Documento de diseño Aplicaciones Industriales

Andrés Tena De Tena

Javier Albaráñez Martínez

Índice

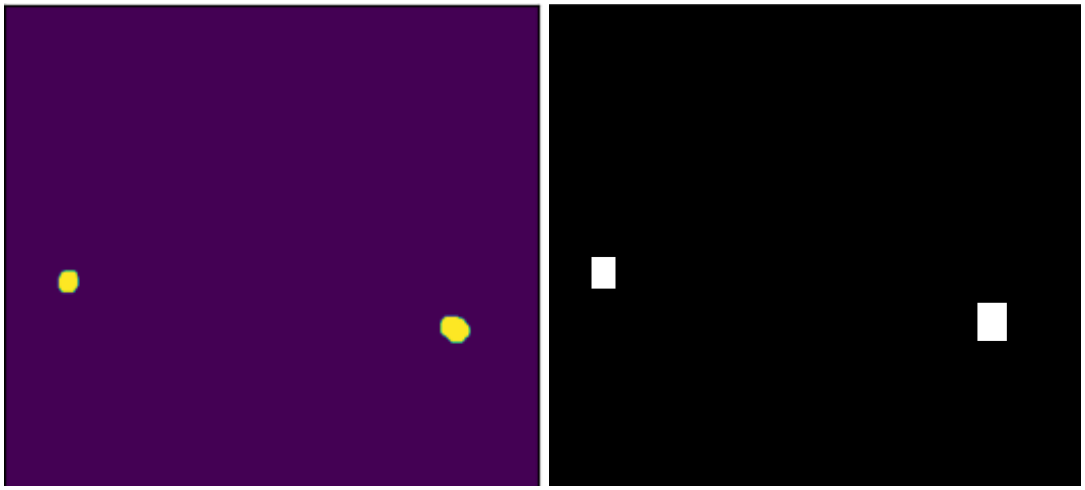
I. Introducción

Se ha desarrollado un prototipo de la aplicación final el cual, ya es funcional. Además, también se han diseñado tanto el esquema UML como el diagrama de secuencia de la aplicación, los cuales se mostrarán a continuación. Enlace a GitHub:

https://github.com/AlbaranezJavier/AIVA_2021_AJ

1. Test Prediction vs GT

Este test realiza una comparación entre el gropundtruth proporcionado por el cliente y la máscara de imperfecciones extraída por el algoritmo. Para pasar dicho test la diferencia entre ambas imágenes tendrá que estar por debajo de un umbral. A continuación, se muestran imágenes tanto de la máscara obtenida por el algoritmo como del groundtruth.



a) b)

c)

Figura 1: a) Imagen de la máscara extraída por el algoritmo, b) Groundtruth de la imagen y c) Imagen original.

La máscara extraída depende de un factor umbral, para determinar que umbral es el más adecuado se ha realizado una tabla con datos estadísticos para visualizar así que efectos causa la variación de dicho umbral. Por ejemplo, para la imagen anterior la tabla calculada fue la siguiente:

PASSED [100%]

Population N = TP+TN+FP+FN 215696	Predicted Positive PP = TP+FP 827	Bias pp = PP/N 0.38 [0. 0.]%	Predicted Negative PN = FN+TN 214869	Inverse Bias pn = PN/N 99.62 [100. 100.]%	
Real Positive RP = TP+FN 1583	TP 596	Recall tpr = TP/RP 37.65 [35. 41.]%	FN 987	FNR fnr = FN/RP 62.35 [59. 65.]%	LR+ LR+ = tpr/fpr 342.27
Prevalence rp = RP/N 0.73 [1. 1.]%	Precision tpa = TP/PP 72.07 [68. 76.]%	Performance tp = TP/N 0.28 [0. 0.]%	FN Accuracy fna = FN/PN 0.46 [0. 0.]%	Incorrect Rejection Rate fn = FN/N 0.46 [0. 0.]%	LR- LR- = fnr/tnr 0.62
Real Negative RN = FP+TN 214113	FP 231	FPR fpr = FP/RN 0.11 [0. 0.]%	TN 213882	Specifity tnr = TN/RN 99.89 [100. 100.]%	Odds ratio dor = LR+/LR- 552.05
Null Error Rate rn = RN/N 99.27 [99. 99.]%	False Discovery Rate fdr = FP/PP 27.93 [24. 32.]%	Delivered Error Rate fp = FP/N 0.11 [0. 0.]%	Inverse Precision tna = TN/PN 99.54 [100. 100.]%	Correct Rejection Rate tn = TN/N 99.16 [99. 99.]%	Informedness = tpr - fpr 37.54%
	Chi square -8342659.76 p=1.0	Correlation (TP*TN - FP*FN)/(PP*RP*RN*PN)**.5 3049.75 [nan nan]	Prob. Random Agreement pra = (PP*RP + PN*RN)/(N)**2 172.0%	Markedness = tpa - fna 71.61%	Accuracy acc = (TP + TN) / N 99.44 [99. 99.]%
	Matthews Corr. Coeff. mcc = (chi / N)**.5 nan	IoU = TP/(N-TN) 33.0 [30. 36.]%	Cohen Kappa = (acc-pra)/(1-pra) 1.01	Misclassification Rate err = (PF+FN)/N 1.0 [1. 1.]%	F1 score = 2*TP/(RP+PP) 0.49 [47. 52.]%

