

BRUNO POLICARPO TOLEDO FREITAS

*Desenvolvimento de uma aplicação de
Processamento de Imagens utilizando a
biblioteca lili2*

Porto Alegre

Junho de 2011

BRUNO POLICARPO TOLEDO FREITAS

*Desenvolvimento de uma aplicação de
Processamento de Imagens utilizando a
biblioteca lili2*

Orientador:

Altamiro Amadeu Susin

Co-orientador:

Letícia Vieira Guimarães

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

Porto Alegre

Junho de 2011

AGRADECIMENTOS

À minha mãe, por ter me dado amor e carinho durante esses 6 anos e meio.
Ao meu pai, por ter suado muito para que eu tivesse a oportunidade que agarrei com as
duas mãos.
Ao meu irmão, que buscou mostrar-me a sua maneira de ser feliz, pensando que com ela
eu também seria feliz.
Ao demais familiares, que mesmo longe contribuíram com energias positivas.
Aos Mestres que conheci durante o curso, por terem feito com que eu aprendesse muito
mais do que simplesmente Engenharia e Computação.
Aos meus "chefes" do LaPSI e CPD, por terem compreendido como eu melhor funcionava.
Aos meus amigos, por terem ajudado-me a relaxar enquanto trilhava esta árdua
caminhada
A todos, um sincero "obrigado".

Sumário

Resumo

Abstract

Lista de abreviaturas e siglas

Lista de Figuras

1	Introdução	p. 9
1.1	Definição de Imagem Digital	p. 10
1.2	Ferramentas de Processamento Digital de Imagens	p. 11
2	Alterações na arquitetura da lili2	p. 13
2.1	Arquitetura da biblioteca lili2	p. 13
2.1.1	Mapas de tons de cinza	p. 13
2.1.2	Mapa binário	p. 14
2.1.3	Mapas numéricos	p. 15
2.2	Implementação do algoritmo	p. 16
3	Ambiente de desenvolvimento	p. 19
3.1	IDE	p. 19
3.2	Bibliotecas utilizadas	p. 22
	wxWidgets -	p. 22
	OpenCV -	p. 22

4 A aplicação	p. 23
4.1 Interface	p. 23
4.2 A Lógica por trás da interface	p. 25
4.3 Resultado das amostras	p. 30
4.3.1 Gol branco se aproximando lentamente	p. 30
4.3.2 Gol branco com reflexo	p. 32
4.3.3 Chevete Azul com outro carro passando	p. 33
4.3.4 Chevete Azul com reflexo	p. 34
4.3.5 Chevete Azul rápido, com reflexo	p. 35
5 Conclusão	p. 36
6 O Futuro	p. 38
Referências	p. 39
Glossário	p. 40

Resumo

O projeto lili2 surge como uma ferramenta incremental no ensino de processamento de imagens: a cada nova turma da disciplina, novas funções são adicionadas à biblioteca, refletindo o desenvolvimento intelectual dos alunos. Para isso, é utilizada a linguagem de programação C++, visando trazer os alunos mais próximos do ambiente onde ocorre o desenvolvimento de aplicações de PDI, e é utilizada apenas a biblioteca padrão do C++, visando simplificar o processo de desenvolvimento. Além disso, a biblioteca também almeja ser utilizada em aplicações embarcadas, utilizando para isso templates, uma feature do C++ que permite implementar compilação condicional.

Como estudo de caso de uso da biblioteca, foi implementado um detector de veículos, utilizando para isso um algoritmo baseado naquele desenvolvido por Alexandre Haupt (4). O motivo dessa escolha se deve principalmente por ser um trabalho feito no LaPSI e por ter sido utilizada a primeira versão da biblioteca, escrita em C.

Nas próximas páginas, serão mencionadas, primeiramente, as adições que foram feitas à organização lógica da biblioteca. Isso é importante pois deseja-se que um método de PDI seja "encaixado" da forma mais genérica possível dentro da biblioteca. Por exemplo, perguntas como "faz sentido o algoritmo x funcionar para y e z?" são constantemente feitas para que sejam adicionados novos templates que possam generalizar o "algoritmo x" para as classes "y" e "z".

Depois, será discutida questões relativas à configuração do ambiente de desenvolvimento: IDE e bibliotecas utilizadas além da lili2. Finalmente, chegaremos na aplicação: Seus principais objetivos, a interface, e por último como ela foi logicamente criada.

Abstract

The project lili2 is an incremental tool to help the teaching of Image Processing: for each new class, new functions are added to the library, reflecting the learning of the students. For that, it is used the programming language C++, aiming to bring the students closer to the reality where the development of such applications happen, and only the C++ Standard library as a requirement, aiming to simplify its development. The library also aims to target embedded applications, using for that templates, a C++ feature that allows to implement conditional compilation.

As a Case Study of its use, it was implemented a Vehicle Detector, using an algorithm developed by Alexandre Haupt. The reason for that choice was because it was a work done on LaPSI and because it used the first version of the library, written in C.

On the next pages, it will be written which additions were made to the logic organization of the library. That is important because when we want to insert a new Image Processing method, we want it to be put on the most generic form possible. For instance, questions like "Is it possible for this algorithm x to work for both classes y and z " are important so we can add a new template generalizing that algorithm for these classes.

After that, it will be said a little about how the development environment was configured for the project: IDE and additional libraries used. Finally, the main application will be discussed: its main objectives, its interface, and then the hidden logic behind its implementation.

Lista de abreviaturas e siglas

PDI - Processamento De Imagens

LaPSI - LAboratório de Processamento de Sinais e Imagens

lili/lili2 - Lapsi Image processing LIbrary

IDE - Integrated Development Environment

Lista de Figuras

1	Imagem transmitida por telégrafo em 1929	p.9
2	Matriz bidimensional representando uma imagem digital. M representa o número de linhas da imagem e N o de colunas	p.10
3	Nova arquitetura dos mapas em tons de cinza dentro da lili2	p.14
4	Resultado do operador gradiente em um mapa com bordas com contraste muito alto. A máscara em b) está centrada no pixel marcado pela cor azul. Em c) podemos ver claramente o resultado dessa operação, fora da faixa 0-255 de um pixel em tons de cinza	p.15
5	Arquitetura dos Mapas Numéricos	p.17
6	Janela de configuração de variáveis globais no Codeblocks	p.21
7	Estado inicial da aplicação	p.24
8	Estado da aplicação após vídeo ser selecionado. A janela principal é redimensionada de forma a acomodar 2x as dimensões do frame do vídeo escolhido, a fim de ser exibido tanto o veículo detectado pelo algoritmo quanto passos intermediários do processamento	p.24
9	Aplicação rodando o vídeo e processando-o, exibindo o passo da binarização	p.25
10	Máquina de Estados da aplicação	p.26
11	Esta amostra não possuía nenhuma interferência significativa, portanto, o algoritmo se comportou muito bem, marcando com precisão o veículo desejado	p.31
12	Aqui ocorreu pela primeira vez o problema do reflexo, que é aparecer no mesmo frame capturado o veículo e seu reflexo de alguma forma. Aparece aqui um problema da escolha de implementação do algoritmo: ele é incapaz de decidir aquilo que é um veículo daquilo que não é. . .	p.32

- 13 Esta amostra começa com o final de um veículo saindo, depois entrando o Chevette azul. Aqui, o algoritmo demonstrou demorar um pouco para voltar a captar o Chevette, provavelmente devido ao fato de que havia o final de um carro anteriormente sendo seguido, por isso, atrapalhando a fase de correção da trajetória. p. 33
- 14 Esta amostra teve como interferência a presença de um reflexo do veículo. Aqui, sem mecanismos de detecção para decidir o que pode ser um veículo e o que não pode, a aplicação determinava que o Centro de Massa ficava sempre entre os 2 objetos, o carro e o refletido. p. 34
- 15 Esta amostra demonstrou que o algoritmo captura o veículo rapidamente, porém, de novo aparece o problema do reflexo, marcando o centro do veículo como entre ele e o seu reflexo. p. 35

1 *Introdução*

As primeiras aplicações que utilizavam técnicas de Processamento de Imagens datam nos anos 20, quando foram utilizadas técnicas de codificação para transmitir uma imagem da Inglaterra aos Estados Unidos utilizando cabos submarinos. Estas técnicas evoluíram gradualmente, codificando de 5 tons de cinza no início daquela década até 15 ao seu final.



Figura 1: Imagem transmitida por telégrafo em 1929

Embora fossem já consideradas técnicas de PDI, elas ainda não eram consideradas Digitais pois não existiam computadores naquela época. Ou seja, não existia ainda o termo Processamento Digital de Imagens, até mesmo porque, como hoje acontece, o Processamento Digital de Imagens requer tanto espaço de armazenamento como capacidade de processamento. Esse processo acabou agregando-se em ao desenvolvimento dos sistemas computacionais.

Os conceitos que permeiam hoje os nossos computadores foram introduzidos pela primeira vez por John Von Neumann na década de 1940, e consistiam em 2 princípios:

- Uma memória para guardar dados e um programa
- Uma Unidade Operacional e uma de Controle para o sequenciamento

Essas duas idéias correspondem a Unidade Central de Processamento, o coração dos computadores hoje em dia. O desenvolvimento dos computadores continuou avançando, tendo como alguns dos pontos chave a invenção do transistor, das linguagens de programação de alto nível e o avanço das técnicas de integração nos chips.

Essa evolução natural permitiu que em 1960 surgissem os primeiros computadores com capacidade de realizar Processamento Digital de Imagens propriamente dito. Junto com o nascimento do programa espacial americano, começaram a surgir as primeiras técnicas de restauração e melhorias de imagens, objetivando a melhora daquelas oriundas das primeiras sondas espaciais, que vinham distorcidas pela câmera de bordo. Desde então, o campo de Processamento de Imagens digitais vem evoluindo, sendo hoje utilizado nos mais diversos ramos do conhecimento humano, como medicina, geografia, astronomia, arqueologia, e muitos outros.

