Pontifícia Universidade Católica
do Rio de Janeiro

Romano José Magacho da Silva

Integração de um Dispositivo Óptico de Rastreamento a uma Ferramenta de Realidade Virtual

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do título de Mestre pelo Programa de Pós-Graduação em Informática da PUC-Rio.

Orientador: Prof. Marcelo Gattass
Co-orientador: Prof. Alberto Barbosa Raposo

Rio de Janeiro, agosto de 2004

Pontifícia Universidade Católica
do Rio de Janeiro

Romano José Magacho da Silva

Integração de um Dispositivo Óptico de Rastreamento a uma Ferramenta de Realidade Virtual

Dissertação apresentada como requisito parcial para obtenção do título de Mestre pelo Programa de Pós-Graduação em Informática da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Marcelo Gattass
Orientador
Departamento de Informática - PUC-Rio

Alberto Barbosa Raposo
Co-orientador
Departamento de Informática - PUC-Rio

Waldemar Celes Filho
Departamento de Informática - PUC-Rio

Paulo Cezar Pinto Carvalho
Instituto Nacional de Matemática Pura e Aplicada (IMPA)

Flávio Szenberg
Tecgraf - PUC-Rio

Prof. José Eugenio Leal
Coordenador Setorial do Centro
Técnico Científico – PUC-Rio

Rio de Janeiro, 31 de março de 2004

Romano José Magacho da Silva

Graduou-se em Engenharia de Computação no Instituto Militar de Engenharia (IME/RJ) em 1998. Trabalhou no Instituto de Pesquisa e Desenvolvimento do Exército no projeto de Comando, Controle e Informação, no desenvolvimento de ferramentas de informação geográfica. Nos anos 2000 e 2001, atuou como desenvolvedor da aplicação de Comando Tático de Manobra do Sistema de Artilharia do Exército na Indústria de Material Bélico do Brasil.

Ficha Catalográfica

Silva, Romano José Magacho da

Integração de um dispositivo óptico de rastreamento a uma ferramenta de realidade virtual / Romano José Magacho da Silva; orientador: Marcelo Gattass; co-orientador: Alberto Barbosa Raposo. – Rio de Janeiro: PUC-Rio, Departamento de Informática, 2004.

76 f. : il. ; 30 cm

Dissertação (mestrado) – Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui referências bibliográficas

1. Informática – Teses. 2. Realidade virtual. 3. Visão computacional. 4. Dispositivo de rastreamento. I. Gattass, Marcelo. II. Raposo, Alberto Barbosa. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. IV. Título.

CDD: 004

“Memento, homo, quia pulvis es, et in púlverem revertéris”. Gn 3,19.

A Nosso Senhor Jesus Cristo, Rei dos reis e
a Nossa Senhora, medianeira de todas as graças.

Agradecimentos

A Deus, nosso Senhor Jesus Cristo, toda honra e toda glória.

A minha família, minha esposa, meus filhos, meus pais, meus irmãos e meus amigos pelo apoio, compreensão e paciência.

Ao orientador e professor Marcelo Gattass pelo conhecimento e apoio durante o curso.

Ao meu co-orientador, professor Alberto Barbosa Raposo, pelo apoio e pela amizade conquistada.

Aos dois, muito obrigado pela oportunidade de trabalhar e aprender em um tão conceituado grupo como o TeCGraf.

Aos colegas do TeCGraf que de algum modo ajudaram na realização deste trabalho, Felipe, Gustavo, Thiago, Márcio Henrique, Ricardo, Pedro Ponce, Eduardo Thadeu, Eduardo Teles, Manuel, Gabriel, Flávio Szenberg, Felipe Lobo, Pablo, Fred, César Pozzer, prof. Waldemar e a tantos outros que merecem muito mais do que um simples obrigado.

Ao colega de turma Ênio Russo que transmitiu uma invejável sabedoria e paciência, me ajudando a acalmar nos momentos de turbulência.

Ao Exército Brasileiro e ao Instituto Militar de Engenharia.

Resumo

Silva, Romano J M.; Gattass, Marcelo (Orientador); Raposo, Alberto B (Co-orientador). **Integração de um Dispositivo Óptico de Rastreamento a uma Ferramenta de Realidade Virtual.** Rio de Janeiro, 2004. 76p. Dissertação de Mestrado - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Os sistemas de realidade virtual requerem dispositivos de interação que não obstruam o caráter imersivo da aplicação. Com a difusão das câmeras digitais, a captura óptica do movimento do usuário se firmou como uma nova área de pesquisa. Este trabalho apresenta a integração de um dispositivo óptico para interação em uma ferramenta de desenvolvimento de aplicações em realidade virtual. O dispositivo óptico proposto é composto por uma esfera revestida de material retroreflexivo rastreada por quatro câmeras com sensores infravermelhos. O estudo engloba a implementação do dispositivo de rastreamento óptico e sua integração com a ferramenta ViRAL (Virtual Reality Abstraction Layer).

Palavras-chave

Realidade-virtual, dispositivo de rastreamento, visão computacional.

Abstract

Silva, Romano J M.; Gattass, Marcelo (Supervisor); Raposo, Alberto B (Co-supervisor). **Integrating an Optical Tracking Device with a Virtual Reality Toolkit**. Rio de Janeiro, 2004. 76p. M.Sc. Dissertation – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Virtual reality systems require tracking devices that do not harm the application immersive sensation. With the spread of digital cameras, the optical tracking of users' movement has been firmed as a new area of research. This work presents the integration of an optical tracking device with a virtual reality application development library. The proposed optical device is composed of a sphere coated with retroreflexive material, which is tracked by four cameras with infrared sensors. The study contains the implementation of the tracking device and its integration with ViRAL (Virtual Reality Abstraction Layer) tool.

Keywords

Virtual reality, tracking devices, computer vision.

Sumário

Lista de figuras

Lista de tabelas

Uma noite eu tive um sonho...

Sonhei que estava andando na praia com o Senhor, e através do céu, passavam cenas da minha vida. Para cada cena que se passava, percebi que eram deixados dois pares de pegadas na areia, um era meu e outro era do Senhor. Quando a última cena da minha vida passou diante de nós, olhei para trás, para as pegadas na areia, e notei que muitas vezes no caminho da vida havia apenas um par de pegadas na areia. Notei também que isto aconteceu nos momentos mais difíceis e angustiosos do meu viver. Isso me aborreceu, então perguntei ao Senhor:

— "Senhor, Tu me disseste que ao te seguir, Tu andarias sempre comigo, em todo o meu caminho, mas notei que durante as maiores tribulações do meu viver, havia apenas um par de pegadas na areia. Não compreendo porque nas horas em que necessitava de Ti, Tu me deixaste..."

O Senhor respondeu:

— "Meu precioso filho, Eu te amo e jamais te deixaria nas horas de tua prova e de teu sofrimento. Quando vistes na areia apenas um par de pegadas, foi exatamente aí que Eu te carreguei nos braços."

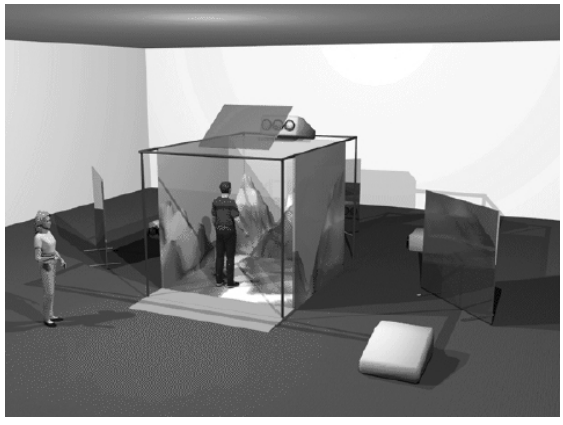
Pegadas na Areia
Autor desconhecido

1
Introdução

Realidade Virtual (RV) possui várias definições, sendo descrita em [2] como sendo uma forma das pessoas visualizarem, manipularem e interagirem com computadores e dados extremamente complexos. Através de outras definições existentes em [6,14,16] pode-se dizer que realidade virtual é uma técnica avançada de interface, onde o usuário pode navegar e interagir em um ambiente sintético tridimensional gerado por computador, estando completa ou parcialmente presente ou imerso pela sensação gerada por canais multi-sensoriais, sendo o principal a visão.

Apesar de suas várias definições, é de comum acordo que a sua essência está nos ambientes tridimensionais baseados em computadores. Geralmente chamados de "mundo virtual" ou "ambiente virtual", esses ambientes são representações do mundo real ou de situações conceituais que podem ser investigados através de navegação, interação e atualização em tempo real.

(i)



(ii)

Figura 1: Ambientes de RV: (i) um capacete de RV; (ii) uma CAVE.

Com o desenvolvimento das tecnologias de RV, diversos tipos de dispositivos de interação e de saída foram surgindo. Para interagir com o mundo virtual, o usuário pode sentar em uma cadeira, utilizando um capacete de RV (Figura 1i) e tendo em sua mão uma luva, ou ele pode estar cercado por paredes de projeção, em uma configuração denominada CAVE [7] (Figura 1ii), movendo

os objetos utilizando um *mouse* sem fio ou um dispositivo óptico. Os sistemas de RV devem ser capazes de funcionar nessas diversas configurações.

Para facilitar o desenvolvimento de aplicações multi-dispositivos, surgiram as bibliotecas de RV. Elas são capazes de fornecer dados como a posição e a orientação do dispositivo, sem que a aplicação saiba qual a sua origem, agindo como uma camada de abstração. Além dos dispositivos de entrada, a aplicação produz a saída visual sem saber se ela será apresentada em um monitor ou em uma parede com várias telas de projeção.

Existe uma categoria de dispositivos de interação que está se destacando devido a crescente disponibilidade de câmeras digitais. Este tipo de dispositivo óptico utiliza a imagem capturada por um conjunto de câmeras digitais para rastrear a posição de um ou mais objetos e, com isso, fornecer dados de posicionamento e orientação para os sistemas de RV.

Uma vantagem desse tipo de dispositivo é a mobilidade que ele provê ao usuário, tanto pela não existência de cabos, quanto pela capacidade de fornecer informações redundantes utilizando mais de uma câmera, sendo menos suscetível à oclusão de uma delas. Outra vantagem é a imunidade a interferências externas, sejam elétricas ou eletromagnéticas.

1.1. Objetivo

O objetivo principal desta dissertação é o desenvolvimento de um dispositivo óptico de rastreamento para ambientes de RV. Este dispositivo é composto por quatro câmeras com emissores e filtros infravermelhos. O objeto a ser rastreado é uma esfera coberta com material retroreflexivo.

A fim de se empregar o dispositivo em ambientes virtuais, é necessário integrá-lo a uma biblioteca de abstração. Algumas bibliotecas foram analisadas e, neste trabalho, foi escolhido o uso do ViRAL (***Vi****rtual* ***R****eality* ***A****bstraction* ***L****ayer*), uma ferramenta em desenvolvimento pelo grupo de Realidade Virtual do TecGraf/PUC-Rio.

O dispositivo óptico foi utilizado para controlar um objeto em uma aplicação, também desenvolvida utilizando o ViRAL.

1.2. Organização da Dissertação

Esta dissertação apresenta um dispositivo óptico de rastreamento, integrado a uma biblioteca de RV. A cena a ser controlada pelo dispositivo será representada por um grafo de cena [10,32] em uma aplicação executada sob a mesma biblioteca.

O Capítulo 2 traz um estudo das tecnologias de RV existentes, mostrando seu rápido crescimento e justificando a necessidade do uso de camadas de abstração, sendo duas delas analisadas: o *VRJuggler* e o *ViRAL*.

O Capítulo 3 apresenta o processamento necessário para rastreamento óptico de um objeto e a sua utilização como um dispositivo de RV. O Capítulo 4 apresenta os principais detalhes da implementação e os resultados obtidos. O Capítulo 5 conclui a tese e apresenta sugestões para trabalhos futuros.

2
Realidade Virtual

Com aplicação em grande parte das áreas do conhecimento, entre elas a medicina, mecânica, treinamento militar, ergonomia, jogos e entretenimento, e com um grande investimento das indústrias na produção de software e hardware para dispositivos de entrada e saída, a realidade virtual vem experimentando um desenvolvimento acelerado nos últimos anos e indicando perspectivas bastante promissoras para os diversos segmentos vinculados com a área.

Um dos benefícios dos ambientes de RV é a capacidade de prover perspectivas vantajosas, impossíveis de serem obtidas no mundo real, como por exemplo, a navegação por dentro do corpo humano ou a análise de simulações físicas ou reações químicas em tempo real. A interação com o ambiente virtual é feita utilizando dispositivos próprios para RV, como capacetes de visualização, luvas, *mouses* tridimensionais, sensores de posicionamento, entre outros.

As aplicações de RV se diferenciam das aplicações convencionais pelos tipos de dispositivos que utilizam e pela característica presencial ou imersiva do usuário no sistema. O usuário se sente imerso quando as sensações e eventos que acontecem no mundo virtual são sentidos por ele no mundo real. Atualmente se desenvolvem pesquisas buscando atuar sobre as percepções visuais, auditivas e tácteis.

Quanto à visão, o objetivo principal é fornecer ao usuário a sensação de profundidade, que não deve ser confundida com a noção de profundidade obtida pela distorção de perspectiva. A distorção de perspectiva permite distinguir objetos que estão próximos dos que estão mais distantes, enquanto a sensação de profundidade permite que o usuário seja imerso no ambiente virtual.

Para atingir esse objetivo, o sistema deve gerar, ao mesmo tempo, duas imagens diferentes, correspondendo às visões dos olhos esquerdo e direito. O cérebro humano processa a diferença entre essas duas imagens gerando uma representação precisa da posição e da forma tridimensional do objeto dentro da cena. Essa é a profundidade tridimensional que experimentamos no mundo real.

Esse tipo de geração de imagem é chamado de visão estereoscópica ou estereoscopia [23].

2.1. Sistemas de Realidade Virtual

Os sistemas de realidade virtual são compostos por dispositivos de entrada (rastreadores de posição, reconhecimento de voz, *joysticks* e mouses, por exemplo) e saída (visual, auditiva e táctil) capazes de prover ao usuário imersão, interação e envolvimento [18]. A idéia de imersão está ligada ao sentimento de se estar dentro do ambiente. A interação diz respeito à capacidade do computador detectar as entradas do usuário e modificar instantaneamente o mundo virtual e as ações sobre ele. O envolvimento, por sua vez, está ligado com o grau de engajamento de uma pessoa com determinada atividade, podendo ser passivo, como assistir televisão, ou ativo, ao participar de um jogo com algum parceiro. A realidade virtual tem potencial para os dois tipos de envolvimento ao permitir a exploração de um ambiente virtual e ao propiciar a interação do usuário com um mundo virtual dinâmico.

Além do hardware, os sistemas de realidade virtual necessitam do software cuja composição é geralmente formada por três componentes: a componente de apresentação, de interação e de aquisição de dados.

A componente de apresentação é responsável pela representação, envolvendo aspectos de interface homem-computador e semiótica [23], carregamento do modelo e saída do sistema.

O componente de interação é responsável pela manipulação e navegação no ambiente. A dificuldade nesse caso é que, ao contrário dos ambientes convencionais, existem poucas metáforas de interação bem definidas para ambientes imersivos.

A aquisição dos dados é a componente mais complexa do sistema de RV. O sistema pode ser criado para funcionar especificamente com um determinado dispositivo de saída (um monitor, por exemplo) e um determinado dispositivo de entrada (um mouse 3D) ou ser escalável e funcionar com qualquer composição de hardware. Nesse caso, a componente de aquisição de dados deve ser capaz de suportar a vasta gama de dispositivos, inclusive os que ainda não foram criados.

Geralmente, os sistemas escaláveis utilizam bibliotecas ou *frameworks* que permitem que o desenvolvedor abstraia a aplicação dos dispositivos utilizados.

2.2. Dispositivos de Rastreamento

Uma das conseqüências do advento da realidade virtual foi a necessidade de se redefinir o paradigma de interface homem-computador. O sistema tradicional mouse-teclado-monitor foi substituído por dispositivos que permitiram maior imersão do usuário no ambiente virtual e o manuseio de todas as potencialidades dessa nova tecnologia.

O modo como os participantes interagem com o sistema de realidade virtual influencia enormemente suas experiências no ambiente virtual, facilitando seu uso, aumentado a sensação de imersão e ampliando a variedade de ações que se pode tomar dentro do ambiente virtual.

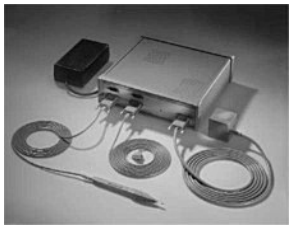
Um importante dispositivo de interação é o rastreador de posição que pode ser utilizado para acompanhar a posição do corpo e os movimentos do usuário, assim como a posição de outros objetos sendo por ele utilizados.

Existe uma variedade de dispositivos de rastreamento, cada um utilizando uma tecnologia diferente, entre eles, os eletromagnéticos, mecânicos, acústicos, inerciais e ópticos. Ao analisar as tecnologias utilizadas pelos rastreadores, três fatores devem ser levados em consideração: precisão e velocidade de resposta do sensor; interferência do meio; restrições (fios, conexões mecânicas, etc.):

- **Eletromagnéticos**
  - Princípio de funcionamento: os rastreadores eletromagnéticos utilizam campos magnéticos para medir posição e orientação. O sistema é composto por transmissor e receptor em forma de bobina. Um sensor unidimensional para estimar a posição no eixo Z, por exemplo, é composto por uma única bobina transmissora orientada na direção Z. Quando uma corrente é aplicada à bobina, um campo magnético é gerado. No receptor, o campo induz uma voltagem máxima proporcional à intensidade do campo magnético medido em uma bobina orientada na mesma direção do campo. A voltagem

induzida fornece a distância do transmissor ao receptor, assim como a diferença de alinhamento entre os eixos.

- Precisão/Velocidade: esses sistemas são bastante precisos, cerca de 1 a 2 mm para posição e 0,1º para orientação. A velocidade de captura de dados é de 100 a 200 medidas/segundo.
- Interferência do meio: a presença de metais e o próprio tubo de raios catódicos do monitor podem causar interferência eletromagnética.
- Restrições: pequeno espaço de utilização devido ao alcance do campo magnético gerado. O receptor deve estar cerca de 1-3 metros do transmissor, não havendo necessidade de linha de visada desobstruída.
- Exemplos: FasTrack da Polhemus (Figura 2i) e o Flock of Birds da Ascension (Figura 2ii).



(i)

(ii)

Figura 2: Dispositivos eletromagnéticos: (i) FasTrack e (ii) Flock of Birds.

- **Mecânicos**
  - Princípio de funcionamento: os rastreadores mecânicos medem ângulos e distância entre juntas. Dada uma posição conhecida, todas as outras podem ser determinadas pela relação entre as juntas. Os rastreadores podem estar presos ao chão ou anexos ao corpo do usuário, usualmente na forma de um exoesqueleto. As rotações e as distâncias podem ser medidas por engrenagens, potenciômetros ou sensores de dobra (Figura 3).

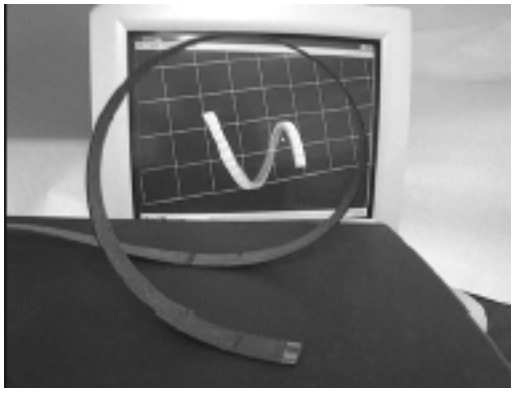
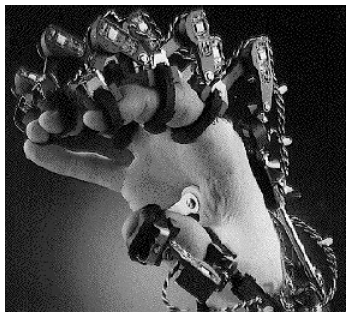


Figura 3: Sensor de dobra.

