A comparative review of Chan-Vese, K-Means, and Otsu image segmentation algorithms

FEKI Hana
ENSTA Paris
hana.feki@ensta.fr

MANSOUR Rayen
ENSTA Paris
rayen.mansour@ensta.fr

ZARGUI Rayen
ENSTA Paris
rayen.zargui@ensta.fr

***Abstract*—Image segmentation is a key step in computer vision that allows the extraction of regions of interest from complex images. This project presents a comparative analysis of several segmentation methods, starting with the Chan-Vese algorithm, which relies on active contour models to segment homogeneous regions. However, after an initial implementation, we discovered that it didn't perform well in terms of contour precision and robustness to intensity variations. As a result, we explored alternative algorithms, such as the Otsu method and k-means clustering which provided better performance. This report outlines the various steps of our work, from the initial implementation to the evaluation of these more effective methods.**

I. Introduction

Computer vision, as an interdisciplinary field combining computer science, mathematics, and artificial intelligence, aims to enable machines to interpret and visually analyze the world, much like human perception [17]. Among the fundamental challenges of this discipline, **image segmentation holds a central position** as it involves partitioning an image into meaningful regions to facilitate its analysis, processing, or use in various applications such as medicine, robotics, and industrial image analysis.

It is important to distinguish between image segmentation, object detection, and image classification. While image classification focuses on labeling an entire image with a single category, and object detection identifies and locates objects within an image, image segmentation is more granular—it divides an image into distinct regions, each representing different objects or areas of interest [17]. Figure 1 illustrates these differences with a visual comparison.

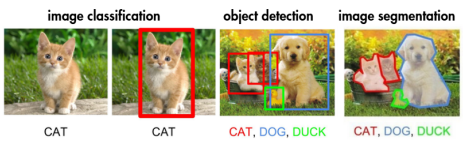

Fig. 1: Comparison of image classification, object detection, and image segmentation [17]

It is also crucial to distinguish between various types of image segmentation, each serving different purposes:

- **Semantic Segmentation**: Assigns a class label to each pixel in an image, grouping pixels with similar characteristics such as color, texture, or shape. This provides a pixel-wise map of an image, enabling detailed and accurate analysis [4].
- **Instance Segmentation**: Differentiates between distinct objects of the same class, providing a segmentation map for each object instance within an image [12].
- **Panoptic Segmentation**: Combines elements of both semantic and instance segmentation, identifying object categories and distinguishing between object instances, offering a comprehensive understanding of the image content [4].

This project focuses on the precision of segmentation algorithms, initially exploring the Chan-Vese method, before evaluating other approaches like the Otsu method and the k-means clustering to enhance results [8]. The report details the different stages of our work, emphasizing the design, implementation, and evaluation of the algorithms, while highlighting the challenges faced and opportunities for improving the robustness and accuracy of segmentation techniques.

II. Context

In this project, the Chan-Vese algorithm was initially chosen for its capability to segment images by considering regional characteristics and boundaries [3]. However, our experiments revealed several limitations:

- **Sensitivity to intensity variations**: The algorithm often struggled with images exhibiting intensity inhomogeneity, leading to imprecise segmentation results.
- **Difficulty in detecting sharp boundaries**: Particularly in noisy medical images, the Chan-Vese model had trouble delineating clear object boundaries.
- **High computational cost**: Processing large images resulted in significant computational demands, making the algorithm less efficient for high-resolution data.

To address these challenges, we explored complementary solutions, including Otsu's method, which utilizes histogram-based thresholding, and k-means clustering techniques tailored for instance segmentation [8]. Recognizing the limitations of the Chan-Vese algorithm, we sought to enhance its performance by integrating preprocessing techniques such as noise reduction filters. Applying filters like the Gaussian filter helped smooth

images and improve boundary detection, leading to more reliable segmentation outcomes in our specific application context [3].

III. Needs Analysis and Feasibility

A. Needs analysis

The primary objective is to efficiently and accurately detect object boundaries within images to facilitate tasks like recognition, analysis, and classification. Precise boundary segmentation is crucial for applications such as object recognition, shape detection, and enhancing the performance of computer vision models. Accurate segmentation delineates areas of interest more clearly, reducing errors in subsequent image processing stages [9].

B. Feasibility

The Chan-Vese algorithm minimizes an energy functional to segment homogeneous regions while preserving boundaries. However, factors like noise and poorly defined objects can affect its effectiveness [15]. To improve robustness and precision, we combined the Chan-Vese approach with preprocessing filters, such as Gaussian smoothing [16], to reduce noise and clarify boundaries. This combination enhanced segmentation performance while preserving essential image details.

IV. Specification

A. Inputs

The system accepts images in both color and grayscale formats. These images, sourced from various origins, often contain noise or low contrast. Therefore, preprocessing is necessary to reduce noise and enhance boundary clarity. This preprocessing includes smoothing techniques like Gaussian filtering and image normalization.

B. Output

The system's output is a binary image where the object boundaries are clearly detected. The contours are delineated with greater precision after applying filters and the refined Chan-Vese algorithm, enabling better analysis of the objects present in the image.

