Uma Arquitetura de Software para Bibliotecas de Realidade Aumentada Baseada em Marcadores Naturais

A Software Architecture for Markerless Augmented Reality Libraries

Douglas C. B. Oliveira [1]
Lidiane T. Pereira [1]
Rodrigo L. S. Silva [1]

Data de submissão: 18/11/2016, Data de aceite: 28/05/2017

Resumo: Um importante tópico de estudo no campo da Realidade Aumentada é o desenvolvimento de sistemas que não necessitam de marcadores fiduciais. Nesses sistemas são utilizadas somente características naturais da cena para posicionar a câmera e realizar a projeção. Essa característica torna as aplicações mais inteligíveis e sua utilização é imediata. Entretanto, não existem muitas bibliotecas que auxiliem o desenvolvimento de aplicações em *Markerless Augmented Reality* (MAR) e, as poucas que existem, geralmente são de código fechado. Este artigo apresenta uma arquitetura de software que pode ser utilizada para a criação de bibliotecas de MAR. Todos os recursos de Visão Computacional serão oriundos da biblioteca OpenCV.

Abstract: An important topic of study in the field of Augmented Reality is the development of systems that do not require the use of fiducial markers in the scene. In these systems only natural features of the scene are used to position the camera and then performing the projection. This characteristic makes the applications more intelligible and its utilization is immediate. However, there are not many libraries available that supports the Markerless Augmented Reality (MAR) application development and the few that exist are commonly not open source. This paper presents a software architecture that can be used for the development of MAR libraries. The Computer Vision resources will be native of OpenCV open source library.

***P*alavras-chave:** Arquitetura de Software; Realidade Aumentada; Características Naturais.

[1]Departamento de Ciência da Computação – Instituto de Ciências Exatas
Universidade Federal de Juiz de Fora (UFJF)
Rua José Kelmer, s/n, Campus Universitário, Bairro São Pedro
CEP: 36036-900 – Juiz de Fora/MG – Brasil
{dcoelhobo,lidianetpereira26,rodrigoluis}@gmail.com

Keywords: Software Architecture; Markerless Augmented Reality; Natural Features.

1 Introdução

Realidade Aumentada (RA) é uma tecnologia que torna possível a sobreposição de objetos virtuais tridimensionais, gerados por computador, em um ambiente predominantemente real. Nesse novo ambiente "aumentado" de informações, tanto os objetos físicos quanto os virtuais podem interagir entre si e com o meio no qual estão inseridos. Um sistema de RA deve ser capaz de criar e manter tal ambiente, processando as interações em tempo real. Um dos principais métodos que tornam a RA possível é a estimativa de pose, que consiste em computar a posição e orientação do observador em relação a algum referencial global. Geralmente isso é feito a partir de um sistema de rastreamento óptico que busca objetos reais previamente conhecidos em cada quadro do vídeo e, através de técnicas de Visão Computacional, computam a pose do observador em relação a esse objeto, que por essa razão são denominados **marcadores**. Esse método é bastante eficiente e financeiramente barato, porém, dependendo da complexidade do objeto, o custo computacional pode ser alto.

O avanço crescente de *hardware* e *software* ocorrido nos últimos anos contribuiu com a área em relação a essa exigência de desempenho. Há poucos anos atrás os marcadores deviam ser artificiais, com características bem consistentes e precisas para acelerar seu rastreamento [1, 2, 3]. Hoje é computacionalmente viável rastrear objetos naturalmente presentes no ambiente, como fotografias, capas de livros e revistas [4, 5, 6] ou mesmo objetos tridimensionais [7, 8, 9]. Esse ramo da RA, que trata dos marcadores naturais, é conhecido como *Markerless Augmented Reality* (MAR).

As aplicações são diversas e atualmente a área tem se popularizado, deixando de ser exclusividade do meio acadêmico e alcançando o mercado. A indústria da publicidade por exemplo, vem utilizando essa tecnologia para criar propagandas mais agradáveis e interativas. O aplicativo móvel **Zappar**[2] possui o apoio de grandes empresas e já promoveu diversos produtos como filmes, bebidas, jogos, entre outros. No meio acadêmico, a área de realidade aumentada em geral é profícua na criação de ferramentas de apoio ao ensino, como sugere o estudo realizado em [10].

Contudo o desenvolvimento de aplicações em MAR ainda está limitado por não haver muitas ferramentas, como bibliotecas e *frameworks*, que auxiliem isso. As poucas bibliotecas que existem geralmente são distribuídas por grandes empresas que comercializam esse tipo de *software*. Logo, não há interesse em disponibilizar o código-fonte. Uma exceção é a clássica e popular biblioteca ARToolKit[3] desenvolvida originalmente por Hirokazu Kato e

[2] http://www.zappar.com/ [Online; acessado em 04/05/2017]
[3] https://artoolkit.org/ [Online; acessado em 04/05/2017]

Mark Billinghurst [1] e especializada em detecção de marcadores fiduciais. Porém, após vários anos sem atualização, os direitos sobre a biblioteca foram adquiridos pela companhia DAQRI[4], que recentemente relançou o *software*, agora contando também com recursos de rastreamento de marcadores naturais.

A nova biblioteca ARToolKit mantém importantes características da original, ela é *open source* e multiplataforma. Entretanto, ela também manteve um aspecto não muito agradável para desenvolvedores, que é o baixo nível do seu código, dificultando o aprendizado. É importante para o crescimento da área que haja mais ferramentas de suporte ao desenvolvimento de aplicações em MAR que possuam uma melhor curva de aprendizagem.

