1. Introducción

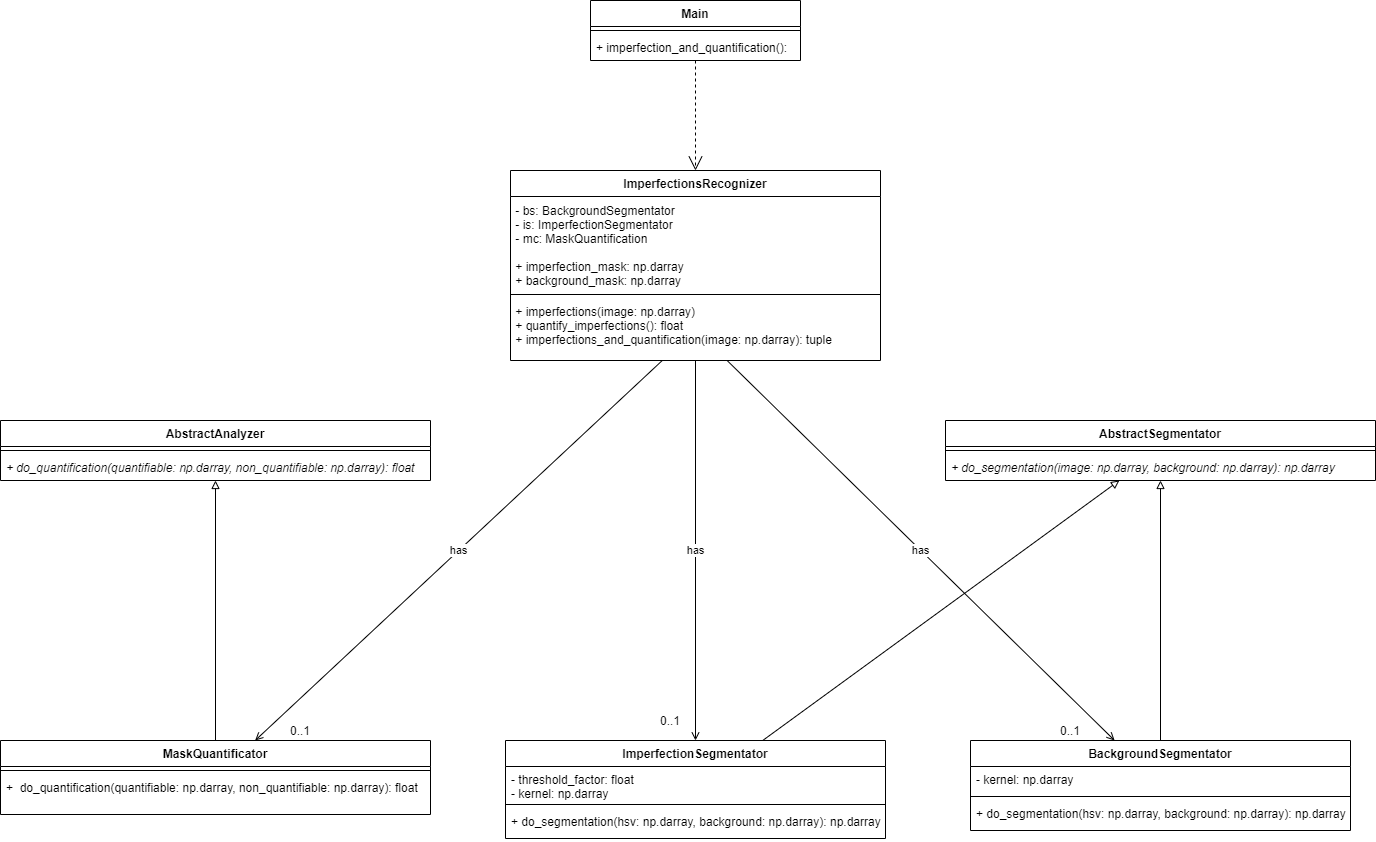
Se ha desarrollado un prototipo de la aplicación final el cual, ya es funcional. Además también se han diseñado tanto el esquema UML como el diagrama de secuencia de la aplicación, los cuales se mostrarán a continuación.

Una descripción del funcionamiento de los test sobre dos o tres ejemplos (incluyendo imágenes)

<https://github.com/AlbaranezJavier/AIVA_2021_AJ>

1. Diseño de la solución

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



*Figura 1: 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()*.

1. 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.

1. 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.

1. 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.

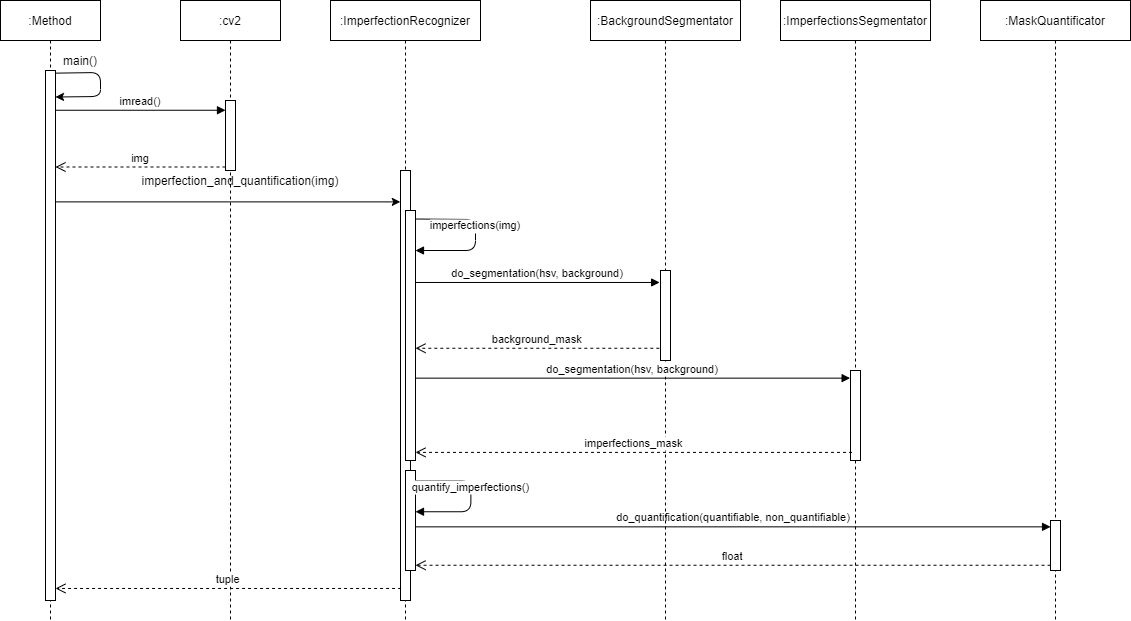
1. Diagrama de secuencia

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 2: Diagrama de secuencia.*