Curso de Imagens Numéricas

P. Kantek

4 de agosto de 2021

Sumário

Capítulo 1

Do que se trata

Da representação pictórica através de uma imagem de alguma realidade adjacente. A importância do tema é evidente: o ser humano é um ser visual. Estima-se que mais da metade do tecido cerebral humano seja dedicado à captura e análise de imagens. Para o registro de imagens, até o século XIX, dependia-se para tanto de um talento humano. Aqui representado por um magnífico exemplo

Trata-se da obra de Leonardo da Vinci, intitulada Mona Lisa (Senhora Lisa) ou La Gioconda, atualmente em exibição no Museu do Louvre em París. O Quadro começou a ser pintado em 1503. Foi levado pelo autor para París e lá comprado pelo rei francês, pelo equivalente a mais de 15Kg de ouro. Por essa razão o quadro está hoje em posse da França. Foi roubado no século XX e temeu-se que nunca mais reaparecesse, mas a polícia francesa foi atrás do ladrão (um funcionário do Louvre) e recuperou o quadro. Ainda sobre o quadro ele é considerado uma das principais aplicações da técnica do *esfumetto*, pela qual parece haver uma névoa entre o espectador e o objeto, e da qual da Vinci foi o maior executor. Conta-se que Leonardo levou mais de 4 anos pintando La Gioconda.

Já por volta de 1820, a tecnologia começou a vir em socorro de quem precisava registrar imagens. A fotografia a seguir é considerada a primeira, obtida por Joseph Niepce em 1826. Esta imagem precisou de uma exposição de 8 horas baixo luz solar.

Embora o pioneiro seja reconhecido como tal, o processo fotográfico (do grego fós=”luz”, e grafis=”estilo”ou ”pincel”) é tradicionalmente encarado como um processo coletivo, no qual diversas invenções e contribuições vão se somando até obter um resultado final.

A idéia é que uma imagem seja capturada por uma câmera e projetada sobre um superfície fotossensível que devidamente tratada por produtos químicos pode ser revelada (destacar os tons da imagem) e fixada (eternizada sem precisar se ambientes artificiais para ser vista ou armazenada). O resultado é uma fotografia da imagem original.

A fotografia se transforma em um consumismo a partir de 1880, quando a empresa Kodak lança o discurso de marketing de que qualquer um podia tirar fotografias, sem ser necessáriamente um profissional, com o recurso da câmera tipo box. Logo complementada pelo filme em rolo, idéia de George Eastmann.

Até meados do século XX, a fotografia era sinônimo de preto-e-branco. A introdução do filme Kodakchrome em 1936 começou a mudar este fato: Agora fotografias podem ser coloridas graças a 3 emulsões distintas, uma para cada cor básica.

Uma característica que terá a sua importância a seguir é que a fotografia até os anos 1990, era algo relativamente custoso: desde as câmeras até os itens consumíveis (filmes e papéis fotográficos). Este fato naturalmente limitou a fotografia até o próximo evento importante.

A introdução da chamada fotografia digital [1] vem a seguir e será o assunto daqui para a frente.

[1] Os franceses, na sua tradicional racionalidade se recusam a tratar a fotografia digital (que afinal, digital vem de *dedos*) e preferem a nomenclatura de fotografia numérica, questionando até o uso da palavra fotografia neste tema. Concordo em parte com os franceses, razão pela qual este curso é de imagens numéricas e não fotografia digital.

Capítulo 2

Imagens Numéricas

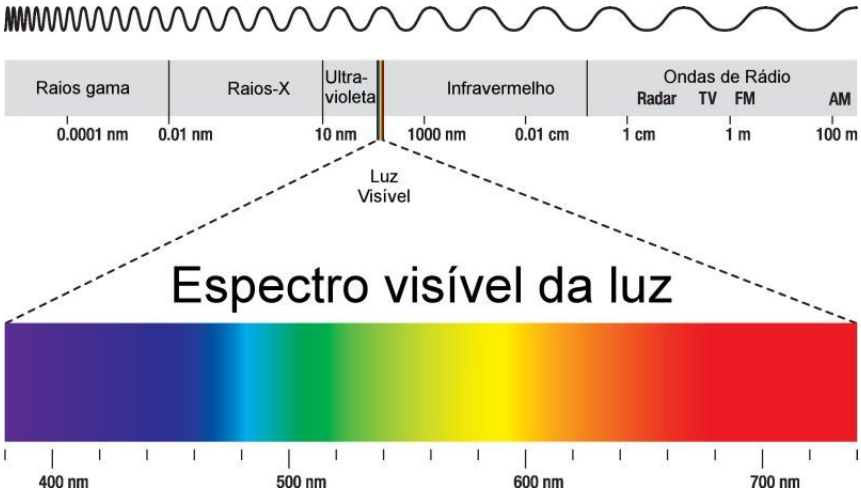
Uma das primeiras aplicações de imagens numéricas ocorre na indústria de jornais, pela precisão do registro fotográfico à distância. A primeira iniciativa é entre Londres e Nova Yorque quando por volta de 1920 o cabo telegráfico Bartlane foi usado para transmitir uma imagem fotográfica (demorando cerca de 3 horas ao contrário da uma semana necessária para levar a fotografia: e a um custo muito menor).

O primeiro sistema Bartlane usava 5 níveis de cinza e em 1929 a capacidade aumentou para 15 níveis de cinza. Embora claramente estas sejam imagens numéricas, elas ainda não são o foco deste trabalho por não usarem computadores em seu manuseio.

O próximo passo é dado pela corrida espacial. Já havia computadores por trás e a impossibilidade agora era tranportar o registro físico da imagem de fora do planeta Terra. Em 31 de julho de 1964, a espaçonave americana Ranger transmitiu a imagem abaixo, 17 minutos antes de colidir com a Lua.

Alguns anos depois foi a vez de Marte nos brindar com fotos da superfície, devidamente transmitidas numericamente.

Um detalhe importante é que os olhos humanos estão limitados por questões físicas a uma parte bastante estreita do espectro eletromagnético. Aquele que vai de cerca de 350 nm até cerca de 750 nm de comprimento de onda, como se pode ver em



A grande aquisição neste ponto é que modificando os parâmetros da imagem, a mesma pode ser deslocada de qualquer extremo em direção à faixa visível possibilitando "ver" imagens originalmente invisíveis. Um exemplo do que se fala está por trás do conceito de tomografia computadorizada. Uma tomografia é uma captura de sucessívos Raios-X de uma parte do corpo e sua posterior reconstituição em uma imagem visível. Inventada de maneira independente por dois pesquisadores, Goodfrey Houndsfield e Allan McCormack, ambos dividiram o Prêmio Nobel de Medicina de 1979 pela descoberta.

2.1 Processamento e análise de imagens

Neste ponto nosso estudo pode ser dividir em dois grandes grupos

processamento de imagens trata das técnicas, algoritmos e procedimentos realizados sobre uma imagem numérica com vistas à uma eventual decisão por olhos humanos.

análise de imagens trata das técnicas, algoritmos e procedimentos realizados sobre uma imagem numérica com vistas a uma decisão por parte de um software (e hardware associado).

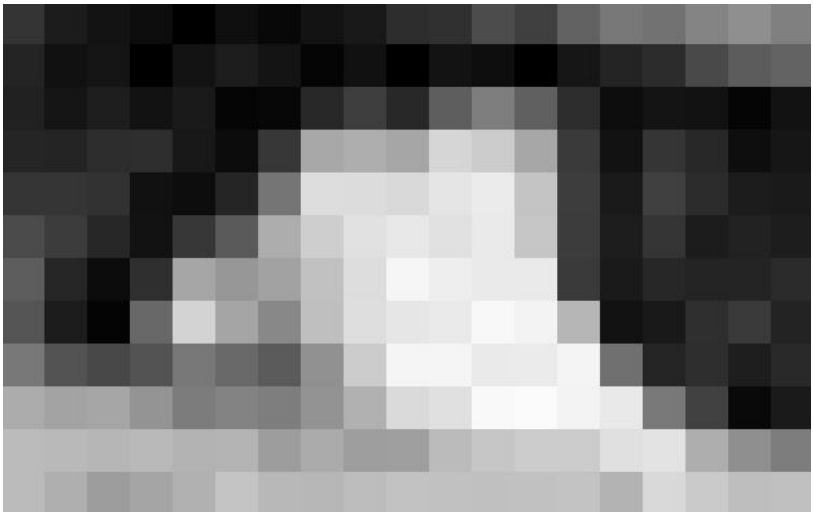
Por razões óbvias, as duas técnicas acima são em geral bastante distintas. Exemplos do primeiro caso incluem astronomia, biologia, medicina, segurança pública, defesa e indústria. Do segundo caso tem-se o reconhecimento de caracteres, visão industrial para inspeção, processamento de impressões digitais, imagens aéreas para meteorologia e avaliação ambiental.

Capítulo 3

O que é uma imagem numérica

É uma matriz bidimensional numérica, composta de células (que na imagem correspondem a 1 pixel) e na qual o valor informa a quantidade de tinta que deve ser colocada naquele pixel, no momento de reconstituir a imagem. Veja-se uma imagem monocromática que contempla um personagem em

Agora faça-se um pequeno recorte de 12 linhas por 19 colunas de uma parte do olho esquerdo da pessoa. O bloco vai devidamente ampliado a fim de poder ser entendido



E, finalmente, veja-se a matriz numérica que teve como origem esta imagem

59	21	19	17	0	20	7	22	25	42	43	84	57	97	124	109	125	146	132
30	18	27	0	20	34	16	7	17	0	21	20	9	14	42	46	79	92	94
32	22	31	11	34	7	0	45	62	44	89	109	90	49	10	17	9	3	18
39	37	41	43	24	8	51	170	167	162	224	211	177	49	22	51	44	18	25
50	55	53	26	7	39	127	223	223	227	227	239	187	60	23	65	43	22	33
82	48	41	14	60	82	161	202	223	231	215	228	206	73	24	49	37	31	37
95	27	16	58	166	144	170	204	228	232	255	209	235	56	24	40	30	41	37
86	31	6	97	210	168	137	182	220	234	222	255	254	170	20	25	45	59	36
123	88	70	92	102	107	102	142	205	249	244	236	235	241	117	30	46	21	43
166	163	159	158	136	127	119	155	182	212	219	248	247	249	224	145	68	23	21
187	190	185	181	170	175	157	168	156	168	181	205	209	204	224	203	178	135	130
185	173	157	162	179	205	183	179	186	192	200	186	198	198	177	231	205	182	193

Como esta imagem tem 256 níveis de cinza, o número zero indica nada de tinta branca, e portanto o pixel correspondente vai ser pintado de preto. Já o número 255 indica a quantidade total de tinta branca no pixel que será completamente branco.

É razoável supor que embora uma imagem possa ser entendida como uma matriz numérica e vice-versa, há que se ter algum registro sobre quem é o que. Simplesmente pela razão de que uma matriz numérica pode ser muitas coisas (além de uma imagem).

Vai daí, que nos primórdios do registro de imagens numéricas cada plataforma de software/hardware fazia suas próprias marcações, o que transformava cada imagem assim registrada em um formato proprietário, isto é, só era conhecido e manuseado pelos seus criadores.

Este fato é fatal para o intercâmbio de informações. É só pensar na Internet de hoje com suas bilhões de imagens e pensar se cada uma delas tivesse que se reportar a alguma plataforma proprietária. A solução para este problema está na criação de padrões: isto é formatos e registros conhecidos e publicados, não proprietários e que garantem que uma vez seguidos não impedirão de compartilhar e ver imagens produzidas por outrém.

Esta história nos remete aos anos 80, quando a IBM desenvolveu um sistema operacional chamado OS/2, que acabou em fiasco: chegou a ser muito pouco usado. Pensado para substituir o MS-DOS, acabou sendo atropelado pelo lançamento do Windows filho único da Microsoft.

Entre outras iniciativas o OS/2 acabou lançando o formato de imagens BMP (abreviação de Bit Mapped). Por ser o primeiro este padrão é aceito até hoje por qualquer software gráfico. Tem problemas (i. Não admite compressão, ii. Não admite animações, iii. Tem a mania da IBM de reservar espaços e usar informações obscuras, entre outros), mas o pioneiro sempre tem o seu valor.

3.1 Preto e Branco versus colorido

É hora de descrever como uma imagem branco e preta é manuseada. Já se teve um vislumbre aí acima, no exemplo do olho. Determinado um nível de cores (ou seja a quantidade de cinzas intermediários além do preto e do branco) define-se cada número na matriz como a quantidade de cor branca naquele pixel. Os formatos mais comum aqui são 2=só preto e branco, chamadas imagens binárias; 16=preto e branco além de 14 cinzas intermediários e no limite 256=preto, branco e 254 níveis intemediários de cinza.

Este uso é simples: na matriz numérica, cada valor corresponde a um pixel e na imagem cada pixel corresponde a um valor numérico. A matriz é sempre retangular, assim como as imagens representadas neste formato.

Já as imagens coloridas começam a se tornar um pouco mais complexas. Primeiro, da física, sabe-se que misturando as 3 cores básicas pode-se obter virtalmente qualquer cor. Este fato é representado pela imagem

Cada pixel agora é representado por 3 números: Red (=vermelho), Green (=verde) e Blue (=azul) através da representação conhecida como RGB. Se estes níveis forem 256 para cada cor e olhando para a imagem acima pode-se ter as seguintes cores:

vermelho	verde	azul	cor resultante
255	0	0	vermelho puro
0	255	0	verde puro
0	0	255	azul puro
255	255	0	amarelo
255	0	255	magenta (cor de rosa)
0	255	255	ciano (azul claro)
255	255	255	branco

A representação colorida mais simples é aquela que para cada pixel apresenta os 3 valores de RGB. Neste caso, cada uma dessas componentes dá origem a uma matriz numérica de mesmo formato. Agora, para uma imagem colorida tem-se 3 matrizes numéricas: uma para cada cor. Este tipo de imagem é conhecida como TRUE COLOR.

3.2 BMP - mapeado

O formato true color acima descrito é conhecido por um possível desperdício de área de memória. Afinal ele permite representar $256 \times 256 \times 256 = 16777216$ cores distintas. Pense numa caixa de lápis de cor com este número de lápis distintos. Nenhum olho humano consegue distinguir tantas cores.

A solução está no que se chamou de imagem mapeada, implementada diretamente pelo formato BMP que estamos descrevendo. A idéia agora é construir uma tabela de cores (tipicamente com até 256 entradas). Cada entrada terá uma certa quantidade de vermelho, verde e azul, bem como será identificada pela sua posição na tabela.

Imagine por exemplo o começo da tabela como sendo

deslocamento na tabela	vermelho	verde	azul	cor resultante
0	0	0	0	preto
1	80	80	80	cinza escuro
2	200	200	200	cinza claro
3	255	255	255	branco
4	200	0	0	vermelho escurecido
5	0	180	0	verde escuro
...	...	...	...	...

A partir desta tabela em particular (que acompanha a imagem) pode-se representar qualquer uma das 256 cores tabeladas simplesmente representando o deslocamento da cor dentro da tabela como sendo o valor específico dentro da matriz numérica única (agora é só uma matriz e não 3 como no formato TRUE COLOR acima descrito). Ou seja com este mecanismo volta-se a ter uma unica matriz numérica e não 3. Como não há almoço grátis, deixa-se de poder representar as 16 milhões de cores e passa-se a poder representar apenas 256 cores diferentes.

Capítulo 4

Formatos

4.1 O formato BMP

A seguir, informações completas (e um tanto indigestas) sobre o formato BMP.

Grosso modo, uma imagem BMP é um arquivo que contém as seguintes informações:

- Bloco de controle
- Tabela cores (se a imagem for mapeada)
- Imagem

O bloco de controle tem a seguinte configuração

Tipo	Tam	Desl	Nome	Descrição	exemplo
char	2	0	type	constante igual a BM	424D
long int	4	2	size	tamanho do arquivo	C6 04 00 00
int	2	6	reserved	zeros	00 00
int	2	8	reserved	zeros	00 00
long int	4	A	off	desloc até a imagem	36 04 00 00
long int	4	E	size	resto do bloco controle	28 00 00 00
long int	4	12	width	n. de colunas	0A 00 00 00
long int	4	16	height	n. de linhas	0C 00 00 00
int	2	1A	planes	planos	01 00
int	2	1C	bitcount	bits / pixel	08 00
long int	4	1E	compress	compressao	00 00 00 00
long int	4	22	sizeimage	tamanho da imagem	90 00 00 00
long int	4	26	pix/m H	pixels / m na horizontal	CE 0E 00 00
long int	4	2A	pix/m V	na vertical	D8 0E 00 00
xxx	8	2E	xxx	xxx	

Note que

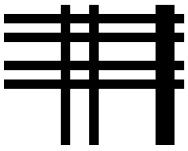
- A tabela de cores tem tamanho variável (1024 bytes no máximo), começa no endereço 36 e cada entrada contém as quantidades de azul/verde e vermelho. (note a inversão)
- O quarto byte de cada entrada é sempre zeros binários.
- A imagem começa em 536 (se a tabela = 1024) e cada linha está alinhada em múltiplo de 32 bits, por questões de eficiência.
- Na tabela, cada número indica a quantidade de tinta, assim: FF = tudo e 00 = nada.
- Como em todas as entradas da tabela de cores, tem-se R = G = B, pode-se concluir que esta imagem é monocromática, ou como se diz "branco e preto".
- a ordem das cores na tabela de cores é BGR, ou o contrário do nome do modelo que é RGB.

Seja como exemplo uma imagem devidamente *dumpeada*: É uma imagem branca, contendo 4 linhas V e 4 linhas H de cor cinza.

O ponto chave para distinguir as imagens BMP mapeadas ou true color está na posição `1C` do bloco de controle, que informa a quantidade de bits por pixel. Os principais valores são `08` indicando 8 bits por pixel, ou seja a imagem é mapeada. Já o conteúdo `18` (em hexadecimal 1×16 que corresponde a $16 + 8 = 24$ ou $8 \times 3 = 24$) indica uma imagem true color, com 3 bytes por pixel.

```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
      -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
0000  42 4D 62 05 00 00 00 00 00 00 36 04 00 00 28 00 BM........6...(.
0010  00 00 13 00 00 00 0F 00 00 00 01 00 08 00 00 00 ................
0020  00 00 60 01 00 00 00 00 00 00 00 00 00 00 00 01 ................
0030  00 00 00 00 00 00 00 00 00 00 01 01 01 00 02 02 ................
0040  02 00 03 03 03 00 04 04 04 00 05 05 05 00 06 06 ................
0050  06 00 07 07 07 00 08 08 08 00 09 09 09 00 0A 0A ................
0060  0A 00 0B 0B 0B 00 0C 0C 0C 00 0D 0D 0D 00 0E 0E ................
0070  0E 00 0F 0F 0F 00 10 10 10 00 11 11 11 00 12 12 ................
0080  12 00 13 13 13 00 14 14 14 00 15 15 15 00 16 16 ................
...
0410  F6 00 F7 F7 F7 00 F8 F8 F8 00 F9 F9 F9 00 FA FA ................
0420  FA 00 FB FB FB 00 FC FC FC 00 FD FD FD 00 FE FE ................
0430  FE 00 FF FF FF 00 FF FF FF FF FF FF 7F FF FF 7F ................
0440  FF FF FF FF FF FF 7F 7F FF 00 FF FF FF FF FF FF ................
0450  7F FF FF 7F FF FF FF FF FF FF 7F 7F FF 00 FF FF ................
0460  FF FF FF FF 7F FF FF 7F FF FF FF FF FF FF 7F 7F ................
0470  FF 00 FF FF FF FF FF FF 7F FF FF 7F FF FF FF FF ................
0480  FF FF 7F 7F FF 00 FF FF FF FF FF FF 7F FF FF 7F ................
0490  FF FF FF FF FF FF 7F 7F FF 00 FF FF FF FF FF FF ................
04A0  7F FF FF 7F FF FF FF FF FF FF 7F 7F FF 00 7F 7F ................
04B0  7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................
04C0  7F 00 FF FF FF FF FF FF 7F FF FF 7F FF FF FF FF ................
04D0  FF FF 7F 7F FF 00 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................
04E0  7F 7F 7F 7F 7F 7F 7F 7F 7F 00 FF FF FF FF FF FF ................
04F0  7F FF FF 7F FF FF FF FF FF FF FF 7F 7F FF 00 FF ................
0500  FF FF FF FF 7F FF FF 7F FF FF FF FF FF FF 7F 7F ................
0510  FF 00 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F ................
0520  7F 7F 7F 7F 7F 7F 00 FF FF FF FF FF FF 7F FF FF ................
0530  FF FF FF FF FF FF 7F 7F FF 00 7F 7F 7F 7F 7F 7F ................
0540  7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 7F 00 FF FF ................
0550  FF FF FF FF 7F FF FF 7F FF FF FF FF FF FF 7F 7F ................
0560  FF 00                 ..
```

A imagem tem 15 linhas e 19 colunas. Os riscos estão nas linhas 7, 9, 12 e 14 e nas colunas 7, 10, 17 e 18.



