3
SEGMENTATION ALGORITHMS

This chapter presents the four segmentation algorithms considered for this study. Only a brief description of each one is given. More details can be found in the papers cited in the following sections.

3.1.
Mean-Shift Segmentation (*MS*)

Proposed by Comaniciu & Meer (2002), Mean-Shift algorithm is a robust method of finding local extrema in the density distribution of a data set. It is an easy process for continuous distributions. However, it is not that simple when it has to deal with discrete data sets, as it is often the case in most Computer Vision problems. It is said "robust" in statistical sense. That is, mean-shift ignores outliers in the data.

The mean-shift algorithm works as follows:

1. Choose a search window, which includes selecting:
    - its initial location;
    - its type (uniform, polynomial, exponential, or Gaussian);
    - its shape (symmetric or skewed, possibly rotated, rounded or rectangular);
    - its size (extent at which it rolls off or is cut off).
2. Put a window for each pixel in the image for initialization.
3. Compute the window's center of mass, which is calculated as the average feature vector of the pixels in the window, whereby the feature vector encompasses spatial as well as appearance components.
4. Move the center of the window to the center of mass.
5. Repeat steps 3 and 4 until the window stops moving or an stopping criterion is reached (e.g. number of iterations, variation of the center

of mass, etc). During this process, some windows will move toward each other till they merge together into one single window.

In more formal terms, the mean-shift algorithm is related to the discipline of *kernel density estimation*, where "kernel" is a function that has a local focus (e.g., a Gaussian distribution). With enough appropriately weighted and sized kernels located at proper points, one can express a distribution of a data entirely in terms of those kernels. Mean-shift diverges from kernel density estimation in that it only relies on gradient estimates (direction of change) of the data distribution. When this change is 0, we are at a stable (though perhaps local) peak of the distribution. There might be other peaks nearby or at other scales.

Let's take a Gaussian kernel as example. This is defined as:

$$K(\boldsymbol{x};\mu,\Sigma) = \frac{(2\pi)^{-d/2}}{|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\mu)^T\Sigma^{-1}(\boldsymbol{x}-\mu)\right) \quad \textbf{(1)}$$

where $\boldsymbol{x}$ will be a d dimensional feature vector and μ and Σ are the mean and the covariance matrix respectively. Considering a zero mean distribution and a univariate distribution,. the Gaussian kernel will take the form:

$$K(\boldsymbol{x};\mu,\sigma^2) = \frac{(2\pi)^{-\frac{1}{2}}}{\sigma} \exp\left(-\frac{1}{2\sigma^2}(\boldsymbol{x})^2\right) \quad \textbf{(2)}$$

where σ is the standard deviation. In this way, the probability density function (pdf) will be given by:

$$P(\boldsymbol{x}) = \frac{1}{\eta}\sum_{i=1}^{\eta} K(\boldsymbol{x}-\boldsymbol{x_i}) = \Omega\sum_{i=1}^{\eta} \kappa\left(\left\|\frac{\boldsymbol{x}-\boldsymbol{x}_i}{\sigma}\right\|^2\right) \quad \textbf{(3)}$$

where η is the number of gaussians, $\Omega = \frac{(2\pi)^{-1/2}}{\eta\sigma}$ and $\kappa(\boldsymbol{u}) = \exp\left(-\frac{1}{2}\boldsymbol{u}\right)$. Hence, as we are seeking the closest maximum of $P(\boldsymbol{x})$, which occurs where the gradient is zero.

$$\nabla P(\boldsymbol{x}) = \Omega \sum_{i=1}^{\eta} \nabla K(\boldsymbol{x} - \boldsymbol{x_i}) \quad \textbf{(4)}$$

Solving eq. (4) yields:

$$\overline{\boldsymbol{x}} = \frac{\sum_i \boldsymbol{x}_i g\left(\left\|\frac{\boldsymbol{x}_i - \overline{\boldsymbol{x}}}{\sigma}\right\|^2\right)}{\sum_i g\left(\left\|\frac{\boldsymbol{x}_i - \overline{\boldsymbol{x}}}{\sigma}\right\|^2\right)} \quad \textbf{(5)}$$

where $g = \frac{d}{du}\kappa(\boldsymbol{u})$ and $\overline{\boldsymbol{x}}$ represents the position of the center of mass. A common stopping criterion is when $\Delta\overline{\boldsymbol{x}} \approx \mathbf{0}$, where $\Delta\overline{\boldsymbol{x}}$, the so called "mean-shift vector", denotes the displacement of the center of mass between two consecutive iterations. An example ilustrating the iterations of Mean Shift algorithm is presented in Figure 5. In this case, the search window was rectangular.