Figura 2: Tabla de datos estadísticos.

Con la experimentación con distintos ejemplos y distintos valores de umbral se ha llegado a la conclusión de que nuestra máscara poseía muchos falsos negativos debido a que el groundtruth es poco preciso. En cuanto a que umbral se debe escoger, se ha deducido que va a ser necesario automatizarlo debido a que no existe uno que obtenga buenos resultados para todas las imágenes.

2. Test Binary Mask

Asegura que la creación de la máscara de una imagen a partir de su etiqueta es la correcta. Para ello se crea la máscara de manera manual y se compara con la generada por el método.

3. Test Stadistics

Confirma que la extracción de las estadísticas de una imagen se hace de manera correcta. Para ello se calculan unas estadísticas de manera manual y se comparan con las obtenidas por el algoritmo.

II. Diseño de la solución

Diagrama UML que representa las relaciones que existen entre las clases que componen la aplicación.

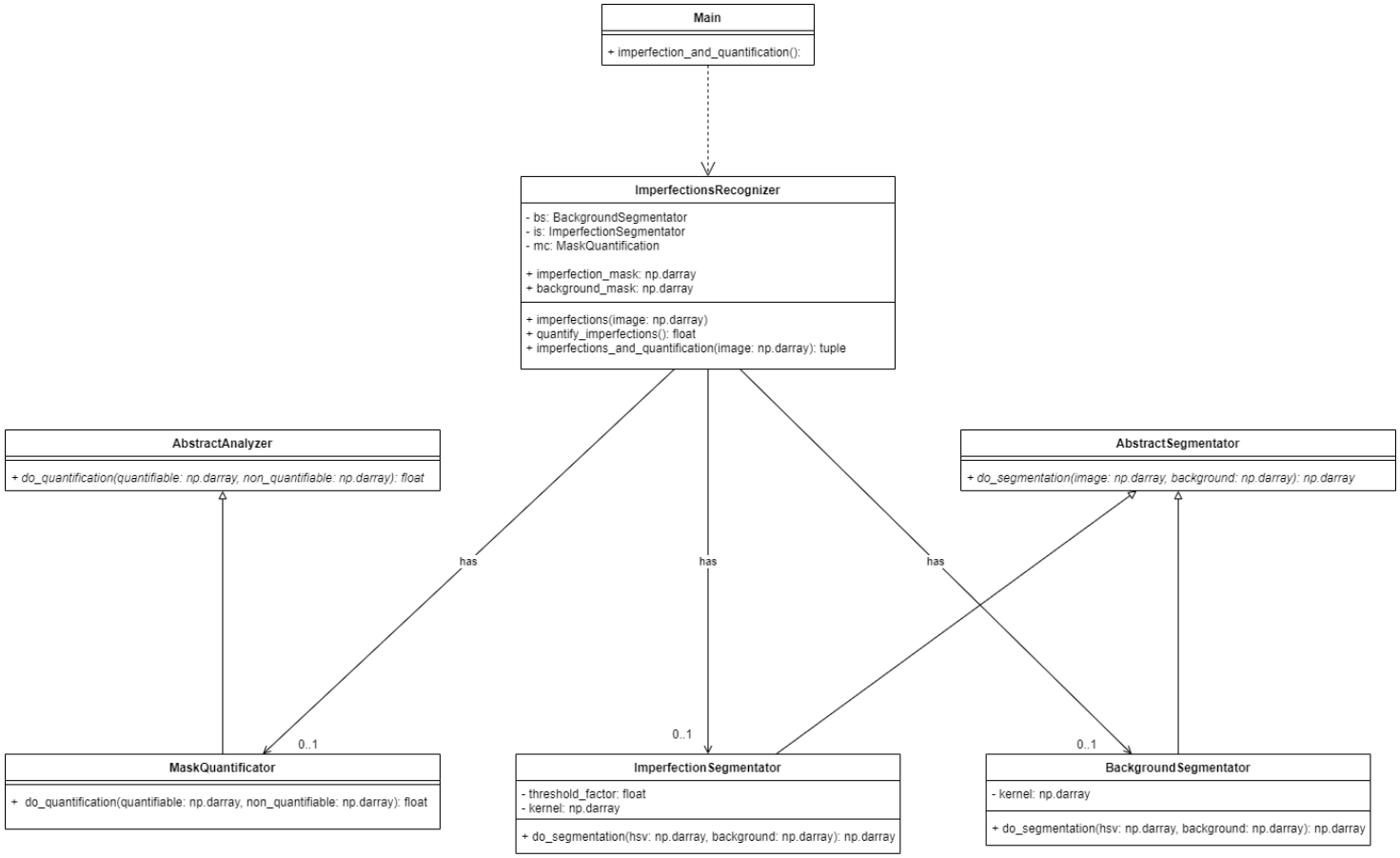


Figura 2: Diagrama UML.

1. ImperfectionRecognizer

Se trata de la clase que gestiona todo el funcionamiento de la aplicación, esta posee un objeto de cada una de las demás clases. Además de almacenar tanto la máscara que define el background como la que define las imperfecciones.

En esta clase están definidos los siguientes tres métodos:

- *imperfections(image)*: El cual se encargará de transformar la imagen en rgb a hsv y de llamar al método *do_segmentation(hsv)* del objeto BackgroundSegmentator al cual le proporcionará la imagen en hsv y devolverá una máscara. También activará el método *do_segmentation(hsv, background_mask)* pero del objeto ImperfectionSegmentator, que recibirá como parámetros la imagen en hsv y la máscara calculada en el paso anterior.
- *quantify_imperfections()*: Se encarga de llamar al método *do_quantification(imperfection_mask, background_mask)* del objeto MaskQuantificator.
