UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

ESCOLA POLITÉCNICA

DEPARTAMENTO DE ELETRÔNICA E DE COMPUTAÇÃO

INTERFACE PARA CONTROLE DE UM PAR DE CÂMERAS FOTOGRÁFICAS DIGITAIS PARA UM SISTEMA DE VISÃO ESTÉREO COMPUTACIONAL

Autor:

Gustavo Guerreiro Basilio Costa

Orientador:

Prof. José Gabriel Rodriguez Carneiro Gomes, Ph.D.

Examinadora:

Prof[a]. Mariane Rembold Petraglia, Ph.D.

Examinador:

Prof. Júlio César Boscher Torres, D.Sc.

DEL

Março de 2008

Dedicatória

À minha família, legítimos batalhadores que me ajudaram a chegar até aqui.

Agradecimentos

A diversos amigos e professores que contribuíram de alguma forma para a conclusão deste projeto. Em especial a Felipe Gomes Dias (M.Sc. Sistemas de Informação, UFRJ) e Thiago Siguenobu Vargas Arakaki (Eng. Eletrônica e de Computação, UFRJ) que me ajudaram com as especificações e detalhamentos técnicos em engenharia de *software*.

Aos professores Osvaldo Pereira, José Paulo Braffman e José Gabriel R. C. Gomes, meus professores durante a graduação, autênticos educadores que valorizam o aluno, mas acima de tudo estimam a pessoa.

À Universidade Federal do Rio de Janeiro, por ter me dado o orgulho e o privilégio de fazer parte de uma instituição de reconhecimento mundial.

À Profª. Priscila Machado Vieira Lima, por ter me dado atenção e ajudado num momento decisivo do projeto.

Ao Prof. Eduardo A. B. da Silva, por me instruir com o conhecimento específico de processamento de imagens.

Ao Prof. Sérgio Palma, por ter me ensinado a pensar de outra forma sobre minha vida profissional.

Resumo

O tema deste projeto é o gerenciamento das funcionalidades (captura de imagens estáticas e vídeo, e configuração) de duas câmeras digitais idênticas, remota e simultaneamente, via computador. Pretende-se com isso integrar um sistema de visão computacional que simula a visão humana em três dimensões.

O objeto de estudo é o par de câmeras digitais Canon PowerShot G7 associado a um computador. A comunicação entre o PC (*Personal Computer*) e as câmeras deve ser feita via portas USB (*Universal Serial Bus*), utilizando as regras estabelecidas pelo fabricante em seu SDK *(Software Development Kit*).

O SDK do fabricante é uma biblioteca de ferramentas de programação escritas em C, cujo ambiente de desenvolvimento é a plataforma Windows. Já o ambiente alvo de um aplicativo feito com o SDK é a plataforma Windows, das versões 95 até a XP.

Algumas técnicas de modelagem de dados e de paralelismo computacional, assim como metodologias específicas para este projeto (por exemplo, sincronizador de disparo de fotos), são aplicadas para se atingir os objetivos do projeto.

O sistema de visão (estéreo) computacional supracitado foi desenvolvido em C++ e, portanto, apresenta compatibilidade com o SDK. Atualmente há uma nova versão deste sistema, que por sua vez, também é compatível com o SDK.

Ao fim do projeto obteve-se um programa com interface amigável onde foram disponibilizadas as funcionalidades de: captura de vídeos; auto-ajuste de foco; visão estereoscópica (parcial), controle de qualidade e dimensões das fotos.

Palavras-chave

Câmera Digital, Canon PowerShot G7, SDK, *Viewfinder*, Fotografia de Alta Resolução, Controle Remoto, *Thread*, Processo, Processador, USB, Visão Computacional, Estereoscopia, Visão Estéreo Binocular, 3D

Índice

Índice de Figuras

Índice de Tabelas

Glossário, Abreviaturas, Siglas, Símbolos e Sinais

Bitmap – Mapa de *bits*. Contém a descrição de cada pixel de uma imagem.

Charset – Códigos que correspondem a um conjunto de caracteres usados para padronização.

CPU – *Central Processing Unit* (ou Unidade Central de Processamento) de um processador. É a parte do computador que interpreta e executa as instruções de um programa.

DCF – *Design rule for* Camera *File system*. Uma especificação da JEITA que define um formato e um sistema de arquivos para câmeras digitais, incluindo a estrutura de diretórios, o método de nomeação de arquivos, o *charset*, o formato do arquivo, e formato dos meta-dados. O DCF é baseado no Exif 2.2.

Device Driver – Programa de controle de um dispositivo que faz a interface entre o sistema operacional e um programa.

Exif – *Exchangeable image file format.* Uma especificação do formato de arquivos de imagem usado em câmeras digitais. O Exif usa os formatos JPEG, TIFF (revisão 6.0) e RIFF WAV, assim como *tags* para meta-dados.

Fps – *Frames Per Second* (ou Quadros Por Segundo). Unidade de medida da taxa com que um dispositivo produz imagens únicas, consecutivas chamadas *frames* (quadros).

Framework – Em computação, um *framework* é uma estrutura de suporte definida em que outro projeto de *software* pode ser organizado e desenvolvido.

Hub USB – Dispositivo que permite que diversos dispositivos USB sejam conectados a mesma porta. Analogamente pode ser descrito como um controlador de dispositivos USB.

GDI – *Graphics Device Interface*. Subsistema do Windows para representar imagens e renderizá-las em dispositivos gráficos como monitores. Usado para desenhar linhas, curvas e fontes, por exemplo.

IBM – Abreviação de *International Business Machines Corporation*. Empresa multinacional de tecnologia de computadores.

IJG – *Independent JPEG Group*.

JEITA – *Japan Electronics and Information Technology Industries Association*

JPEG – *Joint Photographic Experts Group*. Em computação é um método comumente usado para compressão de imagens fotográficas.

JPEGLIB – Biblioteca livre do IJG que contém funções de compressão e descompressão de imagens no formato JPEG.

Kernel – Componente central de sistemas operacionais. É ele quem controla os recursos do sistema.

LIFO – *Last In First Out*. Em computação, refere-se ao modo como dados armazenados em estruturas típicas são processados.

Meta-dado – Dados que identificam ou classificam outros dados.

MFC – *Microsoft Foundation Class*. É uma biblioteca criada pela Microsoft que contém partes da API do Windows.

Mutex – Mutual Exclusion. É o nome dado a um método de sincronização de processos.

PID – *Product ID*. Sigla usada pelo Windows para identificar um dispositivo em uma chave do seu registro.

Pilha – Estrutura de dados no estilo LIFO.

PowerShot G7 – Modelo de câmera usado neste projeto.

Procedures – Procedimento ou rotina a ser executada por um processo/*thread*.

Processo – Em computação é o mesmo que um programa sendo executado.

PsRecSDK – *PowerShot RemoteCapture Software Development Kit*.

RAM – *Random Access Memory* (ou Memória de Acesso Aleatório). É um tipo de armazenamento de dados. Corresponde a circuitos integrados que permitem acesso aos dados armazenados de forma aleatória.

RelCtrl – Nome do programa de exemplo que acompanha o PsRecSDK.

RIFF – *Resource Interchange File Format*. É um formato genérico para armazenamento de dados em pedaços nomeados.

ROV – *Remotely Operated underwater Vehicles*. Nome normalmente dado para um robô submersível que é operado remotamente por uma pessoa a bordo de uma embarcação.

SVE – Sistema de Visão Estéreo. Nome dado ao programa que corresponde ao sistema de visão estéreo existente desde o início deste projeto.

Tag – Palavra-chave ou termo associado a uma informação. Ver meta-dado.

Thread – Em computação uma *thread* é uma seqüência de instruções mais "leve" que um processo que pode ser executada em paralelo com outras.

TIFF – *Tagged Image File Format*. Formato de arquivo de imagens.

UML – *Unified Modeling Language*. Linguagem padrão de modelagem de objetos em engenharia de *software*.

USB – *Univesal Serial Bus*. Barramento serial de dados que se tornou padrão mundial.

VEM – Visão Estéreo Multiplataforma. Nome dado ao novo sistema de visão estéreo, citado neste projeto.

VID – *Vendor ID*. Sigla usada pelo Windows para identificar o fabricante de um dispositivo em uma chave do seu registro.

videoInput – Nome de uma biblioteca livre, que faz a captura de imagens de *webcams*.

WAV – Abreviação de *Waveform*. Padrão de arquivo de áudio criado por Microsoft e IBM.

Webcam – Câmera de vídeo cujo uso principal é em aplicações em rede.

Capítulo 1 – Introdução

1.1 Localização e Conceitos Preliminares

Com a tecnologia contemporânea é comum nos depararmos com a palavra estéreo associada ao som. Caixas de som independentes ou até mesmo fones de ouvido fazem com que uma música, por exemplo, chegue aos nossos ouvidos com diferenças sutis e nos dão a sensação de estarmos envolvidos por uma atmosfera tomada por aquele som.

No caso do homem, a visão também pode ser dita estéreo, uma vez que cada olho enxerga o ambiente ao redor de forma diferente – por estar separado do outro olho por alguns centímetros – e ao receber essas imagens, nosso cérebro as compõe formando uma só, provocando sensações de profundidade e imersão num ambiente.

A visão estéreo (binocular) trata do processamento de informações de um cenário a partir de duas imagens do mesmo cenário. Um erro comum é achar que "visão estéreo" é sinônimo de visão em três dimensões, quando na verdade pode-se realizar a visão em três dimensões sem que se tenha a estereoscopia. Para explicar melhor, uma apresentação mais detalhada sobre a estereoscopia é feita na Seção 2.1.

Sistemas de visão estéreo são abordados neste projeto como programas de computador que têm o objetivo de extrair informações geométricas de um cenário, a partir de duas imagens bidimensionais, usando a estereofotogrametria. Eles têm aplicações das mais diversas, como por exemplo, o uso de veículos remotamente operados para a inspeção visual de instalações submarinas de exploração de petróleo [1].

Num caso típico, a entrada de um sistema de visão estéreo binocular baseia-se em duas fotos digitais com características em comum: as dimensões e a qualidade das imagens, o instante em que as fotos são tiradas (devem ser simultâneas) e o cenário-alvo das fotos. Após terem sido capturadas, tais fotos devem sofrer um processamento a fim de, entre outras coisas, obter dados de sua calibração necessários ao funcionamento do sistema. Esta abordagem divide claramente o problema em três módulos, como mostra a Figura 1.

Figura 1 – Módulos de captura de imagens e pré-processamento compondo a entrada do sistema de visão estéreo

As características exigidas para as fotos não impedem que, freqüentemente, elas sejam tomadas de forma presencial, isto é, com um indivíduo de posse das câmeras. Algumas situações onde sistemas de visão estéreo são usados, no entanto, não permitem que isso ocorra. O caso apresentado neste projeto, onde a visão estéreo é usada para o dimensionamento de objetos em cenários do fundo do mar – a profundidades onde o ser humano não consegue chegar – exige que a captura de imagens seja controlada remotamente.

Na ocasião da proposta deste projeto, em meados de 2007, já havia um sistema de inspeção submarina pronto e em uso industrial, chamado SVE [2] cujo nome foi dado em função das iniciais de "Sistema de Visão Estereoscópica". Desde então, o SVE foi tema de revisão e atualização, sendo que atualmente está disponível um sistema multiplataforma chamado VEM [3] (Visão Estéreo Multiplataforma) com o mesmo núcleo de funções numéricas e funções de entrada e saída de dados do SVE.

Este projeto apresenta o desenvolvimento de um novo módulo de captura de imagens para o SVE: um programa de captura remota e de controle das câmeras fotográficas digitais que geram imagens para o sistema.

1.2 Organização do Trabalho

O Capítulo 2 introduz os fundamentos teóricos da estereoscopia e descreve princípios de obtenção do paralelismo em computação. Esse capítulo é seguido pelo Capítulo 3, onde são relatados os objetivos deste projeto. No Capítulo 4 são apresentados os materiais utilizados no desenvolvimento do módulo de capturas de imagens, que é abordado no Capítulo 5, com suas etapas e detalhamentos técnicos. O Capítulo 6 apresenta os resultados da pesquisa e implementação deste projeto e no Capítulo 7, são dadas as conclusões tiradas durante o processo de desenvolvimento assim como algumas propostas de trabalhos futuros para aprimorar os resultados obtidos.

Capítulo 2 – Fundamentos Teóricos

Este capítulo descreve os fundamentos teóricos utilizados na composição do projeto: uma introdução a estereoscopia e conceitos básicos sobre sistemas de visão estéreo.

2.1 Estereoscopia

Em algum momento durante sua evolução, certas espécies de animais sofreram uma importante alteração genética. Seus olhos passaram a ser dispostos na frente da cabeça, fazendo com que eles perdessem a visão de quase 360 graus que lhes garantia um excelente campo visual, mas assegurando uma característica inédita: a visão binocular ou estereoscópica.

A visão binocular pode ser explicada com um exemplo prático: basta tapar um dos olhos e tentar alcançar um objeto a alguns centímetros. É fácil notar nessa experiência certo desconforto causado pela sensação de inexatidão na distância até o objeto. Isso se dá porque na visão monocular a percepção de profundidade é precária, valendo-se apenas de projeções em perspectiva, onde o tamanho aparente de um objeto diminui com o aumento da distância dele ao observador.

Já na visão estereoscópica essa dificuldade não acontece porque o cérebro interpreta as duas imagens adquiridas por cada olho e as processa junto com as informações obtidas dos nervos oculares sobre o grau de convergência ou divergência dos eixos visuais. Isso permite ao cérebro inferir a que distância do observador a interseção dos eixos visuais se encontra em um dado momento.

Um exemplo prático foi sugerido em [4] e pode ser visto abaixo: posicione seu polegar esquerdo em frente a uma bandeirinha alinhando-os ao seu nariz e tente focar a visão no dedo. Nesta situação duas bandeirinhas são vistas, pois o eixo de visual de cada olho converge sobre o polegar. Convergindo então para a bandeirinha, a visão que se tem é a da Figura 2, onde a bandeirinha está sobre a interseção dos eixos.

Esta diferença nas imagens é processada pelo cérebro e então temos a noção de profundidade onde objetos diferentes estão posicionados a distâncias diferentes.

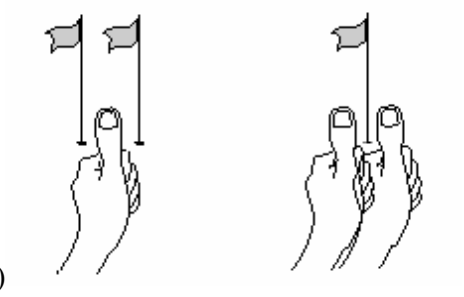

Figura 2 – (a) Com os olhos convergindo para o polegar, a bandeirinha é vista como imagem duplicada; (b) com os olhos voltados para a bandeirinha, a vez é dos dedos.

Podemos dizer que o principal componente de um sistema estereoscópico natural é o cérebro humano pois é ele que faz a fusão das duas imagens resultando em noções de profundidade [4], como ilustrado na Figura 3. Duas imagens ligeiramente diferentes da mesma cena representam o mundo como o veríamos se o cérebro não agisse.

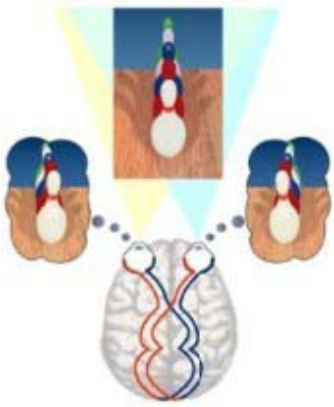

Figura 3 – O cérebro interpreta as diferentes visões da mesma cena.