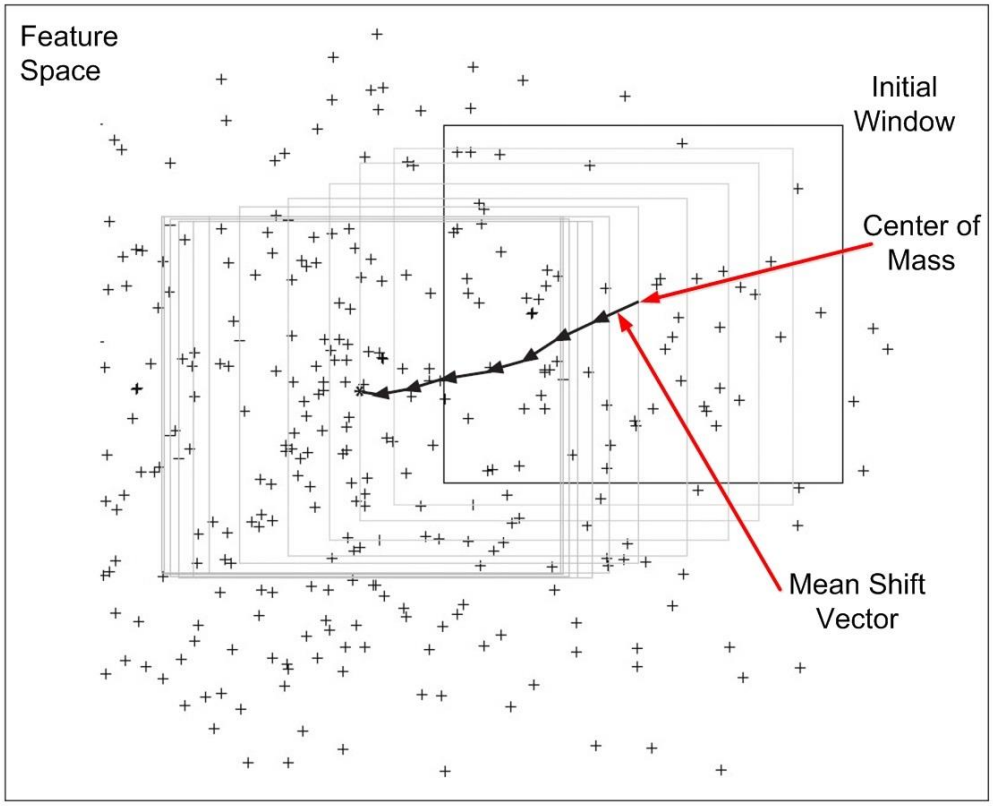

Figure 5: Iterations of Mean Shift algorithm. An initial window is chosen and its center of mass is shifted until it converges (modified from Bradski & Kaehler (2008)).

The implementation used in the experiments was done using the OpenCV Library (Bradski, et al., 2008). It uses two types of features: related position and appearance intensity. As the search window for this implementation was circular, it is necessary to define a radius for each feature (an spatial radius and an spectral radius). The segmentaton is done over a scale pyramid, which allows a fast initial segmentation on the low-resolution image in the highest level of the pyramid. The initial outcome is brought to the next lower level then, the meanshift process is executed again, and so successively until the lowest level of the pyramid. The number of levels is also an input parameter of the algorithm. Therefore, there are three parameters that need to be tuned in this algorithm: spatial and spectral radius, and the number of pyramid levels.