A principal contribuição do presente trabalho é fornecer uma arquitetura de *software* para bibliotecas de *Markerless Augmented Reality*. Dentro deste contexto, este trabalho tem como objetivo investigar quais atributos seriam necessários em uma arquitetura de software para o desenvolvimento deste tipo de biblioteca. Considerando as pesquisas realizadas até o presente momento, este é o primeiro trabalho que descreve em detalhe uma arquitetura completa desta natureza. A arquitetura proposta é independente de plataforma, ambiente e linguagem, sendo somente recomendado o uso de linguagens orientadas a objetos para o seu desenvolvimento, devido ao maior nível de abstração que esse paradigma possui. A fim de agilizar o desenvolvimento do sistema e também abstrair os problemas de implementação, todos os recursos de visão computacional necessários serão oriundos da biblioteca OpenCV[5]. Esta é uma biblioteca *open source*, multiplataforma, disponível em várias linguagens, focada em algoritmos de visão computacional e projetada para aplicações em tempo real. A versão de referência do OpenCV utilizada neste trabalho é a 2.4.10.

Este trabalho está organizado da seguinte forma: A Seção 2 apresenta os trabalhos relacionados. Na Seção 3 é apresentado e detalhado o modelo proposto. Na Seção 4 é apresentada uma prova de conceito a fim de validar o modelo. Por fim, a Seção 5 conclui o trabalho e propõe recursos extras para as potenciais bibliotecas desenvolvidas a partir da arquitetura proposta.

2 Trabalhos Relacionados

Esta seção descreve os trabalhos que mais se aproximam da proposta deste artigo. Entretanto, conforme levantamento bibliográfico realizado, nenhum dos trabalhos pesquisados possui todos os atributos necessários para o desenvolvimento de uma arquitetura de bibliotecas para MAR. Não foi encontrado na literatura nenhum *framework* que permita ao

[4]https://daqri.com/ [Online; acessado em 04/05/2017]
[5]http://opencv.org/ [Online; acessado em 04/05/2017]

desenvolvedor criar sua própria biblioteca, seja com objetivo acadêmico ou comercial.

A biblioteca **Vuforia**[6], produzida e distribuída pela empresa **Qualcomm** é um dos principais *frameworks* modernos para o desenvolvimento de aplicações de RA, possuindo módulos para rastreamento de diversos objetos, entre eles o de marcadores naturais. Ela possui vasta documentação e está disponível para diversas plataformas. Porém, por se tratar de um *software* proprietário, o código é fechado e necessita de licença para desenvolver aplicações comerciais. Como dito anteriormente, a biblioteca ARToolKit [1] é originalmente baseada em marcadores fiduciais, mas recentemente também permite desenvolver aplicações em MAR. Porém, o baixo nível do código e a documentação incompleta dificultam o aprendizado. Esse baixo nível do ARToolKit pode ser conferido em um dos exemplos mais simples da própria biblioteca, chamado *NFT Simple*[7]. Esse exemplo possui cerca de 400 linhas de códigos relevantes para o registro do marcador natural e projeção de um cubo virtual sobre ele. Outro trabalho que também introduz o rastreamento de marcadores naturais a uma biblioteca baseada em marcadores fiduciais é proposto em [11]. Nesse trabalho os autores inseriram o novo rastreador em uma biblioteca de RA voltada para dispositivos móveis, especificamente com sistema operacional Android. Os autores utilizaram uma abordagem híbrida semelhante a que é proposta aqui.

No contexto dos marcadores fiduciais, em [12] foi desenvolvida uma biblioteca, nomeada AVRLib, que utiliza as rotinas de visão computacional do ARToolKit original. Nesse trabalho os autores apresentaram o diagrama de classes da camada orientada a objetos desenvolvida. Os sistemas baseados em marcadores fiduciais possuem um grande problema em relação à visibilidade. Oclusões mínimas dos padrões podem acarretar falhas ou perda total do registro da cena. Por esse motivo é comum esses sistemas usarem uma configuração com múltiplos marcadores responsável por uma só projeção [2, 3]. Os sistemas baseados em características naturais da cena não sofrem tanto com esse problema, pois, como a referência consiste de vários pontos de interesse, tornam-se resistentes a oclusões parciais no conjunto de pontos. Deste modo, descarta-se a necessidade de se criar configurações de múltiplos marcadores naturais com relações entre si.

Em termos de arquitetura de *software*, em [13] foi apresentada uma arquitetura orientada a objetos baseada em arquivos XML (e**X**tended **M**arkeup **L**anguage) nomeada *OpenTracker*, que é capaz de combinar diferentes sistemas de rastreamento, como o óptico e o magnético, permitindo interação entre os objetos detectados pelos dois sistemas. Em [14] foi proposta uma arquitetura para distribuição de ambientes de Realidade Aumentada, como sendo uma ferramenta de apoio a projetos de ensino. O sistema utiliza a biblioteca ARToolKit para construção das interfaces de RA, que são responsáveis pela visualização dos objetos virtuais e pela interação com os mesmos. Cada interface possui uma aplicação-cliente que envia

[6]https://library.vuforia.com/ [Online; acessado em 04/05/2017]
[7]https://github.com/artoolkit/artoolkit5/blob/master/examples/nftSimple/nftSimple.c [On; acessado em 08/05/2017]

os dados referentes ao posicionamento dos objetos a um servidor que, por sua vez, distribui essas informações às demais aplicações-clientes conectados à rede. Estes trabalhos servem como exemplos de características que podem ser incorporadas ao modelo aqui proposto.

