VISÃO COMPUTACIONAL COM A OPENCV – MATERIAL APOSTILADO E VEÍCULO SEGUIDOR AUTÔNOMO

Riccardo Luigi Delai[1]; Alessandra Dutra Coelho[2]

[1] Aluno de Iniciação Científica da Escola de Engenharia Mauá (EEM/CEUN-IMT);
[2] Professora da Escola de Engenharia Mauá (EEM/CEUN-IMT).

Introdução

Visão Computacional, ou Visão Robótica, é um tema que está em franca ascensão. Ela é ponto chave para o desenvolvimento de diversas outras tecnologias, especialmente na área da robótica móvel. Suas aplicações são inúmeras: todas as decisões tomadas baseadas no que pode enxergar fazem parte do escopo da Visão Computacional. A semelhança com o modo de percepção humana do ambiente a torna uma ferramenta extremamente poderosa. Este é um sentido que já é bastante dominado pela raça humana, pois podemos facilmente distinguir dois objetos distintos apenas ao olhar para eles, e classificá-los em categorias, como carro, casa, pessoa.

No entanto, esta tarefa simples é bastante dispendiosa computacionalmente, o que explica porque somente agora o campo da Visão está em tamanha expansão. Os motivos são o barateamento do poder de processamento e da fotografia digital, além da miniaturização de ambos. Isso tornou possível aplicações *real-time*, nas quais as imagens são processadas em tempo semelhante ao do cérebro humano, o que seria impossível com os computadores e sensores de dez anos atrás, tanto por questões de custo quanto de velocidade e tamanho. Atualmente é possível se projetar sistemas simples de visão que possam ser embarcados em robôs autônomos de pequeno porte, em escala reduzida do corpo humano **citar artigo da competição humanoides**.

Entre as aplicações já existentes da Visão Computacional estão: câmeras inteligentes (que buscam rostos ao tirar fotos), controle de qualidade industrial por inspeção visual, sistemas de entretenimento sem controle, realidade aumentada, protótipos de veículos sem motorista, voo não tripulado autônomo, interpretação automática de exames médicos e análise de tecidos biológicos.

A OpenCV é um conjunto de ferramentas de programação para desenvolvimento de aplicações com Visão. Ela engloba também outro conceito importante, especialmente no meio acadêmico, o software livre. A biblioteca é completamente *open-source*, e é distribuída gratuitamente, aberta a colaborações de qualquer indivíduo ou empresa voluntários.

A gratuidade da OpenCV, o baixo custo do poder de máquina e a crescente qualidade das câmeras torna possível o desenvolvimento de sistemas sofisticados de Visão, com baixo investimento e custo de operação. A utilização de câmeras pode inclusive substituir outros sensores e sistemas mais caros, complexos e menos genéricos. Isso evidencia não só o crescimento da área da Visão Robótica, mas também a tendência de aproximação dos computadores à forma como os humanos entendem o ambiente ao seu redor.

O sistema desenvolvido para o veículo é uma prova de conceito do estudo realizado. Ele foi adaptado a partir de materiais de fácil acesso e ampla comercialização, tornando um simples carrinho de controle remoto em um sistema semi autônomo, que tenta encontrar e seguir uma forma pré-determinada.

Materiais e Métodos

<u>Material escrito</u>

Para o desenvolvimento e teste do material escrito, foram necessários a API da biblioteca, um computador com compilador C++ e a OpenCV instalada. Mais

particularmente, foi utilizado o Linux como sistema operacional, por motivos de comodidade na programação e seguindo a ideologia de *software* livre da OpenCV, mas foram feitos também testes com Windows para que o material fosse o mais universal possível.

Também foi necessária uma *webcam* digital USB, para a qual não há grande necessidade de especificação. Idealmente, a resolução deve ser de no mínimo 640x480, e o *frame-rate* de no mínimo 20 fps. É conveniente se ter um editor de imagens instalado, para que se possa criar e editar imagens para cada procedimento experimental.

Antes de partir para a redação, foi preciso detalhar o procedimento de instalação do compilador e da OpenCV no ambiente Windows para criar uma instalação padrão para servir de base para os exemplos. **citar na bibliografia**.

