Universidade do Minho
Escola de Engenharia
Mestrado integrado em Engenharia Informática

Visão por Computador

Ano Letivo de 2017/2018

Toturial 2: Image Recognition

Luís Marques (a68691)

Miguel Silva (a73137)

Janeiro, 2018

VC

Programa de reconhecimento de objetos circulares

1. Leitura da imagem

Para iniciar este programa é necessário no terminal do *matlab* invocar o comando *"image_recognition".*

Posto isto, o programa irá iniciar e perguntar qual o caminho para a imagem. Com o caminho introduzido pelo utilizador o programa lê a imagem (*imread*) e converte-a para tons de cinzento *(rgb2gray).*

2. Introdução de ruído e consequente filtro

Após a leitura da imagem é perguntado ao utilizador qual o ruído que quer introduzir à imagem e de seguida qual a variância ou a intensidade desse mesmo ruído *(imnoise).*

Se o ruído for do tipo *"Salt & Pepper"* é usada a fórmula *imnoise (img_gray,'salt & pepper', intensidade),* caso o ruído seja do tipo Gaussiano, é usado a fórmula *imnoise (img_gray,'gaussian',0, variância}).*

2.1. Salt & Pepper

Seguem-se os resultados da adição de ruido "*Salt & Pepper"* com densidade 0.15 para cada uma das imagens fornecidas.

