Universidade de São Paulo

Instituto de Matemática e Estatística

Departamento de Ciência da Computação

Software Livre para Segmentação Automática de Vasos Sangüíneos em Imagens de Retina

Monografia

Autor:
Jeferson Rodrigues da Silva

Supervisor:
Roberto Marcondes Cesar Junior

Colaboradores:
João Vitor Baldini Soares
Jorge de Jesus Gomes Leandro
Herbert Franz Jelinek
Marcelo Mendonça

São Paulo, 2 de dezembro de 2006

Sumário

1 Introdução

A inspeção de imagens da retina é uma prática comum da comunidade médica para o diagnóstico de doenças como a hipertensão e a retinopatia diabética, que é uma das maiores causas de cegueira em adultos [7]. Se diagnosticada a tempo, a retinopatia diabética pode ser tratada, evitando grandes perdas visuais. Entretanto, nem sempre isso é possível pois o diagnóstico correto dessas doenças muitas vezes requer um acompanhamento regular dos pacientes que muitas vezes se encontram em áreas de difícil acesso, como zonas rurais. Além disso, a quantidade de imagens a serem inspecionadas é muito grande já que existe uma quantidade de pacientes. Por esse motivo, métodos computacionais para facilitar a realização dos exames e também o diagnóstico dessas doenças vêm sendo desenvolvidos. O uso destes métodos permitiria que voluntários e profissionais de saúde não especializados como enfermeiros pudessem fazer o acompanhamento dos pacientes em áreas remotas e de difícil acesso sem a necessidade da presença de um especialista.

Este trabalho teve como objetivo o desenvolvimento de um software livre para a segmentação automática dos vasos sangüíneos em imagens de retina (Figura 1), que representa a primeira etapa para o desenvolvimento de métodos que possam auxiliar especialistas no diagnóstico de doenças. O desenvolvimento de um software com essa finalidade deve levar em consideração diversos fatores como usabilidade e eficiência, isto é, o software deve ser o mais simples possível, exigindo pouca ou nenhuma configuração de parâmetros e também deve utilizar um método de segmentação que possua uma implementação eficiente capaz de produzir resultados rápidos. Por esse motivo, o método de segmentação implementado foi o método descrito em [5], que já possui resultados comprovados. O método envolve a utilização de diferentes respostas da transformada wavelet como características para os pixels das imagens e o uso de um classificador bayesiano com modelo de mistura gaussiana para atribuir a cada pixel a classe *vaso* ou *não-vaso*.

Do ponto de vista da arquitetura do software e tecnologias utilizadas, houve uma preocupação especial em se manter o software desacoplado de quaisquer bibliotecas e também em permitir que o software acomodasse possíveis mudanças futuras como, por exemplo, a adição de novas etapas após a segmentação das imagens. Levando em conta esses fatores, a linguagem escolhida para o desenvolvimento do software foi a linguagem C++, por ser bastante eficiente e por ser uma linguagem orientada a objetos. Foram também utilizadas as bibliotecas GTK (`http://www.gtk.org/`), para construção da interface gráfica, e OpenCV (`http://www.intel.com/technology/computing/opencv/`), para o processamento de imagens. Além disso, um protótipo do software, escrito em Matlab (`http://www.mathworks.com/products/matlab`) por João Vitor Baldini Soares, foi utilizado como base para a implementação do método.

As imagens utilizadas para testes durante o desenvolvimento e também utilizadas neste trabalho foram obtidas do banco de imagens DRIVE [6].