Uma característica importante da diferença entre as imagens obtidas pelos olhos é a disparidade na retina, que é o espaçamento entre o mesmo ponto projetado nas duas retinas. Isso pode ser observado na Figura 4 onde o olho da esquerda observa o pinheiro à esquerda da árvore e o olho da direita observa o pinheiro à direita da árvore.

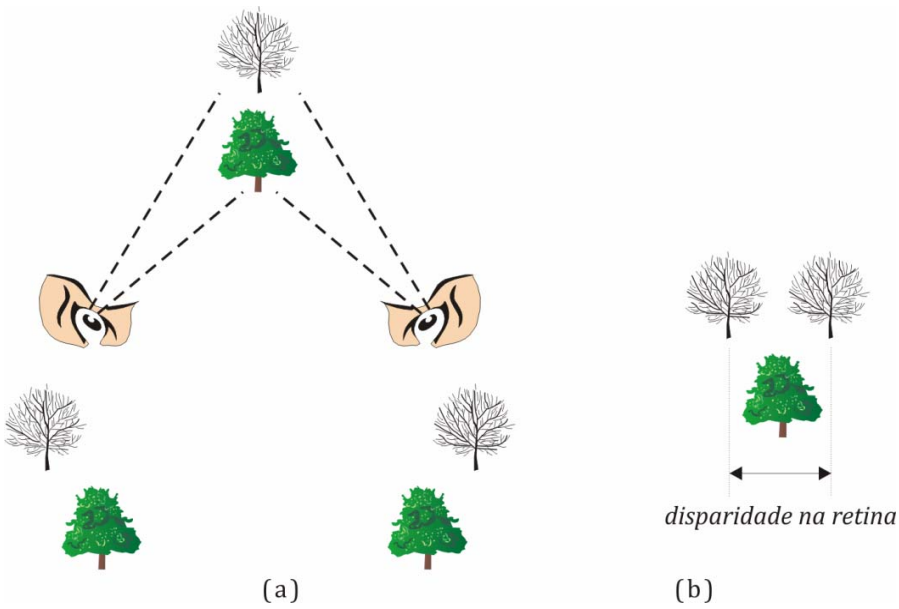

Figura 4 – (a) Visão da mesma cena pelos dois olhos; (b) superposição das imagens e disparidade da retina.

2.1.1 Técnicas e Equipamentos Estereoscópicos

O princípio de funcionamento de dispositivos estereoscópicos é apresentar imagens distintas aos olhos esquerdo e direito do observador para que ele tenha sensação de profundidade, como ocorre com sua visão natural. A seguir, são apresentadas algumas técnicas freqüentemente necessárias para a montagem de um sistema de visão estéreo. Todas estas técnicas usam dispositivos auxiliares.

2.1.1.1 Vídeo Estereoscópico

Uma técnica de geração de imagens para a visão binocular é a do vídeo estereoscópico, onde um par de câmeras é posicionado de forma a simular a distância inter-pupilar humana média. Assim, quando cada imagem for apresentada ao respectivo olho, a composição delas será feita no córtex visual do cérebro, gerando a cena estereoscópica.

Nessa técnica existem duas formas de se arranjar o par de câmeras:

- Em eixo paralelo
- Em eixo convergente

Na primeira, as câmeras são alinhadas de forma que os eixos de suas lentes estejam em paralelo, como mostrado na Figura 5(a). A convergência das imagens pode ser obtida por meio de um deslocamento horizontal das mesmas que altere a distância (reduza a disparidade) entre os pontos correspondentes nas imagens das câmeras esquerda e direita. Já na segunda forma as câmeras são giradas de modo que os eixos das lentes convirjam em algum ponto sobre o plano onde se encontra o objeto alvo, como na Figura 5(b).

Figura 5 – Arranjo das câmeras para o vídeo estereoscópico: (a) eixos paralelos; (b) eixos convergentes.

Em ambos os casos, as câmeras devem estar alinhadas horizontal e verticalmente e a separação entre as lentes deve ser de aproximadamente 65 mm.

2.1.1.2 Polarização da Luz

Para se obter visualização de imagens em três dimensões utiliza-se normalmente alguma técnica para a polarização da luz. Neste caso são empregados filtros polarizadores, que fazem com que projeções de um par de imagens estéreo em uma tela sejam polarizadas em planos ortogonais. O observador utiliza para cada imagem um filtro polarizado conforme os planos de projeção, para impedir que as duas imagens cheguem a ambos os olhos. Isso faz com que cada olho veja apenas a imagem correspondente à polarização adequada; e o cérebro se encarregue de fazer a fusão das

imagens e criar a sensação da visão estereoscópica.

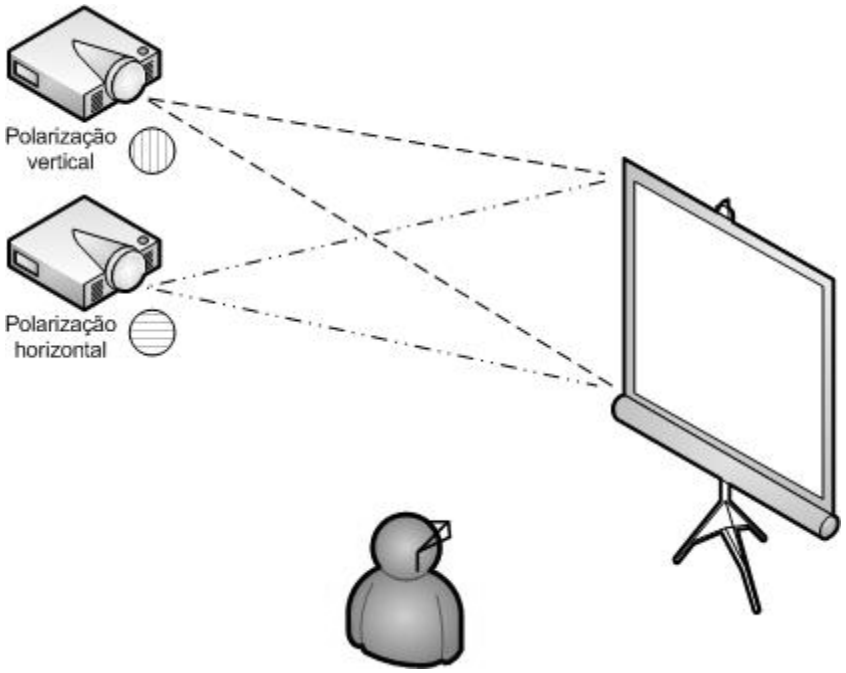

Figura 6 – Técnica de Polarização da Luz.

2.1.1.3 Óculos Obturadores Sincronizados

Nesta técnica o indivíduo assiste a um vídeo exibido num monitor comum e usa um par de óculos especiais, dotados de uma conexão com o PC e lentes feitas de cristal líquido. As lentes dos óculos ficam opacas ou transparentes conforme um sinal de sincronismo vindo do computador, que é controlado pelo sinal de vídeo. A lente esquerda fica transparente e a lente direita fica opaca quando estiver sendo exibido, no monitor, um quadro referente ao olho esquerdo e vice-versa. Isso se repete para cada par de quadros, com visão alternada entre as lentes esquerda e direita. O mesmo efeito pode ser conseguido a partir de uma tela polarizadora controlada pelo sinal de vídeo, colocada sobre o monitor [5]. O espectador assiste ao vídeo usando óculos passivos que têm a lente esquerda polarizada em uma direção e a lente direita polarizada em outra. Os óculos passivos não são controlados pelo PC.

Em geral, a taxa de atualização do vídeo é de 120 quadros por segundo (ou 120 fps), o que leva a cada olho um vídeo com, também, 60 quadros por segundo; o suficiente para ultrapassar a taxa média com que o homem consegue perceber mudanças num cenário (no centro da retina), que é de 30

fps.

2.2 Sistemas de Visão Estéreo

Uma proposta deste projeto lida com as três técnicas apresentadas na Seção 2.1.1 (o vídeo estereoscópico, polarização da luz e óculos especiais que filtram as imagens). Uma técnica não funciona sem as outras e como os óculos e a tela já estavam prontos, os esforços deveriam ser concentrados sobre a obtenção do vídeo estereoscópico. Por outro lado, havia também as exigências das fotos estéreo. Para isso é necessário obter-se simultaneidade e sincronismo no gerenciamento do par de câmeras, que são os assuntos da Seção 2.2.1.

2.2.1 Computação Paralela no Controle do par de Câmeras

Não é possível criar um sistema de visão estéreo binocular sem que se tenham imagens com as características mencionadas na Seção 1.1: fotos simultâneas. Além disso, o vídeo estereoscópico almejado exigia sincronismo entre os quadros. As exigências feitas para essas fotografias indicam que elas devem ser simultâneas, se não perfeitamente, ao menos com uma diferença de tempo tolerável pela aplicação que as usa. Já o sincronismo é importante para a formação do vídeo estereoscópico, pois cada quadro vindo de uma câmera deve ser intercalado com o quadro simultâneo da outra câmera no tempo certo. Para conseguir mostrar na tela do computador imagens sincronizadas e simultâneas é necessário entender como um sistema operacional gerencia seus recursos.

2.2.1.1 Sistema Operacional

Segundo [6] um sistema operacional pode ser definido como um gerenciador de recursos. Ele deve controlar todas as peças de um sistema complexo, como processadores, memória, *timers*, discos, interfaces de rede, e neste projeto dispositivos externos como, por exemplo, as câmeras conectadas às portas USB. O sistema operacional divide tempo e espaço disponíveis entre as requisições que chegam até ele. Imagine que três pedidos de impressão na mesma impressora fossem feitos ao mesmo tempo. Sem o sistema operacional, talvez uma linha de cada documento fosse impressa por vez, gerando um documento confuso e incorreto ao final. Para gerenciar essas requisições o sistema operacional precisa

classificar a fonte de cada pedido e agendá-lo para execução. Alguns conceitos básicos para essa classificação são enumerados a seguir.

a) Processos

Um conceito-chave usado por sistemas operacionais é o de *processo*. Um processo é basicamente um programa em execução. Cada processo tem recursos limitados, e um deles é o espaço de endereçamento (ou *address space*), que é uma lista restrita de endereços na memória onde o processo pode ler e escrever dados. O espaço de endereçamento contém o programa executável, os dados do programa e sua pilha. Outro recurso associado ao processo é um grupo de registradores, incluindo o contador do programa, o ponteiro da pilha e outros.

Um processo pode ser suspenso pelo sistema operacional quando ele ultrapassa o tempo de execução determinado pelo processador. Neste caso, o processo tem que ter seu estado gravado para que ao retornar, retome de onde parou. Com isso, um processo suspenso precisa ter uma referência armazenada numa tabela chamada tabela de processos, que mantém as informações do processo (exceto pelo espaço de endereçamento) até que ele retorne a ser executado. Portanto, um processo suspenso consiste em um elemento na tabela de processos e seu espaço de endereçamento.

Outra informação importante sobre processos é que eles podem criar sub-processos (chamados processos filhos) que podem, por sua vez, criar seus próprios processos filhos (Figura 7). Processos filhos podem estar cumprindo alguma função para seus pais e ao terminar precisam passar informações geradas ou coletadas para seus superiores. Isso nos leva a perceber que é possível haver comunicação entre processos.

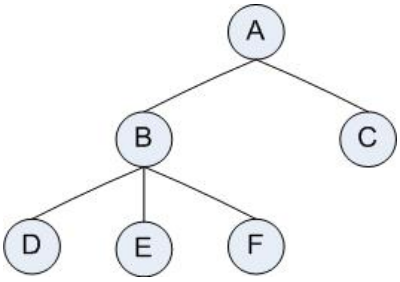

Figura 7 – Hierarquia entre processos.

Além da intercomunicação de processos, outra característica importante é a capacidade da CPU de executar mais de um processo por vez. Uma CPU consegue intercalar a execução de processos, um a um, rodando cada um deles por alguns milissegundos.

Num determinado instante de tempo, somente um processo pode estar sendo executado por uma CPU, mas durante um certo intervalo (por exemplo 1 segundo), uma CPU consegue passar por diversos programas, dando ao usuário a sensação de que os programas estão sendo executados em paralelo (Figura 8). O paralelismo real só pode ser atingido no caso de haver múltiplas CPUs ou um *hardware* específico para a aplicação desejada.

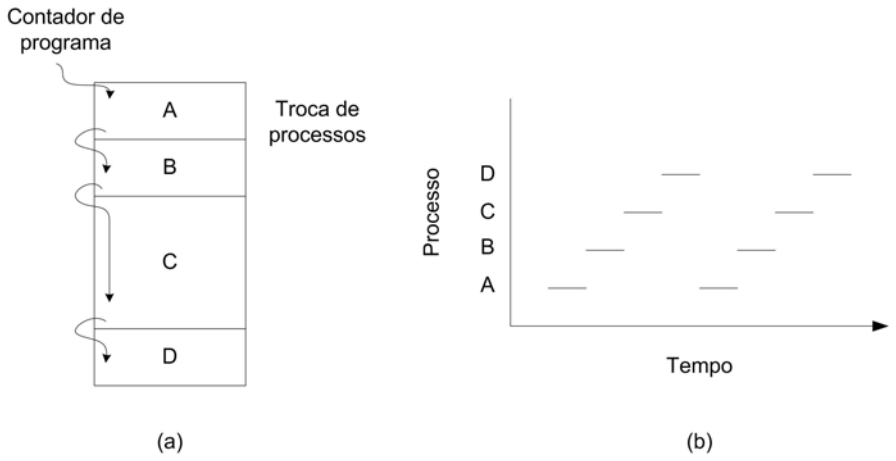

Figura 8 – (a) Multiprogramação de quatro programas. (b) Apenas um processo é executado por vez.

Cada processo tem, portanto, seu espaço de endereçamento e uma única linha de execução (ou linha de controle). Por algumas vezes é desejável que o processo tenha mais de uma linha de controle em um mesmo espaço de endereçamento, sendo executada em (falso) paralelo como se fossem processos diferentes. A função dessa linha de controle é cumprida por uma entidade chamada *thread*, e será mais bem explicada a seguir.

b) Threads

Processos podem ser definidos como uma forma de se agrupar dois conceitos: *recursos* e *execução*. Os recursos agrupados têm alguma relação entre si, e por estarem agrupados sobre uma mesma entidade podem ser gerenciados de uma forma mais inteligente. Exemplos de recursos são arquivos, processos filhos, dados do usuário, etc. A execução por sua vez fica por conta da linha de controle, ou *thread*. Uma *thread* tem um contador de programa, que aponta para a próxima instrução a ser executada, registradores para armazenar suas variáveis e uma pilha (*call stack*) que contém informações dos chamados a *procedures* que ainda não retornaram.

Embora processos e *threads* tenham uma relação de parentesco, eles são entidades que podem ser tratadas de forma independente. Processos são os agrupadores de recursos e *threads* são as entidades agendadas para execução na CPU.

Por serem independentes, múltiplas *threads* adicionam ao processo a característica de distintas linhas de execução dentro de um mesmo ambiente. Por analogia, ter diferentes *threads* em execução num processo é o mesmo que ter diferentes processos em execução numa CPU. Infere-se disso que, da mesma forma que com os processos, a CPU agenda a execução de cada *thread*.

A Figura 9 (tirada de [6]) mostra essa comparação. Em (a), cada processo tem seu espaço de endereçamento e sua *thread* de controle. Por outro lado, em (b), o que se vê é um único processo com três *threads* de controle diferentes. Embora nos dois casos haja três *threads*, no caso de (a) cada *thread* trabalha em um espaço de endereçamento diferente, enquanto em (b) todas as três *threads* compartilham o mesmo espaço de endereçamento.