- *imperfections_and_quantification(image)*: Llama al método *imperfections(image)* y al *quantify_imperfections()*.

2. MaskQuantificator

Clase que se va a encargar de realizar la cuantificación de los valores de una máscara, despreciando el área no cuantificable.

Posee el método *do_quantification(*quantifiable, non_quantifiable*)* que recibe dos parámetros como entrada:

- quantifiable: máscara con los valores a cuantificar.
- non quantifiable: máscara con los valores no cuantificables del total.

Devuelve un valor float con la cuantificación.

3. BackgroundSegmentator

Clase que va a realizar la extracción del fondo de la imagen.

Posee el método *do_segmentation(hsv, background)* que recibe dos parámetros como entrada:

- hsv: imagen en formato hsv.
- background: matriz de unos en la que se mostrará la máscara.

Devuelve una máscara del fondo.

4. ImperfectionSegmentator

Clase que va a generar una máscara de imperfecciones que tiene en cuenta el fondo.

Posee el método *do_segmentation(*hsv, *background)* que recibe dos parámetros como entrada:

- hsv: imagen en formato hsv.
- background: máscara que representa el fondo.

Devuelve una máscara de imperfecciones.

5. Diagrama de secuencia

La aplicación sigue el siguiente esquema, el cual en primer lugar leerá la imagen proporcionada utilizando la librería de opencv. A continuación, llamará al método *imperfection_and_segmentation(img)* de la clase ImperfectionRecognizer que devolverá una tupla con la máscara y el porcentaje de área con imperfecciones.

Para calcular dicha máscara ImperfectionRecognizer llama al método *do_segmentation(hsv, background)* de BackgrounSegmentator que devolverá una máscara que representa el fondo de la imagen. De tal manera que discriminaremos así todo lo que no sea la pieza de madera.

Dicha máscara del fondo será necesaria para calcular la máscara de imperfecciones. Para ello ImperfectionRecognizer activa el método *do_segmentation(hsv, background)* de la clase ImperfectionSegmentator.

