Experiences in Image-Based 3D Reconstruction of Underwater Environments

Vinícius César*, Thiago Farias†, Marcio Bueno*, Judith Kelner*
*Universidade Federal de Pernambuco
Centro de Informática
Recife, PE
email: {vmc, masb, jk}@cin.ufpe.br
†Universidade de Pernambuco
Caruaru, PE
email: thiago.souto.maior@gmail.com

***Resumo*—The 3D reconstruction of underwater environments have demonstrated its importance in many situations, especially in enviroments potentially harmful to human beings. Its is being used to reconstruct shipwrecks, coral reefs and expedition planning, documentation and study, as well as to make easier maintenance in structures lying in deep water. In this paper, a set of techniques previously tested is described and has its relevance explained in the matter of underwater 3D reconstruction. The results confirm the efficiency of the chosen techniques and it can be concluded that this framework can be applied to similar case studies without loss of generality.**

***Keywords*—3D reconstruction; underwater; framework; autocalibration**

I. Introdução

A reconstrução 3D de ambientes submarinos vem sendo utilizada em várias áreas, tanto acadêmicas quanto industriais. A reconstrução de naufrágios e recifes de corais, entre outras estruturas submarinas, tem como principal objetivo estudo, documentação e planejamento de expedições. Já na indústria, a reconstrução 3D é utilizada para guiar robôs e facilitar manutenções em estruturas localizadas em águas profundas, onde o ambiente é hostil ao ser humano.

Quando se trata de uma reconstrução em ambiente submarino é necessário se preocupar com alguns problemas que surgem a partir da submersão. Um dos problemas que ocorre é a variação dos parâmetros de calibração intrínseca da câmera, criando a necessidade de se fazer a calibração no mesmo ambiente em que vai ser executada a reconstrução. Vários fatores podem alterar a calibração, entre eles, a salinidade e a pressão que ocorrem nestes ambientes. A possível mudança destes fatores ao longo do processo de reconstrução requer uma atualização constante dos parâmetros de calibração através da autocalibração.

Outro fator que diferencia a reconstrução submarina das demais é a constante presença de partículas em suspensão que podem ser facilmente confundidas com *outliers* no processo de rastreamento. Desta forma, devem ser utilizados algoritmos robustos capazes de ignorar estas partículas e ao mesmo tempo preservar a precisão da reconstrução.

Estão disponíveis vários algoritmos que podem ser combinados em um *pipeline* de reconstrução 3D de ambientes submarinos. No entanto, este artigo visa detalhar uma determinada configuração de *pipeline*, testada pelos autores, que demonstrou-se eficiente na resolução dos problemas inerentes à reconstrução 3D submarina.

É importante ressaltar que muitos trabalhos na área de reconstrução 3D em ambientes submarinos utilizam técnicas baseadas em SLAM (*Simultaneous Localization and Mapping* [1]). Isto ocorre porque a maioria dos aparatos robóticos utilizados nas incursões submarinas[1] são equipados com diversos sensores, tais como IMU (*Inertial Measurement Unit*), sonar e câmeras. Estes são utilizados por uma estrutura baseada em predição e observação de dados nativa do SLAM. Porém, como este trabalho estende um framework de reconstrução 3D baseado somente em vídeo [2], utilizou-se algoritmos de SfM (*Structure from Motion*) [3] para obter os resultados. Desta forma, foram apresentados apenas como artigos relacionados os que também são baseados em SfM e tiveram seus *pipelines* descritos na íntegra.

Na Seção II são apresentados alguns trabalhos relacionados a *pipelines* de reconstrução 3D que utilizam algoritmos de SfM. Na Seção III está detalhada a metodologia utilizada no *framework* de reconstrução 3D proposto, bem como seus principais algoritmos. Alguns estudos de caso e resultados são apresentados na Seção IV. E por fim, algumas conclusões e trabalhos futuros estão descritos na Seção V.

II. Trabalhos Relacionados

Cavan [4], descreve um *pipeline* de reconstrução 3D submarina não calibrada em três estágios: rastreamento de vídeo, reconstrução projetiva e autocalibração. Na fase de rastreamento de vídeo é utilizado um algoritmo de rastreamento bastante conhecido baseado em fluxo ótico (KLT [5]) implementado através da biblioteca OpenCV [6]. Na fase seguinte, a reconstrução projetiva é realizada através de um tensor trifocal [7], recuperando assim as poses das câmeras e os pontos 3D através da triangulação ótima do espaço projetivo. Ao longo do vídeo, vários tensores são calculados, porém como cada reconstrução é relativa a uma tripla de *keyframes*, elas precisam, portanto, serem combinadas para formar uma

[1]Comumente chamados de ROV (*Remotely Operated Vehicle*) quando manipulados por um operador ou AUV (*Autonomous Underwater Vehicle*) quando são autônomos.