V. Design

A. Class diagram

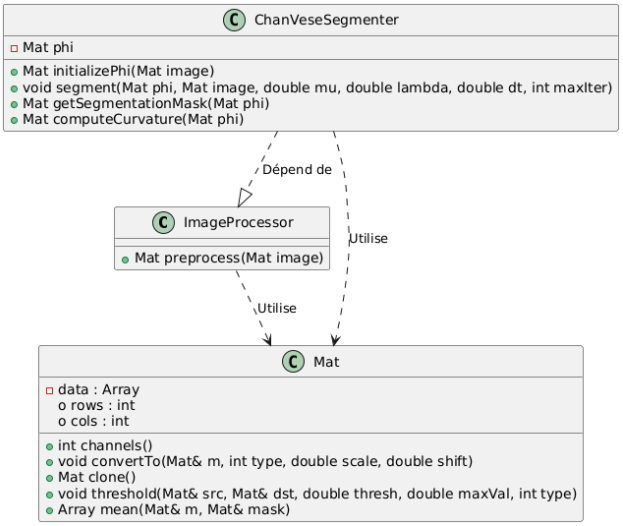

Fig. 2: Class UML Diagram

The class diagram illustrates the interaction among three main classes: `ImageProcessor`, `ChanVeseSegmenter`, and `Mat`.

- **ImageProcessor**: The `ImageProcessor` class is responsible for preprocessing the image. It performs operations such as conversion to grayscale, application of Gaussian blur, and histogram equalization. Once the image is preprocessed, it is passed to the `ChanVeseSegmenter` class for segmentation.

- **ChanVeseSegmenter**: This class applies the Chan-Vese segmentation algorithm to the preprocessed image to separate regions of interest. It also includes methods to initialize the ϕ function, evolve the contours, and generate a binary mask representing the segmented regions.

- **Mat**: Both the `ImageProcessor` and `ChanVeseSegmenter` classes utilize the `Mat` class from the OpenCV library to handle image data as matrices. This facilitates efficient execution of various image processing and segmentation tasks.

VI. Detailed Design

A. Libraries used

In this project, the OpenCV library is utilized for image processing tasks. OpenCV provides functions for operations such as converting images to grayscale, applying Gaussian blur, and performing histogram equalization. It also offers advanced features for image segmentation and a convenient interface for handling image data through the `Mat` class, which represents an n-dimensional dense numerical array suitable for storing images and other data types.

B. Class definitions

1) ImageProcessor Class:

- `preprocess(const Mat &image)`: This method accepts an image as input. If the image is in color, it converts it to grayscale. It then applies Gaussian blur to smooth the image, performs histogram equalization to enhance contrast, and converts the image to a `CV_64F` matrix type for normalization between 0 and 1. The method returns the preprocessed image. These steps improve the suitability of the image for segmentation by reducing noise and enhancing the contrast of regions of interest.

2) ChanVeseSegmenter Class:

- `initializePhi(const Mat &image)`: Initializes the phi function by creating a disk centered in the middle of the image. Pixels inside the disk are set to -1, and those outside are set to +1, establishing an initial boundary for segmentation.
- `segment(Mat &phi, const Mat &image, double mu, double lambda, double dt, int maxIter)`: Performs segmentation using the Chan-Vese algorithm, which iteratively adjusts the phi function based on data and curvature forces. The process continues for a specified number of iterations.
- `getSegmentationMask(const Mat &phi)`: Generates a binary mask from the phi function, where pixels inside the contour are set to 255 (white), and those outside are set to 0 (black).
- `computeCurvature(const Mat &phi)`: Detects and outlines the contours found in the original image. Only contours with an area exceeding a defined threshold are drawn to minimize noise. The resulting image with drawn contours is saved under the specified filename.

These classes collaborate to perform image segmentation based on the Chan-Vese method. The `ImageProcessor` class prepares the image, while the `ChanVeseSegmenter` class executes the segmentation. This approach is particularly effective for segmenting images without clearly defined boundaries.

C. Operating principles

The process begins with image preprocessing using the `ImageProcessor` class, which enhances the image's suitability for the segmentation algorithm. The preprocessed image is then passed to the `ChanVeseSegmenter` class, which initializes a function ϕ representing the contour of the region to be segmented. The Chan-Vese algorithm iteratively adjusts ϕ until the contour converges, effectively separating the region of interest from the background. Finally, a binary mask is generated and saved as a PNG image, representing the result of the segmentation.

VII. Implementation

A. Chan–Vese algorithm

The Chan–Vese algorithm is a variational image segmentation technique that models an image as comprising two relatively homogeneous regions—inside and outside a contour. Unlike gradient-based methods, the Chan–Vese model utilizes a level set function to implicitly represent the evolving contour, facilitating continuous boundary evolution.

1) Mathematical modeling: Given an image $I(x, y)$ over a domain $\Omega \subset \mathbb{R}^2$, the objective is to segment the image into two distinct regions: the interior and exterior of a contour C [3]. The Chan–Vese model defines the following energy functional:

$$E(c_1, c_2, C) = \mu \text{Length}(C) + \lambda_1 \int_{\text{inside}(C)} |I(x,y) - c_1|^2 \, dx\, dy + \lambda_2 \int_{\text{outside}(C)} |I(x,y) - c_2|^2 \, dx\, dy, \tag{1}$$

where c_1 and c_2 are the average intensities inside and outside the contour C, respectively, and μ, λ_1, and λ_2 are parameters weighting each term's significance.