Exemplo:

```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
      -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
0000  42 4D 62 05 00 00 00 00 00 00 36 04 00 00 28 00 BM........6...(
0010  00 00 12 00 00 00 0F 00 00 00 01 00 08 00 00 00 ................
0020  00 00 60 01 00 00 00 00 00 00 00 00 00 00 00 01 ................
0030  00 00 00 00 00 00 00 FF FF 00 80 80 80 00 FF FF ................
0040  FF 00 FF 00 FF 00 00 00 FF 00 00 00 00 00 00 FF ................
0050  00 00 FF FF 00 00 FF 00 00 00 09 09 09 00 0A 0A ................
0060  0A 00 0B 0B 0B 00 0C 0C 0C 00 0D 0D 0D 00 0E 0E ................
0070  0E 00 0F 0F 0F 00 10 10 10 00 11 11 11 00 12 12 ................
...
0410  F6 00 F7 F7 F7 00 F8 F8 F8 00 F9 F9 F9 00 FA FA ................
0420  FA 00 FB FB FB 00 FC FC FC 00 FD FD FD 00 FE FE ................
0430  FE 00 FF FF FF 00 01 01 03 01 01 01 08 01 01 01 ................
0440  01 01 02 01 01 06 01 01 00 00 04 04 03 04 04 04 ................
0450  04 04 04 04 04 04 04 04 04 04 04 04 00 00 01 01 ................
0460  03 01 01 01 08 01 01 01 01 01 02 01 01 06 01 01 ................
0470  00 00 01 01 03 01 01 01 08 01 01 01 01 01 02 01 ................
0480  01 06 01 01 00 00 01 01 03 01 01 01 08 01 01 01 ................
0490  01 01 02 01 01 06 01 01 00 00 07 07 03 07 07 07 ................
04A0  08 07 07 07 07 07 07 07 07 06 07 07 00 00 01 01 ................
04B0  03 01 01 01 08 01 01 01 01 01 02 01 01 06 01 01 ................
04C0  00 00 01 01 03 01 01 01 08 01 01 01 01 01 02 01 ................
04D0  01 06 01 01 00 00 01 01 03 01 01 01 08 01 01 01 ................
04E0  01 01 02 01 01 06 01 01 00 00 01 01 03 01 01 01 ................
04F0  08 01 01 01 01 01 02 01 01 06 01 01 00 00 00 00 ................
0500  03 00 00 00 08 00 00 00 00 00 02 00 00 06 00 00 ................
0510  00 00 01 01 03 01 01 01 08 01 01 01 01 01 02 01 ................
0520  01 06 01 01 00 00 05 05 03 05 05 05 05 05 05 05 ................
0530  05 05 05 05 05 06 05 05 00 00 01 01 03 01 01 01 ................
0540  08 01 01 01 01 01 02 01 01 06 01 01 00 00 01 01 ................
0550  03 01 01 01 08 01 01 01 01 01 02 01 01 06 01 01 ................
0560  00 00                 ..
0570
```

Acompanhe no exemplo: Quantidade de colunas _____ e Quantidade de linhas _____.

Relembrando: as cores estão na forma de misturas de AZUL / VERDE / VERMELHO. azul + vermelho = magenta; azul + verde = ciano; verde + vermelho = amarelo. A tabela de cores comeca em X'36'. Preencha a tabela de cores:

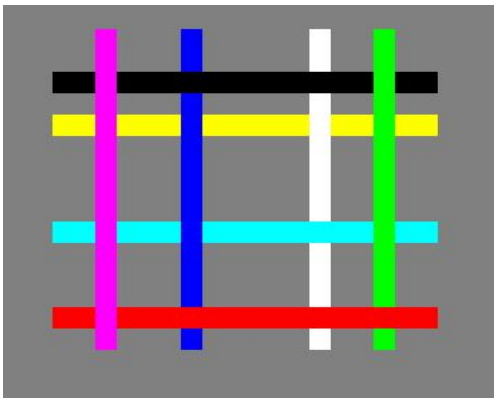
Cor 0		Cor 1		Cor 2	
Cor 3		Cor 4		Cor 5	
Cor 6		Cor 7		Cor 8	

A linha do pé da imagem começa em X'436'. Localize os

bytes na memória. Pela análise desta linha é possível localizar a:
cor de fundo: _____.

Repare que cada linha tem apenas 18 bytes, mas que na memória elas ocupam 20 bytes (múltiplo de 32 bits ou 4 bytes)

Respostas deste exemplo: A imagem tem 18 colunas por 15 linhas. O fundo da mesma é da cor cinza. As colunas estão nas posições 3 (magenta), 7 (azul), 13 (branca) e 16 (verde. As linhas são: 2 (vermelho), 6 (ciano), 11 (amarelo) e 13 (preto. A primeira linha a ser desenhada foi a linha 11 e a última coluna foi a 3. Eis o arquivo/imagem acima:



4.2 Formato JPG

O formato JPG vem de um trabalho de comite técnico. A sigla vem do nome "Joint Photographic Experts Group", que é o grupo responsável pelo padrão JPEG. Acabou se firmando como o padrão de máquinas fotográficas e também de intercâmbio de imagens fotográficas na Internet.

Ao contrário da compressão de texto ou de programas, quando não é aceitável a perda de qualquer informação, no caso de imagens (e sobretudo no caso de vídeo) é perfeitamente aceitável algum nível de degradação da imagem. Isso ocorre devido a dois fatores:

- Nossos olhos (e ouvidos) não são capazes de distinguir mudanças sutis e principalmente
- Imagens (e vídeos) são objetos muito grandes que praticamente exigem serem comprimidos para um manuseio adequado.

As etapas pelas quais uma imagem passa antes de ser guardada em memória como um objeto JPG são as seguintes:

1. A imagem é dividida em blocos de 8×8 pixels. A razão é diminuir a quantidade de operações aritméticas necessárias durante a compressão (ainda que pagando para isso ter uma compressão ligeiramente ineficiente).
2. Cada bloco de imagem sofre a ação da transformada discreta do cosseno. Este procedimento (reversível) gera outra matriz 8×8, mas com uma propriedade interessante. Apenas os valores mais à esquerda e acima é que têm valores significativos. Embaixo e à direita, o valor mais comum é zero.
3. Os dados são quantizados visando diminuir o comprimento dos valores e aumentar a redundância. Aqui ocorre a degradação na compressão, já que este processo é irreversível.
4. Na compressão, os dados quantizados são percorridos em zig-zag. A seguir processos de compressão, tais como run-length, Huffman, QM (esta última protegida por patente) são aplicados gerando o arquivo final.

É de se notar que a maioria das câmeras digitais de fotografia (ou como dizem os franceses: *photo numérique*) já fazem todo este processamento, a fim de economizar memória.

Ao longo do processo, inúmeras otimizações são feitas para torná-lo mais eficiente. A primeira já citada é trabalhar com blocos de 64 valores ao invés de trabalhar com a imagem inteira. Uma segunda otimização ocorre quando os cossenos (uma função transcedental) são previamente calculados e armazenados, sendo depois apenas consultados. Uma terceira otimização ocorre quando o processo é manuseado algebricamente a fim de que as iterações sejam convertidas em um produto matricial de matrizes. Finalmente, usa-se aritmética de ponto fixo (fazendo os ajustes necessários) já que esta sempre é muito mais eficiente que a aritmética de ponto flutuante.

Transformada discreta do cosseno

Baseada no conceito de localidade de referência espacial (probabilidade alta de dois pixeis vizinhos terem valores próximos ou iguais) esta transformação atua muito eficientemente na preparação de dados de fotografias para compressão. A fórmula da transformada bidimensional é

$$G_{ij} = \frac{1}{\sqrt{2n}} C_i C_j \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} P_{xy} cos\left(\frac{(2y+1)j\pi}{2n}\right) cos\left(\frac{(2x+1)i\pi}{2n}\right)$$

Onde n é igual a 8, $C_f = 1$ quando $f > 0$ e $C_f = \frac{1}{\sqrt{2}}$ quando $f = 0$. G_{ij} é o valor na matriz transformada, P_{xy} é o valor do pixel correspondente na imagem, e i, j, x e m variam entre 0 e 7.

A transformada inversa discreta do cosseno bidirecional (conhecida como $IDCT$) apresenta a seguinte formulação:

$$P_{xy} = \frac{1}{4} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} C_i C_j G_{ij} cos\left(\frac{(2x+1)i\pi}{2n}\right) cos\left(\frac{(2y+1)j\pi}{2n}\right)$$

Onde n é igual a 8, $C_f = 1$ quando $f > 0$ e $C_f = \frac{1}{\sqrt{2}}$ quando $f = 0$. P_{xy} é o valor do pixel correspondente na imagem, G_{ij} é o valor na matriz transformada, e i, j, x e m variam entre 0 e 7.

A fórmula da transformada direta pode ser facilmente traduzida a um programa em qualquer linguagem (neste caso em Python)

```
import math
import numpy as np
def c(x):
  if x==0:
    return 1/math.sqrt(2.0)
  else:
    return 1.0
def dct(ima):
  r=np.zeros((8,8),float)
  i=0
  while (i<8):
    j=0
    while(j<8):
      s=0.0
      x=0
      while x<8:
        y=0
        while y<8:
          t=ima[x][y]
          s=s+t*(math.cos((1+2.0*y)*j*math.pi/16.0))*
             (math.cos((1+2.0*x)*i*math.pi/16.0))
          y=y+1
        x=x+1
      t=0.25*(c(i))*(c(j))*s
      r[i][j]=t
      j=j+1
    i=i+1
  return r
im=np.array([[1, 19, 37, 55, 73, 91, 109, 127],
[19, 37, 55, 73, 91, 109, 127, 145],
[37, 55, 73, 91, 109, 127, 145, 163],
[55, 73, 91, 109, 127, 145, 163, 181],
[73, 91, 109, 127, 145, 163, 181, 199],
[91, 109, 127, 145, 163, 181, 199, 217],
[109, 127, 145, 163, 181, 199, 217, 235],
[127, 145, 163, 181, 199, 217, 235, 253]],float)

for linha in dct(im):
    for val in linha:
        print ('%7.1f '% val,end='')
    print ()
```

Exemplo

Suponha um bloco de imagem com os seguintes valores

```
  1   19   37   55   73   91  109  127
 19   37   55   73   91  109  127  145
 37   55   73   91  109  127  145  163
```

```
 55   73   91  109  127  145  163  181
 73   91  109  127  145  163  181  199
 91  109  127  145  163  181  199  217
109  127  145  163  181  199  217  235
127  145  163  181  199  217  235  253
```

A transformada discreta do cosseno para este bloco é

```
 1016.0 -328.0  0.0 -34.3  0.0 -10.2  0.0 -2.6
 -328.0    0.0  0.0   0.0  0.0   0.0  0.0  0.0
    0.0    0.0  0.0   0.0  0.0   0.0  0.0  0.0
  -34.3    0.0  0.0   0.0  0.0   0.0  0.0  0.0
    0.0    0.0  0.0   0.0  0.0   0.0  0.0  0.0
  -10.2    0.0  0.0   0.0  0.0   0.0  0.0  0.0
    0.0    0.0  0.0   0.0  0.0   0.0  0.0  0.0
   -2.6    0.0  0.0   0.0  0.0   0.0  0.0  0.0
```

Note a quantidade enorme de valores 0.0, o que vai facilitar muito a compressão da imagem. Outro detalhe importante é que a taxa de compressão pode ser ajustada como sendo um parâmetro do método.

4.3 Padrão GIF

Antes de estudar o padrão GIF vai-se estudar o algoritmo usado para comprimir os dados que compõe o fluxo da imagem. Trata-se do algoritmo LZW.

Algoritmo Lempel-Ziv Welch (LZW)

Acrônimo dos nomes dos seus inventores Lempel, Ziv e Welch. Lempel e Ziv publicaram suas idéias em 1977 (ZIV, Jacob, LEMPEL, Abraham; Compression of Individual Sequences Via Variable-Rate Coding, IEEE Transactions on Information Theory, September 1978) e Terry Welch refinou o algoritmo em 1984. (WELCH, T. A. (June 1984). "A technique for high-performance data compression."Computer. Vol. 17, pp. 8-19.)

Em síntese, o algoritmo troca conjuntos de caracteres por códigos simples. Não há análise prévia do conteúdo e a compressão ocorre quando um conjunto grande de símbolos é trocado por um único código. Os códigos LZW podem ter qualquer número de bits, mas precisam ser maiores do que a representação de 1 caracter (ou seja, maiores do que 8 bits). Os primeiros 256 códigos são assinalados para o conjunto padrão inicial. Em um exemplo com códigos de 12 bits, de 0 a 255 são caracteres simples e de 256 a 4095 são para sequências.

Este algoritmo foi o escolhido para comprimir imagens do tipo GIF.

Sejam os algoritmos

```
1: função COMPRIME {le caracteres e imprime códigos numéricos}
2: ST ← primeiro caracter
3: while há caracteres na entrada do
4:    CA ← próximo caracter
5:    if ST + CA está no dicionário then
6:       ST ← ST + CA
7:    else
8:       imprima o número da linha de ST {vem a ser o codigo de ST}
9:       adicione ST + CA no dicionário {na primeira linha disponivel}
10:      ST ← CA
11:   end if
12: end while
13: imprima o número da linha de ST {vem a ser o código}
14: fim {função}
```

```
1: função DESCOMPRIME {le códigos numéricos e imprime caracteres}
2: OL ← primeiro código
3: CA ← caracter que está na linha OL
4: imprima CA
5: while há códigos na entrada do
6:    NE ← próximo código
7:    if NE aponta para linha vazia em dic then
8:       ST ← caracter (es) da linha OL de dic
9:       ST ← ST + CA
10:   else
11:      ST ← caracter (es) da linha NE de dic
```


12: **end if**
13: imprima ST
14: CA ← primeiro caractere de ST
15: inclua o caracter "OL"+ CA na primeira linha livre de dic
16: OL ← NE
17: **end while**
18: fim {função}

Problemas

O clássico problema dos algoritmos baseados em dicionários é a memória necessária. Quanto mais dados do arquivo são lidos, mais seqüências precisam ser armazenadas no dicionário, e com isso, necessita-se de mais memória. O outro problema que isso causa é de espaço de endereçamento: os códigos das entradas no dicionário crescem junto com ele (em geral ocupam $log_2(N)$ bits, onde N é o tamanho, ou número de entradas, do dicionário). Várias abordagens podem ser adotadas para lidar com esse problema, as mais simples são:

- Congelamento do dicionário (quando o dicionário atinge o tamanho limite, ele fica "congelado"e mais nenhuma entrada pode ser adicionada nele).

- Esvaziamento (o dicionário é esvaziado e a compressão começa toda de novo). Essa técnica pressupõe que é mais vantajoso usar a redundância local que a redundância global para a compressão. Corresponde a separar o arquivo em "blocos"comprimidos separadamente.

- Uso de uma política de esvaziamento que seja comum tanto ao compressor quanto ao descompressor (apagar entradas mais antigas, por exemplo).

Exemplo

Para este exemplo:

- Considere textos construidos apenas com as 10 primeiras letras

- O dicionário terá 40 entradas. As 10 primeiras são ABCDEFGHIJ

- As mensagens terão 30 caracteres de comprimento

Seja comprimir a mensagem:

```
BDEFFFFFFFECEGGAFHCFJDICEBHJCE
```

Após a aplicação do algoritmo acima sobre esta mensagem o dicionário ficará

1	A
2	B
3	C
4	D
5	E
6	F
7	G
8	H
9	I
10	J
11	BD
12	DE
13	EF
14	FF
15	FFF
16	FFFF
17	FE
18	EC
19	CE
20	EG
21	GG
22	GA
23	AF
24	FH
25	HC
26	CF
27	FJ
28	JD
29	DI
30	IC
31	CEB
32	BH
33	HJ
34	JC
35	
36	
37	
38	
39	
40	

1	A
2	B
3	C
4	D
5	E
6	F
7	G
8	H
9	I
10	J
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	
32	
33	
34	
35	
36	
37	
38	
39	
40	

1	A
2	B
3	C
4	D
5	E
6	F
7	G
8	H
9	I
10	J
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	
32	
33	
34	
35	
36	
37	
38	
39	
40	

E o resultado será

```
2 4 5 6 14 15 6 5 3 5 7 7 1 6 8 3 6 10 4 9 19 2 8 10 19
```

4.3.1 O padrão GIF

```
Header: 6 bytes (obrigatório)
0-2: GIF
3-5: 87a ou 89a

Descritor de tela lógica: 7 bytes (obrigatório)
0-1: largura da imagem em pixels (max=65536)
2-3: altura da imagem (max=idem)
4 bit 0:indicador de tabela cores global (0=nao, 1=sim)
  bits 1-3: numero de bits por cor - 1(111=7 aqui significa 8 bits/cor)
  bit 4: 1=tabela global classificada, 0=nao
  bits 5-7: tamanho da tabela (2^{este valor+1}) entradas
5: indice da cor de background
6: razao da altura [entre 1:4 e 4:1 = (valor+15)/64]

Se tiver tabela global de cores, vem a seguir (tamanho entre 6 e 768 b)

Descritor de imagem: tamanho 10 bytes (obrigatorio ao menos 1)
```



```
identificador 2C
byte 0: 2C
1-2: deslocamento lateral desta imagem
3-4: deslocamento vertical ...
5-6: largura desta imagem
7-8: altura desta imagem
byte 9 - bit 0: 0=não há tabela local de cores; 1=sim
  bit 1: 1=imagem entrelacada, 0=nao
  bit 2: 1=tabela local ordenada, 0=nao
  3-4: reservado
  5-7: tamanho da tabela local de cores

Dados: primeiro byte: numero inicial de bis usados pelo LZW
Demais: sub-blocos de dados. Cada um começa por byte de tamanho e
        termina por 00.

Outros Blocos
21F9 = bloco de controle
21FE = bloco de comentário
2101 = bloco de texto plano
21FF = bloco de aplicação
3B = terminador de fluxo

21F9 - Bloco de controle:
byte 0: tamanho
byte 1-bits 0-2: reservado (zeros)
       bits 3-5: metodo de substituicao (0=nada, 1=não há substituição;
       2=área retornada a cor de background, 3=voltar ao que havia antes;
       4 a 7: (uso futuro)
       bit 6: 1=aguardar ação do operador, 0=não aguardar
       bit 7: 0=não há cor transparente 1=há cor
byte 2-3: retardo em centésimos de segundo
byte 4: cor transparente (quando há)
byte 5: 00 (terminador)
```

4.4 Padrão FITS

FITS é uma sigla que significa *Flexible Image Transport System* e é um padrão aberto que define um formato de arquivo digital usado para armazenar, transmitir e processar imagens (originalmente) científicas. O padrão nasceu na área de astronomia justamente para permitir o intercâmbio de imagens digitais obtidas em telescópios de todas as naturezas possíveis. O arquivo é suficientemente genérico para acomodar muitos dados científicos, além da imagem em si. Por exemplo, é comum haver dados de fotometria, informações de calibração, filtros usados além de muitas informações sobre a imagem em si, como por exemplo a região do céu fotografada, o que chamamos metadados.

O padrão FITS surgiu em 1981 e evoluiu gradualmente desde então, a versão mais recente é a 3 de 2008. O mote do FITS é *once FITS, always FITS* e ele tem a pretensão de ser legível e manuseável mesmo depois que o software e o hardware usados para criar a imagem original não existam mais. O formato ganhou visibilidade quando foi escolhido pela Biblioteca do Vaticano para disponibilizar imagens de documentos daquele lugar, talvez a mais importante biblioteca de documentos antigos do mundo, repleta de manuscritos dos séculos V, VIII, XI e assim por diante.

Suas principais características:

Não proprietário O formato FITS é mantido pela comunidade científica internacional, em particular a comunidade de astrônomos e astrofísicos e como tal ele não é formato proprietário. Atualmente a especificação é mantida pelo Grupo de Trabalho FITS da IAU, União Internacional de Astronomia.

Bem documentado O formato FITS está completamente documentado na Referência FITS, livremente disponível. A versão 3 foi publicada no *Astronomy and Astrophysics* volume 524 de dezembro de 2010.