978-0-7695-5123-4/13 $26.00 © 2013 IEEE
DOI 10.1109/NAVCOMP.2013.19

única reconstrução, e para isto foi utilizada uma técnica de alinhamento hierárquico levando em conta os *keyframes* em comum entre as reconstruções. Entre alguns estágios do *pipeline* é utilizado um refinamento não linear das poses e dos pontos 3D (*Bundle Adjustment* [7]). A terceira e última etapa é a de autocalibração, responsável por transformar uma reconstrução projetiva, que não é apropriada para visualização ou medição, em uma reconstrução métrica na qual isto pode ser realizado. O foco do trabalho do Cavan, que também é o diferencial do seu trabalho, foi o desenvolvimento de uma nova técnica de autocalibração que utiliza uma abordagem híbrida entre duas outras técnicas pré-existentes.

Sedlazeck *et al.* [8] descreve o *pipeline* utilizado para realizar as reconstruções em seus estudos de caso. Neste *pipeline*, a calibração da câmera é realizada num passo anterior à reconstrução 3D. Após a calibração, os passos de reconstrução seguem o modelo clássico de SfM, como proposto em [7], porém utilizando um filtro especial de detecção de *outliers* devido ao alto nível de ruído gerado pelo ambiente marinho. O rastreamento é feito utilizando somente imagens de uma câmera HD através do algoritmo KLT [5]. Assim que um par de *keyframes* é encontrado, a inicialização do sistema é realizada a partir da geometria epipolar das imagens, e no decorrer do vídeo, as demais imagens são adicionadas. As demais poses das câmeras são calculadas sequencialmente a cada quadro através de correspondências 2D-3D (algoritmos PnP). Após a reconstrução da cena esparsa é aplicado um refinamento não linear global (*bundle adjustment*) para minimizar o erro das estimativas dos pontos 3D e do caminho das câmeras. Uma malha mais detalhada é computada num momento posterior utilizando uma estimativa para todos os pixels, gerando uma reconstrução densa da cena. Também são aplicados filtros nas imagens provenientes da câmera para corrigir as cores capturadas e exibir um modelo texturizado mais próximo das cores reais. Embora o artigo retrate o *pipeline* de reconstrução utilizado, os resultados apresentam um estudo do erro apenas na correção das cores, e não na reconstrução 3D em si.

Brandou *et al.* [9] também mostra um *pipeline* para reconstrução 3D submarina baseado num sistema manufaturado pelos autores que provê visão estéreo. Toda a reconstrução 3D é realizada *offline* e consiste de casamento das imagens, calibração das câmeras, recuperação da estrutura da cena e estimativa da superfície densa. O casamento das imagens é realizado através de uma técnica de extração de descritores e casamento de *features* da imagem. O algoritmo utilizado é o SIFT [10] (*Scale Invariant Feature Transform*), que é aplicado às imagens provenientes do sistema estéreo, gerando os casamentos necessários para a estimativa da geometria epipolar, que nesta fase serve para eliminar falsos casamentos. A calibração das câmeras também é feita de maneira *offline*, determinando previamente os parâmetros intrínsecos e extrínsecos (posicionamento e orientação relativos) de todas as câmeras que compõem o sistema estéreo. Posteriormente, são adicionadas informações contidas na geometria epipolar da cena, desta vez para calcular as poses das câmeras e preparar o sistema para a triangulação dos pontos 2D extraídos pelo SIFT. Após a estimativa da estrutura da cena (pontos 3D e câmeras), o erro é minimizado utilizando um método de minimização não-linear global. Após a recuperação da cena esparsa, é construído um mapa estéreo através da retificação das imagens adquiridas e da minimização de energia de um sistema construído a partir de um algoritmo de corte de grafo. De maneira similar a [8], o trabalho não menciona o erro da reconstrução gerada pelo *pipeline* descrito no artigo.

III. Framework

O *framework* de reconstrução foi projetado para reconstruir sequências de vídeo utilizando a arquitetura descrita em [2]. O processo é executado de forma *online*, ou seja, os *frames* do vídeo são processados à medida que são capturados. O *framework* recebe como entrada um vídeo e produz como saída um conjunto de poses de câmera, que corresponde à posição de câmera relativa à cada *frame* capturado, e uma nuvem de pontos 3D, que corresponde à estrutura dos objetos da cena reconstruída.