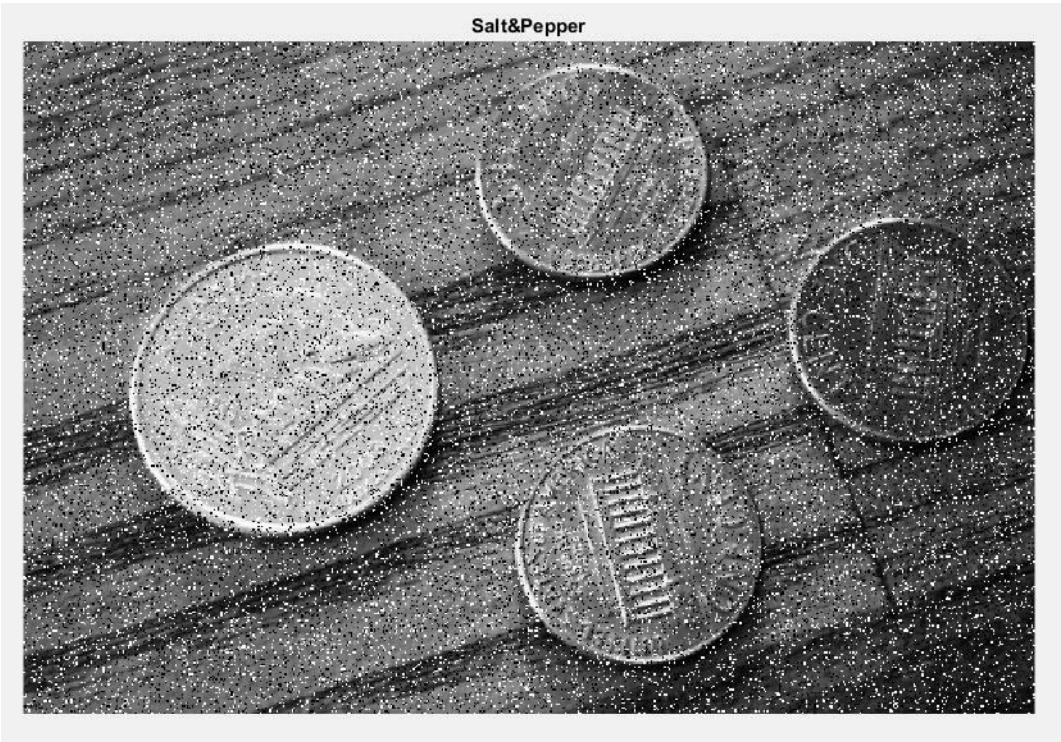

Figura 1: coins.jpg com ruído "Salt & Pepper" de densidade 0.15

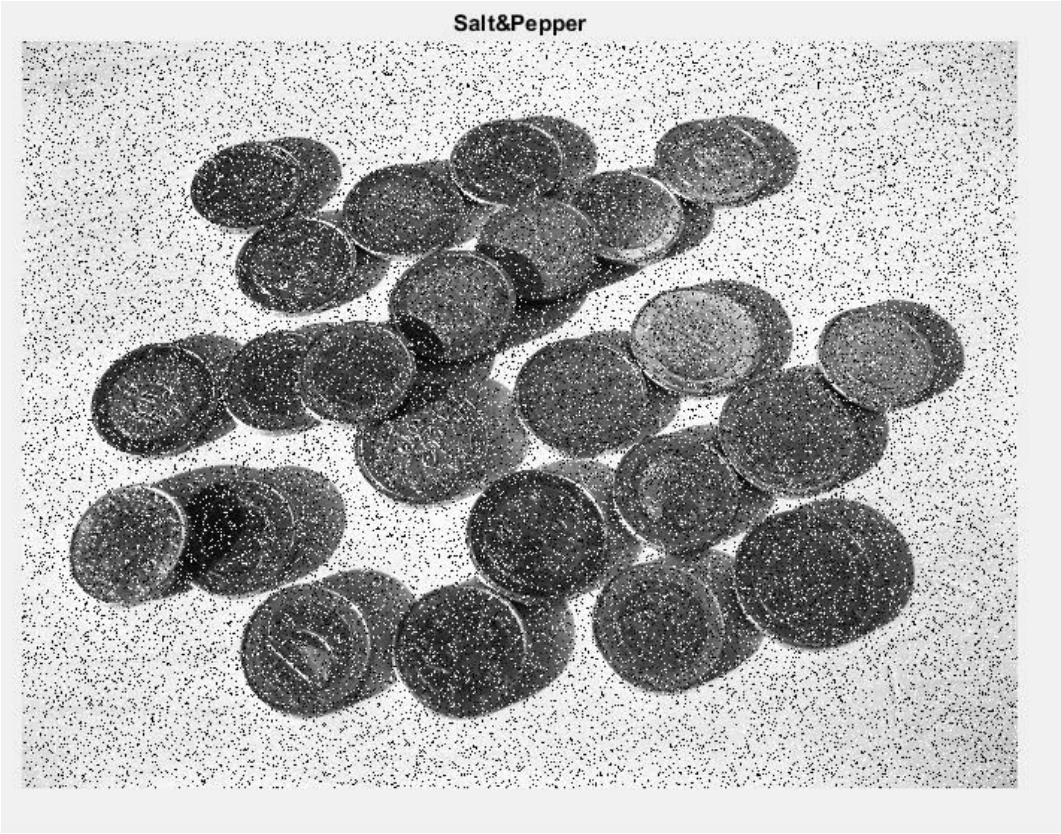

Figura2: coins2.jpg com ruído "Salt & Pepper" de densidade 0.15

