $h_s = 3$ ,  $h_r = 7$ ,  $M = 3200$

Method	PRI	VI	GCE	Added PTIME
Kmeans4/Ncut10	0.7474	2.5852	0.3338	0.2878
Kmeans10/Meanshift	0.7445	2.5471	0.3156	0.3073
Kmeans4/Meanshift	0.7635	2.5065	0.3129	0.3226
Meanshift/Ncut10	0.7375	2.5728	0.3437	0.3294
Kmeans4/Meanshift/Ncut10	0.7586	2.4645	0.3153	0.5799

Tab. 4: Evaluation results for hypergraph method. The hypergraph was cut into 7 regions to produce a segmentation. The 'X/Y' operator means that methods X and Y have been combined.

Method	PRI	VI	GCE	Added PTIME
Kmeans4/Ncut10	0.7503	2.6684	0.3246	0.2574
Kmeans10/Meanshift	0.7476	2.7628	0.2937	0.3297
Kmeans4/Meanshift	0.7625	2.6294	0.2947	0.3194
Meanshift/Ncut10	0.7499	2.7773	0.2853	0.3256
Kmeans4/Meanshift/Ncut10	0.7599	2.5780	0.3110	0.5385

Tab. 5: Evaluation results 2 for hypergraph method. The hypergraph was cut into  $K$  regions where  $K$  is the mean of the number of regions in the combined segmentations. The 'X/Y' operator means that methods X and Y have been combined.

## Part IV. Android application

An Android application in the second part of this semester project, in order to demonstrate a practical example of the previous work. People love playing with pictures taken with their Iphone or Android phone and there exist many application in the market, such as *ColorTouch* or *ColorSplash Photo* that let the user recolor one part of an image in a funny way, the original image being turned into grayscale. However, none of them (as of November 2011) is able to provide semi-automatic recoloring on a region of interest. That is to say, the user has to go over the whole region they want to recolor with their fingers, including the border. This is quite time-consuming and having a precise recolored region is not always possible. An intelligent application is presented in this part, where the recoloring process takes advantage of multiple segmentations of an image. The user will just have to tap on an object in order to recolor it entirely! In this part, the specification is first described, followed by some relevant implementation details. Finally, some examples and possible extensions are presented.

### 9 Specification

#### 9.1 Overview

The main goal was to demonstrate the use of hypergraphs for image segmentation, while having fun with images. Figure 6 shows the big picture of the process, at a very high level. The segmentation part is not done on the cell phone, as it requires quite a lot of processing. For that purpose, a server serves requests, computes the different segmentations and sends back the hypergraph to the user. Then, the client just has to recolor the segments he/she likes, either with the original color, or with a custom color. One important fact is that the client just has to communicate with the server once! What's more, an image is always resized and compressed on the phone, before it is sent over the channel, so that costly 3G communication is reduced if *WIFI* is not enabled.

#### 9.2 Application

The application requires two different screens, one for choosing an image, let's call it the home screen, and another for the recoloring part. The following subsections lists the requirements of each screen. See figure 7 for a capture of each screen.

##### Home Screen

A list of actions is proposed to the user:

- Pick a picture from the gallery
- Take a picture with the embedded camera
- Ping the server in order to test connectivity

When a picture is selected, a loading symbol will appear and the picture will be first resized to a maximum weight of *100Kb*, and maximum dimensions

