Computação Visual
Relatório de Projeto OpenCV - Rust Detector

Rui Miguel da Silva Oliveira - 89216 – P3
Rui Pedro Pereira Santos - 89293 – P1

Resumo - O presente artigo apresenta o desenvolvimento de uma solução para segmentação e classificação de zonas com corrosão em objetos metálicos.

O artigo começa por apresentar o objetivo e o enquadramento da aplicação com a disciplina de Computação Visual e mais tarde serão apresentadas as decisões tomadas e consequentemente as conclusões/resultados finais.

I. INTRODUÇÃO

ENQUADRAMENTO COM A DISCIPLINA

A disciplina de Computação Visual visa focar conhecimentos associados à Síntese e Análise de Imagens através de áreas como Processamento de Imagem.

O **rust_detector** é exemplo disso mesmo.

OBJETIVO

O principal objetivo do **rust_detector** é desenvolver uma aplicação que a partir de uma ou mais imagens deteta zonas com corrosão em objetos metálicos.

As zonas identificadas deverão ser classificadas visualmente conforme a gravidade da corrosão.

II. REQUISITOS

1. Elaboração de um aplicação em Python que utilize a biblioteca OpenCV
2. Manipulação de imagens
3. Destacamento do local onde poderá existir ferrugem/corrosão
4. Menu com as várias funcionalidades da aplicação
5. Escala com a quantidade/gravidade de ferrugem na imagem
6. Obtenção de resultados de várias de uma forma simples e fácil
7. Permitir que o utilizador use a aplicação em qualquer imagem que deseje
8. Apresentar uma escala com a intensidade de corrosão

III. MODULARIZAÇÃO

Tendo em vista conseguir ter um projeto bem estruturado e, dentro dos possíveis, modular, foram utilizados os elementos presentes na tabela 1.

Para executar o jogo, é necessário executar o ficheiro **rust_detector.py.**

rust_detector.py	Ficheiro *python* que contém o código da aplicação desenvolvida.
Corrosion	Pasta fornecida pelos professores para efeitos de teste que contém fotografias de metais com ferrugem.
No_Corrosion	Pasta fornecida pelos professores para efeitos de teste que contém fotografias de metais sem ferrugem.
processImg(img)	Função que aceita uma imagem e retorna a manipulação da mesma onde os pixels que correspondem a um local onde existe ferrugem se devem manter e os restantes devem assumir a cor negra.
getIntensityImg(img)	Função que aceita uma imagem já processada e retorna a percentagem de pixels não negros, a imagem a preto e branco e a imagem com intensidades de corrosão definidas.
getPercent(img)	Função que aceita uma imagem e retorna a percentagem de pixels que não apresentam uma cor negra.

Tabela 1: Módulos utilizados no rust_detector

IV. IMPLEMENTAÇÃO

DETEÇÃO DA CORROSÃO

Ao observar várias imagens com metais que apresentavam corrosão na sua superfície, rapidamente se percebeu que esta seguia um padrão de cores. Assim, decidiu-se detetar a corrosão através das cores dos pixels e marca-los como uma superfície enferrujada quando as cores destes faziam parte do intervalo estabelecido.

Começou-se por definir boundaries no espaço de cores RGB (ou BGR, visto que o OpenCV apresenta imagens como NumPy arrays em ordem inversa), onde cada entrada é uma lista de um tuplo com dois valores: uma lista dos limites inferiores e uma lista de limites superiores.

Olhando para o exemplo: ([17, 15, 100], [50, 56, 200])
Este tuplo define que todos os pixels na imagem que têm a R >= 100, B >= 15 e G >= 17 e R <= 200, B <= 56 e G <= 50 serão considerados vermelhos.

Depois de se definirem as boundaries, foi utilizada a função cv2.inRange para fazer a deteção de cor. Foi também necessário converter os limites superior e inferior em NumPy arrays.

Para criar a imagem output, foi então aplicada a máscara criada simplesmente chamando a função cv2.bitwise_and(), mostrando apenas os pixels que têm um valor correspondente na máscara.

