Análise Experimental da conversão RGB/YCbCr e codificação Run-length para compressão de imagens em Sistemas de Tempo Real Baseados em VANTs

Vinícius Eduardo S. Lara[1], **Celso Costa**[1], **Leticia Vieira Guimarães**[1]

[1]Universidade Estadual do Rio Grande do Sul (UERGS - Unidade Guaíba)
R. Santa Maria, 2300 – Bom Fim Velho, Guaíba – RS, Brazil

{vinicius-lara, celso-costa, leticia-guimaraes}@uergs.edu.br

Abstract. *Currently many applications make use of VANT's that capture RGB images and send them to a ground base in real time. The size of the transferred image is fundamental in meeting the temporal needs of the application. Run-length (RLE) is a data compression technique suitable for systems where there are long sequences of repetitions. An RGB/YCbCr translation was constructed and the RLE compactation algorithm was run in each one of the two colors spaces. The results of the experimental study indicate that run-length technique, applied to the YCbCr color system, presents advantages with respect to RGB, both in processing time and in relation to the compactation rates.*

Resumo. *Atualmente muitas aplicações que fazem uso de VANT's capturam imagens RGB e as enviam para uma base em terra, em tempo real. O tamanho da imagem transferida é fundamental no atendimento dos requisitos temporais da aplicação. Run-length (RLE) é uma técnica de compressão de dados adequada para longas sequências de repetições. Foi realizada a conversão RGB/YCbCr e executado o algoritmo de compactação RLE em cada um desses dois espaços de cores. Os resultados obtidos neste estudo experimental, indicam que a técnica run-length, aplicada ao sistema de cores YCbCr, apresenta vantagens em relação ao RGB, tanto em tempo de processamento quanto em relação as taxas de redução.*

1. Introdução

Uma imagem pode ser definida como uma matriz, de tamanho arbitrário, de pixels, onde cada pixel possui um canal que varia de acordo com o modelo de cor previamente determinado para cada imagem. Inúmeras aplicações são baseadas em imagens obtidas em tempo real por Veículos Áereos Não Tripulados (VANT's). Exemplos são sistemas de vigilância, sistemas de busca e salvamento, agricultura de precisão, etc. Muitos desses sistemas possuem requisitos temporais, nos quais é preciso executar uma ação em um tempo determinado, sob pena da ocorrência de uma falha, que pode ter consequências graves. Tipicamente, em sistemas baseados em VANT's, as imagens são obtidas por uma câmera e são enviadas para uma base em terra, para análise e tomada de decisão. Reduzir o tamanho da imagem é fundamental em Sistemas de Tempo Real, nos quais é preciso enviar imagens e existem limitações nos meios físicos de comunicação.

Para diminuir o tamanho da imagem e atender estes requisitos temporais foram analisadas as técnicas de compressão de imagem e vídeo JPEG [Agostini, Luciano V. et al.

2017] e H264 [Samson Ch., Sastry Vuk. 2017] . Estas técnicas utilizadas atualmente, apesar de atender plenamente, com folga, os requisitos de compressão, a sua implementação se torna inviável em processadores de pequena capacidade estruturados em um VANT. Assim, foram consideradas técnicas mais simples como a codificação Run-length (RLE) a conversão de RGB/YCbCr que poderiam realizar a compressão necessária no tempo requisitado, técnicas estas que estão embutidas em esquemas de codificação mais complexos como JPEG e H264.

O RGB [Rafael C. Gonzalez, Richard E. Woods, and Steven L. Eddins. 2003], é um sistema de cores em que o vermelho (red), o verde (green) e o azul (blue) são combinados de várias formas de modo a produzir um largo espectro cromático, sendo este, geralmente, o formato original das imagens produzidos por câmeras digitais. O YCbCr [K. Jack 2011] é outro espaço de cores representando respectivamente a luminância (Y), a crominância vermelha (Cr) e a crominância azul (Cb). A conversão do sistema RGB para o sistema YCrCb apresenta vantagens, pois o sistema visual humano é mais sensível à luminância (Y) [Sareesh 2013]. Isso possibilita reduzir a quantidade de bits utilizado para representar as cromas Cr e Cb sem perda aparente de qualidade para o olho humano.