1.1 Definição de Imagem Digital

Uma Imagem Digital pode ser definida como sendo uma função bidimensional $f(x, y)$, onde cada coordenada possui um grau de luminosidade. Tal imagem pode ser então adquirida por, a título de exemplo, uma câmera digital, a qual realiza duas operações: amostragem, que consiste em digitalizar os valores x e y da imagem, e quantização, que é digitalizar as intensidades de cada par de coordenada. O resultado desses processos então gera uma matriz bidimensional, indexada pelos valores x e y com cada par correspondendo a uma intensidade amostrada.

$$f(x, y) = \begin{bmatrix} f(0, 0) & f(0, 1) & \cdots & f(0, N - 1) \\ f(1, 0) & f(1, 1) & \cdots & f(1, N - 1) \\ \vdots & \vdots & & \vdots \\ f(M - 1, 0) & f(M - 1, 1) & \cdots & f(M - 1, N - 1) \end{bmatrix}.$$

Figura 2: Matriz bidimensional representando uma imagem digital. M representa o número de linhas da imagem e N o de colunas

1.2 Ferramentas de Processamento Digital de Imagens

Com a Imagem Digital em mãos, capturada e guardada dentro do computador, surge mais um problema: como manipular essa imagem de forma fácil, permitindo que desenvolvamos um algoritmo de Processamento Digital de Imagens?

Em face desse problema, foram criadas ao longo desse último século ferramentas que possibilitassem essa manipulação. Essas ferramentas evoluíram quase no mesmo passo com que os métodos foram evoluindo e sendo criados, culminando no fato de que muitas delas resolvem uma gama gigantesca de problemas e também fossem utilizadas em outras áreas que se interseccionam como Processamento de Imagens, tais como Robótica e Visão Computacional. Um exemplo desse tipo de ferramenta é a OpenCV, que hoje conta com mais de 2500 algoritmos implementados e é usada em aplicações de alta performance no mundo inteiro.(11)

Um problema que essas ferramentas acabam tendo, porém, é o alto grau de complexidade que elas adquirem quanto mais elas evoluem. Por serem complexas, os sistemas acabam exigindo um grande esforço para o aprendizado das bibliotecas. Adicionalmente as aplicações podem ter outros tipos de requisitos, por exemplo, de ser carregadas em plataformas embarcadas. Visando principalmente esses 2 campos, o de ensino e o de plataformas embarcadas, o projeto lili consiste em implementar um ambiente de Ensino para Processamento de Imagens.

Para o ensino, seus principais objetivos são, então, trazer os alunos mais próximos da realidade onde tais aplicações são desenvolvidas ao mesmo tempo em que oferece uma plataforma os com recursos mais importantes dentro da área. Para este objetivo, escolheu-se utilizar C++ como linguagem de desenvolvimento, e apenas como requisito a biblioteca padrão do C++, visando simplificar o processo de desenvolvimento e adaptação fácil e rápida nos mais diversos ambientes.

Para aplicações embarcadas, busca-se baixo consumo e código reduzido. Sua utilização em plataformas embarcadas é baseada em templates, uma feature do C++ que permite generalizar operações para diversos tipos de imagens e também permite implementar

"compilação condicional", já que funções definidas em templates só são compiladas se forem chamadas dentro de uma aplicação.

Como estudo de caso para demonstrar a aplicação da biblioteca lili e para detectar a necessidade de novas funções, este trabalho implementa um algoritmo de detecção de veículos desenvolvido no LaPSI por Alexandre Haupt (4). Uma das características da biblioteca lili é que seu desenvolvimento é incremental: cada novo algoritmo ou método desenvolvido em função de um nova aplicação é integrado ao sistema para uso futuro.

Tal escolha se deve ao fato de que ele foi desenvolvido utilizando a primeira versão da lili, e também por ser um algoritmo desenvolvido, como já foi dito, dentro do Laboratório, além de motivações pessoais como talvez, no futuro, ajudar a descongestionar nossas vias de circulação pública já congestionadas por veículos vazios.

2 *Alterações na arquitetura da lili2*

A primeira parte do desenvolvimento da aplicação consistiu em trabalhar sobre a biblioteca de Processamento de Imagens lili2, visto que havia a necessidade de serem implementadas diversas operações necessárias.

Serão, primeiro, descritas as alterações feitas na arquitetura da biblioteca, para em seguida serem descritas como as operações necessárias foram implementadas.

2.1 **Arquitetura da biblioteca lili2**

Durante a fase de adição de novas classes e funcionalidades sempre há um cuidado em fazê-las de forma que as operações, que logicamente possam ser aplicadas a mais de um tipo de mapa, necessitem serem implementadas apenas uma vez, sendo este o principal motivo pelo qual a biblioteca é baseada em *templates*. Serão, então, mostradas quais as novas classes e funcionalidades adicionadas à lili2 para este trabalho, bem como a forma com que elas estão organizadas dentro do modelo de desenvolvimento utilizado.

2.1.1 **Mapas de tons de cinza**

Atualmente, a biblioteca lili2 possui dois mapas que representam tons de cinza : *LGrayMap*, que representa um tom de cinza através da média entre as intensidades RGB, e *LGrayMap765*, que representa através da soma entre as elas, visando aumentar a precisão do tom obtido. Portanto, é de se esperar que quaisquer algoritmos que trabalhem com tons de cinza devam funcionar sob qualquer um dos dois mapas. Porém, não existia na biblioteca uma maneira de implementar um algoritmo apenas uma vez e depois adaptá-lo para a classe desejada.

Para resolver esse impasse, foi criado mais um *template*, o *LBaseGrayMap*. Ele é derivado da classe-base que representa um mapa de imagem, a *LImgMap*, e tem o objetivo de servir como base para a implementação dos algoritmos que trabalham sob tons de cinza. Uma vez que os dois tipos de pixels são implementados utilizando o template *LBaseIntens* que, por sua vez, possui métodos para recuperar as intensidades máximas e mínimas do tipo em questão, as funções que dependem de tais propriedades puderam ser facilmente implementadas na classe recém-criada.

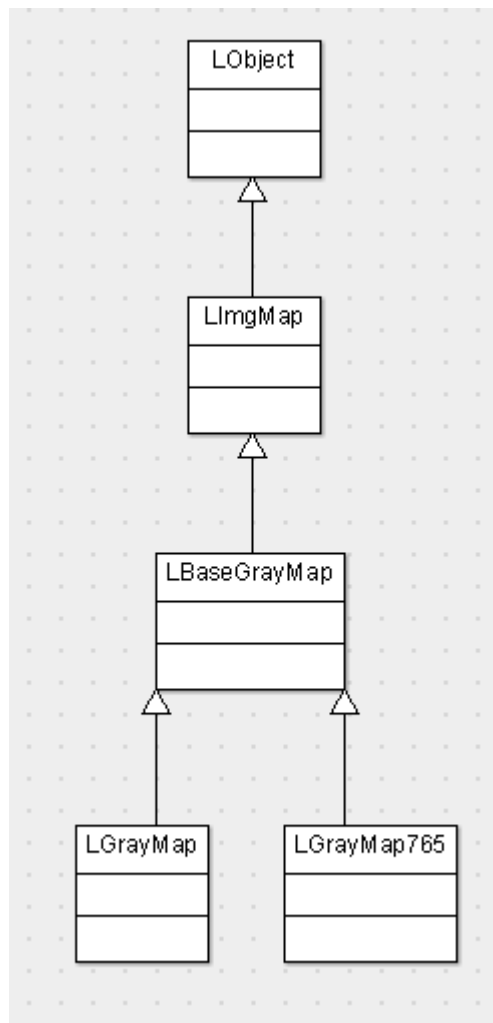


Figura 3: Nova arquitetura dos mapas em tons de cinza dentro da lili2

2.1.2 Mapa binário

As operações morfológicas muitas vezes operam sob mapas compostos de imagens binárias, ou seja, de valores 0 ou 1, representando uma área de interesse segmentada sob a qual se deseja trabalhar. Uma maneira que poderia ser utilizada para implementar tais

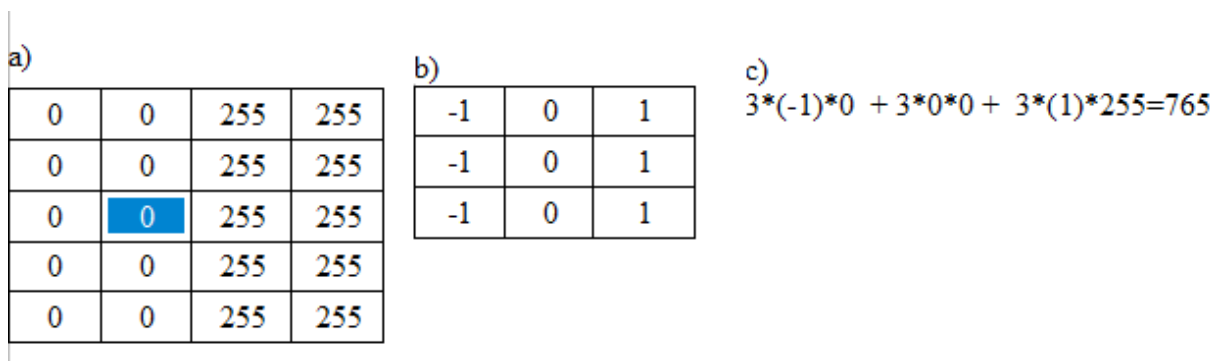


Figura 4: Resultado do operador gradiente em um mapa com bordas com contraste muito alto. A máscara em b) está centrada no pixel marcado pela cor azul. Em c) podemos ver claramente o resultado dessa operação, fora da faixa 0-255 de um pixel em tons de cinza

mapas poderia ser utilizando os mapas em tons de cinza, indicando que a intensidade 0 corresponderia a 0, e a 1 a 255.

Porém, uma implementação assim traria problemas lógicos dentro da biblioteca. Por exemplo, uma operação de binarização a partir de um limiar geraria na verdade um mapa que também seria de tons de cinza, e operações morfológicas de mesma natureza, como por exemplo a erosão e a dilatação; existem para ambas classes de imagens com definições bem distintas.

Foi, então, adicionado mais um mapa, denominado *LBinMap*, o qual foi definido como sendo um mapa de pixels binários, *LBinPix*, de tipo-base booleano podendo então assumir apenas os valores **true**, indicando intensidade 1, ou **false**, indicando 0.

2.1.3 Mapas numéricos

Os mapas numéricos existem para o cálculo de operações que resultam em valores fora da faixa de representação de pixels de imagens. Um exemplo de tal operação seria a detecção de bordas em uma imagem utilizando o operador Gradiente, como o de Sobel ou o de Prewitt: uma imagem em tons de cinza variando de 0 a 255 que tenha uma borda caracterizada por um conjunto de valores com intensidade baixa seguido por um conjunto com valores altos resultaria em um pixel com intensidade fora dessa faixa, assim como é mostrado na figura 4

Dentro da biblioteca lili2, existem no momento dois tipos de mapas: o mapa de números em ponto flutuante, *LDoubleMap*, e o mapa de números complexos *LComplexMap*. Assim como ocorria no caso dos mapas em tons de cinza, existem também operações que existem para os dois tipos de mapas e, mais uma vez, não havia uma maneira que permitisse a implementação de um algoritmo uma única vez para posterior adaptação a cada um dos mapas.