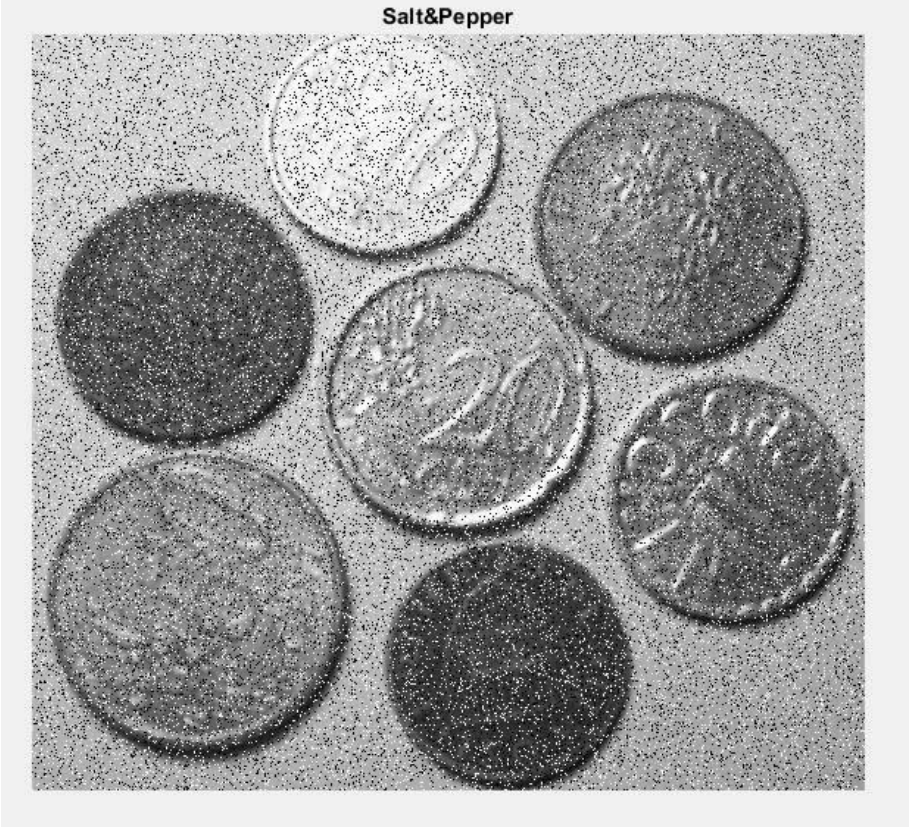

Figura 3: coins3.jpg com ruído "Salt & Pepper" de densidade 0.15

2.2. Gaussiano

Para apresentar o resultado da adição de ruído Gaussiano nas três imagens, escolheu-se utilizar uma média de 0 e variância 0.05.