Run-length [Samson Ch., Sastry Vuk. 2017] é uma técnica de codificação utilizada para comprimir cadeia de dados no formato digital nos quais existem sequências longas de dados repetidos. Com essa técnica, uma imagem tem seu tamanho original reduzido e, considerando imagens que devem ser enviadas de um VANT para um computador em terra, o tempo para o envio da imagem fica menor.

Este trabalho apresenta um estudo experimental comparativo da aplicação da técnica de compressão de imagens run-length (RLE) nos sistemas de cores RGB e YCbCr, com e sem perda na qualidade das imagens. A comparação levará em conta o tempo de execução do algoritmo de compactação em cada um dos dois sistemas de cores e as taxas de redução, com e sem perdas na qualidade da imagem.

2. Trabalhos Relacionados

[GUPTA, BANSAL 2015] realizaram trabalho com metodologia semelhante porém aplicaram a compactação apenas em imagens no sistema de cores RGB, sem análise de tempo do desempenho do algoritmo ou a possibilidade de perda na qualidade da imagem para um ganho maior na compressão.

[KAUR, SAXENA, SINGH 2017] fizeram comparação da codificação *run-length* com algoritmos diferentes, sem explicitamente tratarem a comparação com modelos de cores diferentes. Mostraram um bom desempenho do RLE em comparação aos demais algoritmos mas não apresentaram resultados em relação ao tempo ou possibilidade de perda na qualidade da imagem.

3. Metodologia

O programa de redução, o *Compressor de Imagem*, foi codificado em *C++* com o auxílio da biblioteca do *OpenCV* para *C++*, que tem como intuito o desenvolvimento de programas na área de visão computacional.

Inicialmente, a imagem é carregada em uma matriz, cujo número de linhas e colunas corresponde à altura e a largura da imagem. Cada posição dessa matriz representa um

pixel da imagem, e possui três canais de cores, de acordo com a cor do pixel, conforme se pode ver na eq. (1), que é a forma geral de uma matriz M, onde $a_{i,j}$ são os elementos da matriz.

Tradicionalmente, a biblioteca *OpenCV* retorna as imagens no sistema de cores RGB, sendo assim é necessário converter para YCbCr quando for necessário.

$$M = \begin{bmatrix} a_{1,1} & \dots & a_{1,j} \\ \vdots & \ddots & \vdots \\ a_{i,1} & \dots & a_{i,j} \end{bmatrix} \tag{1}$$

Antes de estabelecer o processo realizado pela execução do algoritmo é necessário estipular uma taxa de perda. A taxa de perda é uma referência; ela serve como margem para efetuar a análise do valor limite da diferença que o canal de cor do pixel pode ter em relação ao mesmo canal de seus pixels adjacentes.

Se a diferença, em módulo, entre o valor do canal do pixel em relação aos seus adjacentes for menor que a taxa de perda, então esses valores serão homogeneizados através da média aritmética dos mesmos. Isso aumenta a redundância de dados, melhorando então a aplicação do RLE.

A taxa de perda foi definida como o produto do percentual de perda da imagem, que é um valor entre 0 e 100% definido pelo sistema, com o valor máximo que o canal de cores pode expressar (que no caso do RGB é 255), conforme a eq. (2):

$$T_P = P_{\%} \cdot 255 \tag{2}$$

A Figura 1 apresenta o código executado em todas as linhas da matriz da imagem, para todos os canais, que permite aumentar a redundância de dados para aplicação do RLE. Após realizar esse processo é necessário executar o algoritmo run-length.

As Figuras 1 e 2 apresentam as transformações necessárias, aplicadas na primeira linha da matriz de um dos canais de cor da imagem. *IMG_COLS* é o número de colunas da imagem. *TAXA_PERDA* é a taxa de perda da imagem, conforme Fig. 2. *R* é a matriz do canal R da imagem. *CR* é a matriz comprimida do canal R da imagem. *novaColuna* é o índice da nova matriz comprimida do canal R, que possui tamanho arbitrário e reduzido em relação à matriz do canal R original. A representação *contador.R(1, coluna)* é uma forma de guardar o referencial, que é a quantidade de vezes que aquele valor foi repetido, junto com o próprio dado.

```
int inicio = 0, fim = 0, coluna = -1, acumulador = 0,
contador = 0;

while(coluna < IMG_COLS) {
    if(abs(R[1, coluna + 1] - R[1, coluna]) < TAXA_PERDA) {
        if(inicio == -1) {
           inicio = coluna;
        }
        acumulador += (R[1, coluna] + R[1, coluna + 1]);
        contador   += 2;
    } else {
        fim = coluna;
        for(int i = inicio; i < fim; i++) {
            CR[1, i] = acumulador/contador;
        }
        acumulador = 0;
        inicio     = -1;
        contador   = 0;
    }
    coluna += 1;
}
```

Figura 1. Algoritmo de redundância reduzido.