Largamente adotado O formato é comumente usado pelos astrônomos. Há muitas aplicações de software para criar e ver os arquivos, além de permitir a importação e exportação em outros formatos também usuais.

Transparência Uma grande vantagem é a simplicidade do formato. Nada é comprimido (o que introduziria grande risco em caso da corrupção de um único bit...) e os dados são guardados ora em ASCII puro, ora em formatos de codificação padronizados.

Auto contido Todas as informações necessárias para representar corretamente a imagem (na tela ou na impressora) estão incluídas dentro do arquivo. Links externos ou fontes de recursos externos não são permitidos.

Independência de dispositivo Não há nenhuma dependência a algum hardware ou software específicos em nenhum ponto do arquivo. Esta característica é muito importante na compatibilidade para a frente, pois não temos idéia do que vem por aí.

4.4.1 Formato FITS

O arquivo é formado por conjuntos de header + dados, tantos quantos houver interesse. O primeiro conjunto é chamado primário e é obrigatório. Quando só tem o primário o arquivo é conhecido como Arquivo FITS básico e quando tem uma ou mais extensões seguindo o conjunto primário é chamado de Arquivo FITS multi-extensão. Cada conjunto (chamado na documentação de *FITS structure*) é formado por um número inteiro de blocos FITS, cada um com 2880 bytes e escrito em ASCII puro ($2880 = 80 \times 36$). Todos estes blocos ficam juntos, antes dos dados. Por ASCII puro entende-se caracteres de 32 até 126.

O bloco de header contém uma série de palavras chaves seguida de seu valor. Assim, cada header tem espaço para 36 palavras chave. O último bloco de header deve conter a palavra chave `END` que marca o fim lógico do header. O que sobrar de espaço depois, deve ser prenchido com espaços.

O conjunto de dados que vem depois, (se presente) deve consistir de um arranjo (array) de dados de 1 a 999 dimensões (isto é definido pela palavra chave `NAXIS` O array inteiro é representado por um stream contínuo de bits que começa no primeiro bit do primeiro bloco de dados. Cada dado consiste de um número fixo de bits como determinado na palavra chave `BITPIX`. O índice do eixo 1 é o que varia mais rapidamente, depois o do eixo 2, e assim por diante. O último eixo (`NAXIS`) é o que varia mais lentamente. Não há espaço ou qualquer caractere especial entre o último valor em uma linha ou plano e o primeiro valor da próxima linha ou plano. A contagem começa em 1. Veja na figura como é isso:

```
A(1, 1, ..., 1),
A(2, 1, ..., 1),
...,
A(NAXIS1, 1, ..., 1),
A(1, 2, ..., 1),
A(2, 2, ..., 1),
...,
A(NAXIS1, 2, ..., 1),
...,
A(1, NAXIS2, ..., NAXISm),
...,
A(NAXIS1, NAXIS2, ..., NAXISm)
```

Se o bloco de dados não preenche o último bloco ele deve ser preenchido com bits zero.

O bloco de header é formado por registros de 80 caracteres (36 por bloco) e eles consistem de uma palavra chave, um indicador de valor (se necessário), um valor opcional e um comentário opcional. As palavras podem aparecer em qualquer ordem, com poucas exceções. A palavra chave deve ter até 8 posições, alinhada à esquerda e com brancos à direita, sempre em letras maiúsculas. As posições 9 e 10 podem ter "`=␣`". A seguir o valor da palavra chave e opcionalmente um comentário. Entre o valor e o comentário deve-se escrever "`␣/␣`". Em todo este conteúdo (inclusive comentários) deve-se usar apenas os caracteres ASCII de 32 até 126 – hexadecimal `20` até `7E`. Variáveis de valor lógico devem ser representadas por `T` ou `F`.

Eis as principais palavras chave:

`SIMPLE` Deve conter um valor verdade (`T`) se este arquivo obedece ao padrão FITS.

`BITPIX` Contém um inteiro e indica o número de bits que representa cada valor no array de dados. Os valores possíveis são: 8, 16, 32, 64 – para inteiros – e -32 ou -64 para ponto flutuante.

`NAXIS` Inteiro não negativo não maior que 999 indicando quantos eixos tem o array de dados.

`NAXIS`**n** Esta palavra deve estar presente para todos os valores de n de 1 até `NAXIS`. Indica o tamanho da dimensão do array de dados.

`END` Indica final do cabeçalho. Esta palavra chave não tem valor associado

Veja-se um exemplo disto:

```
SIMPLE   = T / file does conform to FITS standard
BITPIX   = 16 / number of bits per data pixel
NAXIS    = 2 / number of data axes
NAXIS1   = 250 / length of data axis 1
```



```
NAXIS2   = 300 / length of data axis 2
OBJECT   = 'Cygnus X-1'
DATE     = '2008-10-27'
END
```

O número total de bits do array de dados primário, sem contar o preenchimento eventualmente necessário para completar o bloco de 2880 caracteres, é $N_{bits} = |BITPIX| \times (NAXIS1 \times NAXIS2 \times ... \times NAXISm)$

A cor no arquivo FITS

Paradoxalmente, a cor não tem um significado fixo no arquivo FITS. Ela pode (deve) ser livremente interpretada numa negociação entre gravador e leitor. O formato lava as mãos sobre isso. Para imagens coloridas do uso no nosso dia a dia, pode-se fazer a seguinte combinação: true color (8 bits=0..255) por canal R, G, B e mais um quarto canal de transparência. Total 32 bits por pixel. Neste caso, pode-se usar o inteiro de 32 bits no registro do arquivo. Se uma maior precisão de cor é exigida (como por exemplo na captura de pergaminhos e iluminuras da Biblioteca Vaticana), pode-se fazer: 16 bits por canal R, G ou B chegando portanto a 65.536 níveis em cada cor, e mais um canal de transparência com o mesmo nível de valores. Total 64 bits por pixel, usando-se neste caso os inteiros de 64 bits. Ou pode-se definir qualquer outra combinação, eventualmente usando-se outros esquemas de cor (como CMYB, por exemplo). O que resta importante no formato, é que cada valor sempre tem uma configuração que é múltipla de 8 bits, ou seja não há aqui os problemas de truncamento, padding ou stuffing bits de outros formatos.

Como ver arquivos FITS

Existe uma grande variedade de visualizadores de arquivos FITS, e como tudo no padrão, são livres para download e instalação. Um programa que é bem completo e funciona inclusive como editor para este tipo de arquivo é o FV, que tem versões para plataformas windows, unix e Mac. Seu site é https://heasarc.gsfc.nasa.gov/ftools/fv/. Vale uma visita. Outro é DS9, que pode ser baixado em http://ds9.si.edu/site/Home.html.

Especial referência deve ser feita em relação à linguagem Python. Usando a vocação desse ambiente, (106.000 pacotes em mar/17), há um pacote completo chamado ASTROPY que dá toda a funcionalidade para operar magnificamente com arquivos FITS. Mais detalhes em www.astropy.org, onde há um tutorial muito bom.

Capítulo 5

Plataformas

Agora é hora de dedicar algum interesse aos softwares, computadores, dispositivos, linguagens e similares que permitem que imagens numéricas sejam criadas, guardadas, vistas, modificadas, transmitidas, impressas em papel e assim por diante.

5.1 Captura

Para a captura de uma imagem pode-se lançar mão de

câmeras do tipo digital, que permitem fotografar uma cena e gerar o arquivo (geralmente JPG ou RAW. Este último é dedicado à guarda dos dados capturados sem nenhum pré ou pós processamento, ou seja em estado bruto) para passar para o próximo elemento da cadeia.

scanners dispositivos que permitem capturar imagens já impressas em papel, gerando o mesmo tipo de arquivo.

smartfones os mais modernos contam com uma ou mais câmeras de captura. A novidade aqui é a disponibilidade e a portabilidade deste dispositivo.

capturador de tela Praticamente todos os computadores contam com algum auxílio de captura de tela. Nestes casos, para capturar alguma coisa publicada na Internet, basta apresentá-la na tela e chamar a captura, procedendo-se depois ao processamento da imagem capturada como desejado.

5.2 Python

Para manipular qualquer imagem numérica o pré-requisito óbvio é um computador. Mas, além dele são necessárias mais coisas.

Comecemos com a linguagem Python e seus abundantes pacotes. Python é uma criação genial de Guido Van Rossum em 1991. Ela representa uma mudança sutil mas profunda na maneira de programar. Até então – por uma série de razões – talvez a principal sendo a raridade e alto custo da computação, privilegiava-se a economia e eficência da programação. Python rompe isto sugerindo que o que se deve economizar (ou eficientizar, vá lá) é o esforço humano na programação. Isto explica o fabuloso sucesso da linguagem.

Com a palavra a wikipédia *Python é uma linguagem de propósito geral de alto nível, multiparadigma, suporta os paradigmas orientado a objetos, imperativo, funcional e procedural. Possui tipagem dinâmica e uma de suas principais características é permitir a fácil leitura do código e exigir poucas linhas de código se comparado ao mesmo programa em outras linguagens.*

Python pode ser buscado em `www.python.org`. Embora existam disponíveis duas grandes versões (a 2 e a 3), a 3 sempre deve ser a escolhida. A versão última atual (em 23/03/21) é a 3.9.2. Roda em praticamente tudo que tem um computador por trás (Windows, Linux, Amiga, Android, ...).

5.2.1 PIL

Python Imaging Library (PIL) é um pacote que viabiliza o tratamento de imagens em programas Python. Ele permite criar, modificar e converter arquivos de imagens em larga variedade de formatos. A documentação completa deste pacote está em `http://www.pythonware.com/library/ pil/handbook/index.htm`.

Instalação Tendo acesso à Internet, entre no diretório de instalação do Python, no subdiretório `Scripts` dentro da árvore de instalação do Python. Abrindo uma janela CMD nesse diretório execute

```
pip install pil
```

Importação Há duas maneiras de importar. A mais simples é `>>> from pil import *` Aqui todos os objetos do pacote são incluidos no espaço de nomes atual. A segunda maneira, mais higiênica, é atribuir um nome ao pacote e referenciar seus objetos qualificando-os por este nome. O comando agora é `>>> import pil as pl`. O nome do pacote agora é `pl`.

Modos O modo de uma imagem descreve a maneira pela qual ela representa cores. O modo é identificado por uma string:

Modo	B.	Descrição
'1'	1	Preto e branco, 1 bit por pixel.
'L'	1	Escala de cinza, 8 bits por pixel.
'P'	1	Cor mapeada por paleta. 8 bits por pixel. A paleta da classe `ImagePalette` descreve as cores.
'RGB'	3	True color, três bytes por pixel.
'RGBA'	4	True color mais uma banda de transparência. O canal A vale 0 para transparente e 255 para opaco.
'CMYK'	4	Cores ciano-magenta-yellow-black.
'YCbCr'	3	Formato de vídeo.
'I'	1	pixels valendo inteiros de 32 bits
'F'	1	pixels valendo floating de 32 bits.

Tamanho Os tamanhos de objetos em imagens é escrito por uma tupla `(w,h)` w=largura e h=altura.

Coordenadas O sistema de coordenadas considera o ponto `(0,0)` como sendo o canto alto da esquerda.

Ângulos Dados em graus. Zero graus é a direção X. Os ângulos crescem no sentido contrário ao do relógio.

Bboxes Um bounding box ou bbox é um retângulo da imagem. É definido por uma tupla de 4 números, x_0, y_0, x_1, y_1, onde x_0, y_0 é o alto à esquerda e x_1, y_1 é o baixo à direita. Ele inclui os pontos iniciais, mas, coerente com o jeito de pensar do Python, não inclui os pontos finais.

Cores

- Para imagens de banda única, a cor é o valor do pixel. Em imagem '1', 0 é preto e 1 é branco. No modo 'L', o valor está entre `[0,255]`, onde 0 é preto e 255 é branco.
- Para imagens multi-banda, a cor é uma tupla com um valor para cada banda. Na imagem 'RGB', a cor (255,0,0) é vermelho.
- Pode-se usar nomes no estilo do CSS, com uma string no formato `#rrggbb` em hexadecimal.
- Para pixels RGB em decimal, use uma string no formato `'rgg(0,255,0)'`. Este exemplo é verde.
- Para pixels RGB em percentagem use uma string no formato `'rgb(r%,g%,b%)'`.
- Para HSV use um string no formato `'hsl(H,S%,L%)'`. H é o ângulo de hue em graus: 0 é vermelho, 60 é amarelo, 120 é verde e assim por diante. S é a saturação: 0% é cinza e 100% é saturação total. O brilho L é 0% para preto, 50% para normal e 100% para branco.
- em sistemas Unix, pode-se usar nomes como 'white' ou 'coral', por exemplo.

Filtros Algumas operações reduzem o número de pixels, (como criar um thumbnail) e podem usar filtros para calcular os novos valores de pixels. Estes incluem

- NEAREST: Usa o valor do pixel mais próximo
- BILINEAR: Interpolação linear sobre um conjunto 2×2 de pixeis adjacentes
- BICUBIC: Interpolação sobre um conjunto de 4×4 pixeis.
- ANTIALIAS: Pixeis vizinhos são estudados para achar o novo valor.

criando objetos da classe Image Após um `Import Image`, pode-se criar instâncias deste objeto (que contém uma imagem). As seguintes funções retornam um objeto Image:

Image.open(f) Lê uma imagem do arquivo f, dado como uma string.

Image.new(m,t,color=None) Cria uma nova imagem. m é modo, t é tamanho. Se a cor é oferecida ela é o fundo, senão é preto.

Image.blend(i1, i2, a) Mistura i1 e i2, usando a fórmula (para cada pixel) $p_1 \times (1-a) + p_2 \times a$.

Image.eval(f,i) Aplica a função f a cada pixel da imagem i
Há mais, veja na documentação...

Atributos do objeto Image

.format Contém o formato do arquivo, por exemplo 'GIF'

.mode o modo da imagem, como uma string

.size uma tupla `(w,h)`

.palette se for o caso

.info Um dicionário com dados associados à imagem.

Métodos do objeto Image

.save(f,format=None) Grava a imagem no arquivo f. Formato pode ser 'JPEG' ou 'TIFF'. Se o formato for omitido ele será recuperado do nome do arquivo

.convert(modo) Converte para modo. Por exemplo para converter para 'CMYK' escreva `im.convert('CMYK')`.

.copy() Retorna uma cópia da imagem

.crop(bbox) retorna apenas o conteúdo da bbox

.filter(nome) retorna a imagem filtrada

.histogram(mask=None) retorna uma lista de valores (histograma)

.resize(size,filter=None) modifica o tamanho. O default é NEAREST.

.rotate(theta) rotaciona ao redor do centro em θ graus

.show() mostra a imagem

Módulo ImageDraw Permite criar imagens a partir do zero. Seus métodos incluem `arc`, `chord`, `ellipse`, `line`, `pieslice`, `point`, `polygon`, `text`, `textsize`, ...

Módulo ImageFilter Permite aplicação de filtros. Chamado com `import ImageFilter`. Possui os seguintes `BLUR`, `CONTOUR`, `DETAIL`, `EDGE_ENHANCE`, `EDGE_ENHANCE_MORE`, `EMBOSS`, `FIND_EDGES`, `SMOOTH`,`SMOOTH_MORE`, `SHARPEN`.

Módulo ImageFont Para especificar fontes

ImageTk Para escrever módulos para o Tk.

Formatos de arquivos suportados

BMP Os modos de leitura/gravação: 1, L, P e RGB

EPS As extensões podem ser EPS ou PS e os modos: 1, L, P, RGB

GIF Só o modo P

JPEG As extensões: JPG, JPE e JPEG. Os modos: L, RGB e CMYK

PDF Não lê e para gravar aceita os modos 1 e RGB.

PNG Aceita os modos 1, L, P, RGB, RGBA
tem mais, veja a documentação.

5.2.2 Open CV

O pacote OCV (*Open Source Computer Vision*) agrega funções para processamento e análise de imagens. O pacote foi desenvolvido originalmente em C++ e rapidamente migrado para Python. Usa como pré-requisitos os seguintes pacotes

- Numpy: numerical python
- Matplotlib: gerador de gráficos no estilo Matlab

Não precisa se preocupar, se o instalador detectar que estes pacotes não estão presentes ele (o instalador) vai carregar e instalar estes dois pacotes.

A instalação do openCV se dá pelo comando

```
pip install opencv-python
```

emitido no diretório onde estiver instalado o pip.exe. Tipicamente é

```
cd\python\scripts
```

Depois de instalar o pacote, teste o funcionamento do mesmo dentro do python emitindo:

```
>>>import cv2
```

Se o python responder `>>>` sem nenhuma outra mensagem, é porque a instalação deu certo.

Algumas funções

A seguir uma lista de algumas funções do pacote

Função	o que faz	forma de chamar
Ler uma imagem	carrega para dentro do programa uma ou mais matrizes (vide canal)	`im=cv2.imread('arquivo.jpg',dica)`

`dica` é opcional e indica como a leitura deve ser feita. Seus valores:

1 leia o arquivo em true color

0 leia o arquivo em niveis de cinza

-1 leia o arquivo como ele foi criado

Após este comando, a variável `im` tem associada as seguintes informações:

variável	conteúdo
`im.shape[0]`	altura da imagem em pixels
`im.shape[1]`	largura da imagem em pixels
`im.shape[2]`	quantidade de canais da imagem

Função	o que faz	forma de chamar
Mostrar uma imagem	abre uma janela e escreve nela a imagem	`im=cv2.imshow('nome da janela',im)`
Gravar a imagem em disco	grava a imagem (eventualmente alterada)	`cv2.imwrite("arquivo.jpg",im)`

Se a imagem for branco e preto, haverá um único canal (uma matriz com o nome de `im`. Se a imagem for true color, haverá 3 matrizes de mesmo tamanho, a primeira sendo `blue` a segunda `green` e a terceira `red`. Acompanhe o programa abaixo

```
def ima1():
  import cv2
  imagem=cv2.imread('jovem.jpg')
  for i in range(0,imagem.shape[0],10):
    for j in range(0,imagem.shape[1],10):
#       (a,b,c)=imagem[i,j]
      imagem[i:i+5,j:j+5]=(0,255,255)
  cv2.imshow('olhe',imagem)
ima1()
```

Função	o que faz	forma de chamar
Desenha uma linha	desenha uma linha na imagem	cv2.line(im, (origem), (destino), cor)
Desenha um retângulo vazado	cria um retângulo se largura = -1, figura é cheia. Default é 1.	cv2.rectangle(im,(esqsup),(dirinf0, cor, largura-risco)
Desenha circulo	desenha um circulo	cv2.circle(im, (centro), raio, cor)

Acompanhe o programa

```
import numpy as np
import cv2
im = cv2.imread('oba.jpg')
vermelho = (0,0,255)
verde=(0,255,0)
azul=(255,0,0)
cv2.line(im,(5,5),(30,60), verde)
cv2.rectangle(im,(20,30),(100,120),azul,-1)
(X, Y) = (im.shape[1]//2, im.shape[0]//2)
cv2.circle(im,(X,Y),30,azul)
```

Para recortar (crop) uma imagem, faça:

```
import cv2
im=cv2.imread('oba.jpg')
recorte = im[100:200, 100:200]
cv2.imshow("Recorte",recorte)
cv2.imwrite("recorte.jpg",recorte)
```

Função	o que faz	forma de chamar
redimensionar	redimensiona uma imagem	imn=cv2.resize(im,(tamanho novo), interpolation=método)

o método pode ser:

cv2.INTER_AREA for shrinking

cv2.INTER_CUBIC (slow)

cv2.INTER_LINEAR for zooming.

Por default o método usado é `INTER_LINEAR`. Veja o código

```
import numpy as np
import cv2
im=cv2.imread('oba.jpg')
largura=im.shape[1]
altura=im.shape[0]
proporcao=float(altura/largura)
largura_nova=500
altura_nova=int(largura_nova*proporcao)
tamanho_novo=(largura_nova, altura_nova)
imredimens=cv2.resize(im,tamanho_novo,interpolation=cv2.INTER_AREA)
cv2.imshow('Resultado',imredimens)
cv2.waitKey(0)
```

A seguir, como fazer flip vertical e horizontal

```
import cv2
im = cv2.imread('oba.jpg')
flip_hor = im[::-1,:]
flip_vert = im[:,::-1]
flip_ambos = im[::-1,::-1]
... gravar, mostrar, etc
```

Para rotar uma imagem:

```
def ri(): #roda imagem
  import cv2
  im=cv2.imread("pedro_jovem.jpg")
```



```
  (alt,lar)=im.shape[:2]
  centro=(lar//2,alt//2)
  m=cv2.getRotationMatrix2D(centro,-10,1.0) #-10 graus
  imr=cv2.warpAffine(im,m,(lar,alt))
  cv2.imshow("rodada",imr)
  cv2.waitKey(0)
ri()
```

5.2.3 Numpy

Embora não específico para imagens, mas sim para computação numérica, este pacote implementa o conceito de array n-dimensional. Também o conceito de funções universais além de centenas de funções e constantes a usar em computação matemática e científica. É pré-requisito em muitos outros pacotes (ex: scipy, sympy, matplotlib, tensorflow, pygame, astropy, opencv, pulp, pandas entre outros).