3 Arquitetura Proposta

Nesta seção será detalhada a arquitetura de *software* proposta. O modelo que aqui será descrito é embasado no sistema para MAR proposto em [4]. Como a extração de características de uma imagem é um processo custoso, o trabalho citado propõe a implementação de um sistema híbrido. A ideia é evitar que novas características sejam extraídas a cada quadro do vídeo. Assim, o sistema atua sobre dois estados: modo *lost* e modo *tracking*.

Com o sistema em modo *lost*, as características são extraídas usando algum algoritmo extrator, neste caso usou-se o ORB (***O**riented FAST and **R**otated **BRIEF***) [15], cuja principal característica é fazer um balanceamento entre qualidade e performance. Outros extratores famosos como o SIFT (***S**cale-**I**nvariant **F**eature **T**ransform*) [16] e o SURF (***S**peeded **U**p **R**obust **F**eatures*) [17] obtêm uma qualidade superior, mas deixam muito a desejar no desempenho, além de serem patenteados. O sistema em modo *lost* ainda é modificado para a estimativa de pose da câmera, onde utilizou-se o algoritmo EPnP (***E**fficient **P**erspective-**n**-**P**oint*) [18] junto ao laço do RANSAC (***RAN**dom **SA**mple **C**onsensus*) [19].

O modo *tracking* do sistema, por sua vez, utiliza o algoritmo de fluxo óptico para rastrear as características extraídas de um quadro anterior no quadro corrente. Na estimativa de pose da câmera utilizou-se o algoritmo iterativo de mínimos quadrados de Levenberg-Marquardt [20].

O Apêndice A traz o diagrama de classes do sistema proposto. Este não é um diagrama completo, somente os atributos, métodos e relações essenciais para o sistema foram modelados. A proposta do trabalho é dar um ponto de partida para o desenvolvimento de bibliotecas que podem e devem possuir características distintas entre si. Logo, o modelo aqui apresentado contém apenas a estrutura básica necessária e deve ser complementado. Esse diagrama foi projetado para ser o mais genérico possível, não considerando a plataforma e linguagem a serem utilizadas no desenvolvimento da biblioteca. Entretanto, foram utilizadas algumas notações da linguagem C++ na representação dos identificadores, como a de *namespace* para enfatizar o uso da biblioteca OpenCV.

Nas subseções seguintes o modelo será descrito em detalhe. Classes, métodos e funções do OpenCV necessárias ao seu desenvolvimento serão citadas, mas seus respectivos funcionamentos não serão explicitados por estarem fora do escopo deste trabalho. Maiores detalhes a respeito dos recursos do OpenCV utilizados podem ser acessados em sua docu-

mentação *online*[8].

3.1 Fluxo Geral

O ciclo básico de execução de uma aplicação de realidade aumentada é composto pelas seguintes etapas:

1. Captura e configuração do quadro de vídeo
2. Para cada marcador
   (a) Rastreamento do marcador no quadro
   (b) Estimativa de pose da câmera
   (c) Renderização

O fluxo de execução mais detalhado desse ciclo está modelado no diagrama de atividades apresentado no Apêndice B. Nele é possível observar como se relacionam os principais módulos do sistema durante sua execução. O restante desta seção minucia cada classe presente no modelo proposto, explicando o propósito de cada atributo e a ação realizada por cada método. A descrição de cada atividade é feita durante a explicação do método que a executa.

3.2 Marcadores Naturais

A representação dos marcadores naturais no sistema é feita pela classe *NaturalMarker*. Cada objeto dessa classe possui um identificador, uma variável booleana que indica o modo em que o marcador se encontra, um retângulo delimitador que representa a porção da imagem onde o marcador está localizado, os pontos de interesse (em *pixels*) da imagem do marcador, além dos descritores de características correspondentes. Os objetos dessa classe persistem ainda suas coordenadas de mundo 3D e o modelo do objeto virtual a ser aumentado na cena sobre o referencial do marcador. Por fim, a relação com a classe *Matches* fornece mais um atributo que armazena as correspondências encontradas no último quadro de vídeo processado.

O modelo suporta qualquer quantidade de marcadores, ou seja, a princípio um sistema implementado a partir desse modelo possibilitaria a projeção de vários objetos distintos em um mesmo quadro sobre os diferentes marcadores. Contudo, na prática isso requer um poder computacional bastante elevado, visto que para cada marcador seria necessário fazer o *match* entre ele e o quadro de vídeo corrente. Logo, no desenvolvimento será necessário limitar o

[8]http://docs.opencv.org/2.4.10/ [Online; acessado em 04/05/2017]

número de marcadores simultâneos no sistema, o que pode variar de acordo com a plataforma e linguagem escolhidas.

No trabalho de Ufkes [4] é possível registrar quantos marcadores o usuário desejar, porém a projeção se dá somente no melhor marcador da cena. A definição do melhor marcador é feita na fase chamada de *Frame Lookup* do modo *lost*, onde um descritor-vocabulário é casado com o quadro de vídeo corrente e com cada marcador registrado criando-se histogramas. O marcador que tiver o histograma com a menor distância em relação ao histograma do quadro é definido como potencialmente o melhor da cena. Somente esse marcador terá seu descritor casado com o descritor de características do quadro e, havendo um número considerável de bons *matches*, o mesmo será considerado visível e a estimativa de pose será feita em relação a ele.

