Proyecto:

# Clasificación y localización de defectos en superficies de metal

Documento de Diseño

Autores:
**Natalia Lourdes Pérez García de la Puente**
**Vicente Gilabert Maño**
**Luis Rosario Tremoulet**
Modelo: IEEE/ANSI 830-1998

# Contenidos

# 1. Introducción

Se va a realizar una descripción del funcionamiento principal del sistema de visión artificial sobre diversos ejemplos y con esquemas UML.

El repositorio donde se va a trabajar para el desarrollo del proyecto es el siguiente:

https://github.com/Luisrosario2604/AIVA_2022_METAL

## 1.1. Descripción del Funcionamiento

La aplicación *ImperDetect* recibe una imagen de 200x200 en escala de grises. Una vez comprobada que la imagen es válida según las especificaciones, se realiza una inferencia sobre un modelo como YOLOv5 para la clasificación y localización de los defectos. El sistema devolverá una clasificación de la imagen y guardará una imagen con los defectos detectados dibujados en su posición.

## 1.2. Ejemplos

A continuación, se muestran los resultados que se han obtenido sobre algunas de las imágenes de prueba.

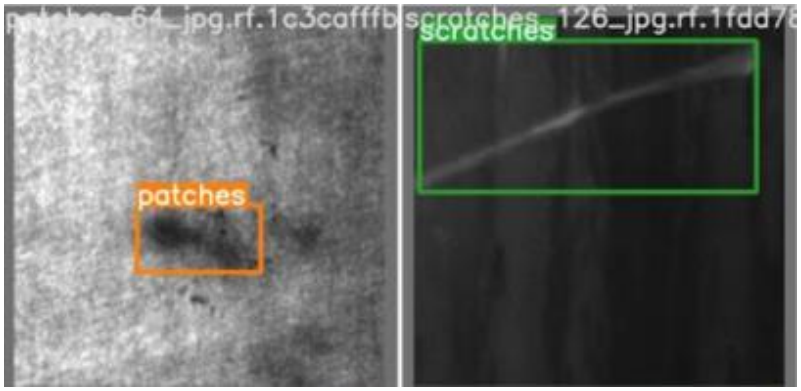


*Figura 1. Ejemplo de una detección simple*

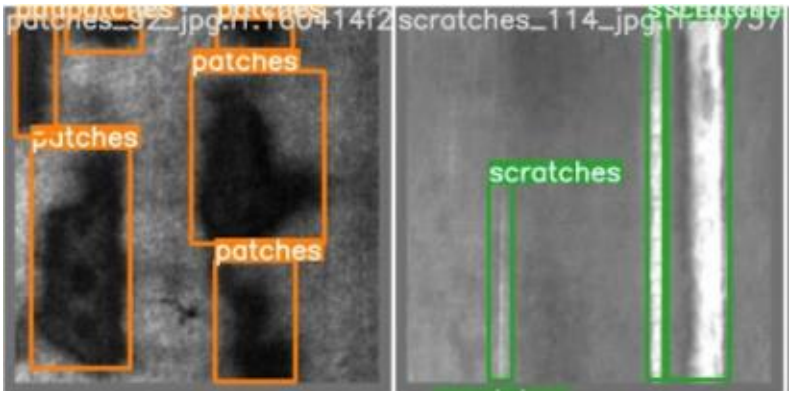


*Figura 2. Ejemplo de una detección múltiple*

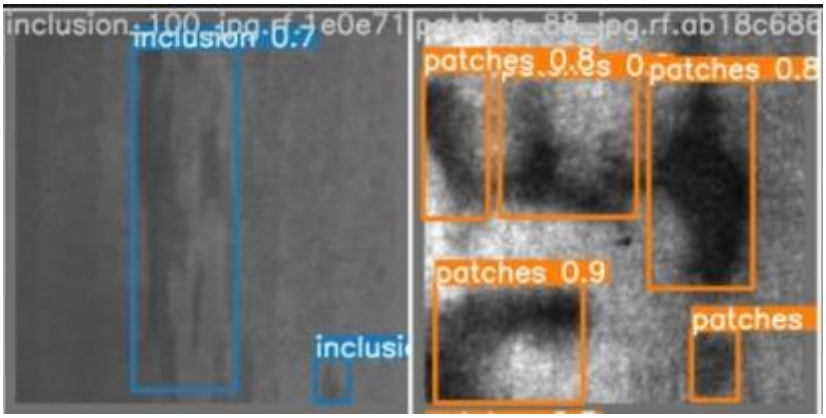


*Figura 3. Ejemplo de una detección múltiple*

## Observaciones:

De forma general, la red neuronal es capaz de detectar los tres tipos de imperfecciones en el conjunto de *test*. En ciertas ocasiones predice dos tipos de imperfecciones en la misma imagen, como se puede observar en las figuras 4 y 5.