To facilitate implementation, the contour C is represented by a level set function $\phi(x, y)$ such that:

$$C = \{(x, y) \in \Omega \mid \phi(x, y) = 0\}. \tag{2}$$

The regions are defined by:

$$\text{Interior} : \phi(x, y) < 0, \quad \text{Exterior} : \phi(x, y) \geq 0. \tag{3}$$

The energy functional is reformulated in terms of ϕ using the Heaviside function $H(\phi)$ and its derivative $\delta(\phi)$:

$$E(\phi, c_1, c_2) = \mu \int_\Omega \delta(\phi(x,y)) \, |\nabla \phi(x,y)| \, dx\, dy + \lambda_1 \int_\Omega H(\phi(x,y)) \, |I(x,y) - c_1|^2 \, dx\, dy + \lambda_2 \int_\Omega (1 - H(\phi(x,y))) \, |I(x,y) - c_2|^2 \, dx\, dy. \tag{4}$$

The goal is to evolve $\phi(x, y)$ to minimize $E(\phi, c_1, c_2)$. This minimization is typically performed via gradient descent, leading to the evolution equation:

$$\frac{\partial \phi}{\partial t} = \delta(\phi) \left[\mu \, \text{div}\left(\frac{\nabla \phi}{|\nabla \phi|} \right) - \lambda_1 (I - c_1)^2 + \lambda_2 (I - c_2)^2 \right]. \tag{5}$$

2) Chan-Vese algorithm pseudocode: The algorithm proceeds as follows [6]:

```
// Input: A grayscale image I
// Output: A binary image after
    segmentation

Begin
```

```
    // Step 1: Define the initial contour
    Define phi(x, y) for (x, y) in Omega

    // Step 2: Repeat until convergence
    Repeat
        // Step 2.1: Compute the average
            intensities inside and outside
            the contour
        c1 = integral(Omega, I(x, y) * H(
            phi(x, y))) / integral(Omega, H
            (phi(x, y)))
        c2 = integral(Omega, I(x, y) * (1 -
             H(phi(x, y)))) / integral(
            Omega, (1 - H(phi(x, y))))

        // Step 2.2: Evolve phi(x, y) using
             the evolution equation
        d(phi) / dt = delta(phi) * [mu *
            div(grad(phi) / |grad(phi)|) -
            lambda1 * (I - c1)^2 + lambda2
            * (I - c2)^2]

        // Step 2.3: Reinitialize phi(x, y)
             if necessary

    Until convergence

    Return the segmented image
End
```

The process continues until $\phi(x, y)$ converges, defining the contour $C = \{\phi(x, y) = 0\}$.

B. *Image preprocessing*

Before applying the **Chan-Vese** algorithm, it is crucial to prepare the image to reduce noise, enhance contrast, and facilitate contour identification. Effective preprocessing improves algorithm convergence and segmentation accuracy [5].

1) Grayscale conversion: The initial step is to convert the image to grayscale, as the Chan–Vese algorithm operates on intensity variations rather than color.

2) Gaussian blurring: Applying a Gaussian blur smooths the image, reducing high-frequency noise that may interfere with segmentation.

3) Histogram equalization: Histogram equalization enhances contrast by redistributing intensity values, making regions of interest more distinguishable.

4) Normalization: Converting the image to a `CV_64F` matrix type and normalizing pixel values between 0 and 1 standardizes data for numerical stability.

5) Level set initialization: An initial level set function $\phi(x, y)$ is defined, often as a signed distance function representing a simple shape (e.g., a circle). This serves as the starting contour.

6) Parameter selection: Choosing appropriate values for parameters μ, λ_1, λ_2, time step Δt, and iteration count is crucial for segmentation performance.

7) Grayscale conversion: If the original image has three color channels (RGB), it is converted using the following transformation:

$$I_{\text{gray}}(x, y) = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

This conversion simplifies the data and focuses only on variations in light intensity.

```
// Convert to grayscale
    if (image.channels() == 3)
        cvtColor(image, gray,
            COLOR_BGR2GRAY);
    else
        gray = image.clone();
}
```

8) Noise reduction using gaussian filtering: Real-world images often contain noise, which can greatly affect the segmentation process. To address this issue, a **Gaussian filter** is used. This filter smooths the image, reducing noise while preserving important structures [16].

The filter is defined by the convolution of the image with a Gaussian function:

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

In our case, we use a kernel of size 5×5 with a standard deviation (σ) of 1.0. This smoothing reduces abrupt intensity variations due to noise while preserving significant contours.

```
// Gaussian filtering to reduce noise
GaussianBlur(gray, blurred, Size(5, 5),
    1.0);
```

9) Contrast enhancement using histogram equalization: After smoothing, the next step is to enhance the contrast of the image using **histogram equalization**. This technique redistributes grayscale levels to use the full intensity range (0 to 255), thereby improving the distinction between different regions in the image.