Por utilizar a abordagem SfM, o *framework* possui duas restrições: a cena deve ser rígida, ou seja, não pode haver movimento ou deformação dos objetos da cena; e a câmera deve realizar um movimento de translação, para que se possa obter imagens em múltiplos pontos de vista. Ainda existe outra restrição de que a cena deve ser rica em textura para que o rastreador possa detectar e rastrear os pontos no decorrer do vídeo.

A cada *frame* são extraídas as *features* e computadas as correspondências com os *frames* anteriores. Para isto, utiliza-se o extrator *Good Features to Track* [11] e o casamento das *features* entre os *frames* é realizado pelo KLT [5], [12]. O *pipeline* de reconstrução é representado por uma máquina de estados composta por dois estados: *Inicialização* e *Pronto*, ilustrados na Figura 1. O primeiro estado é o de Inicialização, que estima as poses das primeiras câmeras, a calibração e uma estrutura inicial de pontos 3D. Após seu término, começa o estado Pronto, que dura até o final do vídeo. Nele, cada *frame* é processado utilizando as correspondências provenientes do rastreamento.

Na fase de rastreamento, o *framework* busca sempre obter o máximo de *features* rastreadas. Isso permite que sejam geradas nuvens de pontos 3D mais densas, mesmo sendo a reconstrução esparsa. Porém, devido à limitação do rastreador, podem ocorrer erros como falsos casamentos e *drift* que podem degradar a precisão da reconstrução. Embora muitos pontos sejam rastreados, apenas um subconjunto desses pontos com melhores coeficientes de Harris [13] será utilizado em todas as etapas do *pipeline*. A quantidade de pontos presentes nesse subconjunto é determinada previamente pelo usuário numa fase de configuração. Os demais pontos serão apenas utilizados na fase de triangulação e os que possuírem um erro de reprojeção abaixo de um limiar determinado pelo usuário serão integrados à estrutura final.

A. Inicialização

A primeira etapa da Inicialização é a seleção de *keyframes*, como ilustrado pela Figura 1. Ela é responsável por processar os pontos 2D e selecionar três *frames* chaves utilizando o algoritmo GRIC [14] (*Geometrical Robust Information Criterion*) entre o *frame* atual e o último *keyframe* encontrado, garantindo assim que exista um deslocamento mínimo entre as poses das câmeras utilizadas para capturar os *keyframes* necessário para os algoritmos de SfM. O primeiro *frame* do vídeo, por definição, torna-se o primeiro *keyframe* da reconstrução. Ao

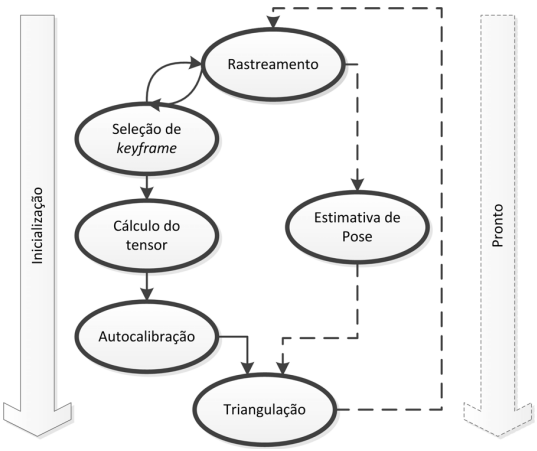


Figura 1. Esquema do pipeline

selecionar o terceiro *keyframe*, é realizada uma reconstrução inicial baseada nas *features* rastreadas entre os três *keyframes*, obtendo uma nuvem esparsa de pontos 3D e três poses de câmera. Ainda na Inicialização, é extraída também a calibração da câmera, que é considerada fixa por todo o vídeo.

É possível reconstruir com dois ou três *keyframes*, porém foi escolhida a segunda opção neste *framework* por três motivos: há uma maior quantidade de informação para computar a estrutura inicial; o processo de detecção de *outliers* é mais robusto; e existem menos casos de configuração de cenas que impediriam a reconstrução (casos degenerados [7]).

Dadas as correspondências das *features* nos três *keyframes*, o primeiro passo é a detecção de *outliers*, que são falsos casamentos ocorridos no rastreamento e degradam a precisão da reconstrução. Para isso utilizou-se a abordagem do RANSAC [15]: gerar hipóteses aleatórias com um subconjunto mínimo de amostras e selecionar a melhor hipótese com base em alguma métrica estabelecida. Neste *framework* utilizou-se o LMedS [16], uma variação do RANSAC, para a escolha robusta de hipóteses. Para gerar a hipótese, utilizou-se o algoritmo *Six-Point-Three-View* [7], que computa até três reconstruções a partir de seis pontos e três imagens. Como o algoritmo pode gerar até três soluções possíveis, todas elas são avaliadas pelo LMedS. A métrica utilizada foi o erro de reprojeção.