Essa abordagem de Ufkes pode ser implementada no sistema aqui proposto diretamente no método principal da aplicação. Para isso, bastaria trocar o laço que percorre todos os marcadores pelo processo de seleção. Além disso seria necessário realizar um pré-processamento dos marcadores a fim de construir o descritor-vocabulário através da clusterização dos descritores característicos.

3.3 Descritores de Características

Os descritores de características são vetores numéricos que descrevem o entorno de cada ponto de interesse na imagem. A definição do ponto de interesse varia entre os detectores e as informações descritas variam entre os extratores. Descritores SIFT e SURF originais são compostos, respectivamente, de 128 e 64 números reais. Descritores ORB, por sua vez, possuem 256 valores binários. Esses e outros descritores estão implementados na biblioteca OpenCV sob uma mesma interface e possuem, inclusive, versões para processamento paralelo.

Ufkes compara em seu trabalho cada um desses algoritmos e os dados obtidos estão dispostos na Tabela 1. Foram utilizados dois *laptops* (Lenovo W520 e Alienware M17) e um *tablet* (Asus TF201) na execução dos testes que rodaram sem processamento paralelo por 1000 quadros de vídeo a uma resolução de 640x480 *pixels*. Todos os algoritmos avaliados fizeram a extração de 300 características por imagem em cada execução. A partir desses resultados, fica visível como a performance do ORB é muito superior em relação às demais técnicas comparadas. Um resultado não esperado é o desempenho do SURF inferior ao do SIFT, visto que o SURF foi proposto como sendo uma versão mais veloz do SIFT. O próprio autor menciona esse resultado não convencional e comenta que a implementação do SIFT no OpenCV possui várias otimizações. Ainda, os demais parâmetros de cada técnica também podem influenciar na performance do algoritmo, mas esses não foram mencionados no artigo.

Para determinar se um dado marcador está visível em um certo momento do vídeo

Tabela 1. Velocidade (ms/frame) de extração de descritores SIFT, SURF e ORB em diferentes plataformas (Fonte: [4])

Plataforma	*Clock* (GHz)	Velocidade de Extração		
		SIFT	SURF	ORB
M17	3.6	66.5	88.1	**3.60**
W520	2.2	84.7	160	**5.87**
TF201	1.4	694	1034	**41.0**

faz-se a correspondência entre os descritores do objeto marcador e da cena. Essa etapa é conhecida como *match*. Essencialmente, busca-se, para cada vetor que descreve o objeto, aquele na cena cuja distância é mínima. Isso pode ser feito através do método de força-bruta ou adotando alguma heurística. O OpenCV já possui diversos *matchers* disponíveis implementados sob uma mesma interface de fácil aprendizado. Na documentação da biblioteca é possível obter exemplos de utilização das diversas técnicas expostas aqui.

3.4 Rastreamento

A classe responsável por rastrear os marcadores naturais em cada imagem do vídeo é a *HybridTracker*. Ela possui uma variável booleana para indicar quando há ao menos um marcador perdido no sistema. O valor dessa variável é utilizado para evitar que novas características sejam extraídas de todos os quadros do vídeo. Quando todos os marcadores estiverem em modo de *tracking* não é necessário extrair tais descritores, uma vez que as características do quadro anterior serão utilizadas pelo fluxo óptico para estimar o movimento da cena e, assim, a posição das características no quadro corrente. Para isso, torna-se necessário persistir a imagem do vídeo e os pontos rastreados na mesma.

Outro atributo da classe de rastreamento é a referência para o objeto que define os algoritmos a serem utilizados pela biblioteca. A classe *TrackingAlgorithms* foi pensada como uma interface para as técnicas de visão computacional presentes no OpenCV, mais especificamente, as que estão relacionadas ao rastreamento de objetos. Dessa forma a biblioteca se torna customizável, uma vez que é possível alterar seu núcleo.

O método *Registry* é responsável por construir o marcador natural, fazendo a extração das características e registrando o modelo do objeto virtual que será aumentado sobre ele. Este é um método que deve ser evocado durante a fase de configuração da aplicação. O método *Update* faz a atualização do quadro de vídeo, extraindo os descritores característicos quando houver algum marcador perdido no sistema. Este deve ser chamado antes de iniciar o processo de rastreamento dos marcadores em cada quadro, o que é feito pelo método *Find*.

O método *Find* é a principal operação da classe e seu objetivo é, dado um marcador

e o quadro corrente, retornar as melhores correspondências entre os descritores do objeto e da cena. Nesse método é implementada a abordagem híbrida, que depende exclusivamente do modo em que o marcador se encontra. Se estiver em modo *lost*, as características da cena terão sido extraídas durante a execução do *Update*, restando agora realizar o *match* dos descritores. No caso do marcador não estar perdido, então passa-se ao fluxo óptico. A função *cv::calcOpticalFlowPyrLK* implementa a técnica proposta por [21].

Exceto no primeiro quadro do vídeo, onde todos os marcadores estão por padrão no modo *lost*, é o resultado do rastreamento que definirá o modo do marcador na iteração seguinte. Para entrar em modo *tracking* é necessário um bom número de características casadas. Ufkes, em sua pesquisa [4], determinou de forma empírica um limiar de 20 pontos para que a troca do sistema seja praticada.

Conforme o tempo vai passando, menos pontos vão sendo rastreados pelo fluxo óptico, devido às oclusões temporárias, movimentação brusca, condições de iluminação etc. Quando o número de pontos rastreados fica abaixo do limiar definido, o marcador volta ao modo *lost*. Uma forma de otimizar a extração de características é executá-la, sempre que possível, apenas sobre a porção da imagem que se encontra o marcador e não na imagem completa, diminuindo o espaço de busca.