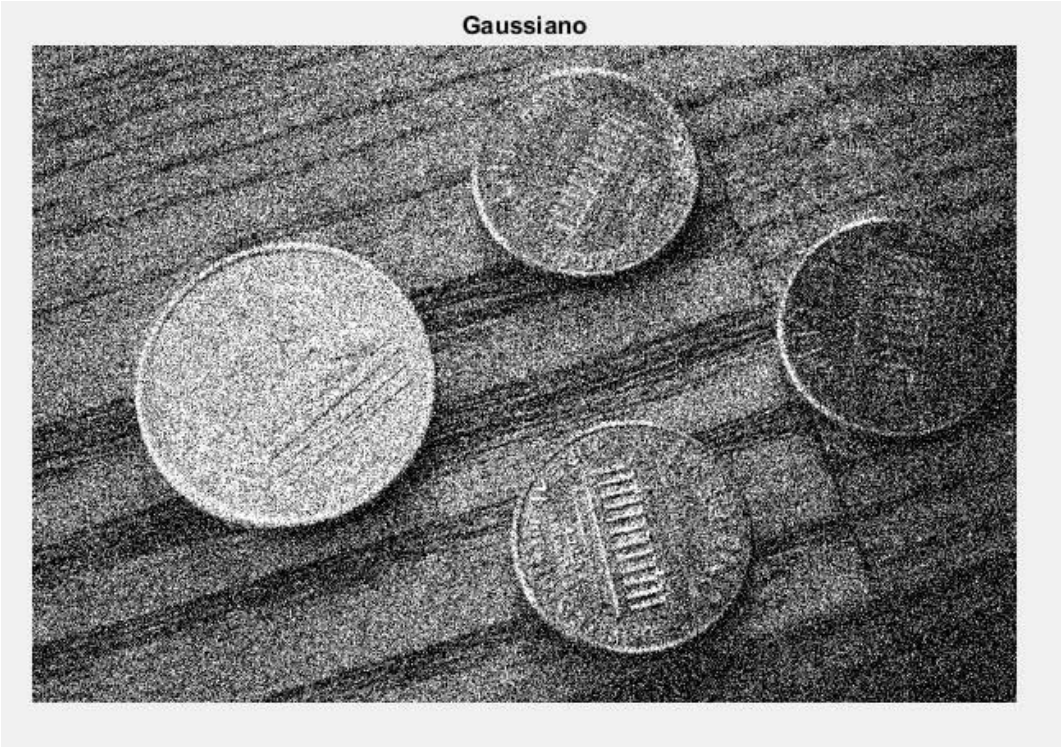

Figura 4: coins.jpg com ruído Gaussiano com média 0 e variância 0.05

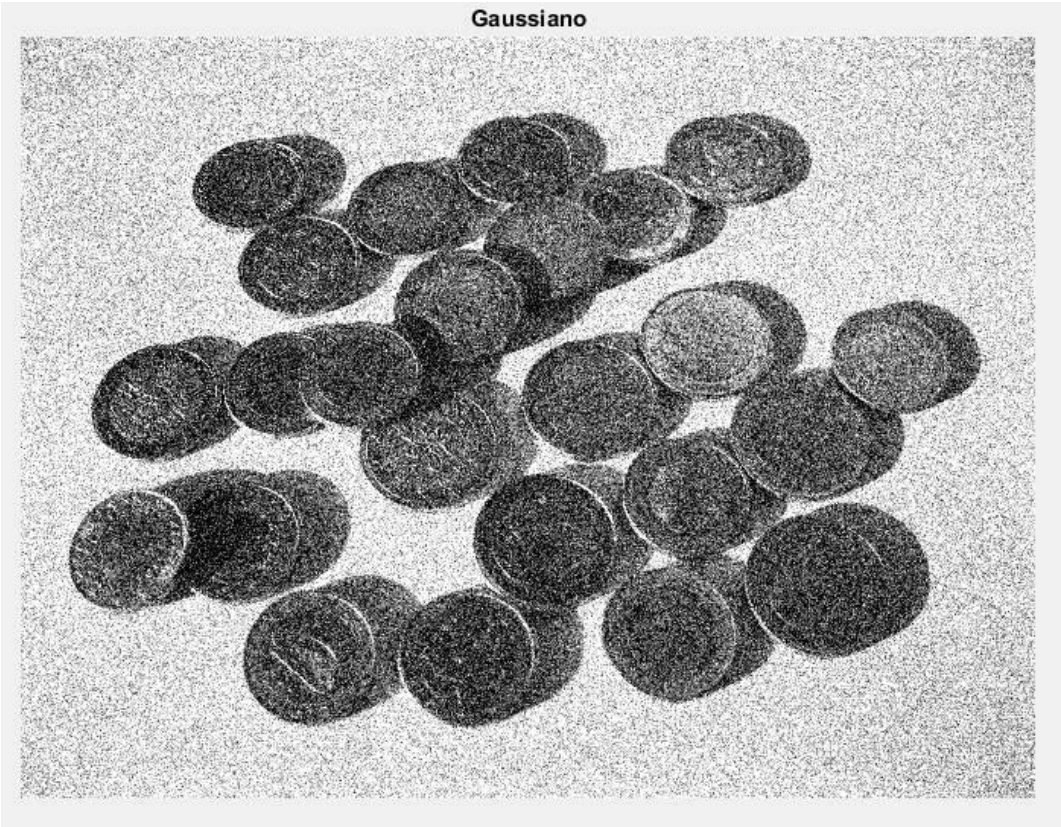

Figura 5: coins2.jpg com ruído Gaussiano com média 0 e variância 0.05

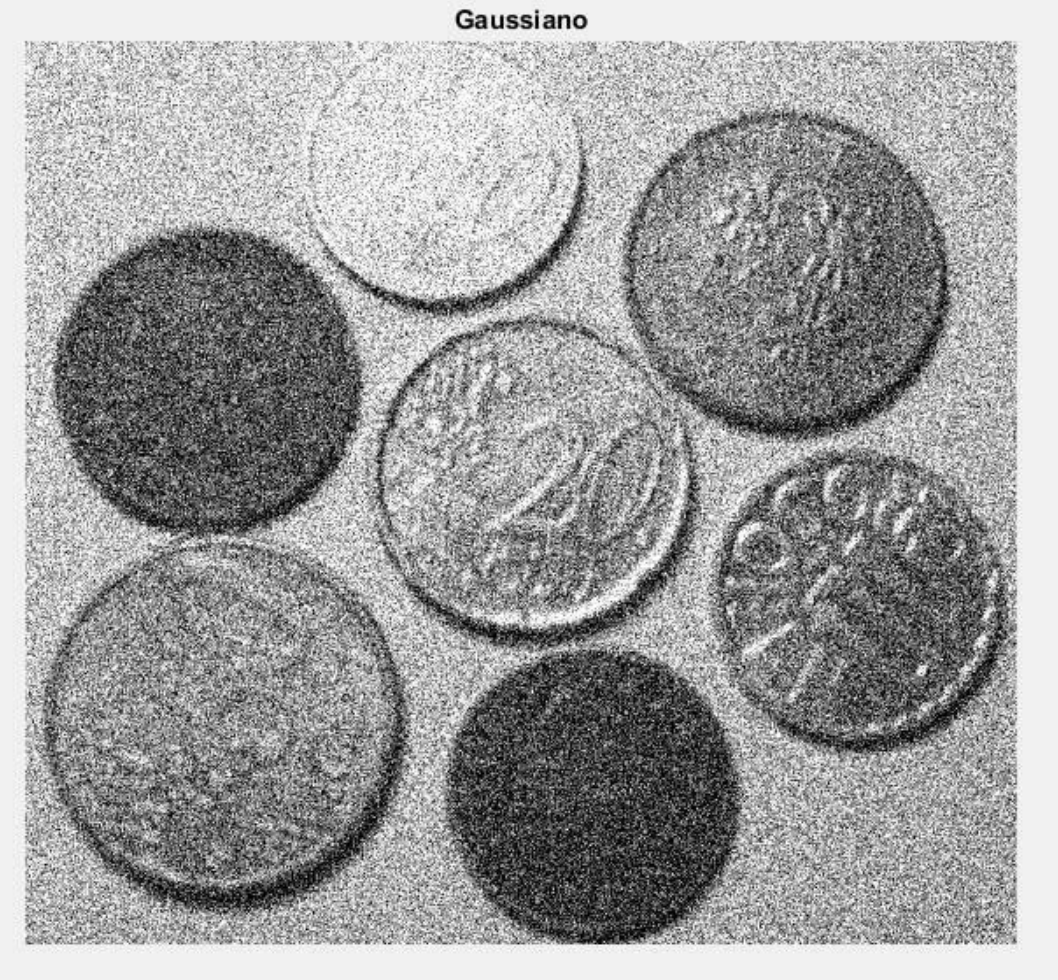

Figura 6: coins3.jpg com ruído Gaussiano com média 0 e variância 0.05

3. Pré-processamento de imagem

Com testes empíricos encontrámos os melhores resultados com o uso de uma função de pré-processamento de imagem. Após a aplicação do ruído é feita uma equalização de contraste para reduzir a intensidade da imagem (histeq).

Posto isto, é aplicado um filtro à imagem na tentativa de remover o ruído, no caso do ruído ser Gaussiano, o filtro é o de Gauss *(imgaussfilt),* caso o ruído seja *"Salt & Pepper",* o filtro a aplicar será o filtro das medianas (*medfilt2*).