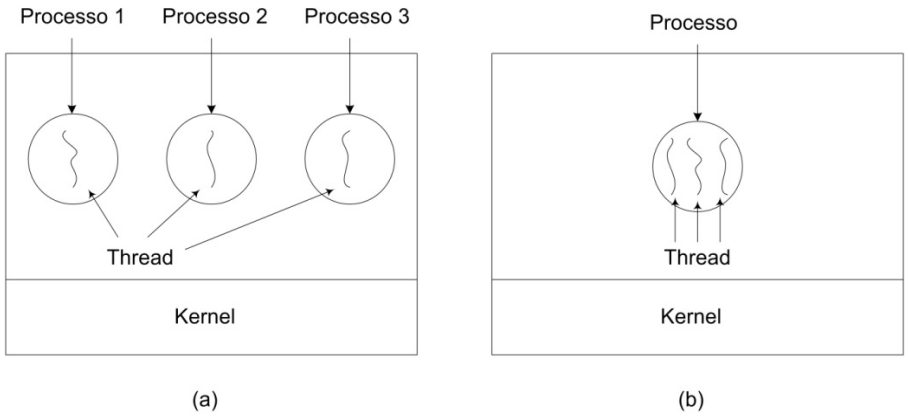

Figura 9 – (a) Três *threads*, uma em cada processo; (b) um processo com três *threads*.

Em um caso como este, onde acontece o chamado *multithreading* (mais de uma *thread* sob um mesmo processo), a CPU se reveza entre a execução das *threads* assim como faz com cada processo em separado. Isso acontece tão rapidamente que dá a impressão de que as *threads* estão sendo executadas em paralelo. Mais uma vez porém, o paralelismo não é real.

É importante notar que cada *thread* tem sua própria pilha. Por sua vez, as pilhas contêm quadros onde são guardadas as informações das *procedures* chamadas pela *thread* mas que ainda não retornaram. Essas informações incluem as variáveis locais da *procedure* e o endereço de retorno que deve ser usado quando a *procedure* terminar. Portanto, fica fácil perceber que ao criar uma *thread*, o processo deve passar uma *procedure* como parâmetro para a *thread* para que essa *procedure* seja executada. Essa *procedure* pode chamar outras *procedures*, que também irão ser colocadas na pilha da *thread*. Toda essa informação sobre *threads* revelou algumas pistas sobre as razões pelas quais alguém poderia querer usá-las:

- A principal razão é, certamente, a possibilidade de realizar tarefas em paralelo com um conjunto de dados em comum.

- Um segundo motivo é o fato de que *threads* são muito mais "leves" que processos, justamente por não terem recursos ativamente ligados a ela. Logo, elas podem ser criadas e destruídas mais rapidamente que processos.

- Em terceiro lugar, pode-se citar a agilidade de execução de tarefas que ocorre quando elas são divididas e distribuídas em tarefas menores que podem ser executadas ao mesmo tempo. Neste projeto, este ponto de vista se aplica no momento da captura das fotos, quando em vez de salvar as fotos em seqüência, a gravação é distribuída e ambas são salvas ao mesmo tempo, cada uma em seu respectivo arquivo.

Mas toda essa aparente independência e desempenho das *threads* têm seu preço. Num programa *multithreaded*, por trabalharem em um mesmo espaço de endereçamento as threads podem querer acessar a mesma região de memória ao mesmo tempo, por exemplo, uma para ler e outra para escrever. Isso causaria conflitos. A seção seguinte aborda a forma de se evitar esses conflitos entre *threads* (e igualmente entre processos).

c) Regiões Críticas e Mecanismos de Sincronização

Processos diferentes podem realizar operações que conduzam a disputas por um recurso em comum. Para evitar essas disputas é necessário usar alguma forma de contenção dos processos, de modo que seja proibido que mais de um processo tenha acesso ao mesmo recurso ao mesmo tempo. Assim dito, o que se deseja é a *exclusão mútua*, ou seja, uma maneira de se fazer com que, se um processo tem autorização para acessar um recurso compartilhado, o outro tenha sua autorização excluída. Essas operações que levam a disputas por recursos, estão em partes do programa que, por isso, são chamadas de regiões críticas. Para tratar das regiões criticas neste projeto, foram usadas duas formas conhecidas de mecanismos de sincronização via *software*: o *mutex* e a *troca de mensagens*.

Mutex: É uma variável que pode estar em um de dois estados: bloqueado ou desbloqueado. Ela é usada de maneira que quando uma *thread* precisa de acesso a uma região crítica, ela solicita o bloqueio da região chamando uma função como, por exemplo, *mutex_lock().* Se o *mutex* estiver desbloqueado, a região estará acessível e a *thread* poderá seguir em frente. Caso contrário, se o *mutex* já estiver bloqueado, a *thread* que solicitou o bloqueio do *mutex* ficará bloqueada até que a *thread* que estiver acessando a região crítica naquele momento saia e o *mutex* seja desbloqueado (usando-se algo como *mutex*_unlock()).

Se várias *threads* encontram a região bloqueada e ficam presas pelo *mutex*, quando ele for desbloqueado, uma das *threads* será escolhida aleatoriamente e entrará na região crítica, solicitando novamente o bloqueio do *mutex*.

Troca de Mensagens: outro método de comunicação entre processos é a troca de mensagens. Ele se baseia em duas funções: *send()* e *receive()*. A primeira envia uma mensagem para um único destinatário e a segunda recebe mensagens de uma ou mais origens. Se nada chegar ao receptor, ele pode bloquear a execução até que uma mensagem chegue. A troca de mensagens é largamente usada em sistemas onde os processos que querem se comunicar estão sob uma rede de computadores e isso faz com que haja diversos problemas a serem enfrentados, como por exemplo a perda de mensagens durante a transmissão. No caso deste projeto, entretanto, não há comunicação via rede e a troca de mensagens foi usada entre *threads*, onde a *thread* filha enviava para sua criadora uma mensagem e a

superiora respondia (às filhas ou ao usuário) conforme a mensagem recebida.

d) Agendamento do Processador

Sistemas operacionais para computadores pessoais modernos são multiprogramáveis, ou seja, podem executar mais de um programa ao mesmo tempo. Com isso múltiplos pedidos de execução são enviados à CPU e o sistema operacional tem que lidar com cada um deles seguindo um algoritmo de escolha – pois uma CPU só pode executar um processo por vez.

A parte do sistema operacional (SO) que faz essa escolha é chamada de agendador (ou *scheduler*) e o algoritmo que ela usa é conhecido como algoritmo de agendamento. Os agendamentos de processos e *threads* têm pontos em comum, mas há também algumas diferenças. O interesse aqui é em nível de *threads*, portanto, esta seção irá se concentrar nesse caso.

Agendamento de Processos: existem muitos algoritmos de agendamento (por exemplo *Round Robin*, Prioridades, Múltiplas Filas), mas em geral a idéia é a mesma: cada processo ganha do agendador uma fatia de tempo da CPU, chamada *quantum* (cuja ordem de grandeza normalmente é milissegundos). Quando esse tempo passa, o agendador faz com que outro processo seja executado pela CPU e o processo que estava sendo executado fica em estado de espera, aguardando ser chamado novamente para continuar seu projeto de onde parou. Metodologia semelhante é usada com o agendamento das *threads*.

Agendamento de *Threads*: quando múltiplos processos têm múltiplas *threads*, existem dois níveis de paralelismo em jogo: o de processos e o de *threads*. Dependendo de quem for encarregado do tratamento das *threads* (o *kernel* do SO ou o usuário), o agendamento da CPU será diferente. Digamos que o sistema operacional não saiba da existência das *threads* (usuário é o encarregado). Nesse caso, o agendador irá separar tempos de execução da CPU somente para processos e dará a esses processos o poder de controlar o uso deste tempo. Por exemplo, considere um processo P, que recebe do agendador um *quantum* de 50 ms e tem dentro dele três *threads*: P1, P2 e P3. Como o processo tem o controle sobre o tempo que lhe foi dado, ele decide executar P1. Como o agendador do SO não tem poder de mudar o estado das *threads*, a *thread* continua executando, até que o tempo do processo

acabe e o SO interrompa este processo. Quando P rodar novamente, P1 continuará sua execução de onde parou, até que acabe novamente o tempo de P. Mas se P1, P2 e P3 consumirem pouco tempo de P – assuma que as três *threads* executam tarefas por, por exemplo, 5 ms cada – então a CPU consegue executar a seqüência P1, P2, P3, P1, P2, P3, P1 durante os 50 ms de P. Considerando-se agora a situação onde o *kernel* tem conhecimento sobre a existência das *threads*, o *kernel* escolhe, por exemplo, P1 para executar. O processador trabalha com *quantum* de 50 ms, o que dá ao *kernel* a chance de executar, por exemplo, dez *threads* (já que cada uma teria 5ms de execução) ou um processo durante este tempo.

Levando-se em conta um outro processo, Q, que possui as *threads* Q1, Q2, Q3; uma possível seqüência de execução seria P1, Q1, P2, Q2, P3, Q3.

Nota-se com isso que em situações onde o *kernel* tem poder de agendar *threads*, a execução de uma delas pode ser mais atrasada que no outro caso por ser interrompida mais vezes. Neste projeto a metodologia adotada leva em conta que as *threads* são controladas em nível de *kernel* (pois essa é a forma como o Windows trabalha), ou seja, a consistência no tempo de execução de tarefas depende somente do sistema operacional.

Capítulo 3 – Objetivo

Neste capítulo o objetivo global é mostrado e em seguida dividido em outros específicos que visam separar o controle das câmeras da captura de imagens das mesmas.

3.1 Objetivo Global

Sistemas de visão estereoscópica têm grande aplicação na inspeção visual e no dimensionamento de objetos que estejam em áreas de difícil acesso ou até mesmo nocivas à presença humana, como por exemplo na inspeção de instalações submarinas de exploração de petróleo em águas profundas, que é o caso neste projeto.

A meta principal do projeto é fazer com que se tenha controle sobre duas câmeras usando-se as funcionalidades que são fornecidas por elas. Assim, deve-se construir um aplicativo que implemente a comunicação entre o PC e as duas câmeras a fim de se compor o módulo de entrada de um sistema de visão estéreo binocular previamente determinado (ver Seção 1.1).

Embora as funcionalidades fornecidas pela câmera sejam muitas (por exemplo, configuração de *timer* para fotos, nome do proprietário, identificação do modelo), apenas uma pequena parte delas tem aplicação prática neste projeto. Uma listagem das funcionalidades de interesse é dada na Tabela 1.

Tabela 1 – Funcionalidades das câmeras e suas aplicabilidades práticas.

Característica	Aplicação Prática
Captura remota de vídeo ao vivo	• Vídeo estereoscópico • Controle de ROVs à distância
Disparo remoto de fotografias	• Controle das câmeras à distância • Entrada do SVE
Fotografias digitais em alta resolução	• Entrada do SVE • Dimensionamento com alta precisão
Ajuste de foco, de exposição e de balanço de brancos	• Vídeo estereoscópico • Controle à distância
Controle sobre as dimensões e qualidade das fotos	• Configuração das câmeras • Entrada do SVE

3.2 Objetivos Específicos

A estratégia adotada para o desenvolvimento do projeto foi a divisão do objetivo principal em metas menores a fim de se agrupar funcionalidades semelhantes em tarefas distintas. Assim, três metas adicionais foram especificadas, conforme descrito a seguir.

3.2.1 Meta I: Captura de Vídeos Ao Vivo

Consiste em obter as imagens capturadas ao vivo pelas câmeras e mostrá-las no computador. Além disso, devia-se indicar de alguma forma a origem dos quadros obtidos (hora da câmera esquerda, hora da direita). Isso garante o uso futuro da técnica do vídeo estereoscópico com a tela e óculos apontados na Seção 2.1.1.

3.2.2 Meta II: Captura das Fotos

Pode ser dividida em duas partes: disparo remoto das duas fotos e transferência dos dados das câmeras para arquivos no computador. Como a aplicação dessas fotos é a visão estéreo, elas deveriam ser, em teoria, simultâneas. Na prática, se não for possível conseguir simultaneidade perfeita, o uso indicado para o sistema (inspeção submarina) permite que se admita que o movimento dos objetos em cena será lento o suficiente para ser desprezível entre um quadro de uma c6amera e um quadro de outra câmera.

3.2.3 Meta III: Configuração das Câmeras

Disponibilizar a configuração de propriedades das câmeras no aplicativo.

A Figura 10 ilustra a divisão do objetivo global nas suas metas menores:

Figura 10 – Objetivo principal dividido em metas.

Capítulo 4 – Material

O Capítulo 4 descreve os materiais utilizados e suas características, apresentando-os em dois grupos: *hardware* e *software*.

4.1 Hardware

Nesta Seção é apresentado o "ferramental" que deve ser usados tanto para se construir este módulo como para usá-lo comercialmente também.

4.1.1 Câmeras Digitais

Para a aquisição das imagens ao vivo e fotos foi usado um par de câmeras digitais da Canon, modelo PowerShot G7. O motivo de escolha e as características desse modelo de câmera são apresentados a seguir.

4.1.1.1 Características da Câmera

A PowerShot G7 da Canon é uma câmera que está há mais de 2 anos no mercado mas que, para os padrões atuais, ainda está entre as melhores em termos de resolução de fotos. Ela conta com um *kit* de desenvolvimento de *software*, o PsRecSDK que é uma biblioteca de funções para controle remoto de câmeras da Canon por computador. O SDK [7] pode ser usado para captura remota de imagens, controlada por computador. Este modelo de câmera tem 10 megapixels de resolução e pode ser controlada remotamente pelo PC através de um cabo USB.

Figura 11 – Câmera Fotográfica Digital Canon PowerShot G7.

Algumas características técnicas da câmera são dadas abaixo. Uma tabela retirada do *site* do fabricante [8] contendo todas as especificações disponíveis é apresentada no Apêndice A.

Tabela 2 – Resumo das características técnicas da câmera.

Característica	Valor
Formato de Gravação de Imagem (Imagem Estática)	18 Combinações: (L / M1 / M2 / M3 / S / W) x (SF / F / N)
Resolução em *Pixels* (Imagem Estática)	L: 3648 x 2736, M1: 2816 x 2112, M2: 2272 x 1704, M3: 1600 x 1200, S: 640 x 480, W: 3648 x 2048
Viewfinder	Mira com *zoom* e imagem real
Dimensões (L x A x P)	106,4 x 71,9 x 42, 5 mm

a) Viewfinder

Em fotografia, um *viewfinder* é o meio por onde o fotógrafo observa a cena para compor e dar foco à imagem. Podemos assim, vulgarmente, chamá-lo de "mira" ou *display*. Os *Viewfinders* podem ser ópticos ou eletrônicos – como no caso deste projeto. Os *viewfinders* eletrônicos vêm ganhando cada vez mais espaço com as câmeras digitais. Com o modelo de câmera usado aqui, podem-se obter fotos ao vivo. Ao mostrar essas fotos, ininterruptamente, é possível obter a mesma imagem do *viewfinder* "real" da câmera, como se fosse um vídeo. Por essa razão, o fabricante disponibiliza no SDK uma funcionalidade de obtenção de imagens ao vivo, que denomina apropriadamente de *Viewfinder*. Neste

documento a palavra *display*, se referirá ao conceito como conhecido na fotografia, enquanto que *Viewfinder*, à função do SDK que fornece essas imagens.

b) Formato de Vídeo

O vídeo dos *Viewfinders* deve ser mostrado pelo aplicativo, e para isso é necessário saber algumas características do vídeo, como por exemplo as seguintes: a que taxa o vídeo é enviado pelas câmeras para o PC, quais as dimensões das imagens que compõem este vídeo e qual o formato com que essas imagens saem das câmeras. De antemão, somente era possível saber a resposta à última das questões. As outras características (taxa e dimensões) foram definidas durante o desenvolvimento do projeto e a metodologia para isso é mostrada no Capítulo 5.

c) Fotos de Alta Resolução

O formato de arquivo usado pela PowerShot G7 é compatível com o DCF, que é uma especificação da JEITA baseada no Exif 2.2, que por sua vez usa os formatos JPEG, TIFF (revisão 6.0) e RIFF WAV, assim como *tags* para meta-dados. A câmera possui cinco dimensões de fotos diferentes e três configurações de qualidade (ver Apêndice A) para as fotos, que são disponibilizadas (por padrão) em JPEG.