Una vez hemos calculado la máscara de imperfecciones, es necesario cuantificar que área abarcan dichas imperfecciones. Para esto ImperfectionRecognizer utilizará el método *do_quantification(quantifiable, non_quantifiable)* de la clase MaskQuantificator.

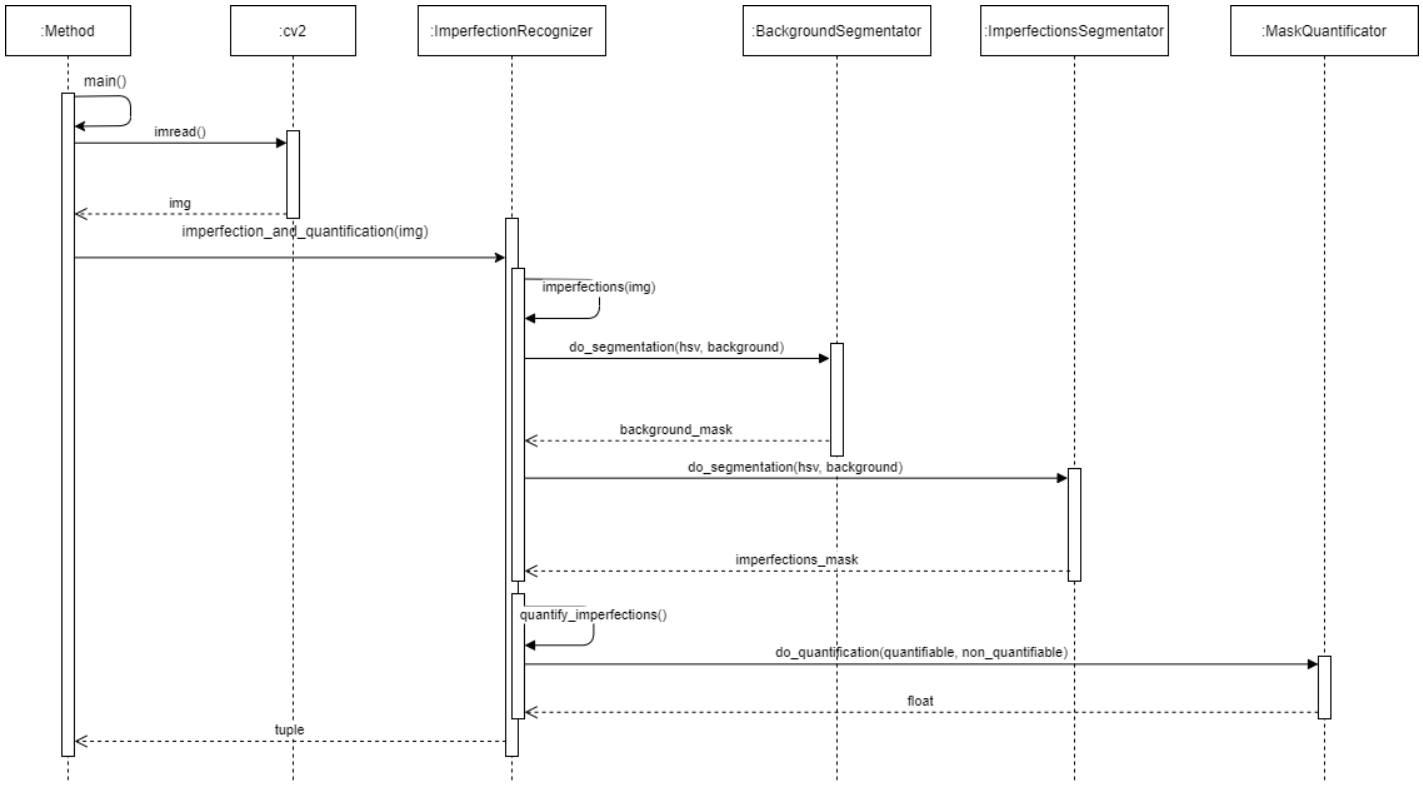


Figura 3: Diagrama de secuencia del método.

6. Diagrama de secuencia de Background Segmentator

En este apartado se describe el método principal *do_segmentation()* utilizado en la clase BackgroundSegmentator.