Deste modo, obteve-se o seguinte código:

```
boundaries = [[ ([58, 57, 101], [76, 95, 162]) ]]
boundaries += [[ ([26, 61, 111], [81, 144, 202]) ]]
boundaries += [[ ([44, 102, 167], [115, 169, 210]) ]]

def processImg(img):
    output = []
    for b in boundaries:
        for (l, u) in b:
            l = np.array(l, dtype = "uint8")
            u = np.array(u, dtype = "uint8")
            mask = cv2.inRange(img, l, u)
           output+=[cv2.bitwise_and(img,img,mask=mask)]

    final = output[0]
    for o in output:
        final = cv2.bitwise_or(final, o)
    return final
```

Obtiveram-se os seguintes resultados:

Figura 1: Imagem de teste Corrosion/18.jpg

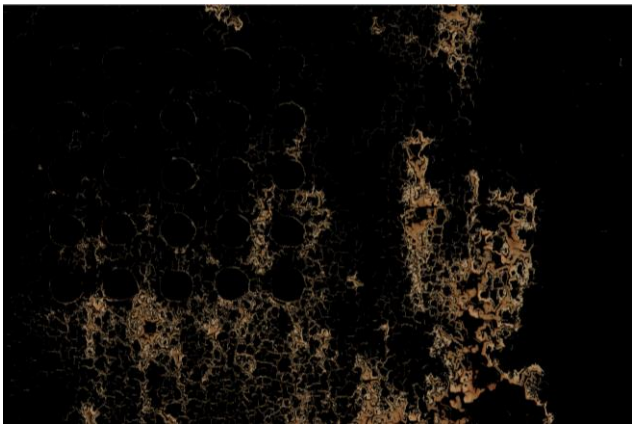

Figura 2: Primeira versão da Processed_Image_1 da imagem de teste Corrosion/18.jpg

Após analisar os resultados, concluiu-se que as zonas mais escuras da zona enferrujada não estavam abrangidas pela máscara. Foi então necessário adicionar mais um tuplo *boundaries: boundaries += [[([0, 20, 40], [50, 70, 150])]]*

Após esta alteração, obtiveram-se os seguintes resultados:

Figura 3: Segunda versão da Processed_Image_1 da imagem de teste Corrosion/18.jpg

PERCENTAGEM DE CORROSÃO

Considerou-se importante adicionar uma forma de o utilizador perceber que percentagem da imagem é que tinha ferrugem visto que certas imagens que contivessem pouca superfície enferrujada poderiam ser consideradas pouco relevantes.

Assim, foi criada a função *getPercent(img)* onde são contados os pixels da imagem que não apresentam uma cor negra e é retornada a percentagem da imagem que contém ferrugem.

Deste modo, sempre que o utilizador avalia a corrosão numa imagem, é lhe fornecido a quantidade de ferrugem que foi identificada pelo programa, como se pode verificar nas imagens seguintes:

Figura 4: Imagem de teste Corrosion/11.jpg

```
Insert the image's name: 11.jpg
Applying masks |##############################| 4/4
Processing |##############################| 426400/426400
Do you want to crop the image ? (y/n)
n
Percentage of rust in the image: 12.38%
```

Figura 5: Output do programa para a imagem Corrosion/11.jpg

RECORTE

Com o objetivo de poder selecionar uma determinada área da imagem para a correspondente avaliação e detecção de ferrugem, foi implementado um sistema de recorte de imagem. Este sistema permite selecionar um retângulo dentro da imagem original e submeter a parte selecionada para avaliação. Assim o utilizador seleciona, com o cursor, um ponto na imagem e arrasta até outro ponto onde depois pode confirmar premindo a tecla "Enter" ou voltar a "desenhar" o retângulo se não confirmar. O processo pode ser cancelado com a tecla "C". Esta ferramenta é útil pois não só permite fazer uma avaliação mais específica da imagem (e evitar falsos positivos), como também melhora o tempo de execução do programa.