Após a detecção e exclusão de *outliers*, são computadas as poses das três câmeras referentes a cada *keyframe*, utilizando somente as correspondências consideradas *inliers*. Para isso, estima-se o tensor trifocal, que consiste em um tensor $3 \times 3 \times 3$ que agrega as restrições entre as correspondências nos três *keyframes*. Para estimar o tensor utilizou-se a técnica descrita por Hartley em [7]. Essa técnica primeiro estima uma solução de forma linear que será utilizada como valor inicial, uma vez que o problema é não-linear. Para refinar o resultado utiliza-se uma minimização não-linear de Levenberg-Marquardt [7]. A partir do tensor, pode-se computar três matrizes de câmera na forma canônica referentes aos três *keyframes*. Nesse ponto do *framework*, a reconstrução é projetiva e o plano do infinito pode estar no meio da cena, ocasionando um fenômeno conhecido como quiralidade. Antes de executar o passo de autocalibração, é preciso realizar a correção da quiralidade. A principal consequência desse fenômeno são pontos triangulados atrás das câmeras. Para corrigir isso, é computado uma transformação $\mathtt{H}$ 4×4 que move toda a cena para o mesmo lado do plano do infinito. Uma descrição do método através de programação linear é dada em [7]. Como esse processo requer pontos 3D, alguns pontos escolhidos aleatoriamente são triangulados apenas para a correção da quiralidade.

Para que seja obtida a reconstrução euclidiana da cena, é preciso calcular agora a calibração da câmera. Para isso, foi utilizada uma abordagem modificada do algoritmo de [17], que por sua vez, é a fusão das técnicas [18] e [19]. No modelo *pinhole*, a calibração da câmera possui cinco graus de liberdade: foco, razão aspecto entre o foco na direção x e y, *skew* e a posição do ponto principal no eixo x e y. Porém, uma abordagem muito utilizada no processo de autocalibração é diminuir os graus de liberdade da calibração da câmera. A grande maioria das câmeras no mercado possuem as seguintes restrições: (1) possui o ponto principal no centro da imagem, (2) razão aspecto do foco igual a um e (3) *skew* muito próximo a zero. Isso torna o processo de autocalibração mais simples, pois é possível determinar apenas o valor da distância focal, sabendo que os demais graus de liberdade obedecem às restrições do mercado. Outra característica das câmeras do mercado é que existe uma faixa de valores possíveis para o foco, fato que é utilizado durante o processo de autocalibração. Outra simplificação utilizada no *framework* é que durante toda a captura, o foco da câmera permanece constante. Assim, o problema se reduz a encontrar apenas um valor de foco, e não um foco para cada câmera (*frame*).

Algorithm 1 Autocalibração

1: $melhor\_custo \leftarrow \infty$
2: $\mathtt{H}^* \leftarrow \mathtt{I}$
3: **para todo** $f \in L$ **faça**
4: $\quad custo\_atual \leftarrow 0$
5: $\quad$ **para** $\mathtt{P} \leftarrow \mathtt{P}_i, i = 2 \ldots n$ **faça**
6: $\quad\quad$ calcula-se π_∞ com $\mathtt{P}$ ▷ Descrito em [18]
7: $\quad\quad$ calcula-se $\mathtt{H}$ com π_∞ e f ▷ Descrito em [17]
8: $\quad\quad \mathtt{P}^* \leftarrow \mathtt{PH}$
9: $\quad\quad$ obter $\mathtt{K}$ a partir de $\mathtt{P}^*$ usando a decomposição RQ
10: $\quad\quad custo\_atual \leftarrow custo\_atual + \mathcal{C}(\mathtt{K})$
11: $\quad$ **fim para**
12: $\quad$ **se** $custo\_atual < melhor\_custo$ **então**
13: $\quad\quad \mathtt{H}^* \leftarrow \mathtt{H}$
14: $\quad\quad melhor\_custo \leftarrow custo\_atual$
15: $\quad$ **fim se**
16: **fim para**
17: **retornar** $\mathtt{H}^*$