A aplicação do algoritmo, neste caso específico, consiste em analisar se o valor do canal dos pixels de uma linha são iguais, pois quanto maior a redundância melhor será a compressão. Sendo assim, é necessário realizar a comparação de igualdade, entre o valor do canal dos pixels, conforme descrito na Figura 2.

O referencial *contador*, adicionado junto ao dado repetido, foi definida como sendo do tipo *float* do *C++*. A escolha do tipo de dado *float* se deu pelo fato de que sua representação custa o mesmo que o dado do tipo *int*, 4 bytes. Sendo assim, a parte inteira do dado fica sendo *contador*, que é o número de vezes que o canal se repete, e a parte decimal fica sendo *R(1, coluna)*, que é o valor do canal de cor, conforme descrito na Figura 2.

```
int coluna = 0, novaColuna = 0, contador = 0;

while(coluna < IMG_COLS) {
    if(R[1, coluna] == R[1, coluna + 1]) {
        contador += 1;
    } else {
        CR[1, novaColuna] = (contador.R[1, coluna]);
        novaColuna += 1;
    }
    coluna += 1;
}
CR[1, coluna] = R[1, coluna];
```

Figura 2. Algoritmo run-length reduzido.

Executando o processo de redundância e codificação run-length se obtém um conjunto de matrizes, de todos os canais de cores, comprimidas prontas para serem envia-

das através de qualquer meio físico de comunicação. No receptor, as matrizes recebidas serão descompactadas e reconstruídas, afim de formar uma única matriz de imagem. A descompactação consiste em replicar a quantidade de valores repetidos, do canal, na mesma quantidade indicada na matriz.

3.1. Conversão entre sistemas de cores:

Para se fazer análise com imagens no sistema de cores YCbCr, é necessário converter os valores dos canais a partir dos valores RGB, gerando YCbCr e depois voltar do YCbCr para o RGB, sendo preciso utilizar as equações (3, 4) para obter tal resultado, conforme descrito na Conversão de Cores do OpenCV [OpenCV: Color conversions 2018].

Equações de conversão de RGB para YCbCr:

$$\begin{cases} Y = 0,299 \cdot R + 0,587 \cdot G + 0,114 \cdot B \\ Cb = (B - Y) \cdot 0,564 + 128 \\ Cr = (R - Y) \cdot 0,713 + 128 \end{cases} \tag{3}$$

Equações de conversão de YCbCr para RGB:

$$\begin{cases} R = Y + 1,403 \cdot (Cr - 128) \\ G = Y - 0,714 \cdot (Cr - 128) - 0,344 \cdot (Cb - 128) \\ B = Y + 1,773 \cdot (Cb - 128) \end{cases} \tag{4}$$

4. Descrição dos Algoritmos

A técnica run-length empregada se constituiu de três etapas:

4.1. Carregar:

1. Ler a Imagem;
2. Criar um vetor da imagem para armazenar a imagem
3. Converter os canais de cores, de RGB para YCbCr ou vice-versa, se necessário;
4. Armazenar cada canal de cor em sua respectiva coordenada no vetor da imagem.

4.2. Comprimir:

1. Criar um vetor para armazenar a imagem comprimida;
2. Definir um percentual de redução entre $[0, 100]$ e calcular a taxa de perda;
3. Ler um canal de cores do vetor da imagem partindo da primeira linha, de cima para baixo, da esquerda para direita;
4. Se os valores adjacentes do canal forem iguais ao valor atual ou possuírem variação de diferença inferior ao percentual de perda contabiliza $+1$ no contador;
5. Caso o contrário, guardar o valor do contador junto com o último valor lido no vetor comprimido da imagem;
6. Realizar estes passos para todos os canais de cores.

4.3. Descomprimir:

1. Criar um vetor descomprimido da imagem de tamanho igual à imagem antes da compressão;
2. Ler um canal de cores do vetor comprimido da imagem partindo da primeira linha, de cima para baixo, da esquerda para direita;
3. Repetir o valor, de acordo com o contador referencial que ele possui, no vetor descomprimido da imagem até reconstruir a imagem.