O pacote é muito extenso e completo, seu manual tem 1542 páginas e pode ser encontrado via google, buscando `numpy-ref-1.12.0.pdf` ou uma versão mais recente.

Instalação Tendo acesso à Internet, entre no diretório de instalação do Python, no subdiretório `Scripts` dentro da árvore de instalação do Python. Abrindo uma janela CMD nesse diretório execute

```
pip install numpy
```

Se não souber fazer isto ou não tiver direitos na máquina em questão, mude de estratégia e use o Winpython. Para seu governo este pacote bem como muitos outros, já está pré-instalado no `winpython`.

Importação Há duas maneiras de importar. A mais simples é `>>> from numpy import *` Aqui todos os objetos do pacote são incluidos no espaço de nomes atual. A segunda maneira, mais higiênica, é atribuir um nome ao pacote e referenciar seus objetos qualificando-os por este nome. O comando agora é `>>> import numpy as np`. O nome do pacote agora é `np`. Pode-se usar qualquer outro nome, mas este material (e 99% da comunidade Python) o conhece como np.

Observar ainda que há diversos subpacotes dentro de numpy. Isso surgirá mais adiante em alguns exemplos. Os mais usados são `linalg` que agrega funções da álgebra linear, `fft` da transformada discreta de Fourier, `random` que trata diversas funções de geração de aleatórios e de distribuições estatísticas.

arranjos Um arranjo em numpy é um array n-dimensional. Quando n=1, o array é unidimensional logo é um vetor e neste caso é parecido com a lista convencional. A coisa começa a melhorar quando n=2 (matriz) ou mesmo quando n é maior do que 2. Em qualquer caso o a classe do pacote é ndarray e alguns de seus métodos são:

array Cria um array a partir de dados já existentes. Tipo tem muitas opções, sendo as mais comuns int, float, complex.

```
>>> import numpy as np
>>> a=np.array(elementos, tipo)
>>> b=np.array([1,2,3,4])
>>> b
array([1, 2, 3, 4])
>>> c=np.array([1,2,3],float)
>>> c
array([1., 2., 3.])
>>> d=np.array([[1,2,3],[0,0,7],[3,3,9]])
>>> d
array([[1, 2, 3],
       [0, 0, 7],
       [3, 3, 9]])
>>> d1=np.array(d,float)
>>> d1
array([[ 1., 2., 3.],
       [ 0., 0., 7.],
       [ 3., 3., 9.]])
```

Interessante que se você mandar imprimir a matriz ela será exibida sem as vírgulas

```
>>> print(d)
[[1 2 3]
 [0 0 7]
 [3 3 9]]
```

`np.set_printoptions(suppress=True)` → desabilita impressão em notação científica.

Na criação as listas podem ser fornecidas como tuplas (usando-se parênteses ao invés de colchetes), mas esta estratégia não será desenvolvida aqui.

Indexação Elementos individuais podem ser recuperados dos arrays. Podem também ser alterados. Cada dimensão do array exigirá uma especificação, separadas por vírgulas. Acompanhe

```
>>> d[1,2]
7
>>> c[1]
2.0
>>> d[1,1]=99
>>> d
array([[ 1,  2,  3],
       [ 0, 99,  7],
       [ 3,  3,  9]])
```

Fatiamento Usando a mesma estratégia do python, sub-arrays podem ser obtidos usando fatiamento

```
>>> e=np.array([[4,7,8,10],
   [11,12,16,19],[20,21,23,25],
   [40,41,50,51]])
>>> e
array([[ 4,  7,  8, 10],
       [11, 12, 16, 19],
       [20, 21, 23, 25],
       [40, 41, 50, 51]])
>>> e[1:3,1:3]
array([[12, 16],
       [21, 23]])
>>> e[0:4,1:4]
array([[ 7,  8, 10],
       [12, 16, 19],
       [21, 23, 25],
       [41, 50, 51]])
>>> e[1,1:4]
array([12, 16, 19])
```

Zeros Outra maneira de criar o array contendo zeros. O formato é `zeros((forma),tipo)`. Acompanhe:

```
>>> f=np.zeros((3,2),complex)
>>> f
array([[0.+0.j, 0.+0.j],
       [0.+0.j, 0.+0.j],
       [0.+0.j, 0.+0.j]])
>>> g=np.zeros((4,5),int)
>>> print(g)
[[0 0 0 0 0]
...
 [0 0 0 0 0]]
```

O `fill` preenche com um valor pré-determinado.

```
>>> a = np.fill((3,12),-1,int)
```

cria uma matriz de 3 linhas por 12 colunas de inteiros iguais a -1.

Outros criadores O `eye` funciona como a matriz identidade, mas valendo para matrizes não quadradas. A identidade exige apenas a ordem da matriz (já que ela é quadrada).

```
>>> h=np.eye(3,2)
>>> h
array([[1., 0.],
       [0., 1.],
```



```
       [0., 0.]])
>>> j=np.identity(3)
>>> j
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

Atributos Um array tem os atributos `ndim` que é um escalar indicando a quantidade de dimensões e `shape` que é uma tupla indicando a quantidade de elementos em cada dimensão.

```
>>> j.ndim
2
>>> j.shape
(3, 3)
>>> g.shape
(4, 5)
```

Reorganizar Para reorganizar um array já existente, usa-se o método `reshape`.

```
>>> a1=np.arange(0,15,2)
>>> a1
array([0,2,4,6,8,10,12,14])
>>> a2=a1.reshape((2,4))
>>> a2
array([[ 0,  2,  4,  6],
       [ 8, 10, 12, 14]])
```

Para reorganizar in-place usa-se o método `resize`.

```
>>> b1=np.array([1,7,8,9,10,11])
>>> b1.resize([3,2])
>>> print(b1)
[[ 1  7]
 [ 8  9]
 [10 11]]
```

Funções a seguir algumas (tem inúmeras mais, veja a documentação):

```
>>> a=np.array([[1,4,6],[7,8,1]])
>>> print(a)
[[1 4 6]
 [7 8 1]]
```

a.mean()	média
a.var()	variancia
a.sum()	soma
a.prod()	produto
a.max()	maior valor
a.min()	menor valor
a.argmax()	índice do maior
a.argmin()	índice do menor
np.unique(a)	valores únicos

```
>>> print(a.mean())
4.5
>>> print(a.max())
8
>>> print(a.argmax())
4
>>> print(np.unique(a))
[1 4 6 7 8]
```

Funções Universais As funções universais (ufuncs) se aplicam elemento a elemento dentro dos arrays. Acompanhe o exemplo abaixo para identificar o funcionamento das ufuncs.

```
>>> c1=[1,2,3]
>>> c2=[3,7,9]
>>> c1+c2
[1, 2, 3, 3, 7, 9]
>>> d1=np.array([1,2,3])
>>> d2=np.array([3,7,9])
>>> d1+d2
array([ 4,  9, 12])
```

Broadcasting Para broadcasting (aplicar um escalar a um array)

```
>>> a é array([1,2,3,4])
>>> a+3
array([4,5,6,7])
>>> b*5
array([0.5, 3.5, 40., 45.])
```

multiplicação matricial

```
>>> a1=np.array([[1,2,-4],
     [3,-1,5]])
>>> a2=np.array([[6,-3],
     [1,-2],[2,4]])
>>> np.dot(a1,a2)
array([[  0, -23],
       [ 27,  13]])
```

Produto vetorial

```
>>> x = (1, 2, 0)
>>> y = (4, 5, 6)
>>> print np.cross(x, y)
[12 -6 -3]
```

Determinante

```
>>> m=np.array([[2,3],[-1,-2]])
>>> m
[[ 2 3]
[-1 -2]]
>>> np.linalg.det(m)
-1.0
```

Matriz inversa

```
>>> good = np.array([[2.1,3.2],
       [4.3,5.4]])
>>> np.linalg.inv(good)
[[-2.23140496 1.32231405]
[ 1.7768595 -0.8677686 ]]
```

Sistema de equações lineares A solução de um sistema de equações lineares por exemplo

```
2x + y = 19
x - 2y = 2
cuja solução é (x=8 and y=3):
>>> coef=np.array([[2,1],[1,-2]])
>>> cont=np.array([19, 2])
>>> np.linalg.solve(coef, cont)
[ 8. 3.]
```

Se $m > n$, onde m são equações e n são incógnitas, o comando é `c=np.linalg.lstsq(a,b)` que devolve a solução que minimiza o erro cometido. (`solve`, acima, exige $m = n$).

Transposta

```
>>> a=np.array(range(6),
     float).reshape((2,3))
>>> a é array([[ 0., 1., 2.],
               [ 3., 4., 5.]])
>>> a.transpose()
array([[0.,3.],[1.,4.],[2.,5.]])
```

Concatenação

```
>>> a=np.array([1,2],float)
>>> b=np.array([3,4,5],float)
>>> c=np.array([7,8,9],float)
>>> np.concatenate((a,b,c))
array([1.,2.,3.,4.,5.,7.,8.,9.])
>>> a=np.array([[1,2],[3,4]])
>>> b=np.array([[5,6],[7,8]])
>>> np.concatenate((a,b))
array([[ 1., 2.],
[ 3., 4.],
[ 5., 6.],
[ 7., 8.]])
>>> np.concatenate((a,b), axis=0)
array([[ 1., 2.],
[ 3., 4.],
[ 5., 6.],
[ 7., 8.]])
>>> np.concatenate((a,b), axis=1)
array([[ 1., 2., 5., 6.],
[ 3., 4., 7., 8.]])
```

Funções matemáticas abs, sign, sqrt, log, log10, exp, sin, cos, tan, arcsin, arccos, arctan, sinh, cosh, tanh, arcsinh, arccosh, and arctanh. floor (inteiro inferior), ceil (inteiro superior), e rint (inteiro mais próximo = arredondado).

Geração de aleatórios Gera números reais entre 0 e 1.

```
>>> np.random.rand(2,2)
array([[0.07724462, 0.93519669],
       [0.44908731, 0.12275935]])
```

Inteiros aleatórios. O formato é início (inclusive), final (exclusive) e quantidade. Os dois últimos são opcionais.

```
>>> np.random.randint(1,50,3)
array([40,  4, 49])
>>> np.random.randint(6) # dado
4
```

Ordenação

```
>>> a=np.array([[1,4,7],[8,3,9],
      [22,11,13]])
>>> a
array([[ 1,  4,  7],
       [ 8,  3,  9],
       [22, 11, 13]])
>>> np.sort(a)
array([[ 1,  4,  7],
       [ 3,  8,  9],
       [11, 13, 22]])
>>> np.sort(a,axis=None)
array([1,3,4,7,8,9,11,13,22])
>>> np.sort(a,axis=0)
array([[ 1,  3,  7],
       [ 8,  4,  9],
       [22, 11, 13]])
```

Distribuição normal Esta função retorna simulações de uma distribuição normal. Para obter $N(\mu, \sigma^2)$, basta fazer $sigma * np.random.randn(...) + mu$. Onde μ = média da distribuição e σ^2 é a variância. Veja-se o exemplo

```
>>> 2.5 * np.random.randn(2, 4) + 3
array([[-4.49, 4.00, -1.81, 7.29],
       [ 0.39, 4.68,  4.99, 4.84]])
```

Cria um array $2, 4$ contendo simulações de $3, 6.25$. Lembre que $2.5^2 = 6.25$.

Distribuição binomial Cria simulações desta distribuição. Usa 2 parâmetros: `n`, que significa o número de tentativas e `p` que vem a ser a probabilidade de sucesso da distribuição ($0 \leq p \leq 1$). A função retorna para cada simulação o número de sucessos. Lembrando que a probabilidade é

$$P(N) = \left(\begin{array}{c} n \\ N \end{array} \right) p^N (1-p)^{n-N}$$

Veja-se o exemplo:

```
>>> n=10
>>> p=0.5
>>> s=np.random.binomial(n,p,1000)
```

O exemplo acima faz 1000 simulações, cada uma jogando 10 moedas e informando a quantidade de caras em cada simulação.

Desvio padrão em arrays

```
>>> a = np.array([[1, 2], [3, 4]])
>>> np.std(a)
1.1180339887498949
>>> np.std(a, axis=0)
array([ 1., 1.])
>>> np.std(a, axis=1)
array([ 0.5, 0.5])
```

Embaralhando uma lista Supondo uma lista L já existente e querendo embaralhá-la, no próprio local

```
>>> L=[1,4,7,22,11,5,4,77]
>>> L
[1, 4, 7, 22, 11, 5, 4, 77]
>>> np.random.shuffle(L)
>>> L
[4, 5, 7, 1, 4, 22, 77, 11]
```

Ler e Gravar dados Suponha uma variável x como em

```
>>> x=np.random.rand(3,2)
>>> x
array([[0.65615773, 0.64436913],
       [0.54172899, 0.9726057 ],
       [0.27936652, 0.09892232]])
```

Para guardá-la em disco (note que este formato é ilegível por humanos) para posterior recuperação pode-se fazer

```
>>> x.dump('c:/apl2/lixo.dad')
```

E depois basta fazer

```
>>> y=np.load('c:/apl2/lixo.dad')
```

Para colocar uma constante Imagine um array de 6 linhas por 10 colunas que deve ser preechido com 22

```
>>> z=np.empty((10,6))
>>> z.fill(22)
```


Iteração com elementos

```
>>> a=np.arange(9).reshape(3,3)
>>> a
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> for x in np.nditer(a):
    print(x)
...
0 1 2 3 4 5 6 7 8
```

Para medir alguma execução

```
import time
ini=time.time()
... o que quer medir
fim=time.time()
print("tempo em segs",fim-ini)
```

5.3 Octave

Derivado do sofisticado ambiente MatLab (Matrix Laboratory) um programa que custa mais de 1000 dólares cada cópia e que é largamente usado em escolas de engenharia pelo mundo. A iniciativa Octave é uma implementação Freeware deste produto e para quase todos os efeitos práticos, ambas são equivalentes.

Fica claro que MatLab/Octave não são específicos para manipulação de imagens. São ambientes e linguagens de programação absolutamente genéricos e muito completos. Aqui é que vai se dar ênfase a suas capacidades de processamento e análise de imagens.

Como o nome informa o elemento básico de dados em Octave é a matriz. Um único valor em Octave é considerado como uma matriz de 1 linha por 1 coluna.

O comando básico para carregar uma imagem em Octave é

```
>> pkg load image % carrega o pacote de imagens
>> i=imread('d:/imas/campanar.jpg');
>> size(i)
ans =
   719   593     3
```

Eis aqui um arquivo colorido, true color de 719 linhas por 593 colunas. Note que o comando termina por um `;`. Se assim não for, o comando responderá com os valores carregados em `i`. O ponto e vírgula ordena ”trabalhe em silêncio”. Veja também que o nome do arquivo é fornecido entre aspas simples.

Depois deste comando, se você disser `i`, o Octave vai mostrar a matriz numérica que contém a imagem. Para olhar a imagem em sí, o comando é

```
imshow(i)
```

Este comando abre uma janela e mostra nela a imagem carregada.

Agora a leitura de uma imagem em 256 niveis de cinza

```
>> i=imread('d:/imas/campanarc.jpg');
>> size(i)
ans =
   719   593
```

e agora a mesma imagem, mas *true color*

```
>> [i,map]=imread('d:/imas/campanari.tif');
>> size(i)
ans =
   719   593
>> size(map)
ans =
   256     3
```

O comando para obter informações a respeito de um arquivo imagem em Octave é

```
>> imfinfo('arquivo')
```

Há muitas informações aqui, vamos às que interessam:

nome do arquivo

data quando ele foi criado/alterado

Filesize tamanho do arquivo

format tipo do arquivo/imagem

widht/Height largura/altura da imagem

bitdepth quantidade de bits para representar 1 pixel. Aqui há os seguintes valores

- **1** imagem binária
- **8** pode ser um arquivo mapeado via tabela de cores ou um arquivo com 256 níveis de cinza
- **24** uma imagem true-color (8 bits para cada cor RGB)

colortype indexed, truecolor ou grayscale

eventual tabela de cores representada neste caso por valores reais entre 0 e 1

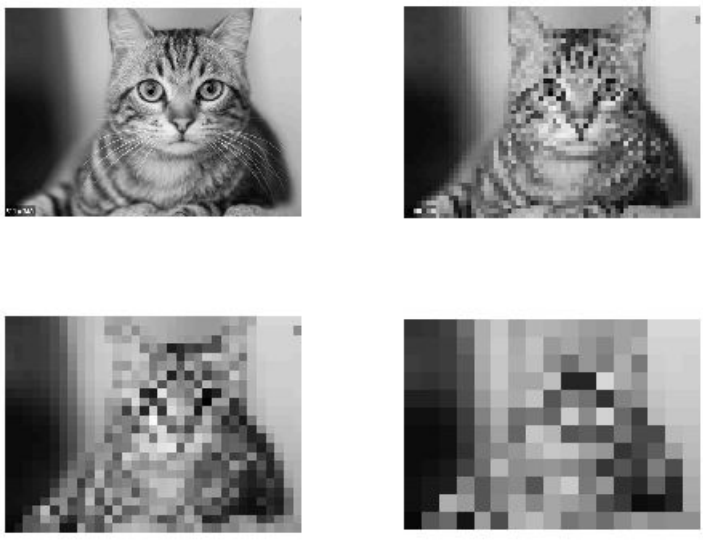
Conversões de imagens Pode ser necessário ao longo do trabalho, converter um tipo de imagem para outro. As funções Octave são

Função	Como usa	Exemplo
`ind2gray`	Converte indexada em cinza	`y=ind2gray(x,map)`
`gray2ind`	Converte cinza em indexada	`[y,map]=gray2ind(x)`
`rgb2gray`	Converte RGB em cinza	`y=rgb2gray(x)`
`gray2rgb`	Converte cinza em RGB	`y=gray2rgb(x)`
`rgb2ind`	Converte RGB em indexada	`[y,map]=rgb2ind(x)`
`ind2rgb`	Converte indexada em RGB	`y=ind2rgb(x,map)`

A produção de um mosaico de uma imagem pode ser assim efetuada