A configuração com tamanho "*Large*" e qualidade "*Superfine*" produz fotos como a da Figura 12. Essas fotos utilizam os 10 megapixels disponíveis no sensor e são mais bem aproveitadas (que fotos de qualidade mais baixa) pelo sistema de visão estéreo.

Figura 12 – Foto tirada a uma distância de aproximadamente 0,7 km mostrando a alta resolução da câmera.

4.1.2 Ambiente-Alvo

As especificações dos ambientes alvo são delimitadas basicamente pelas necessidades do SDK. Para que o sistema alvo possa rodar a aplicação cliente, desenvolvida com o PsRecSDK, o computador deve possuir as seguintes características:

- Processador Pentium ou melhor
- Pelo menos 64 MB de RAM (exceto Windows 2000 SP4/XP)
- Pelo menos 128 MB de RAM (Windows 2000 SP4/XP)
- Adaptador de vídeo e monitor com pelo menos 800 x 600 *pixels* e 256 cores (8 bits)
- Sistema operacional Windows 98SE, Windows ME, Windows 2000 ou Windows XP
- Porta paralela para a saída do sinal de sincronismo para a tela polarizadora
- Porta USB (no caso de só haver uma, é preciso também um *hub* USB para conectar-se as duas câmeras. Porém, este caso não foi testado e pode haver redução na velocidade de captura de imagens.)

4.1.3 Ambiente de Desenvolvimento

O ambiente de desenvolvimento também tinha algumas restrições, dadas abaixo:

- Processador Pentium ou melhor
- Pelo menos 64 MB de RAM (exceto Windows 2000 SP4/XP)
- Pelo menos 128 MB de RAM (Windows 2000 SP4/XP)
- Adaptador de vídeo e monitor com pelo menos 800 x 600 *pixels* e 256 cores (8 bits)
- Sistema operacional Windows 98SE, Windows ME, Windows 2000 ou Windows XP

- O *framework* de desenvolvimento escolhido foi o Microsoft Visual C++ 6.0 (o utilizado neste projeto possui licença). Embora não houvesse restrições quanto a que ferramenta usar, esta escolha foi feita por se tratar de uma aplicação amplamente usada e bem-aceita (pensando-se no reaproveitamento do código-fonte gerado neste projeto).

- Porta paralela

- Portas USB

4.1.4 Óculos e Tela Polarizadora

Trata-se de óculos com lentes polarizadas, uma na vertical (esquerda) e outra na horizontal. A tela é posta sobre o monitor onde é mostrado o vídeo estereoscópico e tem a função de polarizar a luz emitida pelo monitor. Um sinal de sincronismo é enviado pelo PC para a tela polarizadora, acompanhando a troca de quadros do vídeo e alternando a orientação da polarização da luz que a atravessa. Apesar de não ter sido usado, este material estava disponível para uma futura fase de observação do vídeo estereoscópico.

4.1.5 Suporte

Um suporte metálico mostrado na Figura 13 foi construído especialmente para alinhar as câmeras e colocá-las a uma distância mais próxima possível da distância inter-pupilar humana. A menor distância possível entre as câmeras é 14 cm (superior aos 65 mm dos seres humanos) e a distância pode ser aumentada de 2 cm em 2 cm até cerca de 40 cm. Note (a partir da Figura 11) que cada câmera tem aproximadamente 11 cm de largura, sem contar o espaço ocupado por conectores mini-USB.

Figura 13 – Suporte metálico.

4.2 Software

Aqui são apresentadas bibliotecas de suporte usadas na construção do projeto.

4.2.1 SDK

Como já foi dito antes, o SDK é um *kit* de desenvolvimento de *software*. É disponibilizado pelo fabricante em forma de uma biblioteca (escrita em C) de funções para controle das câmeras [7]. A biblioteca deve ser usada junto ao aplicativo que for desenvolvido.

4.2.2 JPEGLIB

Esta é uma biblioteca do IJG que possui funções de codificação e decodificação de imagens em formato JPEG. Ela foi usada na etapa de aquisição de vídeos ao vivo, para converter as imagens JPEG das câmeras para *bitmaps* (formato usado pelo Windows para exibição de imagens na tela).

Capítulo 5 – Desenvolvimento

O desenvolvimento deste projeto foi dividido em oito etapas, que são apresentadas a seguir.

5.1 Etapa 1: Definição do Método de Abordagem

Inicialmente havia algumas opções para se atingir a meta principal do projeto: o uso do SDK, a construção de um *hardware* específico para captura dos vídeos e fotos, o uso de uma biblioteca de captura de imagens de *webcams* chamada videoInput e o uso de outra biblioteca – da PUC (Pontifícia Universidade Católica), que também capturava imagens de *webcams*.

Após se analisar a documentação do SDK, verificou-se que havia um diferencial nele se comparado ao Windows XP e seus *device drivers* mais modernos[1]. Embora ambos disponibilizassem o disparo e captura de fotos remotamente; com o SDK era possível também obter as imagens ao vivo dos *viewfinders* das câmeras. Neste ponto o SDK demonstrou-se perfeitamente alinhado com os objetivos deste projeto e esta foi uma característica determinante para sua escolha.

A documentação do *kit* de desenvolvimento também rendeu a descoberta de que, usando o SDK, era possível controlar duas câmeras sob um mesmo processo, mas que uma câmera poderia ser controlada apenas por um programa e não mais que isso. Além desse motivo, as outras vantagens sobre os demais eram: a total compatibilidade com as câmeras, a disponibilidade (de parte) do código fonte do SDK e a documentação bem feita.

Os primeiros passos seriam em direção à integração do SDK a uma aplicação com interface gráfica o mais simples possível. Para criar a interface gráfica havia a possibilidade de se usar a wxWidgets (uma biblioteca multiplataforma de código aberto) ou a MFC (*Microsoft Foundation Class*). A segunda opção foi escolhida em detrimento da primeira por duas razões:

i. O fato de o SDK ter sido desenvolvido para a plataforma Windows faz com que a tentativa de criação de um programa multiplataforma perca o sentido.

[1] Até o mês de apresentação deste projeto.

ii. O ambiente de desenvolvimento especificado utilizaria o Microsoft Visual C++ 6.0 (por recomendação do manual do SDK), e a integração deste aplicativo com a MFC já estava pronta – ao contrário da wxWidgets.

A decisão tomada levou em consideração ainda a documentação da MFC, com detalhamento muito superior à documentação da wxWidgets.

5.2 Etapa 2: Estudo do SDK

Após escolher um modo de abordagem ao problema, foi preciso fazer alguns testes e entrar em contato com a forma de reação das câmeras ao uso do SDK. O que se queria aqui era descobrir como usar o SDK. Ficou decidido que, como esta era uma etapa mais de reconhecimento que de desenvolvimento propriamente dito, apenas uma câmera seria usada na maior parte do tempo.

Além do manual, o SDK vinha acompanhado de um programa de exemplo [7] que usava diversas de suas funcionalidades; e que foi usado como referência neste projeto.

Feita a escolha da MFC, a aplicação gráfica foi criada e a partir daí a integração com o SDK foi muito simples, apenas adicionando-se a biblioteca (DLL) respectiva ao projeto no Visual C++. Inferiu-se do manual que uma estrutura de dados é responsável pela identificação das câmeras. Uma delas é a estrutura prDeviceInfoTable, cujos membros mais importantes para este relatório são o nome do modelo e o nome interno (dado pelo sistema operacional) do dispositivo. Outro fator importante detectado nesta fase foi que o SDK trabalhava com funções de *callback* para enviar comandos e principalmente receber dados das câmeras.

Funções de *callback* são funções que são chamadas por outras funções [9]. Elas permitem que funções em uma camada de baixo nível em um programa chamem funções em uma camada de nível superior. Por exemplo, uma biblioteca chama uma função escrita pelo desenvolvedor de um aplicativo (Figura 14). Em alguns casos, essas funções são registradas na camada mais baixa e chamadas pela mesma somente quando um determinado evento ocorre. Este é o caso deste projeto, onde as câmeras chamam funções de *callback* disparadas por eventos de causa externa.

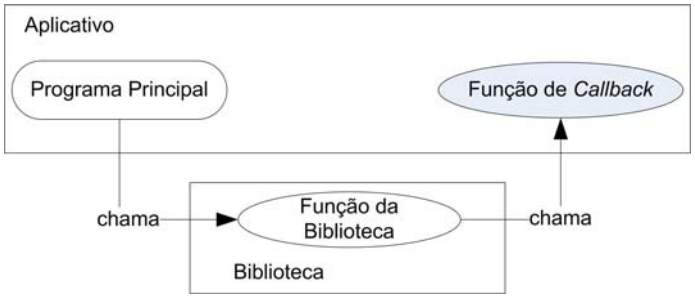

Figura 14 – Exemplo de uso geral de funções de *callback*

O SDK é dividido em alguns grupos de funções:

- Funções básicas (conectar, desconectar, etc.)
- Funções de controle remoto de captura (iniciar controle, disparar foto, etc.)
- Funções de *display* (iniciar *Viewfinder*, auto-ajuste de imagem, etc.)
- Funções de ajuste de propriedades (obter qualidade das fotos, ajustar dimensões das fotos, etc.)

O passo seguinte é implementar o controle de liga/desliga (conectar/desconectar) das duas câmeras. Para conseguir isso, no entanto, é necessário passar pela implementação das rotinas de identificação das câmeras.

5.3 Etapa 3: Definição do Protocolo de Identificação das Câmeras

Conectar apenas uma câmera é diferente de ter as duas sob responsabilidade, uma vez que deve haver distinção entre qual é a câmera direita e qual é a esquerda. Se as referências estiverem trocadas, as imagens na tela também estarão. Cada câmera deve ser identificada de maneira única pelo SDK, e o meio mais simples de se descobrir como o SDK faz isso é verificar a estrutura prDeviceInfoTable após a conexão, já que ela contém o nome interno do dispositivo (*DeviceInternalName*). Após a verificação constata-se, então, que esse nome é um conjunto de caracteres que se assemelha muito a nomes de chaves no registro do Windows.

Verifica-se, com isso, que a forma como as câmeras ficam disponíveis ao SDK é dependente da

identificação feita no sistema operacional. O Windows reconhece as câmeras como dispositivos USB e as configura, criando chaves no seu registro que contêm informações importantes sobre cada câmera.

No Windows XP os dispositivos instalados são classificados e em seguida registrados sob a chave HKEY_LOCAL_MACHINE\SYSTEM\ControlSet00x\Control\Class. Para cada classe de dispositivo é criada uma chave que a rotula (por exemplo, "Dispositivos de imagens", "Dispositivos de infravermelho", etc.). Dentro dessas chaves (Figura 15), cada aparelho que se instala deixa seu carimbo, isto é, cria uma chave numérica, que é incrementada automaticamente a cada instalação. O dispositivo preenche esta chave com alguns dados seus como por exemplo "*DeviceType*", "*DeviceID*", etc. Este carimbo continua no registro, mesmo depois que o computador é desligado.

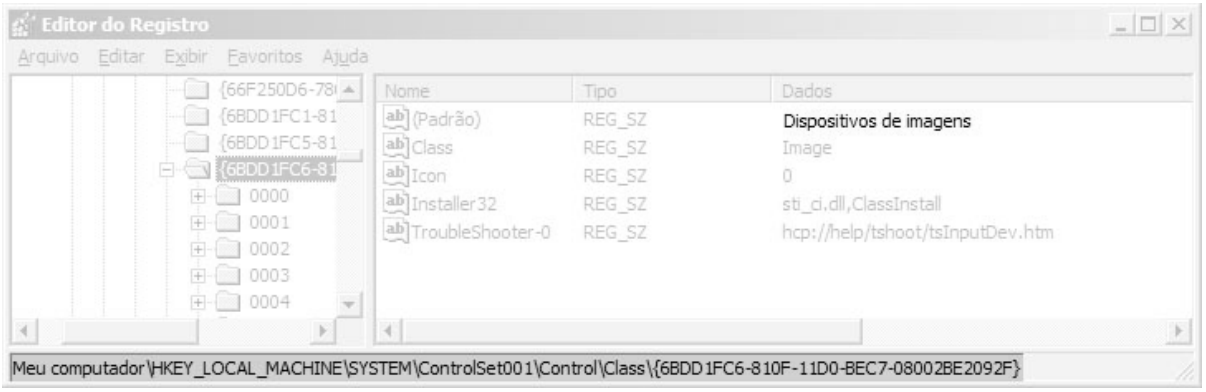

Figura 15 – Câmeras são consideradas "Dispositivos de imagens" e ficam na chave em destaque.

Uma outra classificação dos dispositivos é feita em HKEY_LOCAL_MACHINE\SYSTEM\ControlSet00x\Enum\USB, onde os dispositivos são agrupados por semelhanças, em chaves cujos nomes contêm um identificador do tipo do dispositivo e outro do fabricante, no formato VID_VVVV&PID_PPPP. De acordo com a Microsoft, VID significa *Vendor ID* e VVVV são dois pares de bytes que identificam o fabricante. Da mesma forma, PID significa *Product ID* e PPPP identifica o dispositivo.

No caso das câmeras Canon PowerShot G7 a chave criada tem nome Vid_04a9&Pid_3125. Dentro dessa chave é criada uma outra para cada câmera deste modelo que for encontrada e estiver atualmente em uso. Um atributo importante desta última chave é o "*Driver"*, mostrado na Figura 16, que corresponde ao *DeviceInternalName* no SDK.

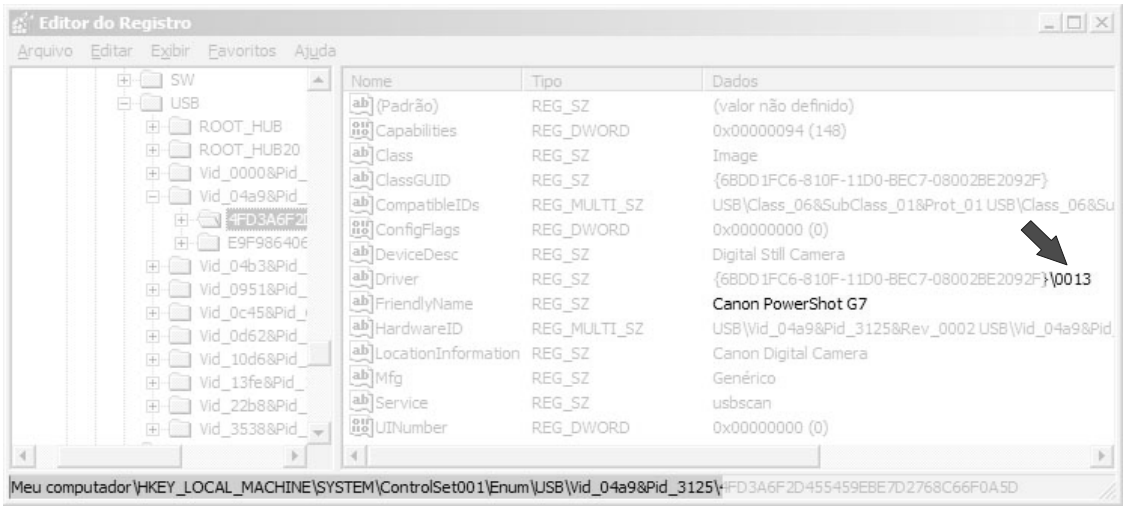

Figura 16 – Identificação de uma câmera, com seu número único no registro apontado pela seta.