O retorno do método são as correspondências encontradas por um dos sistemas de rastreamento junto com um valor de erro para cada correspondência. Este valor pode ser utilizado tanto para filtrar melhor as correspondências como para avaliar a acurácia do rastreamento. Entretanto, a métrica usada no cálculo do erro não é a mesma em todas as configurações possíveis de *matchers*, que dependem da função de distância utilizada. Ainda, o algoritmo de fluxo óptico no OpenCV apresenta duas possíveis medições de erro. Assim, torna-se necessário definir uma métrica própria compatível com qualquer configuração do sistema.

3.5 Estimativa de Pose

A classe *Camera* representa a câmera de vídeo real que filma o cenário. Essa classe mantém persistidos seus parâmetros intrínsecos e os coeficientes de distorção da lente. Tais dados são obtidos após a realização do processo de calibração da câmera. O OpenCV fornece funções para realizar essa calibração que geram um arquivo YAML como saída. Esse arquivo deve ser passado como parâmetro de entrada para o método *LoadConfiguration* que deve lê-lo e carregar as informações para a classe. Há ainda o atributo *resolution* que guarda o tamanho em *pixels* das imagens filmadas.

O método *Projection* computa a matriz de projeção perspectiva da câmera em função dos parâmetros intrínsecos, da resolução e das distâncias entre a câmera e os planos de visualização recebidos por parâmetro. O cálculo dessa matriz está diretamente relacionado à API

gráfica utilizada. Logo, para sua correta implementação deve-se verificar a documentação da API.

O principal método da classe é o método *Pose*, que calcula a matriz de transformação da câmera em relação ao marcador natural. Ele recebe como parâmetros as coordenadas de mundo do marcador em questão e as coordenadas correspondentes na imagem, além da variável que define o modo do sistema. Esse também é um método com comportamento híbrido, assim como sugere Ufkes em seu trabalho [4].

Com o marcador em modo *lost*, a estimativa de pose da câmera será feita pelo algoritmo EPnP dentro do laço do RANSAC. Esse método pode ser implementado utilizando a função *cv::solvePnPRansac* habilitando a *flag CV_EPNP*. Para os marcadores em modo *tracking* o método é o dos mínimos quadrados de Levenberg-Marquadt. Para utilizá-lo no OpenCV basta chamar a função *cv::solvePnp* habilitando a *flag CV_ITERATIVE*.

A saída de ambas funções são duas matrizes, uma de rotação e outra de translação, representando a orientação e posição da câmera, respectivamente. Deve-se agora converter essas matrizes para o formato da API gráfica utilizada. No OpenGL, por exemplo, o formato é uma matriz 4x4 homogênea linearizada por coluna em um vetor de 16 posições. Com as matrizes de projeção e pose pode-se realizar a renderização do objeto virtual sobre o marcador produzindo o efeito da Realidade Aumentada.

3.6 Interfaces

A classe *Application* é uma interface com todo o sistema. Ela interliga os módulos da biblioteca, encarregados pela parte de visão computacional, com a API gráfica que se encarrega da renderização. Além disso, gerencia o laço principal automatizando o desenvolvimento de aplicações. Esta classe foi inspirada na classe de mesmo nome da biblioteca AVRLib. Ela é também uma classe abstrata, ou seja, especificações devem ser implementadas para cada API que se deseja prover suporte.

O padrão de projetos *Builder* foi aplicado nessa parte do modelo a fim de desacoplar a construção da aplicação de sua representação concreta. Isso faz com que todas as etapas de configuração sejam realizadas antes da construção do objeto. Desse modo, a representação concreta do objeto pode conter somente métodos que, de fato, poderão ser utilizados durante a execução do programa. Outro ponto importante é a possibilidade de prover uma interface comum para a construção de diferentes representações da aplicação.

Os principais métodos da classe são os controladores *Start*, *Stop*, *Pause* e *Resume*, que são autoexplicativos, além do laço principal descrito na seção 3.1, onde todos os processos da aplicação são realizados.

4 Prova de Conceito

A fim de demonstrar o potencial desempenho do modelo proposto neste trabalho, o mesmo foi implementado usando a linguagem C++ no sistema operacional *Windows*. Como API gráfica utilizou-se a biblioteca GLUT/OpenGL. O código fonte está disponível na plataforma GitHub[9]. Não houve neste trabalho a preocupação em implementar um modelo otimizado com as melhores configurações de algoritmos possíveis e os melhores fatores. O objetivo dessa seção é mostrar que o modelo proposto é válido se implementado, ou seja, ele opera da forma desejada.

Na implementação foi considerado somente marcadores planos. Desse modo, após o processamento do *tracker*, a partir das correspondências encontradas, computou-se a matriz de homografia usando a função *cv::findHomography* que fornece a transformação projetiva entre os conjuntos de pontos correspondentes. Em outras palavras, a homografia é uma relação entre dois planos em diferentes perspectivas, no caso, o plano do marcador ortogonal à câmera (usado na inicialização do objeto *NaturalMarker*) e o plano do marcador observado no quadro de vídeo. A partir da homografia, computou-se os quatro vértices do marcador no quadro, usando a função *cv::perspectiveTransform*. Esses pontos são os correspondentes 2D das coordenadas 3D de mundo do marcador e, portanto, são usados no cálculo de pose da câmera.