```
clear;
pkg load image
x=imread('gato.jpg');
a=x(:,:,1);  % vermelho ou R  -- podia ser 2=green ou 3=blue
b=imresize(imresize(a,1/8),8,'nearest');
c=imresize(imresize(a,1/16),16,'nearest');
d=imresize(imresize(a,1/32),32,'nearest');
subplot(2,2,1),imshow(a);
subplot(2,2,2),imshow(b);
subplot(2,2,3),imshow(c);
subplot(2,2,4),imshow(d);
```

Com o seguinte resultado



Capítulo 6

Estratégias

PAI: Convoluções (ponto, linha e borda) - 754

Processamento de imagens: melhoria imagens para serem vistas por humanos. Análise de imagens: melhoria e tratamento de imagens para alguma decisão a ser tomada por um computador. Olho humano: esfera aquosa de 2 cm de diâmetro. A luz entra pela iris que mede entre 2 e 8 mm. Na retina há 7 milhões de cones (cor) e 100 milhoes de bastonetes (monocromá tico). O olho opera em ampla faixa na ordem de 10^{10}. (Não ao mesmo tempo, é claro).

Em termos computacionais uma imagem é uma matriz numérica. Cada número é $f(x, y)$ e o resultado de $i(x, y) \times r(x, y)$. i=iluminância e r=refletância. Alguns i: dia ensolarado=900, nublado=100, escritório=10, noite de lua cheia=0,01 lumens/m^2 ou lux. Alguns r: neve=0,93, parede branca=0,80, aço inox=0,65, veludo negro=0,01.

Imagens são formadas por valores no intervalo L_{min} até L_{max}. L_{min} vale 0 e L_{max} em geral é uma potência de 2. Imagens coloridas são formadas por 3 cores primárias aditivas.

Operações Aritméticas. Experimente pensar como ficaria o resultado de:

Para imagens Branco e Preto

1. O complemento para L_{max}: _____.
2. Multiplicação de cada valor por 2: _____.
   E o excesso ? _____.
3. Divisão por k: _____.
4. Limiarização: _____.
5. Arredondamento para a dezena mais próxima
   : _____.
6. Duas imagens somadas pixel a pixel: _____.
7. Subtração de 2 imagens: _____.
8. AND em imagens com 2 niveis de cinza
   : _____.
9. OR idem: _____.

Para imagens color

Os exemplos acima para 1, 2 ou 3 cores ao mesmo tempo. Alguns exemplos:

1. Cor vermelha=0: _____.
2. Cor verde $\times$ 2: _____.
3. Os 3 pixels iguais à média entre os 3 valores
   : _____.
4. A negativa do vermelho combinada com a azul e verde normais
   : _____.
5. A negativa do verde e do azul combinada com anormal do vermelho
   : _____.

6. um degradê vertical na verde combinada com as demais normais
   : _____.
7. um degradê horizontal vermelho combinada com as demais normais
   : _____.
8. degradê vertical verde + degradê horizontal vermelho + azul normal
   : _____.
9. degradê CENTRAL do vermelho
   :_____.
10. limiarização do azul no nivel 100
    : _____.
11. limiarização do azul e do vermelho em 150
    : _____.
12. transformada do seno coxabranca aplicado ao verde
    : _____.
13. Imagine mais algumas operações... (pode batizá-las à vontade)

Operações orientadas à vizinhança

Cada pixel tem o seu valor afetado pelo conteúdo dos pixels seus vizinhos. Por exemplo, imagine uma imagem onde cada pixel é substituido pela média dos seus 8 vizinhos: _____.

Para realizar estas operações em geral é usado o operador CONVOLUdãO. Neste operador uma máscara deslizante "passeia"por todos os pixels da imagem original, dando origem a uma nova imagem (convoluida).

Suponha a seguinte imagem original

a	b	c
d	e	f
g	h	i

Primeiro preenche-se as linhas e colunas externas à imagem com um valor qualquer. Depois, convolui-se a imagem. Se o preenchimento for com 1 a matriz ficará:

1	1	1	1	1
1	a	b	c	1
1	d	e	f	1
1	g	h	i	1
1	1	1	1	1

Se a máscara for, por exemplo

2	2	2
3	3	3
1	1	1

o novo pixel a' será $= a' = 2\times1+2\times1+2\times1+3\times1+a\times3+b\times3+1+d+e\div9$.

e assim, sucessivamente para todos os pixels da imagem original. Note que como não há negativos na máscara, há que se dividir por 9

PAI: Equalização de histograma - 755

Esta técnica está incluída nas técnicas de modificação do histograma. Elas modificam as cores da imagem modificando os valores de "tinta"de cada pixel. Antes de estudá-la precisa-se definir o que é histograma de uma imagem.

Histograma

Histograma de uma imagem é um conjunto de números que indicam quantos pixels apresentam determinado nivel de cinza. Exemplo, seja uma imagem de 4 linhas x 10 colunas e 8 níveis de cinza

0	0	0	0	1	1	2	2	3	2
0	0	0	1	1	2	2	3	3	3
0	0	0	1	2	3	4	5	6	6
0	0	0	1	1	3	3	3	4	1

teria o seguinte histograma:

0	1	2	3	4	5	6	7

As vezes ele é apresentado como um gráfico e as vezes como % do total.

Transformações ponto-a-ponto

As transformações que usam o histograma são chamadas ponto-a-ponto pois só usam o valor original do pixel e não a sua vizinhança.

Equalização de histograma

A transformação mais conhecida é a chamada EQUALIZAÇãO DE HISTOGRAMA que busca distribuir os valores de cada nivel de cinza de maneira a ter um histograma uniforme. Para esta transformação usa-se uma função, que pode ser – por exemplo – a função de distribuição acumulada.

Esta transformação tem a habilidade de aumentar o contraste das imagens forçando o surgimento de detalhes importantes onde antes só havia manchas cinza. Pode também ser usada em imagens coloridas, aplicando-se o método a cada cor individualmente.

Figura 6.1: Imagem original

Figura 6.2: e equalizada

Outras transformações parecidas

Distribuição linear de níveis de cinza

Uma vez obtido o intervalo de variação de níveis de cinza, o mesmo é esticado de maneira a abranger todo o espectro. O histograma muda de figura geométrica. Exemplo: Seja uma imagem 4 x 10 com 8 niveis de cinza (0..7)

0	0	0	0	1	1	2	2	3	2
0	0	0	1	1	2	2	3	3	3
0	0	0	1	2	3	3	3	2	1
0	0	0	1	1	3	3	3	2	1

Neste exemplo, a quantidade de níveis de cinza usados é: _____ Como a variação da imagem é de 8 níveis, pode-se aumentar o salto de cada nível, por um fator _____ melhorando o contraste de maneira linear

Compressão de histograma

Usa o caminho inverso da equalização, e quando aplicada a uma imagem diminui-lhe o contraste.

Limiarização

Muito usada em análise de imagens (quando o computador tem que tomar uma decisão em relação ao que vê). Visa destacar o objeto de seu fundo. Por exemplo, ao examinar a qualidade de fabricação de uma peça, usando a limiarização, um programa poderia decidir sobre a área da peça (em pixels). Note que a técnica é necessária, por pois mais simples que seja a imagem não é possível trabalhar com valores absolutos de cinza (pode estar um dia claro ou um dia nublado...)

Por exemplo, a imagem 4×10, com 8 níveis:

0	0	0	0	1	1	2	2	3	2
0	0	0	1	1	2	2	3	3	3
0	0	0	1	2	3	3	3	2	1
0	0	0	1	1	3	3	3	2	1

Seria convertida na imagem

0	0	0	0	0	0	7	7	7	7
0	0	0	0	0	7	7	7	7	7
0	0	0	0	7	7	7	7	7	0
0	0	0	0	0	7	7	7	7	0

se fosse limiarizada com nivel ≥ 2

Exemplo de equalização de histograma

Para estudar o exemplo, considere a imagem 2×7 com 10 níveis de cinza

1	2	3	2	2	3	1
1	1	3	2	2	1	1

1. Você deve construir o histograma: 0=0; 1=6; 2=5; 3=3; 4=0; 5=0; 6=0; 7=0; 8=0; 9=0
2. Construa a tabela de controle (nn x 6) (nn=niveis de cinza)

   col 1 Escreva todos os valores de cinza (0 a 9)

   col 2 Escreva o resultado do histograma na ordem em que foi construido

   col 3 Escreva (coluna 1) dividido pelo maior nivel de cinza (ou seja 9, neste caso)

   col 4 Escreva (coluna 2) dividido pelo numero de pixels (ou 14)

   col 5 Acumule a (coluna 4)

   col 6 Escreva o indice da (coluna 3) que mais se aproxima da (coluna 5)
3. Finalmente, os pixels que tiverem valor (coluna 1) devem ser reescritos na imagem que esta sendo equalizada com o valor da (coluna 6).

(1) cinza	(2) hist	(3) $col_1 \div 9$	(4) $col_2 \div 14$	(5) col_5 acum	(6) novo valor
0	0	0.000	0.000	0.000	0
1	6	0.111	0.428	0.428	4
2	5	0.222	0.357	0.785	7
3	3	0.333	0.214	0.999	9
4	0	0.444	0.000	0.999	9
5	0	0.556	0.000	0.999	9
6	0	0.667	0.000	0.999	9
7	0	0.778	0.000	0.999	9
8	0	0.889	0.000	0.999	9
9	0	1.000	0.000	0.999	9

4. E com isto, a imagem original ficaria:

4	7	9	7	7	9	4
4	4	9	7	7	4	4

6.1 Algoritmo 1: split-and-merge - 925f

(Horovitz e Pavlidis,1976). Um dos métodos usados para analisar uma imagem é a de determinar quais são as regiões que a compoem. Em certo sentido, esta operação faz a mesma coisa que a operação de encontrar as bordas de uma imagem. Entretanto, devido ao ruído e a ambigüidade pode ser bem interessante usar ambos os métodos para analisar a mesma imagem e com isso diminuir a incerteza nela presente. A relevância desta busca está no raciocínio de que regiões delimitam áreas de interesse dentro da imagem que se analisa.

Define-se uma região dentro de uma imagem como uma porção dela onde não há grande variação de intensidade (ou alguma outra propriedade, como por exemplo a textura). Afirma-se que dentro da região, a propriedade considerada não pode variar abruptamente.

Formalmente falando, uma região dentro de uma imagem é um conjunto de pixels conectados satisfazendo duas propriedades:

1. A região é homogênea. A homogeneidade pode ser definida como a exigência de que a diferença de intensidade dos valores de pixel dentro da região seja inferior a um valor dado, ϵ.
2. Unindo-se duas regiões adjacentes o resultado não satisfará o princípio da homogeneidade acima descrito.

O algoritmo que vai ser estudado aqui (Split-and-Merge devido a Horovitz e Pavlidis em 1976), começa com uma única região candidata: a imagem inteira. Para efeito de ilustração, suponhamos que a imagem seja um quadrado consistindo de 2^l linhas $\times 2^l$ colunas.

Provavelmente a região candidata não vai satisfazer o critério de região, porque o conjunto de pixels não vai obedecer ao princípio da homogeneidade.

Fase SPLIT: Agora, para todas as regiões que não satisfazem a propriedade da homogeneidade, a região é quebrada em 4 partes, através da divisão vertical e horizontal ao meio da região candidata.

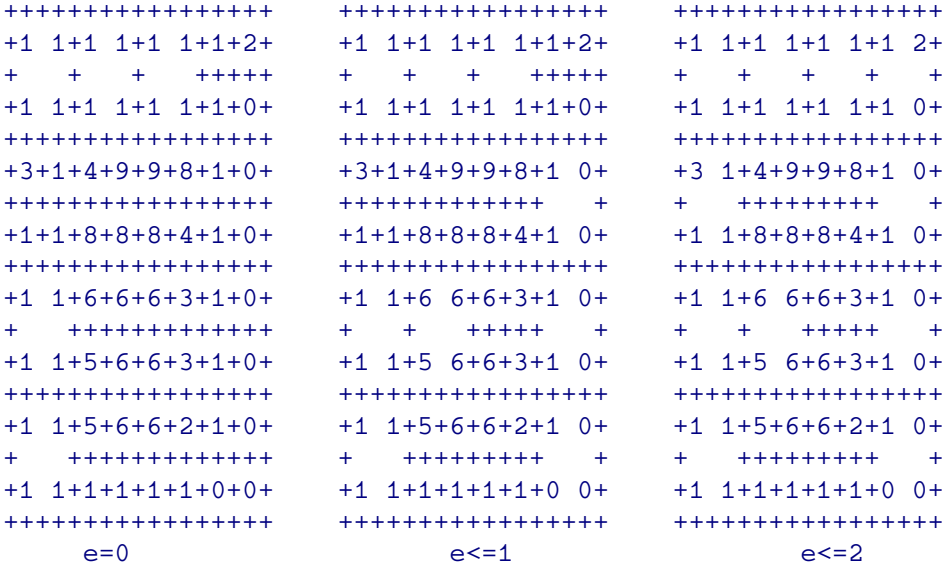
O processo prossegue até que nenhuma divisão adicional precise ser feita.

Fase MERGE: Depois, regiões candidatas que sejam adjacentes são juntadas, desde que seus pixels satisfaçam ao princípio da homogeneidade. As junções podem ser feitas em diferentes ordens, resultando diferentes regiões finais. Pode-se ir juntando antes de chegar ao último corte, mas por simplicidade, pode-se fazer a junção somente após acabar todos os cortes.

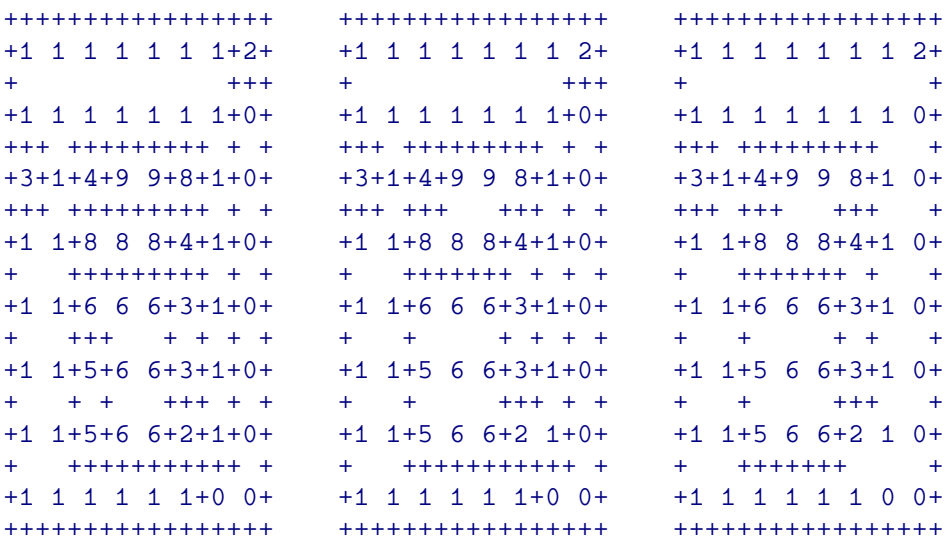
Um exemplo: Seja uma imagem em baixa resolução de 8×8 pixels, em 10 niveis de cinza conforme segue

```
1 1 1 1 1 1 1 2
1 1 1 1 1 1 1 0
3 1 4 9 9 8 1 0
1 1 8 8 8 4 1 0
1 1 6 6 6 3 1 0
1 1 5 6 6 3 1 0
1 1 5 6 6 2 1 0
1 1 1 1 1 1 0 0
```

Aplicando o algoritmo de SPLIT a esta imagem com $\epsilon = 0$, com $\epsilon \leq 1$ e com $\epsilon \leq 2$ fica, respectivamente:



A posterior recomposição das bordas, através do algoritmo de MERGE, fica sendo:



Outro Exemplo Seja uma imagem 8×8 contendo 10 niveis de cinza (entre 0 e 9). Aplique o algoritmo de split-and-merge (Horovitz e Pavlidis, 1976) com $\epsilon = 0$ e ache o comprimento (em cruzes) das divisória central horizontal e da divisória central vertical.

Por exemplo, na imagem,

```
+++++++++++++++++
+1 1 1 1+2+3 3+1+
+ +++   + +++++ +
+1+2+1 1+2+1+2+1+
+++++ +++++++++ +
+2+1 1+2+3+2+4+1+
+++++ +++++ +++ +
+1+3+1 1 1+2+1 1+
```



```
+ +++   +++++ +++
+1 1 1 1+3+1 1+2+
+   +++++++++ +++
+1 1+2 2+1+2+1+3+
+   +++ + +++++++
+1 1 1+2+1 1+2+5+
+     +++++ +++ +
+1 1 1 1+3+1 1+5+
++++++++++++++++
```

os valores procurados são 12 e 16. Isso significa que a linha central tem 12 cruzes e a coluna central 16.

6.2 Algoritmo 2: Bloquinhos de Waltz - 925f

Usaremos aqui imagens de um mundo particular sujeito às seguintes restrições:
* Haverá apenas poliedros em cena (faces planas e arestas retilíneas)
* A iluminação não provocará sombra
* Os poliedros não contém fendas
* Em qualquer ponto nunca se interceptam mais do que 3 superfícies.

Este tipo de cena recebe o nome de *poliedros com vértices triédricos* e no que nos importa, gera imagens compostas apenas de linhas retas.

Na imagem formada, os planos podem se interceptar apenas e somente apenas de 3 maneiras: Um tipo especial de aresta que conecta planos que na realidade não tem conexão física, chamada aresta de oclusão (Winston a chama de linha de contorno). Neste tipo de aresta, apenas um dos planos é totalmente visível, o outro está escondido.

Deve-se marcar este tipo de aresta na imagem usando uma flecha `(->-)` e o sentido da flecha é tal que o plano ocultado está à esquerda da flecha. Winston diz que o objeto contornado deve ficar à direita da flecha, o que vem a ser a mesma coisa.

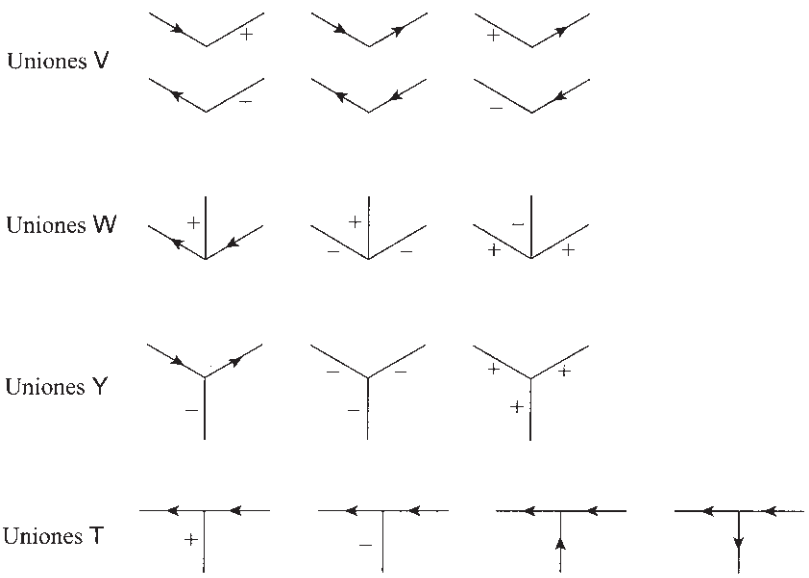
Para as arestas que conectam dois planos com ambos visíveis, existem dois casos: as arestas convexas (identificadas com um sinal de +) ou côncavas, estas identificadas por um sinal de −. Procure lembrar que uma caixa de fósforos sobre a mesa terá arestas convexas enquanto a aresta que conecta a parede e o piso da sala onde estamos é uma arestas côncava.

A questão chave aqui é que o mundo físico dos poliedros impõe certas restrições para que a imagem exista na realidade. Levando estas restrições adiante, em quase todos os casos será possível rotular corretamente todas as arestas da imagem, e consequentemente será possível inferir características físicas da cena representada na imagem.

Os casos excepcionais ficam por conta dos poucos casos em que a ambigüidade presente impedir a correta análise da imagem. Neste caso, novas imagens (com variações sobre o ponto de tomada da imagem) ou mais informações adicionais da cena serão necessárias. Hoffman (Hof00) descreve com bastante profundidade e amplidão a eliminação da ambigüidade através de mudanças de ponto de vista e vale a pena ser consultado.

Antes de começar a marcar as arestas e a propagar seus efeitos, uma etapa necessária é rotular as junções de arestas segundo 4 categorias (Dewdney sugere 5), que recebem o nome de V, W, Y e T. Deve-se ressaltar que TODAS as junções possíveis estão aqui representadas. Impressiona o número baixo de junções possíveis.

Acompanhe o desenho



Depois de ter marcado todas as junções, pode-se passar à segunda fase do algoritmo que busca identificar as arestas. Para identificar as arestas, podem-se usar as seguintes heurísticas:

⋆ Ao examinar a imagem, suporemos que todos os objetos estão suspensos no espaço. Assim cada aresta do fundo do objeto (o contorno) tem apenas arestas flecha.

⋆ Note que há apenas uma junção "W" que tem flechas em suas pontas. Assim, a outra aresta é um "+".

⋆ Há apenas um "Y" que recebe "+" em uma aresta. Ocorrendo isso, todas as demais arestas são também "+".

Finalmente, as restrições vão sendo repassadas adiante, de modo que uma aresta originalmente rotulada em uma extremidade, deve obviamente apresentar o mesmo rótulo na outra extremidade. Afinal, são objetos físicos.

6.3 Operadores Morfológicos - 926

6.3.1 Erosão Binária

Seja uma imagem binária I formada por uma grade retangular de $m \times n$ pixels. Seja um elemento estruturante E também retangular formado por $k \times l$ pixels. A *erosão* da imagem I pelo elemento E é uma nova imagem obtida deslizando-se o elemento estruturante pela imagem original, e tomando apenas os pixels da imagem original que quando colocados no centro do elemento estruturante garantiram a inclusão de todos os pixels significativos do elemento estruturante dentro da imagem original. Seja o exemplo: Supondo a imagem binária, onde o zero está representado por um "."e o um está representado por um "X".

$$I = \begin{bmatrix} . & . & . & . & . & . \\ . & X & X & X & . & . \\ . & X & X & X & X & . \\ . & . & X & X & X & . \\ . & . & . & X & X & . \\ . & . & . & X & X & . \\ . & . & . & . & . & . \end{bmatrix}$$

E seja o elemento estruturante 3×3 assim formado

$$E = \begin{bmatrix} . & X & . \\ . & (X) & . \\ . & X & . \end{bmatrix}$$

O elemento central da máscara, reconhecido por um "()" é o elemento principal.

O elemento estruturante deve percorrer todos os pontos da imagem original, de maneira a que o elemento principal do mesmo (usualmente o elemento que está no centro) coincida com todos os elementos da imagem original.

Em cada um desses momentos, (o elemento central da máscara estruturante posicionado sobre um determinado pixel da imagem original), deve-se fazer a seguinte pergunta: Existe ALGUM pixel 1 da máscara estruturante sobre um pixel valendo 0 da imagem ?
Se a resposta for SIM, esta posição correspondente na saída (a imagem erodida) deve ser 0, senão ela será 1.

Por exemplo, no caso acima, quando o elemento principal (o $2,2$ da máscara) está sobre a posição $1,1$ da imagem, existem pelo menos um (na verdade, 3) pixels 1 da máscara sobre pontos que não são 1 na imagem. Por isso, a posição $1,1$ da imagem erodida será zero. Seguindo adiante, a primeira posição valendo 1 na saída corresponde ao pixel $3,3$ da entrada.

A imagem erodida, fica

$$E \textbf{ ero } I = \begin{bmatrix} . & . & . & . & . & . \\ . & . & . & . & . & . \\ . & . & X & X & . & . \\ . & . & . & X & X & . \\ . & . & . & X & X & . \\ . & . & . & . & . & . \\ . & . & . & . & . & . \end{bmatrix}$$

Os efeitos da operação de erosão podem ser assim descritos

- Diminui as partículas da imagem
- Elimina grãos de tamanho menor do que o tamanho do elemento estruturante
- Aumenta os buracos da imagem
- Permite a separação de grãos próximos.

Transformação hit-miss

Para realizar esta operação precisamos de dois elementos estruturantes E^i e E^e que formam um conjunto para a aplicação de hit-miss. Define-se o conjunto V como $V = (E^i, E^e)$ e neste, ambos os subconjuntos devem ser disjuntos, senão a

transformação não está definida. O conjunto E^i será usado para testar a parte interna da imagem e o conjunto E^e será usado para a parte externa, ou melhor dizendo a imagem complementar à imagem interna. Com isto, temos a seguinte definição:
A transformação **him** sobre o conjunto X a partir dos elementos estruturantes $V = (E^i, E^e)$ é $\mathbf{him}^v(X) = X$ **him** $V = \{x : E^i_x \subset X; E^e_x \subset X^c\}$

A partir desta definição, um ponto de X pertence a X **him** V se e somente se E^i "cabe" em X e E^e "cabe" em X^c.

Uma definição alternativa da transformação hit-miss, usando a definição da erosão pode ser: E **him** $I = (E^i$ **ero** $I) \cap (E^e$ **ero** $I^c)$, onde I^c é a imagem complementar de I (o negativo...).

Para poder trabalhar esta operação morfológica, precisamos definir uma nova máscara formada por 3 símbolos:

o Representa os pixels que formam o conjunto E^e

X Representa os pixels que formam o conjunto E^i

? Representa os pixels irrelevantes, que não fazem parte de nenhum dos conjuntos E^e ou E^i.

6.3.2 Afinamento

Uma característica importantes das transformações é o **homotopismo**. Uma transformação é homotópica quando ela não altera a quantidade de componentes de uma imagem. (Por componente, aqui entende-se a parte claramente isolada do fundo). A operação de afinamento é uma transformação homotópica na qual os componentes vão tendo a sua espessura reduzida até um número pequeno (usualmente 1 pixel), sem que haja mudança no número ou tipo de componentes. A definição de afinamento usa a transformada hit-miss. X **afi** $V = X$ / $(V$ *him* $X)$ onde a operação X / Y deve ser entendida como $X \cap Y^c$. O processo completo de afinar consiste em iterar a imagem com um ou vários pares de de elementos estruturantes até não haver mais modificações no resultado final. Afinar é um processo homotópico e portanto ele não altera a propriedade da conectividade. Quando o resultado final é alcançado a transformação vira idempotente e portanto continuar afinando não alterará o resultado alcançado.

A escolha do par estruturante é importante. A literatura (vide, por exemplo, FAC96, pág. 72) apresenta diversas sugestões de pares para afinar. Esta não é uma escolha trivial e os parâmetros para ela ainda não são completamente dominados.

Vejamos um exemplo de afinamento, na forma de uma matriz a_0, com a máscara m_a