Por isso, foi adicionado mais um *template*, chamado de *LBaseNumMap*, para implementar as operações que fossem comuns aos mapas numéricos, e o *LSingleNumMap* para implementar operações que existam para os mapas em ponto flutuante e de inteiros. Além disso, como a última frase deixa implícito, foi criado o mapa de números inteiros, chamado de *LIntMap*. O objetivo foi o de utilizar as mesmas implementações da classe *LDoubleMap* para a deste último mapa, visando proporcionar maior desempenho para as operações numéricas de um procedimento de PDI em que seja aceitável a perda da precisão proporcionada pelo mapa *LDoubleMap* em prol de um maior desempenho.

Em suma, o mapa complexo agora é derivado diretamente da classe *LBaseNumMap*, e desta última é derivado o *template* *LSingleNumMap* e, por fim, são derivados os mapas *LDoubleMap* e *LIntMap*. Essa estrutura pode ser vista em 5

2.2 Implementação do algoritmo

Agora que as devidas mudanças dentro a estrutura da biblioteca lili2 foram mostradas, será desenvolvido como o algoritmo utilizado para a detecção de veículos foi implementado dentro da biblioteca. São elas:

1. **Converção para tons de cinza** - Esta operação já existe na biblioteca.
2. **Normalização** - A normalização foi implementada no *template* *LBaseGrayMap*, de forma que tanto os mapas *LGrayMap* e *LGrayMap765* pudessem se beneficiar dessa operação.
3. **Converte o frame para um Mapa em Ponto Flutuante** Esta operação também já existia na biblioteca.
4. **Filtro da média** - Esta operação também já existia na biblioteca. Infelizmente, não foi trabalhado a otimização deste algoritmo, o que poderia melhorar bastante o

algoritmo como um todo.

5. **Detecção de movimento** - Esta operação também já existia na biblioteca.
6. **Binarização** - O método estatístico exige antes a computação do Histograma da imagem. Uma classe responsável por calcular histogramas já existia na biblioteca, portanto, foi adicionado a ele um novo método, chamado de *GetThreshold*, o qual é responsável por realizar esse cálculo de Limiar baseado em estatística pura e simples.
7. **Fechamento** - Não existia nenhuma operação morfológica dentro da biblioteca, por isso, foi adicionado mais um mapa, o mapa binário, como já foi dito anteriormente. Nesse mapa então foi implementado as operações morfológicas mais comuns, dentre elas a erosão/dilatação e, conseqüentemente, a abertura e o fechamento.

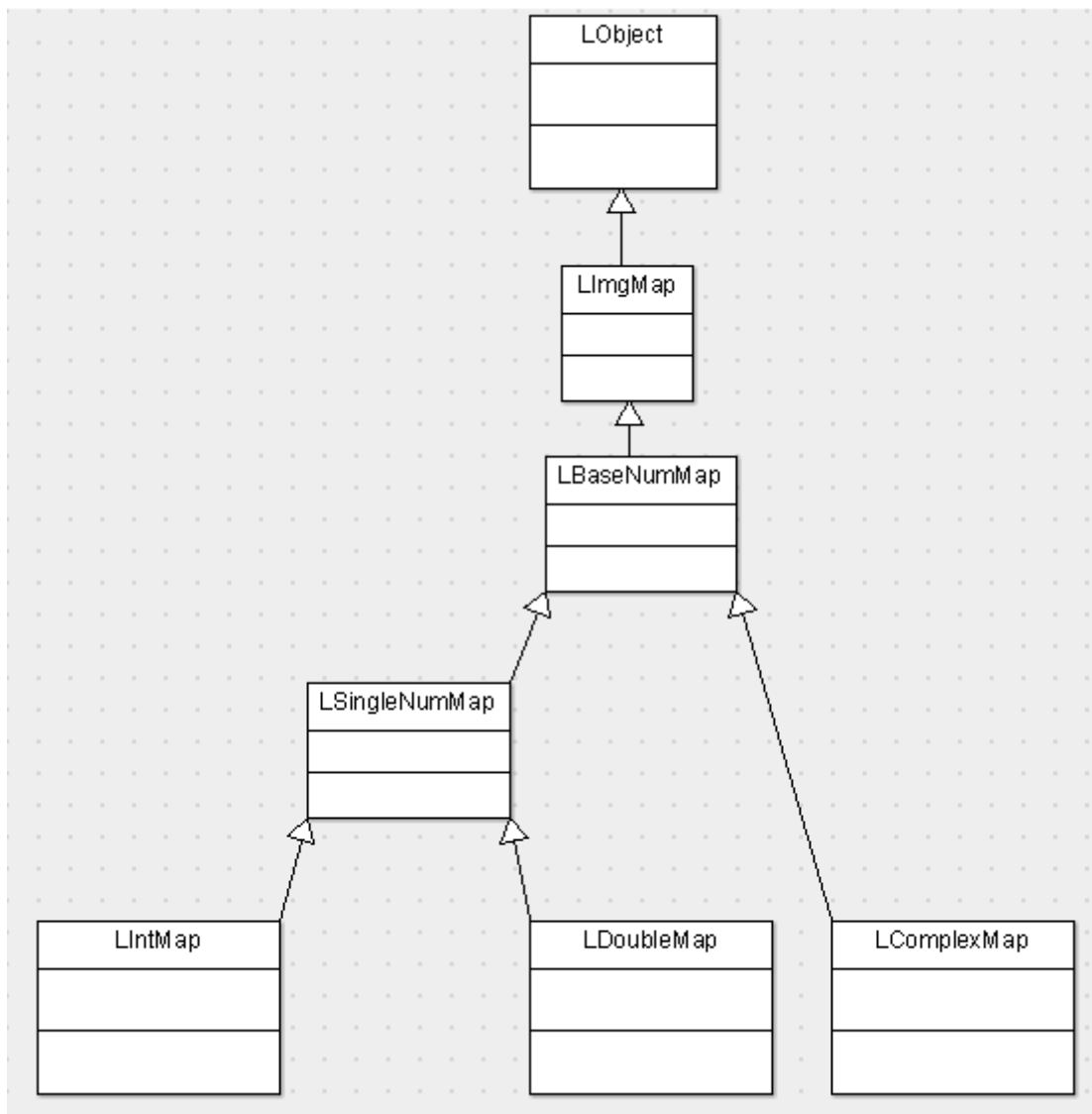


Figura 5: Arquitetura dos Mapas Numéricos

8. **Centro de Massa** - Esta operação também foi adicionada no novo mapa binário.

3 *Ambiente de desenvolvimento*

O primeiro passo para o início do desenvolvimento da aplicação foi a configuração do ambiente de desenvolvimento. Questões como IDE, bibliotecas utilizadas e configurações gerais serão discutidas nesta seção. Como idéia geral, foram utilizadas ferramentas que estiveram sempre presentes no desenvolvimento da *lili2* nestes últimos anos, bem como muito de sua filosofia de desenvolvimento: ser portável, principalmente entre sistemas Windows e baseados em Unix, e totalmente livre.

3.1 IDE

Começaremos pela IDE. Foi utilizado o Codeblocks 10.04, lançado em Maio do ano passado. Ela vem sendo utilizada no desenvolvimento da *lili2* há mais de 2 anos, logo, nada mais natural do que ser utilizada uma ferramenta com a qual já se sabe trabalhar.

Para este projeto, foi utilizado extensivamente uma *feature* proporcionada por esta IDE: as variáveis globais. Como o próprio nome sugere, uma variável global pode ser utilizada em todos os projetos que sejam criados pela IDE. Neste caso, a utilização das variáveis globais teve dois objetivos:

- Especificar as diretivas de compilação, como pastas de inclusão de bibliotecas e headers.
- Proporcionar a parametrização das configurações específicas de cada Sistema Operacional acima citadas.

A janela de configuração de variáveis globais é mostrada na figura 6. Nela, podemos notar algumas das variáveis que a IDE já cria por padrão. O valor-base da variável é descrito

no campo **base** e é obrigatório, todos os demais são opcionais. Logo, se quisermos utilizar a variável global "lili2" dentro do projeto, utilizaremos a sintaxe $\$(\#lili2)$ que será retornado o valor contido no campo **base**. Podemos também definir sub-variáveis dentro da nossa variável global lili2, por exemplo, a sub-variável **lib** é definida como sendo o valor-base mais a extensão **/windows/lib** - a variável lili2 configura as pastas com os caminhos dos headers e da biblioteca. Se quisermos então utilizar esta sub-variável utilizaremos a sintaxe $\$(\#lili2.lib)$.

Uma outra funcionalidade que as variáveis globais desse ambiente tem é a de se especificar um outro conjunto de variáveis de mesmo nome, mas com valores diferentes. Isso permite rapidamente mudar o conjunto de bibliotecas do projeto sem mudar as configurações do projeto em si. Por exemplo, na figura 6 estamos trabalhando com o conjunto *development*, a qual configura as bibliotecas utilizadas no modo *debug*, possibilitando o debugging, muito útil durante o desenvolvimento. Ao final do desenvolvimento, com o software estável, é mudado então para o conjunto *release*, que contém as bibliotecas compiladas utilizando as flags de otimização do compilador.

As variáveis globais podem ser utilizadas de qualquer maneira dentro do projeto, porém, é sempre interessante que tal utilização seja da maneira mais lógica possível, até para evitar confusões com os os nomes das variáveis criadas por padrão. Abaixo, uma breve explanação de como as variáveis expostas na figura 6 foram utilizadas neste projeto:

- **base** - No Windows, o valor-base de uma variável global contém o caminho padrão de instalação de uma biblioteca utilizada no projeto. No Linux, o valor-padrão é nulo. O valor-base das variáveis globais não são utilizados no projeto
- **include** - No Windows, contém o caminho dos headers das bibliotecas utilizadas. No Linux, este valor é nulo.
- **lib** - No Windows, contém o caminho das bibliotecas. No Linux, este valor é nulo.
- **cflags** - Define as flags de compilação passadas ao compilador.
- **lflags** - Define as flags de ligação passadas ao compilador.

