Computação Gráfica

Adalberto Shindy Hassobe	11295762
Bruno de Santana Braga Contreras	11208072
Lucas Mendes Sales	11270736
Luiz Felipe Vieira Cordeiro	9844811
Victor Ferreira Lopes	11270802

Projeto de processamento gráfico

SÃO PAULO

2022

1. Objetivo

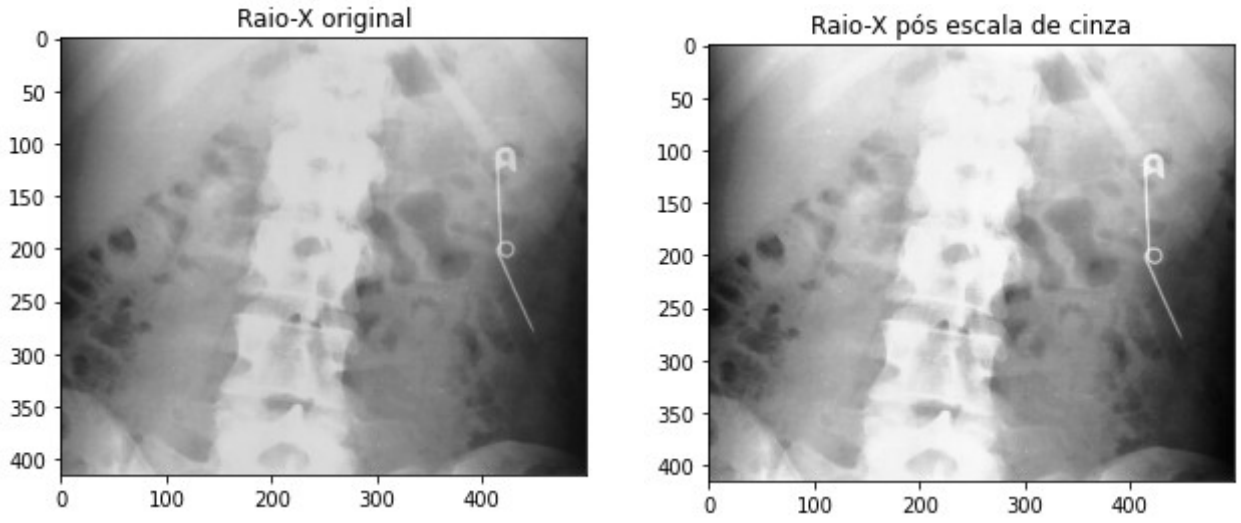
O presente projeto visa detectar possíveis corpos estranhos, comumente ingeridos por crianças, presentes em radiografias, sem auxílio de inteligência artificial. A ideia consiste em processar várias imagens de radiografias e separá-las em diretórios diferentes: um exclusivo para as imagens com detecções e outro para imagens sem detecções.

De acordo com estudos pediátricos sobre o tema, grande parte de corpos estranhos sintéticos aspirados possuem formas mais regulares (moedas, tampas de canetas, bolinhas, entre outros) - https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7947743/. Sob as condições corretas, esses objetos são facilmente diferenciáveis de órgãos e estruturas ósseas através de radiografias. Portanto, a proposta do projeto visa utilizar detectores de linhas e círculos para ajudar a classificar arquivos de imagens radiográficas, puramente com o processamento das imagens.