```
ma = 0 1 1
     2 1 1
     2 2 0
```

Para a construção da máscara acima, usou-se uma codificação especial. A intenção é colocar na mesma máscara 2 conjuntos distintos, denominados de B^i e B^e, respectivamente os estruturantes internos e externos da imagem. Como ambos são conjuntos disjuntos isso pode ser feito. Usando a referência vista acima, temos:

conjunto	visto acima como	valor aqui
B^i	x	1
B^e	o	2
tanto faz	?	0

6.3.3 Esqueletização

A esqueletização é apenas a automação do procedimento de *afinação* visto acima, até que nenhum pixel a mais seja retirado da imagem anterior.

6.3.4 Algoritmos

erosão Trivial, conforme explicado acima.

hitmiss Aplica a transformada hit-miss

1: matriz função hitmiss (máscara, imagem)
2: m1 ← máscara = 1
3: m2 ← máscara = 2
4: devolve (m1 erosão imagem) ∧ (m2 erosão ∼ imagem)

afinamento Aplica o operador afinamento

1: matriz funç ao afinamento (máscara, imagem)
2: temporário ← máscara him imagem
3: devolve imagem ∧ ∼ temporário

esqueletização Aplica a esqueletização a uma imagem

```
1: matriz função esquelet (máscara, imagem)
2: ct1 ← quantidade de uns em imagem
3: ct2 ← 9999999
4: while ct1 ≠ ct2 do
5:    imtemp ← afinamento (máscara, imagem)
6:    ct2 ← quantidade de uns em imtemp
7:    imagem ← imtemp
8: end while
9: devolva imagem
```

6.4 Visão computacional - 928

Depois que a imagem já foi processada usando técnicas vistas em aulas passadas, pode-se passar à fase de análise da imagem, na qual características da cena são extraídas e usadas para a tomada de decisão. Usualmente, como uma única imagem pode corresponder à diversas cenas (base de todas as ilusões de ótica), usualmente são necessárias ou imagens adicionais da mesma cena, ou informações não visuais adicionais, ou ambas as coisas.

Este conhecimento pode ser específico ou genérico e pode assumir inúmeras formas, não estando limitado de nenhuma maneira. (O que está na cena, onde estão as fontes de iluminação, quais as fontes de ruído e o seu valor, a cena é estática ou dinâmica, os entes são autônomos ou não e de que tipo, objetos regulares, irregulares, côncavos, convexos, lineares, cônicos, cores, brilhância, refletância (albedo), existência de texels, etc etc. Aqui vale sempre a regra: quanto mais genérico for o dispositivo analisador, maior sera a demanda por informações adicionais. O reverso é verdade: quanto mais específico for o analisador menos informação é necessária.

Na abordagem deste exercicio, usa-se a segunda estratégia, considerando características do mundo físico que limitam as possíveis coisas que estão em cena. Usaremos aqui imagens de um mundo particular sujeito às seguintes restrições:

- Haverá apenas poliedros em cena (faces planas e arestas retilíneas)
- A iluminação não provocará sombra
- os poliedros não contém fendas
- em qualquer ponto nunca se interceptam mais do que 3 suerfícies.

Este tipo de cena recebe o nome de *poliedros com vértices triédricos* e no que nos importa, gera imagens compostas apenas de linhas retas.

Na imagem formada, os planos podem se interceptar apenas e somente apenas de 3 maneiras: Um tipo especial de aresta que conecta planos que na realidade não tem conexão física, chamada aresta de oclusão (Winston a chama de linha de contorno). Neste tipo de aresta, apenas um dos planos é totalmente visível, o outro está escondido.

Deve-se marcar este tipo de aresta na imagem usando uma flecha `(->-)` e o sentido da flecha é tal que o plano ocultado está à esquerda da flecha. Winston diz que o objeto contornado deve ficar à direita da flecha, o que vem a ser a mesma coisa.

Para as arestas que conectam dois planos com ambos visíveis, existem dois casos: as arestas convexas (identificadas com um sinal de $+$) ou côncavas, estas identificadas por um sinal de $-$. Procure lembrar que uma caixa de fósforos sobre a mesa terá arestas convexas enquanto a aresta que conecta a parede e o piso da sala onde estamos é uma arestas côncava.

A questão chave aqui é que o mundo físico dos poliedros impõe certas restrições para que a imagem exista na realidade. Levando estas restrições adiante, em quase todos os casos será possível rotular corretamente todas as arestas da imagem, e consequentemente será possível inferir características físicas da cena representada na imagem.

Os casos excepcionais ficam por conta dos poucos casos em que a ambigüidade presente impedir a correta análise da imagem. Neste caso, novas imagens (com variações sobre o ponto de tomada da imagem) ou mais informações adicionais da cena serão necessárias. Hoffman (Hof00) descreve com bastante profundidade e amplidão a eliminação da ambigüidade através de mudanças de ponto de vista e vale a pena ser consultado.

Antes de começar a marcar as arestas e a propagar seus efeitos, uma etapa necessária é rotular as junções de arestas segundo 4 categorias (Dewdney sugere 5), que recebem o nome de V, W, Y e T. Deve-se ressaltar que TODAS as junções possíveis estão aqui representadas. Impressiona o número baixo de junções possíveis.

Acompanhe o desenho

Depois de ter marcado todas as junções, pode-se passar à segunda fase do algoritmo que busca identificar as arestas. Para identificar as arestas, podem-se usar as seguintes heurísticas:

- Ao examinar a imagem, suporemos que todos os objetos estão suspensos no espaço. Assim cada aresta do fundo do objeto (o contorno) tem apenas arestas flecha.
- Note que há apenas uma junção “W” que tem flechas em suas pontas. Assim, a outra aresta é um “+”.

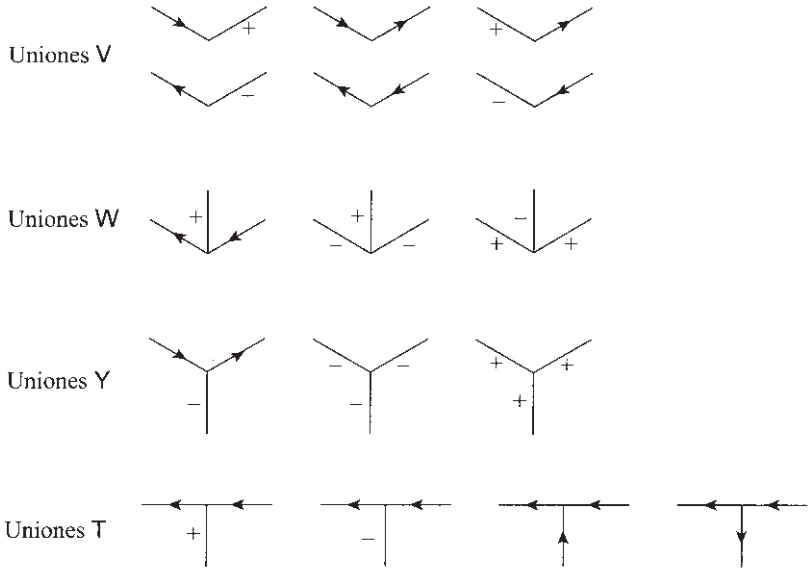


Figura 6.3: Quatro tipos de junções

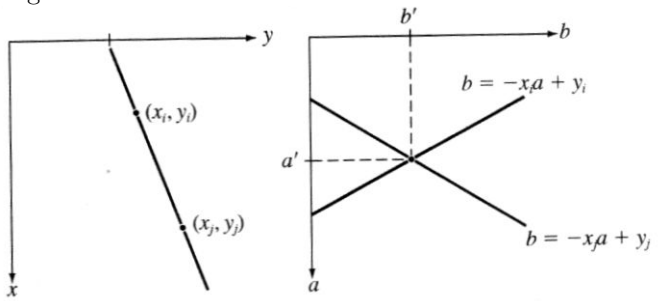
- Há apenas um "Y" que recebe "+" em uma aresta. Ocorrendo isso, todas as demais arestas são também "+".

  Finalmente, as restrições vão sendo repassadas adiante, de modo que uma aresta originalmente rotulada em uma extremidade, deve obviamente apresentar o mesmo rótulo na outra extremidade. Afinal, são objetos físicos.

6.5 Transformada de Hough - 952

Suponha uma imagem com n pontos. Por alguma razão precisa-se encontrar subconjuntos desses pontos que pertençam a uma linha reta dentro da imagem. Uma possível solução (força bruta) é encontrar primeiro todas as retas determinadas por pares de pontos e em seguida encontrar todos os subconjuntos de pontos que estejam próximos a essas retas em particular. Este método envolve procurar $\sim n^2$ retas e executar $\sim n^3$ comparações de pontos para as retas. A tarefa é computacionalmente proibitiva para qualquer aplicação com exceção das triviais ou *toy domains*.

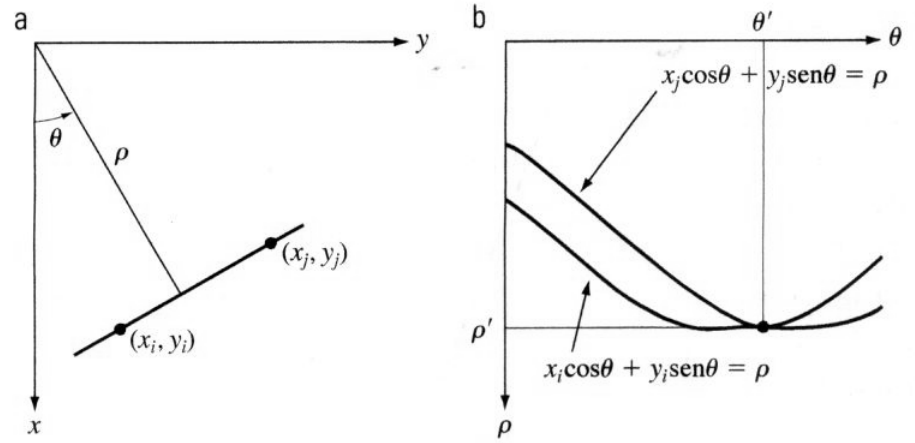
Hough propôs em 1962 um método que acabou sendo chamado de *Transformada de Hough*. Considere um ponto (x_i, y_i) no plano xy e a equação geral de uma reta $y_i = ax_i + b$. Muitas retas passam por (x_i, y_i), mas todas satisfazem a $y_i = ax_i + b$ para diferentes valores de a e b. Ao escrever esta equação como $b = -x_i a + y_i$ no plano ab (chamado espaço de parâmetros) produz a equação de uma única reta para o par fixo (x_i, y_i). Um segundo ponto (x_j, y_j) também tem uma reta no espaço de parâmetros associado a ele e, a menos que sejam paralelas, esta reta cruza a reta associada à (x_i, y_i) em algum ponto a', b' em que a' é a inclinação e b' a intersecção da reta tanto contendo (x_i, y_i) quanto (x_j, y_j) no plano xy. Na verdade todos os pontos nesta reta pertencem a retas no espaço de parâmetros que se cruzam em (a', b'). As figuras abaixo ilustram este fato

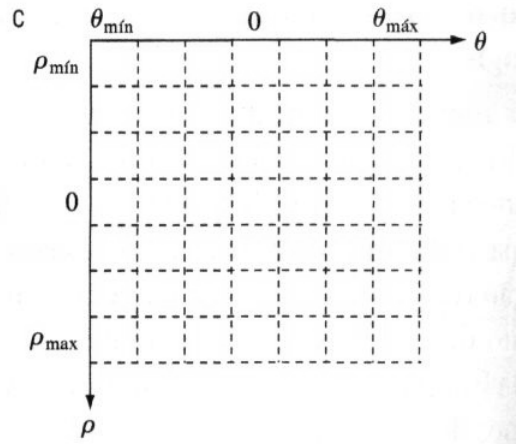


Uma dificuldade prática com esta abordagem é que a inclinação de uma reta se aproxima de ∞ conforme a reta se aproxima da vertical. Para contornar essa dificuldade usa-se a representação normal de uma reta em coordenadas polares $x\cos\theta + y\sin\theta = \rho$. A figura a abaixo ilustra a interpretação geométrica dos parâmetros ρ e θ. Uma reta horizontal tem $\theta = 0^o$ com ρ igual à intersecção positiva de x. Cada curva senoidal da figura b representa a família de retas que passam por um ponto (x_k, y_k) no plano xy. O ponto de intersecção (ρ', θ') na figura b corresponde à reta que passa tanto por (x_i, y_i) quanto por (x_j, y_j) na figura a.

A atratividade computacional da transformada de Hough surge da subdivisão do espaço de parâmetros $\rho\theta$ nas chamadas células acumuladoras como a figura c ilustra sendo que (ρ_{min}, ρ_{max}) e $(\theta_{min}, \theta_{max})$ são os esperados intervalos de valores dos parâmetros $-90^o \leq \theta \leq 90^o$ e $-D \leq \rho \leq D$, em que D é a distância máxima entre os cantos opostos em uma

imagem.





Propriedades Definindo o espaço da imagem (xy) e o espaço de parâmetros ($\theta\rho$) tem-se as seguintes propriedades:

1. Um ponto no espaço da imagem corresponde a uma senóide no espaço de parâmetros
2. Um ponto no espaço de parâmetros corresponde a uma reta no espaço da imagem
3. Pontos que caem na mesma reta em xy correspondem a curvas com o mesmo ponto em comum no espaço $\theta\rho$
4. Pontos que caem na mesma curva em $\theta\rho$ correspondem a retas que passam por um ponto em xy

Definindo o arranjo acumulado, como resultado final da aplicação da transformada, tem-se

- Células com valores mais altos correspondem a pontos colineares na imagem
- A equação da reta na imagem é

$$\rho = x\cos\theta + y\sin\theta$$

- O valor da célula representa o número de pontos na imagem que pertencem àquela reta.

Um exemplo numérico Seja a imagem binária

```
1 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 0 0 0 0 0 1
```

Após a aplicação da transformada de Hough

```
 0 -90 -75 -60 -45 -30 -15  0 15 30 45 60 75 90
-5   0   0   0   0   0   0  0  0  0  0  0  0  0
```



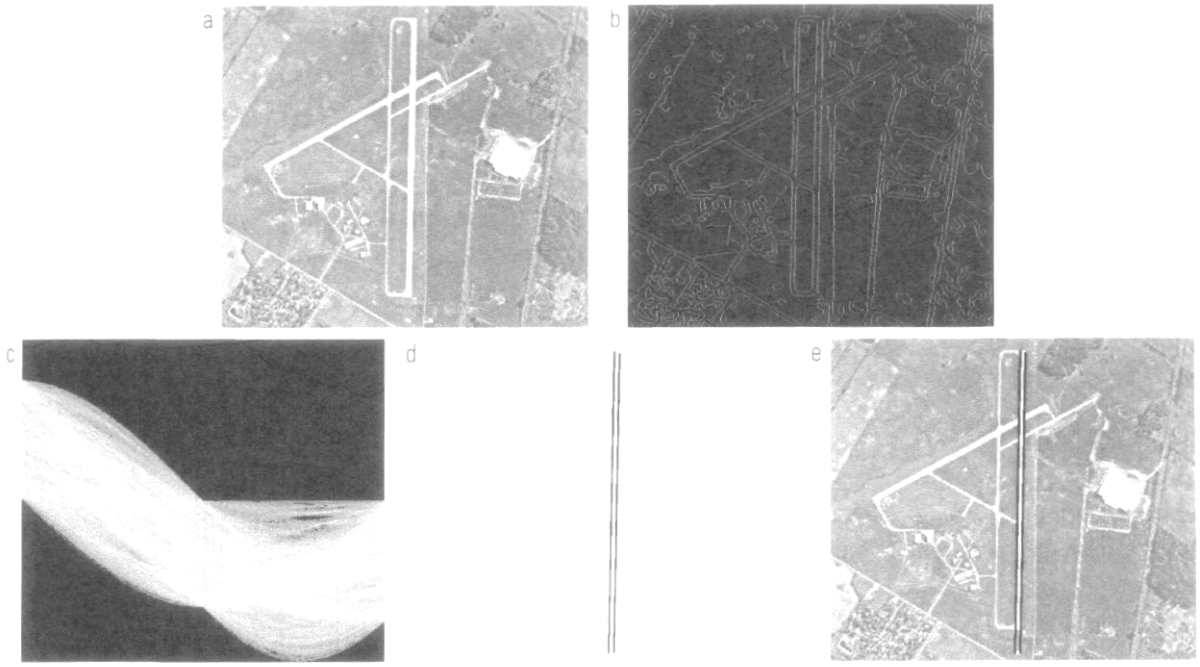
```
-4   2   1   1   0   0   0   0   0   0   0   0   0   0
-3   1   1   0   1   1   0   0   0   0   0   0   0   0
-2   0   1   2   0   0   0   0   0   0   0   0   0   0
-1   2   1   0   0   0   1   0   0   0   0   0   0   0
 0   0   0   1   3   1   0   0   0   0   0   0   0   0
 1   0   1   0   0   0   1   2   1   1   1   1   1   2
 2   0   0   0   0   2   1   0   1   0   0   0   1   0
 3   0   0   1   1   0   1   1   1   1   0   1   1   1
 4   0   0   0   0   1   1   2   1   2   3   2   1   2
 5   0   0   0   0   0   0   0   1   1   1   1   1   0
```

ou na forma de senóides