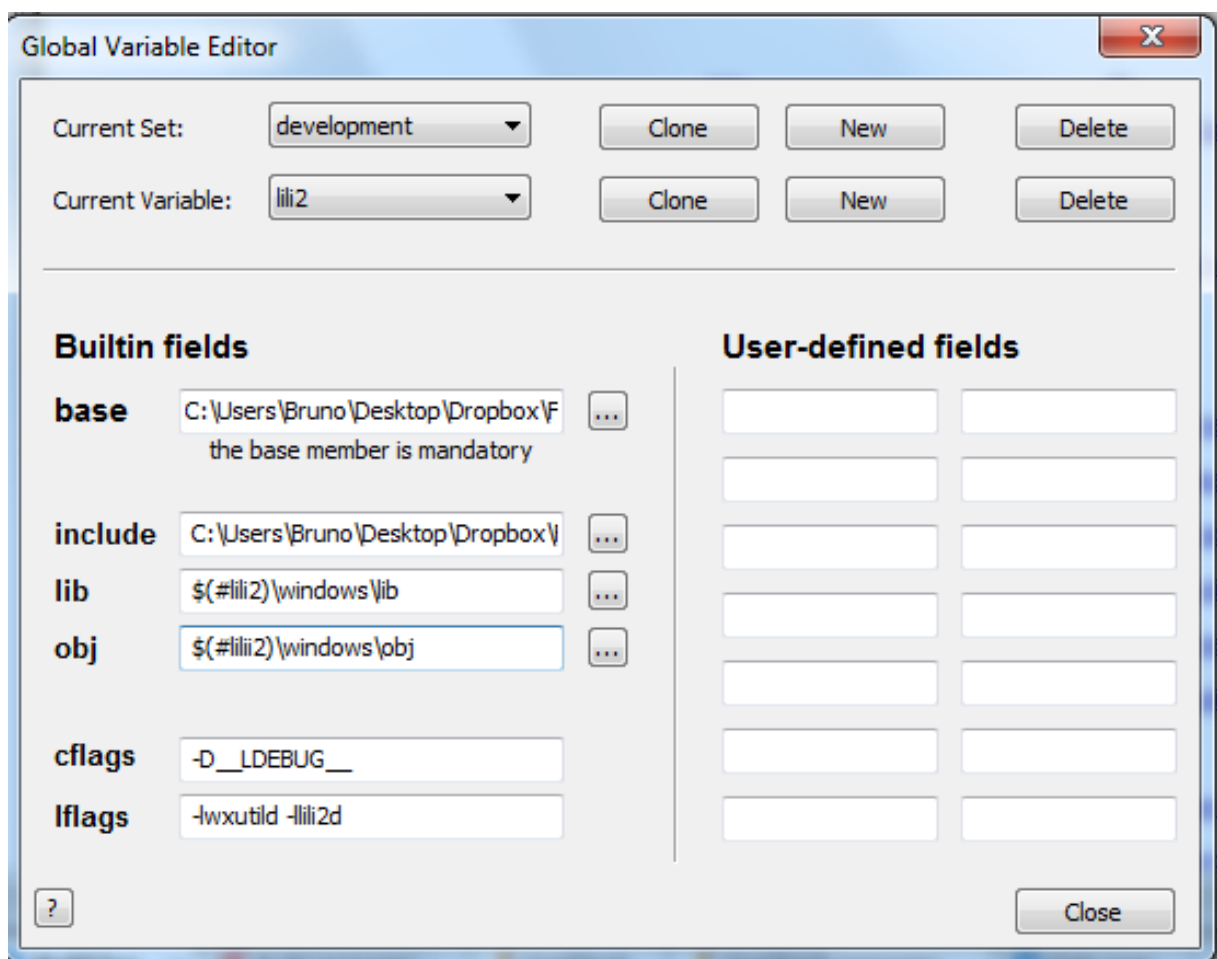


Figura 6: Janela de configuração de variáveis globais no Codeblocks

3.2 Bibliotecas utilizadas

Além da *lili2*, foram utilizadas 2 bibliotecas adicionais neste projeto: a *wxWidgets* (10) e a *OpenCV* (11). Nesta seção, serão discutidas o porquê de terem sido utilizadas e como se deu essa utilização.

wxWidgets - Esta biblioteca é um *toolkit* para facilitar o desenvolvimento cross-plataforma, oferecendo uma API para o desenvolvimento de GUI's e estruturas de dados de forma independente do Sistema Operacional, tais como threads, mutexes e acesso a sistemas de arquivos e diretórios.

Neste projeto, seu principal uso foi para criar a GUI da aplicação, mas seu uso não se limitou apenas a isso. Foram utilizadas também suas abstrações de threads e, consequentemente, seus mecanismos de exclusão mútua, principalmente semáforos.

OpenCV - A *OpenCV* é uma biblioteca para Visão Computacional em Tempo-Real. Segundo o próprio site (11), atualmente ela conta com mais de 2000 algoritmos otimizados e é utilizada desde aplicações de cunho artístico até as utilizadas em robótica.

Como a essência do projeto é utilizar como ferramenta de PDI a biblioteca *lili2*, não foram utilizadas todas essas funções, e sim apenas o módulo responsável por realizar a leitura de vídeos no formato mpeg, formato de vídeo das amostras com as quais se trabalhou. A escolha por utilizar a biblioteca somente por este módulo é devido a sua facilidade em ser utilizado.

4 A aplicação

Definidas as operações básicas necessárias à biblioteca e configurado o ambiente a ser utilizado para o desenvolvimento, finalmente chegamos ao desenvolvimento da aplicação no Desktop. A idéia era utilizar a aplicação como uma ferramenta para o desenvolvimento do software na placa, portanto, os principais objetivos para ela seriam:

- Proporcionar uma maneira de, a partir das amostras de teste, debuggar a aplicação de PDI passo-a-passo, observando os resultados intermediários do algoritmo utilizado.
- Verificar o desempenho do algoritmo quando rodado em uma máquina com maiores recursos computacionais.

Porém, desenvolver a aplicação acabou tornando-se um desafio maior do que o esperado, culminando em tornar-se o objetivo final deste trabalho.

4.1 Interface

Começaremos com a interface da aplicação.

Como pode ser visto na figura 4.1, ao iniciar a aplicação conta apenas com os botões básicos para controle do fluxo de vídeo, todos eles devidamente desabilitados.

Clicando no menu *Arquivo -> Abrir*, é aberta uma janela de diálogo solicitando a seleção do vídeo a ser processado. Selecionado o vídeo, a aplicação automaticamente expande a janela de forma a acomodar o tamanho dos frames do vídeo, duas vezes, além de ser adicionada uma opção no menu para seleção do passo do processamento a ser exibido na tela. Esse estado é mostrado na figura 4.1.

Agora, é só uma questão de iniciar o processamento. Isso, naturalmente, ocorre quando é pressionado o botão "Play". Os demais botões controlam as devidas opções pertinentes: "Pause" permite travar a execução do vídeo, "Stop" pára totalmente o processamento,

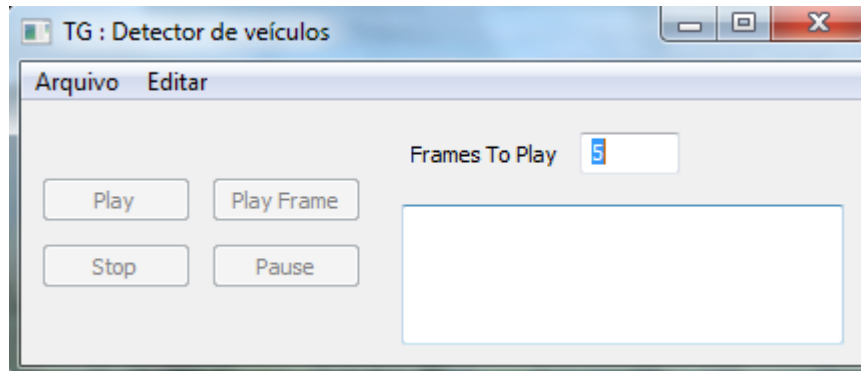


Figura 7: Estado inicial da aplicação

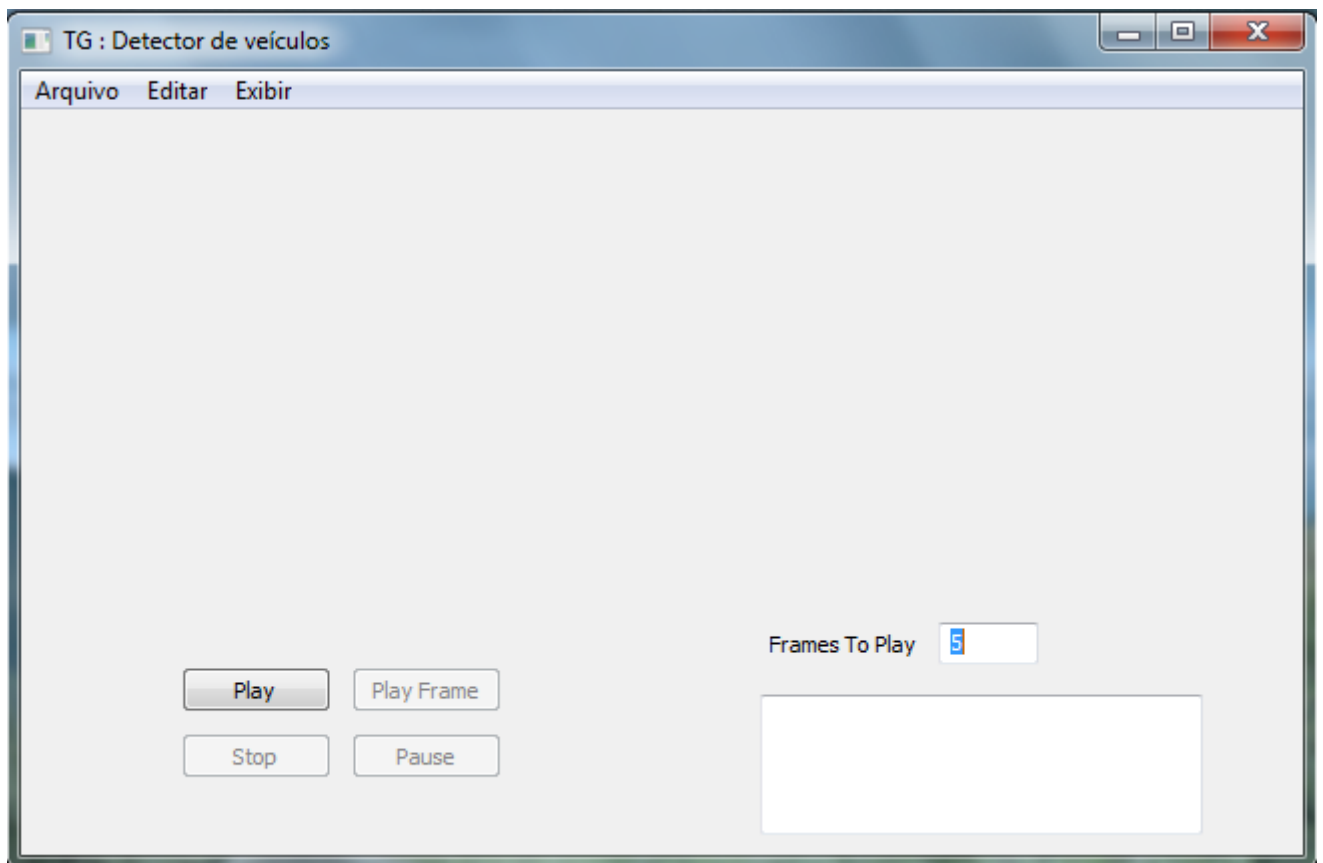


Figura 8: Estado da aplicação após vídeo ser selecionado. A janela principal é redimensionada de forma a acomodar 2x as dimensões do frame do vídeo escolhido, a fim de ser exibido tanto o veículo detectado pelo algoritmo quanto passos intermediários do processamento

e "Play Frames" executa uma certa quantidade de frames da aplicação.