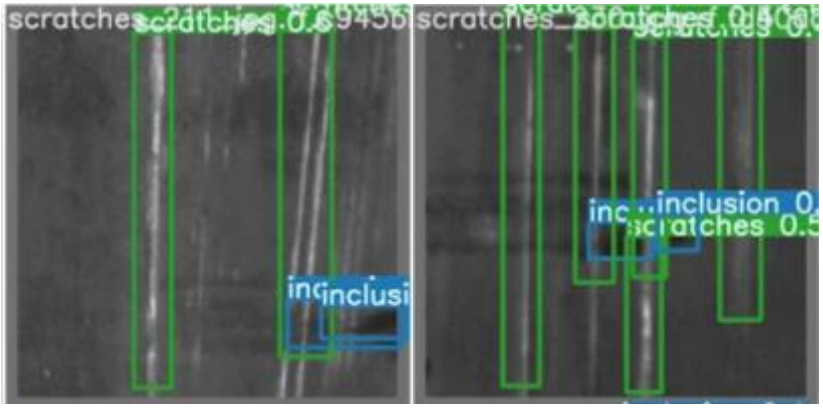

*Figura 4. Ejemplo de detecciones múltiples de diferentes clases*

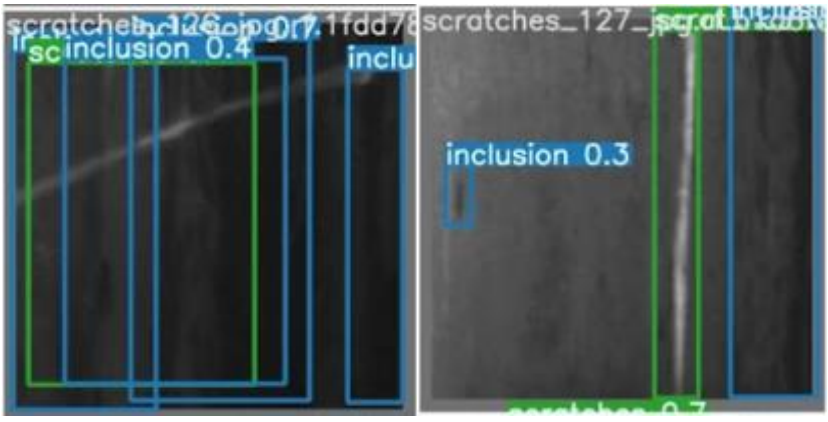


*Figura 5. Ejemplo de detecciones múltiples de diferentes clases*

# 2. Diseño de la Solución

A continuación, se va a realizar una explicación del diseño de la solución acompañada de distintos diagramas UML a modo de esquemáticos.