Uma vantagem dos sistemas mecânicos é a facilidade de adição da funcionalidade de *force feedback*. Esse tipo de dispositivo restringe o movimento do usuário aplicando uma força contrária ao seu movimento.

- Precisão/Velocidade: por serem mecânicos, possuem alta precisão (0,1º de rotação). A latência média é de 200 ms.
- Interferência do meio: não sofrem interferência do meio.
- Restrições: a própria arquitetura do rastreador pode restringir o movimento do usuário, caso o mesmo seja preso ao chão ou possua muitas juntas.
- Exemplos: o Phantom (Figura 4i) da Sensable Technologies e a mão mecânica da EXOS (Figura 4ii).

(i)



(ii)

Figura 4: Dispositivos mecânicos: (i) Phantom da Sensable e (ii) mão mecânica da EXOS.

- **Acústicos**
  - Princípio de funcionamento: rastreadores acústicos utilizam, tipicamente, ondas sonoras ultra-sônicas para medir distância. Os

métodos mais usados são o cálculo do tempo de vôo e a coerência de fase. Em ambos, o objetivo é converter tempo em distância.

Um único par transmissor/receptor fornece a distância do objeto em relação a um ponto fixo. O resultado é uma esfera em cuja superfície o objeto está localizado. Como visto na Figura 5, a adição de um segundo receptor restringe a região a um círculo e um terceiro receptor restringe a dois pontos, sendo um deles geralmente descartado. Portanto, para estimar a posição são necessários um transmissor e três receptores ou um receptor e três transmissores. Para estimar posição e orientação, são necessários três transmissores e três receptores.

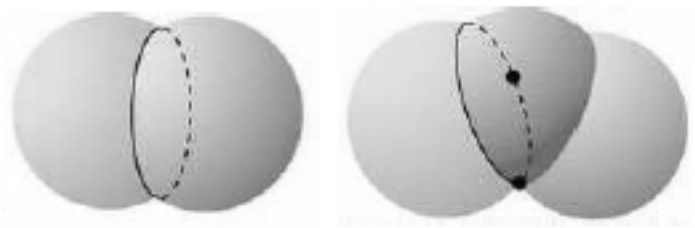


Figura 5: Interseção entre duas esferas (um círculo) e entre três (dois pontos).

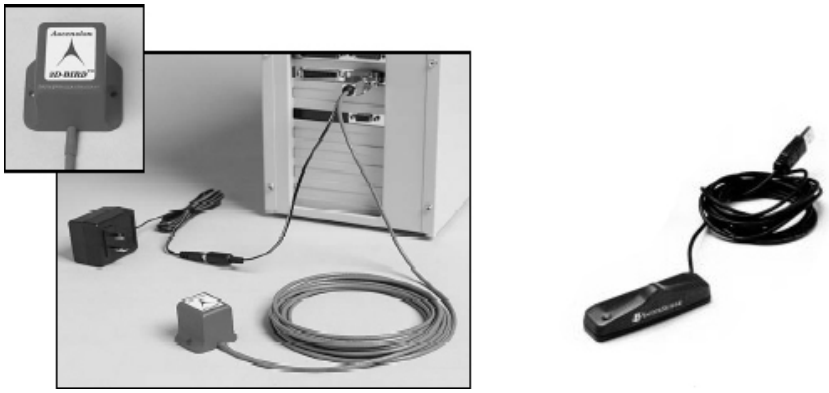
- Precisão/Velocidade: existe um atraso inerente a espera do sinal. Esse atraso é intensificado devido à baixa velocidade de propagação do som.
- Interferência do meio: as propriedades do som limitam esse método. O desempenho é degradado na presença de um ambiente ruidoso ou devido a geração de ecos. O som deve percorrer um caminho sem obstrução entre os alto-falantes e os microfones.
- Restrições: a configuração do sistema não é cara, pois o equipamento necessário é composto de microfones, alto-falantes e um computador.

  Devido às restrições de interferência, a distância média entre receptor e transmissor são alguns metros, contudo, sistemas mais precisos podem cobrir áreas de até 40x30m.
- Exemplos: Logitech Tracker (Figura 6i) com alcance médio de 15 metros e o FarReach da Infusion Systems (Figura 6ii) com alcance de 12 metros.

(i) (ii)

Figura 6: Dispositivos acústicos: (i) Logitech Tracker e (ii) FarReach.

- **Inerciais**
  - Princípio de funcionamento: utilizam magnetômetros passivos, acelerômetros e girômetros. Os magnetômetros passivos medem o campo magnético do ambiente (geralmente da Terra) e fornecem medidas angulares. Os girômetros fornecem medidas angulares mais precisas e os acelerômetros fornecem medidas lineares. Todos são baseados na segunda lei de movimento de Newton, sendo assim, o sistema deve integrar a leitura para obter a velocidade e a posição.
  - Precisão/Velocidade: Devido à etapa de integração, o erro obtido a cada passo tende a aumentar. A utilização de filtros de correção e outros sensores ajuda a diminuir esse erro.
  - Interferência do meio: Não existe interferência, pois o sistema é autocontido, não havendo necessidade de um ponto externo para obtenção de dados;
  - Restrições: Não existe limitação física para o espaço de trabalho, sendo o mesmo limitado somente pela conexão entre o dispositivo e o computador.
  - Exemplos: 3D-Bird da Ascension Technology (Figura 7i) e o Intertrax2 da InterSense (Figura 7ii).



(i) (ii)

Figura 7: Dispositivos inerciais: (i) 3D-Bird e (ii) Intertrax2.

- **Ópticos**
  - Princípio de funcionamento: baseado na análise da projeção bidimensional de uma imagem ou na determinação dos ângulos de feixes da varredura para calcular a posição e orientação de um dado objeto. Os sensores ópticos são geralmente câmeras (por exemplo, CCD), um detector 4Q ou um diodo de efeito lateral.

    Um CCD é um conjunto de detectores recebendo imagens no plano focal da câmera. Um detector 4Q é um componente plano capaz de gerar sinais especificando o centro do feixe de luz que incide em sua superfície. Um diodo de efeito lateral é um componente que gera um sinal proporcional à posição da luz chegando em um eixo.

    Quando o sensor utilizado é uma câmera, técnicas de visão computacional devem ser utilizadas para determinar a posição do objeto. Se somente uma câmera for utilizada, é possível determinar um segmento de reta que passa pelo objeto detectado e pelo centro de projeção da câmera. Usando mais de uma câmera, pode-se determinar a posição e orientação do objeto.
  - Precisão/Velocidade: a velocidade de captura depende muito do sensor empregado. Uma câmera padrão NTSC consegue capturar imagens a taxas de 30 quadros por segundo, limitando a amostragem, enquanto câmeras digitais podem capturar a taxas de 200 a 1000 quadros por segundo. A precisão dos dados depende das técnicas de visão computacional empregadas: calibração de câmera,

extração de informação da imagem e utilização de filtro para evitar tremidos.

- Interferência do meio: O laser e outros emissores podem refletir em objetos próximos atrapalhando a medição.
- Restrições: A câmera deve estar sempre enxergando o objeto sendo rastreado e o emissor de luz não pode estar obstruído. Uma solução com três ou quatro câmeras oferece redundância e permite que uma ou duas sejam bloqueadas antes do sistema deixar de funcionar.
- Exemplos: Existem muitos equipamentos ópticos utilizados em captura de movimento [25], porém poucos em realidade virtual. Alguns exemplos de dispositivos usados em RV são o DragonFly [24] (Figura 8), o LaserBird da Ascension Technology (Figura 9) e o produto final desta tese, descrito no capítulo 3.

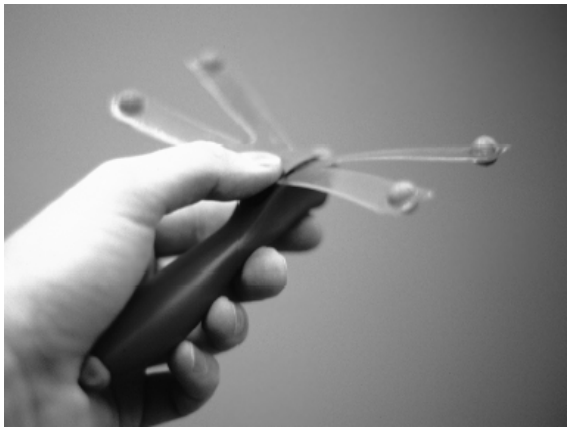


Figura 8: DragonFly - dispositivo óptico desenvolvido pelo Instituto Fraunhofer de Engenharia Industrial da Alemanha.

Figura 9: LaserBird da Ascension Technology: sensor e feixe de laser.

2.3. Ferramentas

Os sistemas de RV precisam de um software para apresentar a aplicação para o usuário final. Os softwares de RV são complexos, pois seus dispositivos são complexos. A fim de facilitar a programação dessas aplicações, foram criadas várias soluções com o intuito de prover um ambiente único de desenvolvimento.

Um ambiente de desenvolvimento permite que o usuário se concentre na programação da aplicação ao invés de se preocupar com a gerência do ambiente de RV. Ele define uma arquitetura que inclui componentes para gerência de dispositivos de entrada, apresentação das saídas visual, auditiva e outras, além de um mecanismo de configuração. Alguns ambientes podem ainda estender a arquitetura a fim de gerenciar recursos como alocação de memória, multi-tarefas e comunicação.

Várias ferramentas para desenvolvimento de aplicações em RV existem disponíveis comercialmente ou gratuitamente, entre elas, o Avango [27], Vess [8], Diverse [1], VRJuggler [5] e o ViRAL (*Virtual Reality Abstraction Layer*), sendo sendo as duas últimas analisadas neste trabalho.

2.3.1. VRJuggler

O VRJuggler é uma biblioteca de realidade virtual, com código aberto, multi-plataforma e independente de dispositivo. Ela fornece uma abstração dos dispositivos de entrada e saída.

O VRJuggler é composto de vários módulos, sendo *vrj*, *jccl*, *gadgeteer* e *vpr* os principais. O módulo *vrj* é responsável pela integração de todos os módulos. Ele fornece uma camada acima do sistema operacional, permitindo que uma aplicação desenvolvida com o VRJuggler possa executar em diferentes sistemas, não só operacionais, mas em diversos sistemas de RV, como em um monitor, em uma CAVE ou em um HMD.

O módulo *jccl* é um sistema de configuração baseado em XML. O usuário utiliza arquivos de configuração para definir quais os dispositivos de entrada serão utilizados e a disposição dos dispositivos de saída. Baseado nos arquivos de configuração, a biblioteca carrega os módulos de controle (*drivers*) adequados. Os

arquivos de configuração possuem um meta-arquivo para que o sistema de configuração identifique os parâmetros dos dispositivos sendo carregados.

Como o sistema está baseado em meta-arquivo, não é possível estender o tipo das propriedades dos dispositivos. Com isso, os únicos tipos que podem ser utilizados são os que o VRJuggler fornece, como inteiro, real, vetor, caracteres e alguns tipos abstratos de dados.

A biblioteca permite que novos dispositivos sejam criados pelo desenvolvedor, porém, ele não é capaz de incorporar esses novos dispositivos na interface de configuração, a não ser que o meta-arquivo seja modificado e a descrição do novo dispositivo inserida a ele.

O módulo *gadgeteer* é responsável pela gerência de dispositivos. Ele trata da configuração, controle, aquisição e representação do dado dos dispositivos de RV. Novos dispositivos são criados através da implementação de uma ou mais interfaces ilustradas na Figura 10.

A classe *vjPosition* representa um dispositivo que fornece como dado uma matriz de transformação. Um objeto *vjDigital* é um dispositivo que possui dados binários, do tipo 'verdadeiro' ou 'falso'. Um exemplo desses dispositivos são os botões do *mouse*. Um objeto *vjAnalog* é um dispositivo que possui informações reais contínuas.

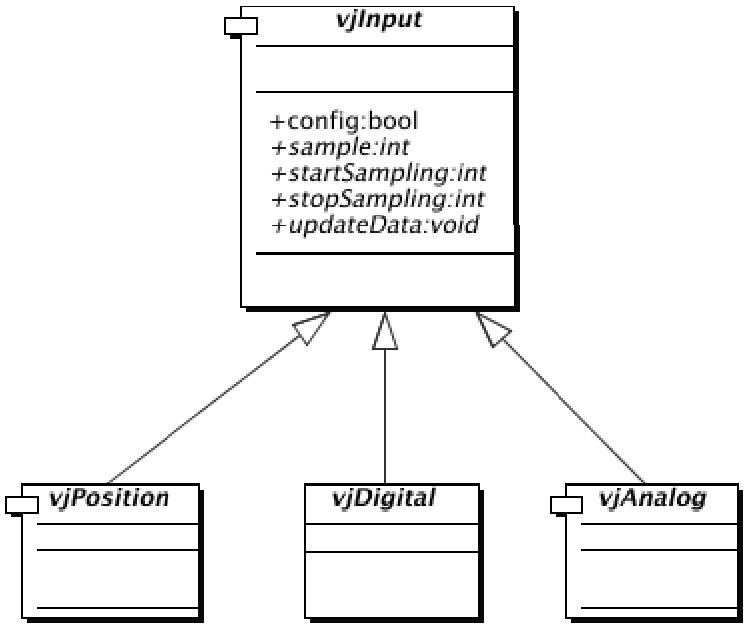


Figura 10: Classes de dispositivo do VRJuggler.

Um *joystick*, por exemplo, pode ser criado herdando sua interface das classes *vjPosition* e *vjDigital*, pois o mesmo é capaz de prover informações de posição e digitais, através de seus botões.

O método *config* é chamado durante a leitura dos arquivos de configuração. O método *sample* é onde as leituras dos dispositivos são feitas. A leitura é feita é uma *thread* separada e os dados são armazenados para uso futuro. Os métodos *startSampling* e *stopSampling* são responsáveis por iniciar e parar a execução dessa *thread*. O método *updateData* atualiza os dados do dispositivo, ou seja, recupera os dados que a *thread* estava armazenando e disponibiliza este dado para a aplicação. Esse método, portanto, deve ser *thread-safe*.

Finalmente, o módulo *vpr* é responsável por prover uma abstração independente de plataforma para *threads*, *sockets* (TCP/UDP) e comunicação serial.

Uma aplicação no VRJuggler é um objeto e sua classe deve derivar da interface de aplicação da biblioteca. A Figura 11 ilustra as principais interfaces de aplicação disponíveis nativamente no VRJuggler.

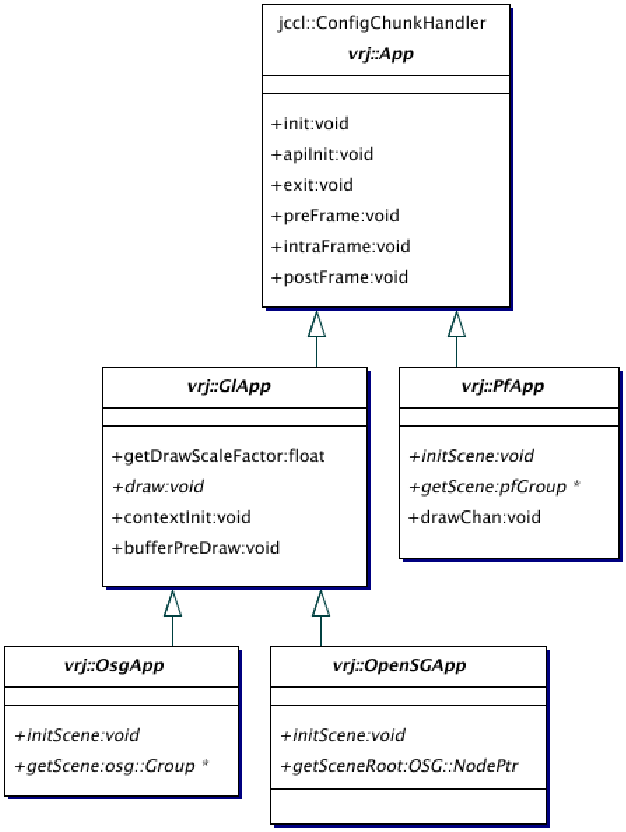


Figura 11: Hierarquia de classes de aplicação do VRJuggler.

A classe *vrj::App* é a interface que define os principais métodos que uma aplicação deve implementar. A classe *vrj::GlApp* é, também, uma interface para uma aplicação que irá utilizar OpenGL [19], e a classe *vrj::PfApp* é uma interface para uma aplicação que irá utilizar OpenGL Performer [20]. Derivadas de *vrj::GlApp* estão as aplicações que utilizam grafos de cena para renderização, como a *vrj::OsgApp* que utiliza o OpenSceneGraph [21].

Ao definir a interface da aplicação, o VRJuggler tem controle do laço principal aplicação. A função ***main*** de uma aplicação desenvolvida com o VRJuggler está descrita na Tabela 1.

```
int main( int argc, char *argv[] )
{
1 vrj::Kernel *kernel=vrj::Kernel::instance();  // pega o kernel
2  simpleApp *app = new SimpleApp(); // Cria o objeto da aplicação
3  kernel->loadConfigFile(argv);     // Configura o kernel
4  kernel->start();                  // Inicia a thread do kernel
5  kernel->setApplication( app );    // Passa a apl. para o kernel
   kernel->waitForKernelStop();    // Bloqueado até o kernel parar
   return 0;
}
```

Tabela 1: Função main de uma aplicação utilizando o VRJuggler.

Na linha 1 é retornado o objeto núcleo (*kernel*) do VRJuggler. Na linha 2, um objeto da aplicação (S*impleApp*) é instanciado. Na linha 3, os arquivos de configuração são passados para o objeto *kernel* para que ele se configure.

Na linha 4, o VRJuggler começa a ser executado. Nesse momento, o *kernel* cria uma nova *thread* de execução e começa o seu processamento interno.

Qualquer mudança de estado do *kernel* a partir de agora exige uma reconfiguração do VRJuggler, seja através da inclusão de novos arquivos de configuração ou da mudança da aplicação ativa. Isso permite que o VRJuggler seja reconfigurado em tempo de execução, porém, somente para adição de novos dispositivos. Os dispositivos que já estavam sendo utilizados não podem ser modificados.

A aplicação que será executada pelo *kernel* foi configurada na linha 5.

A Tabela 2 apresenta algumas vantagens e desvantagens no uso da biblioteca VRJuggler:

Vantagens	Desvantagens
• Código aberto; • Multi-plataforma, multi-dispositivos de RV; • Abstração dos dispositivos através de arquivos de configuração; • Desenvolvimento da aplicação é simples, bastando criar uma classe que derivará de alguma das interfaces ilustradas na Figura 11; • Compilação de uma nova aplicação é bastante rápida (tendo já os arquivos binários pré-compilados do núcleo do VRJuggler).	• O arquivo de configuração não é fácil de ser editado manualmente; • A interface de edição de arquivos de configuração, desenvolvida em Java, é complicada e não permite que novos dispositivos sejam configurados através dela; • A compilação dos módulos do VRJuggler é bastante complicada, exigindo que sejam sempre utilizados os arquivos binários pré-compilados; • O VRJuggler depende de muitas bibliotecas externas, tornando ainda mais difícil a sua compilação; • O usuário não possui controle do laço principal da aplicação; • Uma aplicação desenvolvida com o VRJuggler é completamente imersiva, ou seja, não é possível utilizar ao mesmo tempo o VRJuggler e um sistema de diálogos convencional (menus, listas, botões, etc.); • Os eventos gerados pelos dispositivos de entrada são pré-definidos e não podem ser estendidos; • Reconfiguração em tempo de execução é limitada.

Tabela 2: Vantagens e desvantagens do VRJuggler.

2.3.2. ViRAL (*Virtual Reality Abstraction Layer*)

O ViRAL é uma ferramenta desenvolvida em C++ e Qt [28], utilizada para facilitar o desenvolvimento de aplicações de RV. As aplicações que utilizam o ViRAL não precisam saber, por exemplo, em quantas janelas, com quantos usuários ou com quais dispositivos ela irá executar, porque o ViRAL abstrai o contexto onde elas serão executadas.

O ViRAL pode ser usado de duas maneiras: como a aplicação principal ou embutido. Ao ser executado como a aplicação principal, ele carrega as suas janelas e menus e os diversos *plugins* criados pelos desenvolvedores. Um *plugin* é um arquivo utilizado para alterar, melhorar ou estender as operações de uma aplicação principal (no caso, o ViRAL).