```
211
11 11
 12
21   1
  131
 1   12111112
    21 1   1
  11 1111 111
    112123212
       11111
```

Numerando os pontos na imagem como 1...2, ...3..., 4...5, no resultado da transformada o ponto valendo 3 na linha 0, coluna -45 corresponde à reta que contém os pontos 2, 3 e 4 no plano xy. Estes 3 pontos estão em uma reta que passa pela origem ($\rho = 0$) e inclinação -45. Já o segundo 3, na linha 4, coluna 45, representa a reta formada por 1, 3 e 5.

Uma aplicação prática A figura abaixo mostra uma imagem aérea de um aeroporto:



A imagem a é a obtida pela câmera. A b é a mesma imagem devidamente limiarizada. A c é a representação da aplicação da transformada de Hough. A d é o retorno das linhas campeãs no valor numérico da transformada. E, finalmente a e reforça na imagem original a pista de aterrisagem procurada.

Generalização Qualquer curva que possa ser representada por sua equação parametrizada pode ser tratada através da transformada de Hough, como por exemplo:
Uma circunferência de equação

$$(x-a)^2 + (y-b)^2 = r^2$$

pode ser traduzida em um arranjo acumulador $A(a, b, r)$.
Uma elipse de equação

$$\frac{(x-x_0)^2}{a^2} + \frac{(y-y_0)^2}{b^2} = 1$$

pode ser levada a um arranjo $A(a, b, x_0, y_o)$ e assim por diante.

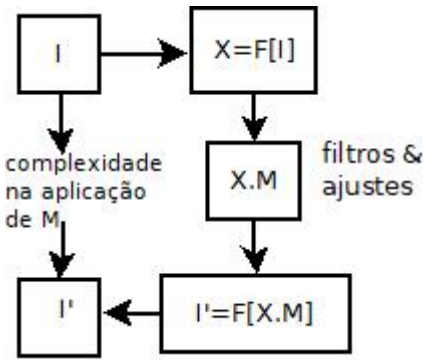
O código Python

```
import numpy as np
import math as ma
def aplhough(x,d):
  ra=ma.ceil((2*len(x))**0.5)
  ac=np.zeros(((2+2*ra),(2+ma.floor(180/d))),int)
  for i in range(len(x)):
    for j in range(len(x[0])):
      if x[i][j]!=0:
        for theta in range(-90,91,d):
          rr=((j+1)*ma.cos((theta*ma.pi)/180))+ \
             ((i+1)*ma.sin((theta*ma.pi)/180))
          rr=int(np.sign(rr)*(np.abs(rr*2))**0.5)
          col=len(ac[0])-ma.ceil((91-theta)/d)
          ac[ra+rr][col]=ac[ra+rr][col]+1
  theta=-90
  for j in range(len(ac[0])-1):
    ac[0][j+1]=theta
    theta=theta+d
  lim=-ra
  for i in range(len(ac)-1):
    ac[i+1][0]=int(np.sign(lim)*((lim**2)/2))
    lim=lim+1

  return ac
ima3=...obtem a imagem limiarizada
h=aplhough(ima3,5)
print(h)
```

6.6 Transformada de Fourier - 953

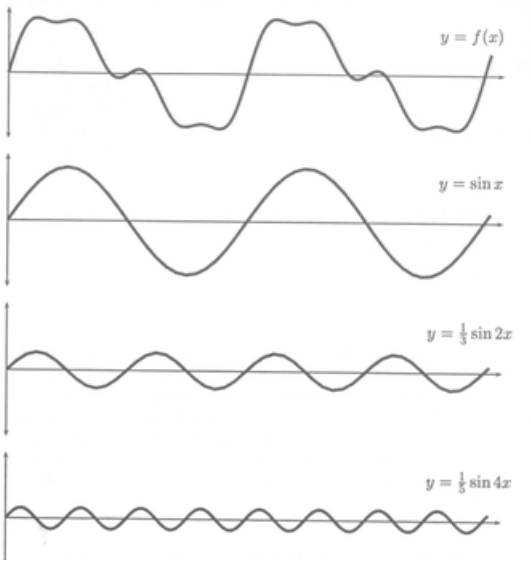
Como vimos, no processamento de imagens digitais é muitas vezes necessário destacar bordas ou ao contrário suavizar mudanças abruptas, como por exemplo quando se busca eliminar o ruido sal&pimenta. Entra em ação aqui a Transformada de Fourier, por diversas razões. Uma delas é a eficiência: ao aplicar a convolução de um filtro de 32×32 sobre uma imagem de 512×512 seriam necessárias 268 M multiplicações. O mesmo filtro aplicado sobre a transformada de Fourier da imagem e sua posterior transformada inversa demanda apenas 14 M multiplicações, ou apenas 5% do esforço computacional, que é muito menor se a imagem for maior... Outra razão importante para usar a TF é que ela destaca a frequência da imagem, e portanto facilita o manuseio dessa componente da imagem.



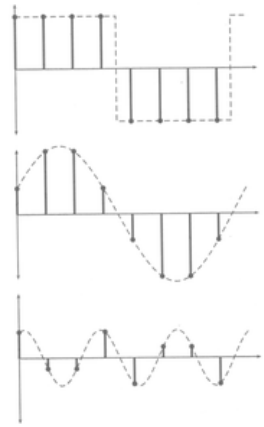
Jean Jacques Fourier foi um cientista francês que acompanhou Napoleão na sua estada no Egito e entre outras descobertas descreveu o calor que é entidade fundamental e importante na física: está presente na Lei Fundamental da Termodinâmica: "A entropia do universo é crescente". A descoberta que nos interessa aqui é que qualquer sinal periódico pode ser decomposto em séries de senos e cossenos escalados e somados. A boa notícia é que o processo inverso sempre é possível e quando executado não acarreta nenhuma perda de informação do sinal original. Veja por exemplo a função

$$f(x) = \sin x + \frac{1}{3}\sin 2x + \frac{1}{5}\sin 4x$$

com as somas indicadas



Este sinal digital pode ser obtido como uma soma de apenas duas senoides



A Transformada discreta de Fourier (DFT) pode ser expressa: Supondo $f = [f_0, f_1, f_2, ..., f_{N-1}]$ como uma sequência de comprimento N, a *Transformada Discreta de Fourier* como a sequência $F = [F_0, F_1, F_2, ..., F_{N-1}]$ onde

$$F_u = \frac{1}{N} \sum_{x=0}^{N-1} e^{-2\pi j \frac{xu}{N} fx}$$

e a transformada inversa de Fourier tem aspecto semelhante

$$x_u = \sum_{x=0}^{N-1} e^{2\pi j \frac{xu}{N} Fu}$$

Comparando ambas, percebe-se apenas 2 diferenças: i. Na inversa não há o fator de escala $1/N$ e ii. O sinal dentro da exponencial, muda. Em Python, eis o cálculo da DTF

```
import numpy as np
def DFT_slow(x):
  """Calcula a transformada de fourier
    para um vetor 1D """
  x = np.asarray(x, dtype=float)
  N = x.shape[0]
  n = np.arange(N)
  k = n.reshape((N, 1))
  M = np.exp(-2j * np.pi * k * n / N)
  return np.dot(M, x)

DFT_slow(0, 1, 2, 3, 4, 5, 6, 7)
 28
 -4J9.656854249
 -4J4
```

```
-4J1.656854249
-4
-4J-1.656854249
-4J-4
-4J-9.656854249
```

O procedimento acima, embora correto, pode demorar uma eternidade no cálculo de DFT para vetores (ou matrizes) de grande tamanho. Na área de imagens esta questão é premente.

Fast Fourier Transform Estamos no ano de 1965. O Presidente Lindon Johnson dos EUA tinha um comitê de aconselhamento científico. Dele faziam parte dois cientistas, Richard Garwin e John Tuckey. O primeiro precisava desesperadamente calcular muitas transformadas de Fourier com rapidez, e o segundo estava calmamente resolvendo, com papel e lápis, algumas T.F. cabeludas.

Garwin, chegou, viu e se interessou pelo assunto. Ele questionou Tuckey sobre o que este estava fazendo. Tuckey mostrou sua nova abordagem para resolver T.F.

Garwin era da área de informática, voltou logo ao centro de pesquisa da IBM em Yorktown Heighs para ter a técnica de Tuckey implementada em computador.

Achou um programador recém-contratado, (um estagiário !) que perambulava sem nada importante para fazer, e entregou a encomenda para ele. Seu nome era James Cooley e ele fez e implementou o programa. Segundo suas palavras *"...achei que o projeto seria esquecido, e eu mesmo logo o esqueci"*.

Mas, aconteceu o contrário. Começaram a chegar requisições para obter cópias do programa, a comunidade acadêmica ficou em polvorosa, e a seguir, em 1965, Cooley e Tuckey publicaram o hoje famoso *"An algorithm for the machine calculation of complex Fourier series."*

Estava criado o algoritmo Cooley/Tuckey, também conhecido como FFT, de fast fourier transform. Com ele o custo computacional de calcular a transformada de Fourier, caiu de n^2 para $n.log_2 n$.

```
def FFT(x):
  """Implementacao recursiva da FFT 1D
    de Cooley-Tukey """
  x = np.asarray(x, dtype=float)
  N = x.shape[0]
  if N % 2 > 0:
    raise ValueError("tamanho deve ser 2^n")
  elif N <= 32:  # limite de recursividade
    return DFT_slow(x)
  else:
    Xpar = FFT(x[::2])
    Ximpar = FFT(x[1::2])
    fator = np.exp(-2j*np.pi*np.arange(N)/N)
    return np.concatenate
      ([Xpar+fator[:N//2]*Ximpar,
      Xpar + fator[N//2:]*Ximpar])
```

Note que a transformada recebe um vetor de reais e devolve um vetor de complexos. Quando se trata de imagens, não se pode pedir para visualizar um conjunto de complexos. Mais ainda, a visualização fica prejudicada pela força do elemento (0,0) – o primeiro – que como se viu vale a soma de todos os elementos do vetor. Por isso, costuma-se fazer 3 coisas para visualizar a transformada:

i. Em $f = fft(x)$, mostrar $abs(f)$ ao invés de f, já que este procedimento transforma um complexo em real.

ii. Mostrar $1 + log(f)$ ao invés de f, já que isto atenua a importância do elemento 0,0 sobre os demais e

iii. $fftshift(f)$ ao invés de f. Esta função auxiliar simplesmente centraliza o elemento 0,0 em vez de deixá-lo no canto superior esquerdo.

A questão agora é como passar da FFT de um vetor para a FFT de uma imagem. A resposta vem de uma característica de independência (ou separabilidade) da FFT. Para obter a transformada de uma matriz (imagem) pode-se calcular primeiro a DFT das linhas e depois calcular a DFT das colunas e depois multiplicar o resultado.

Matlab ou seu primo livre: Octave Ambientes fantásticos para visualizar imagens e suas transformadas

```
x = imread(arquivo de imagem)
y = fft(x)
xl = ifft(y)
y = fft2(x)
```



```
xl = ifft2(y)
z = fftshift(x)
imshow(y)
```

Teorema da Convolução Dadas a imagem S e uma máscara (filtro) M, e a Transformada de Fourier F, tem-se

$$S \otimes M = F^{-1}[F(S).F(M)]$$

Graças a este teorema transforma-se uma convolução (passear pela máscara sobre todos os pixels da imagem) em uma multiplicação elemento a elemento simples sobre a imagem. Em Octave:

```
a = imread(...);
cf = fftshift(fft2(a));
c = ...mascara (mesmo tamanho que a valendo 1/0)
cfl = cf.*c (multiplic. elemento a elemento)
cfli = ifft2(cfl)
imshow(cfli)
```

Lembrando que c pode ser passa-alta, passa-baixa, gaussiano, remoção de ruído periódico, remoção de movimento, etc. Para saber mais: McAndrew, alasdair. **Introduction to Digital Images with Matlab**. Wellington, Victoria University, s.d. Disponível na Internet em https://www.pdfdrive.com/ ou no pendrive do instrutor.

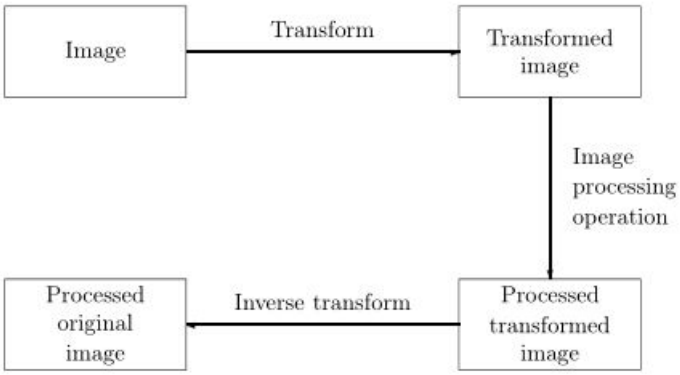
Capítulo 7

Alguns Tópicos Interessantes

7.1 Capítulo 5 de [Mca00] - Processamento de pontos

Qualquer operação de processamento de imagens transforma o valor do cinza de cada pixel. Entretanto operações deste tipo podem ser divididas em 3 classes baseado na informação requerida para executar a transformação. Do mais complexo para o mais simples, tem-se

Transformadas Requer-se um conhecimento de todos os níveis de cinza de toda a imagem para obter a transformada. Em outras palavras a imagem inteira é processada como um único largo bloco. Isto pode ser visto em



Filtros Espaciais para trocar o nível de cinza de um determinado pixel, só se precisa saber o nível da vizinhança do pixel em questão.

Operações Pontuais o valor do pixel é trocado sem conhecer nenhum outro valor..

Ainda que as operações pontuais sejam as mais simples, elas contém algumas das mais poderosas e largamente usadas de todas as operações de imagens. Isto é especialmente verdade no *pré-processamento* de imagens, onde uma imagem é processada, antes de sua execução principal.

7.1.1 Operações Aritméticas

Estas operações são a aplicação de uma função $y = f(x)$ aplicada a cada valor de cinza da imagem. Uma possibilidade de função é adicionar ou subtrair uma constante a cada pixel ou $y = x \pm C$, ou então multiplicar cada pixel por uma constante ou $y = Cx$. Em qualquer caso deve-se garantir que o resultado esteja no intervalo 0..255, aplicando-se neste caso a definição

$$y \leftarrow \begin{cases} 255 & se\ y > 255 \\ 0 & se\ y < 0 \\ y & senao \end{cases}$$

Para organizar este fato, o Octave tem as funções `imadd` e `imsubtract` que limitam o resultado a 0..255. Veja nos exemplos

```
>> pkg load image
>> a = imread('d:/imas/cidadec.jpg');
>> b = imadd(a,100);
>> c = imsubtract(a,100);
```

Com o seguinte resultado

original

soma de 100

subtração de 100

Veja-se agora as aplicações de multiplicação e divisão

```
>> pkg load image
>> a = imread('d:/imas/cicadec.jpg);
>> d = immultiply(a,2);
>> e = imdivide(a,2);
```

com o resultado (a imagem original é a mesma de cima, `cidadec.jpg`).

multiplicação por 2

divisão por 2

7.1.2 Negativo

Vejamos agora como obter o negativo de uma foto. A função é `imcomplement`.

```
>> f = imcomplement(a)
```

complemento (negativo)

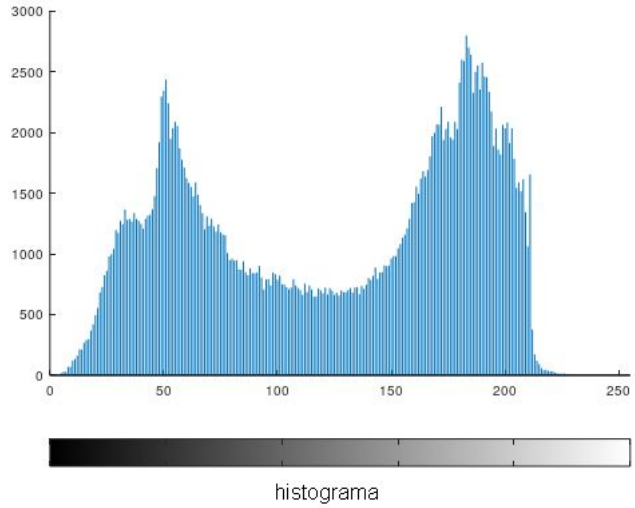
7.1.3 Histograma

Agora vamos estudar o conceito de histograma de uma imagem. É um gráfico indicando a quantidade de vezes que um dado nível de cinza aparece na imagem. Pode-se inferir várias coisas de uma imagem simplesmente olhando o seu histograma como em

- Em uma imagem escura, os níveis de cinza estarão concentrados na parte baixa do eixo
- Em uma imagem clara, os níveis estarão concentrados na parte alta
- Em imagem bem constrastada os níveis estarão bem distribuídos sobre um largo trecho do eixo

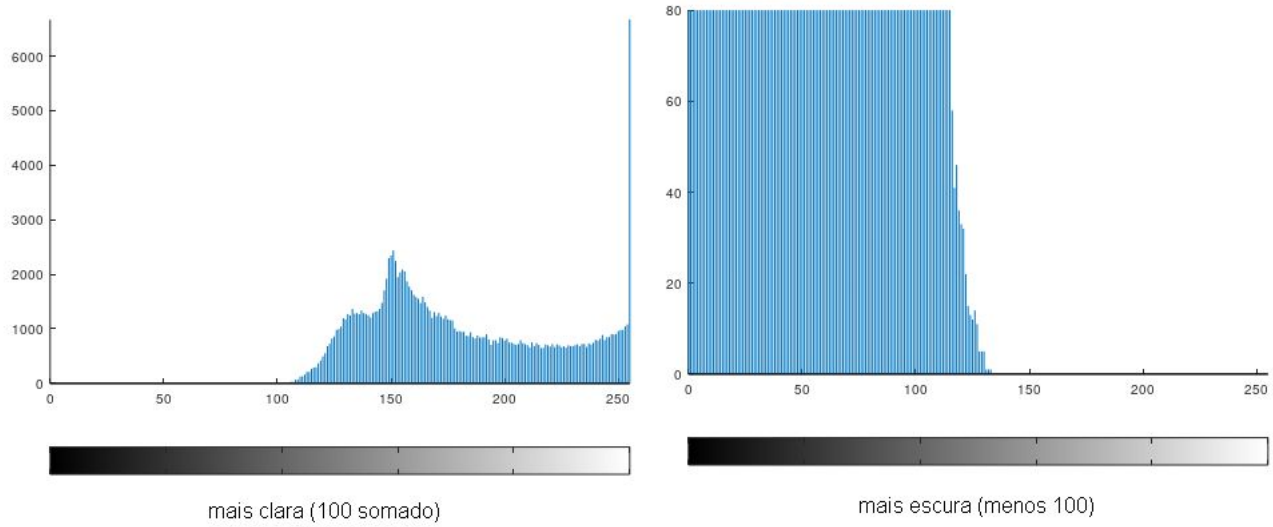
Para olhar o histograma, usa-se a função `imhist` como em