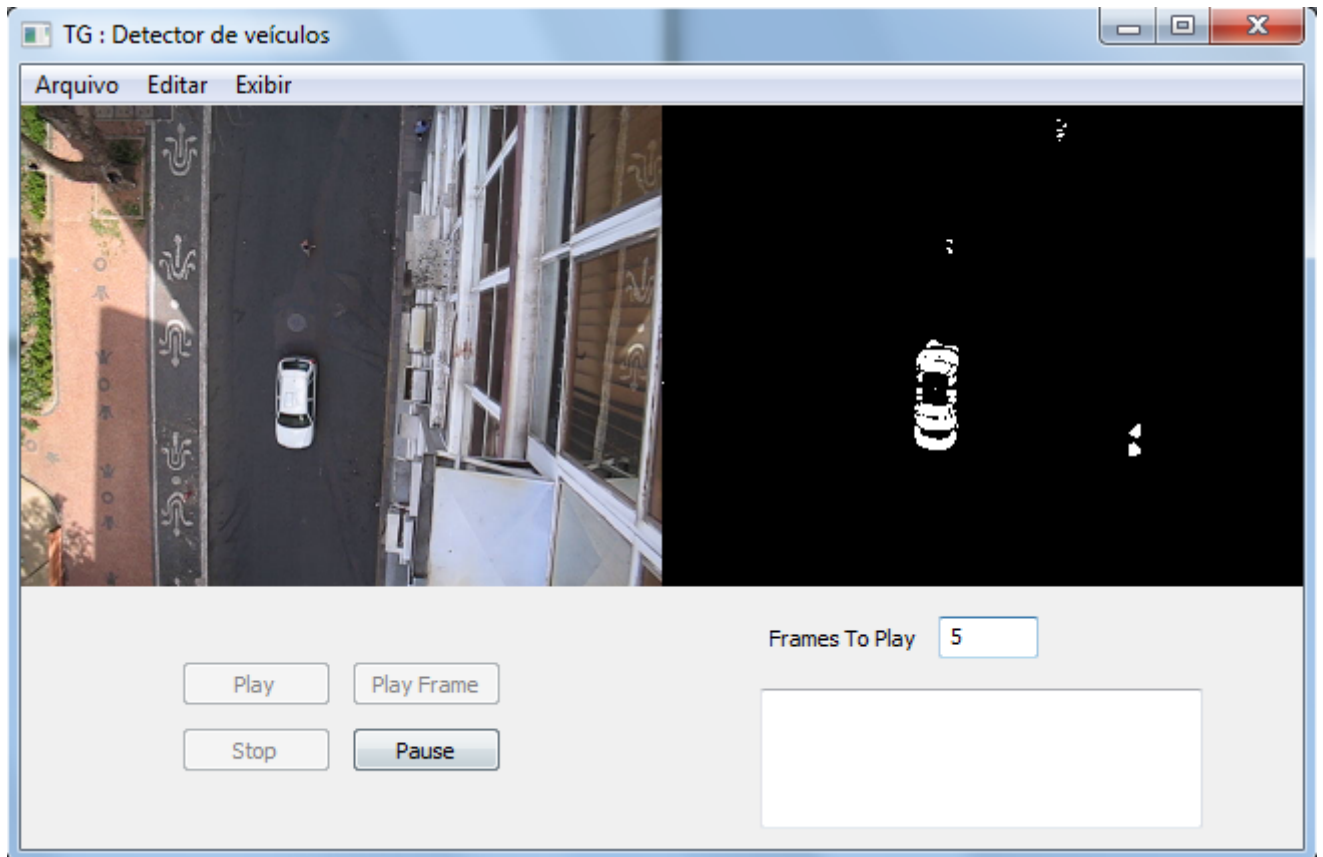


Figura 9: Aplicação rodando o vídeo e processando-o, exibindo o passo da binarização

Adicionalmente, em "Exibir" é possível selecionar o passo do processamento a ser exibido na janela da direita - no caso da figura 4.1, é exibido o primeiro passo do algoritmo, a transformação do frame para Grayscale. Clicando em *Edit -> Preferences*, é possível alterar alguns dos comportamentos do algoritmo utilizado, bem como outras diversas opções.

4.2 A Lógica por trás da interface

Agora que a interface foi explicada, será desenvolvido o que realmente ocorre por trás dos panos com a aplicação.

Em primeiro lugar, as principais funções de controle do que está ocorrendo na aplicação são implementadas em função dos botões de fluxo de exibição da sequência de vídeo -

"Play", "Pause", "Play Frames", e "Stop". Decidiu-se implementar uma máquina de estados, associando a cada botão uma sequência de operações a serem executadas.

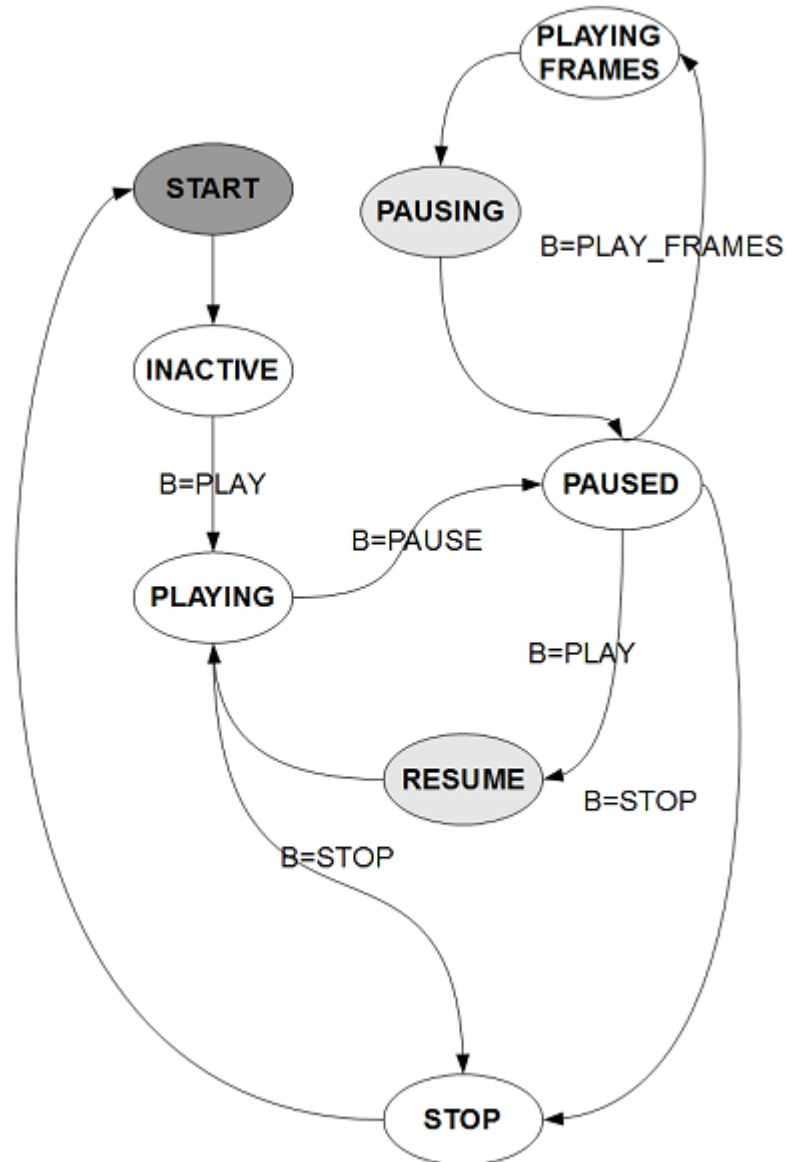


Figura 10: Máquina de Estados da aplicação

Na figura 4.2, podemos ver os estados que compõem a máquina. As transições são ativadas pelo clique em cada botão, representados pela letra "B" em cada transição. Para iniciar a máquina, é necessário selecionar antes o arquivo de vídeo a ser processado, indo em *File -> Open*. Com exceção dos estados RESUME e PAUSING, para cada um dos

estados é associado uma sequência de operações, a saber:

- **INITIAL** - Neste estado, apenas é desabilitado todos os botões que controlam o fluxo da aplicação.
- **INACTIVE** - Este estado é disparado quando se seleciona o vídeo a ser processado, indo em *File -> Open*. Ele cria uma thread que será responsável por fazer a leitura e redimensiona as áreas responsáveis por exibir os frames do vídeo original e processado, de acordo com o tamanho dos frames do vídeo ou do tamanho desejado pelo usuário.

Além disso, ele cria mais uma barra nas tarefas para permitir a "troca" entre as visualizações dos passos do algoritmo utilizado, permitindo que possamos ver os resultados intermediários do algoritmo, possibilitando um "debug pelo olho" da aplicação.

- **PLAYING** - Quando o botão *Play* é pressionado, a aplicação muda para este estado e efetivamente inicia o processo de leitura e processamento dos frames. Ao ser iniciado tal processo, a thread principal da aplicação cria mais uma thread, a qual agora será responsável por realizar a leitura do vídeo. Tal processo consiste nos seguintes passos:

1. Captura um frame do vídeo, em formato de frame OpenCV.
2. Converte-o para o formato de imagem wxWidgets.
3. Escalona o frame, caso seja necessário/desejado.
4. Converte o frame wxWidgets para um mapa LRGBMap, da lili2.
5. Coloca o frame LRGBMap na fila da thread de processamento.
6. Realiza um "Yield", esperando um tempo para voltar à leitura de frames do vídeo proporcional ao seu FPS.
7. Sinaliza um evento de desenho para a thread principal.