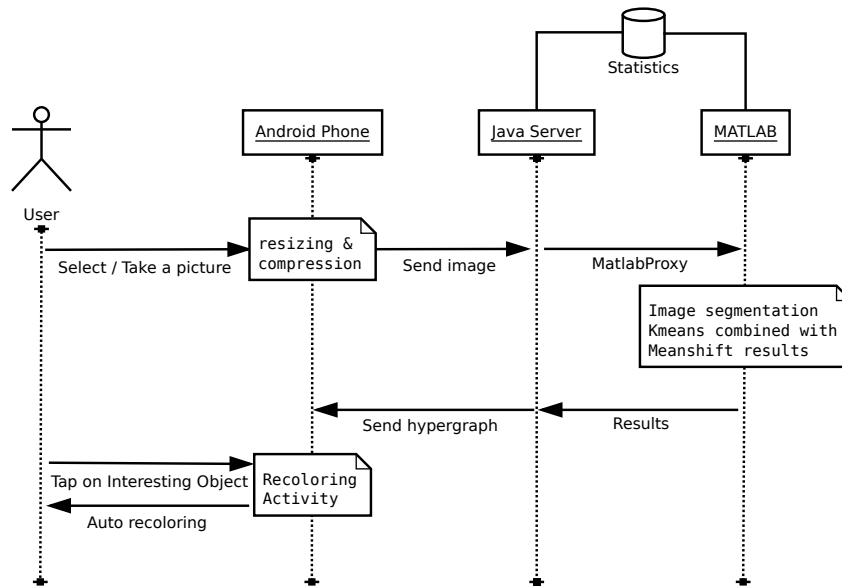


Fig. 6: Overview of the recoloring process

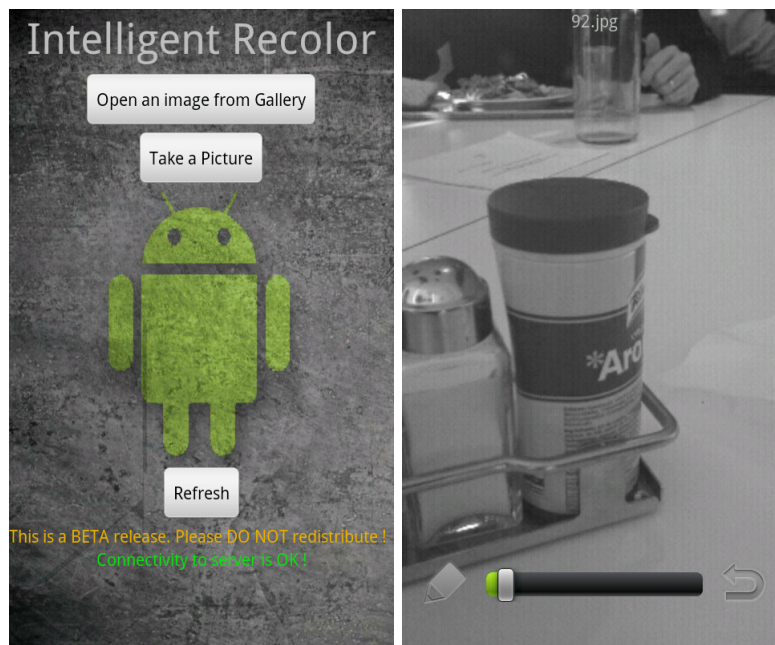


Fig. 7: Home and recoloring screens

of  $400 \times 400$  pixels, then sent to the server. Feedback is given to the user. The application jumps to the recoloring screen as soon as the phone successfully receives results from the server. Would the segmentation fail or any other error occur, a message would then be displayed. The following menus should also be available:

1. **Help:** display a dialog with some hints and explanations about the application
2. **About:** display a dialog with copyright and contact information
3. **Exit:** quit the application

### Recoloring Screen

The original image is displayed in grayscale. It is zoomable (multi-touch) and movable. There are three buttons stacked on top of the image:

1. A color picker. A new color or the original color can be selected
2. A seekbar that controls the size of the expansion. It defines the number of superpixels that must be recolored, starting from the point where the user tapped the image (seed). More explanations about the expansion can be found in section 10.1.
3. A reset button: it clears all recolored segments

Three more menus should be added to this screen:

1. **Home:** Go back to the Home screen
2. **Save:** Save the image to the external memory in JPEG format
3. **Share:** export the image to other applications, such as Facebook, Flickr or Emails

## 9.3 Server

The server has to handle clients' requests. It should be able to interact with Matlab and provide a results 100% of the time. It also collects some statistics: the IP and port of the client connection, the timestamps of a request (Start & End date) and the results of the segmentation. For privacy purposes, the results for each image are kept on the server for a fixed amount of time, then deleted. The number of history days should therefore be customizable.