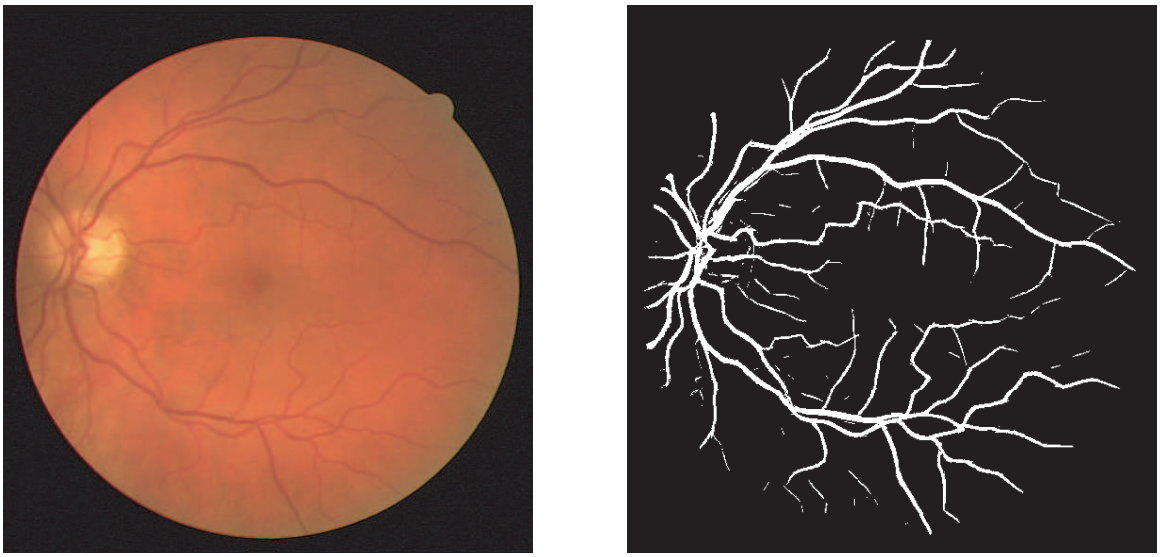


Figura 1: Imagem da retina e sua segmentação automática correspondente

2 Método de Segmentação

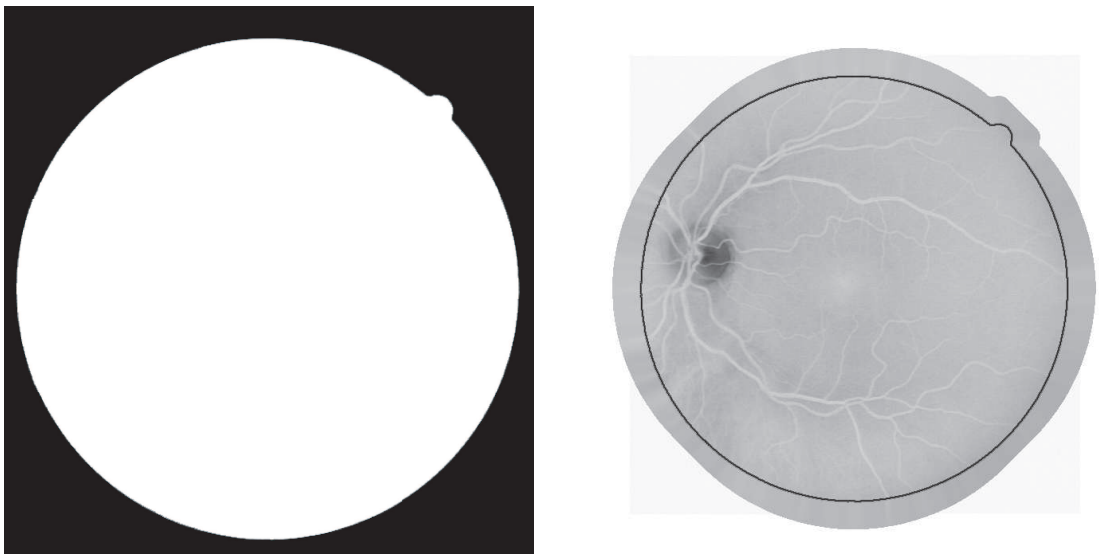
O método utilizado considera os pixels das imagens como objetos que são representados por vetores de características. Inicialmente, é necessário treinar um classificador estatístico com base em um conjunto de treinamento. Este conjunto é gerado a partir de imagens da retina e suas segmentações manuais correspondentes. O treinamento do classificador é realizado então, utilizando o espaço de características dos pixels das imagens juntamente com seus rótulos, que indicam se um pixel pertence à classe *vaso* ou *não-vaso*. Com o classificador já treinado, o processo de classificação consiste em gerar os vetores de características para os pixels das imagens e fornecê-los ao classificador que atribuirá a cada pixel a probabilidade daquele pixel pertencer à classe *vaso*.

A seguir são descritos o processo de geração das características e os processos de treinamento e classificação.

2.1 Características dos Pixels

Para a extração de características, somente os pixels que se encontram no interior do campo de visão da câmera são considerados. Esta região de interesse é determinada por uma máscara que é gerada automaticamente para cada imagem antes da geração das características (Figura 2(a)).

Os valores obtidos na geração das características podem variar muito de uma característica para outra e também de imagem para imagem. Para reduzir essa disparidade e facilitar o processo de classificação, cada característica é normalizada com base em sua própria média e desvio padrão.



(a) Máscara que delimita a região de interesse

(b) Extensão artificial com contorno delimitando a imagem original

Figura 2: Etapas de pré-processamento

A seguir são descritas as características que compõem o vetor de características de cada pixel para imagens coloridas.

2.1.1 Pré-processamento

As imagens da retina normalmente apresentam um forte contraste nas bordas do campo de visão da câmera. Esse contraste pode gerar respostas muito altas da transformada wavelet durante a extração das características. Por esse motivo, um algoritmo iterativo é utilizado para gerar um prolongamento da imagem ao longo da borda do campo de visão da câmera. Em cada iteração, os pixels da borda são substituídos pela média dos valores dos pixels vizinhos internos mais próximos. O resultado da aplicação do algoritmo depois de várias iterações pode ser visto na Figura 2(b).

2.1.2 Canal Verde Invertido

O canal verde invertido da imagem é utilizado como uma das características de um pixel (Figura 3(b)). Ao analisar os canais vermelho, verde e azul invertidos da imagem, podemos observar um maior contraste entre os vasos e o resto da imagem no canal verde. Além disso, os canais vermelho e azul costumam apresentar muito ruído (Figuras 3(a) e 3(c)).

(a) Vermelho (b) Verde (c) Azul

Figura 3: Comparação entre os canais invertidos de uma imagem

2.1.3 Transformada Wavelet Contínua

Sejam $x, b \in \mathbb{R}^2$ e seja $f \in L^2$ uma imagem representada por uma função de energia finita definida sobre $\mathbb{R}^2$. Então podemos definir a transformada wavelet contínua como sendo:

$$T_\psi(b, \theta, a) = C_\psi^{-1/2} \frac{1}{a} \int \psi^*(a^{-1} r_{-\theta}(x - b)) f(x) d^2x \tag{1}$$

onde $\psi, b, \theta, a, C_\psi$ e ψ^* representam a wavelet analisadora, o vetor deslocamento, o ângulo de rotação, a escala, a constante de normalização e o complexo conjugado da wavelet analisadora, respectivamente. No método em questão, a wavelet analisadora utilizada é a wavelet bi-dimensional de Gabor (também conhecida como wavelet de Morlet) que é definida como:

$$\psi_M(x) = exp(jk_0x)exp\left(-\frac{1}{2}|Ax|^2\right) \tag{2}$$

onde $j = \sqrt{-1}$ e $A = diag[\epsilon^{-1/2}, 1], \epsilon \geq 1$ é uma matriz diagonal 2×2 que define o alongamento da wavelet na direção desejada.

O uso da transformada wavelet de Gabor permite detectar estruturas direcionais com freqüências específicas e por esse motivo é possível realizar a detecção dos vasos e a filtragem dos ruídos da imagem em um único passo [1]. Os parâmetros da wavelet analisadora foram ajustados para que a transformada gerasse respostas altas para pixels associados a vasos sangüíneos quando aplicada sobre o canal verde invertido da imagem.

Para ser usada como característica, a transformada wavelet de Gabor é aplicada sobre o canal verde invertido da imagem variando o ângulo θ de 0° até 170° com passo de 10°. Em cada etapa, os vasos que possuem orientação θ são detectados e acumulados no resultado intermediário obtido até o momento. Os módulos máximos das respostas das transformadas são tomados como características dos pixels. A figura 8 mostra os resultados intermediários e finais da aplicação da transformada wavelet contínua em uma imagem.

Além de aplicar a transformada utilizando diversas orientações, a transformada também é aplicada variando-se a escala (parâmetro a) a fim de detectar tantos os vasos mais grossos como os vasos mais finos. Ao final do processo, o conjunto formado pelas respostas de módulo máximo sobre todos os ângulos para diversas escalas é utilizado para compor a parte restante do vetor de características dos pixels.