Figura 7: Filtro da mediana SNR=14.0124 dB

Figura 8: Filtro da mediana SNR=13.8906 dB

Figura 9: Filtro da mediana SNR=13.8982 dB

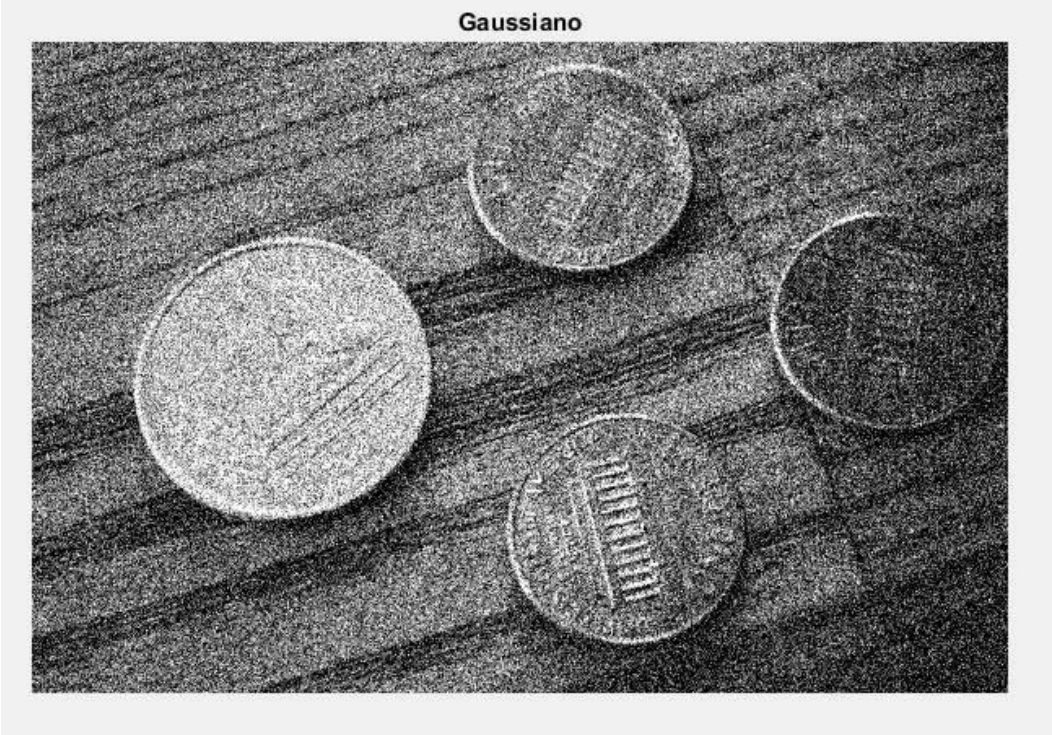

Figura 10: Filtro Gaussiano SNR=13.8611 dB

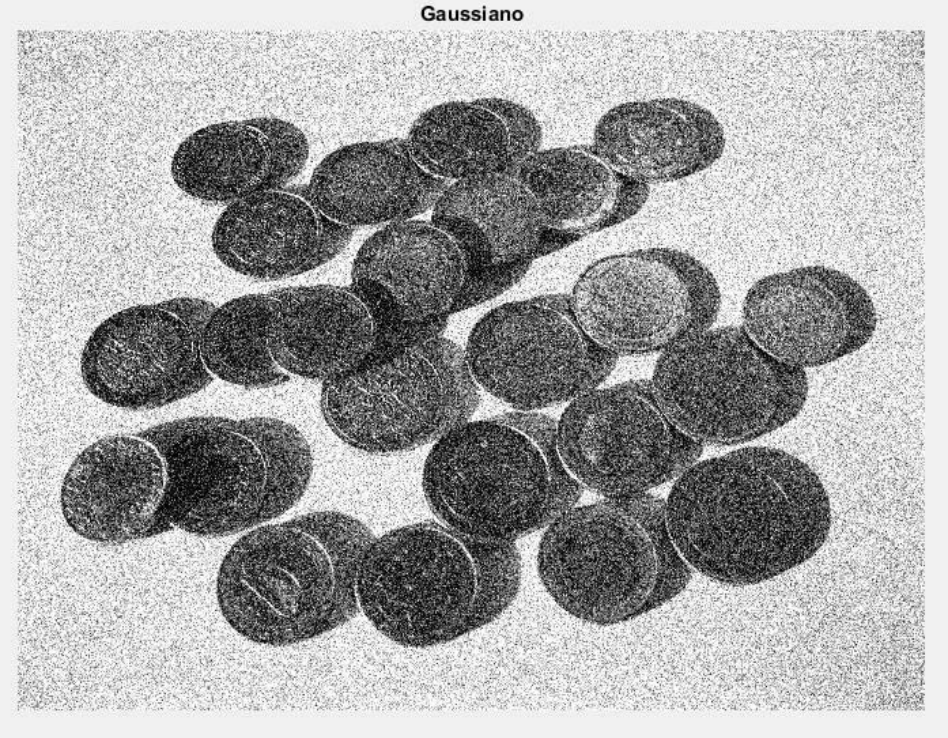

Figura 11: Filtro Gaussiano SNR=12.2712 dB

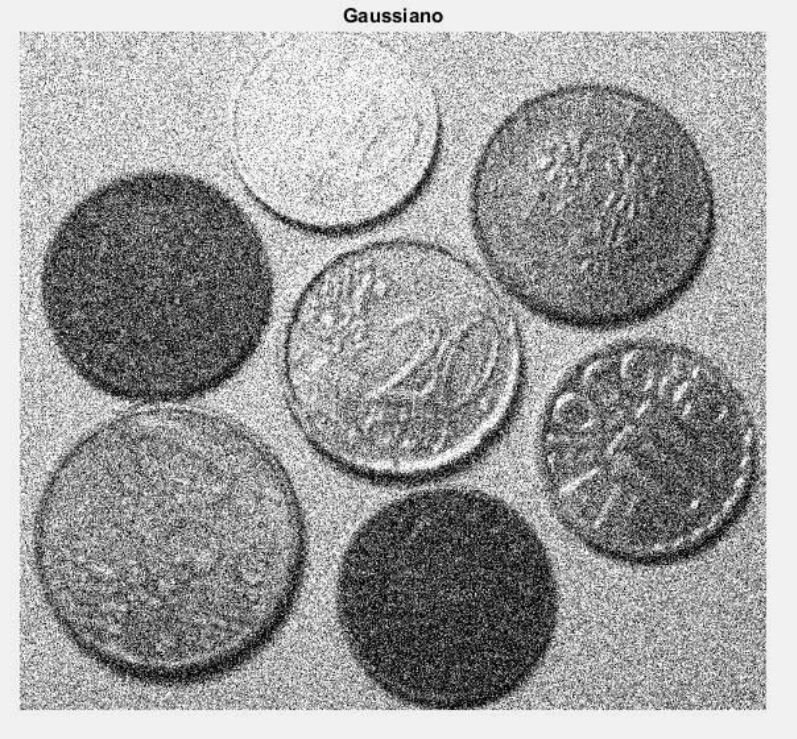

Figura 12: Filtro Gaussiano SNR=13.8197 dB

4. Preenchimento da imagem

Com o intuito de obter uma imagem melhor definida, tentamos preencher o fundo da imagem com enchimento de pixels através da função *imfill* e o atributo *holes*. Na imagem seguinte

podemos ver o efeito desta otimização após ser aplicado o filtro mediano na sequência da introdução de ruído *"Salt & Pepper*" com densidade 0.15.

Figura 13: Preenchimento da imagem

5. Segmentação da imagem

Para a segmentação da imagem o grupo optou por escolher a segmentação *Kmeans* (kmeans_seg). Nas imagens seguintes podemos ver como ficam as imagens com segmentação *Kmeans*, após ser introduzido o ruído *"Salt & Pepper"* com densidade 0.15 e o posterior processamento da imagem.