O algoritmo de autocalibração, descrito no Algoritmo 1, possui pequenas variações em relação a [17]. Dado que é conhecido o intervalo L em que o foco está, a estratégia da técnica consiste em varrer esse intervalo e, para cada valor do foco, obter uma medida através da função de custo $\mathcal{C}$ dada pela Equação 1. Para minimizar a discrepância de valores que geram instabilidade numérica é necessário realizar a normalização das câmeras [17] antes da execução deste

algoritmo.

$$\mathcal{C}(\mathtt{K}) = \frac{w_s \mathtt{K}_{12} + w_a(\mathtt{K}_{11}^2 + \mathtt{K}_{22}^2) + w_{x_c}\mathtt{K}_{13} + w_{y_c}\mathtt{K}_{23}}{(\mathtt{K}_{11}^2 + \mathtt{K}_{22}^2)} \quad (1)$$

Na Equação 1 o custo da matriz de calibração é computado com base nas três restrições citadas anteriormente e os valores w_s, w_a, w_{x_c} e w_{y_c} são pesos que atuam nessas restrições. Uma possível valoração para estes pesos pode ser encontrada em [4].

De posse da saída do Algoritmo 1, $\mathtt{H}^*$, pode-se transformar todas as câmeras em suas equivalentes no espaço euclidiano. Na sequência, utilizando as câmeras e as correspondências das *features*, é possível realizar a triangulação e assim obter uma estrutura inicial da cena. O algoritmo DLT [7], quando usado para triangulação, não é invariante à projetividade, se tornando ineficaz na triangulação de cenas projetivas. Entretanto, como a cena foi autocalibrada e se encontra num espaço euclidiano, pode-se aplicar o algoritmo DLT para triangulação. Por fim uma minimização não-linear do erro de reprojeção é aplicada aos pontos 3D utilizando o algoritmo de Levenberg-Marquardt.

São triangulados todos os pontos rastreados, inclusive aqueles não utilizados nas outras etapas do *pipeline*. Mesmo sendo triangulados, esses pontos 3D continuarão sem participar das demais fases do *pipeline* e compõem unicamente a cena reconstruída. A triangulação é o último passo da Inicialização. Nesse momento foi obtido um conjunto de câmeras e pontos 3D formando a estrutura inicial da cena. Inicia-se então o estado Pronto que continua a reconstrução processando cada novo *frame* e incrementando a estrutura inicial de pontos 3D e câmeras.

B. Pronto

A cada novo *frame*, o rastreamento identifica a posição dos pontos 3D já triangulados na nova imagem. Dessa forma pode-se estabelecer um conjunto de correspondências de pontos 2D e 3D e utilizar um algoritmo de estimativa de pose de câmera, comumente chamado de PnP, para determinar a pose da câmera do *frame* sendo processado.

À medida que a cena é capturada, novos pontos 3D podem ser visualizados pela câmera e serão detectados e rastreados pelo KLT. A partir das matrizes de câmeras é possível triangular os pontos que surgem, agregando o resultado na nuvem de pontos final. Da mesma forma que novos pontos são rastreados, muitos podem ser perdidos, e isto não causa problemas ao *pipeline*. Os novos pontos triangulados também servem para estimar a pose da próxima câmera e assim continuar o processo de reconstrução que é iterativo e incremental.

A triangulação continua sendo realizada pelo DLT e sempre utilizando todas as imagens em o que o ponto aparece. Pontos já triangulados não precisam ser retriangulados quando aparecem em uma nova imagem. Porém, assim o fazendo, seu erro será minimizado

Para estimar a pose da câmera utiliza-se o algoritmo EPnP [20] em conjunto com um algoritmo de refinamento CPC [21] que melhora a precisão da câmera através de uma minimização não-linear. A escolha desses algoritmos foi dada com base em [22], um estudo que mediu a performance de vários algoritmos PnP em reconstrução 3D.

(a) Sequência do *sino*

(b) Sequência do *casco*

(c) Sequência do *cano*

Figura 2. *Frames* iniciais e finais de cada sequência.

Como o processo de obtenção de pontos 2D ainda é suscetível a falsos casamentos, antes de estimar a pose da câmera, é preciso realizar a detecção de *outliers*. Nesse caso, também se utiliza o LMedS, com o algoritmo EPnP para estimar a hipótese com quatro correspondências 2D-3D.