2.2 Classificadores Estatísticos

Para gerar a segmentação das imagens, o método utiliza um classificador bayesiano com modelo de mistura gaussiana para modelar as funções densidade probabilidade de cada classe. Para que seja possível realizar a classificação, o classificador deve passar primeiro por uma etapa de treinamento utilizando imagens que foram segmentadas manualmente (Figura 5).

2.2.1 Treinamento

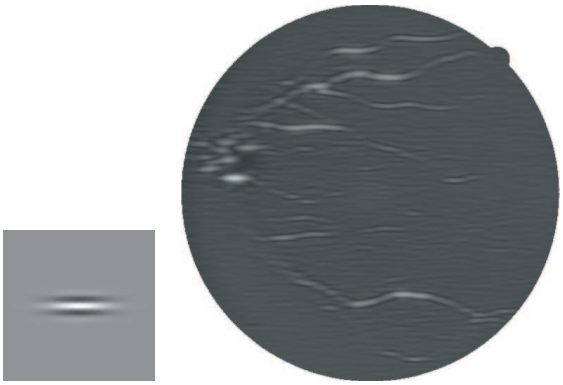
O processo de treinamento começa com a criação do conjunto de treinamento. As imagens de treinamento têm suas características extraídas e associadas aos rótulos, que são obtidos a partir de segmentações manuais das mesmas imagens, formando um conjunto de amostras. Um subconjunto das amostras é escolhido aleatoriamente e utilizado para treinar o classificador que irá estimar as funções densidade probabilidade das duas classes possíveis: C_1, para pixels que representam vasos sangüíneos, e C_2, para pixels que não representam vasos.

As funções de densidade de probabilidade (verossimilhanças) são descritas como combinações lineares de funções gaussianas:

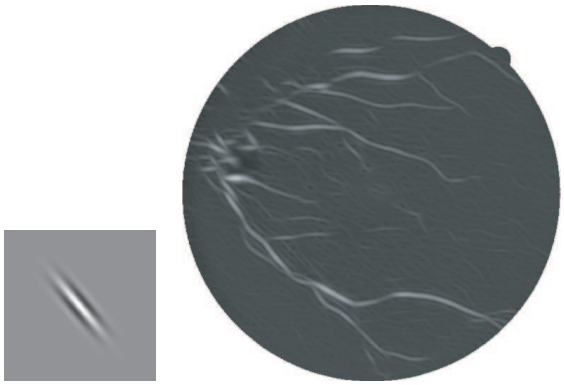
$$p(v|C_i) = \sum_{j=1}^{k_i} p(v|j, C_i) P_{ij} \tag{3}$$

onde k_i é o número de gaussianas que modelam a verossimilhança da classe C_i, v é o vetor de características, $p(v|j, C_i)$ é uma distribuição gaussiana d-dimensional (onde d representa a dimensão do vetor de características) e P_j é o peso da gaussiana j.

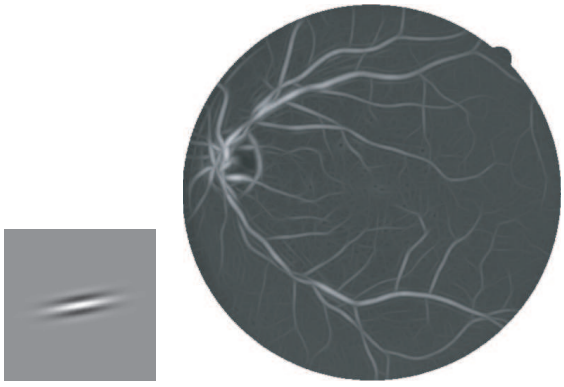
Para estimar as funções densidade probabilidade das classes, o algoritmo *Expectation-Maximization* é utilizado [8]. Utilizando as amostras do conjunto de treinamento, o algoritmo estima os parâmetros



(a) Respostas acumuladas para a aplicação da transformada wavelet com $\theta = 0°$



(b) Respostas acumuladas para a aplicação da transformada wavelet até $\theta = 50°$



(c) Respostas acumuladas para a aplicação da transformada wavelet até $\theta = 170°$

Figura 4: Representação da wavelet de Gabor e as respostas de módulo máximo acumuladas para cada etapa

(a) Imagem original (b) Segmentação Manual 1 (c) Segmentação Manual 2

Figura 5: Segmentações manuais de uma imagem do fundo da retina. As imagens foram obtidas do banco de imagens DRIVE

e os pesos das gaussianas ao mesmo tempo em que encontra um máximo local para a função de verossimilhança sendo estimada.

O algoritmo *Expectation-Maximization* é um algoritmo iterativo e cada iteração é composta de duas etapas: uma etapa de estimativa e uma etapa de maximização. Com cada iteração, a estimativa da função de verossimilhança é melhorada e o algoritmo pára quando a melhora se torna suficientemente pequena ou após ter executado um número de iterações pré-determinado.

2.2.2 Classificação

Com o classificador já treinado, a fase de classificação se torna algo muito simples e pode ser realizada de forma eficiente. Para realizar a classificação, cada pixel tem seu vetor de características gerado e submetido à regra de decisão do classificador.

O classificador bayesiano com modelo de mistura gaussiana utiliza a seguinte regra para a classificação:

$$\begin{aligned} &\text{Escolha } C_1 \text{ se } p(v|C_1)P(C_1) > p(v|C_2)P(C_2); \\ &\text{caso contrário, escolha } C_2 \end{aligned} \tag{4}$$

onde v é um vetor de características, $p(v|C_i)$ representa a verossimilhança da classe i que foi obtida na fase de treinamento e $P(C_i)$ é estimado pela proporção de amostras da classe i no conjunto de treinamento.

3 Desenvolvimento do Software

3.1 Planejamento

Inicialmente, alguns requisitos básicos foram definidos e influenciaram a escolha das tecnologias utilizadas para o desenvolvimento do software. Basicamente, o software deveria poder ser executado na plataforma Windows e possivelmente em outras plataformas também. O processo de instalação deveria ser simples assim como a interface do software com o usuário. Um outro ponto importante foi o de desenvolver o software de maneira que ele pudesse ser modificado facilmente para adicionar novas funcionalidade e também para testar novos métodos.