A segunda maneira de se utilizar o ViRAL é embuti-lo em uma aplicação. Ele funcionará como escravo e a aplicação principal será o mestre. O ViRAL possui seis sistemas que podem ser utilizados embutidos em uma aplicação, sendo eles, o de ambiente, usuário, janela, dispositivo, cena e *plugin*.

2.3.2.1. Sistema de Cena

Uma Cena (os termos sublinhados se referem aos objetos dos sistemas da biblioteca) é uma aplicação gráfica que é carregada como um *plugin* pelo ViRAL, sendo que várias cenas podem ser carregadas e utilizadas simultaneamente.

O sistema de cenas mantém uma floresta de cenas carregadas, sendo que a raiz de cada árvore é a Cena propriamente dita e seus descendentes são chamados de Objetos de Cena. Cada Cena e Objeto de Cena possui uma interface de configuração que é carregada pelo sistema quando solicitada.

Quando o ViRAL é utilizado embutido, a aplicação principal implementará a sua interface gráfica convencional (botões, menus e janelas) e exportará para o ViRAL as Cenas e os Objetos de Cenas que serão carregados para exibição no ambiente virtual. Isso permite que a aplicação satisfaça os requisitos convencionais de um sistema, adicionando a eles novas funcionalidades que somente são úteis em um ambiente virtual.

2.3.2.2. Sistema de Usuários

O termo Usuário no ViRAL se refere à "entidade" que está visualizando a cena, ou seja, ele é uma câmera no mundo virtual. Um Usuário pode ser a visão de uma pessoa real posicionada em uma CAVE, uma visão externa dessa pessoa, ou uma câmera móvel, por exemplo.

O Usuário possui como atributos a distância interocular (para visualização estereoscópica), a orientação da cabeça, e a posição e orientação do corpo. Outro atributo importante que o Usuário possui é a Cena da qual ele irá participar. Além disso, o Usuário pode estar acompanhado de várias superfícies de projeção.

2.3.2.3. Sistema de Janela

Um *plugin* de Cena (ou uma aplicação mestre, no caso de uso embutido), pode ser executado em diversas configurações de Janelas. Para cada Janela está associada uma projeção. A projeção é a conexão da Janela com a Cena, pois uma projeção pertence a um Usuário e o Usuário contém uma Cena.

Uma Janela possui um atributo (canal) que define como será exibida a projeção: visão mono, do olho esquerdo, do olho direito ou em estéreo. A Figura 12 ilustra a interface de configurações de janelas no ViRAL.

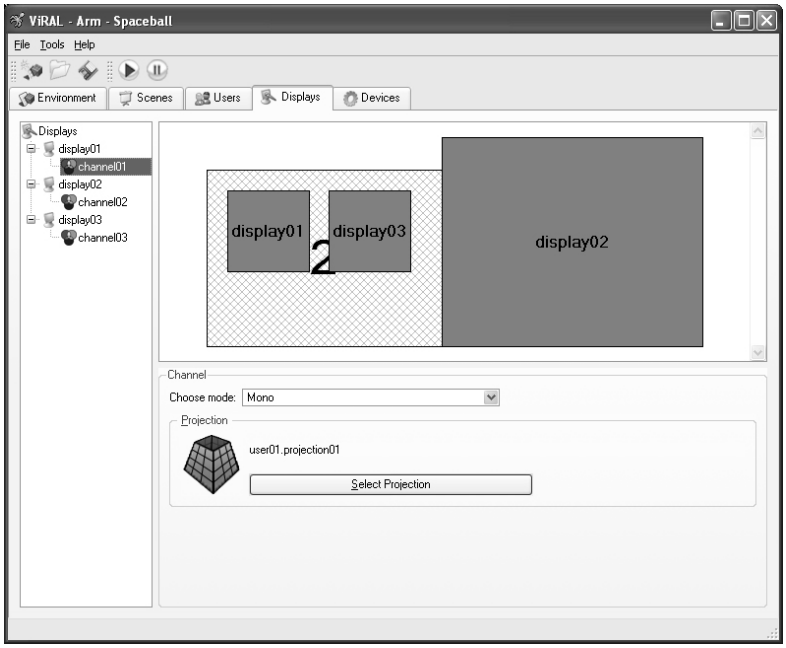


Figura 12: Interface de criação de janelas no ViRAL.

2.3.2.4. Sistema de *Plugin*

No ViRAL existem 3 tipos de *plugins*: de Dispositivo, de Cena e de Objeto de Cena. Cada um desses *plugins* herdam de uma classe específica do ViRAL e exportam seus métodos. O sistema de plugins é responsável pelo carregamento dinâmico e conexão de eventos.

Um dispositivo novo é criado como uma biblioteca de carregamento dinâmico. Nessa biblioteca, o desenvolvedor deve criar uma classe que herda sua interface da classe *vral::Device*. A Tabela 3 mostra a interface dessa classe e os principais métodos que devem ser implementados.

```
namespace vral
{
  class Device : public Object
  {
      virtual const Object * getSender() const;
      virtual const Object * getReceiver() const;
      virtual void dispatchChanges() = 0;
      virtual QWidget * loadPropertyFrame();
  };
}
```

Tabela 3: Interface da classe *vral::Device*.

De baixo para cima, o método *loadPropertyFrame* é responsável por retornar o diálogo (*Qwidget* do Qt) de configuração daquele dispositivo. A Figura 13 ilustra o diálogo de configuração de uma luva de realidade virtual.

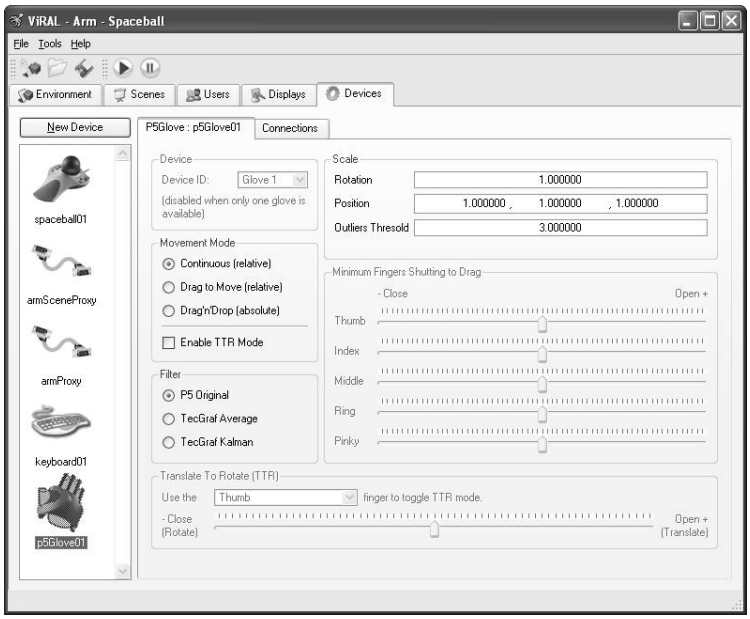


Figura 13: Configuração de uma luva de RV no ViRAL.

O método *dispatchChanges* transmite os eventos de um dispositivo, geralmente uma vez a cada passo de execução. O sistema de eventos no ViRAL utiliza o mecanismo de *signal/slot* do Qt. Esse mecanismo é utilizado para programação de componentes e comunicação entre objetos. Os sinais (*signals*) são emitidos por objetos quando os mesmo mudam de estado. As aberturas (*slots*) são utilizadas para receber sinais.

Os métodos *getSender* e *getReceiver* são utilizados para implementação de um mecanismo de redirecionamento (*proxy*). Um exemplo de uso desse mecanismo é quando se deseja transformar um Usuário do ViRAL em um Dispositivo. A partir de um dispositivo de redirecionamento, os eventos podem ser enviados a um Usuário. Dessa forma é possível modificar a orientação e a posição do usuário através de um dispositivo de RV.

Os *plugins* de Cena e de Objetos de Cena possuem o mesmo princípio de funcionamento dos *plugins* de Dispositivo, porém as interfaces desses *plugins* são definidas pelas classes *vral::Scene* e *vral::SceneObject*, como visto na Tabela 4.

```
namespace vral
{
  class SceneObject : public Object
  {
    virtual QWidget * loadPropertyFrame() = 0;
  };

  class Scene : public SceneObject
  {
    virtual bool isReady() const = 0;
    virtual bool checkTranslation() = 0;

    virtual void initializeContext() = 0;
    virtual void preDrawImplementation);
    virtual void drawImplementation();
  };
}
```

Tabela 4: Interface das classes *vral::Scene* e *vral::SceneObject*.

No ViRAL existe uma hierarquia que define quem é Cena e quem é Objeto de Cena. Uma Cena é a raiz dessa hierarquia e todos os seus filhos são objetos dessa cena. O método *loadPropertyFrame* carrega o diálogo de configuração do Objeto de Cena. *isReady* informa ao ViRAL que a Cena está pronta para ser executada. O método *checkTranslation* valida uma movimentação dentro do ViRAL.

As Cenas são responsáveis por configurar o estado do OpenGL para que o ViRAL possa utilizá-lo para renderização. Para isso existe o método

initializeContext. Os métodos *preDrawImplementation* e *drawImplementation* têm por finalidade atualizar a cena e renderizá-la, respectivamente. O ViRAL calcula automaticamente as matrizes para serem carregadas pelo OpenGL baseado nas configurações de projeção e do modo de exibição de um canal.

```
SimpleScene::SimpleScene() : Scene()
{
      mQuadricObj = gluNewQuadric();
      gluQuadricOrientation( mQuadricObj, GLU_OUTSIDE );
}
SimpleScene::~SimpleScene()
{
      gluDeleteQuadric( mQuadricObj );
}
void SimpleScene::initializeContext()
{
      glClearColor( 1.0f, 1.0f, 1.0f, 1.0f );
      glEnable( GL_CULL_FACE );
}
void SimpleScene::drawImplementation( const
                              vral::SceneDrawingData &data )
{
      glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
      glEnable( GL_LIGHTING );
      glEnable( GL_COLOR_MATERIAL );

      glMatrixMode( GL_PROJECTION );
      glLoadMatrixf( data.getProjection().get() );

      glMatrixMode( GL_MODELVIEW );
      glLoadMatrixf( data.getModelview().get() );

      glPushMatrix();
            glTranslatef( 0.0f, 0.0f, -50.0f );
            glColor3f( 1.0f, 0.0f, 0.0f );
            gluQuadricNormals( mQuadricObj, GLU_SMOOTH );
            gluSphere( mQuadricObj, 15.0, 32, 32 );
      glPopMatrix();
}
```

Tabela 5: Exemplo de uma cena simples criada no ViRAL.

A Tabela 5 mostra um exemplo de uma cena simples que implementa os métodos *initializeContext* e *drawImplementation*. A estrutura *SceneDrawingData* contém as matrizes de modelo e projeção que serão utilizadas para configurar a câmera do OpenGL.

2.3.2.5. Sistema de Dispositivo

O sistema de dispositivo é responsável pela conexão e transmissão de eventos. Os eventos são conectados através da interface ilustrada na Figura 14. A cada quadro o ViRAL chama o método *dispatchChanges* dos seus *plugins*. Apesar dos *plugins* poderem funcionar em *threads* independentes, os dados só devem ser transmitidos mediante essa chamada de método.

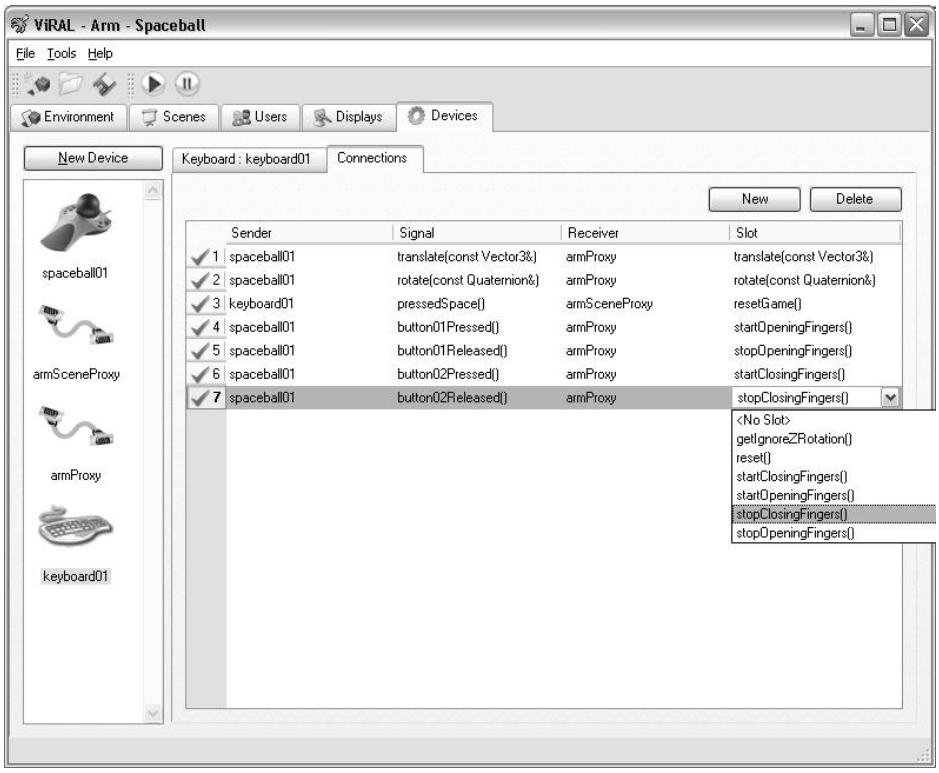


Figura 14: Interface de conexão de eventos no ViRAL.

Utilizando o mecanismo de *signal/slot* do Qt, o ViRAL permite que as conexões sejam configuradas em tempo de execução. Com isso, o usuário pode trocar, a qualquer momento, o dispositivo com o qual ele está interagindo com a aplicação.

2.3.2.6. Sistema de Ambiente

O Sistema de Ambiente é responsável pela gravação e leitura de dados de uma configuração do ViRAL. Ele armazena os arquivos utilizando um formato XML comprimido.

Esse sistema visita todos os objetos instanciados dentro do ViRAL (dispositivos, janelas, cenas, etc.) e grava as suas informações utilizando um padrão de projeto chamado Memento [11]. Todo *plugin* criado para o ViRAL deve implementar dois métodos herdados de sua classe base. A Tabela 6 ilustra como um *plugin* de cena grava e recupera as informações de um arquivo. A classe *vral::DataNode* é a classe que implementa o padrão Memento.

```
class CaptureDevice : public vral::Device
{
   // Métodos para gravação e leitura de dados
   virtual void getMemento( vral::DataNode &node )
   {
      // inclui o memento do pai
      Object::getMemento( node );

      node.setInt( "gridSize", mGridSize );
      node.setInt( "cameraNumber", mCameraNumber );
   }
   virtual void setMemento( const vral::DataNode &node )
   {
      mGridSize = node.getInt( "gridSize" );
      mCameraNumber = node.getInt( "cameraNumber" );
   }

public:
   // Atributos que devem ser gravados
   int mCameraNumber;
   int mGridSize;
}
```

Tabela 6: *Plugin* de cena com métodos de gravação e leitura de arquivo.

2.3.2.7. Vantagens e Desvantagens

A Tabela 7 apresenta algumas vantagens e desvantagens da utilização do ViRAL, analisadas pelo perfil do desenvolvedor de novas funcionalidades. A utilização do Qt pelo ViRAL traz uma grande vantagem que é a rápida construção de interfaces gráficas, além de ser multi-plataforma.

Vantagens	Desvantagens
• Interface gráfica através do Qt; • Uso de sistema de *plugins*; • Uso embutido permite integrar aplicações *desktop* e aplicações imersivas; • Rápido desenvolvimento de novos *plugins*; • Criação de eventos arbitrários permitindo programação de componentes; • Reconfiguração em tempo de execução.	• Exige conhecimento de orientação a objeto, tornando o aprendizado mais demorado; • Novas aplicações ou *plugins* devem ser desenvolvidas com Qt, que é uma ferramenta comercial.

Tabela 7: Vantagens e desvantagens do ViRAL.

3
Rastreamento

Em Realidade Virtual (RV) e Realidade Aumentada (RA), o processo de acompanhamento das coordenadas de um objeto em tempo real chama-se **rastreamento**. Os objetos mais comuns a serem rastreados são os capacetes de RV e dispositivos manuais de interação tri-dimensional. Em muitos casos, a posição (três graus de liberdade) e a orientação (três graus de liberdade) do objeto devem ser recuperados, dando origem a sistemas de rastreamento com seis graus de liberdade.

Para um sistema de RV e RA, o rastreamento deve possuir as seguintes características:

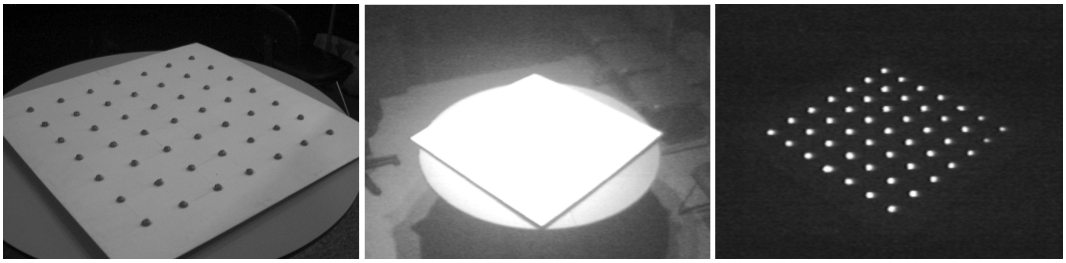
- Velocidade: as aplicações devem funcionar a taxas de 30 a 40 Hz, para não atrapalharem a sensação de imersão;
- Precisão: as aplicações de RA, principalmente, necessitam que os objetos virtuais casem com os objetos reais, no processo de registro [3, 4];
- Ruído: quando não há movimento do objeto sendo rastreado, o mesmo deve estar parado no mundo virtual;
- Robustez: pequenos movimentos no mundo real devem gerar pequenos movimentos no mundo virtual. Os pontos fora da curva devem ser devidamente descartados;
- Mobilidade: o dispositivo que está sendo rastreado e o rastreador não devem atrapalhar o movimento do usuário;
- Predição: a fim de evitar atrasos na renderização, técnicas podem ser utilizadas para prever a posição futura do objeto sendo rastreado.

O rastreamento óptico pode oferecer vantagens sobre os métodos magnéticos ou mecânicos. Ele permite que sejam utilizados dispositivos sem fio, facilitando a mobilidade do usuário, rastreamento de múltiplos objetos, oferece pouca sensibilidade a interferências externas e pode ser utilizado com câmeras digitais convencionais, tornando-se um produto de fácil acesso. A maioria dos

sistemas ópticos utiliza objetos cobertos por materiais retroreflexivos e receptores com filtros infravermelhos. Na Figura 15 podemos observar um capacete de RV com 3 esferas cobertas com esse tipo de material que serão rastreadas para determinar a posição e orientação da cabeça do usuário.

Figura 15: HMD rastreado por esferas retroreflexivas.

A Figura 16 ilustra a visualização de um conjunto de esferas retroreflexivas utilizando-se uma câmera convencional, uma câmera com sensores infra-vermelhos dentro de uma sala com lâmpadas incandescentes e outra em uma sala com lâmpadas fluorescentes. O problema ilustrado é que a luz amarela contém a componente infravermelha, atrapalhando a captura da imagem.



(i) (ii) (iii)

Figura 16: Esferas retroreflexivas: (i) Câmera convencional; (ii) Câmera com filtro infravermelho em ambiente com lâmpadas incandescentes; (iii) Câmera com filtro infravermelho em ambiente com lâmpadas fluorescentes.

A abordagem mais simples em rastreamento de dispositivos de interação é a utilização de somente um marcador (geralmente esférico). Essa abordagem permite que apenas 3 graus de liberdade (posição) sejam recuperados, utilizando mais de uma câmera. Outra abordagem existente é a utilização de marcadores com uma posição tri-dimensional conhecida. Utilizando propriedades projetivas invariantes, apresentadas em [31], é possível comparar a posição dos pontos da

imagem com padrões em uma base de dados. O padrão que melhor representa o padrão do modelo é escolhido. É possível determinar a posição e a orientação do dispositivo utilizando imagens de mais de uma câmera.