O material escrito não é uma tradução do conteúdo dos livros. Em muitas situações, os livros voltados à programação citam apenas que determinado procedimento é mais demorado do que outro, enquanto os livros mais teóricos ignoram os aspectos de implementação e abordam apenas os conceitos algorítmicos e matemáticos, não levando muito em conta o custo computacional. Para um curso de engenharia, no qual os leitores têm tanto conhecimento matemático quanto de implementação computacional, é necessário oferecer um panorama geral, que demonstre os motivos tanto teóricos quanto práticos de cada procedimento. Portanto, deve ser buscado um equilíbrio tanto entre teoria e prática quanto para a abrangência do material, para que seja possível o desenvolvimento de projetos diversos, mas mantendo-o compacto e objetivo.

Dentro de cada capítulo, cada rotina proposta deve ser testada cuidadosamente. Por ser uma biblioteca bastante dinâmica, com atualizações frequentes, muitas das soluções propostas na literatura já estão obsoletas ou foram alteradas. Assim, a consulta em fóruns especializados, documentação atualizada, *changelogs* e até testes para verificação devem ser feitos com frequência.

A melhor forma de se passar um conhecimento na área de processamento de imagem é utilizar as próprias imagens como exemplo, mostrando o que os operadores fazem sobre elas. É nesta etapa que o editor de imagens se torna interessante, para gerar imagens que se adequem aos exemplos do material. Nos casos de morfologia por exemplo, é conveniente se trabalhar com imagens bastante simples, para que se perceba imediatamente o efeito causado pelos algoritmos, enquanto nos processos com histogramas e detecção de bordas é mais interessante a demonstração em uma imagem mais sofisticada, como uma fotografia real, para que o efeito geral sobre toda a superfície seja mais notado do que o particular em pontos específicos.

Trabalhar diretamente com câmera e vídeos como exemplos não é uma boa prática. É muito difícil se ter um fundo completamente uniforme durante todos os quadros, e uma câmera completamente livre de ruídos. Assim, alguns dos processos não funcionam tão bem quando aplicados diretamente sobre uma imagem vinda de uma câmera. A utilização da câmera varia conforme a aplicação, cabendo ao implementador definir que filtros devem ser aplicados antes de partir para o processamento.

Foi dada preferência a procedimentos no domínio especial da imagem, ao invés do domínio da frequência como apresentado grande parte da literatura teórica. As transformadas de Fourier (FFT) direta e inversa são relativamente lentas frente ao domínio especial, como as imagens se apresentam e são armazenadas, podendo comprometer a velocidade total dos processos.

Veículo autônomo

Para provar os conceitos estudados sobre Visão Computacional e demonstrar uma possível aplicação na área de robótica móvel, foi desenvolvido um sistema que identifica um símbolo pré-definido em uma imagem, vinda de uma câmera, e comanda um pequeno veículo para segui-lo.

O veículo é um carrinho de controle remoto, cujo controle foi adaptado para que fosse

possível controlá-lo por um sistema externo, no caso uma plataforma Arduino. Como no controle original os movimentos de acelerar, ré, direita e esquerda eram controlados por botões simples, o microcontrolador simula uma chave variando suas saídas entre *input* (estado de alta impedância, chave aberta) e *output* nível baixo, que corresponde à chave fechada no controle. Isto dispensa componentes extra na adaptação do controle.

O Arduino é ligado a um computador via USB, que envia dados através de emulação serial para que esse altere o estado no controle. São enviados comandos de apenas uma letra, representando situações desejadas no controle, e por consequência no carrinho. O computador toma as decisões de que movimento realizar baseado nas imagens vindas da câmera de um celular, posicionado na parte superior do carrinho, que envia seus quadros via Bluetooth.

Para esta última etapa (comunicação câmera-computador) foi necessário o uso de um *software* proprietário pago. Isso acontece porque o celular não permite que programas sem certificação de empresas de segurança tenham acesso a dados considerados sensíveis, como a câmera. Acessos a esta obrigam o usuário a apertar um botão a cada quadro obtido, confirmando deste, o que inviabilizaria o projeto.