Levando em conta esses fatores, foram realizados alguns testes com as linguagens Python, C e C++. Das três linguagens, a linguagem C foi logo descartada por não ser uma linguagem orientada a objetos, o que tornaria bem difícil manter a extensibilidade do código. Em seguida, a linguagem Python foi descartada por precisar ser instalada de forma não trivial na plataforma Windows. Dessa forma, a linguagem C++ foi escolhida por ser uma linguagem bastante eficiente e também por permitir atingir os objetivos desejados com certa facilidade.

Como idéia inicial, pensamos em desenvolver o software como um plug-in para o GIMP (`http://www.gimp.org`), um software livre para trabalhar com imagens, pela possibilidade de utilizar os seus diversos plug-ins já existentes para implementar o método de segmentação mais facilmente. Ao fazer alguns testes, os plug-ins mostraram-se pouco configuráveis, disponibilizando parâmetros muito limitados, não permitindo realizar as operações necessárias para a implementação do método. Também havia uma certa falta de informações sobre como desenvolver um plug-in para o GIMP, fazendo com que muito tempo fosse perdido na busca de informações sobre o assunto em listas de discussão. Por esse motivo, o software foi desenvolvido de forma a funcionar como um aplicativo independente.

Para o desenvolvimento da interface gráfica, a biblioteca GTK foi utilizada. Ela foi escolhida inicialmente principalmente por poder ser integrada facilmente com o GIMP mas permaneceu como escolha definitiva mesmo após o GIMP não ser mais utilizado.

Finalmente, a biblioteca OpenCV foi escolhida para fazer o processamento de imagens por possuir implementações eficientes para todas as funções necessárias para a implementação do método escolhido.

3.2 Arquitetura do Software

3.2.1 Modelagem do método de segmentação

Para a implementação do método os fatores mais importantes foram a necessidade de criar uma arquitetura que permitisse a modularização das etapas do algoritmo de segmentação e a

extensibilidade.

O ponto de partida para a escolha da arquitetura foi a separação das várias etapas do método utilizado. Foram identificadas várias etapas as quais poderiam ser realizadas separadamente:

- **Geração da Máscara -** Etapa responsável pela detecção e criação da máscara que determina a região de interesse da imagem.
- **Extensão Artificial -** Etapa responsável pela criação da extensão artificial da imagem com máscara.
- **Geração de Características -** Etapa responsável pela geração do conjunto de características da imagem extendida.
- **Classificação -** Etapa responsável pela classificação da imagem com base no conjunto de características.

Podemos notar que em cada etapa um objeto é recebido, transformado e então devolvido. Além disso, todas as etapas, exceto a primeira, precisam do resultado da etapa anterior para poderem realizar sua função. Por esse motivo, o algoritmo de segmentação pôde ser modelado utilizando o padrão **Pipes and Filters** [3].

Um *filter* (filtro) possui um tipo de entrada e um tipo de saída e tem a função de transformar o tipo de entrada no tipo de saída. Dessa forma, podemos representar cada uma das etapas descritas acima como um filtro. Por exemplo, o filtro de geração de características transforma a imagem estendida em um conjunto de características.

Uma *pipeline* (linha de montagem) é formada pela ligação de dois ou mais filtros. Dois filtros podem ser ligados se o tipo de saída do primeiro filtro for igual ao tipo de entrada do segundo filtro e nesse caso os filtros são compatíveis. Nesse caso, temos uma *pipeline* com dois filtros que transforma o tipo de entrada do primeiro filtro no tipo de saída do segundo filtro. A figura 6 demonstra a composição de filtros para formar uma *pipeline*.

A codificação do padrão foi feita utilizando-se uma versão um pouco simplificada da implementação descrita em [4] (Figura 7). Basicamente, essa implementação define uma classe `Filter` que é uma classe gabarito (template) em C++. Essa classe possui dois parâmetros de gabarito que representam o tipo de entrada e o tipo de saída do filtro. A classe gabarito também define implementações para os método `input` e `attach`. O método `input` recebe o tipo de entrada, invoca o método para transformá-lo no tipo de saída e finalmente invoca o método `input` no próximo filtro da *pipeline* (se ele existir) utilizando o tipo de saída como entrada. O método `attach` faz uma conexão com um outro filtro compatível, armazenando uma referência interna para esse filtro.

Para utilizar a classe gabarito `Filter`, uma classe precisa apenas definir os dois parâmetros do gabarito e implementar o método `transform` que recebe o tipo de entrada como parâmetro e devolve o tipo de saída. Uma vantagem da implementação escolhida é que ela permite que a conexão entre os filtros seja verificada em tempo de compilação, isto é, conexões entre filtros com tipos incompatíveis geram erros de compilação. Como vantagem adicional, essa implementação permite que as *pipelines* sejam criadas e modificadas em tempo de execução.

Para manter os filtros independentes uns dos outros foi necessário definir tipos genéricos que poderiam ser utilizados para a comunicação entre os filtros. Os tipos genéricos definidos na implementação foram: o tipo `Imagem`, que é uma representação genérica de uma imagem com três canais de cores, e o tipo `Conjunto de Características`, que representa o conjunto de todos os vetores de características dos pixels de uma imagem. Essa abordagem também permite que os filtros que utilizem bibliotecas diferentes possam se comunicar mesmo que utilizem representações diferentes para uma imagem, por exemplo. Um exemplo de utilização na implementação é o filtro de geração de máscara que recebe uma imagem e devolve uma imagem com uma máscara anexada. Internamente, o filtro transforma o tipo `Imagem` para o tipo de imagem utilizado pela biblioteca OpenCV. Após a geração da máscara, o resultado é convertido novamente para o tipo genérico `Imagem` que pode ser utilizado pelo filtro seguinte, mesmo que este utilize uma outra biblioteca para trabalhar com imagens.

As vantagens da utilização do padrão e da implementação escolhida são muitas: primeiramente, os processos de treinamento e classificação podem ser modelados através de *pipeline*; cada filtro é independente e pode ser modificado sem alterar o restante do código; novos filtros podem ser criados e conectados a uma *pipeline* para adicionar novas etapas ao método.

Para a implementação do algoritmo de segmentação foram definidos quatro tipos de filtro correspondentes às quatro etapas descritas anteriormente. Num primeiro instante, uma *pipeline* correspondente às três primeiras etapas é utilizada para gerar as características das imagens de teste para o treinamento do classificador. Depois disso, uma outra *pipeline*, com todas as quatro etapas, é utilizada para fazer a classificação das imagens.