2. Desenvolvimento

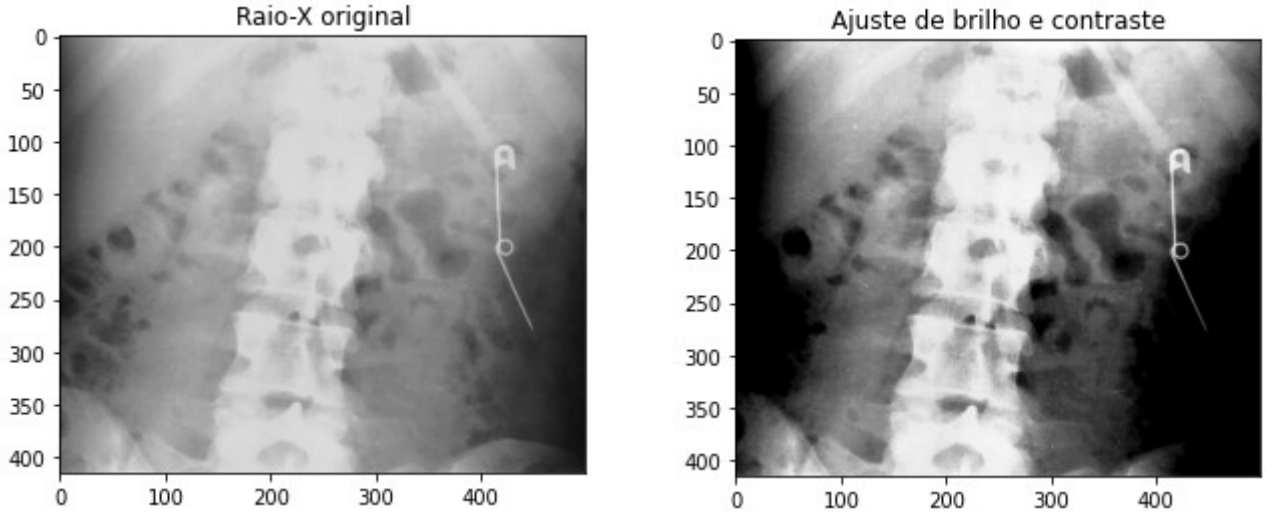
O projeto foi concebido através da linguagem Python, utilizando as bibliotecas OpenCV (https://opencv.org/), numpy (https://numpy.org) e matplotlib (https://matplotlib.org/).

Uma vez carregada a imagem, ela é inicialmente submetida a conversão para escala de cinza. Embora muitas radiografias já venham nessa configuração, chegou-se ao consenso de que seria mais sensato prevenir o processamento de possível artefatos que pudessem comprometer as próximas etapas.



Comparação pós-conversão para escala de cinza

A imagem resultante é submetida a uma redução de brilho e um moderado aumento do contraste a fim de atenuar a interferência de outras tonalidades claras oriundas da estrutura óssea e aumentar o destaque para os objetos presentes na radiografia. Para mensurar o grau de ajuste, foi utilizada uma fórmula para normalizar os valores da escala dentro do intervalo [-127, 127], similar a escala utilizada no GIMP. A equação foi obtida através de fóruns e artigos sobre o tema de processamento de imagens e está devidamente relatada nas referências do projeto.



Comparação pós-processamento de brilho e contraste

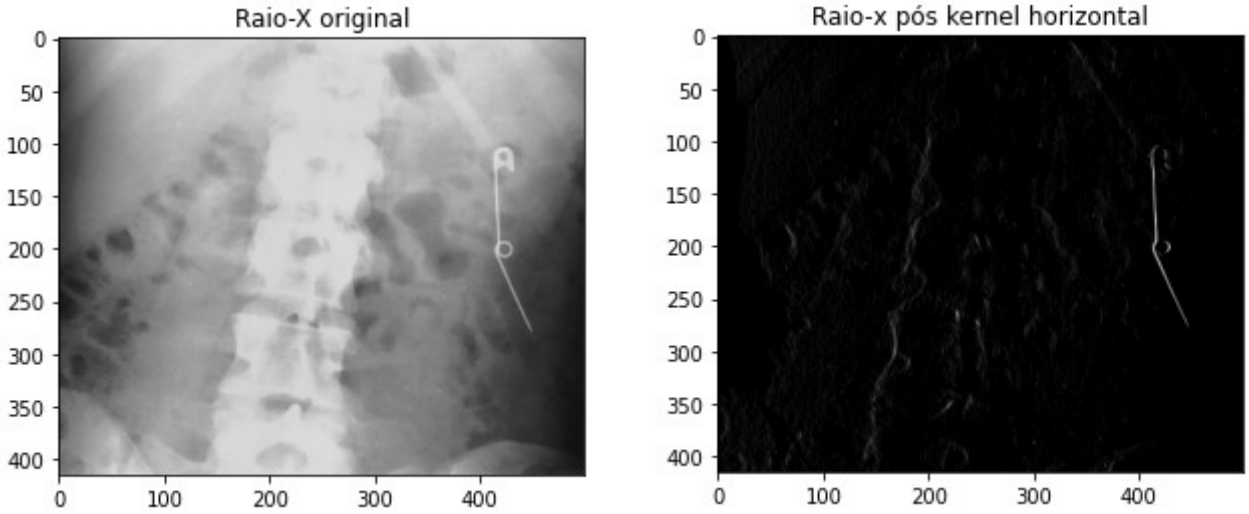
A radiografia ajustada em termos de brilho e contraste é posteriormente submetida ao processo de detecção de bordas pelo algoritmo Sobel. Primeiro, ela é submetida ao kernel horizontal:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Kernel horizontal (G_X)