Com a imagem da câmera, o computador inicia o processo de busca do símbolo, com base em uma referência digital deste. O processo, repetido para cada frame obtido, segue os seguintes passos:

1. Obter a imagem da câmera.
2. Suavizar (*smooth*) a imagem vinda da câmera, para diminuir o ruído espúrio. O mecanismo de suavização escolhido foi o *blur* gaussiano.
3. Converter a imagem colorida para preto-e-branco (escala de cinza).
4. Aplicar *thresholds* crescentes, gerando imagens binárias (preto e branco, apenas). São aplicados valores crescentes para que o símbolo buscado possa ser visto em várias condições de luminosidade ambiente. Para cada imagem gerada neste item:
5. Aplicar rotina de detecção de contornos em toda a imagem. A imagem binária torna-se uma sequência de pontos que representam as fronteiras entre regiões pretas e brancas.
6. Simplificar os contornos, utilizando o algoritmo de Douglas-Peucker de aproximação de polígonos. Selecionar contornos quadrados pela relação de aspecto, número de vértices e tamanho mínimo. Isso cria uma lista de regiões candidatas a conter o símbolo, buscando pelo quadrado externo.
7. Obter todas as regiões quadradas candidatas novamente na forma de imagem.
8. Procurar o quadrante com maior quantidade de *pixels* brancos para cada região do item anterior.
9. Rotacionar as imagens do item 7 para que sigam a orientação da referência, baseado no quadrante mais branco identificado no item 8 e no ângulo formado entre a linha inferior do contorno e o eixo horizontal da imagem.
10. Calcular os momentos da imagem binária, resultado do item anterior. Os momentos utilizados são tais que a ordem em X somada com a ordem em Y seja menor ou igual a 3, com X e Y positivos.
11. Uma nota é composta com o quadrado da diferença entre os momentos de cada ordem da referência e da imagem candidata. A isso é somado dez vezes a diferença entre a quantidade de figuras presentes na referência e no candidato, sendo uma figura qualquer porção preta envolvida por branco ou branca envolvida por preto.
12. A região candidata com menor nota, que esteja acima de um limiar estabelecido experimentalmente é considerada como um *match* válido. O limiar pode ser alterado durante a execução do programa, para mudar a tolerância deste. O controle é dado na forma de uma barra deslizante.
13. Se houver algum *match*, o ponto central (centro do quadrado) é considerado para avaliar sua posição. Se este estiver nos primeiros 30% da imagem, na parte esquerda, o carrinho é comandado para virar à esquerda. Se estiver nos 70% mais à direita, é enviado um comando para que o carrinho vire à esquerda. Para determinar a distância

do carrinho à figura, é utilizada a área do quadrado externo. Se esta for menor que 5% da imagem, o carrinho deve avançar. Se for maior que 12%, ele deve andar de ré e se afastar. Todos estes parâmetros podem ser alterados em tempo de execução.