Figura 6: ROI selector da imagem de teste Corrosion/22.jpg

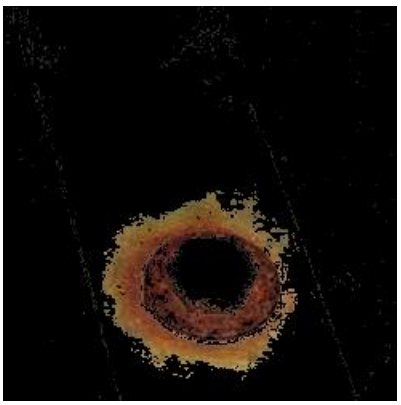

Figura 7: Processed_Image_1 da imagem de teste Corrosion/22.jpg já recortada

INTENSIDADE DE CORROSÃO

Considerou-se relevante a possibilidade do utilizador perceber que áreas da imagem tinham um nível de corrosão maior ou menor. Para isso, optou-se por definir 3 níveis de intensidade de corrosão nas quais se aplicam 3 cores: amarelo, laranja e vermelho Após aplicar alguns efeitos nas imagens processadas, chegou-se à conclusão que a partir de uma imagem a preto e branco se conseguia perceber que as cores mais claras correspondiam a um nível de corrosão baixo (amarelo) e as cores mais escuras correspondiam a um nível alto (vermelho). Assim, criou-se uma nova função getIntensityImg(img) onde se aplicou a função cv2.cvtColor(img, cv2.COLOR_BGR2DRAY) após o primeiro processamento (para eliminar as superfícies sem corrosão) ser aplicado. Depois, apenas se teve de avaliar a cor dos pixels e em que intervalo estes se encontram para atribuir-lhes uma das 3 cores anteriormente referidas na nova imagem.

Visto que já se está a percorrer todos os pixels da imagem, acrescentou-se também código para obter a percentagem de pixels não negros, evitando assim uma chamada da função getPercent(Img) que obrigava a percorrer todos os pixels da imagem uma vez mais.

Foram obtidos os seguintes resultados:

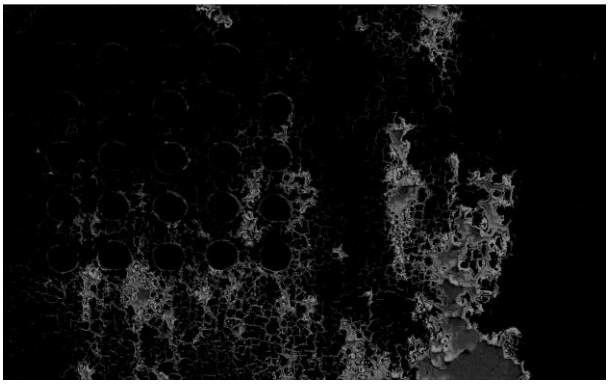

Figura 8: Black_&_White da imagem de teste Corrosion/18.jpg após o primeiro processamento

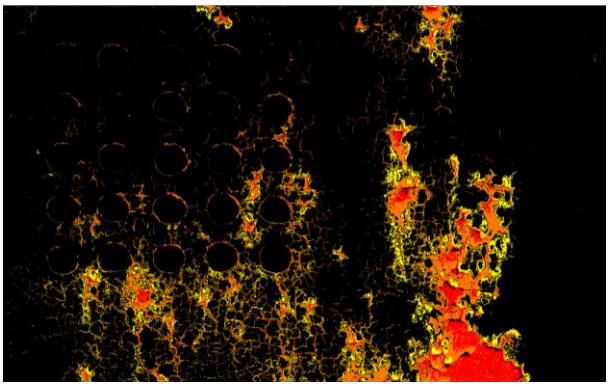

Figura 9: Processed_Image_2 da imagem de teste Corrosion/18.jpg

AVALIAÇÃO DA CORROSÃO EM MÚLTIPLAS IMAGENS

Considerou-se que num contexto mais real de uso da aplicação em contexto, a avaliação de ferrugem seria feita em múltiplas imagens. Deste modo, criou-se uma função que permite ao utilizador, a avaliação de múltiplas imagens a partir do nome do diretório. Assim, existe um processamento e são apresentadas as percentagens de área com ferrugem detetada para cada uma das imagens presentes na pasta selecionada. O utilizador pode também definir uma percentagem mínima de área com corrosão para que o output seja uma lista de imagens que satisfaçam essa condição.

```
RUST DETECTOR

Select an option:
1 Detect rust on a single image
2 Detect rust on several images
3 Quit
2
Select the folder where the image is:
multiple
['11.jpg', '12.jpg', '13.jpg', '14.jpg', '15.jpg']
Do you want to set a minimum percentage of rust per image ? (y/n)
y
Insert the minimum percentage
20
11.jpg - 12.38%
12.jpg - 25.01%
13.jpg - 32.08%
14.jpg - 49.59%
15.jpg - 17.66%
List of images:
['12.jpg', '13.jpg', '14.jpg']

Select an option:
1 Detect rust on a single image
2 Detect rust on several images
3 Quit
```

Figura 10: Análise de várias imagens dentro do diretório "multiple" e retorno de uma lista com as imagens com mais de 20% de área com corrosão detetada

V. RESULTADOS

Neste capítulo serão apresentadas imagens de teste e o output final do programa assim como uma breve descrição. O processamento da imagem Corrosion/15.jpg teve no geral um bom resultado. No entanto alguns problemas foram evidentes como o facto dos pretos não serem identificados como ferrugem.