The transformation equation for intensity is as follows:

$$I'(x, y) = \text{round}\left(\frac{255}{N} \sum_{k=0}^{I(x,y)} p_k\right)$$

where N is the total number of pixels and p_k is the probability of intensity level k occurring.

This transformation reveals details in dark or overly bright regions of the image.

```
// Contrast enhancement using histogram
    equalization
equalizeHist(blurred, equalized);
```

Finally, the image is normalized so that its values range between 0 and 1, facilitating further processing.

Conclusion: The set of preprocessing steps helps improve the quality of the image before applying the Chan-Vese algorithm. The conversion to grayscale, Gaussian filtering, and histogram equalization contribute to optimizing the segmentation accuracy by minimizing disturbances caused by noise and intensity variations.

VIII. Validation tests

Validation tests are essential to ensure the robustness, reliability, and accuracy of the Chan-Vese segmentation algorithm implementation. In this chapter, we detail the various tests conducted to verify the proper functioning of the core classes of the application, addressing both unit tests and end-to-end tests.

A. Unit tests of core classes

Unit tests aim to validate the behavior of the essential methods of each class: `ImageProcessor` and `ChanVeseSegmenter`.

1) ImageProcessor class: This class handles image preprocessing: conversion to grayscale, Gaussian blur, and histogram equalization.

Test objectives:

- Verify correct conversion to grayscale.
- Confirm the effect of Gaussian blur.
- Validate histogram equalization.

Fig. 3: Original image [7]

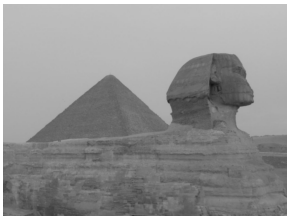

Fig. 4: Grayscale image

a) Grayscale conversion test: Visual inspection of grayscale uniformity and preservation of major contours.

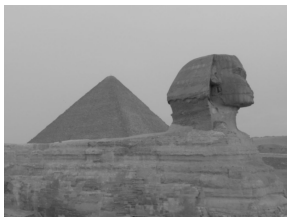

Fig. 5: Grayscale image

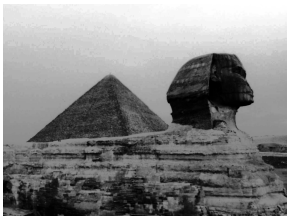

Fig. 6: Preprocessed image

b) Gaussian blur and histogram equalization test: Small details and noise should be smoothed out, while major contours remain visible.

2) ChanVeseSegmenter class: This class manages the initialization of the level set field ϕ and the Chan-Vese segmentation.

a) Test objectives:

- Verify correct initialization of ϕ.
- Observe the evolution of ϕ through iterations.
- Ensure the convergence of the algorithm.

Fig. 7: Blurred Image

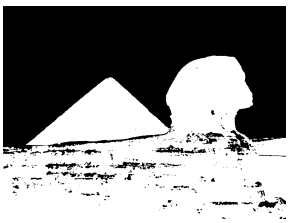

Fig. 8: Initialization of ϕ

b) Test of ϕ initialization: The level set should be negative inside the objects and positive outside. The initial contours should coincide with the transitions in brightness of the image.

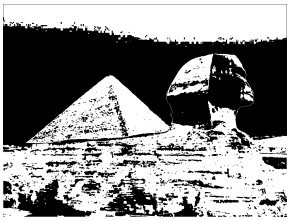

Fig. 9: ϕ after 50 iterations

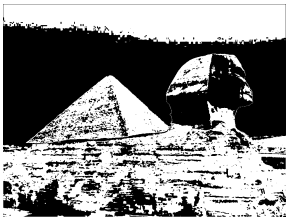

Fig. 10: ϕ after 100 iterations

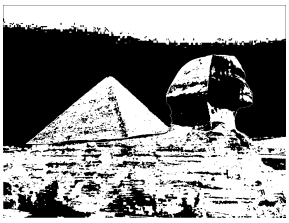

Fig. 11: ϕ after 150 iterations

c) Evolution test of ϕ: We observed the contour progression towards the object boundaries.

IX. End-to-end tests of fundamental classes

A. Test of preprocessing and initialization of ϕ

Fig. 12: Original image [7]

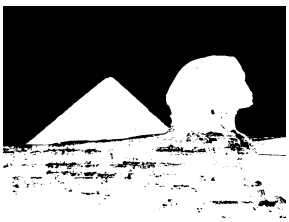

Fig. 13: Initialization of ϕ

Correct grayscale conversion and noise reduction. ϕ should reflect the interior and exterior regions of objects.

B. Test of segmentation chan-vese

Fig. 14: Original image

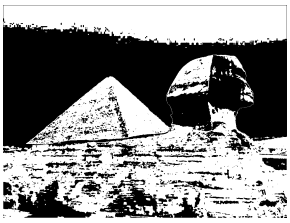

Fig. 15: Segmented image

Smooth contour convergence, although some artifacts remain visible.