Também foi implementado um Controle de Congestionamento a fim de evitar que o buffer com frames prontos a serem processados estoure. Para isso, foi criada uma Barreira inicializada com o valor "x", onde "x" é número máximo de frames na fila de aptos para processamento. Esta Barreira é compartilhada entre a thread de leitura

e a thread principal.

Antes de se iniciar o processo de leitura, é criada mais uma thread, a qual será responsável por fazer o processamento dos frames já lidos. Esta thread funciona da seguinte maneira:

1. Ao receber um novo frame para ser processado, retira esse frame da fila. A thread de processamento sabe que há um novo frame através de uma barreira compartilhada entre a thread de leitura e de processamento.
 2. Aplica o algoritmo de detecção. Este algoritmo consiste nos seguintes passos:
 - (a) Converter o frame para tons de cinza.
 - (b) Aplicar a normalização.
 - (c) Ver se este é o primeiro frame
 - Se sim:
 - i. Converte o frame para um Mapa em Ponto Flutuante.
 - ii. Aplica o filtro da média.
 - iii. Guarda o frame já que ele é o primeiro frame do vídeo processado.
 - Se não:
 - i. Converte o frame para um Mapa em Ponto Flutuante.
 - ii. Aplica o filtro da média.
 - iii. Detecta movimento realizando a subtração entre o frame atual e o anterior.
 - iv. Binariza a imagem utilizando o método estatístico.
 - v. Aplica a operação de fechamento.
 - vi. Calcula o centro de massa da imagem binária resultante.
- **PAUSED** - Este estado é disparado quando se clica no botão "PAUSE", e só pode ser atingido se houver um vídeo rodando.

O efeito de vídeo pausado foi implementado utilizando uma estrutura composta por um Mutex, um Semáforo e uma variável booleana. O semáforo atua com uma barreira, e o Mutex protege modificações na variável booleana. A barreira é inicializada com o valor "0" e o Mutex com "1".

Quando o botão "Pause" é clicado, primeiro é setado o valor da variável booleana para **true**. Na próxima iteração do laço de leitura do vídeo, entra-se no Mutex e é feito o teste sob a variável, que será portanto verdadeira. Isso faz com que se entre em um fluxo de execução onde se sai dessa área protegida pelo Mutex e entre na barreira. Quando então for clicado em um dos botões "Play" ou "Play Frames", a barreira é liberada e o processamento continua.

Em outras palavras, o algoritmo de pausa é o seguinte:

1. Entra na área do Mutex de Pausa
 2. Testa o Valor da variável indicando se deve pausar
 3. Se sim, deve pausar:
 - (a) Sai do Mutex
 - (b) Entra na Barreira. Quando for clicado "Play" ou "Play Frames", a barreira é liberada e o fluxo continua
 4. Se não, não deve pausar:
 - (a) Sai do Mutex
- **PLAY_FRAMES** Este estado só pode ser atingido caso o vídeo tenha sido pausado - ele não pode ser atingido em um vídeo que ainda não tenha começado a ser processado. A função deste estado é fazer o processamento frame-a-frame do vídeo, possibilitando ao desenvolvedor ver o que está acontecendo de forma mais clara.

Para fazer esse efeito de frame-a-frame, foi utilizada a mesma estrutura do "Pause". Porém, ao invés de esperar uma operação de "Post" oriundo de um clique no botão "Play", são feitos "n" "Posts", onde "n" é quantidade de frames que se deseja processar. Quando esses "n" frames terminarem de ser processados, a máquina volta ao estado "PAUSED". Tal quantidade pode ser alterada pela interface do programa.

Expressando esse algoritmo passo-a-passo:

1. Entra na área do Mutex de Pausa.
2. Modifica a variável que indica se se deve pausar para "true".
3. Enquanto $i < n$
 - (a) Executa um "Post" no Semáforo de Pausa

4. Sai do Mutex de Pausa

- **RESUME** Este estado aplica um "Post" no semáforo responsável por pausar o processamento, continuando com o fluxo normal de execução. Ele só é atingido se a máquina estiver no estado PAUSED.
- **STOPPED** Este estado pára o processamento do vídeo. Primeiro, ele sinaliza à thread de leitura do vídeo para sair do laço de leitura, e esta sinaliza à thread de processamento o mesmo evento. Depois, através de uma chamada "Wait" na thread principal, se espera pelo término e destruição de tudo. Ao final do processo, a máquina de estados volta ao estado INACTIVE.
- **PAUSING** Este estado não chegou a ser utilizado no final, porém, foi colocado porque se previu que talvez fosse ser necessário tê-lo. Felizmente, esse não foi o caso.

4.3 Resultado das amostras

4.3.1 Gol branco se aproximando lentamente

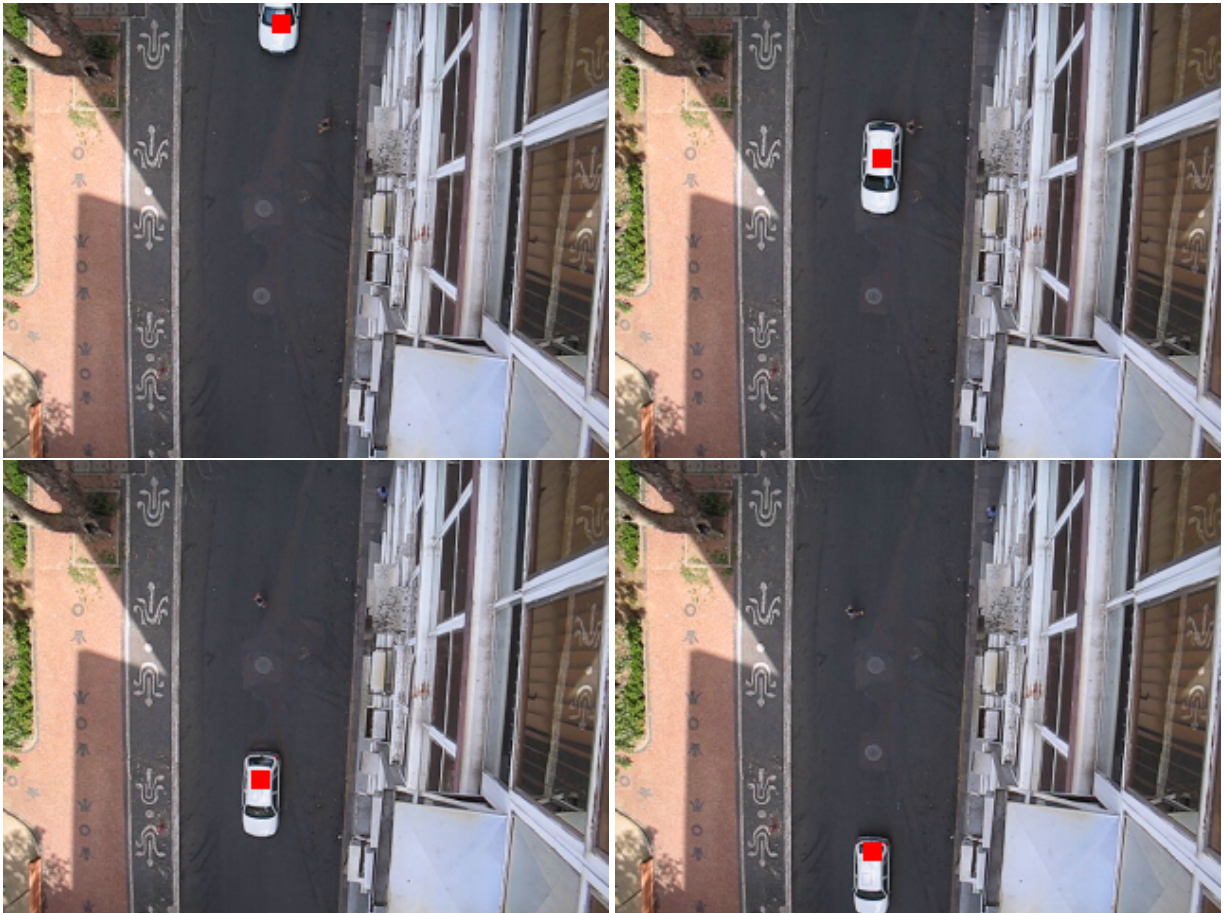


Figura 11: Esta amostra não possuía nenhuma interferência significativa, portanto, o algoritmo se comportou muito bem, marcando com precisão o veículo desejado