IV. Resultados

O *framework* proposto foi testado utilizando trechos de vídeos de documentários relacionados a naufrágios. São boas entradas para testar o *framework* por se tratarem de vídeos submarinos e obtidos de formas imparciais, pois não foram produzidas para serem executadas em nosso *framework*. Alguns *frames* das sequências utilizadas estão ilustrados na Figura 2. Duas sequências foram extraídas do filme *Titanic at 100: Mystery Solved*.[2] A primeira sequência recebeu o nome *sino* (Figura 2(a)) e a segunda recebeu o nome *casco* (Figura 2(b)). A terceira sequência foi extraída de um filme disponível na web [3] e retrata o naufrágio do navio Cristobal. O trecho desse filme utilizado como entrada para o *framework* foi chamado de *cano* (Figura 2(c)).

As sequências *sino* e *casco* possuem resolução 1280×720 e a sequência *cano* possui resolução 1920×1080. Estes mesmos trechos possuem respectivamente 44, 34 e 49 *frames*. Por não se conhecer a câmera utilizada nas filmagens, a calibração da câmera também é desconhecida e será descoberta pelo processo de autocalibração.

O hardware utilizado nos testes foi um processador i7-3960X a 3.3 GHz com 24GB de memória RAM DDR3 e

[2]Produzido por Edgeworx Studios em 15 de abril de 2012 e distribuído pelo History Channel.

[3]http://www.youtube.com/watch?v=7khZxnGze4c

sistema operacional Windows 7 64 bits. O *framework* foi implementado em C/C++ e seu tempo de execução é limitado pelo rastreamento de pontos 2D. A depender da quantidade de pontos e do tamanho da imagem, o rastreamento pode ser custoso (mais de um segundo) para rastrear um único *frame*. Porém, é um algoritmo altamente paralelizável e já existem versões otimizadas em placas gráficas que funcionam em tempo real [23]. As outras etapas (seleção de *keyframes*, estimativa de pose, triangulação, etc.) são executadas rapidamente, somando menos de 25 milissegundos por *frame*. Deste modo, o *framework* tem potencial para rodar em tempo real, desde que se otimize a etapa de rastreamento.

As sequências de vídeo foram reconstruídas pelo *framework* e o resultado é ilustrado na Figura 3. A nuvem de pontos 3D é melhor visualizada através de vídeos. Os vídeos relativos às reconstruções das entradas utilizadas estão disponíveis na web[4]. O erro de reprojeção também foi mensurado e reportado na Tabela I. Esse erro corresponde ao erro de reprojeção médio de todas as correspondências 2D-3D estabelecidas na reconstrução de todas as imagens. Esse erro foi computado utilizando o subconjunto de pontos 2D utilizados em todas as etapas do *pipeline*. Os pontos triangulados utilizados apenas para enriquecer a malha não entram no cálculo.

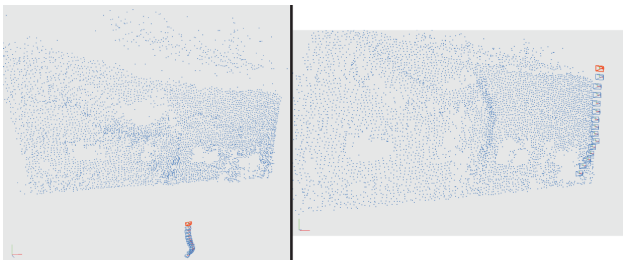
O rastreamento de muitos pontos 2D permite obter muitos pontos 3D, cerca de alguns milhares de pontos, caracterizando assim uma reconstrução esparsa. Na Figura 3 também está representado o caminho que a câmera realizou, porém só algumas câmeras reconstruídas são mostradas para fins ilustrativos.

Na reconstrução do *sino*, Figura 3(a), foram utilizadas 1000 *features* para reconstrução e no total foram reconstruídos 5339 pontos. Nota-se que foi reconstruída corretamente a superfície no fundo da cena, inclusive com as cavidades das janelas. A haste metálica também foi reconstruída. Devido aos poucos *frames* disponíveis, alguns *outliers* que ligam a ponta da haste à superfície não foram identificados. O caminho de câmera também foi reconstruído fielmente, pois está realizando um movimento vertical para baixo. O erro de reprojeção se mostrou baixo (vide Tabela I), sendo menor que meio pixel indicando uma boa precisão da reconstrução.