A função *cv2.Sobel()* recebeu os seguintes parâmetros:

- A imagem resultante das transformações anteriores;
- A ordem da derivada na direção X (horizontal)
- A ordem da derivada na direção Y (vertical)
- O tipo de imagem: pixels com 16 bits e 1 canal de cor (CV_16UC1)

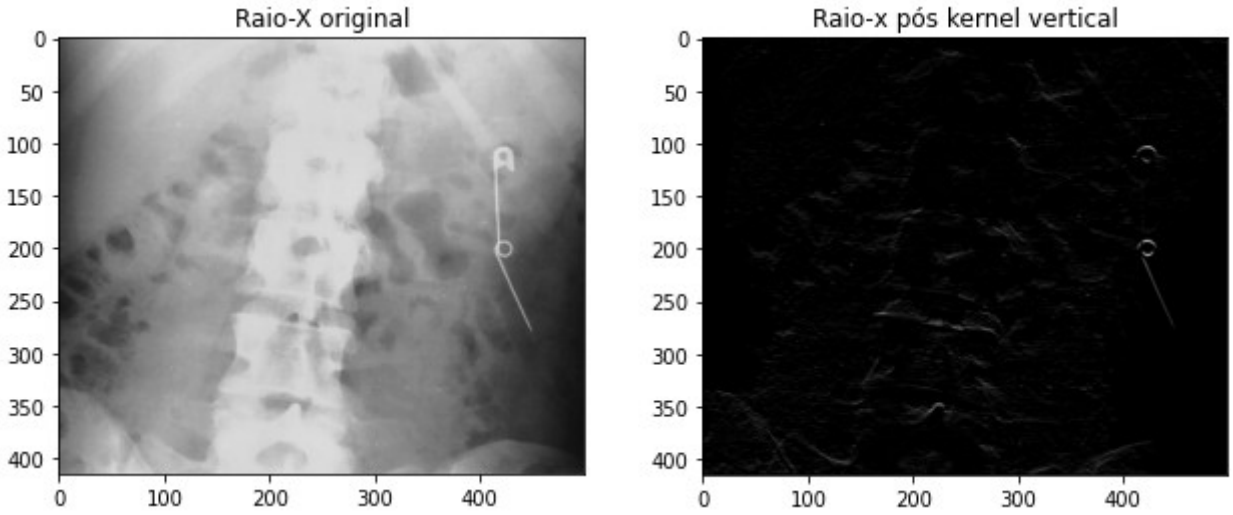


Comparação pós-Sobel (G_X)

A aplicação do algoritmo de Sobel repete-se, mas no sentido vertical (derivada X = 0, derivada Y = 1)

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

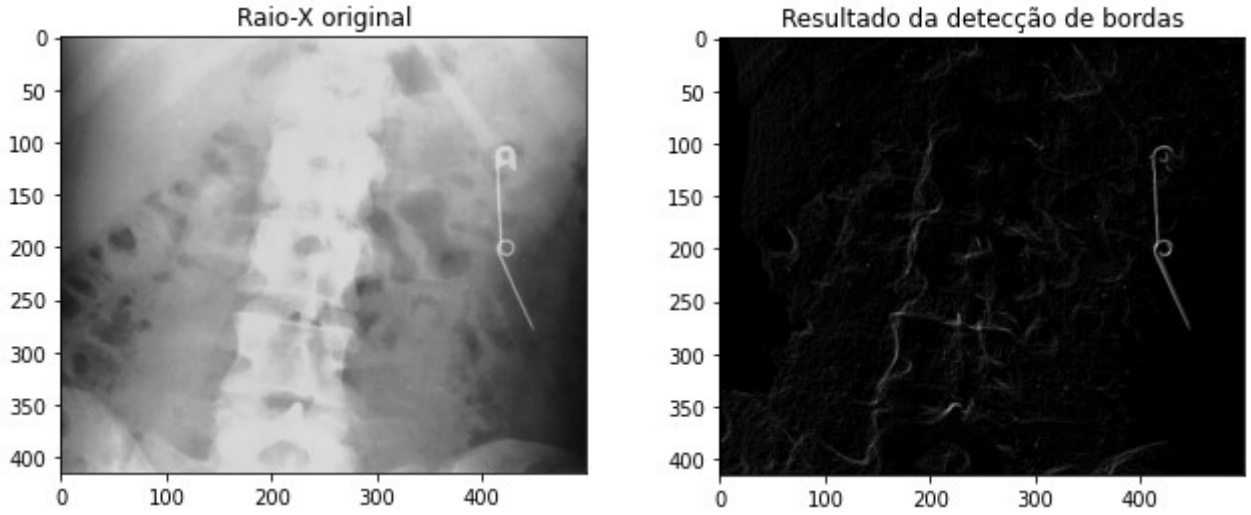
Kernel vertical (G_Y)