14. Com o carrinho reposicionado, voltar ao passo 1.

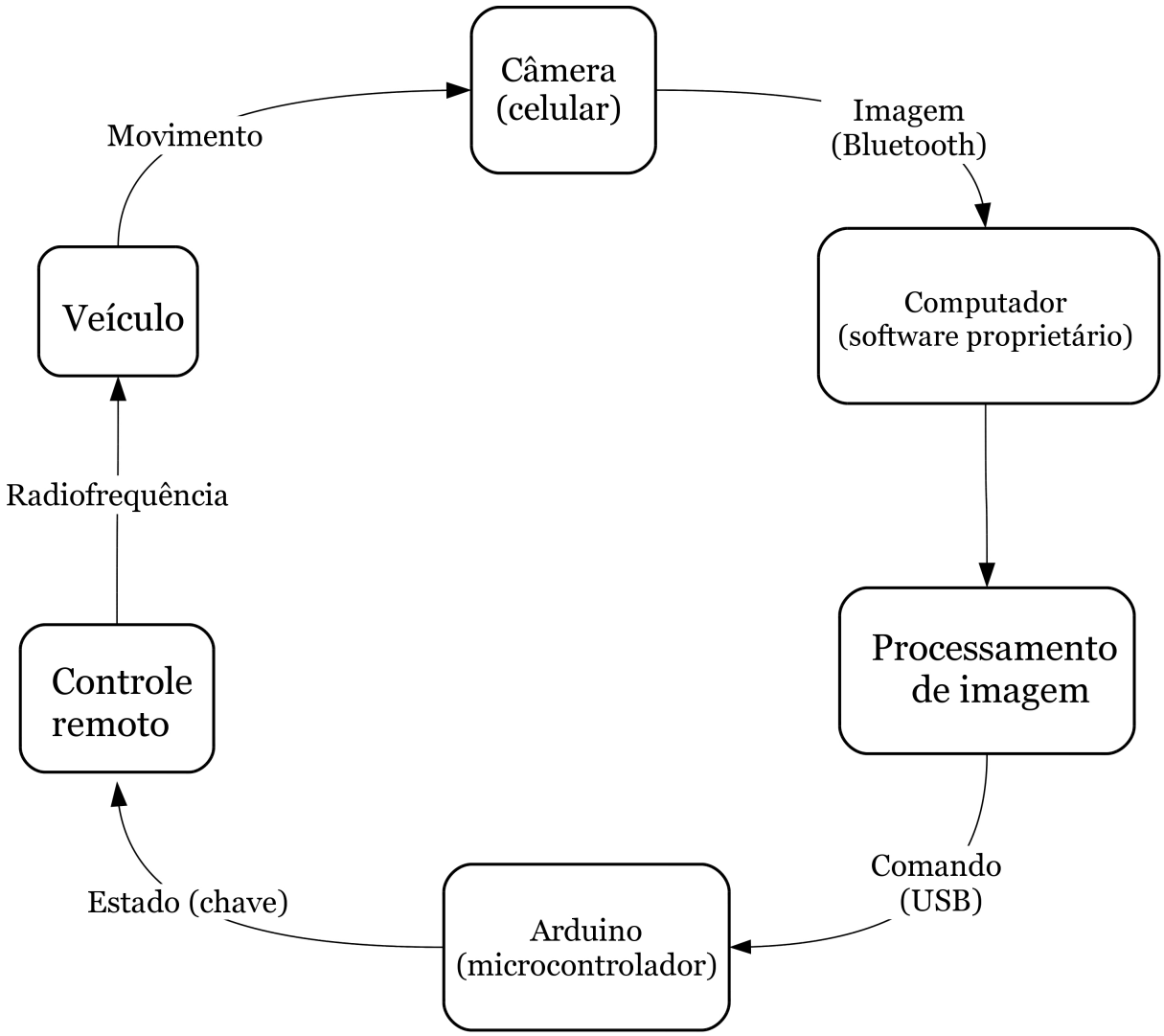

Figura 1 - Esquema do funcionamento do sistema do veículo

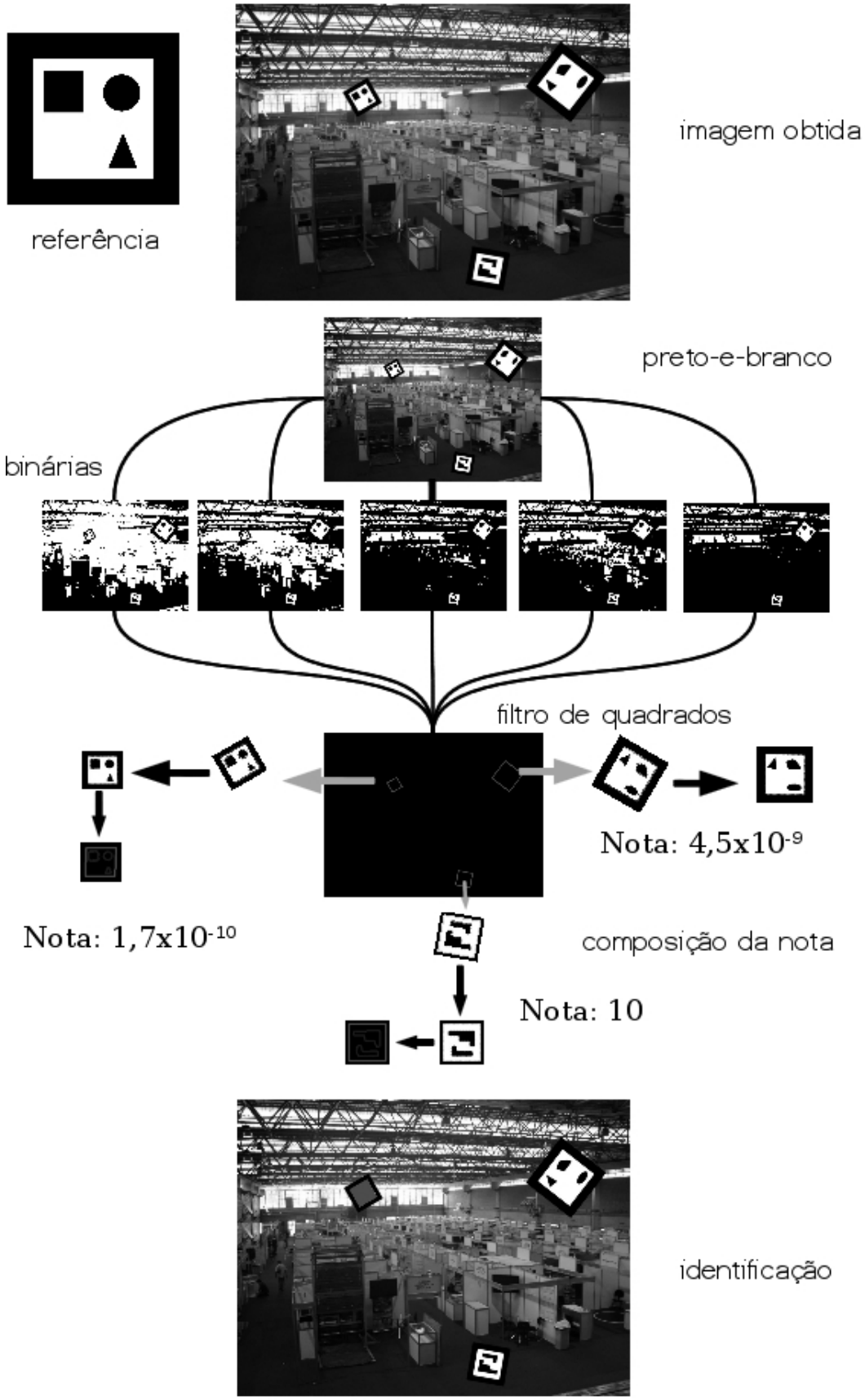

Figura 2 - Processo de busca da referência no programa

Resultados e Discussão

Veículo autônomo

O sistema de identificação do símbolo funcionou bem quando testado em imagens estáticas, mesmo com ruído e com a adição de símbolos semelhantes ao procurado, sempre identificando corretamente a referência com a nota mais baixa.

Com a câmera adaptada no carrinho, o sistema também se comportou bem. Os maiores problemas são com a movimentação do carrinho, que não tem grande ângulo de curva nem controle de intensidade de aceleração ou ré. Por causa disso, o carrinho tende a avançar na direção do símbolo assim que o encontra ao longe, mas ao perceber que está perto e acionar o freio, a inércia dele o movimenta além do ponto de parada esperado, deixando-o próximo demais. Quando está muito próximo, a ré é acionada para tentar manter a distância ideal, mas o mesmo acontece, e o carrinho fica em um ciclo de frente e ré sem encontrar a distância ideal prevista.

Para minimizar este problema, seria ideal que o carrinho tivesse controle de velocidade, para que acelerasse proporcionalmente à distância que se encontra da referência.

Para diminuir o custo de processamento, o programa poderia não procurar pelo símbolo a cada *frame* recebido, mas tentar segui-lo uma vez encontrado. Isso poderia ser feito com algum algoritmo de fluxo óptico, como o de Lucas-Kanade.