## 10 Implementation details

Some comments on tricky parts of the implementation are presented here below. For a quick reference, Figure 8 displays an overview of all important Java classes used in the application. Refer to the comments of the code if you feel like learning more about one part of the implementation.

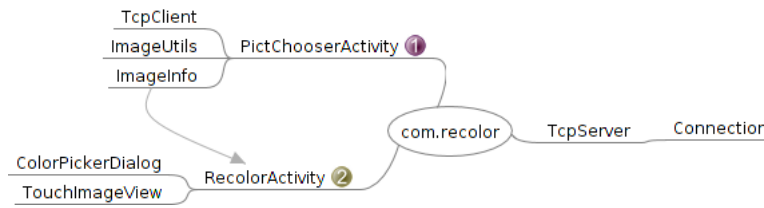


Fig. 8: Java classes

## 10.1 Activities

### PictChooserActivity

The home screen is implemented in the `PictChooserActivity` class. On startup, a connectivity test to the server is made. If it fails, a warning message will be shown to the user, saying that the server might be down. Once the user has chosen or taken a picture, a thread is started. It will resize and compress the image, initiate a TCP connection to the server, send the image and get the resulting segmentation. The data and messages exchanged between the server and the client form the protocol displayed in figure 9.

Error messages will be displayed as `Toast` messages, which are short duration notifications that pop up on the window, whereas feedback to the user is given via a `ProgressDialog`. In order to pass messages (feedback or error) from the `ProcessImageThread` to the main thread, a `Handler` object is used. The special message *next* is sent when the `processImage()` function successfully finishes, thus allowing the application to jump to the recoloring screen.

All information needed by the recoloring screen is stored in an `ImageInfo` object. The most important fields are the `incidentMatrix`, `edgeWeights` that describe the hypergraph and `pixelMapping` that maps each pixel to its super-pixel. Refer to part 4 for more information about those fields. In order to use Matrix operations, the Jama Toolkit [tool6] has been used extensively.

### RecoloringActivity

The first thing that this activity does is displaying the chosen image in grayscale embedded in a `TouchImageView` object. Before any modification is made to the image, an original copy is saved so that it can be restored at any time with the reset button.

I have used and extended the multi-touch image view toolkit available at [tool7] so that touch events would call the recoloring function with the good seed point, without regard of the actual zoom and displacement of the image. For that purpose, I had to find the relative coordinates of the image with respect to the current view and add one listener listening to click events. One click event is triggered when the user touches the image and the displacement is smaller than 3 pixels.

The color chooser has been slightly adapted from the code found at [tool8]. It is shown in figure 10. As the user wants to recolor some segments of the image while keeping its visual content, only the hue value of a color can be used. The