4.1 Vídeos

Para avaliação da implementação feita, considerou-se duas sequências de imagens gravadas pelos autores utilizando uma *webcam Microsoft LifeCam* 720p. A sequência ORTHO possui 750 quadros 640x480p capturados a uma taxa de 25 quadros por segundo (FPS) com duração de 30 segundos. Essa sequência agrega diferentes escalas e oclusões parciais do marcador, mas quase nenhuma mudança de perspectiva, capturando o marcador praticamente sempre na ortogonal. A segunda sequência é a PERSP que consiste de 753 quadros 640x480p também capturados à 25 FPS (30.12s de duração). Essa sequência possui pouca variação de escala e oclusões parciais, mas agrega uma grande mudança de perspectiva. O marcador utilizado nessas sequências está ilustrado pela Figura 1.

4.2 Resultados

A execução dos testes[10] ocorreu em um *laptop* ASUS equipado com a 3ª geração do processador Intel Core i7-3537U@2GHz, placa gráfica NVIDIA GeForce GT 740M, 8GB de memória DDR3 e com sistema operacional *Windows* 10. Foram utilizadas 3 configurações

[9]https://github.com/AVRGroup/Markerless-AVRLib [Online; acessado em 04/05/2017]
[10]Vídeos com os testes disponíveis em https://goo.gl/fsLND0 [Online; acessado em 04/05/2017]

Figura 1. Marcador utilizado nas sequências ORTHO e PERSP. Note que a imagem possui muitos detalhes, o que aumenta o número de características identificáveis.

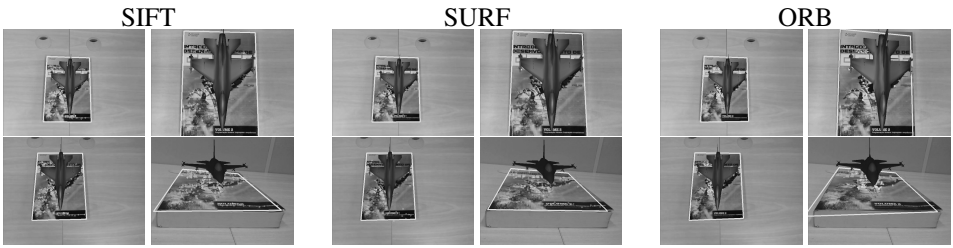

Figura 2. Primeiro e quingentésimo quadros da execução de cada configuração para as sequências ORTHO (cima) e PERSP (baixo).

da classe *TrackingAlgorithms*. Todas as configurações usam o algoritmo de fluxo-óptico proposto em [21] e o *matcher brute-force*. O que muda são os detectores e extratores e, por isso, estes são usados para identificar a configuração. Aqui foram testadas as principais técnicas que são o SIFT, o SURF e o ORB. A função usada no *matcher* é a distância euclidiana para SIFT e SURF e a distância de *Hamming* para o ORB. Esta última é o número de *bits* 1 resultante da operação binária **xor** entre os descritores. Os parâmetros utilizados são os padrões do OpenCV, exceto para o número máximo de características extraídas pelo ORB, onde aumentou-se o limite de 500 para 1500 a fim de melhorar os resultados visuais.

Foi medida a taxa de quadros por segundo médio (FPS) de 10 execuções de cada configuração em cada uma das sequências avaliadas, assim como o desvio padrão (STD) e o número médio de características casadas em uma execução. Os resultados obtidos estão dispostos na Tabela 2, onde nota-se o desempenho superior do SURF em relação às demais

Tabela 2. Taxa de quadros por segundo médio, em Hertz, de 10 execuções para cada sequência e cada configuração de algoritmos extratores avaliadas

	SIFT			SURF			ORB		
	#Keys	FPS	STD	#Keys	FPS	STD	#Keys	FPS	STD
ORTHO	262	17,20	0,29	134	19,64	0,25	281	17,46	0,27
PERSP	278	17,10	0,17	146	22,47	0,29	250	18,87	0,16

técnicas testadas. O aumento do limite de características extraídas pelo ORB teve grande impacto na performance desta configuração. Usando o padrão do OpenCV a execução passava facilmente de 30fps, mas os objetos virtuais ficavam muito instáveis, por isso optou-se pela alteração do parâmetro. Entretanto, em termos de acurácia o SIFT se sobressai. Neste trabalho, a acurácia não foi medida numericamente, mas observada visualmente (Figura 2). Para isso, considerou-se os critérios de estabilidade da projeção e o deslocamento dos vértices computados através da homografia em relação à sua posição real. Todos os algoritmos testados podem entregar resultados melhores aos apresentados aqui após um estudo detalhado das diferentes configurações de seus parâmetros.

5 Conclusão e Trabalhos Futuros

O presente trabalho propôs uma arquitetura de *software* para bibliotecas de *Markerless Augmented Reality* com o objetivo central de incentivar e criar condições para o desenvolvimento deste tipo biblioteca. A arquitetura proposta é independente de plataforma e linguagem. O modelo proposto utiliza a biblioteca *open source* OpenCV para dar suporte a todos os recursos de Visão Computacional necessários, agilizando a implementação do sistema. Considerando a implementação de futuras aplicações, a arquitetura foi planejada com base no modelo utilizado pela biblioteca AVRLib [12]. Esta biblioteca implementa o laço principal de execução do sistema internamente, simplificando o desenvolvimento de aplicações. Também é objetivo deste artigo servir como ponto de partida para aqueles que queiram ingressar na área e entender o funcionamento desse tipo sistema.

O desequilíbrio entre a demanda de poder computacional necessário ao rastreamento óptico de marcadores naturais e a exigência de processamento em tempo real das aplicações de RA foi um dos principais problemas identificados. O meio encontrado de contornar o problema foi utilizar um sistema híbrido, como o proposto em [4]. A evolução do *hardware* presenciada nos últimos anos também contribui na solução desse problema para a área.