X. Conclusion of validation tests

Unit and end-to-end tests confirm the correct functioning of the main steps of the algorithm:

- **Preprocessing** enhances image quality for segmentation.
- **Initialization of** ϕ provides reliable initial contours.
- The **Chan-Vese algorithm** effectively converges to object boundaries.

The Chan-Vese segmentation algorithm has proven effective in detecting object contours in an image, particularly when regions are well differentiated in terms of intensity. Validation tests have shown that the initialization and convergence of the algorithm enable a generally reliable segmentation.

However, some limitations have been observed. The high computational cost can slow down processing, especially for large images. Additionally, sensitivity to initialization can lead to errors if the initial contour is poorly positioned. The algorithm also struggles with complex textures or low-quality images, requiring specific adjustments to improve segmentation robustness.

To address these limitations, we have explored other segmentation methods to optimize performance and enhance the accuracy of results.

XI. Exploration of other segmentation solutions

A. Otsu's method

Otsu's method is an automatic thresholding technique used in image processing to segment an image into two classes (usually foreground and background) based on the grayscale histogram [11]. The main idea is to find an optimal threshold T that maximizes the *between-class variance*. This is based on the assumption that
textbfthe image histogram is bimodal.

1) Mathematical explanation: In this report, we will explain each step of this method, starting from an initial image and detailing the process until obtaining a binarized image. Let us take the example of the following image, represented by a grayscale level matrix:

$$\begin{pmatrix} 203 & 204 & 205 & 206 & 206 & 206 \\ 198 & 199 & 201 & 191 & 136 & 194 \\ 193 & 166 & 126 & 136 & 61 & 163 \\ 166 & 99 & 97 & 107 & 103 & 164 \\ 109 & 111 & 105 & 107 & 107 & 101 \\ 83 & 84 & 84 & 93 & 100 & 108 \end{pmatrix}$$

This 6-row by 6-column image contains 36 pixels, whose values, called intensities, range from 61 to 206. Each intensity represents a grayscale level, where 0 would correspond to black and 255 to white in a classical 0 to 255 scale. The objective is to find a threshold T that allows each pixel to be classified as either a "dark" class (value 0) or a "bright" class (value 1). Here is how Otsu's method proceeds step by step.

a) Step 1: Flatten the image into a vector: To analyze the intensities, we start by "flattening" the image into a single vector containing all pixel values read row by row. This gives us:

$$[203, 204, 205, 206, 206, 206, 198, 199, 201,$$

$$191, 136, 194, 193, 166, 126, 136, 61, 163, 166,$$

$$99, 97, 107, 103, 164, 109, 111, 105, 107, 107,$$

$$101, 83, 84, 84, 93, 100, 108]$$

This 36-element vector is easier to manipulate for the calculations that follow.

b) Step 2: Compute the histogram and probabilities: Next, we construct a **histogram**, which is a count of how often each intensity appears in the vector. For example, the value 206 appears 3 times, the value 107 also appears 3 times, while the value 61 appears only once. Here are some examples extracted from our vector:

- Intensity 61: frequency = 1
- Intensity 83: frequency = 1
- Intensity 84: frequency = 2
- Intensity 107: frequency = 3
- Intensity 206: frequency = 3

For intensities that do not appear (such as 0, 1, or 255), the frequency is 0. This histogram provides an overview of the distribution of intensities in the image.

Then, we calculate the **probability** of each intensity. Since there are 36 pixels in the image, the probability P_i of an intensity i is simply its frequency divided by the total number of pixels:

$$P_i = \frac{\text{Frequency of intensity } i}{36}$$

For example:

- $P_{61} = \frac{1}{36} \approx 0.0278$
- $P_{84} = \frac{2}{36} = \frac{1}{18} \approx 0.0556$
- $P_{107} = \frac{3}{36} = \frac{1}{12} \approx 0.0833$
- $P_{206} = \frac{3}{36} = \frac{1}{12} \approx 0.0833$

These averages indicate the "typical value" of each class for a given threshold.

c) Step 3: Compute between-class variance: To determine if a threshold t is optimal, Otsu uses the **between-class variance**, denoted $\sigma_B^2(t)$. This measure evaluates how well-separated the two classes are. The greater the means $\mu_1(t)$ and $\mu_2(t)$ differ, and the more balanced the class sizes are, the greater the variance [14]. It is defined as:

$$\sigma_B^2(t) = \omega_1(t) \cdot \omega_2(t) \cdot [\mu_1(t) - \mu_2(t)]^2$$

- $\omega_1(t) \cdot \omega_2(t)$: balances the sizes of both classes.
- $[\mu_1(t) - \mu_2(t)]^2$: this squared difference measures the difference between class means.

A threshold that maximizes this variance provides an optimal separation, as the classes are highly distinct.

d) Step 4: Find the optimal threshold: We compute $\sigma_B^2(t)$ for each value of t between 0 and 255, and choose the threshold T that gives the highest variance [14]:

$$T = \arg \max_{0 \le t \le 255} \sigma_B^2(t)$$

In our example, these calculations should be done numerically with the vector values. In practice, T will be an intensity that effectively separates dark and bright pixels, typically around 150 or 160, but this depends on the exact results.

e) Step 5: Binarize the image: Once T is determined, we apply this threshold to the initial image to obtain a binary image. For each pixel $I(x, y)$ in the image:

- If $I(x, y) \le T$, replace it with 0 (dark class).
- Otherwise, replace it with 1 (bright class).