Na presente dissertação, o dispositivo proposto será composto por apenas uma esfera. Ela será rastreada por quatro câmeras a fim de prover redundância. Com apenas uma câmera, é possível determinar um segmento de reta que parte da sua posição e intercepta a imagem no ponto detectado. Com duas câmeras é possível determinar, com uma certa aproximação, a interseção entre dois segmentos e estabelecer a posição da esfera no espaço. Essa aproximação existe porque, devido a erros de calibração, as retas não se interceptam realmente. A heurística empregada para determinar a "interseção" foi a de se utilizar o ponto mais próximo entre as duas retas. Com mais de duas câmeras, o resultado pode ser estabelecido com mais precisão, além do sistema permitir que a visão de algumas câmeras seja bloqueada em algum momento, provendo maior mobilidade ao usuário.

3.1. Calibração da câmera

A etapa inicial no processo de rastreamento feito por câmeras digitais consiste na calibração das câmeras. Calibrar uma câmera é encontrar parâmetros (vistos mais adiante) que permitam definir uma matriz de transformação que levam os pontos do mundo tridimensional para o plano da imagem gerada por aquela câmera.

Existem diversas técnicas de calibração de câmeras descritas na literatura, sendo que o algoritmo utilizado nesta tese foi o algoritmo proposto por Tsai em [29, 30].

3.1.1.
Método de Tsai

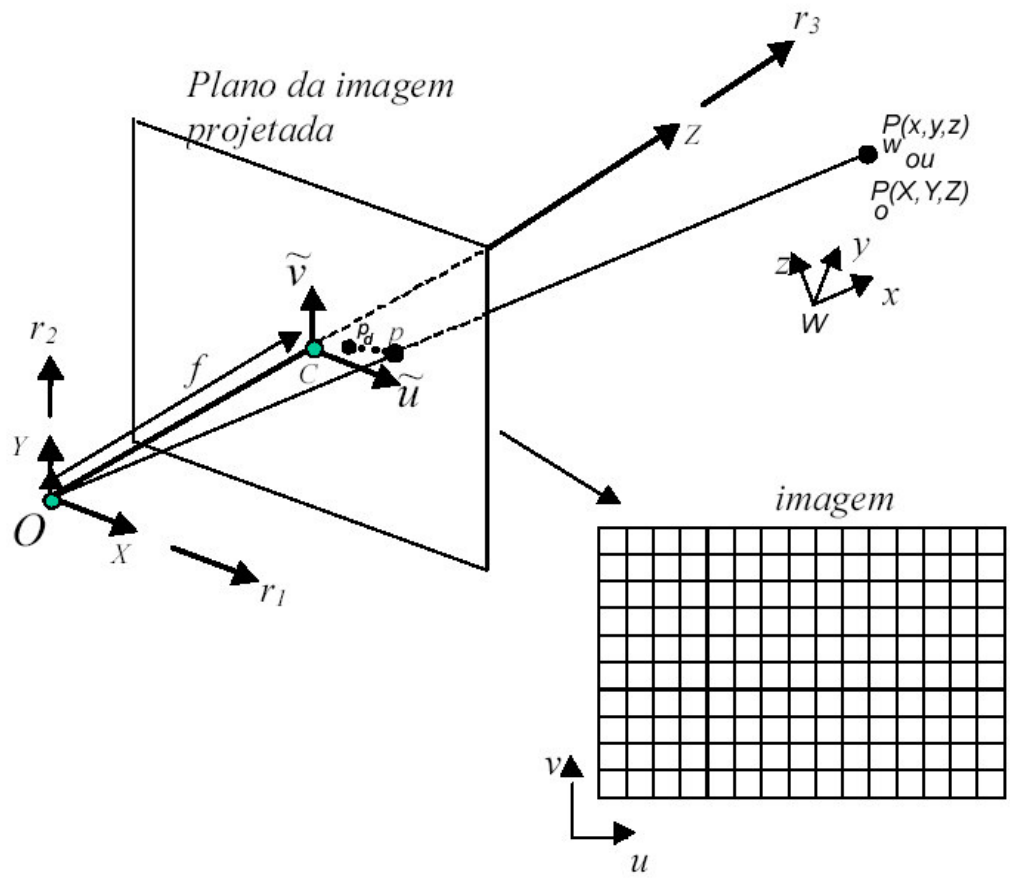


Figura 17: Modelo de câmera "*pinhole*".

A Figura 17 ilustra o modelo de câmera "*pinhole*" utilizado para calibração. (x, y, z) é a coordenada do ponto ***P*** no sistema de coordenadas do mundo 3D. (X, Y, Z) é a coordenada do ponto no sistema de coordenadas da câmera, com origem no ponto ***O***, que é o seu centro óptico, e com seu eixo *z* coincidindo com o eixo óptico. $(\tilde{U}, \tilde{V})$ é o sistema de coordenadas da imagem centralizado em ***C*** (interseção entre o eixo focal e o plano da imagem) e paralelo ao eixo (X, Y). ***f*** é a distância focal, ou a distância entre o plano da imagem e o centro óptico. $p(\tilde{u}, \tilde{v})$ é a coordenada na imagem do ponto (X, Y, Z) se o modelo de câmera fosse perfeito e não houvesse distorção. $p_d(\tilde{u}_d, \tilde{v}_d)$ é o ponto real na imagem onde (X, Y, Z) é projetado. Contudo, como o sistema do computador é discretizado em *pixels*, alguns parâmetros que relacionam a coordenada do ponto na imagem com a coordenada (u, v) devem ser especificados e calibrados. Os

quatro passos da transformação total do ponto (x, y, z) em (u, v) estão ilustrados na Figura 18.

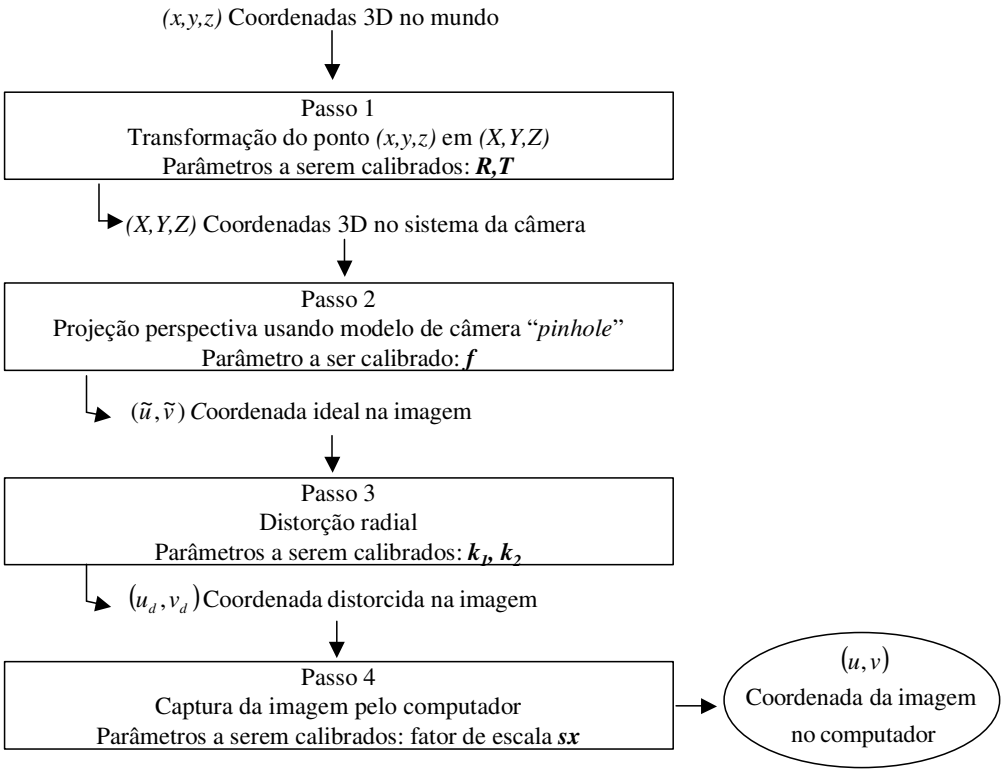


Figura 18: Os quatro passos da transformação de coordenadas do mundo em coordenadas da imagem no computador

Passo 1: Transformação do ponto em coordenadas do mundo (x, y, z) para as coordenadas do sistema da câmera (X, Y, Z)

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = R \begin{bmatrix} x \\ y \\ z \end{bmatrix} + T, \tag{3.1}$$

onde R é uma matriz de rotação 3x3

$$R \equiv \begin{bmatrix} \vec{r}_1 \\ \vec{r}_2 \\ \vec{r}_3 \end{bmatrix} = \begin{bmatrix} r_{1x} & r_{1y} & r_{1z} \\ r_{2x} & r_{2y} & r_{2z} \\ r_{3x} & r_{3y} & r_{3z} \end{bmatrix}, \tag{3.2}$$

e T é um vetor de translação

$$T \equiv \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}. \tag{3.3}$$

Os parâmetros a serem calibrados são *R* e *T*. Convém observar que o ponto $T = (t_x, t_y, t_z)$ representa a origem dos sistema do mundo escrita em coordenadas da câmera. As matrizes *R* e *T* podem ser vistas como uma única matriz $[R \mid T]$

$$[R|T] = \begin{bmatrix} \vec{r}_1 & t_x \\ \vec{r}_2 & t_y \\ \vec{r}_3 & t_z \end{bmatrix} = \begin{bmatrix} r_{1x} & r_{1y} & r_{1z} & t_z \\ r_{2x} & r_{2y} & r_{2z} & t_y \\ r_{3x} & r_{3y} & r_{3z} & t_z \end{bmatrix} \tag{3.4}$$

Passo 2: Transformação das coordenadas da câmera (X, Y, Z) para as coordenadas ideais da imagem $(\tilde{u}, \tilde{v})$ usando projeção perspectiva

$$\tilde{u} = f \frac{X}{Z} \tag{3.5}$$

$$\tilde{v} = f \frac{Y}{Z}. \tag{3.6}$$

O parâmetro a ser calibrado é a distância focal *f*.

Passo 3: Distorção radial das lentes: os parâmetros a serem calibrados são os coeficientes de distorção. A modelagem das distorções da lente pode ser encontrada em [17].

Passo 4: Transformação das coordenadas reais da imagem em coordenadas da imagem no computador levando em consideração os tamanhos dos *pixels* da tela.

Os parâmetros listados acima podem ser divididos em duas categorias: *intrínsecos* e *extrínsecos*. Os parâmetros intrínsecos da câmera são sua distância focal (f), os coeficientes de distorção da lente, o fator de escala (tamanho de um *pixel* no plano de captura da imagem) e as coordenadas da origem (u_o, v_0) no plano da imagem. Os parâmetros extrínsecos são a matriz de rotação *R* e o vetor de translação *T*.

O modelo ilustrado acima pode ser simplificado se não levarmos em consideração os passos 3 e 4, admitindo, portanto, que não existe distorção radial e que o centro da imagem no computador coincide com o centro óptico.

As equações (3.4), (3.5) e (3.6) podem ser compatibilizadas a fim de produzir uma transformação direta do ponto (x, y, z) do mundo no ponto (u, v) na imagem

$$\begin{bmatrix} us \\ vs \\ s \end{bmatrix} = \begin{bmatrix} f\vec{r}_1 & ft_x \\ f\vec{r}_2 & ft_y \\ \vec{r}_3 & t_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}, \tag{3.7}$$

onde (us, vs, s) são coordenadas homogêneas. Multiplicando, encontramos:

$$u - u_0 = f \frac{r_{1x}x + r_{1y}y + r_{1z}z + t_x}{r_{3x}x + r_{3y}y + r_{3z}z + t_z} \tag{3.8}$$

e

$$v - v_0 = f \frac{r_{2x}x + r_{2y}y + r_{2z}z + t_y}{r_{3x}x + r_{3y}y + r_{3z}z + t_z}. \tag{3.9}$$

Uma observação a ser feita é que a escala existente na imagem está embutida na distância focal *f*.

Na presente dissertação, o método de Tsai coplanar foi utilizado, isto é, o método que encontra os parâmetros baseado em pontos do mundo tridimensional contidos no plano $z = 0$.

A partir das equações (3.8) e (3.9), assumindo $z = 0$ para todos os pontos, temos

$$u - u_0 = \tilde{u} = f \frac{r_{1x}x + r_{1y}y + t_x}{r_{3x}x + r_{3y}y + t_z} \tag{3.10}$$

e

$$v - v_0 = \tilde{v} = f \frac{r_{2x}x + r_{2y}y + t_y}{r_{3x}x + r_{3y}y + t_z}. \tag{3.11}$$

Dividindo (3.10) por (3.11) e em seguida o numerador e o denominador da razão da direita por t_y, temos:

$$\frac{r_{1x}}{t_y} x_i \tilde{v}_i + \frac{r_{1y}}{t_y} y_i \tilde{v}_i - \frac{r_{2x}}{t_y} x_i \tilde{u}_i - \frac{r_{2y}}{t_y} y_i \tilde{u}_i + \frac{t_x}{t_y} \tilde{v}_i = \tilde{u}_i. \tag{3.12}$$

O indicador *i* subescrito em cada variável corresponde aos vários pontos amostrados e utilizados no método Tsai.

Obtemos com isso um sistema linear $Au = b$, onde *A* é uma matriz $n \times 5$ e cada linha A_i é dada por $(x_i\tilde{v}_i, y_i\tilde{v}_i, -x_i\tilde{u}_i, -y_i\tilde{u}_i, \tilde{v}_i)$, *u* sendo um vetor dado por

$$\begin{pmatrix} U_1 & U_2 & U_3 & U_4 & U_5 \end{pmatrix} = \begin{pmatrix} \frac{r_{1x}}{t_y} & \frac{r_{1y}}{t_y} & \frac{r_{2x}}{t_y} & \frac{r_{2y}}{t_y} & \frac{t_x}{t_y} \end{pmatrix} \tag{3.13}$$

e cada elemento do vetor *b* sendo dado por $\tilde{u}_i$.

Como $\vec{r}_1$, $\vec{r}_2$ *e* $\vec{r}_3$ são ortonormais, e designando $\alpha = \frac{r_{1z}}{t_y}$ e $\beta = \frac{r_{2z}}{t_y}$, temos que

$$\begin{cases} \alpha\beta = -U_1U_3 - U_2U_4 \\ \alpha^2 + U_1^2 + U_2^2 = \beta^2 + U_3^2 + U_4^2 \end{cases} \tag{3.14}$$

Resolvendo o sistema acima, calcula-se t_y a partir de

$$t_y^2 = \frac{U - \sqrt{U^2 - 4(U_1U_4 - U_2U_3)^2}}{2(U_1U_4 - U_2U_3)^2} \tag{3.15}$$

onde $U = U_1^2 + U_2^2 + U_3^2 + U_4^2$.

Portanto, com estes valores já definidos e utilizando a equação 3.13, é possível determinar os valores de r_{1x}, r_{1y}, r_{2x}, r_{2y} e t_x. Pelo fato dos vetores $\vec{r}_1$ e $\vec{r}_2$ terem normais iguais a um, encontram-se também os valores de r_{1z} e r_{2z}, e conseqüentemente calcula-se $\vec{r}_3$, pois $\vec{r}_1$, $\vec{r}_2$ *e* $\vec{r}_3$ são ortonormais. Finalmente, usando os valores já encontrados e as equações (3.10) e (3.11), calculamos *f* e t_z. Pode haver a necessidade de se ajustar os sinais que inicialmente assumimos como positivos, como, por exemplo, t_y.

Em [26], Szenberg apresenta um estudo de métodos de calibração de câmera, onde ele conclui que o método de Tsai não apresenta bons resultados quando o número de pontos de calibração são poucos. Para resolver esse problema, ele utiliza uma homografia (transformação projetiva planar), na qual é definido um mapeamento 2D para 2D, de um plano contido no mundo real para o plano da imagem, para estimar uma quantidade maior de pontos para serem utilizados no Tsai.

3.2. Pré-processamento

Antes de se calibrar a câmera é necessário um pré-processamento na imagem a fim de se encontrar os objetos a serem rastreados mais facilmente.

A primeira etapa do pré-processamento é a aplicação de um filtro digital para eliminação de pequenos detalhes e ruído que existam na imagem. Um filtro adequado para esta tarefa é o filtro de suavização Gaussiano. Esse filtro é uma

convolução bidimensional cujo núcleo possui algumas propriedades especiais. A forma da distribuição gaussiana unidimensional é dada por:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2}{2\sigma^2}},$$

onde σ é o desvio padrão da distribuição e assumindo que a distribuição tem uma média igual a zero. Um exemplo desta distribuição está ilustrado na Figura 19.

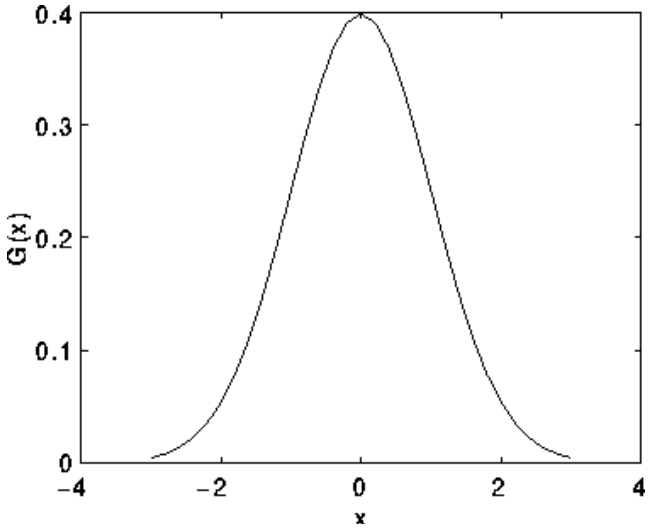


Figura 19: Distribuição gaussiana unidimensional com média 0 e σ = 1.

Em 2D, uma curva gaussiana isotrópica, isto é, circularmente simétrica, possui a forma:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}.$$

Esse distribuição é ilustrada na Figura 20.

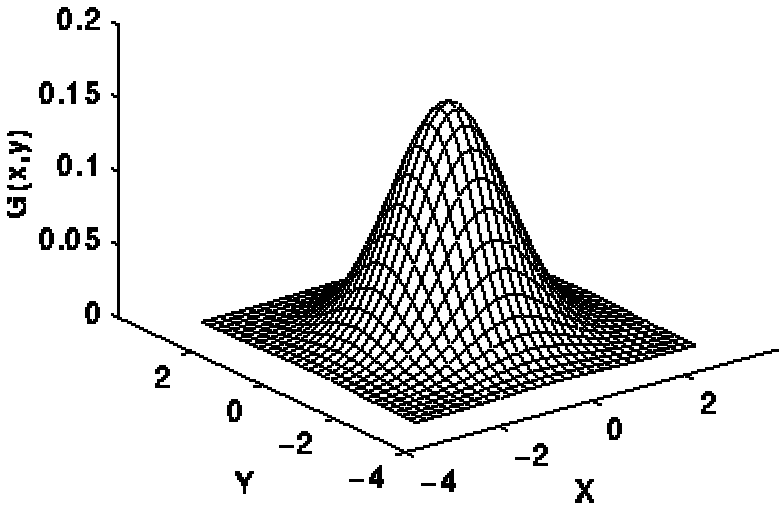


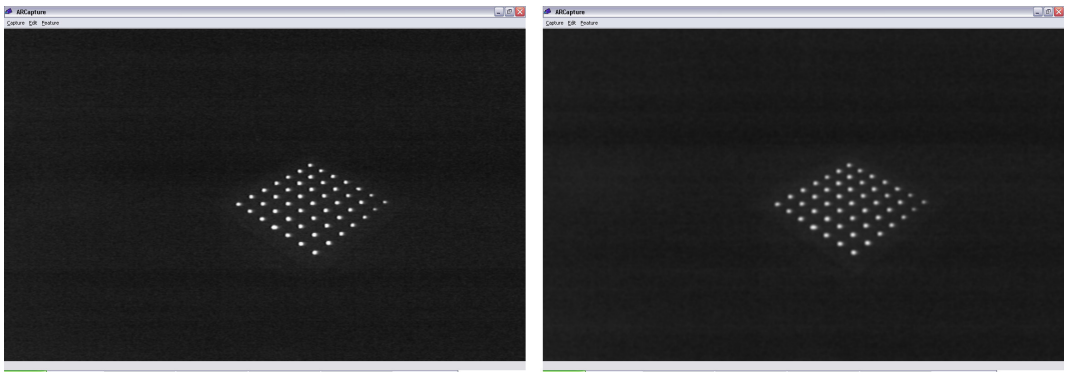
Figura 20: Distribuição gaussiana bidimensional com média (0,0) e σ = 1.