Comparação pós-Sobel (G_Y)

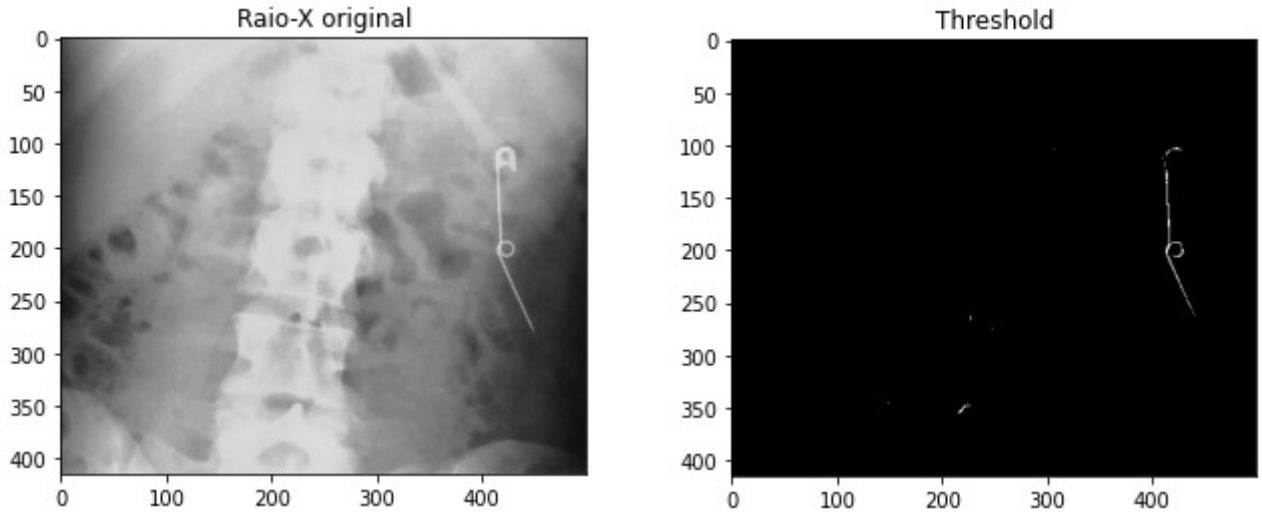
O resultado decorrente das detecções nos sentidos vertical e horizontal é obtido com a conjugação das duas imagens. Primeiro, ambas são convertidas para 8 bits com a função *cv2.convertScaleAbs()* e, em seguida, elas sofrem uma "justaposição" ponderada por meio de pesos com a instrução *cv2.addWeigthed()*. A ideia é ponderar o array de

pixels de forma equilibrada, ou seja, igual (50%) entre as duas imagens (uma submetida ao kernel G_X e a outra submetida ao G_Y. O resultado percebido é a junção das duas imagens.



Comparação pós-Sobel (G_X e G_Y)

Com a obtenção das bordas, a imagem sofre o processamento de Threshold com o limiar igual a 121, um valor médio que apresentou bons resultados nos testes com as imagens disponíveis.