The rule is:

$$\text{Binary image}(x, y) = \begin{cases} 0 & \text{if } I(x, y) \le T \\ 1 & \text{otherwise} \end{cases}$$

For example, if $T = 150$, then 61 becomes 0, 83 becomes 0, but 203 becomes 1, 206 becomes 1, etc. The final image contains only 0s and 1s, representing a black and white segmentation.

B. Pseudo-algorithm [13]

```
// Input: A grayscale image I
// Output: A binary image I1

Begin
    // Step 1: Calculate the histogram and
        probabilities
    Compute the histogram H of the image I
    Compute the probability of each
        grayscale level P from H

    // Step 2: Initialize variables
    maxVariance <- 0
    optimalThreshold <- 0

    // Step 3: Loop through all possible
        thresholds
    For T from 0 to 255 Do
        // Compute class probabilities
        P1 <- Sum of probabilities from 0
            to T
        P2 <- 1 - P1

        // Compute class means
         1  <- Mean of grayscale levels
            from 0 to T
         2  <- Mean of grayscale levels
            from T+1 to 255

        // Compute inter-class variance
        variance <- P1 * P2 * ( 1  -  2 )^2

        // Update the optimal threshold if
            the variance is maximum
        If variance > maxVariance Then
            maxVariance <- variance
            optimalThreshold <- T
        EndIf
    EndFor

    // Step 4: Apply the optimal
        thresholding
    For each pixel (x, y) in the image I Do
        If I(x, y) >= optimalThreshold Then
            I1(x, y) <- 255
        Else
            I1(x, y) <- 0
        EndIf
    EndFor

    Return the segmented image I1
End
```

1) Obtained results: To test the Otsu method, we chose a color image where both the object to be segmented and the background have **distinct hues**. After applying the procedure described above, we obtained the following results:

Fig. 16: Original image [7]

Fig. 17: Segmented image by Otsu

In this segmentation, the Otsu method appears to be **relatively effective** at separating the sky from the main scene (the Sphinx and the pyramid).

Now, for a more complex image containing **multiple objects** and a background with distinct hues, we obtained the following results:

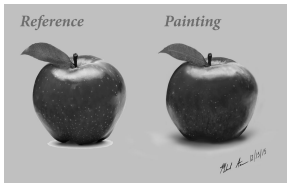

Fig. 18: Original image [2]

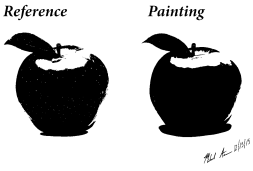

Fig. 19: Segmented image by Otsu

It can be observed here that the segmentation fails to properly capture **the contours and fine details** of the apples.

We also tested the method on a more complicated image with intensity variations and a complex background, and here is the result we obtained:

Fig. 20: Original image [1]

Fig. 21: Segmented image by Otsu

The main drawback of the Otsu method visible here is the poor segmentation of the background. Although the cat is correctly extracted, parts of the background with intensities similar to those of the cat have also been retained. This problem arises because Otsu uses a global threshold, which works poorly when the image has intensity variations or complex backgrounds. An adaptive or multi-threshold approach could improve the results.

XII. Segmentation using the K-means Method

The **K-means** method is an unsupervised classification technique that groups the pixels of an image into K distinct classes based on their intensities [19]. Unlike the Otsu method, which relies on optimizing the inter-class variance to determine a single binary threshold, K-means allows for multi-class segmentation, thus offering better adaptability to complex images.

A. Mathematical Explanation

The K-means algorithm aims to partition a dataset into K clusters by minimizing the intra-class variance defined as [18]:

$$J = \sum_{i=1}^{K} \sum_{x_j \in C_i} \|x_j - \mu_i\|^2 \tag{6}$$

where:

- K is the number of clusters,
- C_i is the i-th cluster,
- x_j is a pixel belonging to cluster C_i,
- μ_i is the centroid of cluster C_i.

The algorithm follows these steps:

1) Initialize K centroids (randomly or with K-means++),
2) Assign each pixel to the nearest centroid,
3) Recalculate the centroids by taking the average of the pixels in each cluster,
4) Repeat steps 2 and 3 until convergence (i.e., when the centroids no longer change significantly).