A idéia da suavização gaussiana é a utilização da distribuição 2D como uma função de “espalhamento pontual”, e isso é alcançado através da convolução. Como a imagem é armazenada com uma coleção de *pixels* discretos, é necessária a construção de uma aproximação discreta da função gaussiana antes de se aplicar a convolução. Na teoria, a distribuição gaussiana é não nula em todos os pontos, exigindo que o núcleo da convolução seja infinitamente grande, mas na prática, a distribuição é efetivamente zero a mais de três desvios padrão da média. Logo, é possível quebrar o núcleo a partir desse ponto. A Figura 21 mostra um núcleo de convolução com valores inteiros que aproxima uma distribuição gaussiana com um desvio padrão $\sigma = 1$.

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Figura 21: Aproximação discreta da função gaussiana com $\sigma = 1$.

Uma vez tendo-se calculado um núcleo satisfatório, a suavização gaussiana é aplicada usando métodos de convolução conhecidos [12]. A Figura 22 mostra o resultado da aplicação da suavização gaussiana. A imagem fica mais “borrada”, porém os pontos brancos estão mais conectados ao seu centro.



(i) (ii)

Figura 22: Filtragem gaussiana: (i) imagem original; (ii) filtro gaussiano.

A próxima etapa do pré-processamento é a transformação da imagem em tons de cinza em uma imagem binária, com uma operação denominada *segmentação* ou *threshold*. Na verdade, a operação necessária é bastante simples e chama-se *threshold* binário.

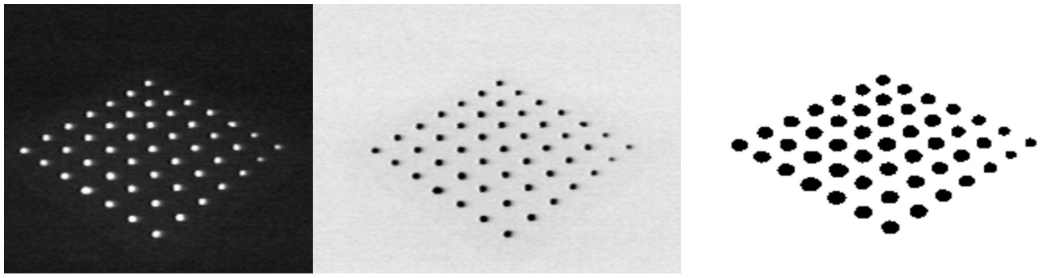
O *threshold* binário funciona da seguinte maneira

$$dst(x, y) = \begin{cases} valorMax, se\ src(x, y) > threshold \\ 0, caso\ contrário \end{cases}, \qquad (3.16)$$

onde

$dst(x, y)$	valor da imagem destino,
$src(x, y)$	valor da imagem de origem,
valorMax	valor para substituição da cor do *pixel* da imagem de origem,
threshold	limite para substituição da cor na imagem de origem.

Como observado na Figura 23, antes de se executar o *threshold* definido pela equação (3.16), a imagem precisou ser invertida.



(i) (ii) (iii)

Figura 23: *Segmentação*: (i) imagem original; (ii) imagem invertida; (iii) imagem segmentada

Após a etapa de pré-processamento, a imagem está pronta para que possam ser extraídos os marcadores para calibração e para que o objeto desejado seja rastreado. No caso da implementação desta dissertação, todos os objetos envolvidos são esferas, logo, algum mecanismo de extração de elipses deve ser executado. É o que será visto na seção a seguir.

3.3. Extração de elipses

A fim de se extrair as elipses ou outros objetos de uma imagem, é necessária a separação das componentes conexas.

A identificação de componentes conexas funciona percorrendo uma imagem, *pixel* por *pixel* (de cima para baixo e da esquerda para direita, por exemplo) a fim de identificar as regiões de *pixels* conexas, isto é, as regiões onde *pixels* adjacentes compartilham os mesmos valores de intensidade *V*. Em uma

imagem binária, temos $V = \{1\}$, contudo em uma imagem em escala de cinza, V possuirá um conjunto de valores, por exemplo: $V = \{51,52,53,\ldots,77,78,79,80\}$. O algoritmo de identificação de componentes conexas funciona com imagens binárias e em tons de cinza e com vários tipos de conectividade entre *pixels*.

A conectividade entre *pixels* descreve a relação entre dois ou mais *pixels*. Para que dois *pixels* estejam conectados, eles precisam validar algumas condições relativas a brilho do *pixel* e adjacência. Primeiramente, seus valores devem estar no mesmo conjunto de valores *V*.

Para formular o critério de adjacência, deve-se primeiro introduzir o conceito de *vizinhança*. Para um *pixel p* com coordenadas (x, y) o conjunto de *pixels* dados por:

$$N_4(p) = \{(x+1, y), (x-1, y), (x, y+1), (x, y-1)\}$$

é chamado de sua 4-vizinhança. Sua 8-vizinhança é definida como:

$$N_8(p) = N_4 \cup \{(x+1, y+1), (x+1, y-1), (x-1, y+1), (x-1, y-1)\}$$

A partir desses conjuntos podemos definir o conceito de 4-conectividade e 8-conectividade. Dois *pixels p* e *q*, ambos tendo valor no mesmo conjunto *V*, são 4-conectados se *q* pertence ao conjunto $N_4(p)$ e 8-conectados se *q* pertence ao conjunto $N_8(p)$. A Figura 24 mostra duas componentes conexas baseadas na 4-conectividade.

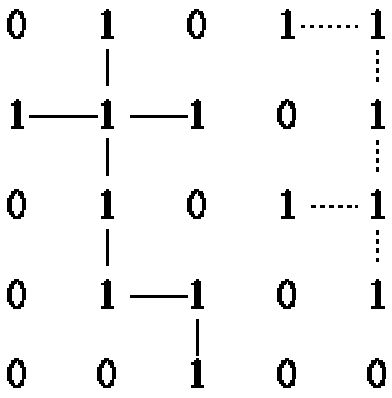


Figura 24: Duas componentes conexas baseado na 4-conectividade.

Para descrever o algoritmo de identificação de componentes conexas, admitimos que a imagem sendo analisada é binária e a conectividade utilizada é a 8.

O algoritmo de identificação de componentes conexas percorre a imagem por linha até encontrar um ponto *p* com valor igual a 1. Quando isso acontece, o algoritmo examina os quatro pontos vizinhos de *p* que já foram encontrados durante o rastreio da imagem, isto é, os vizinhos (i) à esquerda de *p*, (ii) acima de *p*, (iii e iv) as suas duas diagonais superiores. Baseado nessa informação, a identificação ocorre como a seguir:

- se todos os quatros vizinhos são 0, associe um novo identificador a *p*, senão
- se somente um vizinho é 1, associe o seu identificador a *p*, senão
- se um ou mais vizinhos são 1, associe um dos identificadores a *p* e anote as equivalências.

Após completar todo o rastreio da imagem, os pares de identificadores equivalentes são classificados por classes de equivalência e um identificador é associado a cada classe. Como passo final, um novo rastreio é feito na imagem, onde cada identificador é substituído pelo identificador da sua classe de equivalência.

Depois de identificadas as componentes conexas, detectam-se quais dos conjuntos de *pixels* são elipses, a fim de remover possíveis erros na imagem, como a mão do usuário, ou outro objeto que pode vir a surgir. A posição dos marcadores pode ser aproximada pelos centros das elipses encontradas. O método de extração de elipses utilizado foi o de mínimos-quadrado [22].

De posse das elipses, é necessária uma etapa da identificação dos objetos para associação da sua posição na imagem com sua posição no padrão de calibração. No desenvolvimento dessa dissertação, a identificação foi feita manualmente, pois a mesma era executada uma única vez para a mesma configuração de câmeras. Para o rastreamento do objeto, uma única esfera foi utilizada, logo não houve necessidade de identificação.

3.4. Posicionamento da esfera rastreada

A partir das matrizes *R* e *T* obtidas pela calibração de uma câmera, é possível determinar a posição tridimensional da esfera no plano $z = 0$. Com essa

posição, calcula-se um raio que parte da posição da câmera e intercepta o plano, passando pelo centro da esfera.

A fim de determinar a posição real (x, y, z) da esfera, é necessário que haja no mínimo duas câmeras acompanhando-na. De posse do raio que parte das duas câmeras, é possível encontrar a interseção entre elas. Porém, devido a erros provenientes da extração do centro da esfera, dos pixels serem discretos e do próprio método Tsai, a interseção certamente será nula. Logo, o ponto mais próximo entre os dois segmentos de reta deve ser encontrado.

De acordo com a Figura 25, o que queremos encontrar é o ponto médio do vetor w_c.

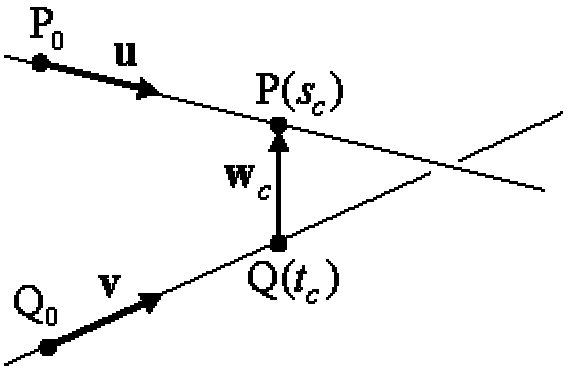


Figura 25: Duas retas não paralelas no espaço. O vetor w_c é o vetor de menor comprimento entre elas.

Podemos escrever as retas como sendo $L_1 : P(s) = P_0 + s(P_1 - P_0)$ e $L_2 : Q(t) = Q_0 + t(Q_1 - Q_0)$. Seja $w(s,t) = P(s) - Q(t)$ um vetor entre pontos das duas retas. O que queremos encontrar é o vetor $w(s,t)$ de menor comprimento para todo *s* e *t*.

Em qualquer espaço *n*-dimensional, as duas linhas, L_1 e L_2, estão mais próximas em um único par de pontos, $P(s_c)$ e $Q(t_c)$, onde $w(s,t)$ possui tamanho mínimo. Se L_1 e L_2 não forem paralelas, então o seguimento de reta $\overline{P(s_c)Q(t_c)}$ que une o par de pontos é perpendicular a ambas as linhas ao mesmo tempo. Nenhum outro segmento entre as retas L_1 e L_2 possui esta propriedade. Logo, o vetor $w_c = w(s_c, t_c)$ é perpendicular as direções $\vec{u}$ e $\vec{v}$, e isto é equivalente a satisfazer as duas equações $u \cdot w_c = 0$ e $v \cdot w_c = 0$.

As duas equações são resolvidas substituindo-se

$$w_c = P(s_c) - Q(t_c) = w_0 + s_c\vec{u} - t_c\vec{v},$$

onde $w_0 = P_0 - Q_0$, nas duas equações lineares:

$$(\vec{u}\cdot\vec{u})s_c - (\vec{u}\cdot\vec{v})t_c = -\vec{u}\cdot w_0$$

$$(\vec{v}\cdot\vec{u})s_c - (\vec{v}\cdot\vec{v})t_c = -\vec{v}\cdot w_0$$

Fazendo $a = \vec{u}\cdot\vec{u}$, $b = \vec{u}\cdot\vec{v}$, $c = \vec{v}\cdot\vec{v}$, $d = \vec{u}\cdot w_0$ e $e = \vec{v}\cdot w_0$, encontramos

$$s_c = \frac{be - cd}{ac - b^2} \quad \textbf{e} \quad t_c = \frac{ae - bd}{ac - b^2},$$

sempre que o denominador $ac - b^2$ for diferente de zero.

A posição do centro da esfera é igual a $\frac{w_c}{2}$. Os pontos Q_0 e P_0 são as coordenadas das câmeras obtidas pela calibração e os pontos P_1 e Q_1 são os pontos amostrados do centro da esfera.

Quando existem *n* câmeras capturando, o mesmo algoritmo pode ser aplicado para um combinação $\binom{n}{2}$ de segmentos de reta. Com o conjunto de pontos eqüidistantes obtidos, pode-se calcular o centro da esfera através da média da posição dos pontos ou descartando os pontos mais distantes.

O algoritmo descrito acima é uma heurística que tenta aproximar a interseção de quatro retas muito próximas que, supostamente, deveriam se interceptar. Essa não é a única heurística existente. Outras maneiras existem, como por exemplo, a utilização do método dos mínimos quadrados para minimizar a distância entre o ponto de interseção e as quatro retas.

3.5. Filtro de Kalman

Um filtro de Kalman [15,33] tem como idéia básica a predição e a correção de predição de um evento. A predição é feita a partir de valores passados e a correção de predição utiliza medições provenientes da observação de um fenômeno. A Figura 26 ilustra o resultado de uma predição, onde o movimento observado difere do movimento previso.

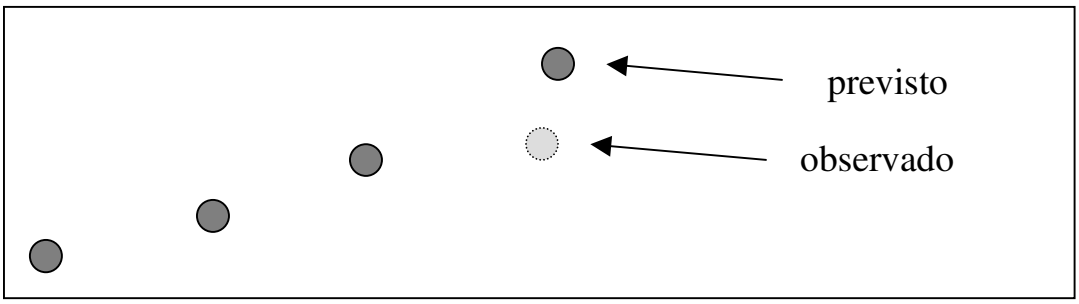


Figura 26: Idéia básica do Filtro de Kalman.

O fenômeno a ser observado e corrigido pelo filtro de Kalman é um processo $x \in \Re^n$ que obedece a uma lei de evolução linear, com a presença de erros aleatórios, ou seja,

$$\underset{n\times 1}{x_k} = \underset{n\times n}{\Phi_k}\ \underset{n\times 1}{x_{k-1}} + \underset{n\times 1}{\varepsilon_k},$$

e uma medição $z \in \Re^m$ descrita por

$$\underset{m\times 1}{z_k} = \underset{m\times n}{H_k}\ \underset{n\times 1}{x_k} + \underset{m\times 1}{\mu_k}, \text{ onde}$$

Φ_k é a matriz de transição do passo anterior *k-1* para o passo *k*, na ausência de ruído no processo,

H_k é a matriz de medição que relaciona o estado x_k à medida z_k e,

$\begin{cases} \varepsilon_k \sim N\left(0, \underset{n\times n}{Q_k}\right) \\ \mu_k \sim N\left(0, \underset{m\times m}{R_k}\right) \end{cases}$ são distribuições normais de probabilidade com média 0 e matriz de covariância Q_k e R_k.

O problema a ser resolvido pelo filtro de Kalman é, dada a estimativa $\hat{x}_{k-1}$ do estado anterior, com sua respectiva matriz de covariância $\hat{P}_{k-1}$, procurar uma solução da forma

$$\hat{x}_k = \tilde{x}_k + K_k\left(z_k - H_k\tilde{x}_k\right), \text{ onde } \tilde{x}_k = \Phi_k\hat{x}_{k-1},$$

de tal modo que o valor esperado do erro quadrático $\left(\mathrm{E}(\hat{x}_k - x_k)^2\right)$ seja mínimo. A matriz K_k é denominada ganho de Kalman.

A solução do filtro de Kalman foi obtida de [33]. Ela é composta de duas etapas: predição e correção. A solução da etapa de predição é

(1) Estimar o processo: $\tilde{x}_k = \Phi_k\hat{x}_{k-1}$

(2) Estimar a matriz de covariância: $\tilde{P}_k = \Phi_k \hat{P}_{k-1} \Phi_k^T + Q_k$

e a da etapa de correção é

(1) Calcular o ganho de Kalman: $K_k = \tilde{P}_k H_k^T (H_k \tilde{P}_k H_k^T + R_k)^{-1}$

(2) Atualizar a estimativa através da medida z_k: $\hat{x}_k = \tilde{x}_k + K_k (z_k - H\tilde{x}_k)$

(3) Atualizar a matriz de covariância: $\hat{P}_k = (I - K_k H_k)\tilde{P}_k$

Dessas fórmulas, o mais importante é concluirmos qual será o comportamento do filtro diante dos valores das matrizes de covariância *Q* e *R*. Para isso devemos analisar o comportamento da matriz *K*. Ela pode ser definida como

$$K = \frac{\Phi_k P_{k-1} \Phi_k^T H_k^T + Q_k H_k^T}{H_k \Phi_k P_{k-1} \Phi_k^T H_k^T + H_k Q_k H_k^T + R_k}$$

Dessa equação concluímos que:

- se $Q >> R$, então $K = {}^{1}\!/_{H}$ e $\hat{x}_k = z_k$, ou seja, se o erro associado ao processo for muito maior do que o erro associado a medição, o filtro de Kalman utilizará os valores medidos como predição;
- se R >> Q, então $K = 0$, $\hat{x}_k = \tilde{x}_k$, ou seja, se o erro da medição for muito maior, o filtro utilizará os valores corrigidos com base no processo.

Observamos também, que o comportamento do filtro depende muito pouco da matriz de covariância inicial P_0.

Na presente tese, o filtro de Kalman foi utilizado para estimar o movimento sendo realizado pela esfera rastreada. Para isso, o modelo empregado foi o seguinte:

$$\begin{bmatrix} X \\ Y \\ Z \\ \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix}_k = \begin{bmatrix} 1 & 0 & 0 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 & dt & 0 \\ 0 & 0 & 1 & 0 & 0 & dt \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix}_{k-1} + \varepsilon \quad \text{e} \quad \begin{bmatrix} Z_X \\ Z_Y \\ Z_z \\ Z_{\dot{X}} \\ Z_{\dot{Y}} \\ Z_{\dot{Z}} \end{bmatrix}_k = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix}_{k-1} + \mu$$

onde (X, Y, Z) é a posição, $(\dot{X}, \dot{Y}, \dot{Z})$ a velocidade e *dt* é a variação em segundos, medido a cada passo. No Capítulo 4, serão dados mais detalhes sobre os valores das matrizes de covariância *Q* e *R*.

3.6. Algoritmo Proposto

O algoritmo proposto é composto de duas etapas: calibração e rastreamento, sendo que elas possuem alguns procedimentos em comum. Ambas as etapas trabalham com a imagem segmentada, logo, as quatro primeiras tarefas são iguais:

a) Captura da imagem pela câmera;
b) Aplicação de filtro gaussiano;
c) Inversão da imagem;
d) Segmentação.

A partir desse momento, a imagem segmentada será utilizada pela etapa de calibração, que é feita uma única vez para cada uma das quatro câmeras utilizadas:

e) Detecção das elipses;
f) Identificação dos pontos de calibração (etapa manual);
g) Calibração da câmera;
h) Gravação dos arquivos de calibração;

Como a etapa de calibração é feita somente uma vez, os arquivos que armazenam as informações das câmeras calibradas serão utilizados posteriormente para recuperação da posição de cada câmera. A etapa de rastreamento possui as seguintes tarefas:

e) Para cada câmera:
    f) Detecção e determinação do centro da esfera sendo rastreada;
    g) Conversão do centro da esfera em coordenadas da imagem para coordenadas do mundo;
    h) Definição do segmento de reta r_i, que passa pela posição da câmera e pelo centro da esfera obtido por essa câmera;
i) Para cada par de segmentos de reta *i* e *j*:
    j) Cálculo do ponto P_{ij} mais próximo entre as retas *i* e *j*;
k) Determinação da média das posições dos pontos P_{ij}.

A média encontrada é o centro da esfera sendo rastreada. Na próxima seção, serão vistos detalhes da implementação e os resultados obtidos.