Figura 11: Imagem de teste Corrosion/15.jpg

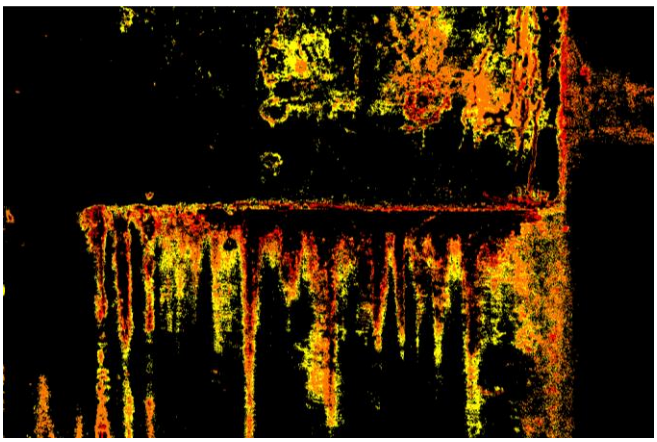

Figura 12: Processed_Image_2 da imagem de teste Corrosion/15.jpg

Pelo teste feito à imagem Corrosion/16.jpg concluiu-se que algumas tonalidades de verde são captadas pelas máscaras e o reflexo do céu/luz na ferrugem faz com que esta não seja identificada como tal.

Figura 13: Imagem de teste Corrosion/16.jpg

Figura 14: Processed_Image_2 da imagem de teste Corrosion/16.jpg

A imagem de teste No_Corrosion/4.jpg não deveria apresentar ferrugem e de facto, pelo teste feito, apenas são identificados alguns pixels nas sombras dos tubos.

Apenas 0.47% dos pixels foram identificados como ferrugem, o que pode ser descartado ao avaliar várias imagens num diretório e filtrando por uma percentagem mínima de ferrugem (Opção 2 do menu inicial).

Figura 15: Imagem de teste No_Corrosion/4.jpg

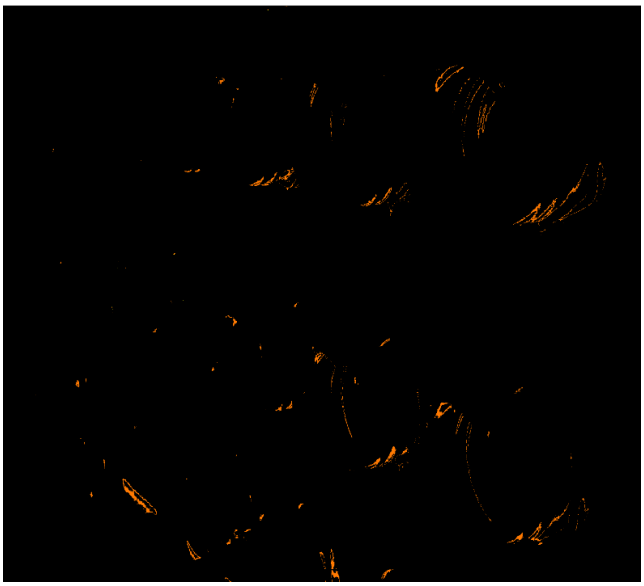

Figura 16: Processed_Image_2 da imagem de teste No_Corrosion/4.jpg

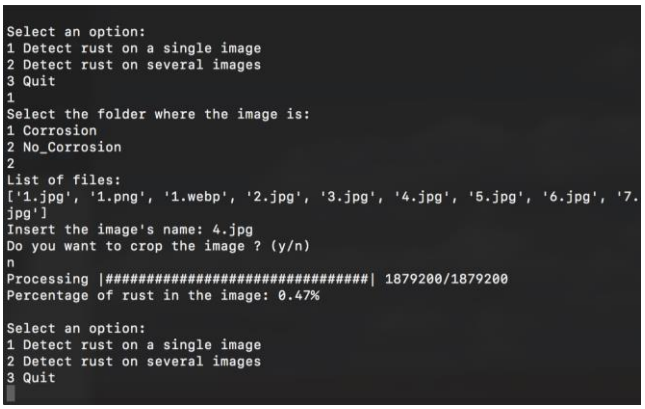

Figura 17: Teste da imagem No_Corrosion/4.jpg

Observando o resultado do processamento nas imagens Corrosion/17.jpg, e Corrosion/18.jpg (Anteriormente apresentada) pode-se concluir que existem ótimos resultados na deteção de corrosão em imagens de superfícies planas sem reflexos de luzes ou sombras.