B. Pseudo-algorithm

```
// Input: A grayscale image I
// Output: A segmented image I1

Begin
    // Step 1: Load and preprocess the
        image
    Load the grayscale image from the input
        file
    Apply Gaussian blur to reduce noise and
        smooth the image variations

    // Step 2: Convert the image for K-
        means processing
    Convert the image into a one-
        dimensional matrix of points (N, 1)

    // Step 3: Initialize the centroids
    Initialize K centroids (randomly or
        using K-means++)

    // Step 4: Run the K-means algorithm
    Repeat until convergence
        // Assigning pixels to centroids
        Assign each pixel to the nearest
            centroid based on Euclidean
            distance

        // Update the centroids
        Recalculate the centroids as the
            average of the pixels in each
            cluster

        // Convergence condition
        Check if the centroids have changed
             significantly; if not, stop
    EndRepeat

    // Step 5: Reconstruct the segmented
        image
        For each pixel (x, y) in the image
            I Do
            Replace the pixel s value with
                the value of its
            centroid
        EndFor

    // Step 6: Display and save the result
    Display the segmented image I1 on
        screen
    Save the result in an output file

    Return the segmented image I1
End
```

C. Results obtained

The K-means algorithm groups the pixels into multiple grayscale intensity classes. Below are the results obtained:

Fig. 22: Original image [7]

Fig. 23: Segmented image by K-means

Now, for a more complex image containing **multiple objects** and a background with distinct hues, we obtained the following results:

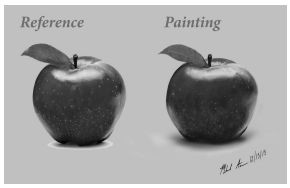

Fig. 24: Original image [2]

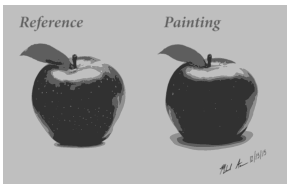

Fig. 25: Segmented image by K-means

We also tested the method on a more complicated image with intensity variations and a complex background, and here are the results:

Fig. 26: Original image [1]

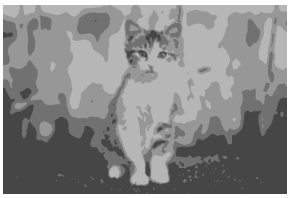

Fig. 27: Segmented image by K-means

The segmentation by K-means applied to the grayscale image enables classification of pixels into several intensity groups, offering a more detailed separation of different areas in the image. However, it can generate fuzzy borders and is sensitive to the choice of the number of clusters. In comparison, the Otsu method performs a binary thresholding that segments the image into two distinct classes (object and background), ensuring a sharp separation but potentially leading to information loss if the intensity histogram is not clearly bimodal [10]. Thus, K-means offers a more progressive segmentation, while Otsu is more direct but sometimes too simplistic for complex images.

D. Conclusion

The K-means algorithm is a powerful alternative for image segmentation. However, it is more complex and requires proper initialization of centroids to achieve optimal results. A possible improvement would be to adapt K-means with advanced preprocessing techniques or integrate machine learning methods to dynamically adjust the number of required classes.

XIII. Work organization

A. Task distribution

The project was structured and divided into several distinct phases to ensure a methodical and efficient progression:

1) **Literature review on segmentation algorithms**: This phase involved exploring various image segmentation techniques, such as thresholding (Otsu), contour-based approaches (like Chan-Vese), and region-based methods, relying on scientific articles and technical resources.
2) **Design and selection of tools**: We evaluated and selected the most suitable tools, including libraries like OpenCV, considering their features, compatibility, and performance for our implementation.
3) **Implementation of algorithms**: This stage involved coding the selected algorithms, ensuring their smooth integration into our development environment, with preliminary tests to verify their functionality.
4) **Validation and performance optimization**: We evaluated the results of the algorithms on different datasets, measured their performance (accuracy, execution time), and optimized the parameters to improve the quality of segmentations.
5) **Writing the report**: Finally, we summarized our work, analyzed the results obtained, and documented the technical choices, challenges encountered, and future improvement opportunities in a final report.

B. Difficulties encountered

The main difficulty encountered during this project was **the installation and configuration of OpenCV in C++**, which required careful management of dependencies, compilers, and specific libraries, as well as adaptation to the particularities of different development environments (operating systems, software versions). This task was time-consuming due to frequent compilation errors and potential incompatibilities. Additionally, at the beginning of our work, we experimented with the Chan-Vese algorithm for segmentation, but the results were unsatisfactory, particularly in terms of accuracy and stability on our input images, due to its sensitivity to initial parameters and lighting variations. This led us to explore other algorithms, such as the Otsu method, to obtain more robust results suited to our specific problem.

XIV. Conclusion

This project allowed for the development and evaluation of image segmentation algorithms, initially focusing on the Chan-Vese method, before exploring other approaches

such as the Otsu method to **overcome the limitations encountered, particularly in terms of accuracy and robustness** in the presence of lighting variations or noise.

The design, implementation, and validation steps highlighted the importance of rigorous image preprocessing (conversion to grayscale, noise reduction, contrast enhancement) to optimize the performance of the algorithms.

Although the Chan-Vese method presented challenges related to its sensitivity to initial parameters, the Otsu method provided more stable and faster results, though it has its own limitations, such as the loss of fine details. The tests conducted validated the reliability of the core classes and processes implemented, while emphasizing the need to adapt the algorithms to the specifics of the input images.

Looking ahead, improvements could include the integration of machine learning techniques, such as convolutional neural networks, for more precise and adaptable segmentation, as well as further optimization of performance for real-time applications. Additionally, our parallel work on **3D brain tumor detection from MRI, conducted in Python**, yielded promising results, demonstrating the potential of these approaches for advanced medical applications.