El método *do_segmentation()* (ver Figura 4) realiza una segmentación de una imagen en formato hsv, utilizando la información almacenada en el tercer canal de intensidad. Se busca el contorno de mayor área en la imagen, representando esta información en una máscara. Después, a este resultado se le aplican procesos de dilatación y erosión para eliminar el ruido de la imagen, obteniendo así el fondo de la imagen.

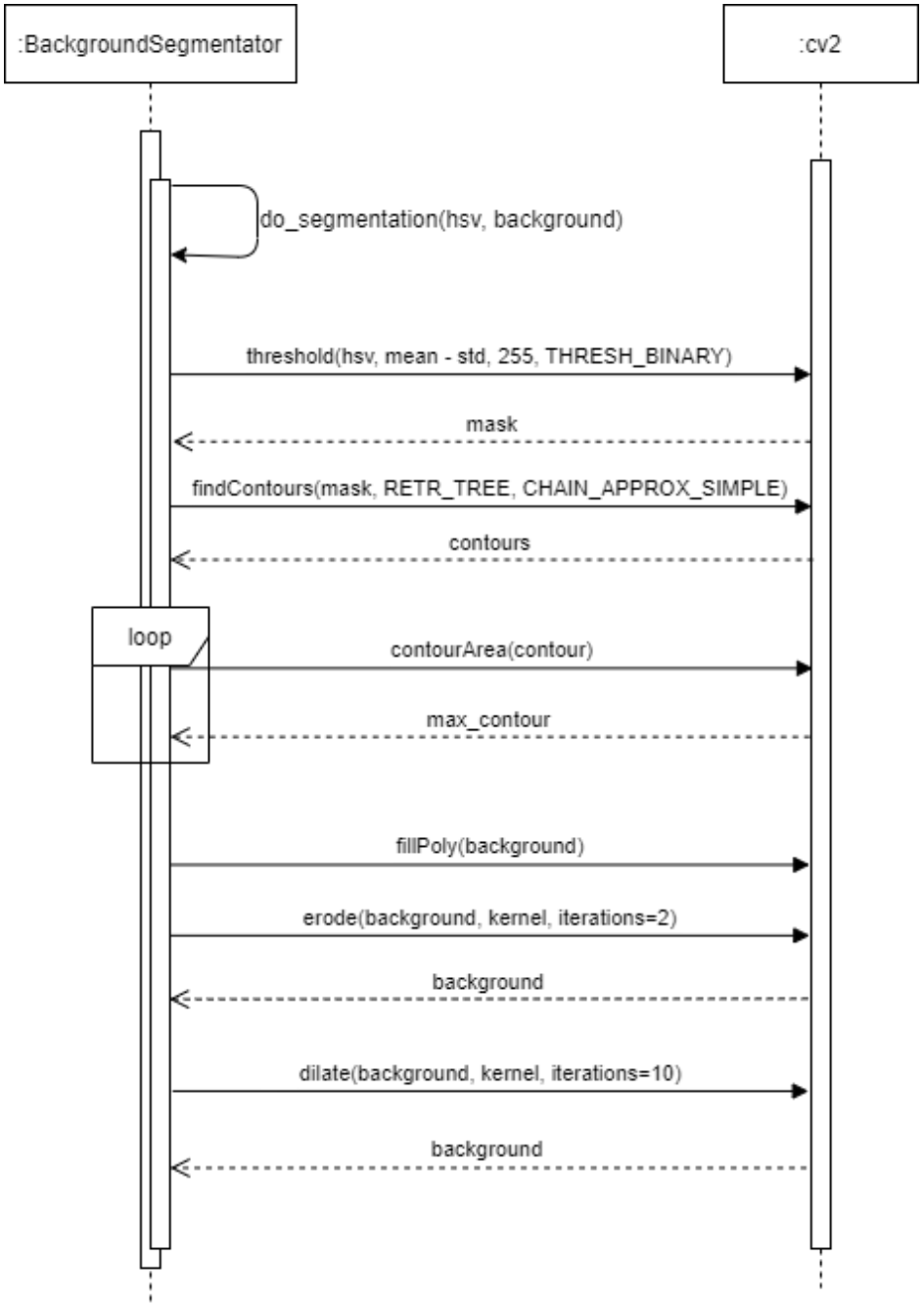


Figura 4: Diagrama de secuencia de BackgroundSegmentator.

7. Diagrama de secuencia de Imperfection Segmentator

En este apartado se describe el método principal *do_segmentation()* utilizado en la clase ImperfectionSegmentator.