Sob a chave de nome "Vid_04a9&Pid_3125" ficam todas as câmeras deste modelo encontradas pelo sistema operacional. E cada uma tem um identificador ("*Driver*") cujo valor é o da chave que identifica a classe do dispositivo ("Dispositivos de imagens") mais o número auto-incrementado que foi citado anteriormente.

Sendo assim, a chave de valor "{IDENTIFICACAO_DA_G7}\0013" identifica uma câmera, enquanto, por exemplo, a chave de valor "{IDENTIFICACAO_DA_G7}\0007" identifica outra.

Finalmente, o protocolo de identificação definido neste projeto é mostrado a seguir:

1. Conectar uma câmera após a outra ao PC, ligá-las e aguardar que sejam instaladas pelo sistema operacional.
2. Ao serem instaladas, as câmeras ganham números auto-incrementados (identificadores) que ficam armazenados no registro.
3. Na primeira vez em que as câmeras são conectadas, a primeira câmera reconhecida pelo Windows ficará com o menor número auto-incrementado.
4. Se não for a primeira vez em que se conecta as câmeras, deve-se executar o aplicativo e verificar através das imagens[2] qual é a esquerda ou a direita.

[2] Alternativamente, como não havia imagens nesta fase, a verificação era feita diretamente no registro.

5. O SDK lista as câmeras colocando-as em ordem crescente conforme o nome da chave. Portanto a câmera que tiver o menor número auto-incrementado será a primeira da lista.
6. O aplicativo considera que a primeira câmera a ser identificada pelo SDK será a esquerda, a segunda será a direita.

A forma de identificação pelo aplicativo apresenta uma inconveniência: uma vez que as câmeras estejam fixadas ao suporte, o aplicativo deve ser utilizado para verificar a coerência das posições. Caso haja algum engano no momento da instalação das câmeras no suporte (isto é, câmera esquerda colocada no lado direito e vice-versa), as imagens ficarão trocadas no aplicativo. Há uma sugestão de trabalho futuro sobre este ponto, entretanto, para implementação de um botão de "troca de lados" das câmeras, ou seja, a esquerda vira direita e vice-versa.

Com o protocolo definido, a conexão de duas câmeras pode ser concluída. Todavia, antes disso é importante existir uma etapa de modelagem e definição das técnicas de abordagem para resolução dos subobjetivos.

5.4 Etapa 4: Modelagem e Técnicas a Aplicar

Depois de ter uma idéia inicial de como o SDK se comportava e de já ter estabelecido o protocolo de identificação das câmeras, havia a necessidade de se determinar as classes (abstrações) necessárias para o desenvolvimento do código, assim como o diagrama de transição de estados UML (descrito em 5.4.1). Outro ponto a ser discutido nesta etapa é como atingir a simultaneidade e sincronia na obtenção das imagens das câmeras.

O objetivo deste capítulo não é entrar no mérito da engenharia de *software*, que é usada neste projeto bastante superficialmente, somente como uma ferramenta de suporte para modelagem. Um projeto completo de *software* usaria muitas outras ferramentas UML que não a simples identificação de classes e o diagrama apresentado aqui.

5.4.1 Modelagem

Com uma análise das necessidades do projeto, as seguintes entidades são identificadas[3]:

Tabela 3 – Entidades identificadas na etapa de modelagem.

Entidade	Papel
Câmera	Representar as câmeras
Sincronizador	Sincronizar os disparos das fotos
Selecionador de Quadro	Escolher qual quadro desenhar no vídeo estereoscópico
Módulo de Controle e Captura	Englobar todas as outras entidades

Da Tabela 3 é possível inferir quais classes irão existir no sistema, uma vez que cada classe é representada por uma entidade. Uma classe, no entanto, pode ser representada por mais de um objeto, como é o caso da câmera; ou seja; na aplicação real, existem duas câmeras, mas a modelagem feita para uma câmera deve ser suficiente para que a segunda possa ser introduzida ao projeto.

O diagrama de transição de estados (DTE) é um diagrama que indica quais e conseqüentemente quantos estados existirão no sistema; o que equivale ao número de telas que devem existir no programa. Logo, da Figura 17 pode-se deduzir que o programa deve ter ao menos cinco telas diferentes pois tem cinco estados principais (excluindo-se os estados inicial e final, representados de forma diferente, que são estados que apenas indicam onde começa e termina a execução do programa).

Neste projeto, a tela inicial é onde se espera pela ordem do usuário de conectar as duas câmeras. Enquanto duas e somente duas câmeras não estiverem presentes, não é feita a conexão. A tela seguinte a de conexão mostra as opções que o usuário terá: iniciar os *Viewfinders*, tirar uma foto, configurar as câmeras ou desconectá-las. Caso o comando dado pelo usuário nesta tela seja o de exibição dos *Viewfinders*, ele verá agora, a opção de exibição do vídeo estereoscópico e as configurações de auto-ajuste de foco, exposição e balanço de brancos. O programa terminará quando o usuário fechá-lo, a partir de qualquer tela.

[3] Pode-se dizer que entidades são todos os indivíduos que podem ser identificados como tendo uma função no projeto. Note que a identificação de entidades varia conforme o nível de abstração que se fizer do sistema, e quanto mais entidades forem identificadas, mais detalhado fica o projeto.

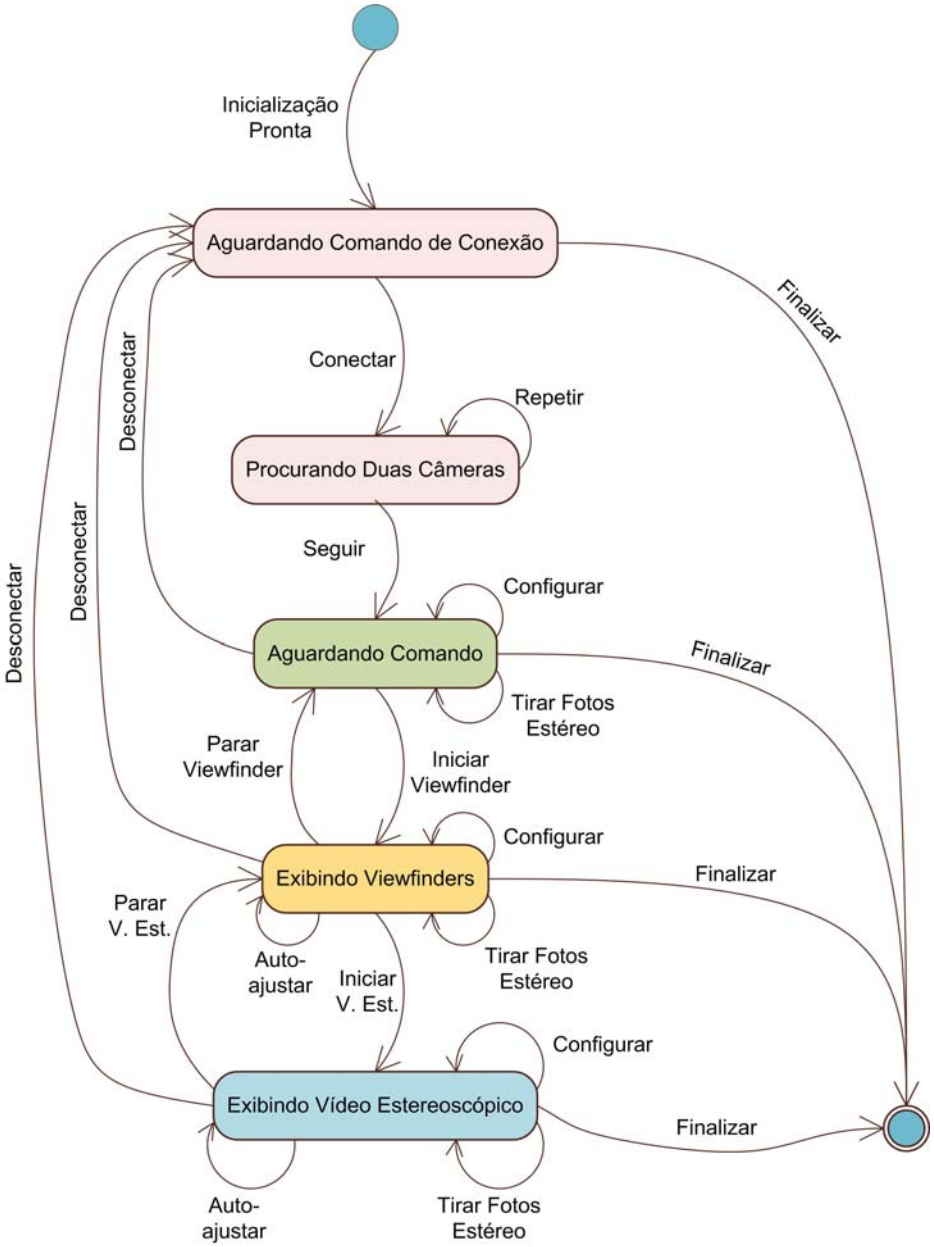

Figura 17 – Diagrama de Transição de Estados

5.4.2 Técnicas a Aplicar

De forma análoga foi feita uma análise para se detectar as formas de solução dos problemas apresentados com os objetivos específicos do projeto.

Problema 1: Captura e exibição de imagens simultaneamente.

Solução: Usar *Threads* (ver Capítulo 2 para explicações sobre threads e razões para seu uso).

Como foi dito antes, as câmeras enviam dados através de funções de *callback*. No caso do *Viewfinder*, existe uma função do SDK (PR_StartViewFinder()) que o inicia e registra em uma câmera a função de *callback* que receberá os quadros.

Seja, por exemplo, a função de *callback* VF_CB_Function(). No momento em que é executada, PR_StartViewFinder() obtém um quadro da câmera e o disponibiliza em um *buffer*, passando a VF_CB_Function() o endereço e o tamanho do *buffer*. Assim, VF_CB_Function() é executada e, ao retornar, a câmera a chama de novo, com um novo quadro no *buffer*. Ou seja, cada quadro novo só é fornecido quando VF_CB_Function() tem sua execução terminada e retorna.

Ao executar, VF_CB_Function() poderia imediatamente desenhar a imagem na tela, mas como esse é um processo significativamente demorado para uma aplicação que requer tempo real, o próximo quadro da câmera só estaria disponível algum tempo depois do que na verdade ele poderia estar pronto. Ou seja, se fosse feita a opção de se desenhar os quadros dentro da função de *callback*, a taxa com que a câmera disponibiliza quadros seria mais baixa que a taxa máxima disponível.

Além disso, as tarefas de aquisição e pintura dos quadros são independentes, e portanto podem ser executadas de forma igualmente independente. No caso das fotos, argumento semelhante é usado. As fotos são quadros de imagens com dimensões maiores que as dos *Viewfinders*. Portanto, o tamanho do *buffer* é ainda maior, o que, obviamente, implica lentidão maior. Assim, o *download* da imagem seria muito demorado e possibilidade de se tirar fotos em intervalos de tempo curtos seria ínfima. Aqui nota-se também a aplicação do motivo número três de uso das *threads*: a divisão de trabalho. Como as tarefas de disparo e captura das fotos são independentes, o uso de *threads* agiliza a captura de fotos.

Problema 2: Comunicação entre *threads*

Solução: Usar *Mutex* e Troca de Mensagens (ver Capítulo 2)

Por fazer uso de *threads* para captura de imagens em paralelo com as funções de *callback*, há a necessidade de comunicação entre as linhas de execução. Enquanto uma função de *callback* captura os dados e os armazena num *buffer*, a *thread* de pintura pega esses dados e os desenha na tela. Se houver pedidos de leitura e escrita ao *buffer* ao mesmo tempo, haverá conflitos e, possivelmente, inconsistência de dados.

Para solucionar este problema, são usados *mutexes* em ambas as funções. Um *mutex* criado na região de memória do processo pai proíbe que haja acessos conflitantes aos *buffers* simultaneamente. O *mutex* requer que uma *thread* tenha deixado a região de memória antes que outra possa acessá-la.

A troca de mensagens também é usada em alguns casos onde o início ou fim de eventos são fatores determinantes para o sincronismo de algumas tarefas como, por exemplo o momento em que o disparo de uma foto termina e a captura desses dados começa.

Problema 3: Formato das imagens incompatível com formato de desenho na tela

Solução: Usar JPEGLIB (ver Capítulo 4)

Na Seção 4.1.1.1c) foi dito que o formato de imagem que as câmeras disponibilizam é o JPEG. No entanto, para mostrar uma imagem na tela o Windows precisa que um mapa de bits (*bitmap*) seja enviado para a GDI (responsável pela renderização na tela). Para fazer essa conversão usa-se a JPEGLIB. Com ela é possível converter imagens JPEG das câmeras para mapas de bits com chamados a algumas funções. É importante notar que esse processo de conversão deve ser feito a cada quadro recebido e, portanto, adiciona certo custo à tarefa de pintar quadros na tela. Isto fortalece a idéia da solução ao Problema 1, de que seriam necessárias *threads* rodando em paralelo para desenhar os quadros.

Em resumo, as técnicas a aplicar são: *threads*, *mutexes*, troca de mensagens e a biblioteca JPEGLIB.

5.5 Etapa 5: Conexão e Desconexão das Câmeras

Após mapear as entidades em classes e estabelecer parâmetros iniciais para as soluções esperadas, esta etapa inicia o desenvolvimento concreto do projeto. Esta é uma etapa curta, que lida com o desenvolvimento do código de conexão e desconexão das câmeras.

Segundo o diagrama de transição de estados estabelecido, o primeiro passo é aguardar a ordem do usuário para então detectar duas câmeras conectadas ao computador. Assim que duas (e somente duas) são detectadas, seus dados são adquiridos e colocados na estrutura prDeviceInfoTable.

Imediatamente depois, a identificação delas é feita seguida pela conexão: primeiro a câmera esquerda, depois a direita. Cada câmera recebe, então, do SDK um número identificador único: um *handle*.

A partir deste momento as câmeras estão prontas para tirar fotos, fornecer *Viewfinders*, admitir configurações ou mesmo serem desconectadas. A desconexão pode ser feita a qualquer momento, a não ser pelo período em que fotos estiverem sendo capturadas.

5.6 Etapa 6: Captura dos Vídeos ao Vivo (Meta I)

Esta etapa trata da captura de vídeos ao vivo; o que na "linguagem" do SDK quer dizer captura e exibição dos *Viewfinders*. Em seguida deve ser feita uma sobreposição de quadros de cada *Viewfinder* com o intuito de se formar um vídeo estereoscópico a uma taxa de, se possível, 120 fps.

