UNIVERSIDADE FEDERAL DE MINAS GERAIS — UFMG
INSTITUTO DE CIÊNCIAS EXATAS — ICEX
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO — DCC

Disciplina:	DCC884 — Visão Computacional
Professor:	Mario Fernando Montenegro Campos (`mario@dcc.ufmg.br`)
Monitor:	Vilar Fiuza da Camara Neto (`neto@dcc.ufmg.br`)
Período:	1º semestre / 2007
Aulas:	Segundas e quartas, das 11:10 às 12:50, na sala 2029 do ICEx
Página:	`http://www.dcc.ufmg.br/verlab/doku.php?id=cursos:visao:index`

3ª Lista de Exercícios

Data da entrega: 07/mai/2007

A lista é individual e as datas de entrega são fixas, devendo o trabalho ser entregue no início da aula. Cada lista vale 5 pontos na nota final. Para cada dia de atraso na entrega será descontado 0,5 ponto do valor da lista.

Esta lista de exercícios é composta de duas partes: a primeira de exercícios teóricos e a segunda com implementações práticas. Não há formato específico para apresentar as respostas e resultados. Porém, clareza na redação é um requisito essencial. Embora o resultado possa ser manuscrito, lembre-se de que "o que não se pode ler não se pode avaliar". Assim, sugere-se que o trabalho seja digitado, podendo o desenvolvimento matemático ser manuscrito.

O Matlab ou Scilab são recomendados para o desenvolvimento das aplicações, pois permitem o desenvolvimento rápido de aplicativos e já incorporam várias rotinas matemáticas e de manipulação de imagens. O Scilab é gratuito e pode ser baixado da página `http://www.scilab.org/`. Você também pode usar outras linguagens de programação, mas nesse caso lembre-se de incluir as instruções de compilação dos programas e eventuais arquivos auxiliares, como "makefiles" ou arquivos de projeto.

Em todos os casos, não se exige nenhuma interface gráfica: os programas podem ler os dados de entrada por argumentos de linha de comando ou por digitação pelo usuário, e podem gravar a saída em arquivos no disco.

Para arquivos gráficos, recomenda-se o uso dos formatos PNG ou JPEG. Nesse sentido o Matlab e o Scilab ajudam, pois ambos disponibilizam rotinas de leitura e gravação para esses formatos. Para os entusiastas de C e C++, a biblioteca FreeImage (`http://freeimage.sourceforge.net/`) é uma alternativa gratuita e multi-plataforma. Para a visualização dos arquivos, os aplicativos IrfanView (Windows) e xv (Linux) são boas ferramentas e também são gratuitos.

Em qualquer situação, os programas devem ser entregues com documentação de uso ou devem ser auto-explicativos.

1 Exercícios Teóricos

1. Explique como o espaço de parâmetros (θ, ρ) oferece melhor discretização que o espaço (m, n) para a Transformada de Hough.

2. A detecção de linhas pode ser implementada como um procedimento do tipo *template matching*, que nada mais é do que a filtragem da imagem com máscaras que respondam a linhas em diferentes orientações, seguida de um limiar para selecionar os pixels que pertençam a linha. Compare o *template matching* com a Transformada de Hough como métodos para a detecção de linhas em imagens.

3. O 1º Exercício Prático trata do cálculo dos autovalores e autovetores correspondentes aos pixels de uma imagem. A Transformada de Hough poderia fazer uso desses dados para melhorar os resultados? Como?

2 Exercícios Práticos

1. O arquivo `corners.m`, disponível no site do curso, calcula os dois autovalores $\lambda_1 \geq \lambda_2$ como apresentado no algoritmo CORNERS na página 84 do livro-texto [Trucco and Verri, 1998]. O autovetor correspondente a λ_1 também é calculado.

   a) Como você pode usar esses autovalores para a detecção de arestas? (O conjunto de arestas detectadas não deve incluir as quinas).

   b) Como você encontra a direção do gradiente?

   c) Implemente um programa com base nas suas idéias expostas nas questões anteriores. Para as arestas, use *nomaximum suppression* mas não execute a histerese. Utilize um algoritmo simples de limiar (*thresholding*) para determinar se um pixel deve ou não pertencer a uma aresta. As quinas não necessitam ser pontos isolados (pule os passos 3 e 4 do algoritmo CORNERS). Seu programa deve produzir uma imagem que é branca exceto onde existe um marcador na forma de ponto nos locais onde existem arestas e um marcador em forma de quadrinho no local onde uma quina for detectada.

      Experimente o seu algoritmo com a imagem `venice.png` (Figura 1), disponível no site do curso. Se quiser, experimente também com imagens da internet.

2. O objetivo deste exercício é escrever uma rotina capaz de extrair as retas presentes em uma imagem. O trabalho é feito em duas etapas: (i) utiliza-se um detector de bordas para extrair todos os pontos onde há uma transição brusca na intensidade dos pixels e (ii) aplica-se um detector de retas para descobrir conjuntos de pontos de borda que estejam alinhados.

   Para os experimentos, use a imagem `frankfurt_airport_1024.jpg` (Figura 2), disponível no site do curso. Se quiser, pode usar qualquer outra imagem, mas é importante que ela contenha retas facilmente identificáveis. (Lembre-se de reduzir a imagem para algo em torno de 800×600 ou 640×480, senão o tempo de processamento poderá ser muito alto.)

   a) Converta a imagem em escala de cinza e passe-a por um detector de bordas. Você não precisa implementar o detector: o Matlab e o Scilab possuem uma função, `edge`, que realiza essa tarefa. Experimente os diversos algoritmos (Canny, Sobel, Prewitt, Roberts, etc.) e escolha o mais adequado para o seu objetivo (ou seja, aquele que melhor realça as bordas retas).

Figura 1: Imagem `venice.png`.

Figura 2: Imagem `frankfurt_airport_1024.jpg`.

b) Implemente a Transformada de Hough para a detecção de retas em uma imagem. Use a parametrização (θ, ρ), que oferece melhor discretização do espaço de parâmetros, conforme discutido em uma das questões teóricas. Execute o seu programa sobre a imagem resultante do detector de bordas. Plote as retas detectadas sobre a imagem original (assim será mais fácil de analisar os resultados).

c) Analise criticamente os resultados obtidos: todas as retas importantes foram detectadas? Há retas detectadas que não existem na imagem original? Há

retas que foram detectadas mais de uma vez — ou seja, com mais de uma combinação de pares (θ, ρ)? Por quê? Filosoficamente, o que poderia ser feito para melhorar os resultados?

Referências Bibliográficas

Trucco, E. and Verri, A. (1998). *Introductory Techniques for 3-D Computer Vision*. Prentice Hall PTR, Upper Saddle River, NJ, USA.