Como trabalho futuro sugere-se estender a arquitetura proposta com potenciais novos recursos como rastreamento de objetos não planos, suporte à utilização de áudio, rede, entre outros. Também propõe-se avaliar as diversas configurações possíveis dos algoritmos do

OpenCV utilizados como base deste trabalho a fim de verificar as melhores combinações em termos de acurácia e desempenho.

6 Contribuição dos Autores

- Douglas Coelho Braga de Oliveira foi o responsável pela pesquisa e desenvolvimento do trabalho;

- Lidiane Teixeira Pereira realizou os testes do sistema e elaborou os vídeos da prova de conceito;

- Rodrigo Luis de Souza da Silva foi o orientador do trabalho.

Referências

[1] KATO, H. Artoolkit 2.33 (alpha version). Disponível em: <http://www.hitl.washington.edu/artoolkit/>. Acesso em: em 05 de maio de 2017.

[2] FIALA, M. Artag, a fiducial marker system using digital techniques. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 15., San Diego. *Anais...* San Diego: IEEE Computer Society, 2005. p. 590–596.

[3] JURADO, S. G. et al. Automatic generation and detection of highly reliable fiducial markers under occlusion. *The Journal of the Pattern Recognition Society*, v. 47, n. 6, p. 2280–2292, Jun 2014.

[4] UFKES, A.; FIALA, M. A markerless augmented reality system for mobile devices. In: International Conference on Computer and Robot Vision, 10., 2013, Regina, SK. *Anais...* Regina, SK: IEEE, 2013. p. 226–233.

[5] HERAKLEOUS, K.; POULLIS, C. Improving augmented reality applications with optical flow. In: IEEE International Conference on Image Processing, 20., 2013, Melbourne. *Anais...* Melbourne: IEEE, 2013. p. 3403–3406.

[6] HAMIDIA, M. et al. Markerless tracking using interest window for augmented reality applications. In: International Conference on Multimedia Computing and Systems (ICMCS), 10., 2014, Marrakech. *Anais...* Marrakech: IEEE, 2014. p. 20–25.

[7] LIMA, J. P. et al. Model based markerless 3D tracking applied to augmented reality. *SBC Journal on 3D Interactive Systems*, v. 01, 2010.

[8] PETIT, A.; MARCHAND, E.; KANANI, K. Augmenting markerless complex 3d objects by combining geometrical and color edge information. In: IEEE International Symposium on Mixed and Augmented Reality (ISMAR), 12., 2013, Adelaide. *Anais...* Adelaide: IEEE, 2013. p. 287–288.

[9] ZHU, W. et al. Real-time 3d model-based tracking of work-piece with monocular camera. In: IEEE/SICE International Symposium on System Integration (SII), 8., 2015, Nagoya. *Anais...* Nagoya: IEEE, 2015. p. 777–782.

[10] SOMMERAUER, P.; MüLLER, O. Augmented reality in informal learning environments: A field experiment in a mathematics exhibition. *Computers & Education*, v. 79, p. 59–68, 2014.

[11] CHEN, P. et al. An improved augmented reality system based on andar. *Journal of Visual Communication and Image Representation*, v. 37, p. 63–69, 2016.

[12] OLIVEIRA, D. C. B.; CAETANO, F. A.; SILVA, R. L. S. Avrlib - an object oriented augmented reality library. In: Workshop de Realidade Virtual e Aumentada, 10., 2013, Jataí. *Anais...* Jataí: IEEE, 2013. p. 54–59.

[13] REITMAYR, G.; SCHMALSTIEG, D. An open software architecture for virtual reality interaction. In: ACM Symposium on Virtual Reality Software and Technology, 8., 2001, Baniff. *Anais...* Baniff: ACM, 2001. p. 47–54.

[14] SILVA, W. A.; RIBEIRO, M. W. S.; JúNIOR, E. L.; CARDOSO, A. Uma arquitetura para distribuição de ambientes virtuais de realidade aumentada aplicada à educação. *Brazilian Journal of Computers in Education*, v. 16, n. 03, 2008.

[15] RUBLEE, E.; RABAUD, V.; KONOLIGE, K.; BRADSKI, G. Orb: An efficient alternative to sift or surf. In: International Conference on Computer Vision, 13., 2011, Barcelona. *Anais...* Barcelona: IEEE Computer Society, 2011. p. 2564–2571.

[16] LOWE, D. G. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, v. 60, n. 02, p. 91–110, 2004.

[17] BAY, H.; TUYTELAARS, T.; GOOL, L. V. *Surf: Speeded up robust features*. Springer Berlin Heidelberg, Maio 2006. p. 404–417.

[18] LEPETIT, V.; NOGUER, F. M.; FUA, P. Epnp: An accurate o(n) solution to the pnp problem. *International Journal of Computer Vision*, v. 81, n. 2, p. 155–166, 2008.

[19] FISCHLER, M. A.; BOLLES, R. C. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, v. 24, n. 6, p. 381–395, Jun 1981.

[20] LEVENBERG, K. A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, v. 2, n. 2, p. 164–168, 1944.

[21] LUCAS, B. D.; KANADE, T. An iterative image registration technique with an application to stereo vision. In: 7th International Joint Conference on Artificial Intelligence, 7., 1981, Vancouver. *Anais...* Vancouver: Morgan Kaufmann Publishers Inc., 1981. p. 674–679.