5.6.1 Captura dos *Viewfinders*

O planejamento para esta etapa é a aquisição das imagens ao vivo das duas câmeras, simultaneamente. Algumas condições para o sucesso desta etapa, são definidas. Do programa de exemplo do SDK, tira-se a primeira delas: a aquisição de *Viewfinders* só pode ser feita se não estiverem sendo capturadas as fotos. E ainda, se os *Viewfinders* estiverem sendo obtidos e for dada a ordem de captura de fotos, a aquisição de quadros dos *Viewfinders* será interrompida. Outra condição é a seguinte: a captura de dados dos *Viewfinders* só pode ser feita de forma que uma simultaneidade quase real seja obtida. É interessante lembrar que, como dito na Seção 2.2.1.1.a), não é possível obter-se simultaneidade perfeita com o modelo de sistema operacional que se tem em computadores pessoais (ambiente alvo do SDK).

Mas apesar de algumas exigências, a maior parte do trabalho da captura é feito pelas câmeras, que geram as imagens e as disponibilizam em um *buffer* no PC. A captura como é chamada nesta etapa, corresponde a receber o endereço e tamanho deste *buffer*. Quem recebe estes dados é a função de *callback* registrada em cada câmera.

Para simplificar, ambas as câmeras usam o mesmo canal para responder a solicitações de quadros, ou seja, a mesma função de *callback*. Isso é possível pois a função de *callback* tem como um dos parâmetros o *handle* da câmera. O nome dado para a função de *callback* é

ViewFinderCallBackFun().

Em ViewFinderCallBackFun() é feito o armazenamento dos quadros, recém adquiridos, em buffers específicos (da classe TCamera) para cada câmera. Cada vez que essa função retorna, a câmera correspondente a chama novamente. E cada vez que a função é chamada, um quadro novo é dado pela câmera responsável pelo chamado. Isso acontece até que a câmera receba a ordem de encerrar o Viewfinder.

Dessa forma, é possível verificar a taxa com que as câmeras disponibilizam seus quadros. Para fazer isso, uma idéia é imprimir num *log*, o handle da câmera e o tempo corrente (em segundos) do momento em que a função de callback do Viewfinder é chamada. Uma série de testes baseados nesta idéia foi feita e os resultados são listados em anexo (Apêndice B). Com esses testes, pode-se verificar que a taxa de renovação de quadros das câmeras é de 60 fps para cada câmera.

Tabela 4 – Análise da taxa de atualização de *frames* para uma câmera.

1	handle=9248568 @ t=5.73400		12	handle=9248568 @ t=5.92100
2	handle=9248568 @ t=5.76500		13	handle=9248568 @ t=5.93700
3	handle=9248568 @ t=5.76500		14	handle=9248568 @ t=5.95300
4	handle=9248568 @ t=5.79600		15	handle=9248568 @ t=5.96800
5	handle=9248568 @ t=5.79600		16	handle=9248568 @ t=5.98400
6	handle=9248568 @ t=5.82800		17	handle=9248568 @ t=6.00000
7	handle=9248568 @ t=5.84300		18	handle=9248568 @ t=6.01500
8	handle=9248568 @ t=5.85900		19	handle=9248568 @ t=6.03100
9	handle=9248568 @ t=5.87500		20	handle=9248568 @ t=6.04600
10	handle=9248568 @ t=5.89000		21	handle=9248568 @ t=6.06200
11	handle=9248568 @ t=5.90600			

Analisando um trecho dessa lista filtrado para apenas uma câmera (Tabela 4), pode-se notar um fato interessante. Na linha 1, um quadro é recebido pela função de *callback* em 5,734 s. Como é sabido que a taxa de atualização de quadros de uma câmera é 60 fps, o quadro da linha 2 deveria ter sido recebido em (aproximadamente) 5,750 s, mas isso não ocorreu. Nota-se também, que o tempo marcado na linha 2 é o mesmo marcado na linha 3. O que indica que a execução (da função de *callback*) correspondente à linha 2 foi atrasada, e que, na verdade, só ocorreu ao mesmo tempo em que a da linha 3. Isso é explicado pelo escalonamento de processos que o sistema operacional faz. A execução equivalente à linha 2 foi postergada pelo sistema operacional, e a *thread* correspondente só saiu do estado *wait* quando ocorreu a execução da linha 3. Assim, o quadro correspondente à linha 2 chegou junto com o quadro correspondente à linha 3.

Nota-se com isso que é preciso usar um mecanismo de proteção para que duas chamadas

executadas ao mesmo tempo não tentem usar a mesma região de memória. Para isso um *mutex* é usado. Quando é feito um chamado à função de *callback*, o *mutex* é bloqueado e protege a região de conflito, e o *mutex* só é desbloqueado quando não há mais riscos de conflito na região.

5.6.2 Exibição na Tela

Exibir os quadros na tela após capturá-los parece muito natural. No entanto, este processo exige cuidado. Cada câmera fornece seu próprio vídeo, independentemente da outra. Por essa e outras razões já citadas, são usadas *threads* para a pintura dos quadros na tela, uma para cada câmera.

Cada *thread* deve fazer, estritamente, o trabalho da pintura em seu respectivo molde na tela (área reservada para a pintura do vídeo), sem interferir nas regiões críticas de memória compartilhada. Portanto, para não haver conflitos com o *buffer* das câmeras – que é escrito pela função de *callback* – as *threads* permanecem em constante estado de espera, até que um quadro seja recebido pelo *buffer* da câmera.

A própria função de *callback*, ao receber o quadro e alocá-lo no *buffer* da câmera, "acorda" a *thread* da câmera respectiva (usando a função ResumeThread(), da MFC [10]), fazendo com que ela seja executada. Quando executada, a *thread* pinta o quadro na tela e em seguida coloca a si própria no estado de espera com a função SuspendThread(). A *thread* só é chamada novamente quando um novo quadro chega. Nota-se, portanto, que as *threads* estão sincronizadas com a função de *callback* e que, caso haja algum atraso na execução da função de *callback* (como é demonstrado com a Tabela 4), a execução da *thread* correspondente também será atrasada. Logo, é necessário usar um *mutex* também em cada uma dessas *threads* para evitar que duas execuções de uma mesma *thread* tentem ler o *buffer* ao mesmo tempo.

A tabela de características da câmera (Apêndice A) indica que ela pode fazer "filmes" com um determinado conjunto de formatos. Não há, contudo, como ter certeza de que algum deles é compatível com o formato do *Viewfinder,* pois o manual do SDK não aponta essa informação. No entanto, do programa de exemplo do SDK podem ser tiradas as dimensões dos quadros do *Viewfinder*: 320 x 240 *pixels*. Para confirmar que essas são mesmo as dimensões do quadro e não apenas as dimensões que o programa de exemplo usa, alteram-se a largura e altura do molde usado no aplicativo deste projeto para, por exemplo, 160 x 120 *pixels* (simulando as dimensões de um outro formato da tabela do Apêndice A. Ainda assim as imagens permanecem com 320 x 240 *pixels*. Sendo assim,

observando-se o Apêndice A, verifica-se que a qualidade de vídeo "*Fast Frame Rate*" é a que está, de fato, relacionada ao *Viewfinder*, pois com esta qualidade se obtêm dimensões de 320 x 240 *pixels* e uma taxa de atualização de 60 fps.

5.6.3 Sobreposição de Quadros

O passo seguinte é a sobreposição de quadros intermitentes das câmeras esquerda e direita, visando à constituição do vídeo estereoscópico. O vídeo por sua vez deve ser formado por todos os quadros de ambas as câmeras a uma taxa de, preferencialmente, 120 fps. No entanto, como a maioria dos monitores de computadores pessoais trabalha com sincronismo vertical de 60 Hz, se uma taxa de 60 fps for alcançada, será suficiente para se obter o efeito desejado de forma aproximada com placas de vídeo comuns em PCs.

Para atingir esse valor, uma sugestão de implementação é a seguinte: usar uma entidade independente das câmeras que leia os *buffers* e os imprima na tela, intercaladamente, na taxa desejada. Neste caso, como a taxa desejada para o vídeo é de 60 fps, cada quadro deve ser impresso a cada 1/60 s (~16ms). Assim, essa entidade – um objeto da classe TFrameSelector – escolhe alternadamente entre as câmeras esquerda e direita, lê um quadro do *buffer* respectivo e o imprime na tela.

Entretanto, TFrameSelector é uma entidade passiva, que só deve ser acionada a cada intervalo regular de tempo. Para fazer essa ativação, deve ser usado um *timer*, que nada mais é que um contador de tempo que gera um evento a cada intervalo Δt. Esse *timer* deve iniciar desligado e somente começar a contagem quando o usuário ordenar. O mesmo deve ocorrer com seu desligamento. Daí a necessidade do uso de duas mensagens. Uma para dar partida ao *timer* e outra para encerrá-lo. Essas mensagens são enviadas pelo usuário, no momento em que "clica" no botão do vídeo estereoscópico.

Com a Figura 18 o processo de formação adotado para o vídeo estereoscópico é mais bem observado.

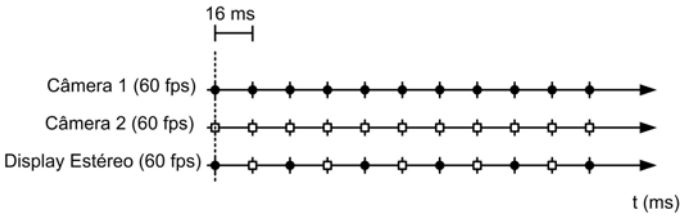

Figura 18 – Formação do vídeo estereoscópico a 60 fps.

Suponha que os quadros da Câmera 1 sejam representados pelas bolinhas preenchidas e os da Câmera 2 sejam representados pelos quadrados. Cada câmera disponibiliza quadros a 60 fps, o que significa um intervalo de (aproximadamente) 16 ms de um *frame* para o outro. A metodologia adotada para a construção do vídeo estereoscópico a 60 fps faz com que o selecionador de *frames* escolha um quadro de cada câmera a cada 16 ms. Assim, a linha "Display Estéreo" da Figura 18 mostra a origem dos quadros do vídeo estereoscópico no tempo. Durante a amostragem dos quadros, o sinal de sincronismo é enviado através da porta paralela; pino 3 (dados = 0x02) para a câmera esquerda, pino 2 (dados = 0x01) para a câmera direita. Um sinal que limpa os bits da porta paralela é enviado antes de cada sinal de sincronismo.

Essa abordagem, todavia, não é eficaz pois assume que a taxa de atualização de quadros das câmeras seja constante. Porém, como foi dito, isso não poder ser garantido, e basta que uma das câmeras tenha problemas com a taxa de atualização de seus quadros para que o TFrameSelector selecione quadros antigos (ver figura abaixo).

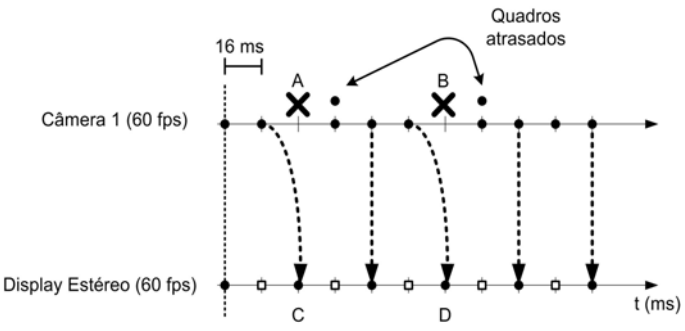

Figura 19 – Quadros atrasados provocando erros no vídeo estereoscópico.

Na Figura 19, os quadros A e B não são fornecidos no momento correto; eles só passam a ficar disponíveis junto com o próximo quadro. Portanto, os quadros C e D do vídeo estereoscópico são na verdade imagens antigas, que causam erros na visualização do vídeo estereoscópico.

Cada câmera fornece vídeo a 60 fps e, portanto, alternativamente, é possível obter-se a taxa de 120 fps. No entanto, mesmo supondo que a taxa de atualização dos frames seja constante ainda haveria a necessidade de sincronizar as câmeras com uma diferença de um *frame*. Se elas não puderem atender esta exigência, esse modo de obtenção dos vídeos conterá erros. Seria necessária a sincronização das câmeras, armazenando-se os quadros da câmera a atrasar num outro *buffer*, o que faria com que se perdesse o tempo real no *Viewfinder* dessa câmera. A Figura 20 e os comentários a seguir explicam melhor esta situação.

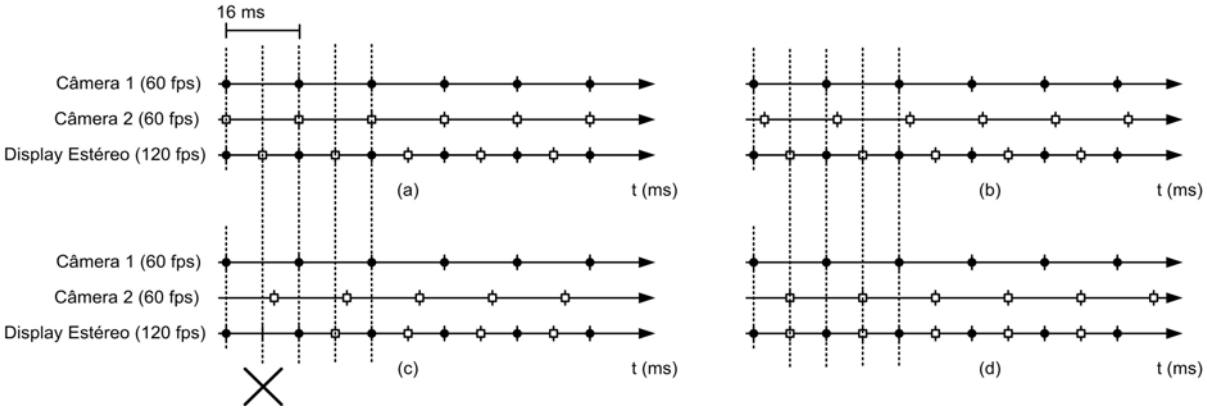

Figura 20 – Em (a) as câmeras estão sincronizadas, em (b) uma está atrasada de menos de um *frame*, em (c) o atraso é de mais de um *frame*, em (d) atraso de exatamente um *frame*.

Observando a Figura 20 (a), nota-se que as câmeras estão perfeitamente sincronizadas. Neste caso, o TFrameSelector faz a amostragem da Câmera 2 erradamente, com diferença de um quadro (~16 ms) e o vídeo estereoscópico fica com quadros antigos. A sincronização da Câmera é necessária. No caso da Figura 20 (b), a Câmera 2 está atrasada de menos de "meio *frame*", mas o atraso é menor que na Figura 20 (a). Apesar do atraso menor, ainda assim é necessária uma sincronização entre as duas câmeras. Já a Figura 20 (c), representa o pior dos casos: um atraso de mais de um frame. Neste caso, o primeiro quadro da Câmera 2 sequer existe no momento em que TFrameSelector o escolhe para o vídeo estereoscópico. Ou seja, na Figura 20 (c), o vídeo ficaria atrasado de "um *frame* e alguns milissegundos"; o maior atraso de todos. A Figura 20 (d) representa o caso ideal. Duas câmeras

atrasadas entre si de um *frame* fazem com que seja eliminada a necessidade de sincronização. O vídeo estereoscópico coleta amostras a cada 8 ms, o que indica que não há trabalho para sincronizar as câmeras.

5.7 Etapa 7: Captura das Fotos (Meta II)

A chamada "captura das fotos" engloba o disparo das fotos e a captura dos dados, com a menor diferença possível de tempo entre as duas câmeras. Mais uma vez, é preciso usar *threads* para se obter paralelismo. Neste caso, no entanto, as *threads* têm um papel diferente, pois nesta etapa há mais do que uma necessidade de paralelismo: é preciso também que haja simultaneidade razoável entre os disparos, já que essa é uma das exigências para fotos estéreo. Em vez de duas *threads* como na abordagem anterior (obtenção dos *Viewfinders*), aqui são usadas quatro. Duas duplas para ser mais correto. A primeira dupla cuida dos disparos simultâneos das fotos e a segunda cuida da captura dos dados de ambas as câmeras, depois de tiradas as fotos.