3.2. Graph-based Segmentation (Gb)

Graph-based approaches, for image segmentation, model the image as an undirected graph $G = (V, E)$ where each pixel is a vertex $v_i \in V$ and edges $(v_i, v_j) \in E$ correspond to pairs of neighboring vertices. Each edge $(v_i, v_j) \in E$ has a corresponding weight $w\left((v_i, v_j)\right)$, which is a non-negative measure of the dissimilarity between neighboring elements v_i and v_j. In this specific case, this weight is some measure of the dissimilarity between two pixels connected by that edge (e.g. the difference in intensity, color, motion, location or some other local attribute).

In this context, a segmentation S is a partition of V into components such that each component (or segment) $C \in S$ corresponds to a connected component in a graph $G' = (V, E')$, where $E' \subseteq E$, i.e. a segmentation is induced by a subset of the edges in E. As it is desired to get elements in the same component similar to each other and, elements in different components to be dissimilar, the weights from edges between two vertices in the same component should be low and, weights from edges between two vertices in different components should be high.

In order to define the boundaries between two components in a segmentation, a predicate D is defined (Felzenszwalb & Huttenlocher, 2004). This predicate will compare the inter-component differences to the within component

differences and is thereby adaptive with respect to the local characteristics of the data.

Before defining the predicate D, it is important to introduce some concepts related to graph theory. First of all, a *tree (T)* is considered as an undirected graph in which any two vertices are connected by exactly one edge. Then, *a spanning tree (ST)* is a kind of *tree* that includes all of the vertices and some or all of the edges of a graph. A *minimum spanning tree (MST)* is a kind of *spanning tree* with a total weight less than or equal to the weight of every other spanning tree. Later, *th*e *internal difference (*$Int(C)$*)* of a component $C \subseteq V$ is defined as the largest weight in the *minimum spanning tree* (Cormen et al., 1990) of the component, $MST(C, E)$. That is:

$$Int(C) = \max_{e \in MST(C,E)} w(e) \quad \textbf{(6)}$$

This measure sets a sufficient condition for a given component C to be connected, i.e. a component C remains connected if only edges of weight at least $Int(C)$ are considered. Then, the *difference between* two components $C_1, C_2 \subseteq V$ is defined as the minimum weight edge connecting the two components. That is:

$$Dif(C_1, C_2) = \min_{v_i \in C_1, v_2 \in C_2, (v_1, v_2) \in E} w\left((v_i, v_j)\right) \quad \textbf{(7)}$$

If there is no edge connecting C_1 and C_2, then $Dif(C_1, C_2) = \infty$. This measure reflects only the smallest edge weight between two components. A consideration to take into account is that it is possible to change the definition using the median weight, or some other quantile, in order to make it more robust to outliers. This small change to the segmentation criterion vastly changes the difficulty of the problem.

As said before, the predicate D evaluates if there is evidence for a boundary between a pair of 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, $Int(C_1)$ or $Int(C_2)$. It is necessary to define a threshold

function, which measures how large is $Dif(C_1.C_2)$ with respect to $Int(C_1)$ and/or $Int(C_2)$. Then, Felzenszwalb and Huttenlocher define the predicate as:

$$D(C_1, C_2) = \begin{cases} true, & if\ Dif(C_1, C_2) > MInt(C_1, C_2) \\ false, & otherwise \end{cases} \quad \textbf{(8)}$$

where the *minimum internal difference*, $MInt$, is defined as:

$$MInt(C_1, C_2) = \min(Int(C_1) + \tau(C_1), Int(C_2) + \tau(C_2)) \quad \textbf{(9)}$$

where τ is a threshold function. This threshold function τ controls the degree to which the *difference between* two components must be greater than their *internal differences.*

As for small components, $Int(C)$ is not a good estimate of the local characteristics of the data, for example in the extreme case when $|C| = 1$, $Int(C) = 0$. For that reason, τ must be a function of the size of the component:

$$\tau(C) = \frac{\theta}{|C|} \quad \textbf{(10)}$$

where $|C|$ denotes the size of C, and θ is some constant parameter. This parameter θ sets a scale of observation, a higher value of θ causes a preference for larger components. Smaller components are allowed too if they satisfy the condition defined by eq. (8).