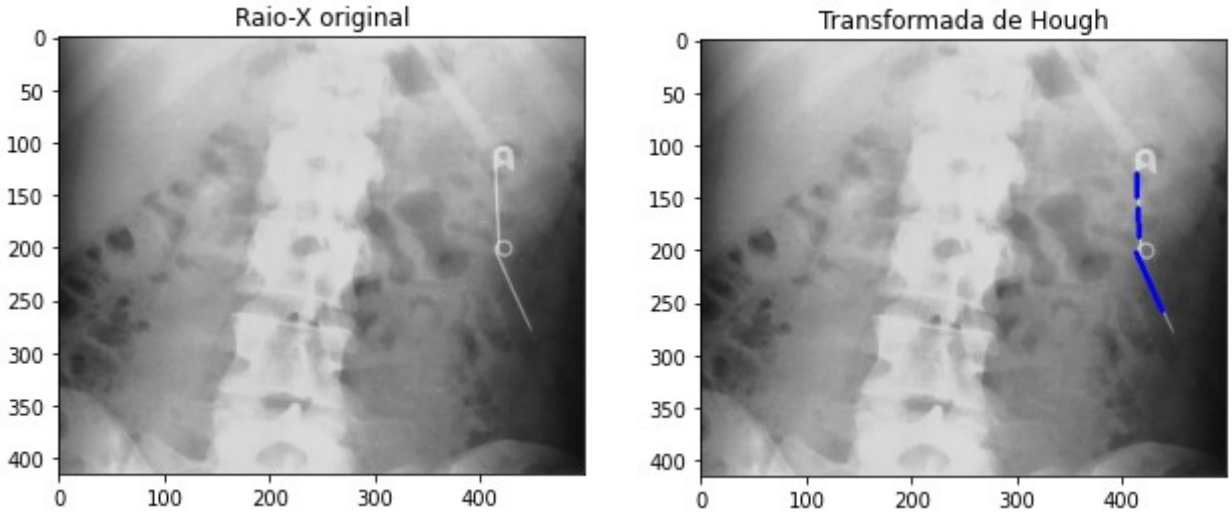


Comparação pós-Threshold

Inicia-se a detecção de linhas através da transformada de Hough probabilística, por meio da instrução cv2.HoughLinesP com os seguintes parâmetros:

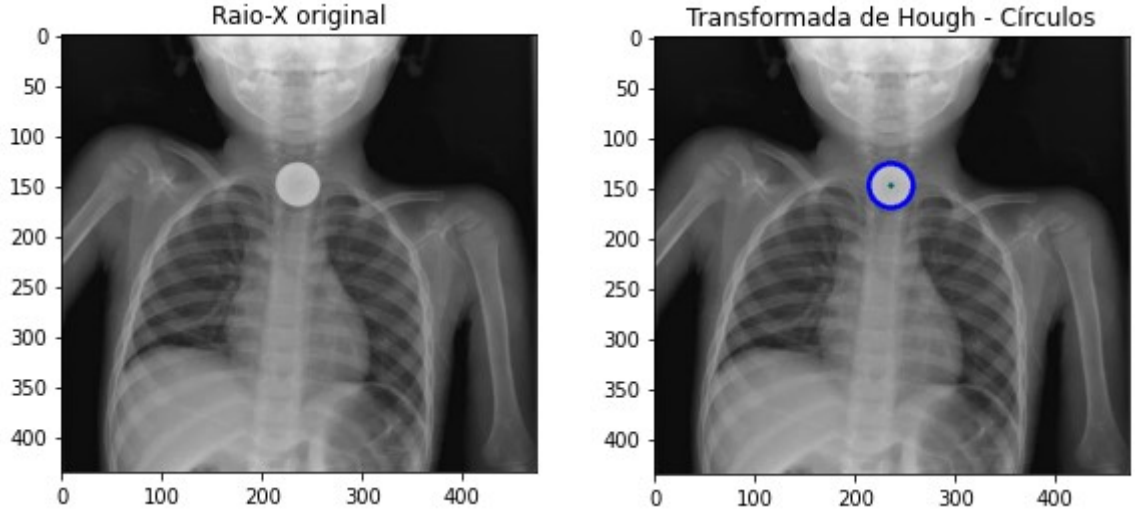
- Imagem binária oriunda do processo de Threshold
- Limite de votos (30): necessário para delimitar o que deve e não deve ser reconhecido como linha, através de um acumulador de votos que atua em função da representação paramétrica das coordenadas (espaço de Hough)

- Comprimento mínimo (3): limita a detecção de linhas ao comprimento especificado
- Intervalo máximo (1): determina a tolerância de falhas (*gaps)* entre os segmentos de uma mesma reta para assumir a detecção como uma linha contígua.



Finalizada a detecção de linhas, inicia-se a detecção de círculos através da função *cv2.HoughCircles()* com os seguintes parâmetros:

- Imagem com os ajustes de brilho e contraste
- Threshold do algoritmo Canny (200): determina o limite para a detecção de bordas pelo método Canny
- Threshold da intensidade de cor para detecção do centro dos círculos (34)
- Intervalo mínimo de detecção entre os círculos (número de linhas / 8)
- Raio mínimo (0): limite inferior para determinar a detecção de círculos (0 significa que não há restrições)
- Raio máximo (70): limite superior para determinar a detecção de círculos



Comparação pós-Hough (Círculos)

Por fim, a imagem é salva no diretório correspondente. Se houver alguma detecção, ela recebe o prefixo padronizado “DETECTED_<nome da imagem original>”, senão recebe o prefixo “NOT_DETECTED_<nome da imagem original>. Todas as saídas geram imagens PNG.