3.2.2 Independência de Bibliotecas

Um dos fatores mais importantes na definição da arquitetura do software foi a necessidade de manter o software desacoplado das bibliotecas utilizadas pois no futuro talvez fosse interessante ou trocar de bibliotecas ou utilizar implementações próprias para as funções utilizadas dessas bibliotecas.

Uma maneira de atingir esse objetivo com relação a interface gráfica foi a utilização de um padrão que separa o software em dois grandes módulos: o *modelo*, que inclui todo o processo de segmentação, e a *visão*, que inclui toda a implementação específica da interface do usuário. O

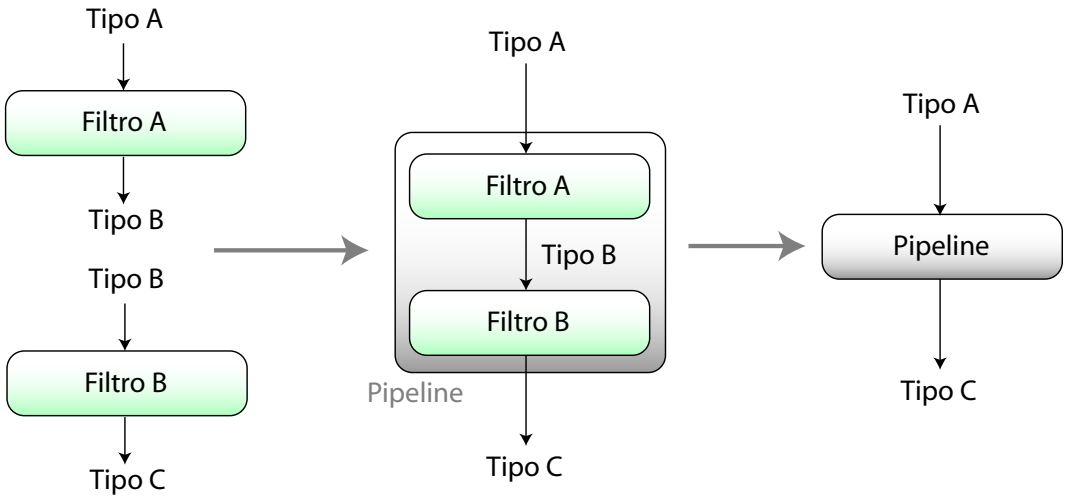


Figura 6: Composição de dois filtros para formar uma *pipeline*. A *pipeline* resultante é equivalente a um filtro que transforma o tipo A no tipo C.

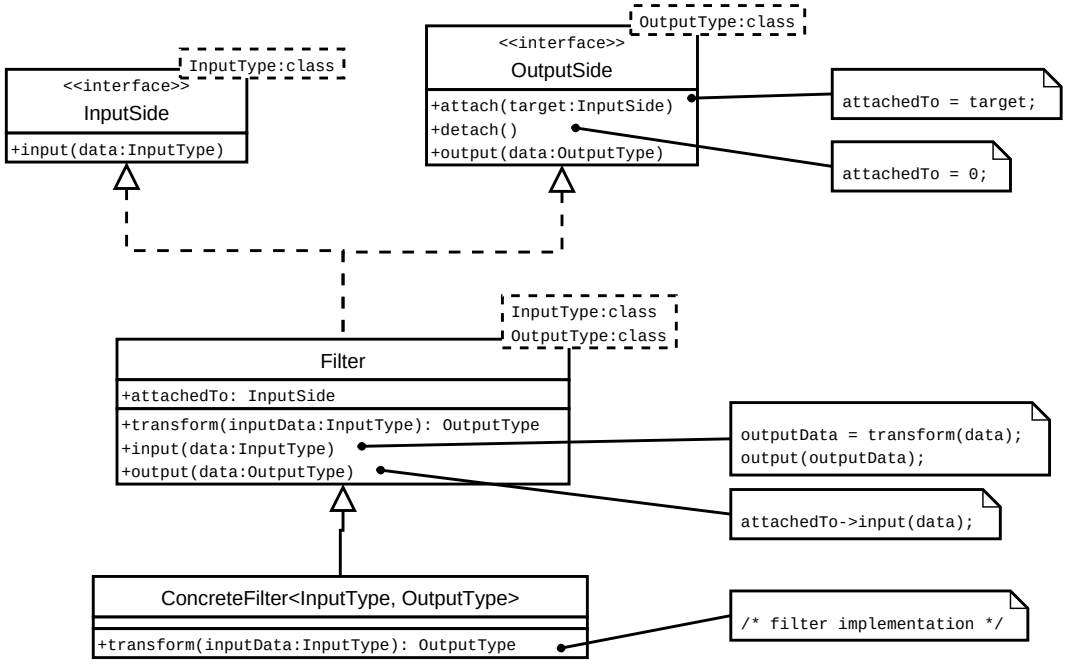


Figura 7: Diagrama UML da implementação do padrão Pipes and Filters

modelo não depende de forma alguma da visão pois este não faz referência alguma à visão utilizada. Já a visão utiliza o modelo de segmentação e fornece a interface que permite que o usuário possa configurar o modelo para realizar as ações desejadas. Com essa separação simples, o processo de segmentação se torna independente da biblioteca utilizada para fazer a interface gráfica. Além disso, novas visões podem ser implementadas para permitir que o software possa ser utilizado de formas diferentes. Como exemplos, podemos fazer uma visão que permite o uso do software através da linha de comandos. Também podemos fazer uma visão que permite utilizá-lo através de uma interface web. Para todas essas visões o modelo utilizado é o mesmo.

Um outro tipo de separação foi feita para tornar o algoritmo de segmentação independente da biblioteca de processamento de imagens. Este outro tipo de separação foi mais complexo pois muitas funções específicas de uma única biblioteca foram utilizadas para a implementação do algoritmo de segmentação. Como essas funções variam muito entre as bibliotecas e nem todas elas implementam as mesmas funções, não é trivial definir uma interface comum a todas elas. A solução adotada foi a de manter implementações separadas para cada biblioteca utilizada. O uso descuidado desse método pode fazer com que o código se torne muito específico, com várias referências à bibliotecas específicas espalhadas pelo código. Para evitar que isso acontecesse, a abordagem utilizada foi o uso do padrão **Abstract Factory** [2].