Now, let's turn to the segmentation algorithm, which is closely related to Kruskal's algorithm for constructing a *minimum spanning tree* of a graph (cf. Cormen et al., 1990). Given an input graph $G = (V, E)$, with N vertices and m edges, the output will be a segmentation of V into components $S = (C_1, \dots, C_r)$ following this procedure proposed by Felzenszwalb:

1. Sort edges into non-decreasing order by weight w.
2. Start with a segmentation S^0, where each vertex v_i is in its own component.
3. Repeat step 4 for $q = 1, \dots, m$

4. Construct S^q given S^{q-1} as follows. Let C_i^{q-1} and C_j^{q-1} be components of S^{q-1} containing v_i and v_j respectively. If $C_i^{q-1} \neq C_j^{q-1}$ and $w\left(\left(v_i, v_j\right)\right) \leq MInt\left(C_i^{q-1}, C_j^{q-1}\right)$, then, S^q is obtained from S^{q-1} by merging C_i^{q-1} and C_j^{q-1}. Otherwise, $S^q = S^{q-1}$.

It can be demonstrated that a segmentation S produced by the algorithm obeys the global properties of being neither *too fine* nor *too coarse* (cf. Felzenszwalb & Huttenlocher, 2004), when using the region comparison predicate D defined in eq. (8).

Typically, the edge set E is constructed considering an 8-connected connectivity. The edge weight function, $w\left(\left(v_i, v_j\right)\right)$, is based on the absolute intensity difference between the pixels connected by an edge, as shown in eq. (11).

$$w\left(\left(v_i, v_j\right)\right) = \left|I(p_i) - I\left(p_j\right)\right| \quad \textbf{(11)}$$

where $I(p_i)$ is the intensity of the pixel p_i. Before the edge computation, a Gaussian filter is applied to the image for a slight smoothing, in order to compensate for digitization artifacts.

The implementation of this algorithm is available on the website of the author. The source code was tested and a portable executable was generated and incorporated to the prototype developed. There are two parameters that need to be tuned in this algorithm: the constant θ of the threshold function τ, and the standard deviation σ of the Gaussian smoothing applied at the beginning of the segmentation procedure.

3.3. Region Merging – based Segmentation (Rm)

Proposed by Happ et al. (2013), this algorithm is a variant of Baatz and Schäpe algorithm (Baatz & Schäpe, 2000). It can be considered as a region merging technique.

Initially, each pixel is considered as one segment. Later, in each step, a pair of objects is merged into one larger object. The merging decision is based on local

homogeneity criteria between adjacent segments. Also, a *merging cost* is associated to each possible merge. These costs represent the *degree of fitting*. For each possible merge, the *degree of fitting* is calculated and only if it is smaller than a given scale parameter, the merge is executed.

Then, a merge with a *degree of fitting* inferior to this *scale parameter* will *fulfill the homogeneity criterion*. The segmentation procedure ends when no further merging can be performed.

There are two variants of this basic algorithm depending on the heuristic to find the right object for a merge (Baatz & Schäpe, 2000). The first variant is called **Best Fitting (BF)**. According to this variant, a segment C_1 is merged with a segment C_2, only if C_1 and C_2 are the pair of neighboring segments that best meet the homogeneity criterion. In other words, the homogeneity condition is met concerning the *scale parameter* and the merge has the smallest *degree of fitting* value compared to all possible merges.

The second variant is called **Local Mutual Best Fitting (LMBF).** In this variant, the Best Fitting condition is relaxed to some extent. A merge always occurs if the best fitting condition is mutual between both segments, i.e. if C_1 is the best fit of C_2 and vice versa.