4
Implementação e Resultados

A dissertação teve dois produtos como resultado da sua implementação. O primeiro foi um programa para calibração das câmeras e o segundo foi um *plugin* para o dispositivo óptico no ViRAL. Além desses dois produtos, foi desenvolvido um *plugin* de cena do ViRAL, a fim de se utilizar o dispositivo óptico para controle de objetos em um grafo de cena.

As ferramentas utilizadas para o desenvolvimento foram o ViRAL, o OpenCV [13], o OpenSceneGraph e a biblioteca Qt da Trolltech [28].

4.1.
Implementação

A Figura 27 ilustra a janela principal do programa de calibração. Nessa interface é possível selecionar qual câmera será calibrada. No caso do nosso programa utilizamos a imagem obtida através de um equipamento que combina o sinal de quatro câmeras, sincroniza-os e envia um único sinal para a placa de captura do computador. Dessa forma, é possível recuperar as imagens das quatro câmeras de uma só vez.

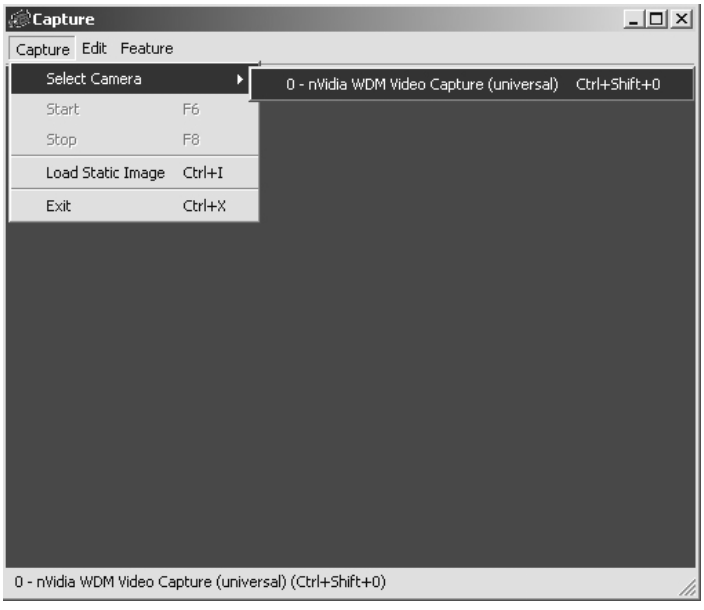


Figura 27: Janela principal do programa de calibração.

Depois de escolhida a câmera e os filtros necessários aplicados, um padrão (Figura 28i) é utilizado para calibração. A (Figura 28ii) ilustra a interface de aplicação de filtros na imagem.

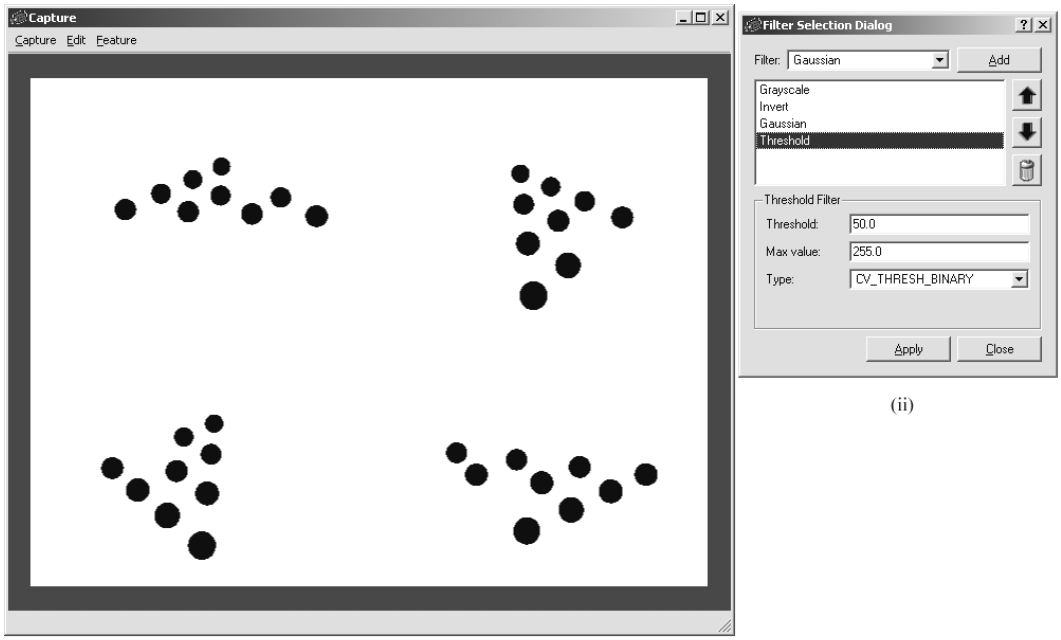


Figura 28: (i) Padrão de calibração; (ii) interface de aplicação de filtros.

O sistema não identifica automaticamente as esferas, ou seja, ele não associa a esfera da imagem à sua correspondente no mundo real. Para isso, o usuário deve manualmente identificá-las, como visto na Figura 29.

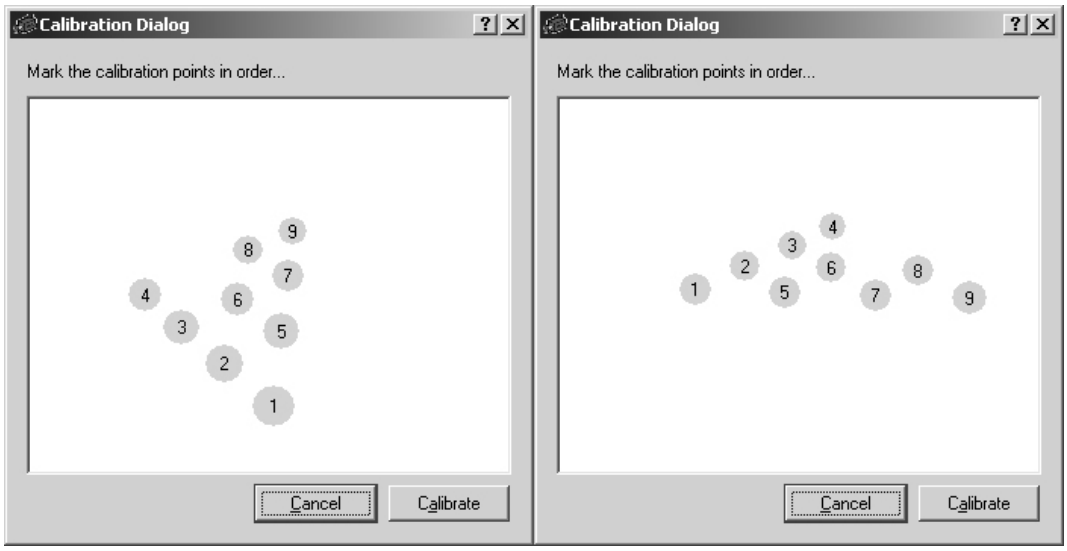
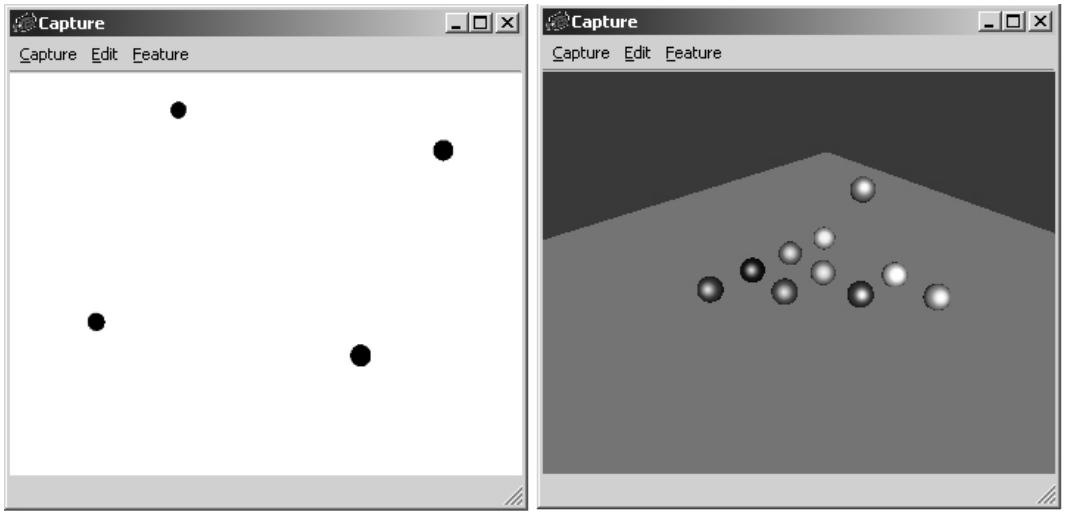


Figura 29: Interface de calibração: identificação das esferas vistas por duas câmeras distintas.

Depois das câmeras estarem calibradas, o padrão de calibração é removido do alcance de visão das câmeras e a esfera retroreflexiva começa a ser rastreada.

A Figura 30i ilustra a imagem dessa esfera e a Figura 30ii apresenta uma esfera virtual posicionada em um modelo tridimensional utilizando a posição obtida através do algoritmo proposto. A esfera em questão está destacada das demais.



(i) (ii)

Figura 30: (i) Imagem capturada da esfera sendo rastreada; (ii) Esfera rastreada inserida em um modelo virtual.

Para fins de testes iniciais, um modelo foi criado em um software de modelagem e, ao invés de capturar a imagem através de quatro câmeras reais, foi utilizada uma imagem combinada de quatro câmeras virtuais gerada pelo próprio programa de modelagem. Com isso, foi possível medir a precisão do algoritmo proposto.

No mundo virtual, a esfera foi posicionada nas coordenadas (12, 8, 6) cm. Através do algoritmo proposto, as posições obtidas utilizando diversas combinações de câmera foram:

Câmeras	**Coordenadas (cm)**	**Erro (distância euclidiana)**
1, 2, 3 e 4	(12,010698; 7,939839; 5,974222)	0,06632 cm
1, 2 e 3	(11,994664; 7,933684; 5,982123)	0,06889 cm
1, 2 e 4	(12,009713; 7,951878; 5,984271)	0,051551 cm
1 e 2	(11,985385; 7,967478; 6,007030)	0,036341 cm
2 e 4	(12,014355; 7,925166; 5,998602)	0,076211 cm
3 e 4	(12,036443; 7,949826; 5,953181)	0,077702 cm

Tabela 8: Erros obtidos utilizando câmeras virtuais.

O algoritmo de calibração foi utilizado, posteriormente, no mundo real e a altura e a distância focal das câmeras estão listadas na Tabela 9.

	Altura (cm)	Distância Focal (cm)
Câmera 1	177,6157	251,237
Câmera 2	182,062	235,780
Câmera 3	161,914	228,041
Câmera 4	229,727	309,485

Tabela 9: Altura e distância focal obtidas pela calibração das câmeras.

As câmeras estão localizadas a aproximadamente 260cm do solo. A diferença entre a altura real e a altura medida é compensada, pelo método de Tsai, na distância focal da câmera.

4.1.1. Integração com o ViRAL – *Plugin* de Dispositivo

O dispositivo óptico foi integrado ao ViRAL através de um *plugin* de dispositivo. A Figura 31 mostra a sua interface de configuração.

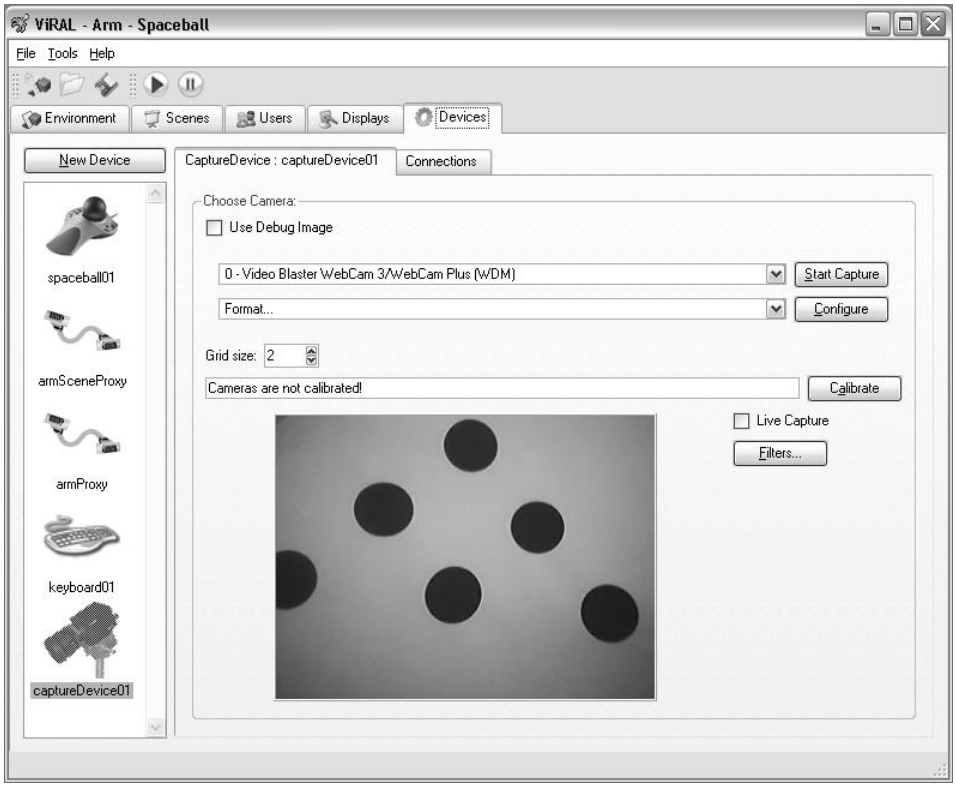


Figura 31: Interface de configuração do *plugin* do dispositivo óptico.

Esse dispositivo exporta dois eventos (*signals*) para o ViRAL: *translate* e *positionChanged* (Tabela 10). O primeiro envia a diferença de movimentação entre dois quadros e o segundo envia a posição absoluta do dispositivo.

```
class CaptureDevice : public vral::Device
{
signals:
  void translate( const Vector3 &translation );
  void positionChanged( const Vector3 &position );

public:
  virtual void dispatchChanges()
  {
    static Vector3 lastTrans;
    Tracker tracker;
    if( tracker.track( x, y, z ) )
    {
       vral::Vector3 translation( x, y, z );
       Vector3 diff = ( translation - lastTrans );
       lastTrans = translation;

       emit translate( diff );
       emit positionChanged( translation );
    }
  }
};
```

Tabela 10: Classe *CaptureDevice*.

A palavra chave *emit* é uma macro do Qt para informar que um sinal (*signal*) está sendo enviado. Os eventos desse dispositivo podem ser conectados aos eventos do *plugin* de cena descrito a seguir, através da interface de conexão de eventos do ViRAL.

O comando *track* da classe *Tracker* é responsável pela execução do algoritmo proposto neste trabalho. A Tabela 11 apresenta o seu pseudo-código:

```
Tracker::track()
{
  Para cada câmera faça
    Determinar a posição P da câmera;
    Encontrar a esfera na imagem da câmera;
    Transformar as coordenadas na imagem em coordenadas C
           do mundo baseado nos dados de calibração;
    Armazenar o segmento de reta PC;
  Para cada par de segmentos de retas
    Armazenar o ponto mais próximo entres eles;
  Calcular e retornar a médias M dos pontos mais próximos;
}
```

Tabela 11: Pseudo-código do método *track* da classe *Tracker*.

A criação de um dispositivo no ViRAL foi bastante simples. A primeira causa disso foi o Qt, pois o mesmo fornece um ambiente integrado de

desenvolvimento que facilita a criação de interface gráfica. Outro ganho do Qt foi quanto à criação de eventos, bastando apenas que fosse definida a palavra-chave *signal* no meio do código para que ele identificasse os eventos exportados.

Além disso, o sistema de *plugins* do ViRAL permitiu que a programação do dispositivo fosse completamente desacoplada do seu núcleo.

4.1.2. *Plugin* de Cena

O *plugin* de cena desenvolvido para o ViRAL é um exemplo simples onde um dispositivo (um *mouse* 3D) controla a posição e a orientação do usuário e o dispositivo óptico movimenta um objeto na cena.

O *plugin* de cena exporta para o ViRAL uma classe de cena *TrackScene* e um objeto de cena *TrackableObject*. A Tabela 12 e a Tabela 13 ilustram os principais métodos implementados nessas classes. Com a utilização do grafo de cena *OpenSceneGraph* foi possível alterar a posição da esfera modificando um objeto *Transformação*.

```
class TrackScene : public Scene
{
public:
  void initializeContext()
  {
    sceneView->setSceneData( mRootNode );
  }
  void preDrawImplementation( const ScenePreDrawingData &data )
  {
    sceneView->update();
  }
  void drawImplementation( const SceneDrawingData &data )
  {
      sceneView->.setProjMatrix( data.getProjection().get() );
      sceneView->setViewMatrix( data.getModelview().get() );

      sceneView->cull();
      sceneView->draw();
  }
private:
  osgUtil::SceneView *mSceneView;
  osg::Node *mRootNode;
}
```

Tabela 12: Classe *TrackScene* de um *plugin* de cena.

```
class TrackableObject : public vral::SceneObject
{
  public slots:
    void translate( const Vector3 &translation )
    {
      osg::Matrix matrix = mTransform->getMatrix();
      matrix = matrix * osg::Matrix::translate(
                                    translation.getX(),
                                   -translation.getZ(),
                                    translation.getY() );
      mTransform->setMatrix( matrix );
    }

    void setPosition( const Vector3 &translation )
    {
      osg::Matrix matrix = mTransform->getMatrix();
      matrix.setTrans( translation.getX,
                      -translation.getZ(),
                       translation.getY() );
      mTransform->setMatrix( matrix );
    }

  private:
    osg::MatrixTransform *mTransform;
};
```

Tabela 13: Classes *TrackableObject* de um *plugin* de cena.

A classe *TrackScene* contém o objeto *SceneView* do *OpenSceneGraph*. Esse objeto é responsável por configurar as matrizes de projeção e chamar os métodos *update/cull/draw*. O objeto *mRootNode* é o nó raiz da cena.

A classe *TrackableObject* contém uma objeto transformação do *OpenSceneGraph*, *mTransform*, que posiciona o objeto sendo rastreado corretamente na cena. A palavra-chave *slots* é um macro do Qt que identifica os eventos de entrada desse objeto. O evento *translate*, como o nome já diz, translada o objeto e o método *setPosition* atribui a ele uma posição absoluta.

A Figura 32, finalmente, ilustra 3 etapas da movimentação de uma esfera com o dispositivo óptico.

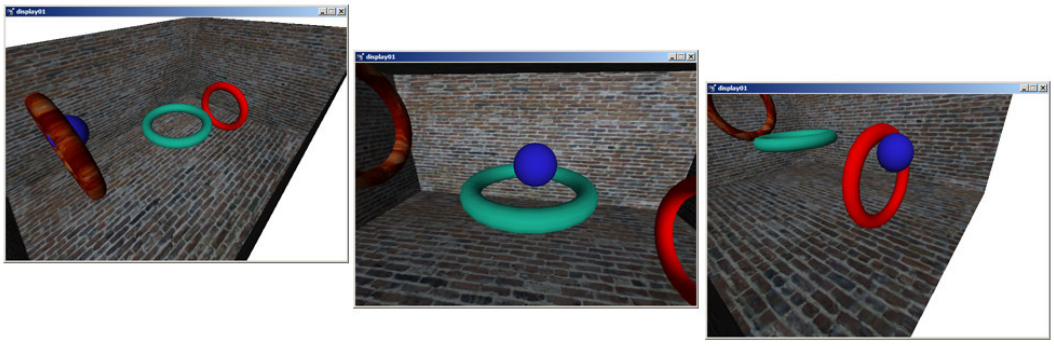


Figura 32: Execução do *plugin* de cena no ViRAL.

4.2. Resultados

A configuração do local de testes está ilustrado na Figura 33. As quatro câmeras foram calibradas e dez testes foram efetuados.