5. Resultados

Os testes foram realizados em duas imagens diferentes com base em dois sistemas de cores diferentes. As imagens, mostradas na Fig. 3 no sistema RGB e na Fig. 4 no sistema YCbCr , são de tamanhos iguais, todas com 640×480 pixels, ou seja, 921600 bytes. Foi utilizado um computador com processador Intel Core i5 de dois núcleos e 1,8 GHz (Turbo Boost de até 2,9 GHz) com cache L3 compartilhado de 3 MB, armazenamento SSD PCIe de 128 GB e chip gráfico Intel HD Graphics 6000 rodando com sistema operacional macOS. A Fig. 4 mostra as mesmas imagens convertidas para o sistema de cores YCbCr. Para cada execução, foi guardada a soma dos tempos gastos para: carregar, comprimir e descomprimir a imagem.

Para análise dos resultados, foi usada a *redução*, que é definida como sendo a quantidade de pixels que foram reduzidos em comparação ao tamanho inicial da imagem, definida pela a equação. (5):

$$R = 100 \cdot (1 - \frac{T_f}{T_i}) \tag{5}$$

onde R é a redução (em porcentagem), T_f é o tamanho final da imagem e T_i é seu tamanho inicial.

Foi realizado o experimento com três percentuais de perda diferentes, partindo do caso com perda nula ($P_{\%} = 0\%$), até o caso com perda de 6%, variando, de 2 em 2%.

Figura 3. foto1.png e foto2.png respectivamente, ambas com 640×480 pixels no sistema de cores RGB.

Analisando os resultados das Tabelas 1 e 2, é possível perceber que foi obtido sucesso na compressão e descompressão com perda de 0%, pois os resultados percentuais da redução foram maiores que zero, ou seja o tamanho da imagem foi reduzido no processo.

Os tempos totais gastos para compactar e descompactar cada imagem de acordo com o percentual de perda envolvido na compressão, na maioria dos casos, é menor no sistema de cores YCbCr, conforme pode ser visto nos gráficos da Fig. 5.

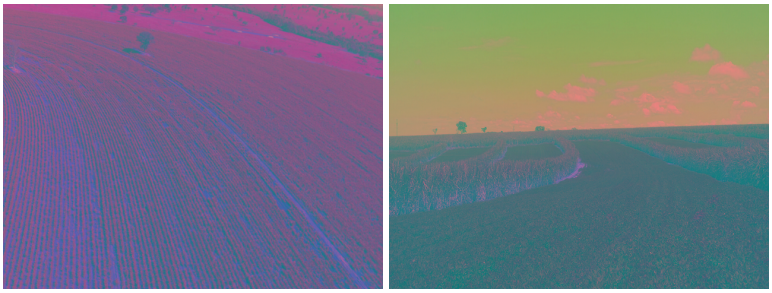

Figura 4. foto1.png e foto2.png respectivamente, ambas com 640×480 pixels no sistema de cores YCbCr.

Tabela 1. Análise no sistema de cores RGB.

Imagem	T_f (bytes)	Perda ($\%$)	Redução ($\%$)	Tempo total (ms)
foto1.png	877990	0	4,731	478
foto2.png	810680	0	12,035	481
foto1.png	544269	2	40,943	488
foto2.png	343139	2	62,767	539
foto1.png	363168	4	60,593	563
foto2.png	223982	4	75,696	491
foto1.png	249165	6	72,963	401
foto2.png	147668	6	83,977	402

Tabela 2. Análise no sistema de cores YCbCr.

Imagem	T_f (bytes)	Perda ($\%$)	Redução ($\%$)	Tempo total (ms)
foto1.png	816330	0	11,422	479
foto2.png	526904	0	42,827	431
foto1.png	319911	2	65,287	463
foto2.png	115859	2	87,428	428
foto1.png	75854	4	91,769	499
foto2.png	165194	4	82,075	389
foto1.png	103484	6	88,771	493
foto2.png	50412	6	94,529	386

Nas Figs. 6, 7, 8 e 9 são apresentados os resultados obtidos visualmente com essa compressão e descompressão. As figuras já estão convertidas novamente para o sistema RGB para melhor visualização. Em locais onde os pixels são muito heterogêneos, ou seja, onde há uma grande diferença entre seus valores, não ocorreram mudanças significativas na perda da qualidade da imagem. Porém, em áreas com pixels mais homogêneos pode-se perceber facilmente uma fusão de cores, o que resulta, em alguns, casos (com alta percentual de perda), na distorção da imagem.