3. Código

O código-fonte, em anexo, encontra-se com os comentários pertinentes para situar o leitor sobre as operações realizadas. As referências externas também estão disponíveis nos trechos inspirados ou literalmente aproveitados de outras fontes. As funções estão declaradas no início do corpo de texto e, ao final, está disposta a função principal (*main()*).

4. Ambiente de criação do projeto

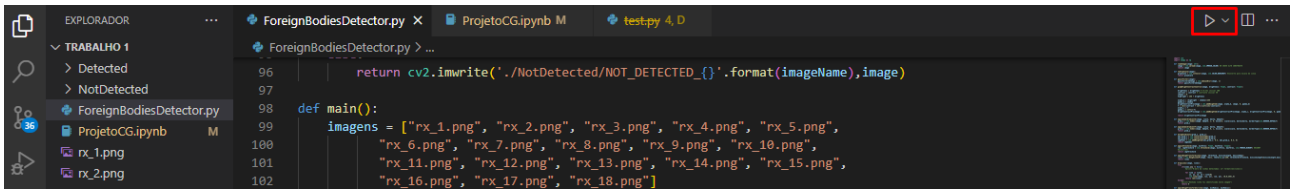
A tabela abaixo descreve os componentes básicos para a criação do projeto de processamento

Componente	Versão
Windows 10 (x64)	21H2
pip (Packet installer for Python)	22.1.2
python	3.10.2
Visual Studio Code	1.69.2
openCV	4.5.4.58
numpy	1.23.1
matplotlib	3.5.2