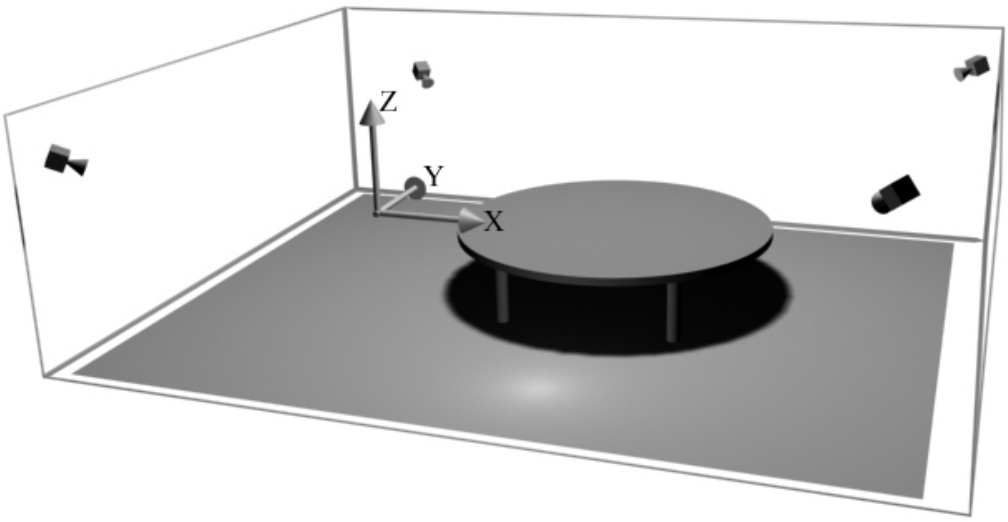


Figura 33: Modelo esquemático do local de testes.

O primeiro teste foi feito com a esfera parada a uma distância de aproximadamente 12cm sobre a mesa, utilizando diversas combinações de câmeras e sem utilização de filtro de Kalman. A calibração prévia das câmeras foi feita com 10 pontos. A posição da esfera foi medida e pode ser vista no gráfico da Figura 34.

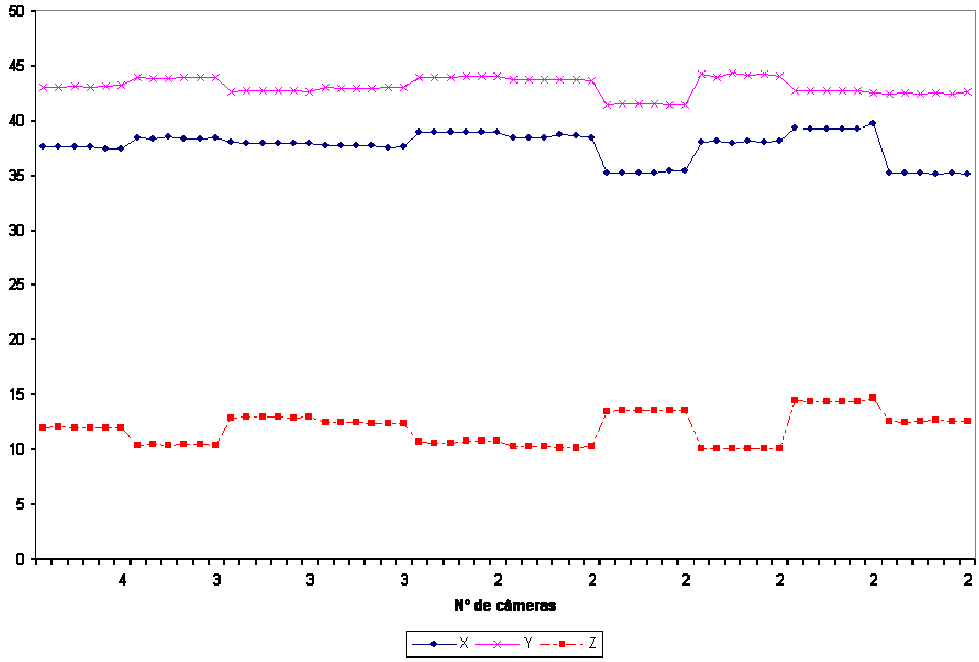


Figura 34: Teste 1 - Esfera em Z = *12* e câmeras calibradas com 10 pontos.

Podemos notar pelo gráfico que quando a configuração de câmeras é mantida, o movimento fica bastante estável. Porém, quando a configuração se modifica, ou seja, uma câmera deixa de localizar o objeto, existe um salto na sua posição. Isso se deve ao fato do fraco resultado da calibração obtida pelo método Tsai.

Uma homografia [26] foi utilizada para se obter mais pontos para calibração das câmeras. No segundo teste, foram usados 121 pontos para calibração, ao invés dos 10 pontos anteriores, e o resultado obtido foi bem melhor (Figura 35).

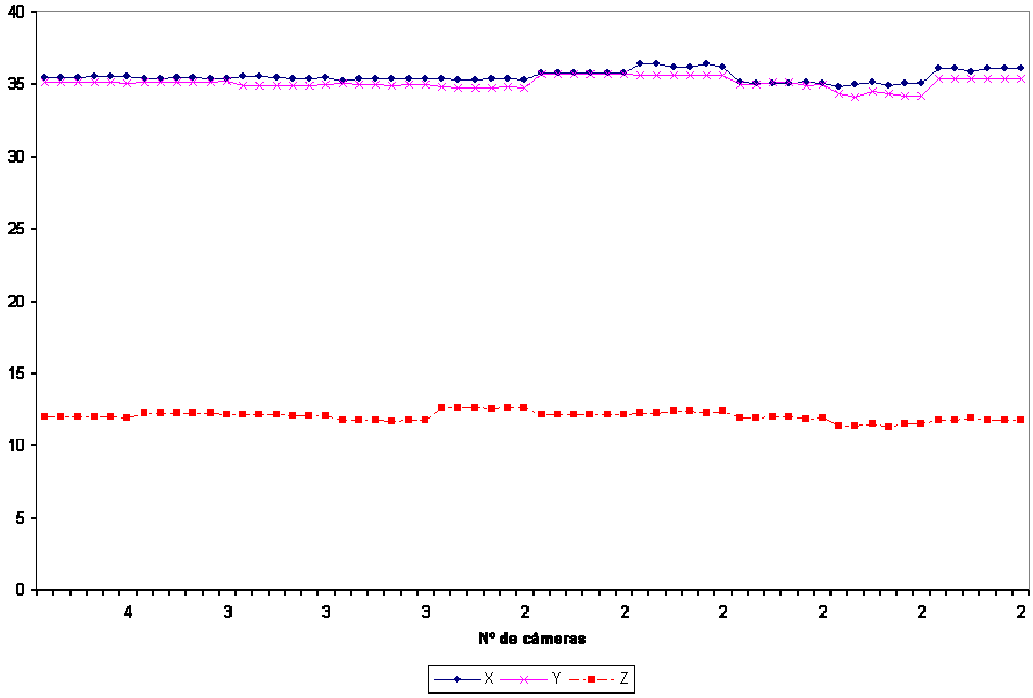


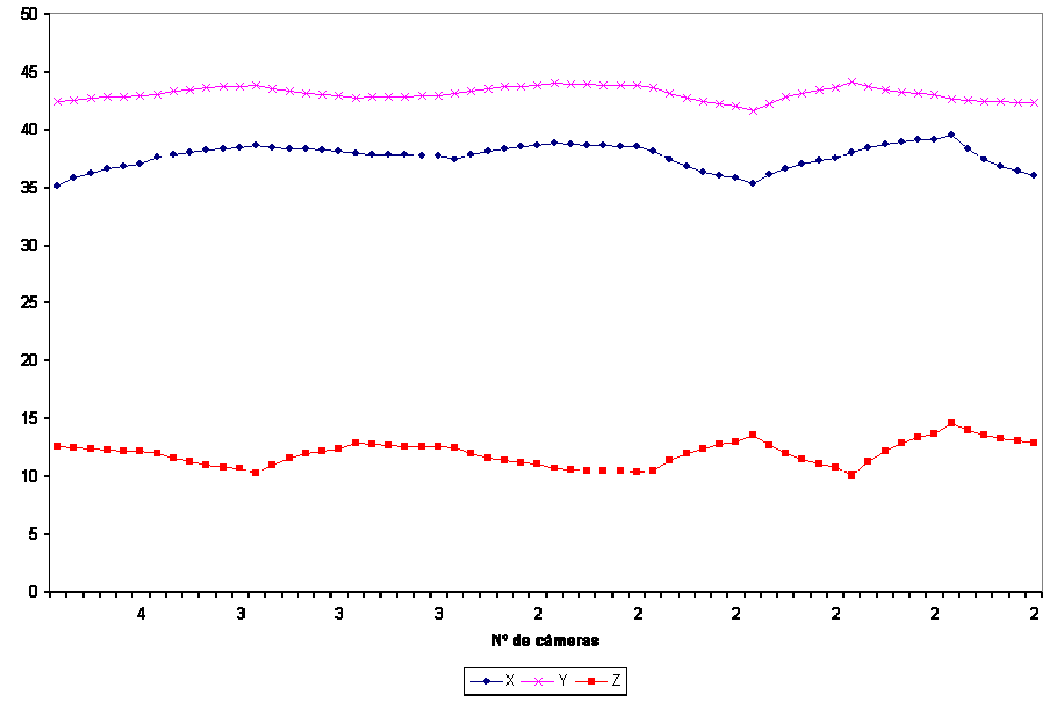
Figura 35: Teste 2 - Esfera em $Z = 12$ e câmeras calibradas com 121 pontos.

As mesmas configurações foram utilizadas para o terceiro e quarto testes, porém o filtro de Kalman foi habilitado. Os valores utilizados nas matrizes de covariância Q, do processo, e R da medida foram:

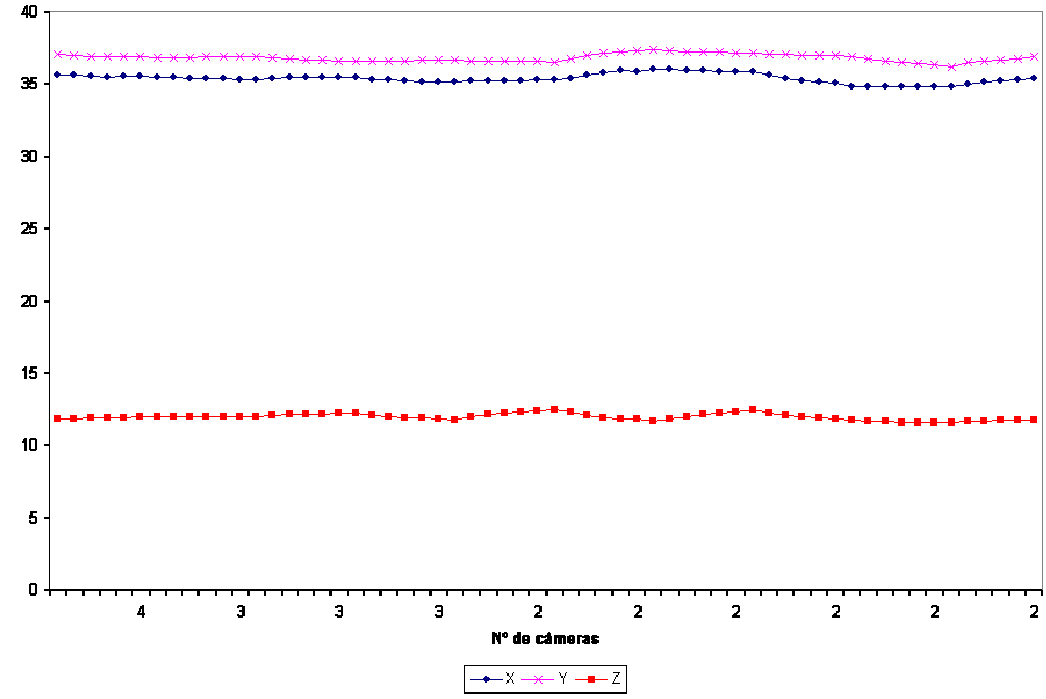
$$Q = \begin{bmatrix} 0{,}01 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0{,}01 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0{,}01 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0{,}1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0{,}1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0{,}1 \end{bmatrix} \text{ e } R = \begin{bmatrix} 0{,}1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0{,}1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0{,}1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0{,}1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0{,}1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0{,}1 \end{bmatrix},$$

onde os primeiros três elementos da diagonal estão relacionados à posição e os três últimos à velocidade. Com esses valores, o filtro tende a suavizar o

movimento quando a posição medida se desvia da sua rota normal, porém mantendo a mesma velocidade da medição. Os resultados destes testes podem ser vistos na Figura 36.



a) Teste 3 - Câmeras calibradas com 10 pontos.



b) Teste 4 - Câmeras calibradas com 121 pontos.

Figura 36: Esfera em *Z = 12* e filtro de Kalman habilitado.

Apesar de ainda existirem saltos quando se modifica a configuração de câmeras, a transição entre uma posição e outra é mais suave, tornando essa diferença mais aceitável. Com 121 pontos de calibração, as variações são praticamente imperceptíveis.

A Tabela 14 apresenta a posição média e o desvio padrão das coordenadas para os testes 1, 2, 3 e 4.

		X	Y	Z
Teste 1	Média	37,64939	43,07366	11,89308
	σ	1,342049	0,791235	1,453631
Teste 2	Média	35,49744	35,06238	11,97544
	σ	0,399178	0,401567	0,338104
Teste 3	Média	37,63999	43,07322	11,91327
	σ	1,022192	0,58011	1,047162
Teste 4	Média	35,35106	36,78472	11,92138
	σ	0,319098	0,259508	0,223056

Tabela 14: Posição média e desvio padrão dos testes 1, 2, 3, 4.

Pelo resultado apresentado, podemos observar uma diferença de cerca de 85% no desvio padrão quando comparamos a cota *Z* do teste 4 em relação ao teste 1. Isso mostra uma fraqueza do método Tsai quando poucos pontos são utilizados.

Os quinto e sexto testes foram feitos repetindo-se os procedimentos dos testes 1 e 2, porém a distância da esfera ao plano *Z* da mesa foi aumentada para 55cm. O resultado do quinto teste está na Figura 37 e o do sexto na Figura 38.

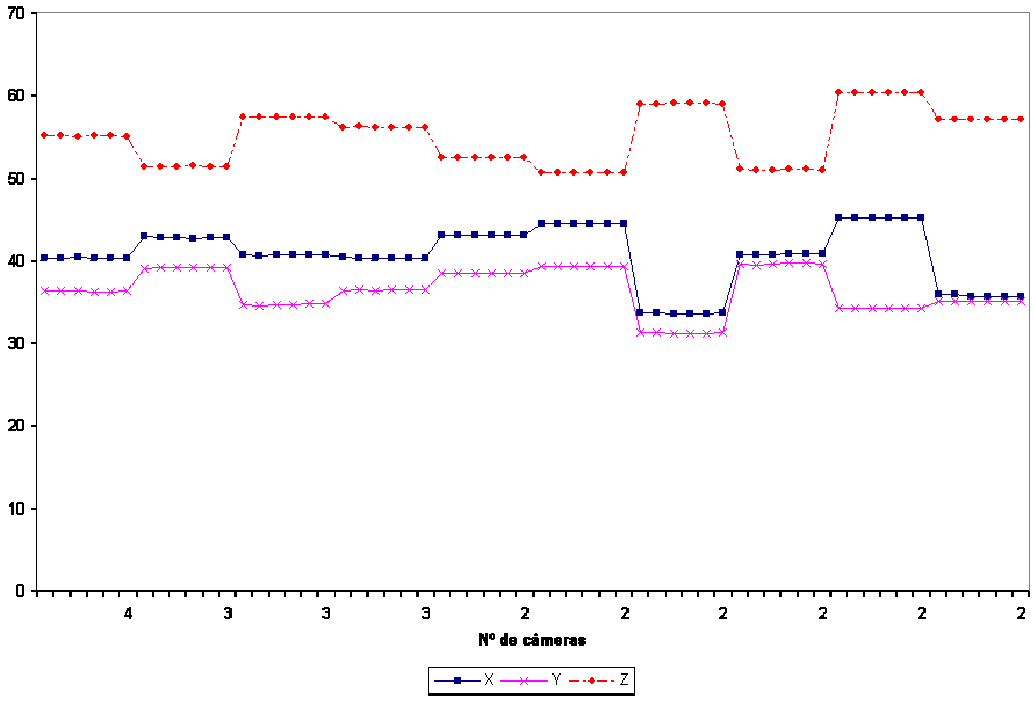


Figura 37: Teste 5 - Esfera em Z = *55* e câmeras calibradas com 10 pontos.

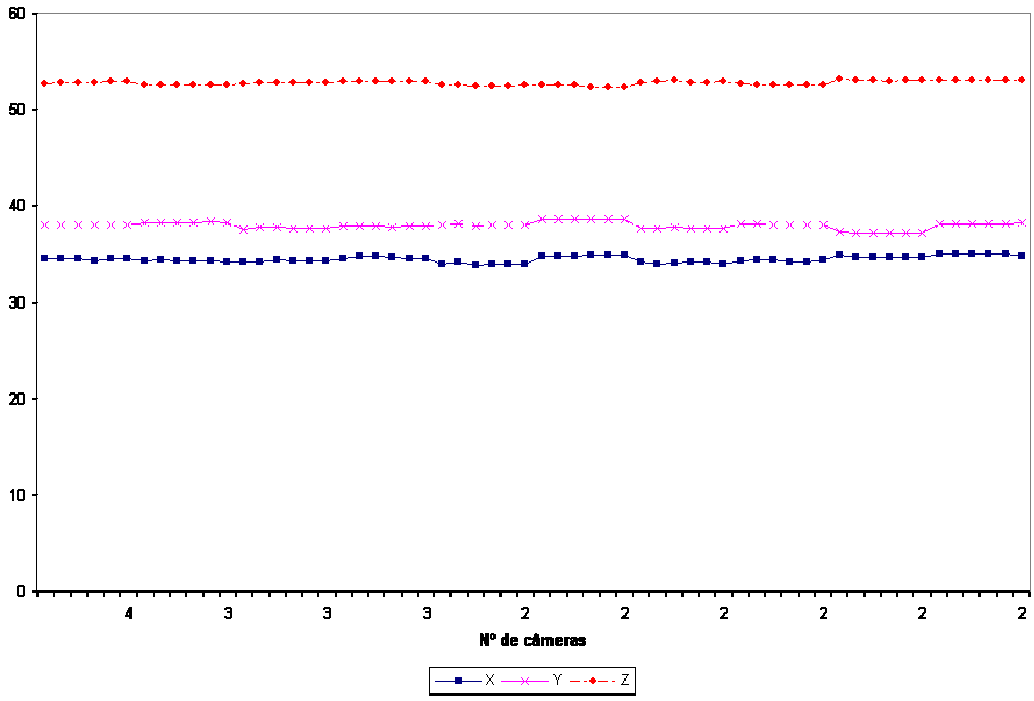
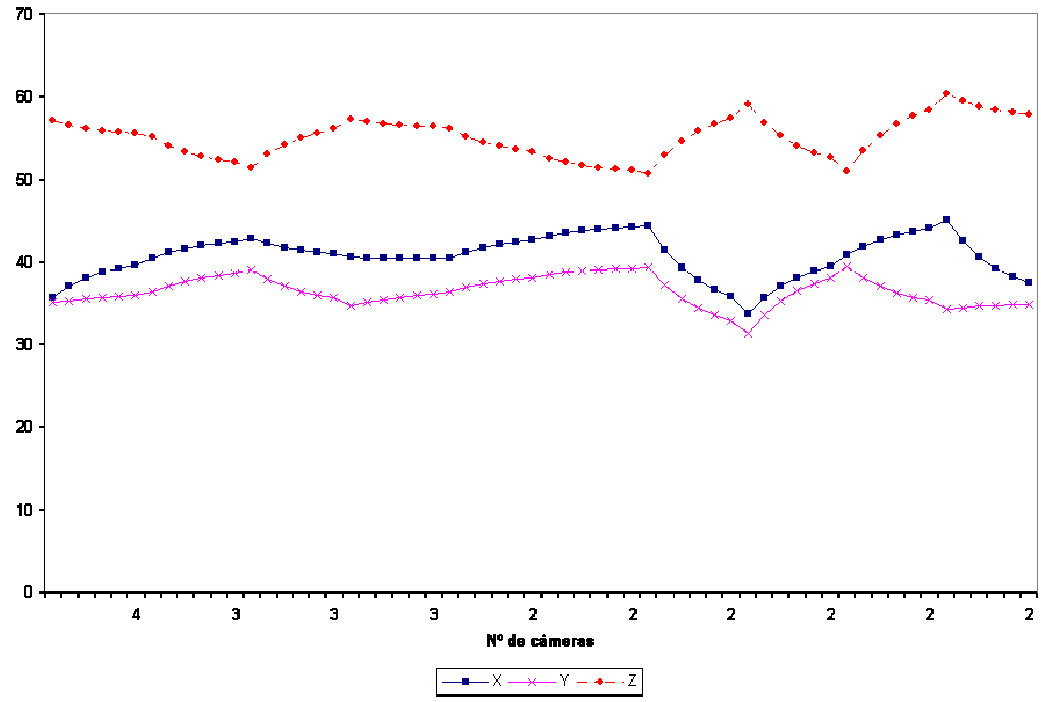


Figura 38: Teste 6 - Esfera em Z = *55* e câmeras calibradas com 121 pontos.