The *merging cost* (f) or *degree of fitting* is composed by a *spectral* (h_{color}) and a *morphological* component (h_{shape}). The relative importance of each one is represented by ω_{color}.

$$f = \omega_{color} \times h_{color} + (1 - \omega_{color}) \times h_{shape} \quad \textbf{(12)}$$

where ω_{color} is a value between 0 and 1. The *spectral component* h_{color} is given by eq. (13**)**, where L is a spectral band and ω_L its respective weight defined by the user, A is the area (in pixels) of a given region. $\sigma_L^{C_1}$, $\sigma_L^{C_2}$ and $\sigma_L^{C_1 \cup C_2}$ are the standard deviations of pixels in regions C_1, C_2 and $C_1 \cup C_2$, which represents the resulting region after the merge (see Figure 6).

$$h_{color} = \sum_L \omega_L \left(A_{C_1 \cup C_2} \times \sigma_L^{C_1 \cup C_2} - \left(A_{C_1} \times \sigma_L^{C_1} + A_{C_2} \times \sigma_L^{C_2} \right) \right) \quad \textbf{(13)}$$

Figure 6: Two possible merging regions, C_1 (blue) and C_2 (red), and the final result $C_1 \cup C_2$ (green).

The *morphological component* is similar to the spectral one, where the per-band standard deviations are replaced by two morphological features, *Smoothness* and *Compactness*, also weighted by a user-defined parameter. Their computation requires measuring the border length of the resulting segment after the merging of two adjacent segments. This operation is computationally expensive and involves a large number of accesses to the global memory in a GPU environment. Taking these issues into consideration, Happ et al. proposed to take two alternative morphological features, *Solidity* and *Compactness* described in Russ (1998), and defined in eq. (14) and (15). *Solidity* is defined as:

$$Sol = \frac{Abox}{A} \tag{14}$$

where A is the segment's area and $Abox$ is the area of its bounding box. Note that the *Solidity*, as the *Smoothness*, is sensitive to the segment's convexity and has its minimum value for rectangular shapes. The second feature, also called *Compactness*, reaches its minimum for circular shapes and is given by:

$$Comp = \frac{dmax}{\sqrt{\frac{4A}{\pi}}} \tag{15}$$

where A is the segment's area and $dmax$ is the length of the major axis of the ellipse with identical second order moment along the axis.

The implementation of this algorithm is available in the web page of the Computer Vision Lab (LVC) from Pontifical Catholic University of Rio de Janeiro (PUC-Rio). The executables for the sequential and parallel versions of the segmentation algorithm, a guide for usage and the necessary DLLs are available there as well.

As there are two heuristics, there are two variations of the algorithm. The variation selected for the experiments was the **LMBF**. There are three parameters that need to be tuned in this algorithm: the *scale parameter*, the *color weight* and the *compactness weight*, which sets the relative importance between *compactness* and *solidity*. Actually, there were three more parameters that were not considered in order to reduce the computational cost. These parameters are the weights assgined to each band (the algorithm works only with three bands). Noticed that they are not so critical as the others; for that reason, they were set with equal values (0.33 for the first band, 0.33 for the second band, and 0.34 for the last band).

3.4. Conditional Random Fields – based Segmentation ($CRFb$)

This algorithm is related to the problem of Semantic Segmentation, which corresponds to a partitioning of pixels into regions that are semantically meaningful to people (Gondra & Xu, 2010). First of all, let´s define semantic segmentation. Then, the following section describes one variant of semantic segmentation which is based on Conditional Random Fields (CRF).

3.4.1. Semantic Segmentation

Most image segmentation algorithms extract regions that satisfy some uniformity (homogeneity) criterion, which is based on low-level data-driven visual features (e.g. color, texture). Those algorithms perform well in narrow domains (e.g. medical images, frontal views of faces), where the variability of low-level visual content is limited. Unfortunately, in broader domains, homogeneous regions do not necessarily (and usually do not) correspond to semantically meaningful units. This is mainly caused by the disconnection

between low-level visual features and high-level semantics, which is commonly referred to as the *semantic gap*.

Semantic segmentation is a supervised learning problem in contrast to low-level unsupervised segmentation.