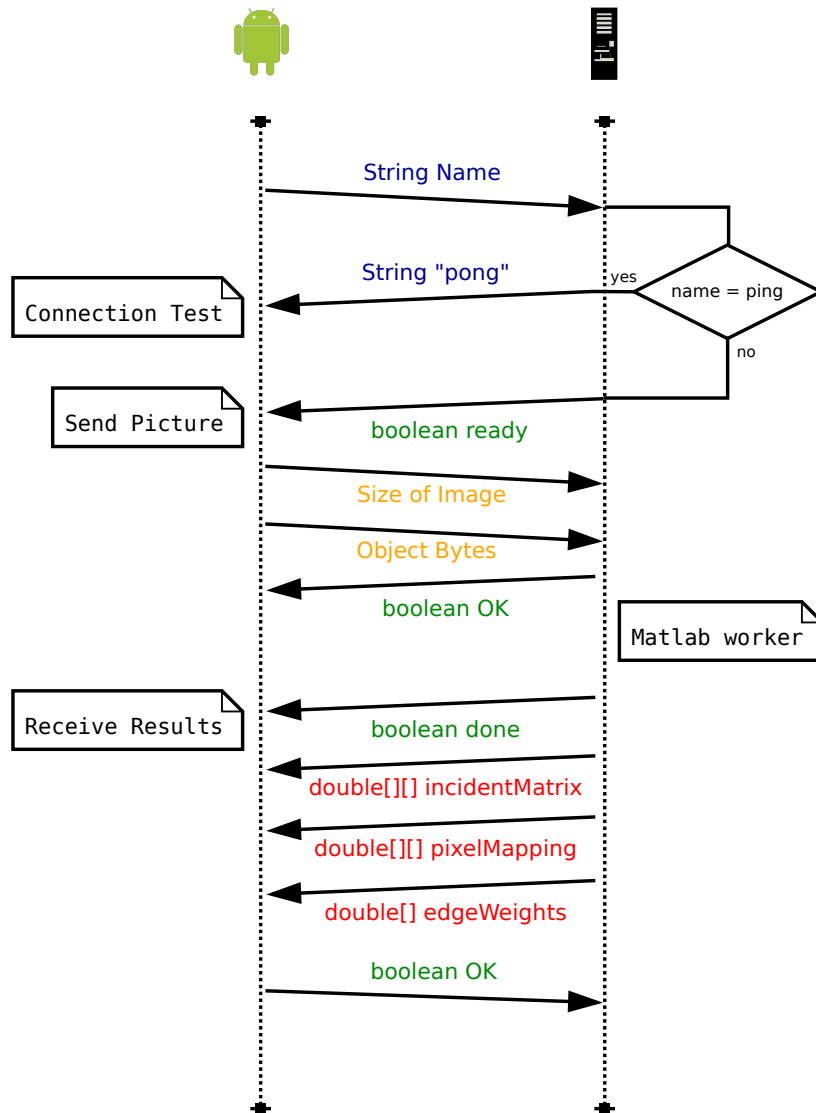


Fig. 9: Protocol used between the Android phone and the Java server

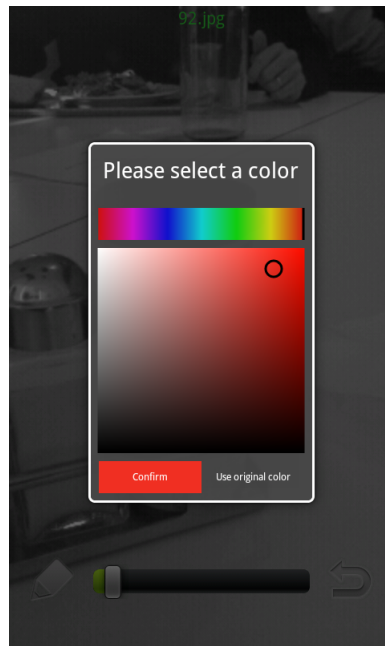


Fig. 10: The colorpicker used to choose a color.

intensity is therefore unchanged and the saturation can be slightly enhanced or reduced via the hard-coded field `SATURATION_ENHANCEMENT`. By default, there is no saturation enhancement. If the user presses the button "Keep original color", then the `currentHue` field is set to `-1`. The recoloring process would then keep the color of the original pixels.

The image is recolored when the user taps on the image or when the value of the `SeekBar` is changed. In the former, the seed (location  $(x,y)$  on the image) is first given to an algorithm<sup>5</sup> that will assign a weight to each superpixel. The closer to 1, the closer the superpixel will be to the seed. The superpixels with the `nbSuperPixels` highest weights will be selected for recoloring and thanks to the `pixelMapping` matrix, all pixels that matches those selected superpixels will be recolored. At the end, the current figure is replaced with the modified image. In the latter, another listener was needed in order to react to changes of the `SeekBar` value. Once it is changed, the number of superpixels is changed accordingly (the minimum being 2) and the previous seed and expansion are used to recolor the image.

## 10.2 Server and Matlab interaction

The Java server listens on port 12345 and starts a new `Connection` for each client. Then, it follows the protocol described in figure 9. The Matlab interaction has been made possible thanks to the `Matlabcontrol` library [tool9]. Once

<sup>5</sup> This expansion algorithm is based on the work presented in section 5.

the received image has been successfully written to the disk, a Matlab function is called via a MatlabProxy:

```
[pixelMapping incidentMatrix edgeWeights] =
processImageForAndroid(filename,myresultsdir)
```

This function computes two segmentations of the given image: Kmeans with  $K=5$ ,  $sx=1.5$ ,  $sy=1.5$ ,  $M=3200$  and Meanshift with  $hs=3$ ,  $hr=7$ ,  $M=3200$ . The segmentations are combined and the hypergraph is generated.