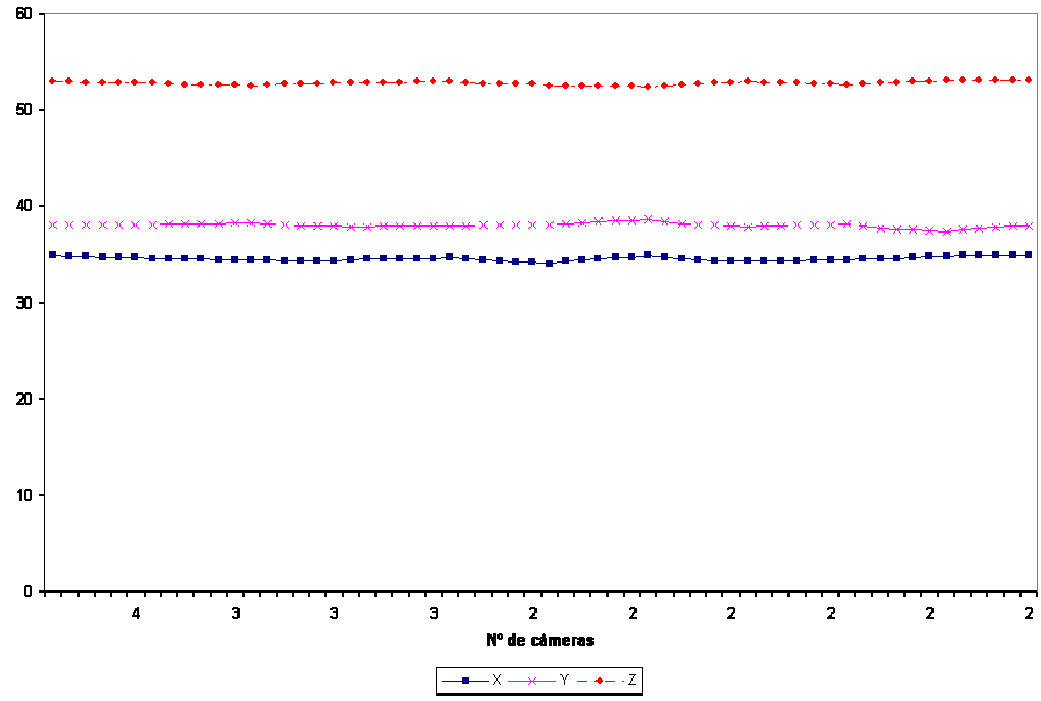
O comportamento foi bastante similar ao resultado anterior, porém, pelo teste 5, notamos que, quanto mais a esfera se afasta do plano de calibração, maior o erro quando se reduz o número de câmeras rastreando-a. Isso se deve ao fato do algoritmo de calibração ser um algoritmo coplanar. O erro pode ser reduzido

aumentando o número de pontos de calibração (verificado no resultado do teste 6) e através da utilização de um algoritmo tridimensional de calibração.

Os testes sete e oito foram realizados com a esfera à 55cm da mesa, porém com filtro de Kalman habilitado. Os resultados são vistos na Figura 39.



b) Teste 7 – Câmeras calibradas com 10 pontos.



b) Teste 8 – Câmeras calibradas com 121 pontos.

Figura 39: Esfera em Z = *55* e filtro de Kalman habilitado.

Da mesma forma, o filtro de Kalman suavizou o movimento quando a configuração de câmeras foi modificada. A Tabela 15 apresenta a posição média e o desvio padrão das coordenadas para os testes 5, 6, 7 e 8.

		X	Y	Z
Teste 5	Média	40,66364	36,43447	55,04712
	σ	3,487908	2,587444	3,358542
Teste 6	Média	34,4313	37,95047	52,74441
	σ	0,330229	0,377904	0,215029
Teste 7	Média	40,66211	36,4109	55,03846
	σ	2,534221	1,779225	2,411372
Teste 8	Média	34,49527	37,96208	52,71731
	σ	0,216536	0,241335	0,181227

Tabela 15: Posição média e desvio padrão dos testes 5, 6, 7, 8.

Notadamente, em todos os testes, o desvio padrão foi muito menor quando as câmeras foram calibradas com 121 pontos. A melhoria do desvio padrão da cota *Z* do teste 8 em relação ao teste 5 foi de 94,6%.

Os dois últimos testes foram feitos utilizando as quatro câmeras e movimentando a esfera na direção *-X* e depois *Y*, como ilustra a Figura 40.

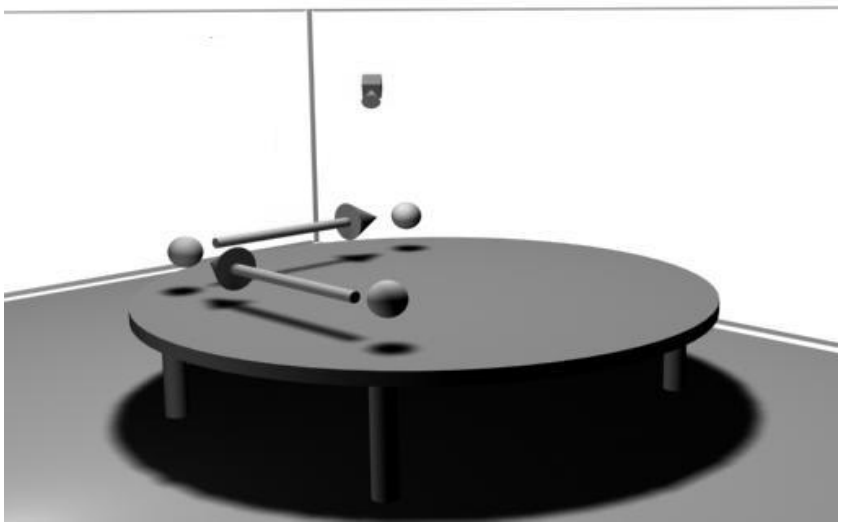


Figura 40: Esquemático do movimento da esfera.

O resultado do penúltimo deles, sem filtro de Kalman, pode ser visto na Figura 41 e o último, com filtro de Kalman habilitado, na Figura 42.

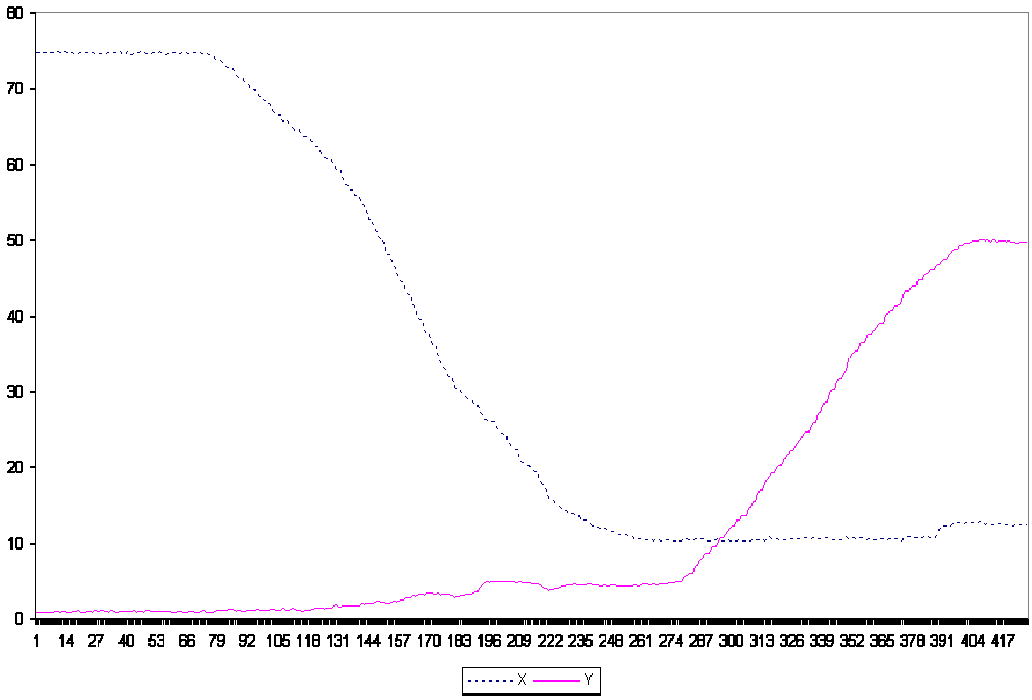


Figura 41: Esfera se movimentando em *X* e depois *Y*.

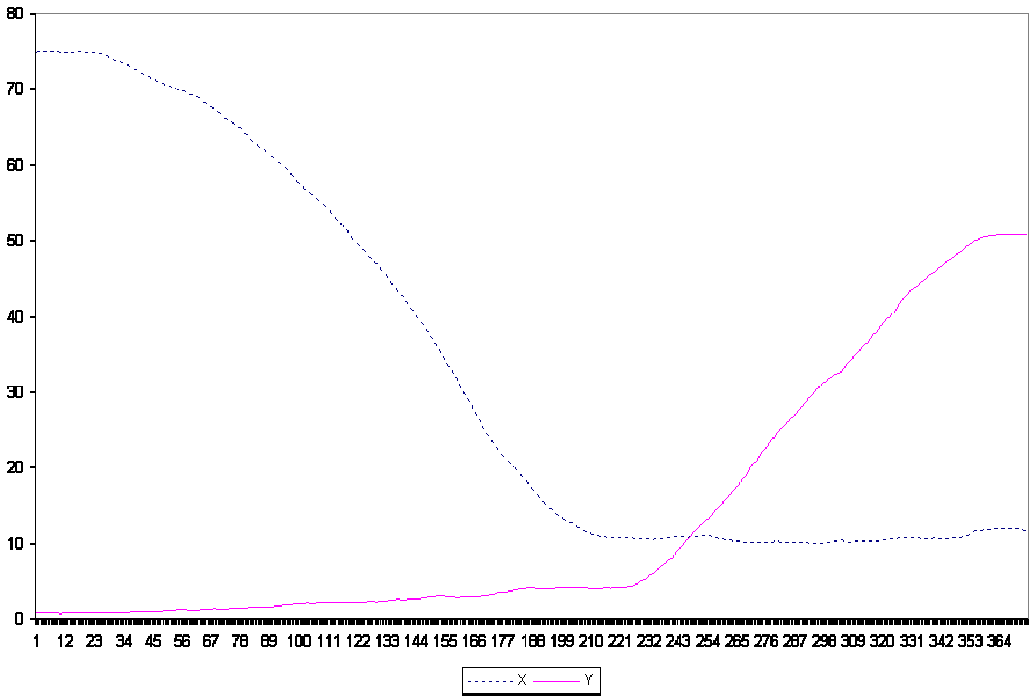


Figura 42: Esfera se movimentando em *X* e depois *Y* com filtro de Kalman habilitado.

Os gráficos ilustram o movimento esperado sendo que, quando o filtro de Kalman estava habilitado, o movimento descreveu uma trajetória mais suave.

5
Conclusão

O primeiro resultado deste trabalho foi a criação de um dispositivo óptico utilizando equipamentos de baixo custo, como câmeras e placas de captura. Pelos resultados obtidos, podemos concluir que o sistema foi bastante preciso ao determinar a posição da esfera, principalmente na ocorrência de oclusão de uma das câmeras. Isso permite que o usuário se movimente sem restrições pelo ambiente de RV.

A precisão alcançada pelo sistema pode ser melhorada utilizando-se câmeras de melhor qualidade e maior resolução. Neste trabalho foram utilizadas câmeras analógicas com baixa resolução.

Um problema encontrado no uso dessas câmeras foi o fato das mesmas deixarem "rastro" na imagem. Isso acontecia quando a esfera movia-se muito rapidamente. Esse comportamento é indesejável, pois o algoritmo proposto não prevê a existência de mais de uma esfera na imagem.

É possível melhorar também a precisão aumentando o número de pontos e a área de calibração das câmeras, usando, eventualmente, um modelo de calibração tri-dimensional.

O segundo resultado foi a integração deste dispositivo no ViRAL e a sua utilização para controle de uma cena. O dispositivo e a cena foram desenvolvidos com *plugins* para o ViRAL, permitindo o seu uso em futuras aplicações de RV.

O desenvolvimento no ViRAL foi bastante desacoplado do núcleo da biblioteca, permitindo a utilizações de outras ferramentas, como o OpenCV e o OpenSceneGraph, sem que isso acarretasse um conflito de compilação.

Uma limitação desse sistema é a necessidade de se utilizar duas ou mais câmeras com sensores infravermelhos. Essas câmeras não funcionam quando colocadas em ambiente com luz solar, tornando seu uso exclusivo em salas fechadas. Outra limitação é quanto ao número de graus de liberdade, pois esse dispositivo fornece apenas 3 graus, enquanto que em uma aplicação de RV, geralmente são utilizados 6 graus.

5.1. Trabalhos Futuros

O trabalho que é a continuação mais natural deste é a construção de um dispositivo óptico que forneça seis graus de liberdade, ao invés de apenas três, permitindo a detecção da rotação do dispositivo e não apenas sua translação no espaço.

Para eliminar a utilização de sensores infravermelhos, um algoritmo de detecção de círculos em imagem poderia ser utilizado para rastrear a esfera, como, por exemplo, a transformada de Hough [9]. Outra abordagem interessante é a detecção de padrões quaisquer. Dessa forma, seria possível rastrear um objeto que não seja esférico.

A fim de melhorar a precisão do algoritmo quando o objeto se afasta do plano de calibração, um modelo tridimensional poderia ser utilizado na calibração das câmeras.

O algoritmo proposto não faz nenhum tipo de previsão do movimento, a não ser através do filtro de Kalman. Esse tipo de técnica pode ser interessante para descartar ruídos que apareçam na imagem e se confundam com a esfera sendo rastreada.

6
Referências Bibliográficas

1 ARSENAULT, L., KELSO, J., KRIZ, R., DAS-NEVES, F. **DIVERSE: a Software Toolkit to Integrate Distributed Simulations with Heterogeneous Virtual Environments**. Technical Report TR-01-10, Computer Science, Virginia Tech, 2001.

2 AUKSTAKALNIS, S., BLATNER, D. **Silicon Mirage: The Art and Science of Virtual Reality**. Editora Peatchpit Press, Berkeley, CA, 1992.

3 AZUMA, R., BISHOP. G. **Improving Static and Dynamic Registration in a See-Through HMD**. Proceedings of SIGGRAPH, In Computer Graphics, Annual Conference Series, pp. 197-204, 1994.

4 AZUMA, R. T. **A Survey of Augmented Reality**. In Presence: Teleoperators and Virtual Environments, pp. 355-385, Agosto, 1997.

5 BIERBAUM, A. **VR Juggler: A Virtual Platform for Virtual Reality Application Development**. Master Thesis, Iowa State University, 2000.

6 BURDEA, G., COIFFET, P. **Virtual Reality Technology**. Editora John Wiley & Sons, Nova Iorque, 1994.

7 CRUZ-NEIRA, C., SANDIN, D. J., DEFANTI, T. A. **Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE**. ACM Computer Graphics, vol. 27, número 2 , pp 135-142, Julho, 1993.

8 DALY, J., KLINE, B., MARTIN, G. **A. VESS: Coordinating Graphics, Audio, and User Interaction in Virtual Reality Applications**. IEEE Virtual Reality, Março, 2002.

9 DUDA, R.O., HART, P.E. **Use of the Hough transformation to detect lines and curves in pictures**. Communications of the ACM, Vol. 15, Issue 1, pp. 11-15, Janeiro, 1972.

10 FERREIRA, A. G. **Uma Arquitetura para a Visualização Distribuída de Ambientes Virtuais**. Dissertação de Mestrado, PUC/RJ, 1999.

11 GAMMA, E., HELM, R., JOHNSON, R., VLISSIDES, J. **Design Patterns - elements of reusable object-oriented software**. Addison-Wesley Longman, Inc., 1995.

12 GONZALEZ, R. C., WOODS, R. E. **Processamento de Imagens Digitais**. Editora Edgard Blucher, ISBN 8521202644, São Paulo, 2000.

13 **Intel Open Source Computer Vision Library**. Intel Research. http://www.intel.com/research/mrl/research/opencv (Janeiro 2004).

14 JACOBSON, L. **Virtual Reality: A Status Report**. AI Expert, pp. 26-33, Agosto, 1993.

15 KALMAN, R. E., **A New Approach to Linear Filtering and Prediction Problems**. Transactions of the ASME--Journal of Basic Engineering, Vol. 82, Series D, pp. 35-45, 1960.

16 KRUEGER, M. W. **Artificial Reality II**. Editora Addison-Wesley, Massachussets, 1991.

17 **Manual of Photogrammetry**. 4ª. Edição, Amer. Soc. of Photogrammetry, 1980.

18 MORIE, J. F. **Inspiring the Future: Merging Mass Communication, Art, Entertainment and Virtual environments**. ACM SIGGRAPH Computer Graphics, 28(2), pp. 135-138, Maio 1994.

19 **OpenGL® - The Industry Standard for High Performance Graphics** http://www.opengl.org (Janeiro 2004).

20 **OpenGL Performer Getting Started Guide**. SGI Document Number 007-3560-004, Janeiro, 2004.

21 **OpenSceneGraph - The High Peformance Open Source Graphics Toolkit** http:// www.openscenegraph.org (Janeiro 2004).

22 RUSS, J. C. **The Image Processing Handbook**. CRC Press, 2ª edição, 1995.

23 SHERMAN, W. R., CRAIG, A. B. **Understanding Virtual Reality: Interface, Application, and Design**. Editora Morgan Kaufmann, 2003.

24 STEFANI, O., HOFFMANN, H., RAUSCHENBACH, J. **Design of Interaction Devices for Optical Tracking in Immersive Environments**. In: Human-Centred Computing: Cognitive, Social and Ergonomic Aspects, Volume 3 of the Proceedings of HCI International, Junho 2003.

25 STURMAN, D. **A Brief History of Motion Capture for Computer Character Animation**. Character Motion Systems, SIGGRAPH Course 9, 1994.

26 SZENBERG, F. **Acompanhamento de Cenas com Calibração Automática de Câmeras**. Tese de Doutorado, DI, PUC/RJ, 2001.

27 TRAMBEREND, H. **Avango: A Distributed Virtual Reality Framework**. Proceedings of Afrigraph '01, ACM, 2001.

28 **Trolltech - Creators of Qt - The multi-platform C++ GUI/API**. http://www.trolltech.org/ (Fevereiro 2004).

29 TSAI, R.Y. **An Efficient and Accurate Camera Calibration Technique for 3D Machine Vision**. CVPR'86 Proceeding, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, FL, Junho 22-26, pp. 364-373, 1986.

30 TSAI, R.Y. **A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses**. IEEE Journal of Robotics and Automation, Vol. RA-3, NO. 4, pp 323-344, Agosto 1987.

31 VAN LIERE, R., MULDER, J. D. **Optical tracking using projective invariant marker pattern properties**. IEEE Virtual Reality, 2003.

32 WALSH, A. E. **Understanding Scene Graphs**. Dr Dobb's Journal, 27:7, 17-26, 2002.

33 WELCH, G., BISHOP, G. **An Introduction to the Kalman Filter**. Course 8, SIGGRAPH, 2001.