Uma `Abstract Factory` é uma classe abstrata que disponibiliza serviços e recursos em forma de métodos. No contexto do software, podemos citar métodos para salvar e carregar imagens que dependam de alguma biblioteca específica como exemplos de serviços. Exemplos de recursos incluem instâncias de classes com implementações específicas das várias etapas do algoritmo de segmentação.

Na implementação do software, uma classe abstrata `Resource Factory` foi definida com métodos para carregar e salvar imagens, e métodos para fornecer diversos recursos, entre eles os filtros que realizam as várias etapas do processo de segmentação. Além disso, uma classe `OpenCVFactory` foi definida herdando as definições da classe `Resource Factory`. A implementação da `OpenCVFactory` conhece as implementações específicas da biblioteca OpenCV e é capaz de fornecer as implementações para os métodos definidos na `Resource Factory`.

Para utilizar essas classes foi criado um esqueleto do algoritmo de segmentação. Esse esqueleto não faz referências diretas à nenhuma implementação específica. Sempre que o algoritmo precisa utilizar algum recurso que dependa de alguma biblioteca, ele utiliza uma instância pré-definida da classe `Resource Factory`. Por exemplo, sempre que o algoritmo precisa abrir uma imagem, ele o faz utilizando o método de carregar imagem disponibilizado na instância única da classe `Resource Factory`. Essa instância única é definida uma única vez na inicialização do software e é feita instanciando alguma implementação da `Resource Factory`, como por exemplo, uma `OpenCVFactory`. Dessa forma, podemos trocar a biblioteca utilizada simplesmente instanciando implementações diferentes da `Resource Factory` na inicialização do software.

3.3 Tecnologias Utilizadas

3.3.1 GTK

O GTK (the GIMP Toolkit) é um software livre multi-plataforma utilizado para desenvolver interfaces gráficas. Atualmente existem versões da biblioteca para várias linguagens como Perl, Python e C/C++. Uma das grandes vantagens em se utilizar a biblioteca é a grande quantidade de informações disponíveis sobre o desenvolvimento de aplicações usando o GTK. O próprio GTK possui uma documentação bem completa que é complementada pelas listas de discussões e pela grande quantidade de softwares de código aberto que a utilizam. Um bom exemplo é o software livre GIMP que é utilizado para manipulação de imagens e que foi desenvolvido utilizando o GTK.

O uso do GTK é bastante simples e basicamente se resume na criação de *widgets* (janelas, botões, imagens, etc.) e na manipulação dos sinais gerados e emitidos pelos *widgets*. Os sinais representam eventos como um clique do mouse sobre um botão ou uma alteração na seleção em uma tabela. Esses sinais podem ser associados a funções que são executadas no momento que os sinais são emitidos/recebidos.

Um grande problema enfrentado na utilização do GTK foi o gerenciamento de memória. Às vezes é um pouco difícil entender quando a memória utilizada por um *widget* é liberada. Além disso é muito fácil esquecer de liberar a memória utilizada. Um exemplo de um vazamento de memória que pode ocorrer facilmente: esquecer de liberar a memória utilizada por uma string devolvida por algum *widget*. Levando em conta que essa é uma operação bem comum (por exemplo, obter o valor digitado em uma caixa de texto) a ocorrência de um vazamento desse tipo não é incomum.

3.3.2 Glade

Uma das tarefas mais trabalhosas ao se utilizar o GTK é o design das janelas, isto é, o posicionamento dos *widgets*, a definição de tamanhos e a escolha de fontes na criação de uma janela. Para facilitar esta etapa o software livre Glade (`http://glade.gnome.org`) foi desenvolvido. O Glade é um editor visual de interfaces gráficas para o GTK. As interfaces podem ser desenhadas com o uso de uma barra de ferramentas e os *widgets* podem ter seus parâmetros editados facilmente com o uso de uma janela de propriedades. O resultado final pode ser salvo em formato XML que posteriormente pode ser utilizado em um software qualquer através da biblioteca libglade (`http://www.jamesh.id.au/software/libglade/`).

3.3.3 OpenCV

A OpenCV (Open Computer Vision Library) é uma biblioteca multi-plataforma que disponibiliza vários algoritmos utilizados na área de visão computacional. Por ser uma biblioteca mais voltada para uso em aplicações em tempo real, as implementações são bastante eficientes. A documentação disponível é um pouco incompleta e é um pouco difícil encontrar mais informações além da documentação oficial.

As grandes contribuições do uso da OpenCV no software foram os vários algoritmos de processamento de imagens, como dilatação, erosão, convolução e a transformada de Fourier. Também foram utilizados as funções para manipulação de matrizes que incluem as várias operações e outras funções como o cálculo de determinante e inversa.

Assim como no GTK, a liberação da memória utilizada pela OpenCV também deve ser feita manualmente. O grande problema é que normalmente aqui os vazamentos são bem maiores pois a quantidade de memória alocada ao trabalhar com imagens e matrizes é bem maior.

4 Resultados obtidos

Após alguns meses de trabalho, o software para segmentação de imagens de retina se encontra em estado funcional. O software final é portável e pode ser utilizado em diversas plataformas contanto que as bibliotecas utilizadas estejam disponíveis para a plataforma alvo.

Atualmente o software permite fazer a segmentação de imagens coloridas e também de imagens de fluorescência angiográfica. Para isso o usuário precisa treinar um classificador que pode ser salvo e utilizado posteriormente sem a necessidade de repetir o treinamento. Todo o processo de treinamento pode ser configurado, isto é, o usuário pode criar o conjunto de treinamento e selecionar as características que serão usadas. Tendo um classificador treinado, o usuário pode classificar uma ou mais imagens de forma bem simples. As imagens segmentadas são salvas em um local especificado pelo usuário e podem ser visualizadas no próprio software após a fase de classificação. As figuras 8(a),8(b),8(c) e 8(d) mostram algumas screenshots do software em execução.

Foram realizados alguns testes de desempenho em comparação ao protótipo que foi utilizado como base para o desenvolvimento. O primeiro teste foi realizado para avaliar o tempo de execução do algoritmo para a geração de um classificador. O classificador gerado nos testes utiliza o canal verde invertido e as respostas da transformada wavelet para as escalas de 2, 3, 4 e 5 pixels como características. O número de amostras utilizadas foi 10000 e apenas uma imagem de teste foi utilizada no treinamento. A tabela 1 mostra os tempos de execução variando-se o número de gaussianas k utilizadas no modelo de mistura gaussiana.