A DIAGRAMA DE CLASSES

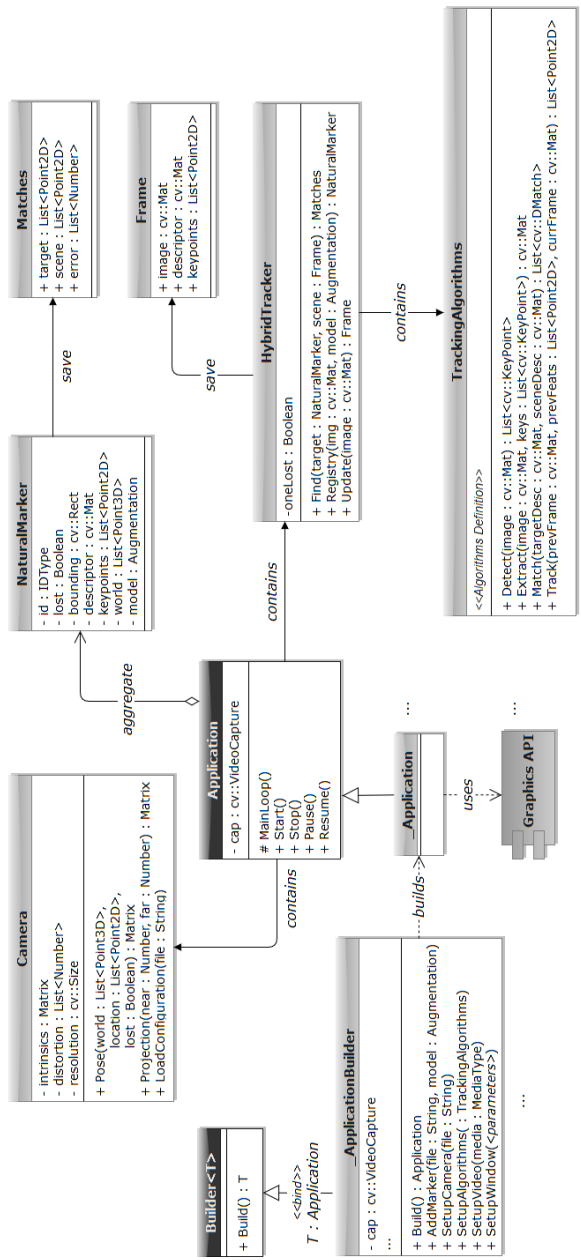

Figura 3. Diagrama de Classes simplificado da arquitetura de *software* proposta. As classes com o cabeçalho mais escuro representam classes abstratas ou interfaces. O padrão de projetos *Builder* foi utilizado para a definição das classes **Application*. Somente um exemplo ilustrativo é apresentado no modelo.

B DIAGRAMA DE ATIVIDADES

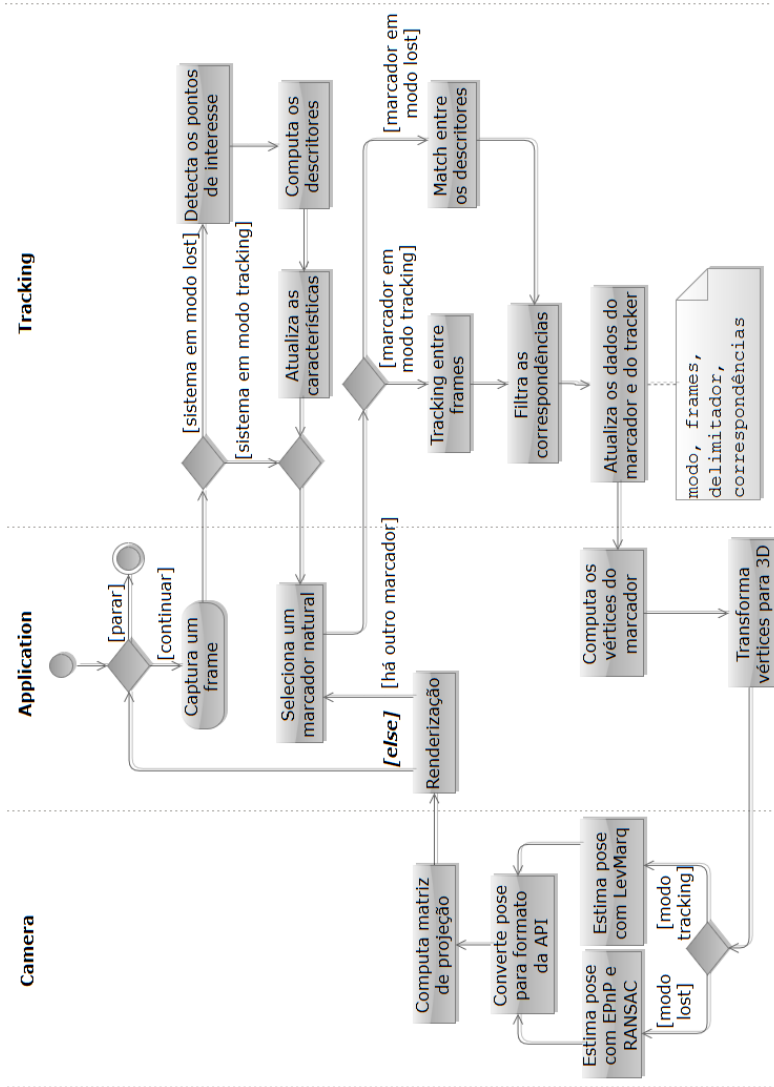

Figura 4. Fluxo de execução do laço principal