4.3.2 Gol branco com reflexo

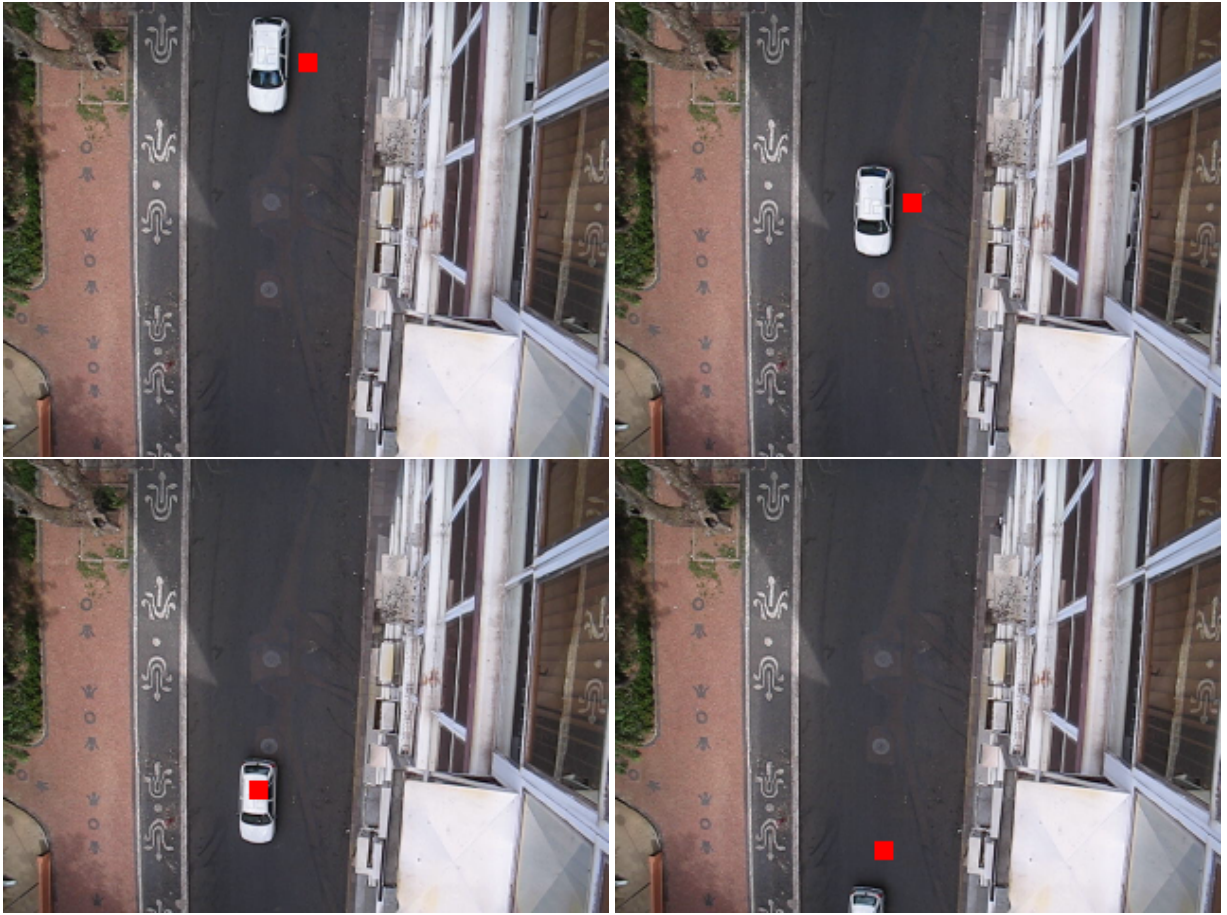


Figura 12: Aqui ocorreu pela primeira vez o problema do reflexo, que é aparecer no mesmo frame capturado o veículo e seu reflexo de alguma forma. Aparece aqui um problema da escolha de implementação do algoritmo: ele é incapaz de decidir aquilo que é um veículo daquilo que não é.

4.3.3 Chevete Azul com outro carro passando

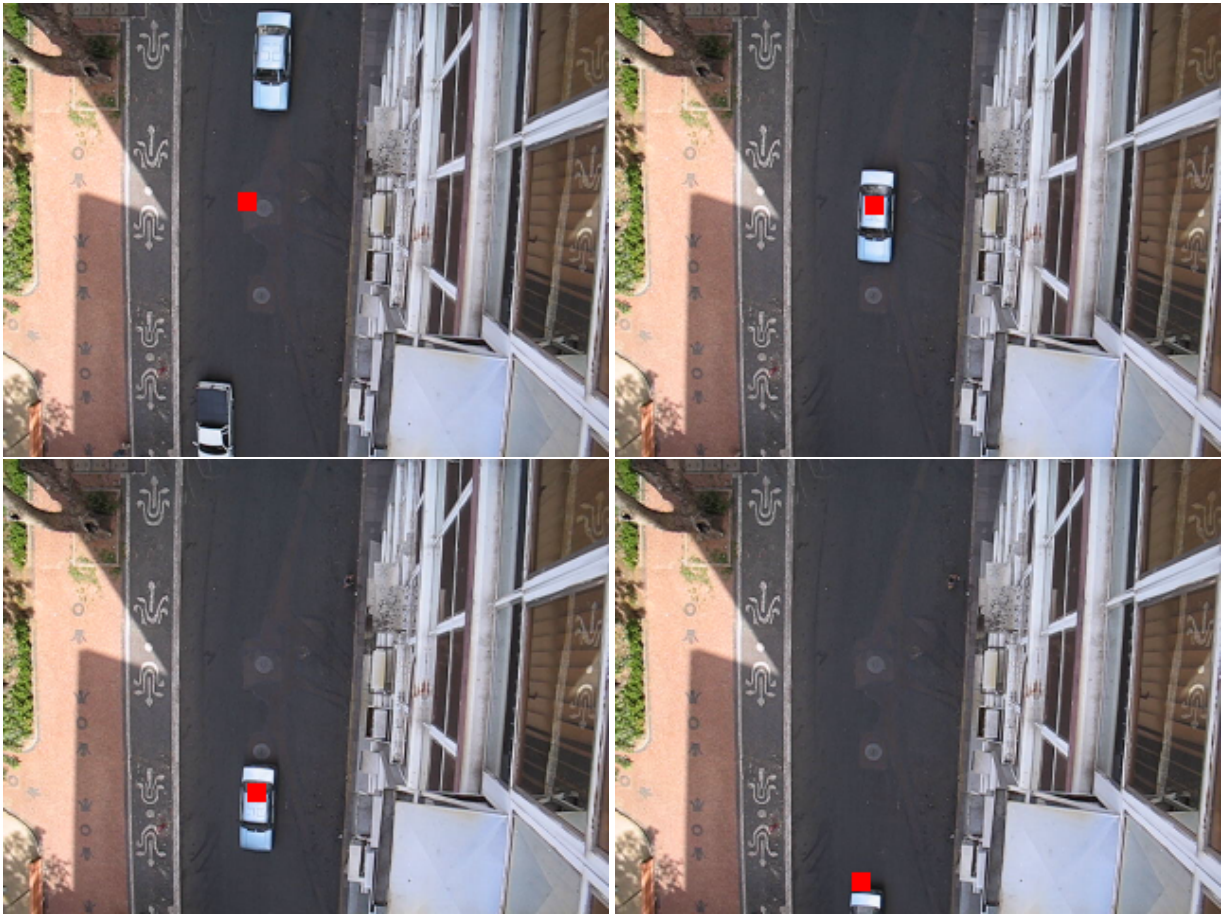


Figura 13: Esta amostra começa com o final de um veículo saindo, depois entrando o Chevete azul. Aqui, o algoritmo demonstrou demorar um pouco para voltar a captar o Chevete, provavelmente devido ao fato de que havia o final de um carro anteriormente sendo seguido, por isso, atrapalhando a fase de correção da trajetória.

4.3.4 Chevette Azul com reflexo

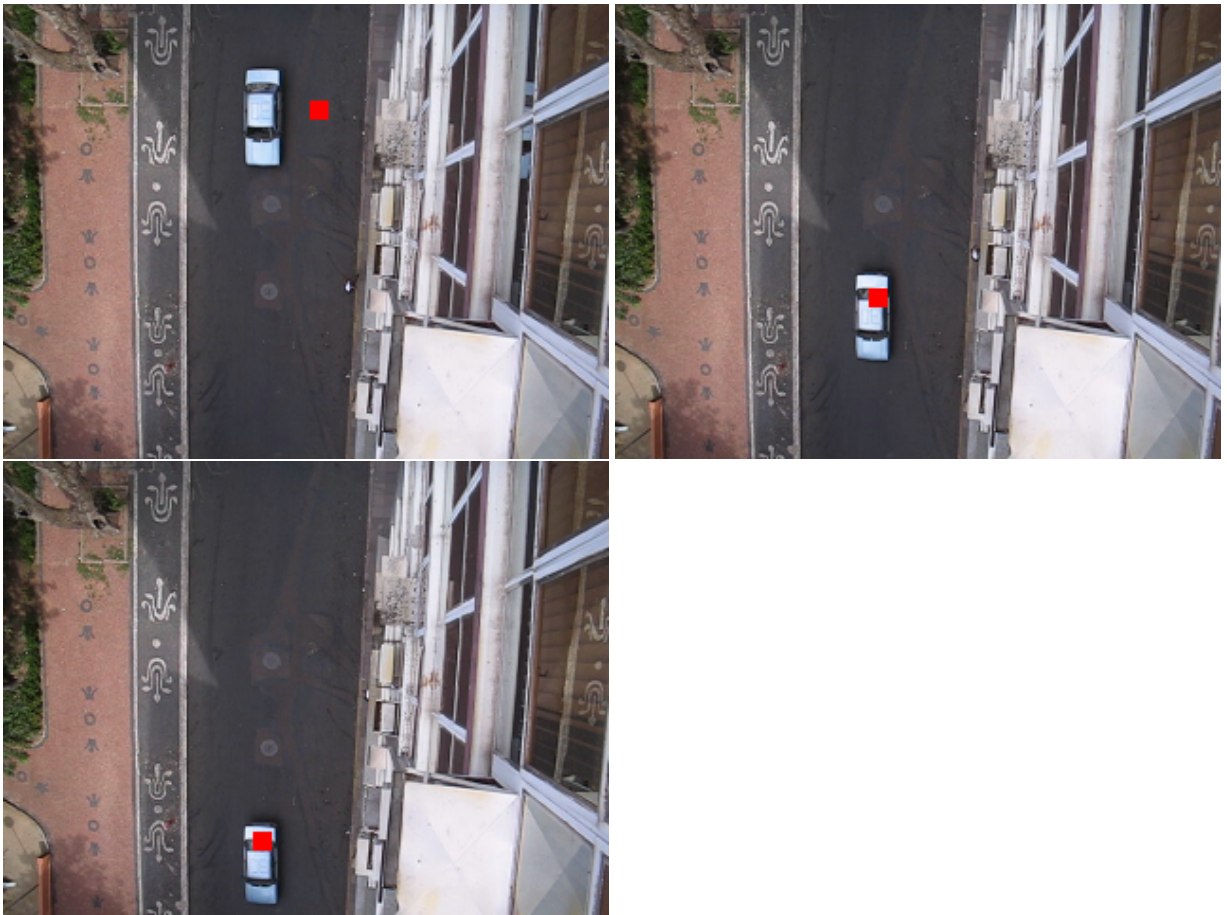


Figura 14: Esta amostra teve como interferência a presença de um reflexo do veículo. Aqui, sem mecanismos de detecção para decidir o que pode ser um veículo e o que não pode, a aplicação determinava que o Centro de Massa ficava sempre entre os 2 objetos, o carro e o refletido.