O computador utilizado nos testes possuía um processador AMD Athlon XP 2600 com 1GB de memória RAM. As implementações foram testadas no sistema operacional Windows.

	Implementação em C++	Implementação em Matlab
$k = 1$	19s	106s
$k = 5$	23s	175s
$k = 10$	28s	222s
$k = 20$	36s	317s
$k = 40$	54s	512s

Tabela 1: Comparação dos tempos de execução do treinamento de um classificador

Os tempos de execução são formados por uma parcela fixa que representa o tempo de geração das características da imagem de teste e por uma parcela variável que depende dos parâmetros do classificador. A parcela fixa pode ser estimada pelo tempo de execução dos algoritmos com $k = 1$ quando a parcela de tempo utilizada no treinamento é desprezível. Dessa forma, podemos observar que o tempo de execução do treinamento aumenta proporcionalmente ao parâmetro k. Isso é devido à complexidade do algoritmo *Expectation-Maximization* que é $O(ksd)$ para cada iteração, com k

representando o número de gaussianas usadas no modelo, s representando o número de amostras utilizadas no treinamento e d representando a dimensão do vetor de características. Também podemos concluir que grande parte da melhora na versão C++ se deve a uma geração mais rápida das características da imagem.

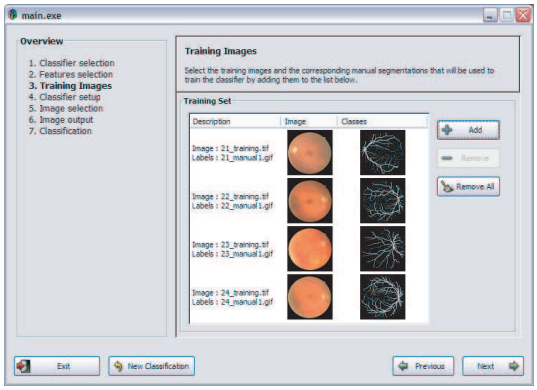
No segundo teste, os algoritmos foram comparados quanto ao tempo de execução para a realização da classificação de uma única imagem. Os classificadores utilizados foram os classificadores gerados no primeiro teste. A tabela 2 mostra os resultados do teste.

	Implementação em C++	Implementação em Matlab
$k = 1$	20s	100s
$k = 5$	19s	102s
$k = 10$	20s	104s
$k = 20$	22s	110s
$k = 40$	25s	120s

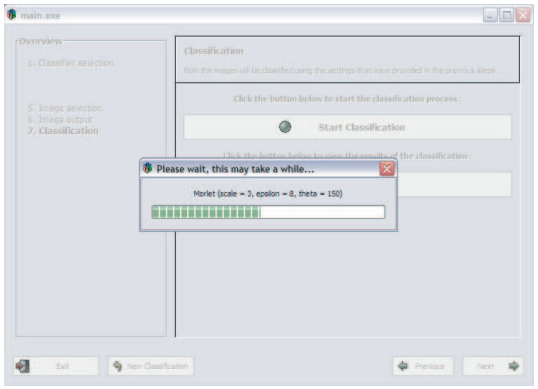
Tabela 2: Comparação dos tempos de execução da classificação de uma imagem

A complexidade do algoritmo de classificação é $O(knd)$. A grande diferença em relação ao tempo de execução da fase de treinamento está no número de pixels a serem classificados n já que o tempo de geração das características é o mesmo e k e d dependem do classificador usado. Na fase de classificação, o número de amostras utilizadas não ultrapassa o número de pixels da imagem já que somente os pixels da região de interesse são utilizados. Na fase de treinamento, as amostras também são extraídas da região de interesse mas elas são extraídas de todas as imagens que compõem o conjunto de treinamento. Assim, mesmo que a quantidade de amostras extraídas por imagem seja menor que na fase de classificação, como o conjunto de treinamento é composto por várias imagens, o número total de amostras normalmente é maior que o número de amostras utilizadas na fase de classificação.

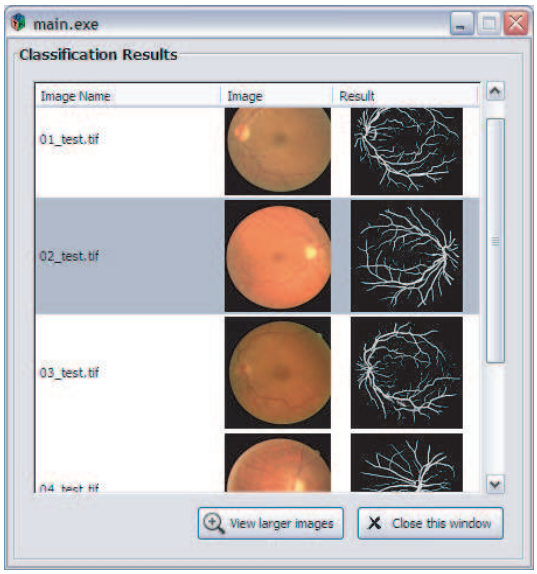
Apesar da fase de treinamento ser bastante demorada isso não chega a ser um problema já que os classificadores treinados podem ser salvos e distribuídos juntamente com o software evitando que um usuário precise fazê-lo. Ainda assim seria interessante buscar alguma implementação mais eficiente para o algoritmo de treinamento.