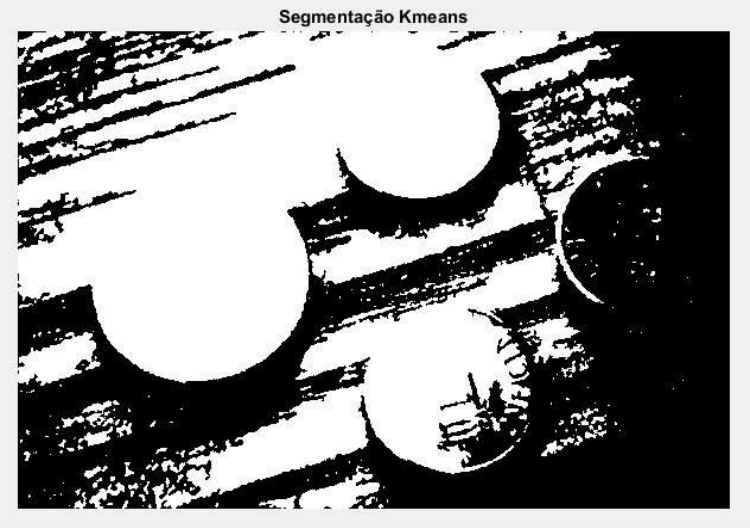

Figura 14: Segmentação Kmeans da coins.jpg

*Figura 15: **Segmentação Kmeans da coins2.jpg***

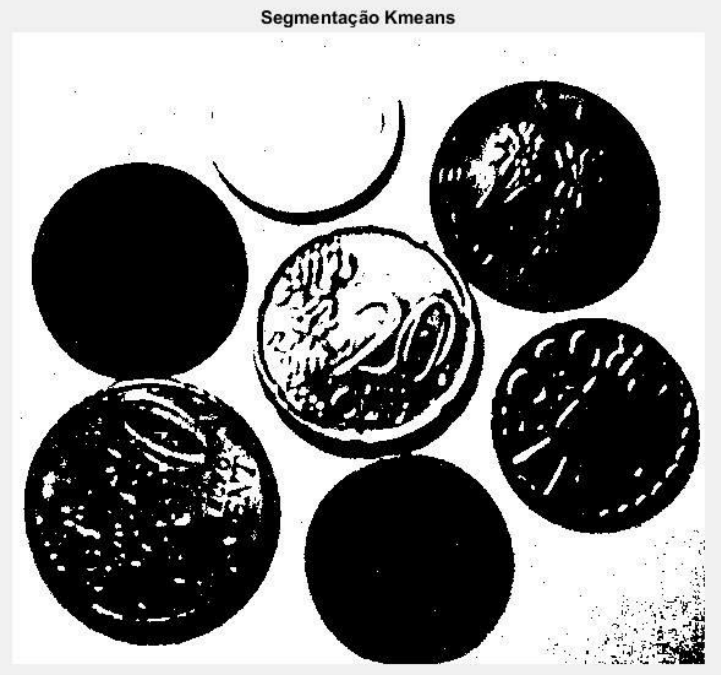

Figura 16: Segmentação Kmeans da coins3.jpg

6. Encontrar as moedas

Para encontrar as moedas decidimos aplicar a transformada de *Hough s*ob a abstração da função *imfindcircles* usando *'twostage'* como *'Method'*. Para aplicar esta transformada na imagem *coins.jpg*, decidimos usar o parâmetro ObjectPolarity = Bright, uma vez que o *background* é um pouco mais escuro que as moedas, Sensitivity = 0.98 e EdgeThreshold = 0.1. Em relação à imagem *coins.jpg2* também usamos esta transformada, mas desta vez com os

parâmetros ObjectPolarity = Dark, uma vez que o *background* é um pouco mais claro que as moedas, Sensitivity = 0.97 e EdgeThreshold = 0.08. Por fim, na imagem *coins3.jpg* repetimos novamente a transformada e após o processamento decidimos usar o parâmetro ObjectPolarity = Dark, que por sua vez o *background* é um pouco mais claro que as moedas, Sensitivity = 0.97 e EdgeThreshold = 0.08. Todas estas diferenças nos atributos da transformada surgem, uma vez que cada imagem tem uma iluminação, ângulo e resolução muito particular.

6.1. Resultados obtidos

Após todo este processo conseguimos localizar quase todas as moedas, à exceção de uma moeda na imagem *coins2.jpg*, sendo esta a imagem que nos causou mais dificuldade devido à posição relativa da camara que fazia com que se criassem sombras e a forma da moeda não fosse tão circular como devia. Além disso, as moedas usadas nessa imagem, eram moedas de duas camadas, ou seja, moedas de 1 e 2 euros em que muitas eram interpretadas como sendo uma moeda sobreposta sobre a outra. Como é óbvio, não conseguimos detetar todas as moedas em todos os casos, ou seja, para todos os tipos de ruídos como podemos ver os erros presentes na imagem seguinte.

Figura 17: Imagem com erro- deteção de sombras e círculos internos de moedas e ainda moedas não detetadas devido a sua forma elítica

Contudo, obtemos resultados muito dentro das expetativas, como podemos ver nas imagens a seguir.

Figura 18: coins.jpg

Figura 19: coins2.jpg

Figura 20: coins3.jpg

6.2. Contagem das moedas

A contagem das moedas é feita, uma vez que a função *imifindcircles* retornam um *array* com o raio das moedas encontradas e basta contar o numero de raios para saber quantas moedas foram encontradas. No melhor caso encontramos 4 moedas na *coins.jpg*, 23 moedas na *coins2.jpg* e por fim, 7 moedas na *coins3.jpg.*