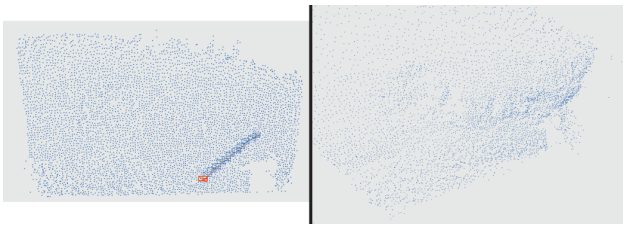
Na sequência do *casco* também foram utilizadas 1000 *features* para realizar todo processo de reconstrução e a quantidade de pontos reconstruídos foi 5330. Na Figura 3(b) percebe-se a reconstrução de toda a superfície com a presença de poucos *outliers*. O caminho de câmera também está bem representado já que está se deslocando para a direita, assim como no vídeo. O erro de reprojeção da reconstrução tem valor próximo à sequência do *sino* (vide Tabela I), mostrando também uma boa precisão dessa reconstrução.

É importante observar que as sequências *sino* e *casco*, ambas do documentário do Titanic, possuem uma logomarca no canto inferior direito do vídeo, visível nas Figuras 2(a) e 2(b). Essas regiões do vídeo não foram desconsideradas e foram detectadas e rastreadas. Como são pontos que não correspondem à cena, eles são considerados *outliers*, que foram detectados durante a fase de reconstrução e descartados. Demonstrando ainda mais a robustez do *framework*.

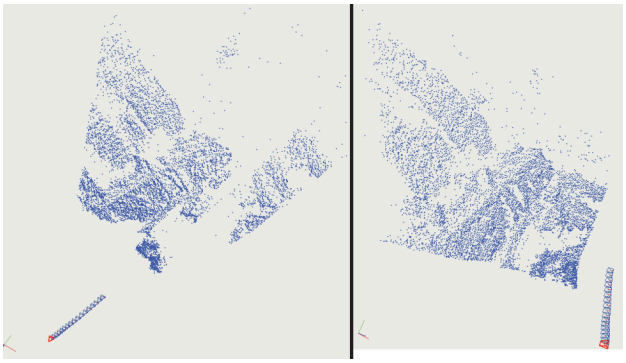
[4]Os resultados estão online através dos links:
http://youtu.be/gg44dEXE_gQ
http://youtu.be/5cRXJE597BU
http://youtu.be/jQleA3YhO_s



(a) Sequência do *sino*



(b) Sequência do *casco*



(c) Sequência do *cano*

Figura 3. Reconstruções das sequências de vídeo. Nestas figuras são ilustrados a nuvem de pontos e o caminho de câmera.

A sequência do *cano* foi submetida ao *framework*, sendo rastreadas 600 *features* para a reconstrução e no total foram triangulados 7688 pontos. Mais pontos que as outras sequências já que a resolução do vídeo é maior. Através da Figura 3(c), percebe-se que a superfície da máquina no plano principal do vídeo foi bem reconstruída. No fundo, onde está o cano, os objetos estão distantes e embaçados devido à água dificultando o rastreamento nesse local. O erro de reprojeção foi superior às demais sequências, porém ainda se vê um bom resultado com valor abaixo de 1 *pixel*. No fundo do vídeo aparece parte de um mergulhador que se move durante a sequência e que também é rastreado. Porém, a detecção de *outliers* foi robusta e o excluiu da cena reconstruída.

V. Conclusão

Este artigo apresenta um *framework* de reconstrução 3D de ambientes submersos baseado em SfM. Este conjunto de técnicas agrega algoritmos robustos encadeados de acordo com

Tabela I. PRECISÃO DAS RECONSTRUÇÕES

	Erro de Reprojeção (*pixels*)
Sino	0.3091
Casco	0.3370
Cano	0.6444

a experiência dos autores em prévios trabalhos na área de reconstrução 3D na atmosfera. Algumas etapas do *pipeline* foram adaptadas para funcionar em ambientes submersos, ganhando em robustez e possibilitando a calibração da câmera independente do ambiente (profundidade, salinidade, etc.). Embora existam outras técnicas de reconstrução 3D, o encadeamento destas técnicas específicas que compõem o *framework* demonstraram-se suficientes para gerar bons resultados na fase esparsa da reconstrução 3D dos pontos e das poses das câmeras, uma vez que os erros de reprojeção se mantiveram em média bem abaixo de 1 *pixel*.

Na literatura, vê-se poucos artigos em que o *pipeline* inteiro é descrito, uma vez que os avanços tecnológicos publicados são relativos a etapas específicas do processo. Artigos que relatam as experiências de utilização de técnicas encadeadas em um *pipeline* e demostrando resultados tanto visuais quanto numéricos do erro gerado durante a reconstrução não são facilmente encontrados, o que faz deste artigo uma contribuição para pesquisadores que procuram uma visão geral de sistemas de reconstrução 3D submarina.