State-of-the-art semantic segmentation algorithms typically consist of three main components. The first one models the local appearance of objects. Taking into account only the local appearance neglects the influence between adjacent locations and leads to very noisy segmentation outcomes. To address this issue, the second component enforces local consistency of the labeling between locations. However, this is insufficient to capture a meaningful image object as a whole. Also, this disregards contextual information, which may be useful. The third component enforces global consistencies, e.g. at a region or image level (Csurka & Perronnin, 2010).

These three components are generally integrated into a unified probabilistic framework such as Random Fields (RF) or, as in our case, as Conditional Random Fields (CRF). A unified framework enables at training time a joint estimation of the model parameters and ensures at testing time a globally consistent labeling (Yang et al., 2010). In spite of the high computational cost that these models imply, they lead to state-of-the-art results.

3.4.2. Generative vs. Discriminative models

There are two approaches in probabilistic techniques: generative and discriminative models. The first ones are focused on modeling the class-conditional probabilistic density/mass functions (pdfs) of input variables and prior class probabilities. Examples of generative models are: Naïve Bayes (NB), Mixture of Gaussians (GMM), Markov Random Fields (MRF) and Hidden Markov Models (HMM), among others. They are called generative because they make possible to generate synthetic data points from pdfs. The second ones estimate directly posterior probabilities, without modelling either explicitly or implicitly the distribution of inputs and outputs. Examples of discriminative models are: Logistic Regression, Support Vector Machines (SVMs) and CRFs,

among others. Due to the aforementioned characteristics of discriminative models, the utilization of these approaches has been increasing in the last years.

3.4.3. Conditional Random Fields (CRF)

The CRF models commonly used in segmentation are characterized by energy functions defined on unary and pairwise cliques as (Kohli et al., 2008):

$$E(\boldsymbol{x}) = \sum_{i \in \mathcal{V}} \psi_i(x_i) + \sum_{i \in \mathcal{V}, i \in N_i} \psi_{ij}(x_i, x_j) \quad \textbf{(16)}$$

Here, $\mathcal{V}$ corresponds to the set of all images pixel, N_i is the neighborhood of the pixel i defined on this set, which is commonly chosen to have 4- or 8-connectivity. The labels constituting $\mathcal{L} = \{l_1, l_2, \dots, l_k\}$ represent the different image objects. The random variable x_i denotes the labeling of pixel i of the image represented by a d – dimensional feature vector. Every possible assignment of the random variables $\boldsymbol{x}$ (or configuration of the CRF) defines a segmentation.

The unary potential ψ_i is defined as the negative log of the likelihood of a label being assigned to a pixel i, $-\log p(x_i|y_i)$, where y_i represents a feature vector. In our case, they are calculated from spectral values, position, Fourier features and Histogram of Gaussians ($HoGs$). Then, the unary potentials used in this work are given by:

$$\psi_i(x_i) = w_1\psi_{col}(x_i) + w_2\psi_{pos}(x_i) + w_3\psi_{fourier}(x_i) + w_4\psi_{hog}(x_i) \quad \textbf{(17)}$$

where w_1, w_2, w_3 and w_4 are parameters weighting the potentials obtained from spectral values (ψ_{col}), position (ψ_{pos}), Fourier ($\psi_{fourier}$) and Histogram of Gaussians (ψ_{hog}) respectively.

The pairwise term ψ_{ij} takes a form of a contrast sensitive Potts model defined as:

$$\psi(x_i, x_j) = \begin{cases} 0, & if\ x_i = x_j \\ \Gamma(i,j), & otherwise \end{cases} \quad \textbf{(18)}$$

where the function $\Gamma(i,j)$ is an edge feature based on the spectral difference of neighboring pixels, formally

$$\Gamma(i,j) = F(\|I_i - I_j\|) \quad \textbf{(19)}$$

where I_i and I_j are the spectral vectors of pixel i and j respectively.

Then, the segmentation problem is solved by finding the least energy configuration of the CRF defined above. Domke implemented and compared two methods for energy minimization: Tree-Reweighted Belief propagation (and its variants such as Loopy BP, TRW-S) and Mean-field. Further information about these methods can be found in Domke (2013), Kolmogorov(2006) and Wang et al. (2005). The implementation of this algorithm was taken from the author's website.