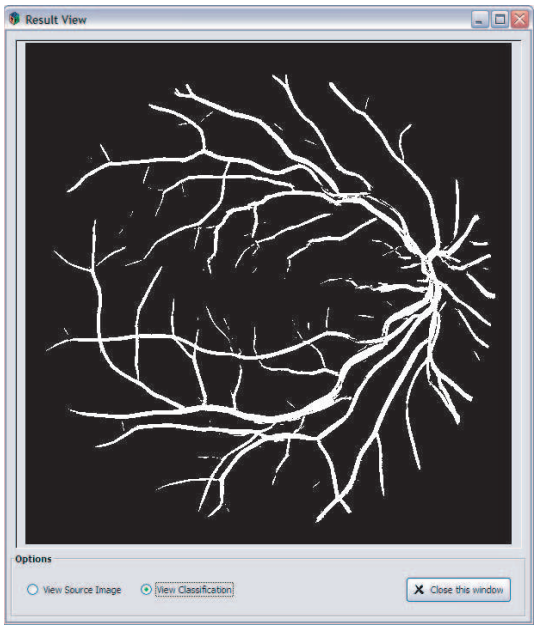
(a) Escolha do conjunto de treinamento



(b) Aguardando o término da fase de classificação



(c) Resultados da fase de classificação



(d) Visão detalhada do resultado da classificação

Figura 8: Algumas screenshots do software em execução

5 Desenvolvimento do Trabalho

5.1 Conceitos e tecnologias estudadas

Para desenvolver o trabalho de formatura, eu precisei estudar diversos conceitos e também tecnologias desconhecidas até então para mim. Como eu não cursei nenhuma disciplina que tratasse especificamente de problemas de visão computacional, foi fundamental estudar alguns conceitos de processamento de imagens como os operadores morfológicos, convolução, detecção de bordas, a transformada de Fourier e a transformada wavelet. Também precisei revisar vários conceitos de estatística e probabilidade que foram bastante utilizados para codificar o classificador bayesiano.

Também precisei aprender a utilizar as bibliotecas GTK e OpenCV. O uso da biblioteca GTK não foi difícil por ela ser amplamente documentada e utilizada na comunidade do software livre. A biblioteca OpenCV, no entanto, possui uma documentação incompleta e muitas vezes não muito informativa. Muitas vezes eu precisei procurar códigos-fonte que utilizassem a biblioteca para entender como algumas funções funcionavam. Finalmente, o uso do software Glade para desenvolvimento de interfaces gráficas foi uma surpresa muito agradável. Após o término da codificação do software em modo texto, em menos de uma semana a interface gráfica estava pronta para ser usada graças ao uso do Glade.

5.2 Relação de conceitos/disciplinas importantes

Muitas das disciplinas que eu cursei tiveram alguma relação com o desenvolvimento do trabalho. As contribuições mais importantes são descritas a seguir:

- **Engenharia de Software e Laboratório de Programação Extrema -** Essas duas disciplinas permitiram a aplicação prática de vários conceitos adquiridos durante o curso. Essas foram as duas únicas disciplinas que permitiram trabalhar com grupos e projetos maiores. Além disso, muitos conceitos de programação orientada a objetos se tornaram bem mais claros após terem sido usados nessas disciplinas. Elas foram de grande ajuda no desenvolvimento da arquitetura do software já que eu já havia aplicado boa parte das técnicas e padrões utilizados no desenvolvimento do software durante o período em que eu as cursei.

- **Computação Gráfica -** Os conceitos de representação e armazenamento de imagens, a representação das cores e até mesmo algumas noções básicas de processamento de sinais foram de grande importância já que na maior parte do tempo as imagens precisavam ser trabalhadas em sua representação interna.

- **Cálculo, Probabilidade e Estatística -** Conceitos de todas essas disciplinas foram utilizados principalmente no estudo e implementação da transformada wavelet contínua e também do classificador bayesiano.

5.3 Relacionamento com o grupo de trabalho

O trabalho de formatura foi desenvolvido junto ao grupo Creativision do IME-USP (`http://www.ime.usp.br/~cesar/creativision/`). O contato com o grupo se dava principalmente em reuniões que eram realizadas semanalmente. Essas reuniões foram muito úteis principalmente na fase de planejamento onde discutimos as tecnologias que seriam utilizadas para desenvolver o software. Depois eu passei a comparecer menos devido a conflitos de horário mas eu mantive o contato através de e-mail e através de reuniões em outros horários. Todos foram de grande ajuda e contribuíram para que o trabalho pudesse ser realizado.

5.4 Conclusões finais

Um dos fatores que me levou a escolher esse projeto como o meu trabalho de formatura foi a possibilidade de desenvolver um software que pudesse ter algum impacto social. Acho isso algo muito importante, principalmente na área de Ciência de Computação. Fico bastante feliz em ter produzido um software que poderá ser utilizado, modificado e que poderá crescer de modo que venha ajudar alguém direta ou indiretamente.

De modo geral, eu fiquei bastante satisfeito com o resultado obtido com o trabalho. Eu pude aprender muito e também pude aplicar muitos dos conceitos vistos no curso. Uma das poucas frustrações encontradas ao longo do desenvolvimento foi o tempo gasto com as sessões de depuração para encontrar vazamentos de memória. Esse tempo poderia ter sido melhor utilizado implementando novas funcionalidades no software.

Ainda pretendo trabalhar no software para adicionar a capacidade de trabalhar com outros tipos de imagem e também fazer algumas alterações na interface gráfica para facilitar o uso. Além disso, talvez fosse interessante procurar alguma implementação mais eficiente para o algoritmo *Expectation-Maximization* e tentar implementar um método semi-automatizado de classificação onde o usuário pode marcar alguns pixels como *vaso* e o classificador, com base nessas amostras, é treinado e utilizado para classificar a mesma imagem.

Referências

[1] J. P. Antoine, P. Carette, R. Murenzi, and B. Piette. Image analysis with two-dimensional continuous wavelet transform. *Signal Processing*, 31:241–272, 1993.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Padrões de Projeto*. Bookman, Porto Alegre, 2000.

[3] R. Meunier. The pipes and filters architecture. pages 427–440, 1995.

[4] E. J. Posnak, R. G. Lavender, and H. M. Vin. Adaptive pipeline: an object structural pattern for adaptive applications. *The 3rd Pattern Languages of Programming conference*, September 1996.

[5] J. V. B. Soares, J. J. G. Leandro, R. M. Cesar-Jr., H. F. Jelinek, and M. J. Cree. Retinal vessel segmentation using the 2-D Gabor wavelet and supervised classification. *IEEE Transactions on Medical Imaging*, 25:1214–1222, 2006.

[6] J. J. Staal, M. D. Abràmoff, M. Niemeijer, M. A. Viergever, and B. van Ginneken. Ridge based vessel segmentation in color images of the retina. *IEEE Transactions on Medical Imaging*, 23:501–509, 2004.

[7] H. R. Taylor and J. E. Keeffe. World blindness: a 21st century perspective. *British Journal of Ophthalmology*, 85:261–266, 2001.

[8] S. Theodoridis and K. Koutrumbas. *Pattern Recognition*. Academic Press, USA, first ed., 1999.