Como trabalhos futuros devem ser incluídas fases de refinamento iterativo da calibração, e dos artefatos da reconstrução (pontos 3D e poses de câmera), bem como uma otimização dos algoritmos de rastreamento para que o *pipeline* possa ser executado em tempo real. Em conjunto com uma análise de tempo de execução de cada fase do *pipeline*, devem também ser adicionadas comparações entre outros algoritmos de reconstrução 3D submarina, semelhante às comparações vistas em [2], [22].

REFERÊNCIAS

[1] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "MonoSLAM: Real-Time single camera SLAM," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 6, pp. 1052–1067, Jun. 2007. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2007.1049

[2] T. Farias, "Metodologia para reconstrução 3d baseda em imagens," Ph.D. dissertation, Centro de Informática - Universidade Federal de Pernambuco, 2012.

[3] Y. Ma, S. Soatto, J. Kosecka, and S. S. Sastry, *An Invitation to 3-D Vision*. Springer, Nov. 2003.

[4] N. Cavan, "Reconstruction of 3d points from uncalibrated underwater video," 2011.

[5] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI '81)*, April 1981, pp. 674–679.

[6] P. R. Koch, "Opencv – open source computer vision," [página web] http://opencv.willowgarage.com/, acessada em 11 junho 2012.

[7] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, ISBN: 0521540518, 2004.

[8] A. Sedlazeck, K. Köser, and R. Koch, "3d reconstruction based on underwater video from rov kiel 6000 considering underwater imaging conditions," in *IEEE OCEANS Conference*, Bremen, Germany, 2009.

[9] V. Brandou, A. Allais, M. Perrier, E. Malis, P. Rives, J. Sarrazin, and P. Sarradin, "3d reconstruction of natural underwater scenes using the stereovision system iris," in *IEEE OCEANS Conference*, Aberdeen, Scotland, Junho 2007.

[10] D. G. Lowe, "Object Recognition from Local Scale-Invariant Features," in *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ser. ICCV '99. Washington, DC, USA: IEEE Computer Society, 1999. [Online]. Available: http://portal.acm.org/citation.cfm?id=851523

[11] J. Shi and Tomasi, "Good features to track," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition CVPR-94*. IEEE Comput. Soc. Press, Jun. 1994, pp. 593–600. [Online]. Available: http://dx.doi.org/10.1109/CVPR.1994.323794

[12] C. Tomasi and T. Kanade, "Detection and tracking of point features," International Journal of Computer Vision, Tech. Rep., 1991.

[13] C. Harris and M. Stephens, "A combined corner and edge detector," in *The Fourth Alvey Vision Conference*, 1988, pp. 147–151.

[14] M. T. Ahmed, M. N. Dailey, J. L. Landabaso, and N. Herrero, "Robust key frame extraction for 3d reconstruction from video streams," in *International Conference on Computer Vision Theory and Applications (VISAPP)*, MAY 2010.

[15] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981. [Online]. Available: http://doi.acm.org/10.1145/358669.358692

[16] P. J. Rousseeuw and A. M. Leroy, *Robust regression and outlier detection*. New York, NY, USA: John Wiley & Sons, Inc., 1987.

[17] N. Cavan, P. W. Fieguth, and D. A. Clausi, "Autocalibration: Finding infinity in a projective reconstruction," in *CRV'11*, 2011, pp. 197–203.

[18] R. Gherardi and A. Fusiello, "Practical autocalibration," in *Proceedings of the 11th European conference on Computer vision: Part I*, ser. ECCV'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 790–801. [Online]. Available: http://dl.acm.org/citation.cfm?id=1886063.1886123

[19] D. Nistér, "Untwisting a projective reconstruction," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 165–183, Nov. 2004. [Online]. Available: http://dx.doi.org/10.1023/B:VISI.0000029667.76852.a1

[20] V. Lepetit, F. Moreno-Noguer, and P. Fua, "Epnp: An accurate o(n) solution to the pnp problem," *Int. J. Comput. Vision*, vol. 81, pp. 155–166, February 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1487388.1487412

[21] H. Araújo, R. L. Carceroni, and C. M. Brown, "A fully projective formulation to improve the accuracy of lowe's pose-estimation algorithm," *Computer Vision and Image Understanding*, vol. 70, no. 2, pp. 227 – 238, 1998. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1077314297906329

[22] V. César, "Metodologia para avaliação de poses de câmera em reconstrução 3d baseada em vídeo," 2012.

[23] J.-S. Kim, M. Hwangbo, and T. Kanade, "Realtime affine-photometric klt feature tracker on gpu in cuda framework," in *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, 27 2009-oct. 4 2009, pp. 886 –893.