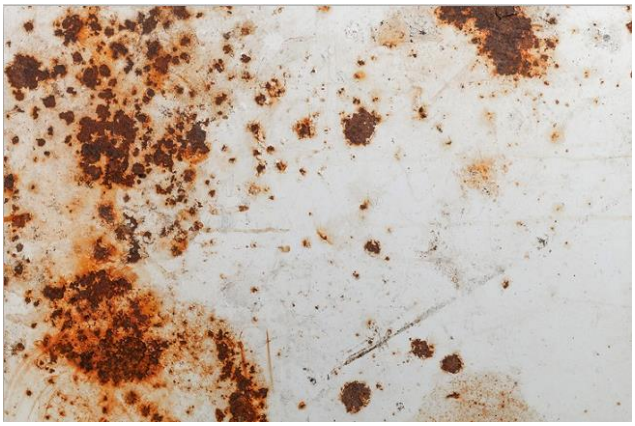

Figura 18: Imagem de teste Corrosion/17.jpg

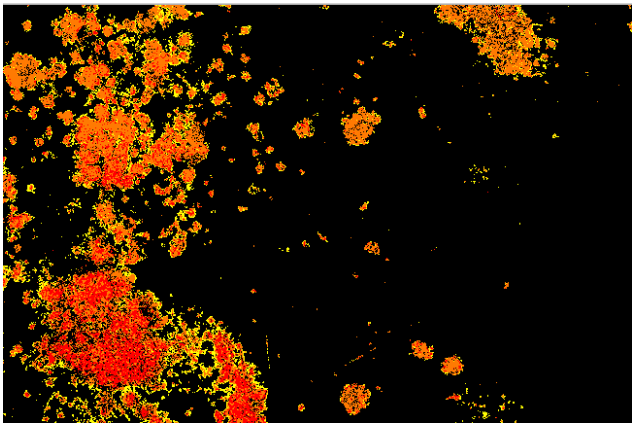

Figura 19: Imagem de teste Corrosion/12.jpg

VI. CONCLUSÕES

Mesmo com resultados bastante positivos após implementação, o resultado não é perfeito.

Considerando que se baseia, numa grande parte, na comparação com a cor característica da ferrugem, imagens que contenham cores semelhantes à mesma, podem levar a falsos positivos.

Algumas tonalidades de verde, amarelo, castanho e cores escuras são muitas vezes consideradas ferrugem erradamente, fazendo com que haja bastantes erros em imagens que contenham vegetação, superfícies amarelas, terra ou algumas sombras.

As cores azul, branco e cinzento não são consideradas, na maior parte das vezes, como zonas de ferrugem. Pelo contrário, são sinal de metal limpo. Considerando que o reflexo toma precisamente essas cores, o ideal para uma maior precisão nos resultados, seria que todas as imagens fossem de superfícies planas, ausência de sombras e que houvesse o mínimo de luz direta para evitar reflexos. Foi em imagens que apresentam estas características que o trabalho desenvolvido obteve o melhor desempenho e ficámos bastante satisfeitos com os resultados obtidos.

VII. TRABALHO FUTURO

O processo de detecção de ferrugem é um problema do mundo real bastante complexo.

Deste modo, para trabalho futuro, seria importante a distinção das diversas superfícies e correspondente detecção de corrosão com uma taxa de precisão superior.

Um estudo mais profundo das cores de ferrugem e uma consequente alteração nas máscaras aplicada nas imagens poderia resultar em avaliações mais precisas e com menos falsos positivos.

Também acreditamos que o uso de inteligência artificial e aprendizagem automática poderia levar a resultados melhores.

VIII. REFERÊNCIAS

[1] https://htmlcolorcodes.com/

[2] https://imagecolorpicker.com/

[3] https://www.pyimagesearch.com/2014/08/04/opencv-python-color-detection/

[5] https://answers.opencv.org/question/178394/detection-of-rust-with-opencv-python/

[1] https://pypi.org/project/progress/

[2] https://learnopencv.com/how-to-select-a-bounding-box-roi-in-opencv-cpp-python/