É perceptível que, independente do percentual de perda ser 0% ou outro valor, o sistema de cores YCbCr se mostrou mais comprimível, e que, com o aumento do percentual de perda, ele se manteve acima do sistema RGB, em média 48% a mais que o sistema de cores RGB. Isto era esperado, já que o sistema YCbCr nivela mais os pixels, tornando-os mais repetitivos e adequados a codificação *run-length* [K. Jack. 2011].

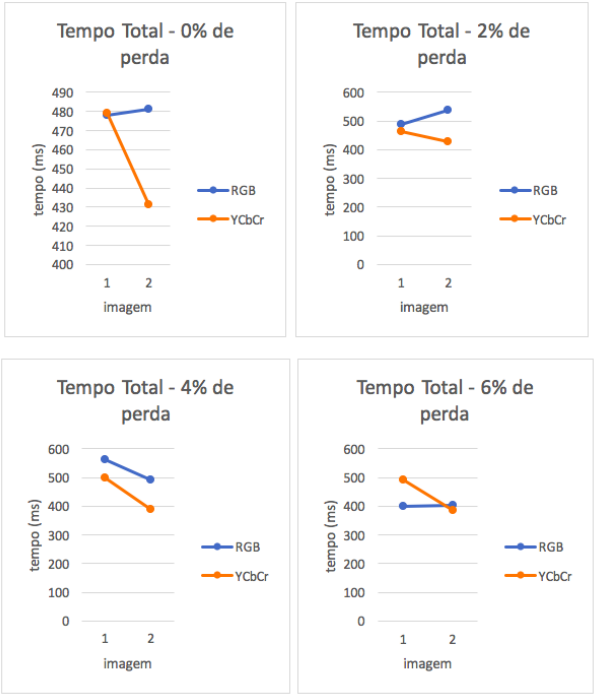

Figura 5. Gráficos que mostram o tempo total para cada foto.

Figura 6. foto1.png e foto2.png, descomprimidas, com percentual de perda de 2% no sistema de cores RGB.

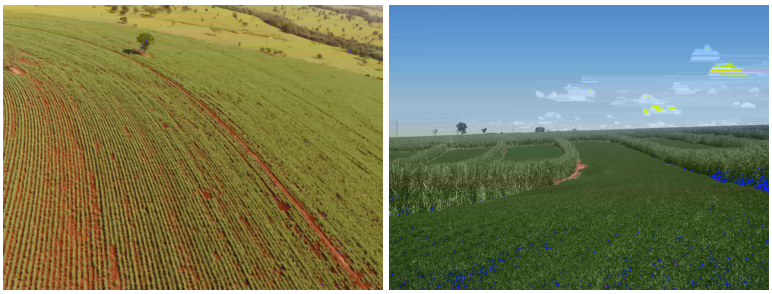

Figura 7. foto1.png e foto2.png, descomprimidas, com percentual de perda de 2% no sistema de cores YCbCr.

Para analisar a redução, foram feitos gráficos para mostrar a diferença entre ambos sistemas de cores, conforme pode ser visto nos gráficos da Fig. 10.

Figura 8. foto1.png e foto2.png, descomprimidas, com percentual de perda de 4% no sistema de cores RGB.

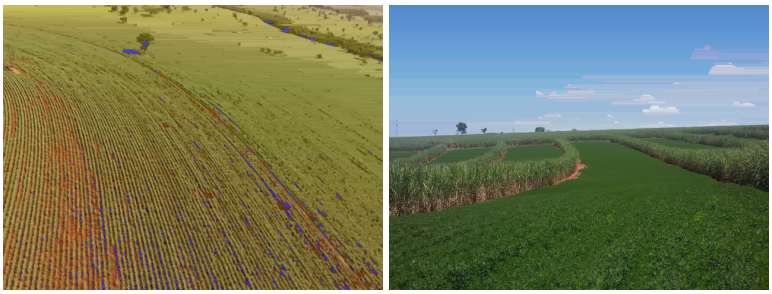

Figura 9. foto1.png e foto2.png, descomprimidas, com percentual de perda de 4% no sistema de cores YCbCr.