## 2.1. Diagrama Estático de Clases

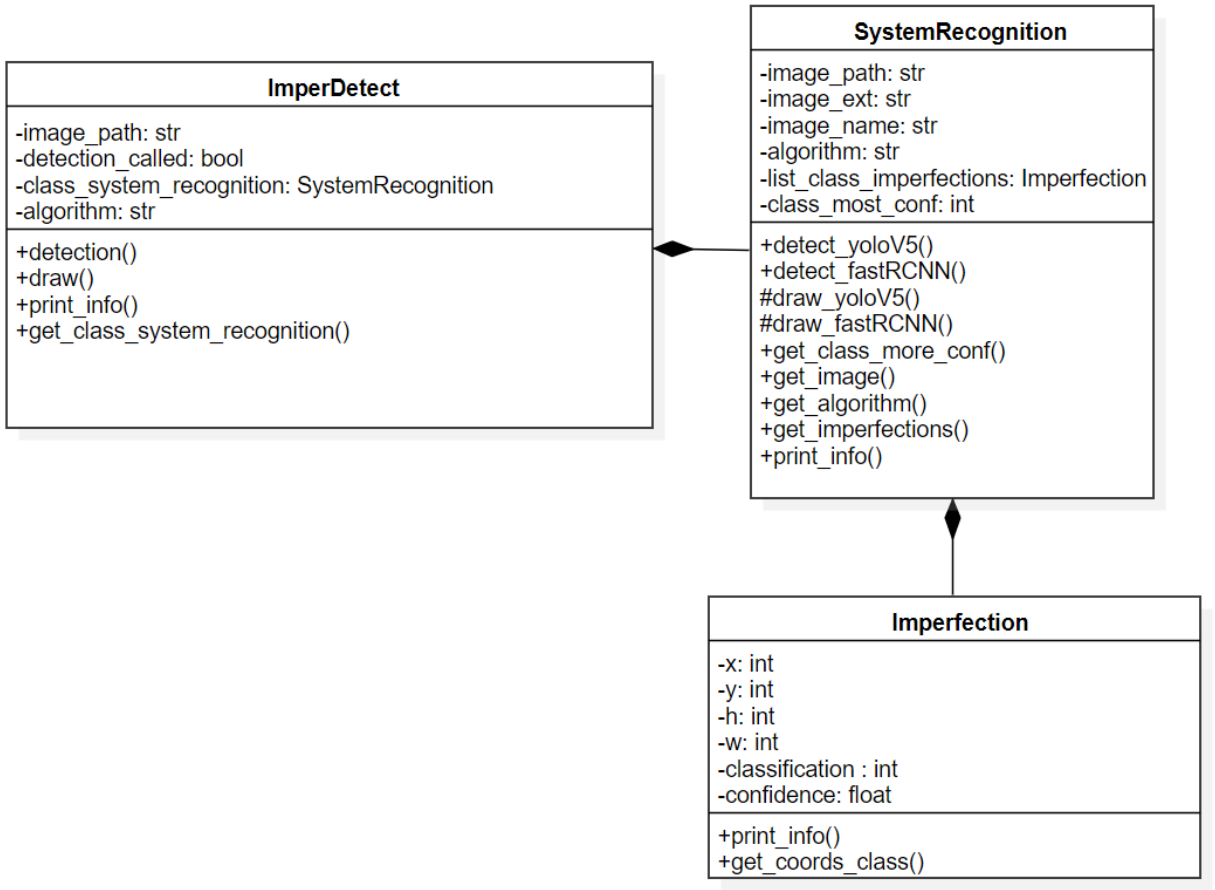


*Figura 6. Diagrama Estático de Clases*

## 2.2. Descripción de las Clases

### 2.2.1. ImperDetect

Clase que recibe la ruta de la imagen y el algoritmo a utilizar (“yoloV5”). Los métodos utilizados son:

- **detection(path_img):** método público que llama a *SystemRecognition* para realizar la detección. Como salida ofrece una lista de *Imperfection* con las coordenadas de detecciones en la imagen y una clasificación final.
- **draw(img, Imperfection):** método público que llama a *SystemRecognition* para realizar el dibujo de *Bounding Boxes* sobre la imagen.
- **print_info():** método para poder visualizar en consola la información de la imagen de entrada.
- **get_class_system_recognition():** método para realizar la llamada a la clase *SystemRecognition*.

### 2.2.2. Imperfection

Esta clase se crea para definir el formato de cada imperfección detectada y contiene las coordenadas y la *classification* a la que pertenece la muestra. Los parámetros de entrada son:

- **x:** posición de la *bounding box en la coordenada x.*
- **y:** posición de la *bounding box en la coordenada y.*
- **h:** altura de la *bounding box.*
- **w:** anchura de la *bounding box.*
- **classification:** clasificación de la *bounding box.*
- **confidence:** porcentaje de precisión de la *bounding box*.

En esta clase disponemos de un método:

- **print_info():** método para poder visualizar en consola la información de la imagen de entrada.
- **get_coords_class():** método para recibir la información de la clase.

### 2.2.3. SystemRecognition

Esta clase es el core de nuestro sistema. Es la encargada de realizar la detección, el postprocesado y el dibujado sobre las imágenes. Los parámetros de entrada son:

- **image_path:** la ruta de la imagen sobre la que se realiza la detección.
- **algorithm:** el modelo empleado para la detección (YOLOv5).

Los métodos más importantes de esta clase son:

- **detec_yoloV5():** método que realiza la inferencia sobre el modelo YOLOv5.
- **draw_yoloV5():** método que realiza los *Bounding Boxes* sobre la imagen basándose en la predicción de YOLOv5.
- **get_class_more_conf():** método que devuelve la *classification* con mayor precisión.