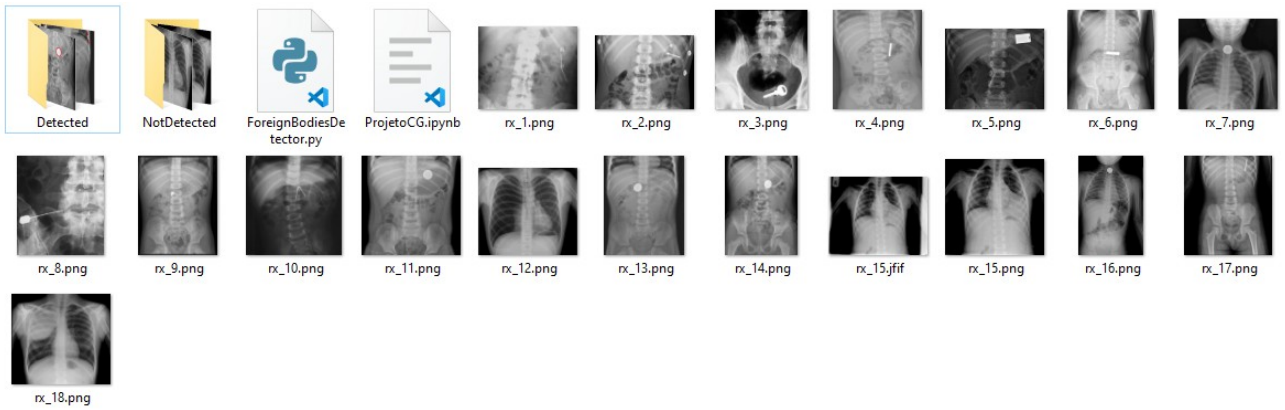
5. Funcionamento

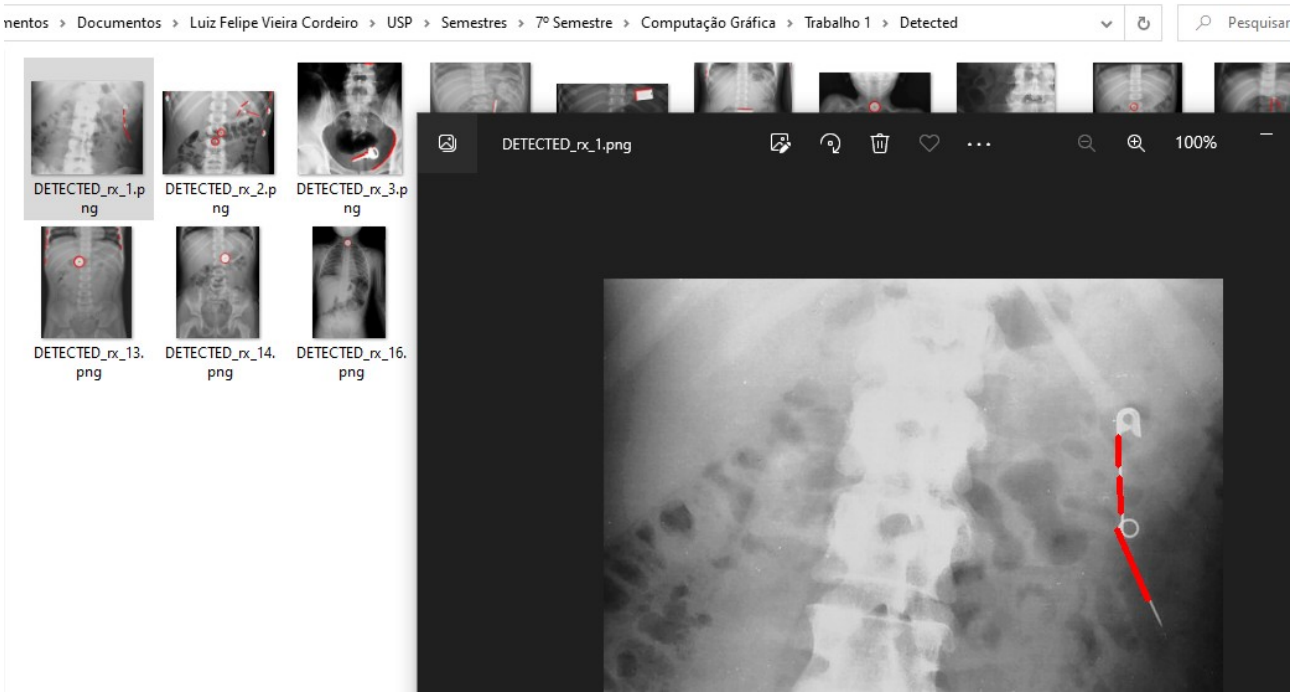
A operação do programa é bem simples, basta executar o código-fonte com alguma IDE Python ou no próprio terminal, na pasta com as radiografias para iniciar a detecção e classificação das imagens

A princípio, as pastas “Detected” e “NotDetected” estão vazias.



Com a execução do código, as mesmas pasta devem conter as respectivas imagens classificadas e com as indicações de detecção.





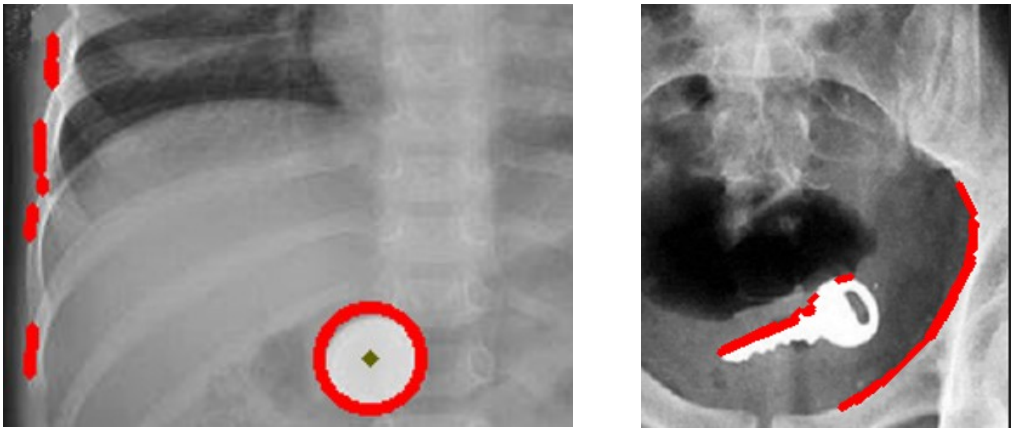
6. Testes

O projeto contou com a análise de 17 radiografias, selecionadas de bancos de dados dedicados ao tema (https://medpix.nlm.nih.gov/), e também obtidas pelo buscador de imagens do Google.

Entre as imagens, temos a seguinte configuração de casos:

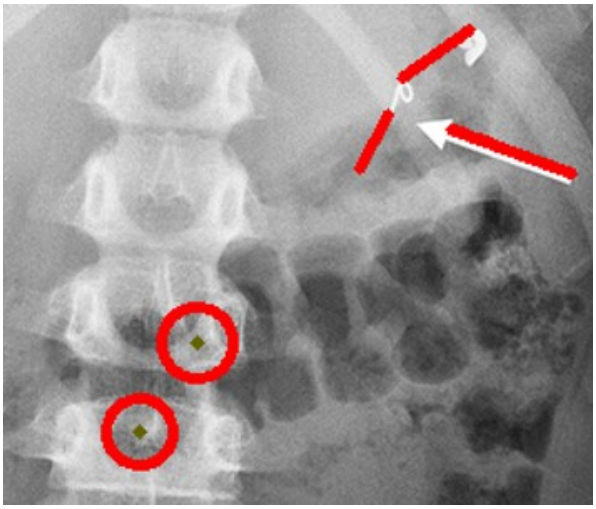
- 6 radiografias com objetos circulares
- 7 radiografias com objetos de contornos retilíneos
- 3 radiografias sem corpos estranhos evidentes
- 1 radiografia com um corpo com contornos curvilíneos e retilíneos

Felizmente, não houve nenhum caso no teste de falsos negativos, dado que nenhuma imagem com corpos estranhos foi categorizada como se estivesse ausente de detecções. Todas as imagens com objetos estranhos ao organismo foram detectadas, embora algumas tenham detectado pequenas linhas no contorno de alguns ossos devido ao brilho destes.



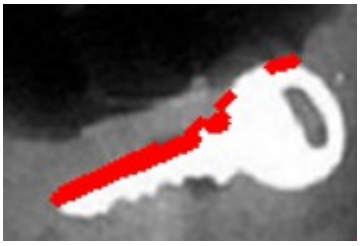
Detecção de linhas em ossos

Outro teste revelou a detecção de círculos em função das vilosidades presentes no intestino expostas na radiografia



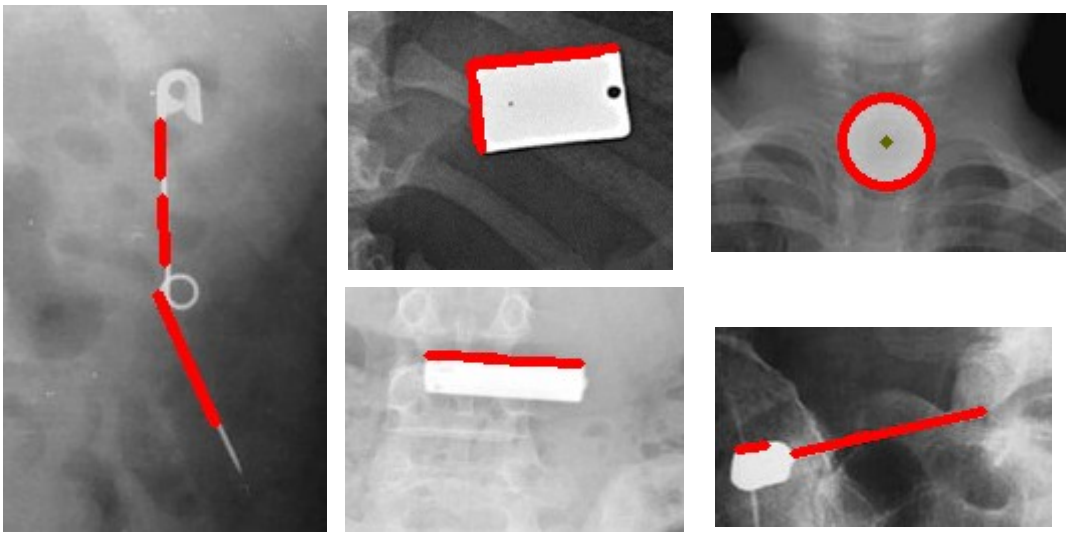
Vilosidades intestinais detectadas como círculos

Uma das imagens envolve a figura de uma chave convencional que, apesar de possuir contornos curvos e retos, apenas foram detectadas as retas presentes.



Chave detectada apenas em função das retas

Na maior parte dos casos, o resultado foi mais preciso:



Alguns casos de detecção bem-sucedidos

7. Referências

Todas as transformações foram estudadas através da documentação oficial do OpenCV:

- Escala de cinza: https://docs.opencv.org/3.4/db/d64/tutorial_js_colorspaces.html
- Sobel: https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html
- Threshold: https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html
- Hough (linhas): https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html
- Hough (círculos): https://docs.opencv.org/3.4/d4/d70/tutorial_hough_circle.html

Algumas operações de suporte também foram exploradas pela documentação oficial do OpenCV:

- Ler e salvar imagens: https://docs.opencv.org/3.4/d4/da8/group__imgcodecs.html

A expressão para controlar o contraste e brilho foram obtidas em um fórum sobre edição de imagens:

- https://codestack.club/questions/1935302/brilho-e-contraste-do-opencv-como-no-gimp