The results are finally converted back to Java's types via a `MatlabTypeConverter` object and sent to the device.

## 11 Examples

On the top row of figure 11, one can see the role of the expansion: increasing the number of superpixels will expand the recolored region to similar segments. On the bottom row, the difference between a recoloring process with the original versus a new color is shown. One can easily turn a ripe mandarin into green colors when the seed is carefully chosen.

## 12 Design choices, limitations and possible extensions

- This application has been developed for Android 2.2 (Froyo) and tested on a HTC desire. It should theoretically work on more recent devices and for other brands.
- The choice of combining the Kmeans and Meanshift methods seems to be reasonable, as they both always converge. From experience, both methods are rather quick and their intersection is always rich, meaning that the expansion algorithm would benefit from having more superpixels in the hypergraph. The chosen parameters are quite general and should work for most images.
- A server is used to compute the different segmentations and the resulting hypergraph. There is no problem with this approach for a demo application like the one developed. However, this is clearly not scalable if the number of client increases. So, if this application is to be published one day, the developer has either to include the segmentation algorithms (Kmeans and Meanshift) on the device, or use multiple servers with load-balancing. The former might be preferred for privacy reasons, but would maybe fail because of the small memory and processing power pocket devices have compared to a real computer.
- The application has not been released yet in the Android Market. So, in order to install it, the user has to enable the "Unknown sources" checkbox in the application settings of the device, and then install the application from the "apk" installation archive.
- There are some hard-coded constants that could be replaced in the future by a Preference Screen, for instance the name and port of the server or the maximum dimensions of the images.