```
>> imhist(a)
```



histograma

Para ter idéia do comportamento do histograma, compare com os histogramas das imagens acima(aquelas que tiveram 100 somados (mais clara) e subtraídos (mais escura)

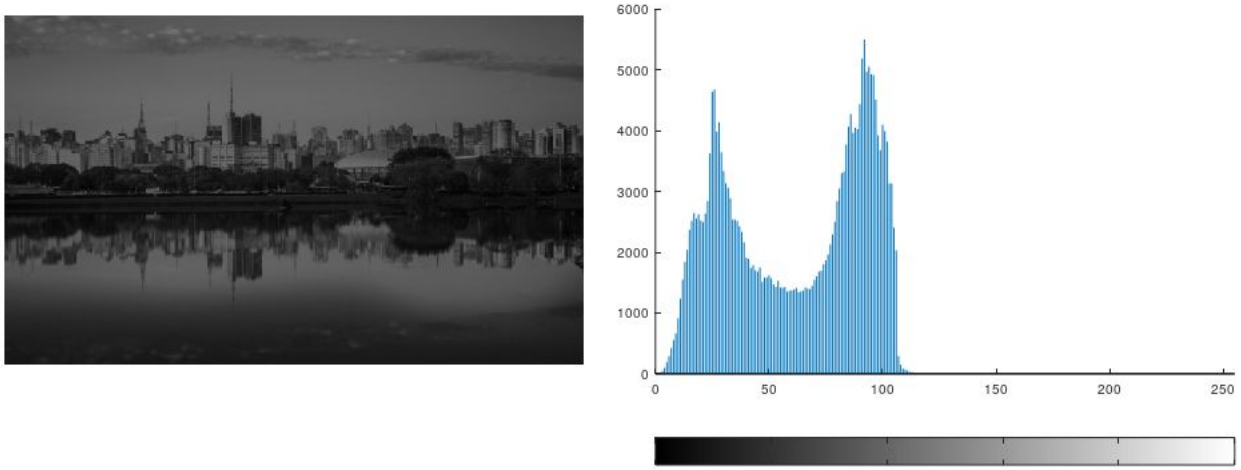


mais clara (100 somado)

mais escura (menos 100)

Uma das técnicas mais usadas em processamento de imagens envolve manipular o histograma com vistas a aumentar (ou diminuir) o contraste de uma imagem. A idéia para obter o melhor constraste é alargar o histograma de maneira a alcançar todo o espectro de níveis de cinza. A função aqui é `imadjust` que tem 3 parâmetros: a imagem, o intervalo de histograma da imagem original – medido em valores reais entre 0 e 1, e o intervalo que deve ser usado na reconstrução da imagem, igualmente medido com reais entre 0 e 1.

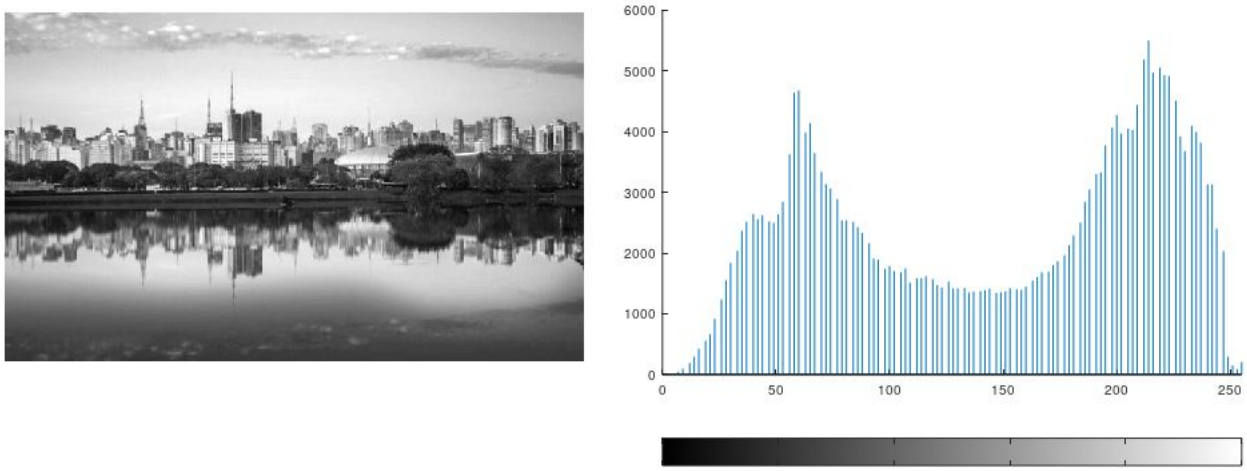
Para entender este método, considere uma imagem contendo muitos pixels escuros, com seu histograma ao lado, como em



Como se pode ver no histograma a variação de cinzas na imagem é entre 0 e 110 (aproximadamente). Como o comando exige um real entre 0 e 1, calcula-se $110 \div 255 = 0.43$ e o nosso intervalo da imagem original é [0,0.43]. A aplicação mais simples manda esticar esse intervalo para [0,1]. É isso que faz o comando

```
>> h = imadjust(e,[0,0.43],[0.1])
>> imshow(h)
>> imhist(h)
```

com o resultado



Perceba como a forma do histograma não muda, ele apenas é esticado para abranger um intervalo maior.

Uma observação é que quando o intervalo é entre 0 e 1 (`[0,1]`) os valores podem ser omitidos – já que este é caso mais comum. Assim o comando acima poderia ter sido escrito como

```
>> h = imadjust(e,[0,0.43],[])
```

Apenas como observação note que o comando `imadjust(x,[],[1,0])` gera o negativo da imagem.

Para imagens true color, deve-se lembrar que todos estes procedimentos poderão ser realizados sobre cada componente de cor, de maneira individual. Ao fazer

```
a = imread('d:/imas/gato.jpg')
```

Se a imagem `gato.jpg` for uma imagem true-color, a variável `a` terá a mesma forma da imagem (linhas e colunas) e terá ainda 3 componentes, a saber `RGB`. Então a componente R é dada por $a(:,:,1)$. A componente G é dada por $a(:,:,2)$ e a componente B é $a(:,:,3)$.

Vejamos uma manipulação destas características

```
>> a = imread('d:/imas/gato.jpg');  # gato.jpg é true color
>> imshow(a)
>> red = a(:,:,1);                  # componente vermelho
>> reda = imadjust(red,[0.1,0.9],[0.5,0.7]); # estreita a componente vermelha
>> nova = cat(3,reda,a(:,:,2),a(:,:,3));
>> imshow(nova)
```

Veja os 2 resultados, o original e o modificado em vermelho

Note que a função cat não é parte do pacote de imagens e ela se limita a concatenar em 3 dimensões as 3 matrizes dadas. Tanto `reda` quanto `a(:,:,2)` e `(a:,:,3)` são matrizes de 369,524 que é o tamanho da imagem do gato.

E, já que estamos brincando, vamos a algumas experiências

```
>> a = imread('d:/imas/gato.jpg');
>> x = cat(3,a(:,:,2),a(:,:,3),a(:,:,1); # note a inversão RGB virou GBR
>> y = cat(3,a(:,:,3),a(:,:,1),a(:,:,2)); # RGB virou BRG
>> imshow(x) ; imshow(y);
```

x = G virou R, B virou G e R virou B

y = B virou R, R virou G e G virou B

Analisando estes resultados pode-se afirmar que a componente R da imagem do gato é a mais forte e isto pode ser avaliado simplesmente olhando a imagem original.

7.1.4 Limiares

Um limiar simples mas muito usado faz com que um pixel se torne 1 se o nível de cinza $>$ limiar T e se torne 0 se nível de cinza $\leq$ limiar T.

Esta técnica é uma parte vital da *segmentação de imagens* usada para isolar objetos do fundo. Também é um componente importante na visão robótica. Limiares podem ser calculados facilmente. Suponha uma imagem cinza guardada em X. O comando `X>T` efetua a limiarização, veja-se

```
>> i=imread('d:/imas/piscinac.jpg');
>> imshow(i)
>> imshow(i>100);
```

gerou o seguinte resultado


Outra possibilidade de limiarização é dada pela seguinte operação:
Um pixel se torna 1 se o nível de cinza está entre T_1 e T_2 e se torna preto em caso contrário. Para implementar isto, supondo a imagem em X escreve-se $X > T_1 \& X < T_2$. Veja-se um exemplo

```
>> x = imread('d:/imas/moca_aguac.jpg');
>> y = x>100 & x<200 ;
>> imshow(x); imshow(z);
```


x


y

7.2 Filtragem Espacial

Na filtragem, um valor de pixel é modificado em função de alguma função aplicada à sua vizinhança. A idéia é mover uma *máscara*, um retângulo com dimensões ímpares ou mesmo com qualquer outra forma. Aplicar a filtragem é calcular uma nova imagem cujos valores de pixel são calculados a partir dos valores sob a máscara. A combinação de uma máscara e uma função é chamada **filtro**. Se o novo valor é calculado por uma função linear de todos os valores de cinza sob a máscara o filtro é chamado filtro linear.

Um simples mas importante filtro linear é usando uma máscara 3×3 e calculando o novo valor do pixel como a média dos valores sob a máscara, acompanhe no esquema

	a	b	c	
	d	E	f	
	g	h	i	

$$\rightarrow \frac{1}{9}(a+b+c+d+E+f+g+h+i) = novo(E)$$

Note que os valores usados $(a..i)$ são os valores da parte da imagem original que se encontra embaixo da máscara que se move pela imagem exceto nos píxeis limite (da primeira e última linha e da primeira e última coluna), para não extravasar a máscara para fora da imagem. Outra estratégia é ignorar este fato (a inexistência de vizinhos) e considerá-los como valendo zero.

Vamos ver a aplicação deste filtro a uma imagem qualquer

```
>> a=ones(3,3)/9
a =
   0.11111   0.11111   0.11111
   0.11111   0.11111   0.11111
   0.11111   0.11111   0.11111
```

```
>> b=imread('d:/imas/mamaec.jpg');
>> c=filter2(a,b,'valid');
>> d=uint8(c);
>> imshow(d)
>> e=ones(5,5)/25;
>> f=filter2(e,b,'valid');
>> g=uint8(f);
>> imshow(g)
>> h=ones(21,21)/(21*21);
>> i=filter2(h,b,'valid');
>> j=uint8(i);
>> imshow(j)
```

Veja o resultado

b: imagem original

d: após média 3x3

g: após média 5x5

j: apos média 21x21

Neste comando a palavra 'valid' determina aplicar o filtro apenas aos pixeis internos. Produz uma imagem um pouco menor. Já a palavra 'same' (que é o valor *default*) mantém a imagem com seu tamanho original e preenche píxeis limite com 0. Como este é o valor padrão, se a palavra é omitida no comando, vale 'same'.

A função `filter2(fil,ima,mét)` pega um filtro (`fil`), aplica-o sobre uma imagem (`ima`) usando um método (`mét`). O valor default é same que produz matriz de mesmo tamanho, preenchendo os limites com zeros.

7.2.1 Passa alta e passa baixa

A frequência de uma imagem é a mudança de valores de cinza de acordo com a distância de píxeis. Componentes de alta frequencia se caracterizam por muita mudança em pouca distância (limites e ruído). Componentes de baixa frequência são componentes da imagem que sofrem pouca mudança em largos pedaços. Isto inclui o fundo, ou largos trechos constantes.

Pode-se então criar filtros

Passa alta reduz ou elimina componentes de baixa frequência

Passa baixa reduz ou elimina componentes de alta frequência

Por exemplo, o filtro 3×3 de média, é um filtro de passa baixa e tende a enevoar ou obscurecer as bordas da imagem, como vimos no caso da mamãe hipopótamo e seu filho.

Já o filtro

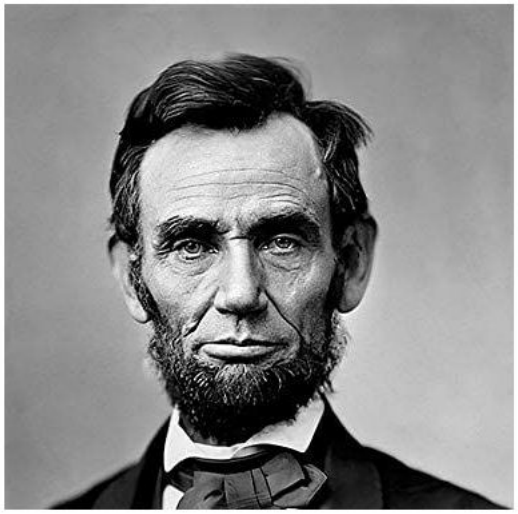
$$\begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

é um filtro passa alta. Note que a soma dos coeficientes do filtro é zero. Isto significa que em regiões da imagem onde há pouca variação, os novos píxeis terão valores próximos a zero, o que caracteriza um passa alta.

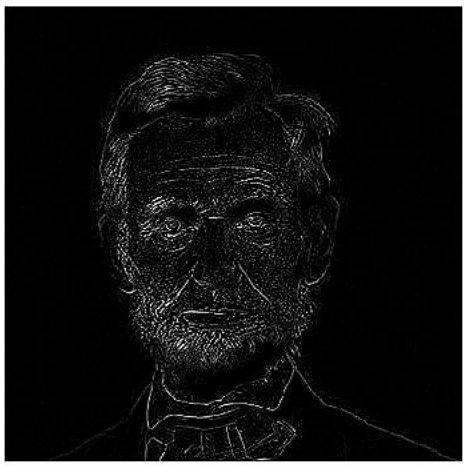
Um outro filtro passa alta é o laplaciano

```
>> pkg load image
>> b=imread('d:/imas/lincoln.jpg');
>> f=fspecial('laplacian')
f =
   0.16667   0.66667   0.16667
   0.66667  -3.33333   0.66667
   0.16667   0.66667   0.16667
>> cf=filter2(f,b);
>> imshow(uint8(cf))
>> f1=fspecial('log')
f1 =
   0.0000053189   0.0012874885   0.0073992526   0.0012874885   0.0000053189
   0.0012874885   0.1731366440   0.4264387919   0.1731366440   0.0012874885
   0.0073992526   0.4264387919  -3.1509801558   0.4264387919   0.0073992526
   0.0012874885   0.1731366440   0.4264387919   0.1731366440   0.0012874885
   0.0000053189   0.0012874885   0.0073992526   0.0012874885   0.0000053189
>> cf1=filter2(f1,b);
>> imshow(cf1/50);
```

A seguir, o original e as duas imagens (cf e cf1)

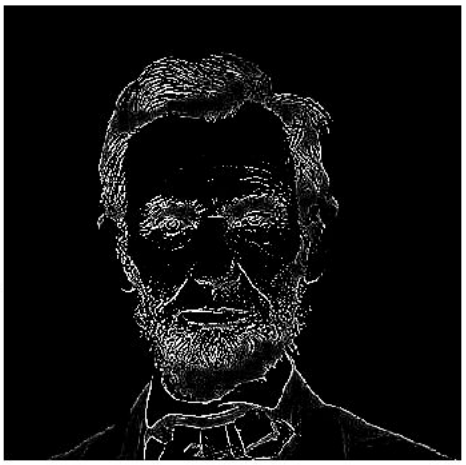


<=== imagem original



<=laplaciano

log 5x5 ==>



7.2.2 Filtros Gaussianos

Já vimos exemplos de filtros lineares (filtro da média e filtros passa alta). A função `fspecial` pode produzir muitos filtros para uso com a função `filter2`.

Filtros gaussianos são uma classe de filtros passa baixa baseados na distribuição de probabilidade gaussiana $f(x) = e^{-\frac{x^2}{2\sigma^2}}$ onde σ é o desvio padrão.

O comando `fspecial('gaussian')` produz uma versão discreta da função acima para duas dimensões. Filtros gaussianos têm um efeito de passa-baixa.

7.2.3 Filtros não lineares

A função a usar é `nlfilter` que aplica um filtro a uma imagem de acordo com uma função pré-definida. Se a função não foi definida previamente ela pode ser criada em um arquivo tipo m (fonte para o Octave). Aqui ocorreram diferenças entre MatLab e Octave. Fica para mais análise...

7.3 Ruído

Pode-se definir ruído como qualquer degradação do sinal da imagem causada por distúrbios externos. Os erros vão aparecer na saída da imagem dependendo do tipo de distúrbio. Usualmente sabe-se o tipo de erro a esperar e portanto pode-se escolher os métodos mais apropriados para reduzir seus efeitos. Limpar a imagem corrompida por ruído é uma área importante da *restauração de imagens*.

7.3.1 Tipos de ruído

Ruído sal e pimenta

Caracteriza a distribuição randômica de pontos brancos ou de pontos negros ou de ambos sobre a imagem. Para gerar este tipo de ruído, a função é `imnoise` que tem diferentes parâmetros. Veja

```
>> a = imread...
>> b = imnoise(a,'salt & pepper');
```

O valor adicionado de ruído aqui é de 10%. Para mais ou menos, incluir um parâmetro opcional, onde um valor entre 0 e 1 indica a fração de pixels que deve ser corrompida. Um valor de 0.2 determina a corrupção de 20% dos pixels.

Ruído gaussiano

É a forma idealizada de *ruído branco*, que é causado por flutuações randômicas no sinal. Ele pode ser observado olhando uma televisão não sintonizada em nenhum canal. Se a imagem é I e o ruído gaussiano é N pode-se modelar uma imagem ruidosa somando ambos ou $I + N$, ou

```
>> a = imread...
>> b = imnoise(a,'gaussian');
```

Como no sal e pimenta o ruído gaussiano admite os valores não obrigatórios de média e variância do ruído. Os valores *default* são 0 e 0.01

Ruído periódico

Se a imagem está sujeita a um ruído periódico (e não ao randômico), o efeito são barras que aparecem sobre a imagem. A função `imnoise` não tem a opção periódica, mas é fácil criar uma. veja-se

```
>> a=imread('d:/imas/lincoln.jpg');
>> s=size(a);
>> [x,y]=meshgrid(1:s(1),1:s(2));
>> p=sin(x/3+y/5)+1;
>> t_pn=(im2double(a)+p/2)/2;
>> imshow(t_pn)f
```


Limpando ruido sal & pimenta

Posto que os pixels brancos e pretos são componentes com alta frequência da imagem, deve-se tratá-los com filtros passa-baixa. Pode-se tentar um filtro da média

```
>> a3=fspecial('average');
>> ima2 = filter2(a3,ima1);
>> ima3 = uint8(ima2);
>> a4=fspecial('average',[7,7]);
>> ima4 = filter2(a4,ima3);
>> ima5 = uint8(ima4);
```

Veja-se a ima1 e a ima3 e ima5:

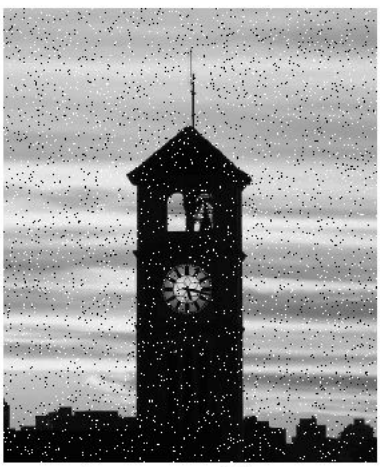
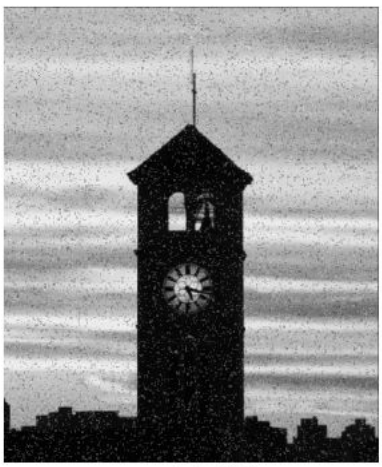


imagem com ruído


média 3x3


média 7x7

oi

Índice Remissivo

Capítulo 8

Referências Bibliográficas

Gon10 GONZALEZ, Rafael e WOODS, Richard. **Processamento Digital de Imagens**. Pearson, São Paulo, 2010.

Sol11 SOLOMON, Chris e BRECKON, Toby. **Fundamentals of Digital Image Processing**. Wiley-Blackwell, Oxford, 2011.

Mca00 McANDREW, Alasdair. **An Introduction to Digital Image Processing with MatLab**. Victoria University, Victoria, 2000.

Ach05 ACHARYA, Tinku e RAY, Ajoy. **Image Processing - Principles and Applications**. Wiley & Sons, Arizona, 2005.

Mat00a MATH, Works The. **Image Processing Toolbox for use with Matlab - User Guide**. MathWorks. sc. 2000.

Mat00b MATH, Works The. **Image Processing Toolbox for use with Matlab - Function Reference**. MathWorks. sc. 2000.

Tut20 TUTORIALSPOINT. **Python Pillow**. sc, www.tutorialspoint.com, 2020.

Fac96 FACON, Jacques. **Morfologia Matemática**. Curitiba, se, 1996.