4.3.5 Chevette Azul rápida, com reflexo

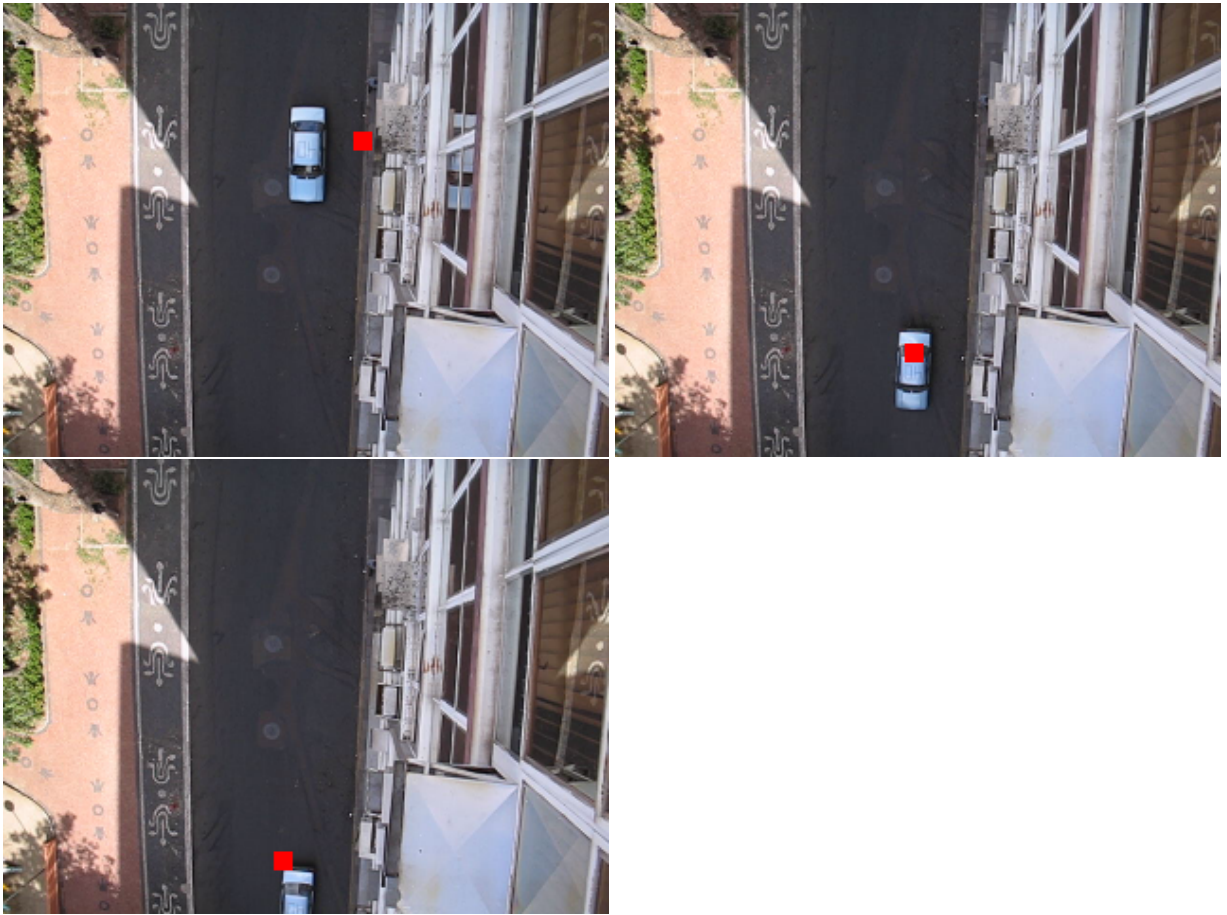


Figura 15: Esta amostra demonstrou que o algoritmo captura o veículo rapidamente, porém, de novo aparece o problema do reflexo, marcando o centro do veículo como entre ele e o seu reflexo.

5 Conclusão

Ao final do desenvolvimento da aplicação, a biblioteca mostrou que pode sim ser utilizada em aplicações de PDI mais complexas, porém, tal uso requer alguns cuidados devido ao modelo de desenvolvimento adotado pela biblioteca *lili2*.

Um desses cuidados é o de evitar o uso de objetos que são constantemente construídos e destruídos, de forma a evitar um uso exagerado e desnecessário da memória. Também é importante cuidar de como se utilizam os operadores e construtores de cópia já que, dentro do modelo, estas duas operações alocam uma área na memória proporcional ao tamanho da imagem a ser copiada e copiam o seu conteúdo, dobrando efetivamente a quantidade de memória consumida. Portanto, o desenvolvedor tem que avaliar se seu código pode se tornar menos oneroso à memória analisando esses fatores.

A biblioteca, por necessitar apenas da biblioteca padrão do C++ e requerer, por causa disso, apenas um compilador C++, acaba se tornando bastante fácil de ser instalada. Porém, isso reduz a quantidade de estruturas que podem ser utilizadas visando otimização, logo, os algoritmos necessitam que sejam implementados de forma ótima, utilizando para isso apenas aquilo que a linguagem e as biblioteca padrão do C++ oferece. Isso acaba se tornando um ponto negativo para a biblioteca quando deseja-se utiliza-la em uma plataformas mais limitadas, comumente encontrado nas plataformas embarcadas e constantemente sendo uma limitação nos projetos de PDI embarcados encontrados mundo afora.

Uma outra característica importante que foi notada é o modelo de desenvolvimento da *lili2*, baseado em templates. Este modelo provou ser interessante porque, para as funções implementadas nos *templates*, não é a biblioteca toda que é compilada ao se ligar a biblioteca, apenas as funções utilizadas, consequentemente reduzindo o tamanho do código

gerado, além de permitir generalizar operações de forma que elas funcionem para mais de um mapa de imagens. O único problema dessa abordagem é que ela requer um esforço de programação extra para contornar algumas limitações da utilização de templates em C++ quando é utilizada derivação, outra *feature* do C++ utilizada extensivamente na lili2.

Uma outra força do modelo lili2 de desenvolvimento é sua facilidade de uso. Embora, indiscutivelmente, ela perca em termos de desempenho e em quantidade de opções oferecidas ao desenvolvedor, ela é farta de exemplos simples, possibilitando que o desenvolvedor aprenda a lógica da biblioteca rapidamente, e sempre procura-se adicionar na biblioteca as funções de forma mais genérica possível, visando a utilização futura destas funções em outros tipos de mapas, pensando sempre em reduzir a quantidade de parâmetros que essas funções tomam para reduzir o *overhead* de aprendizado.

Quanto à biblioteca auxiliar wxWidgets, ela mostrou possuir uma boa arquitetura, resolvendo muito bem o problema da generalização de criação de interfaces gráficas. Contudo, ela possui um defeito que é não possuir uma ferramenta que abstraia totalmente a criação de tais interfaces - o desenvolvedor pode necessitar de mais flexibilidade além daquela que é oferecida pelas ferramentas atuais, conseqüentemente, requerendo que ele entenda mais a fundo o funcionamento da biblioteca.

6 *O Futuro*

A biblioteca lili2 nasceu a partir da idéia de ser utilizada para ensinar Processamento de Imagens ao mesmo tempo em que aproxima os alunos do ambiente no qual o desenvolvimento de aplicações desse tipo ocorrem.

Contudo, na tentativa de se resumir todo um processo de desenvolvimento, acabou-se complicando todo ele, visto que, para facilitar o desenvolvimento de aplicações que utilizem a biblioteca, é necessário utilizar algum tipo de interface, a qual também deve, na medida do possível, oferecer todo um ferramental que facilite essa tarefa.

Atualmente, é necessário utilizar a biblioteca wxWidgets para exibir imagens e, embora esta biblioteca seja muito boa e generalize muito bem o problema de criar interfaces gráficas, ela ainda não possui uma ferramenta que permita facilitar o desenvolvimento de tais interfaces de forma rápida. Os alunos da cadeira de Processamento de Imagens sentiam bastante dificuldade com a biblioteca por causa disso: eles precisavam criar uma interface para os diversos trabalhos que queriam fazer.

Para manter a Filosofia original do projeto, pensa-se que é melhor criar uma interface gráfica que permita ao aluno criar um algoritmo de PDI usando o famoso modelo arrasta-e-solta do Windows. A aplicação que este TCC se propôs a fazer já é uma idéia inicial do que esta interface poderia ser.

Tal interface seria então construída utilizando uma linguagem interpretada como Java ou Python, já que uma das vantagens que estas linguagens oferecem é justamente abstrair inúmeros problemas oriundos das diferenças existentes entre os Sistemas Operacionais - incluindo nesse escopo interfaces gráficas - além de serem linguagens fáceis e intuitivas.

Referências

- 1 Gonzales R., Woods E.: *Digital Image Processing*, Prentice Hall/New Jersey, 2a. edição, 1997
- 2 Wolf W. H.: *Computer as Components: Principles of Embedded Computing Systems Design*, Morgan Kaufmann, 2a. edição, 2008
- 3 Stroustrup B.: *The C++ Programming Language*, Addison-Wesley, 3a. edição, 1997
- 4 Haupt A.; *Detecção de movimento, acompanhamento e extração de informações de objetos móveis*, Porto Alegre, 2004
- 5 Chu S., Yehi M., Cheng K.: *A Real-time, Embedded Face-Annotation System*
- 6 Yamaguchi K., Watanabe Y., Komuro T., Ishikawa M.: *Interleaved Pixel Lookup for Embedded Computer Vision*
- 7 Taglewski R., Wójcikowski M.: *Multi-core processor system for real-time image processing in embedded computer vision applications*, Proceedings of the 2008 1st International Conference on Information Technology, IT 2008 19-21 May 2008, Gdansk, Poland
- 8 Zongtao D., Yanni Z., Zongyuan D.: *A Parallel Language for Embedded Real Time Image Processing*, 2009 International Forum on Information Technology and Applications
- 9 Kang G. S.: *Real-Time Computing: A New Discipline of Computer Science and Engineering*, Proceedings of the IEEE, Vol. 82, No. 1. Janeiro 1994
- 10 *wxWidgets: A portable C++ and Python GUI toolkit*, <http://docs.wxwidgets.org/stable/>
- 11 *OpenCV: Open Source Computer Vision*, <http://opencv.willowgarage.com/wiki/>

Glossário

Processo Em poucas palavras, um programa em execução competindo pelo tempo na CPU.

Thread Fluxo de execução de um processo. Um processo pode ter mais de um fluxo, ou thread, e também compartilhar dados entre esses fluxos, o que pode acarretar em Condição de Corrida e inconsistências nos dados compartilhados entre as threads caso não seja utilizados mecanismos de exclusão mútua.

Semáforo Estrutura de dado proposta por Djikistra, composta por uma fila de processos e um contador. Sobre essa estrutura, podem ser feitas duas operações, todas elas atômicas:

- **Wait** O contador é decrementado. Se o valor do contador ainda for positivo, ainda é possível que mais fluxos "passem" pelo Semáforo. Se não, o processo é posto na fila e o fluxo pára naquele ponto.
- **Post** O contador é incrementado. Se o valor do contador for negativo, o processo no topo da fila é retirado e é lhe dado o direito de continuar onde parou. Se o contador for maior ou igual a 0, ele continua sendo incrementado indefinidamente, possibilitando que mais fluxos passem por aquele ponto.

Mutex Estrutura de dados que visa tornar exclusivo o acesso a dados compartilhados entre múltiplas threads, com o objetivo de evitar que os dados se tornem inconsistentes ao ocorrer acessos múltiplos.