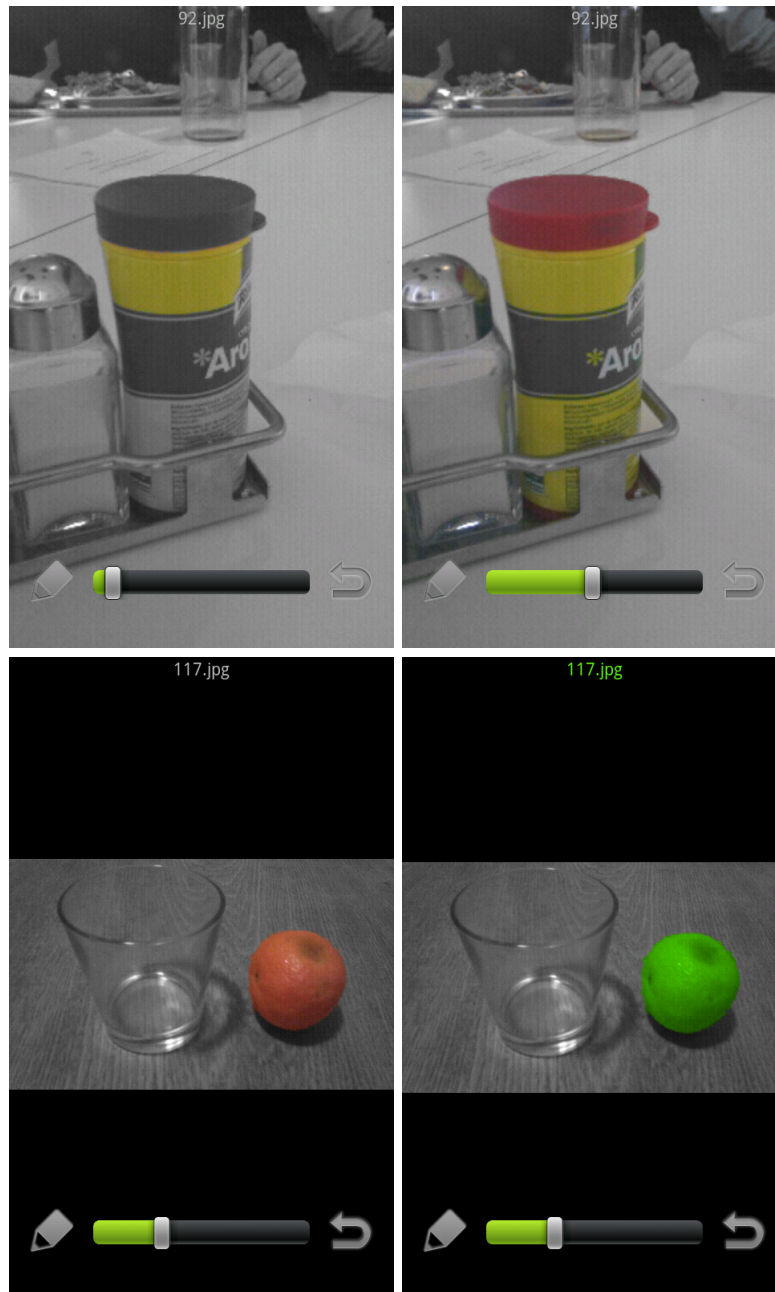


Fig. 11: Example of expansion and color change. In the top row, the number of superpixels is increased. In the bottom row, a new color is chosen in order to recolor one ripe mandarin in green tones.

- 
- In order to improve the accuracy of the application, one could easily extend it with user feedback. The user could for example rate the recoloring process with "stars".

## Part V. Conclusion

The developed Android application has shown how a new hypergraph method for segmentation can be applied to recolor images in an intelligent manner. Thanks to the segmentation information computed by a server, an image can be divided into a set of superpixels and summarized in a hypergraph structure. Expanding the recolored superpixels with the help of the hypergraph and from the user seed subsequently allow objects to be recolored automatically. The new segmentation approach does not beat existing work yet, but has practical usage with only low computational overhead.

I've had much pleasure to work on this project; I could first learn basic segmentation algorithms, see how hypergraphs can be used to solve clustering problems, and finally learn the basics of application development for Android phones. I'm pretty sure that extending this work would be an exciting task.

## Part VI. Future work

This work could easily be continued and improved with more time available. After some small changes, the Android Application could for example be published on the Android Market, its user interface could be extended and the code for the recoloring process could slightly be optimized. Some ideas of extensions have already been given in section 12.

As for the new hypergraph approach, more evaluation thoughts and experiments need to be done. One could evaluate the segmentation algorithms with other sets of parameters, try to find better combinations or try to estimate parameters from the image. There exist much work on clustering improvement techniques and some ideas of [17] or [18] could for example be applied to the problem of image segmentation.

## Acknowledgements

I'd like to thank my supervisors Li Pu and Feng Yang for their good advice and help given during the semester. Special thanks also to Jonathan, Oriane and Valter for their nice feedback regarding the beta version of the Android application.

## Appendix

A DVD containing all the Matlab code, the Java code for the application, the references and this report in tex+pdf format. See the **README** files in the directory for more information about the organization of the code.