6.3. Histograma de tamanhos

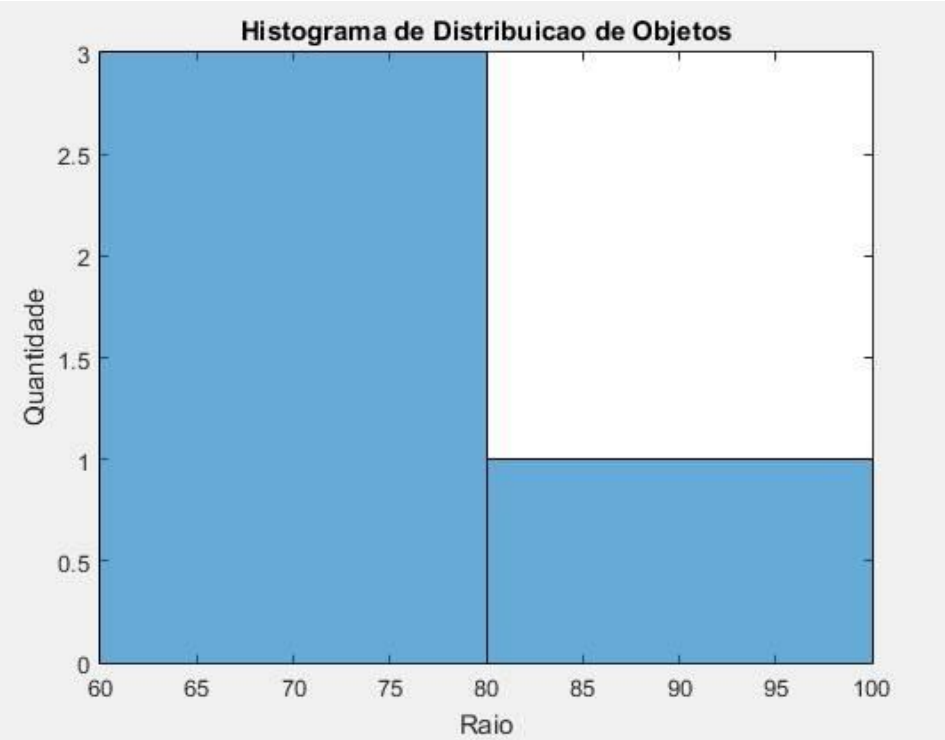

Figura 21: Histograma coins.jpg

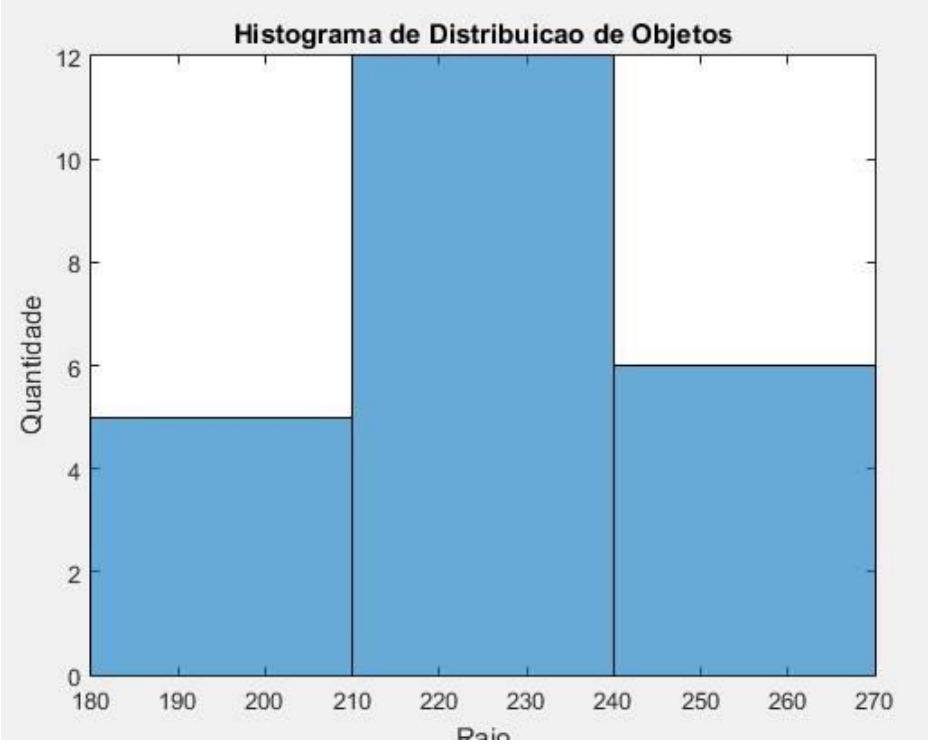

Figura 22: Histograma coins2.jpg

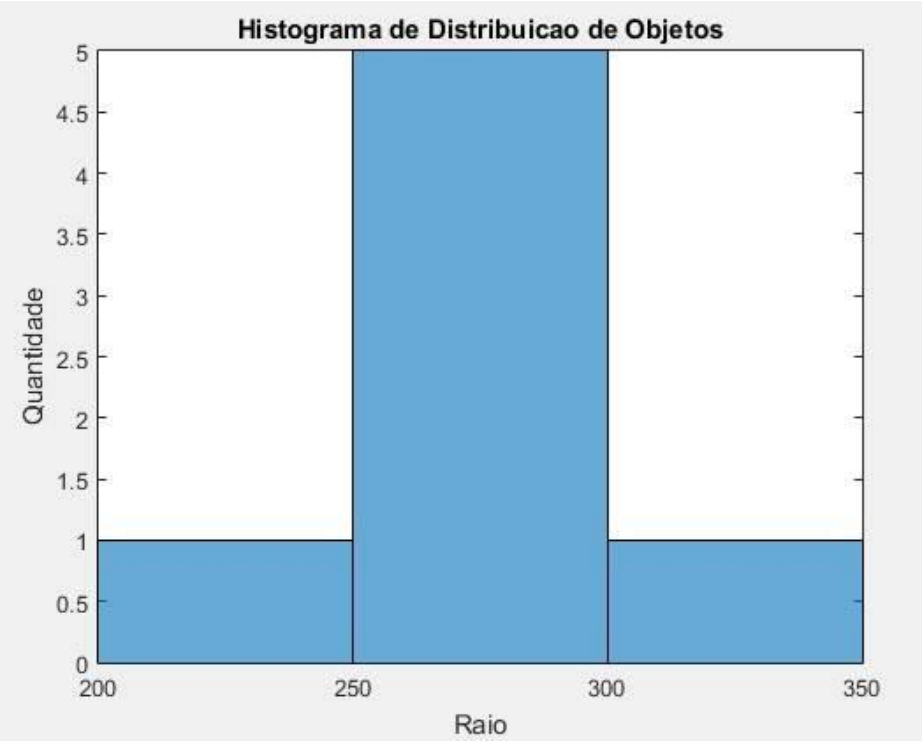

Figura 23: Histograma coins3.jpg