References

[1] *"Just watching my cats can make me happy."*. Pinterest. URL: https://id.pinterest.com/pin/just-watching-my-cats-can-make-me-happy--1129910993984916163/.

[2] *1 Hour Grey Scale Apple Study by ArtofAscension on DeviantArt*. Dec. 16, 2015. URL: https://www.deviantart.com/artofascension/art/1-Hour-Grey-Scale-Apple-Study-578276594.

[3] Rami Cohen. *The Chan-Vese Algorithm*. July 15, 2011. DOI: 10.48550/arXiv.1107.2782. arXiv: 1107.2782[cs]. URL: http://arxiv.org/abs/1107.2782.

[4] Gabriela Csurka, Riccardo Volpi, and Boris Chidlovskii. *Semantic Image Segmentation: Two Decades of Research*. DOI: 10.48550/arXiv.2302.06378. arXiv: 2302.06378[cs]. URL: http://arxiv.org/abs/2302.06378.

[5] Pascal Getreuer. "Chan–Vese Segmentation". In: *Image Processing On Line* 2 (2012), pp. 214–224. DOI: 10.5201/ipol.2012.g-cv. URL: https://doi.org/10.5201/ipol.2012.g-cv.

[6] Pascal Getreuer. "Chan-Vese Segmentation". In: *Image Processing On Line* 2 (Aug. 8, 2012), pp. 214–224. ISSN: 2105-1232. DOI: 10.5201/ipol.2012.g-cv. URL: https://www.ipol.im/pub/art/2012/g-cv/?utm_source=doi.

[7] *Great Pyramid and Sphinx in Cairo - Picture of JAZ Makadi Saraya Resort, Makadi Bay - Tripadvisor*. URL: https://www.tripadvisor.com/LocationPhotoDirectLink-g297550-d302131-i17679326-JAZ_Makadi_Saraya_Resort-Makadi_Bay_Hurghada_Red_Sea_and_Sinai.html.

[8] Jonas Högberg et al. "Comparison of Otsu and an adapted Chan–Vese method to determine thyroid active volume using Monte Carlo generated SPECT images". In: *EJNMMI Physics* 11.1 (Jan. 2024), p. 6. ISSN: 2197-7364. DOI: 10.1186/s40658-023-00609-9. URL: https://doi.org/10.1186/s40658-023-00609-9.

[9] Ghatifernado Inc. *Segmentation Showdown: Tackling Overlapping Objects and Complex Boundaries*. Medium. Mar. 7, 2024. URL: https://medium.com/@ghatifernado.inc/segmentation-showdown-tackling-overlapping-objects-and-complex-boundaries-ecb6c395ae00.

[10] Arpan Kumar and Anamika Tiwari. "A Comparative Study of Otsu Thresholding and K-means Algorithm of Image Segmentation". In: *International Journal of Engineering and Technical Research (IJETR)* 9 (May 2019). DOI: 10.31873/IJETR.9.5.2019.62.

[11] Nicolas Loménie. *Image Processing and Analysis - Introduction for Computational Biology*. URL: https://helios2.mi.parisdescartes.fr/~lomn/Cours/BC/Publis/CompBio4.pdf.

[12] Shervin Minaee et al. *Image Segmentation Using Deep Learning: A Survey*. Nov. 15, 2020. DOI: 10.48550/arXiv.2001.05566. arXiv: 2001.05566[cs]. URL: http://arxiv.org/abs/2001.05566.

[13] Muthukrishnan. *Otsu's method for image thresholding explained and implemented*. Mar. 13, 2020. URL: https://muthu.co/otsus-method-for-image-thresholding-explained-and-implemented/.

[14] Minghao Ning. *Otsu's method*. Medium. Mar. 29, 2019. URL: https://medium.com/@MinghaoNing/otsus-method-db49e2f85093.

[15] Michael Roberts and Jack Spencer. "Chan–Vese Reformulation for Selective Image Segmentation". In: *Journal of Mathematical Imaging and Vision* 61.8 (2019), pp. 1173–1196. DOI: 10.1007/s10851-019-00893-0. URL: https://doi.org/10.1007/s10851-019-00893-0.

[16] Aryaman Sharda. *Image Filters: Gaussian Blur*. Medium. Jan. 25, 2021. URL: https://aryamansharda.medium.com/image-filters-gaussian-blur-eb36db6781b1.

[17] Pulkit Sharma. *Image Classification vs Object Detection vs Image Segmentation*. 2019. URL: https://medium.com/analytics-vidhya/image-classification-vs-object-detection-vs-image-segmentation-f36db85fe81.

[18] Saket Thavanani. *K-Means Clustering: Optimizing Cost Function Mathematically*. Apr. 2020. URL:

https : / / medium . com / analytics - vidhya / k - means - clustering - optimizing - cost - function - mathematically-1ccae156299f.

[19] Xin Zheng et al. “Image segmentation based on adaptive K-means algorithm”. In: *EURASIP Journal on Image and Video Processing* 2018.1 (2018), p. 68. ISSN: 1687-5281. DOI: 10.1186/s13640-018-0309-3. URL: https://doi.org/10.1186/s13640-018-0309-3.