Dentro das duas primeiras *threads*, usa-se uma função do SDK, que envia o comando de disparo para a câmera. A captura é feita por outra função do *kit*, que recebe como parâmetro uma função de *callback*, (por exemplo, Pic_CB_Function()) e devolve o endereço e o tamanho de um *buffer* onde armazena a foto por partes. A cada vez que a função de *callback* é chamada ela recebe um pedaço da foto. E a cada vez que a função de *callback* retorna, a câmera a chama de novo, até que toda a foto tenha sido enviada para o *buffer*.

5.7.1 Disparos Simultâneos

Algumas tentativas de se disparar duas fotos ao mesmo tempo são demonstradas a seguir, classificadas quanto à ordem de grandeza do intervalo de disparo.

Metodologia 1:

A primeira opção é dar os comandos de disparo em seqüência e registrar o intervalo entre os disparos. Para medir este intervalo, é marcada a diferença de tempos entre a primeira e a segunda

chamadas à função de disparo como no pseudo-código abaixo:

```
void OnClick()
{
        // Começa a marcação do tempo
        ComecaMarcacaoDeTempo

        // Dispara fotos em seqüência
        DisparaFoto(Esquerda)
        DisparaFoto(Direita)

        // Termina a marcação do tempo
        TerminaMarcacaoDeTempo
}
```

Fazendo um teste de medição do intervalo para a Metodologia 1 (Apêndice C), nota-se que o resultado encontrado não é satisfatório, já que a diferença mínima entre os disparos é da ordem de 5 s.

Metodologia 2:

Uma proposta para solucionar isso é então criar a classe TSynchronizer, que contém uma variável com função semelhante à de sincronizadores como o *mutex*. A idéia básica desta classe é implementar o pseudo-código abaixo:

```
void OnClick()
{
    // Disparar Fotos simultaneamente
    CriaThread ( ThreadDisparaEsquerda )
    CriaThread ( ThreadDisparaDireita )
}
void ThreadDisparaEsquerda()
{
    CamEsquerda.Pronta = true
    if ( CamDireita.Pronta == false )
    {
        SuspendeThread( ThreadDisparaEsquerda )
    }
    else
    {
        CamEsquerda.Pronta = false
        ResumeThread( ThreadDisparaDireita )
    }

    DisparaFoto(CamEsquerda)
}
void ThreadDisparaDireita()
{
    CamDireita.Pronta = true
    if ( CamEsquerda.Pronta == false )
    {
        SuspendeThread( ThreadDisparaDireita )
    }
    else
    {
        CamDireita.Pronta = false
        ResumeThread( ThreadDisparaEsquerda )
    }

    DisparaFoto(CamDireita)
}
```

A medição do intervalo entre disparos é feita dentro das *threads* que disparam as fotos. Assim, quando a primeira *thread* dispara, inicia-se a contagem de tempo e, quando a segunda *thread* dispara a foto, a contagem de tempo é interrompida. Importante ressaltar que a ordem dos disparos não importa, e nem deveria, já que seria imprevisível. Um teste sobre a Metodologia 2 apresenta resultados como os do Apêndice D, onde se vê que um intervalo nulo entre os disparos ocorre em alguns casos, sendo que, na média, este tempo fica em 47 ms.

5.7.2 Captura dos Dados

Para obter-se os dados das fotos, faz-se necessária a outra dupla de *threads*: a de captura. O processo de captura só é iniciado quando a função PR_RC_GetReleasedData() é chamada, invocando uma função de *callback* como, por exemplo, Pic_CB_Function(). Essa função recebe os dados de uma foto aos poucos, em blocos de tamanho especificado pelo desenvolvedor. E como a câmera tem controle sobre o número desses blocos, ela passa também uma variável indicando o progresso da captura. A função Pic_CB_Function() recebe mensagens da câmera, indicando o progresso da transmissão de dados: o início da foto, a chegada de dados, o fim da transmissão ou um erro na transmissão. Mais uma vez está caracterizada a troca de mensagens entre processos estudada no início deste projeto.

Quando cada *thread* termina o seu *download*, ela envia uma mensagem que é recebida e processada pela janela indicadora do progresso da transmissão. Assim, quando os dois *downloads* acabam, a janela se fecha sozinha, retornando ao estado de espera de comandos (ver Figura 17). Durante o *download* das fotos, caso os *Viewfinders* estejam ativos, eles são interrompidos e só voltam a funcionar depois que toda a transmissão tiver terminado e as fotos tiverem sido armazenadas em arquivos. Essa, aliás, é a forma como são tratadas as fotos: elas são salvas em arquivos com uma nomenclatura peculiar. Os nomes dos arquivos seguem a regra abaixo:

AAAAMMDD.HHMMSS.O.jpg, onde:

- AAAAMMDD = 4 dígitos para o ano, 2 para o mês e 2 para o dia do mês;
- HHMMSS = 2 dígitos para a hora (0-24h), 2 dígitos para os minutos e 2 para os segundos;
- O = origem das fotos: R para direita, L para esquerda

Uma parte importante do projeto é disponibilizar os arquivos das fotos sem se importar com as dimensões delas. Isso não é um problema já que a câmera tem diferentes modos de configuração de qualidade e dimensões de foto, tratados na etapa seguinte.

5.8 Etapa 8: Configuração das Câmeras (Meta III)

O manual do SDK indica a possibilidade de configuração de sessenta e duas propriedades da

câmera. Muitas, no entanto, fogem ao escopo deste projeto, como por exemplo, a prPTP_DEV_PROP_BEEP que configura a emissão de um som ao disparar uma foto. Portanto, apenas as propriedades de configuração de tamanho e qualidade das fotos foram incluídas neste projeto. O auto-ajuste de foco, a exposição da lente e o balanço de brancos também foram inseridos no escopo, por se tratar de funções úteis à visualização dos *Viewfinders* em tempo real.

A possibilidade de configurar o tamanho das fotos é importante por causa do tempo na transmissão de dados. Quanto menores as fotos forem, menor será o tempo de transmissão. Por outro lado, elas devem ter a maior resolução possível para servir ao SVE. Pensando nisso, as configurações citadas são disponibilizadas na interface gráfica do módulo, na forma de listas, exceto pela configuração de ajustes automáticos, que fica sob a forma de botões individuais para cada câmera. As propriedades que podem ser obtidas das câmeras são tomadas de forma que as propriedades da câmera esquerda tornam-se o padrão inicial para as duas. Assim, imagine que a câmera esquerda esteja ajustada para qualidade "*Fine*" e a da direita esteja ajustada para a qualidade "*Normal*". Ao ler as propriedades da câmera esquerda, a câmera da direita também é ajustada para essa configuração.

Existem seis opções de configuração de dimensões das fotos (grande, médio 1, médio 2, médio 3, pequeno e *widescreen*) quando esta configuração é feita diretamente na câmera. No entanto, o SDK não relata tantas opções. Apenas cinco códigos estão disponíveis no manual e, portanto, apenas as mesmas cinco opções são dadas no módulo de captura desenvolvido neste projeto. Já a configuração de qualidade das fotos conta com seis opções no manual do *kit*. Na câmera, no entanto, as opções são apenas três. Somente as três disponíveis na câmera são usadas no aplicativo. O auto-ajuste das câmeras é controlado apenas pela função PR_RC_DoAeAfAwb(), que recebe como parâmetro um valor numérico indicando qual(is) dos três ajustes deve(m) ser feito(s) de uma só vez. A combinação desses valores fornece diversas alternativas para essa propriedade.

Capítulo 6 – Resultados

Este capítulo apresenta os resultados do projeto de forma sucinta. Dentre eles, a interface gráfica do programa mostrada na Figura 21 representa o do objetivo global.

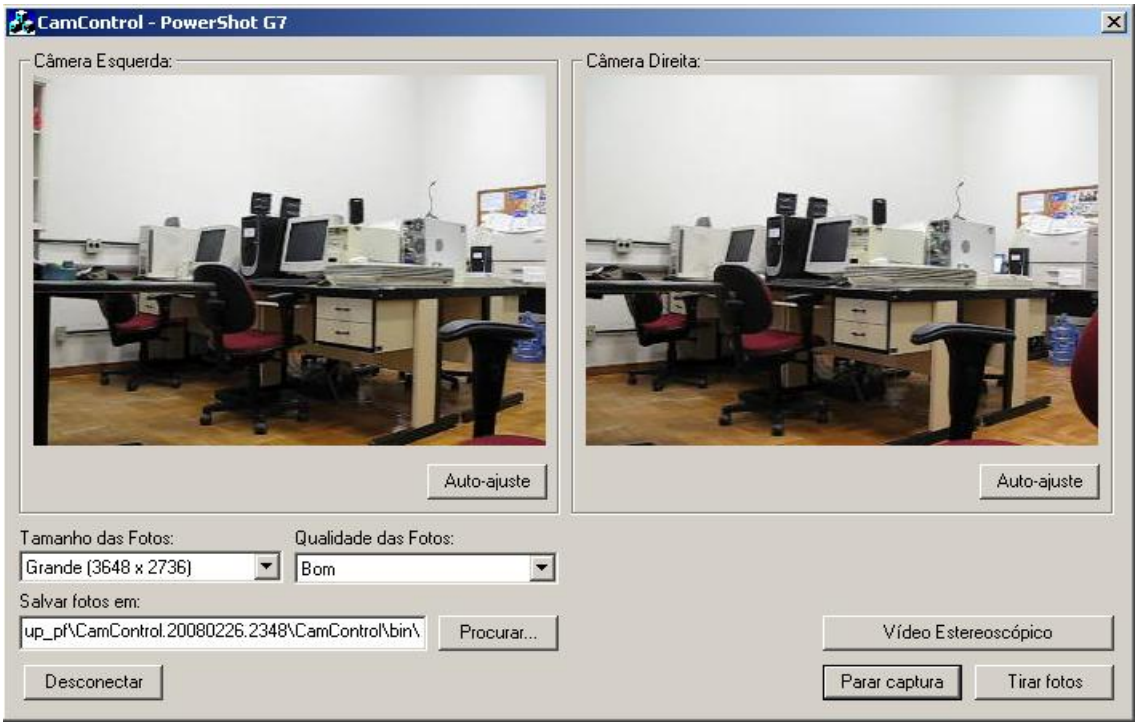

Figura 21 – Tela Principal do Módulo de Controle Remoto e Captura de Imagens.

O módulo de captura e controle desenvolvido tornou possível gerenciar o par de câmeras remotamente, fazendo com que se pudesse observar o cenário filmado em tempo real, disparar fotografias à distância com uma diferença de tempo média de 47 ms (a diferença mínima observada foi 0 s e a máxima 63 ms), e também fazer ajustes de configuração das câmeras. O vídeo estereoscópico, contudo, não foi integralmente observado. Apenas resultados parciais a uma taxa de 60 fps foram mostrados, e as deficiências encontradas estão certamente relacionadas à forma como o vídeo foi obtido.

Capítulo 7 – Conclusão

Se uma das funções do cérebro é processar as imagens estereoscópicas fornecidas pelos olhos e criar a sensação de profundidade, então se pode fazer uma analogia entre o corpo humano e este projeto. Os olhos humanos poderiam ser comparados às câmeras e partes do córtex visual do cérebro ao sistema de visão estéreo, pois, assim como os olhos, as câmeras fornecem – a uma outra entidade – imagens ligeiramente distintas de um mesmo cenário. O cérebro recebe estas imagens e as processa, assim como o sistema de visão estéreo.

O módulo de controle e captura de imagens resultou da sinergia entre áreas de pesquisa heterogêneas, mas que mostraram poder ser fortemente interligadas. Conseqüentemente, será possível, agora, obter fotos remotamente que irão servir de entrada para o SVE (ou o VEM).

O presente projeto cumpre seu papel e documenta a seguir alguns aperfeiçoamentos a serem feitos. Dentre as propostas de aperfeiçoamento, estão a utilização de equipamento adicional e a utilização de técnicas simples para a correção de deficiências conhecidas. A Tabela 5 lista essas propostas e em seguida são feitos comentários sobre essas propostas.

Tabela 5 – Sugestões de aperfeiçoamento deste projeto.

#	Proposta	Aplica-se à
1	Utilização de óculos e tela especiais para observação da imagem em três dimensões	*Hardware* adicional
2	Criação de um botão "Trocar câmeras" que alterne as posições das câmeras e tratamento de erro	Aplicativo
3	Girar uma das câmeras para reduzir o espaçamento entre as lentes a cerca de 65 mm	Câmeras
4	Ajustes finos de foco e zoom	Aplicativo
5	Testar estereoscopia com placa de vídeo especial	*Hardware* adicional
6	Desenvolver biblioteca independente da câmera	Aplicativo

1. Pode-se fazer uso das técnicas apresentadas no Capítulo 2 para observar as imagens

estereoscópicas em três dimensões, ao vivo. Uma das técnicas inclui o uso de óculos especiais e uma tela polarizadora que juntos filtram imagens capturadas.

2. O protocolo atual de conexão das câmeras exige que antes de prendê-las no suporte, seja feito um teste para identificar qual é a câmera esquerda e qual é a direita. Se, por engano, as câmeras forem trocadas de posição no suporte, suas imagens não serão apresentadas como o esperado. Isto é, a câmera presa à esquerda no suporte terá suas imagens exibidas à direita no aplicativo.

Para corrigir isso, a sugestão dada aqui é que seja adicionado ao aplicativo um botão entitulado “Trocar câmeras” que faça a troca de referências das câmeras. Há também um tratamento de erro a ser feito, para o caso em que uma das câmeras se desliga (por exemplo, por falta de energia na bateria).

3. A distância entre as lentes das câmeras deveria ser de 65 mm e, no entanto, devido às dimensões das próprias câmeras e da posição dos conectores mini-USB (ao lado direito das câmeras) essa distância precisou se restringir a valores acima de 140 mm, o que é mais que o dobro desejado para estereoscopia. Entretanto, para a estereofotogrametria, uma distância maior entre as câmeras é desejável. O excesso de distância entre as câmeras poderia levar à necessidade de processamento adicional para suprimir a diferença na medida de profundidade.

É proposto aqui que uma das câmeras seja girada de 180 graus no eixo Z, como mostrado na Figura 22, para eliminar a necessidade de processamento adicional do sistema de visão estéreo. Com isso, um novo suporte teria que ser feito, ajustando-se às novas necessidades de posicionamento das câmeras. Mesmo com a confecção de um novo suporte, ou a adaptação do atual, a demanda por tempo e esforço seria muito menor do que no caso de ter que se fazer com que o sistema de visão estéreo se adapte ao excesso de espaçamento entre as lentes.