## References

- [1] Y J Zhang. An overview of image and video segmentation in the last 40 years. *Advances in Image and Video Segmentation*, pages 1–15, 2006.
- [2] Hongliang Li and King Ngi Ngan. Image/video segmentation: Current status, trends, and challenges. In King Ngi Ngan and Hongliang Li, editors, *Video Segmentation and Its Applications*, pages 1–23. Springer New York, 2011.
- [3] M. Luo, Yu-Fei Ma, and Hong-Jiang Zhang. A spatial constrained k-means approach to image segmentation. In *Information, Communications and Signal Processing, 2003 and the Fourth Pacific Rim Conference on Multimedia. Proceedings of the 2003 Joint Conference of the Fourth International Conference on*, volume 2, pages 738 – 742 vol.2, dec. 2003.
- [4] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24:603–619, May 2002.
- [5] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. In *Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, CVPR '97, pages 731–, Washington, DC, USA, 1997. IEEE Computer Society.
- [6] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33:898–916, May 2011.
- [7] William Brendel and Sinisa Todorovic. Segmentation as maximum-weight independent set. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 307–315. 2010.
- [8] Yuchi Huang, Qingshan Liu, Fengjun Lv, Yihong Gong, and D.N. Metaxas. Unsupervised image categorization by hypergraph partition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(6):1266–1273, june 2011.
- [9] X. Ren and J. Malik. Learning a classification model for segmentation. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 10 –17 vol.1, oct. 2003.
- [10] Hui Zhang, Jason E. Fritts, and Sally A. Goldman. Image segmentation evaluation: A survey of unsupervised methods. *Comput. Vis. Image Underst.*, 110:260–280, May 2008.
- [11] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes challenge 2011 (voc2011) results. <http://www.pascal-network.org/challenges/VOC/voc2011/workshop/index.html>.
- [12] Robert M. Haralick and Linda G. Shapiro. Image segmentation techniques. *Computer Vision, Graphics, and Image Processing*, 29(1):100 – 132, 1985.

- 
- [13] Shai Bagon, Oren Boiman, and Michal Irani. What is a good image segment? a unified approach to segment extraction. In *Proceedings of the 10th European Conference on Computer Vision: Part IV*, pages 30–44, Berlin, Heidelberg, 2008. Springer-Verlag.
  - [14] R. Unnikrishnan, C. Pantofaru, and M. Hebert. A measure for objective evaluation of image segmentation algorithms. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '05) - Workshops - Volume 03*, pages 34–, Washington, DC, USA, 2005. IEEE Computer Society.
  - [15] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: is a correction for chance necessary? In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09*, pages 1073–1080, New York, NY, USA, 2009. ACM.
  - [16] David R. Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. Technical Report UCB/CSD-01-1133, EECS Department, University of California, Berkeley, Jan 2001.
  - [17] Greg Hamerly and Charles Elkan. Learning the k in k-means. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.
  - [18] Lihi Zelnik-Manor and Pietro Perona. Self-tuning spectral clustering. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1601–1608. MIT Press, Cambridge, MA, 2005.

## Tools

- [tool1] Yi Cao. Efficient k-means using jit (matlab code). [http://www.cs.ucf.edu/~vision/Code/k\\_means\\_JIT.zip](http://www.cs.ucf.edu/~vision/Code/k_means_JIT.zip), March 2008.
- [tool2] Chris M. Christoudias and Bogdan Georgescu. Code for the edge detection and image segmentation system. <http://coewww.rutgers.edu/riul/research/code/EDISON/index.html>, 2003.
- [tool3] Stella Yu Timothee Cour and Jianbo Shi. Normalized cuts segmentation code, for matlab. <http://www.seas.upenn.edu/~timothee/software/ncut/ncut.html>, January 2004.
- [tool4] C. Fowlkes P. Arbelaez, M. Maire and J. Malik. Contour detection and image segmentation resources. <http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/resources.html>, 2011.
- [tool5] John Wright and Allen Y. Yang. Image segmentation benchmark indices package. [http://www.eecs.berkeley.edu/~yang/software/lossy\\_segmentation/](http://www.eecs.berkeley.edu/~yang/software/lossy_segmentation/), 2007.



- [tool6] Jama : A java matrix package. <http://math.nist.gov/javanumerics/jama>, 2005.
- [tool7] Mike Ortiz. Panning and pinch zoom added to android imageview. <https://github.com/MikeOrtiz/TouchImageView>, September 2011.
- [tool8] A photoshop like color picker for your android application. <http://www.yougli.net/android/a-photoshop-like-color-picker-for-your-android-application>, September 2010.
- [tool9] A java api to interact with matlab. <http://code.google.com/p/matlabcontrol>, 2011.

## Images

- [img1] Superpixels image. [http://www.cs.sfu.ca/~mori/research/superpixels/figs/sp\\_996.jpg](http://www.cs.sfu.ca/~mori/research/superpixels/figs/sp_996.jpg).



Wooooow, that's the end !  
Thanks for having read this report :-)