## 2.3. Diagrama de Secuencia

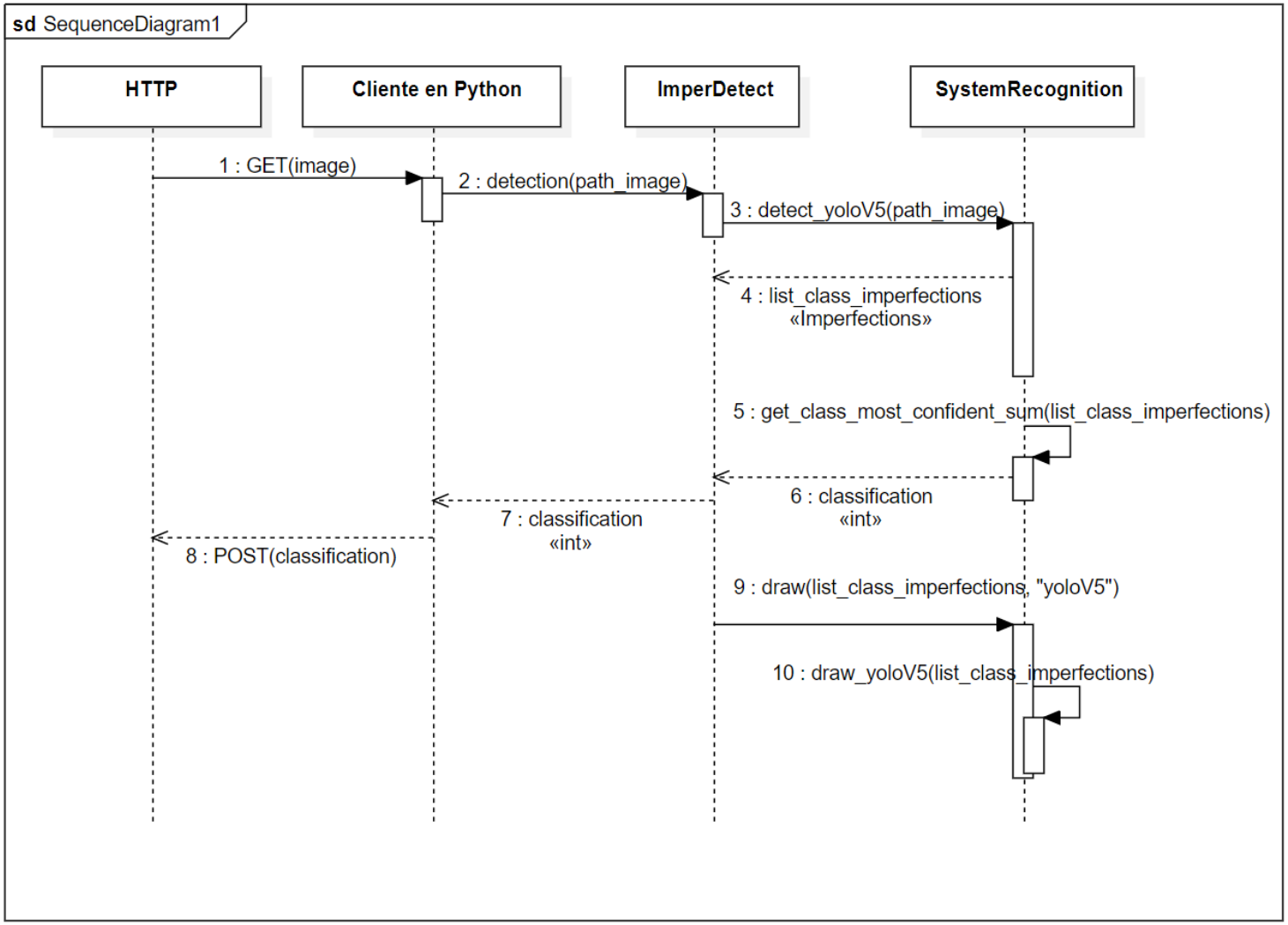


*Figura 7. Diagrama de secuencias*

### 2.3.1. Descripción del Diagrama de Secuencia

En primer lugar, el método *main_algorithm.py* es la llamada a la clase principal ImperDetect.

Esta llamada incluye el *path* de la imagen a la que se pretende realizar la detección y el *algorithm* o modelo (Yolov5) con el que se desea realizar la detección.

*ImperDetect* realiza la llamada a la clase *SystemRecognition* con la instrucción SystemRecognition.detection() para lanzar la inferencia sobre la imagen con el modelo elegido.

Tras la obtención de la detección, se crea un objeto de la clase *Imperfection* como lista de coordenadas de localizaciones.

Se devuelve una clasificación final con el método *get_class_most_confident_sum()* que proporciona una etiqueta para la imagen. Esta etiqueta se devuelve mediante un POST al servidor en C.

Otra funcionalidad, que se debe activar si se desea, es el guardado de las imágenes con las detecciones. Se realiza en base al *path* de la imagen y el algoritmo escogido.