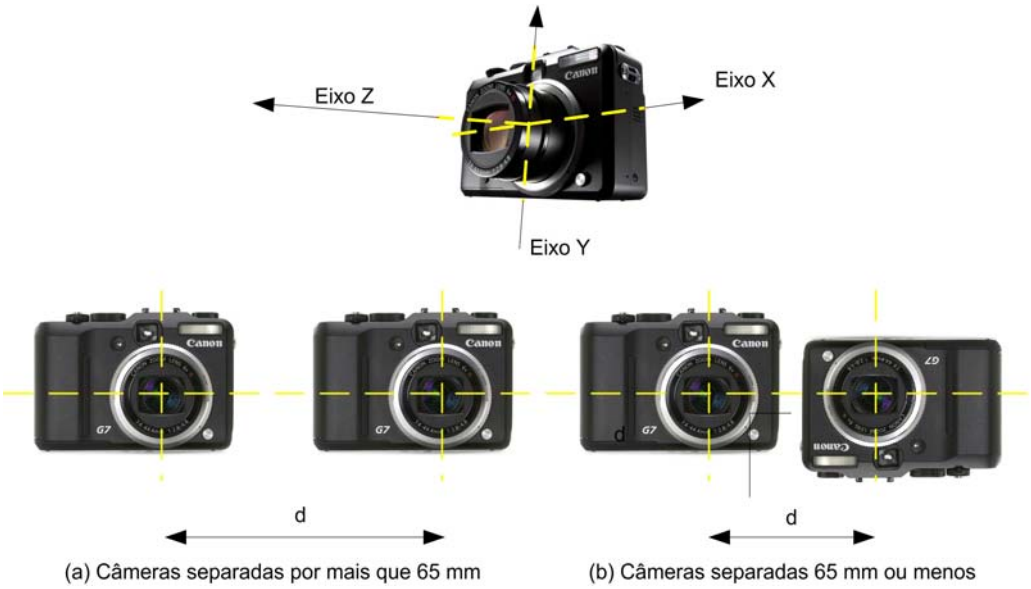

Figura 22 – (a) Distância entre lentes maior que 65 mm, (b) distância menor que 65 mm.

4. Ajustes de foco estão disponíveis através do botão de auto-ajuste, mas se forem necessários ajustes mais precisos eles ainda não podem ser feitos com a interface desenvolvida. O ajuste de zoom também não está disponível na interface. Essas duas idéias podem ser implementadas através de funções do SDK que controlam essas propriedades da câmera.
5. A estereoscopia foi alcançada parcialmente e algumas razões para esse problema foram apontadas. Há aqui mais uma possível razão para este problema: o uso de placas de vídeo diferentes da usada com o SVE. Naquele programa a placa usada é a Realism 100, da 3DLabs. A proposta neste caso é tentar utilizar a mesma placa de vídeo para se adquirir a estereoscopia.
6. A sexta proposta dada nesta seção é a de implementar uma biblioteca semelhante ao SDK mas que seja independente do modelo de câmera a se usar.

Referências Bibliográficas

[1] G. Haubrich, "Sistema de Captura e Processamento de Imagens Estéreo", Tese de Mestrado, COPPE/UFRJ, 2004.

[2] PADS/UFRJ, "Manual do Usuário - Sistema de Visão Estereoscópica", Rio de Janeiro,

[3] J. A. Júnior, "Visão Estéreo Multiplataforma", Dissertação de Projeto Final (em conclusão), UFRJ/DEL, 2008.

[4] C. Kirner e R. Tori (eds.), "Realidade Virtual: Conceitos e Tendências – Livro do Pré-Simpósio SVR 2004", Cap. 11, p. 179-201. Ed. Mania de Livro, São Paulo, 2004.

[5] L. C. Silva, "Método Robusto para a Calibração de Câmeras em Estereofotogrametria", Tese de Doutorado, COPPE/UFRJ, 2003.

[6] A. S. Tanenbaum, "Modern Operating Systems", 2ª Edição, 2001.

[7] Canon Inc., "*PowerShot RemoteCapture Software Development Kit – Software Developer's Guide*", EUA, 2006.

[8] Canon Inc., "Technical Hall - Technical report 2006.12", http://www.canon.com/camera-museum/tech/report/200612/200612.html em 12 de fevereiro de 2008.

[9] Microsoft Corporation, "*MSDN Library - Callback* functions", http://msdn2.microsoft.com/en-us/library/d186xcf0.aspx em 28 de fevereiro de 2008.

[10] Microsoft Corporation, "*MSDN Library Visual Studio 6.0a*", EUA, sem ano.

Apêndice A

Especificações Gerais da Câmera Canon PowerShot G7

Tabela 6 – Tabela de especificações gerais da câmera Canon PowerShot G7.

	PowerShot G7
Sensor de Imagem (CCD)	Pixels efetivos: aproximadamente 10 milhões, 1/1.8" (Total de pixels: aproximadamente 10.4 milhões)
Filtro de Cor	Filtro de Cor Primário
Lentes (equivalente a filme de 35 mm)	35 – 210 mm, f/2,8 (Perto) – f/4(Longe)
Zoom Digital	Aproximadamente 4.0x (aproximadamente 24.0x com zoom ótico)
Monitor LCD	2,5" LCD colorido de silício policristalino de baixa temperatura com aproximadamente 207 mil *pixels* e largo ângulo de visão
Sistema de auto-foco	TTL, AIAF (Prioridade a faces/9 pontos) / AF (Centro 1 ponto)
Distancia de Disparo (Normal)	50 cm – infinito
Distancia de Disparo (Macro)	1 – 50 cm (Perto)
Viewfinder	Viewfinder com imagem real
Tipo de Obturador	Obturador mecânico + Obturador eletrônico
Velocidade do Obturador	15 – 1/2500 s
Método de Medição da Luz	Avaliador/ "Média Centrada"
Compensação de Exposição	+- 2,0 paradas (incrementos de 1/3)
Sensibilidade	Automático, alta sensibilidade, ISSO 80/100/200/400/800/1600
Alcance do Flash	50 cm – 4.0 m (Perto), 50 cm – 2.5 m (Longe) (quando sensibilidade esta ajustada para automática)
Balanço de Brancos	Automático, Luz do dia, Nebuloso, Tungstênio, Fluorescente H, Flash, Submerso ou Customizado

Modos de Disparo	Automático, P, TV, Av, M, C1, C2, Retrato, Paisagem, Cena Noturna, Esporte, Snapshot Noturno, Crianças e Animais, Ambiente Interno, Neve, Praia, Fogos de Artifício, Aquarium, Visão geral da noite Filmagem Subaquática, ISO 3200, Acentuação de Cor, Troca de Cor, Assistente Stitch, Filme
Disparo Continuo	Disparo Normal: Aproximadamente 2,0 disparos/s (Large/Fine) Disparos AF Contínuos: Aproximadamente 0.8 disparos/s (Large/Fine)
Formato de gravação de Imagens Estáticas	18 combinações: (L / M1 / M2 / M3 / S / W) x (SF / F / N)
Numero de Pixels na Imagem Estática	L: 3648 x 2736, M1: 2816 x 2112, M2: 2272 x 1704, M3: 1600 x 1200, S: 640 x 480, W: 3648 x 2048
Filme	L: 1024 x 768 (15 fps) Padrão / Acentuação da Cor/ Troca de Cores: 640 x 480, 320 x 240 pixels (30 fps, 15 fps) Cor Swap: 640 x 480, 320 x 240 *pixels* (30 fps, 15 fps) *Fast Frame Rate*: 320 x 240 pixels (60 fps), Compacto: 160 x 120 pixels (15 fps) *Fast Frame Rate*: 320 x 240 pixels (60 fps), Compacto: 160 x 120 pixels (15 fps)
Formato de arquivo	Compatível com DCF (Exif 2.2) e DPOF
Fornecimento de energia	Bateria lithium-ion bateria: NB-2LH, *kit* adaptador AC: ACK-DC20 (opcional)
Dimensões (L x A x P)	106,4 x 71,9 x 42,5 milímetros
Peso (apenas corpo da câmera)	Aproximadamente 320 g

Apêndice B

Resultados de Testes Sobre a Taxa de Atualização dos Viewfinders

Tabela 7 – Taxa de vídeo dos Viewfinders. Cada célula na tabela representa um quadro recebido.

handle=9248568 @ t=5.7340000000	handle=9248568 @ t=6.4210000000	handle=9248696 @ t=6.8590000000
handle=9248568 @ t=5.7650000000	handle=9248696 @ t=6.4370000000	handle=9248568 @ t=6.8590000000
handle=9248568 @ t=5.7650000000	handle=9248568 @ t=6.4370000000	handle=9248696 @ t=6.8590000000
handle=9248568 @ t=5.7960000000	handle=9248696 @ t=6.4530000000	handle=9248568 @ t=6.8750000000
handle=9248568 @ t=5.7960000000	handle=9248568 @ t=6.4530000000	handle=9248568 @ t=6.8900000000
handle=9248568 @ t=5.8280000000	handle=9248696 @ t=6.4680000000	handle=9248696 @ t=6.9060000000
handle=9248568 @ t=5.8430000000	handle=9248568 @ t=6.4680000000	handle=9248568 @ t=6.9060000000
handle=9248568 @ t=5.8590000000	handle=9248696 @ t=6.4840000000	handle=9248696 @ t=6.9210000000
handle=9248568 @ t=5.8750000000	handle=9248568 @ t=6.4840000000	handle=9248568 @ t=6.9210000000
handle=9248568 @ t=5.8900000000	handle=9248696 @ t=6.5000000000	handle=9248696 @ t=6.9370000000
handle=9248568 @ t=5.9060000000	handle=9248568 @ t=6.5000000000	handle=9248568 @ t=6.9370000000
handle=9248568 @ t=5.9210000000	handle=9248696 @ t=6.5150000000	handle=9248696 @ t=6.9530000000
handle=9248568 @ t=5.9370000000	handle=9248568 @ t=6.5310000000	handle=9248568 @ t=6.9530000000
handle=9248568 @ t=5.9530000000	handle=9248696 @ t=6.5310000000	handle=9248696 @ t=6.9680000000
handle=9248568 @ t=5.9680000000	handle=9248568 @ t=6.5460000000	handle=9248568 @ t=6.9680000000
handle=9248568 @ t=5.9840000000	handle=9248696 @ t=6.5460000000	handle=9248696 @ t=6.9840000000
handle=9248568 @ t=6.0000000000	handle=9248568 @ t=6.5620000000	handle=9248568 @ t=7.0000000000
handle=9248568 @ t=6.0150000000	handle=9248696 @ t=6.5620000000	handle=9248696 @ t=7.0000000000
handle=9248568 @ t=6.0310000000	handle=9248568 @ t=6.5780000000	handle=9248568 @ t=7.0150000000
handle=9248568 @ t=6.0460000000	handle=9248696 @ t=6.5930000000	handle=9248568 @ t=7.0310000000
handle=9248568 @ t=6.0620000000	handle=9248568 @ t=6.5930000000	handle=9248696 @ t=7.0310000000
handle=9248568 @ t=6.0780000000	handle=9248696 @ t=6.6090000000	handle=9248568 @ t=7.0460000000
handle=9248568 @ t=6.0930000000	handle=9248568 @ t=6.6090000000	handle=9248696 @ t=7.0460000000
handle=9248568 @ t=6.1090000000	handle=9248696 @ t=6.6250000000	handle=9248568 @ t=7.0620000000
handle=9248568 @ t=6.1250000000	handle=9248568 @ t=6.6250000000	handle=9248696 @ t=7.0620000000
handle=9248568 @ t=6.1400000000	handle=9248696 @ t=6.6400000000	handle=9248568 @ t=7.0780000000
handle=9248568 @ t=6.1560000000	handle=9248568 @ t=6.6400000000	handle=9248696 @ t=7.0780000000
handle=9248568 @ t=6.1710000000	handle=9248568 @ t=6.6560000000	handle=9248568 @ t=7.0930000000
handle=9248568 @ t=6.1870000000	handle=9248696 @ t=6.6710000000	handle=9248696 @ t=7.0930000000

handle=9248568 @ t=6.2030000000	handle=9248568 @ t=6.6870000000	handle=9248568 @ t=7.1090000000
handle=9248568 @ t=6.2180000000	handle=9248696 @ t=6.6870000000	handle=9248696 @ t=7.1250000000
handle=9248568 @ t=6.2340000000	handle=9248568 @ t=6.7030000000	handle=9248568 @ t=7.1250000000
handle=9248568 @ t=6.2500000000	handle=9248696 @ t=6.7030000000	handle=9248696 @ t=7.1250000000
handle=9248568 @ t=6.2650000000	handle=9248568 @ t=6.7180000000	handle=9248568 @ t=7.1400000000
handle=9248568 @ t=6.2810000000	handle=9248696 @ t=6.7180000000	handle=9248568 @ t=7.1560000000
handle=9248568 @ t=6.2960000000	handle=9248568 @ t=6.7340000000	handle=9248696 @ t=7.1710000000
handle=9248568 @ t=6.3120000000	handle=9248696 @ t=6.7340000000	handle=9248696 @ t=7.1870000000
handle=9248568 @ t=6.3280000000	handle=9248568 @ t=6.7500000000	handle=9248696 @ t=7.2030000000
handle=9248568 @ t=6.3430000000	handle=9248696 @ t=6.7650000000	handle=9248568 @ t=7.2030000000
handle=9248568 @ t=6.3590000000	handle=9248568 @ t=6.7650000000	handle=9248696 @ t=7.2180000000
handle=9248696 @ t=6.3750000000	handle=9248696 @ t=6.7810000000	handle=9248568 @ t=7.2340000000
handle=9248568 @ t=6.3750000000	handle=9248568 @ t=6.7960000000	handle=9248696 @ t=7.2340000000
handle=9248696 @ t=6.3900000000	handle=9248696 @ t=6.7960000000	handle=9248568 @ t=7.2340000000
handle=9248568 @ t=6.3900000000	handle=9248568 @ t=6.8120000000	handle=9248696 @ t=7.2500000000
handle=9248696 @ t=6.4060000000	handle=9248568 @ t=6.8280000000	handle=9248568 @ t=7.2650000000
handle=9248568 @ t=6.4060000000	handle=9248696 @ t=6.8280000000	handle=9248696 @ t=7.2650000000
handle=9248696 @ t=6.4210000000	handle=9248568 @ t=6.8430000000	handle=9248568 @ t=7.2810000000

Apêndice C

Resultados de Testes Sobre Atraso no Disparo entre Fotos: Metodologia 1.

Tabela 8 – Atraso no disparo entre fotos: Metodologia 1. Cada linha corresponde a um par de fotos que foi tirado.

Picture Set 1. Delay in seconds: 5.81200
Picture Set 2. Delay in seconds: 5.32800
Picture Set 3. Delay in seconds: 5.34400
Picture Set 4. Delay in seconds: 5.75000
Picture Set 5. Delay in seconds: 5.40600
Picture Set 6. Delay in seconds: 5.40700
Picture Set 7. Delay in seconds: 5.62500
Picture Set 8. Delay in seconds: 5.45300
Picture Set 9. Delay in seconds: 5.54600
Picture Set 10. Delay in seconds: 5.48400
Picture Set 11. Delay in seconds: 5.49600

Apêndice D

Resultados de Testes Sobre Atraso no Disparo entre Fotos: Metodologia 2.

Tabela 9 – Atraso no disparo entre fotos: metodologia 2. Cada linha representa um par de fotos.

Delay in seconds: 0.06300
Delay in seconds: 0.06300
Delay in seconds: 0.04600
Delay in seconds: 0.04700
Delay in seconds: 0.04700
Delay in seconds: 0.04700
Delay in seconds: 0.04700
Delay in seconds: 0.06300
Delay in seconds: 0.00000
Delay in seconds: 0.00000
Delay in seconds: 0.04700
Delay in seconds: 0.06200
Delay in seconds: 0.04700
Delay in seconds: 0.04700
Delay in seconds: 0.04700
Delay in seconds: 0.04700
Delay in seconds: 0.04700
Delay in seconds: 0.04700
Delay in seconds: 0.06300
Delay in seconds: 0.04600
Delay in seconds: 0.04700
Delay in seconds: 0.06200
Delay in seconds: 0.06200
Delay in seconds: 0.04700
Delay in seconds: 0.04700
Delay in seconds: 0.04600
Delay in seconds: 0.06200