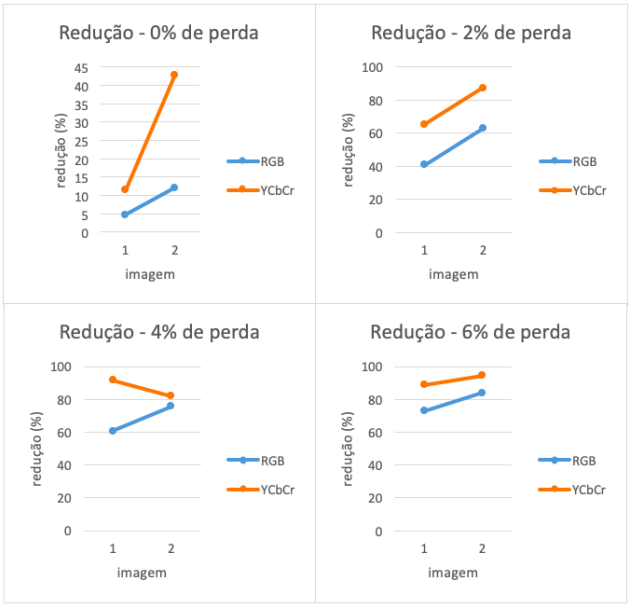

Figura 10. Gráficos que mostram o percentual de redução para cada foto.

6. Conclusão

De acordo com os dados obtidos através da análise efetuada, é possível perceber que a técnica de compactação *run-length* tem alta eficiência [G. Pratishtha, B. Varsha Bansal. 2015], principalmente em áreas com grande redundância de pixels. É perceptível que a compactação obteve maior êxito no sistema de cores YCbCr, tanto em tempo de processamento quanto em nível de compressão. Isso se deve à alta capacidade de homogeneização

de pixels que este modelo possui. Conforme o percentual de perda aumenta, é fácil perceber que, no sistema de cores YCbCr, começaram a aparecer pixels azulados em áreas onde ocorreu grande homogeneização de cores. Uma possível causa para este fenômeno pode ser a técnica empregada para definir a homogeneização de cores, que é a média aritmética. Em trabalhos futuros, para tentar corrigir essa situação, poderiam ser realizadas análises e testes em imagens com alta homogeneidade de pixels para definir uma técnica melhor de homogeneização de cores, que se adeque melhor e afete menos a imagem no sistema de cores YCbCr, e que, de preferência, possa ser usada também no RGB (que não é afetado por esse tipo de fenômeno). A compressão é uma parte muito importante do processamento de imagem. Essa técnica pode trazer benefícios enormes para aplicações que lidam com grande volume de imagens e que precisam dela em um intervalo de tempo crítico.

Referências

Agostini, Luciano V. et al. (2017) *Design and FPGA prototyping of a H: 264/AVC main profile decoder for HDTV*. J. Braz. Comp. Soc. Volume 12, n. 4, pp. 25-36. ISSN 0104-6500.

Pratishtha, G., Varsha, Bansal B. (2015) *The Run Length Encoding for RGB Images*. International Journal of Converging Technologies and Management (IJCTM). Volume 1, Issue 1.

Gupta, G., Gupta, K., L. Jyoti A. (2014) *AN ADVANCED COMPRESSION APPROACH WITH RLE FOR IMAGE COMPRESSION*. International Journal of Advanced Research in Computer Science and Software Engineering. Volume 4, Issue 2.

Jack, K. (2011) *Video demystified: a handbook for the digital engineer*. Elsevier.

Kamalpreet, K., Jyoti, S., Sukhjinder, S. (2017) *Image Compression Using RLE*. International Journal on Recent and Innovation Trends in Computing and Communication. Volume 5, Issue 5.

OpenCV: Color conversions. Disponível em: `https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html`. Acesso em: Julho de 2018.

Gonzalez, C. Rafael., Woods, R. E., Eddins, Steven L. (2003) *Digital Image Processing Using MATLAB*. ISBN-10: 0130085197.ISBN-13: 978-0130085191. Prentice Hall, 1st edition.

Ch, S., Vuk, S. (2017) *An Improved Run Length Encoding Scheme for Image Compression*. IJECS. ISSN: 2319-7242. Volume 6 Issue 3.

Understanding Luminance and Chrominance. Disponível em: `https://wolfcrow.com/understanding-luminance-and-chrominance/`. Acesso em: Janeiro de 2019.