Conclusões

O material escrito desenvolvido servirá como base para a introdução de um laboratório para difundir os conhecimentos de Visão, tanto na grade curricular como para auxiliar o laboratório de robôs autônomos.

O veículo adaptado é o início de um projeto maior de desenvolvimento de um carro autônomo. Ele será reproduzido em escala maior em um miniveículo elétrico, para o qual também será aumentado o grau de automação. Ele será capaz de dirigir-se sozinho por distâncias maiores, evitando obstáculos e seguindo percursos previamente definidos, com a introdução de obstáculos dinâmicos.

Referências Bibliográficas

Autores da OpenCV (2010) *OpenCV Reference Manual v2.1*.

Autores da OpenCV (2010) *OpenCV source code v2.1*.

Bradski, G.; Kaehler, A. (2008) *Learning opencv*. 1ª edição. O'Reilly Media.

Gonzalez, R. C.; Woods, R. E. (2000) *Processamento de imagens digitais*. 2ª edição. Edgard Blücher.

Gomes, M. M. (2010) *Apresentações do curso ecm951 - visão computacional*. Escola de Engenharia Mauá.

Russ, J. C. (1998) *The image processing handbook*. 5ª edição. CRC Press.

Russel, S.; Norvig, P. (1995) *Artificial intelligence - a modern approach*. Prentice Hall.

Tudor, P. (1995) *Mpeg-2 video compression*. Electronics & Communication Engineering Journal.

DERPANIS, K. G. (2004) *The harris corner detector.* *www.cse.yorku.ca/~kosta/CompVis_Notes/*harris_detector.pdf.