El método *do_segmentation()* (ver Figura 5) varía en cuanto a la implementación realizada del método, con el mismo nombre, de BackgroundSegmentator en que en esta ocasión se tiene en cuenta la información de background para extraer una máscara con las imperfecciones presentes y se añade un factor a la binarización de la imagen. Además, se aplican operaciones morfológicas de apertura y cierre con el objetivo de mejorar los resultados.

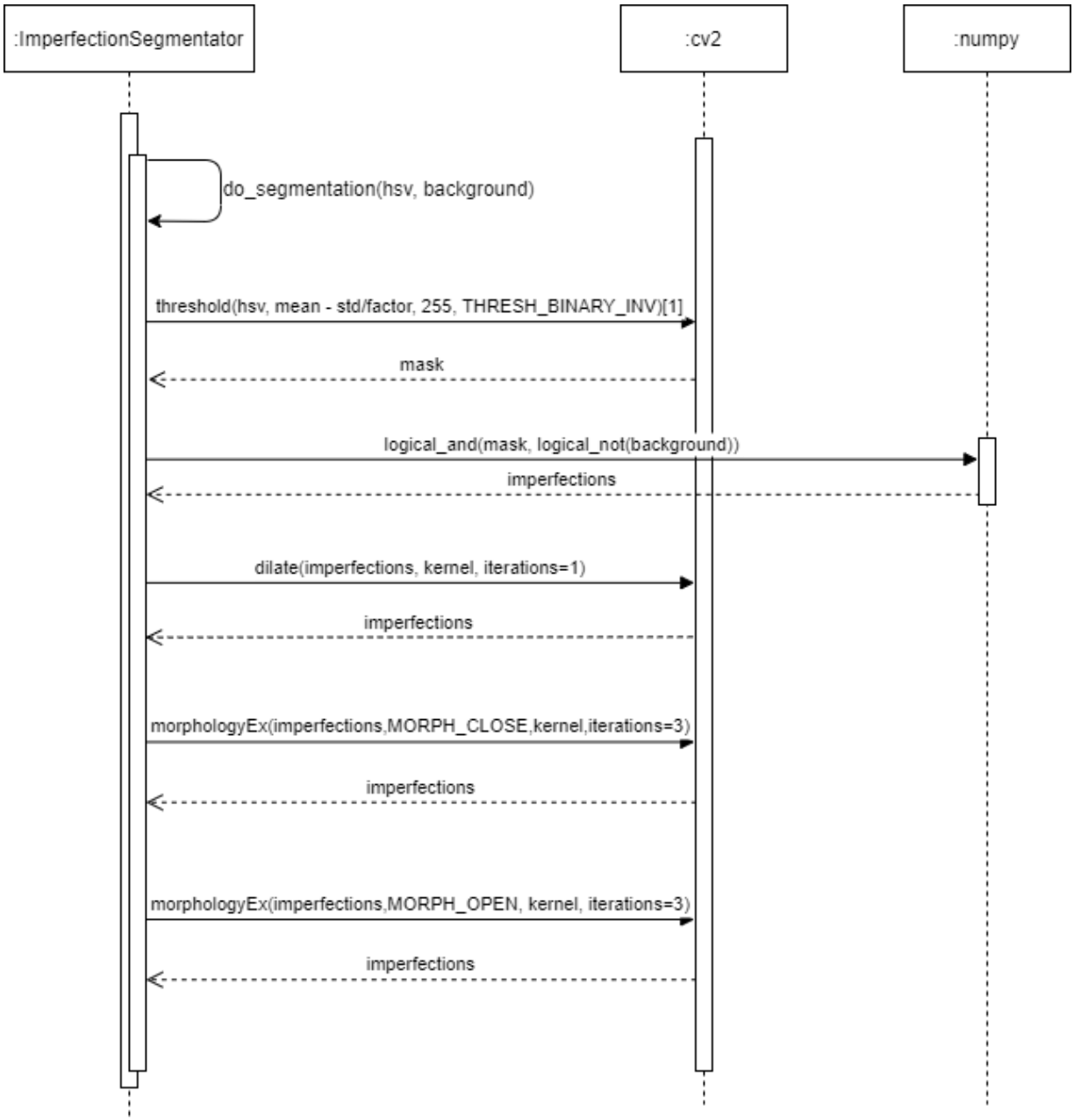


Figura 5: Diagrama de secuencia de ImperfectionSegmentator.