**Universidade Estadual de Campinas**
**Instituto de Computação**

# Planejamento de Trajetória de *Drones* para Reconstrução 3D

Tiago Loureiro Chaves (Aluno)

Profa. Dra. Esther Luna Colombini (Orientadora)

Processo 2020/07892-8
Vigência 01/07/2020 a 30/06/2021
Relatório Final (dezembro — junho)

## Resumo

Reconstruções 3D precisas são fundamentais para aplicações em diversas áreas, desde robótica e realidade virtual até medicina, geologia, agronegócio e arquitetura. Veículos aéreos não tripulados têm sido cada vez mais usados para a captura de imagens à medida que modelos menores e mais acessíveis ficam disponíveis no mercado. Com essas imagens podemos gerar modelos 3D de larga escala da cena sobrevoada, mas a qualidade do resultado depende muito do plano de voo e do trajeto traçado, e ainda requer pilotos experientes para ambientes complexos. Assim, a pesquisa relacionada à obtenção de imagens para a reconstrução de estruturas de grande porte vem ganhando importância, principalmente no que tange a capacidade de percepção destes veículos. Devido à falta de informações precisas e atuais sobre o ambiente, soluções automatizadas para *drones* recorrem a voos em padrões regulares a uma distância aérea segura. No entanto, em muitos casos os pontos de vista capturados são insuficientes para uma reconstrução 3D de alta qualidade. Desse modo, a transição de sistemas automatizados para sistemas autônomos é fundamental. Neste cenário, este projeto propôs-se a explorar formas de otimizar a trajetória de voo em tempo real, a partir essencialmente da visão adquirida através de câmeras RGB acopladas ao *drone*. Apresentamos um conjunto de cenas adequadas para a avaliação do problema de reconstrução usando um simulador foto-realista, disponíveis livremente para uso, assim como métricas que possibilitam a análise quantitativa dos resultados – para comparar a trajetória planejada com a que é de fato traçada, e as nuvens de ponto geradas durante o voo com os modelos 3D verdadeiros. Também sugerimos modificações a uma heurística de qualidade usada em dois trabalhos recentes para a reconstrução de cenas urbanas, viabilizando seu cálculo em tempo real. Por fim, concluímos o relatório com uma discussão de possíveis ramificações que esse projeto pode trazer em trabalhos futuros.

Esther Luna Colombini

# 1 Introdução

Resolver o problema da reconstrução 3D é uma das etapas necessárias para a real compreensão de cenas (*scene understanding*), sendo um objetivo antigo da visão computacional, amplamente abordado na literatura [1, 2]. Entretanto, e não por acaso, a reconstrução de cenas a partir de pontos de vista distintos continua sendo um problema não resolvido [3], mas as diversas aplicações de tal processo o justificam como tema de inúmeros estudos que tentam tratar desse assunto desafiador.

O interesse em técnicas de reconstrução 3D não é restrito a disciplinas da ciência da computação, como visão computacional e computação gráfica. Seu uso vai desde a indústria agrícola e a construção civil — para cálculo de volumes, modelagem digital de terrenos, monitoramento, inspeção, mapeamento e levantamento topográfico — até a arqueologia (com a fotogrametria) e a área médica (na imagiologia).

Dentre tais aplicações, destaca-se o crescente uso de *drones* com câmeras acopladas para a captura de imagens aéreas. Para isso, soluções comerciais provêm trajetórias pré-programadas baseadas em grade ou em hélice (conforme Figura 1).

Porém, quando aplicadas a cenas de modelagem complexa, essas abordagens resultam em lacunas na cobertura total, ou em inconsistências no modelo 3D obtido. Assim, garantir alta qualidade para a reconstrução final ainda dependente do controle manual de pilotos (voo livre).

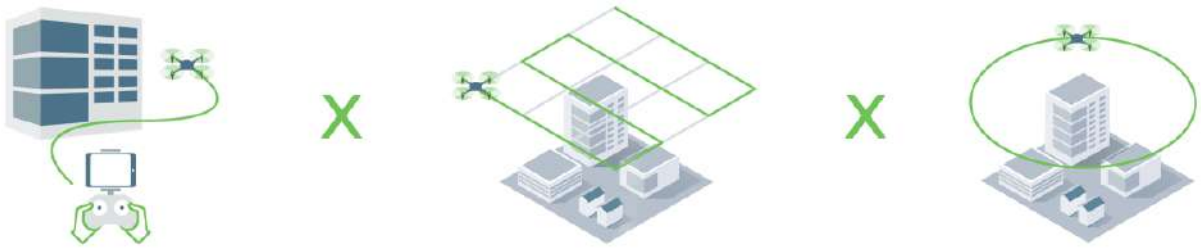

Figura 1: Ilustração esquemática do voo livre, controlado manualmente (esquerda), do voo automatizado com trajetória baseada em grade (centro), e do voo automatizado com trajetória em hélice/circular (direita) [1]

Outro fator relevante quando abordamos a otimização da trajetória de captura realizada por *drones* é que seu tempo máximo de voo, atualmente, é de cerca de 20 a 30 minutos — em contrapartida, a recarga da bateria leva de uma a duas horas — o que torna otimizações nesse processo ainda mais importantes. Adicionalmente, como diversos algoritmos dependem de condições de cena estáveis para uma boa reconstrução (p. ex. iluminação constante), a obtenção de imagens não pode levar muito tempo, o que impossibilita a combinação de dados capturados em dias distintos por exemplo.

Portanto, neste projeto exploramos formas de tornar o voo de *drones* para captura de imagens mais eficiente, de modo que modelos 3D superiores aos obtidos pela rota em grade (ou em hélice) utilizada atualmente por voos automatizados possam ser atingidos. Ressaltamos, como no começo da seção, que

[1] Imagem extraída da página: www.pix4d.com/product/pix4dcapture.

ao otimizarmos a trajetória com foco na reconstrução 3D estamos também influenciando a quantidade de informação que é capturada da cena. Ou seja, ao estudarmos métricas que avaliam a acurácia e completude dos resultados, temos também uma forma de comparar a influência e classificar a capacidade que diferentes trajetórias tiveram em extrair uma estrutura tridimensional a partir das imagens, o que tem reflexos no objetivo almejado de uma compreensão total da cena.

Assim, este projeto de iniciação científica teve como principais contribuições:

1. A construção de uma *baseline* para avaliação da qualidade de modelos reconstruídos a partir das estratégias de voo autônomo utilizadas atualmente.
2. A implementação de métricas de avaliação de acurácia (*precision*) e completude (*recall*) de reconstruções 3D obtidas [4], com base na construção incremental de uma nuvem de pontos da estrutura sobrevoada.
3. A modificação de uma heurística estado-da-arte [5, 6], que estima a contribuição de cada pose (i.e. posição e orientação da câmera acoplada ao *drone*) na qualidade final da reconstrução, para que fosse possível executá-la, e analisá-la, durante o voo.
4. A discussão de possíveis extensões do trabalho que foi desenvolvido para promover ajustes da trajetória em tempo real — com o objetivo de otimizar a captura de imagens para reconstrução, considerando os desvios do plano de voo que decorrem da interação dinâmica do *drone* com o ambiente — e para abordagens do problema de planejamento pela óptica do aprendizado por reforço.

## 2 Visão Geral

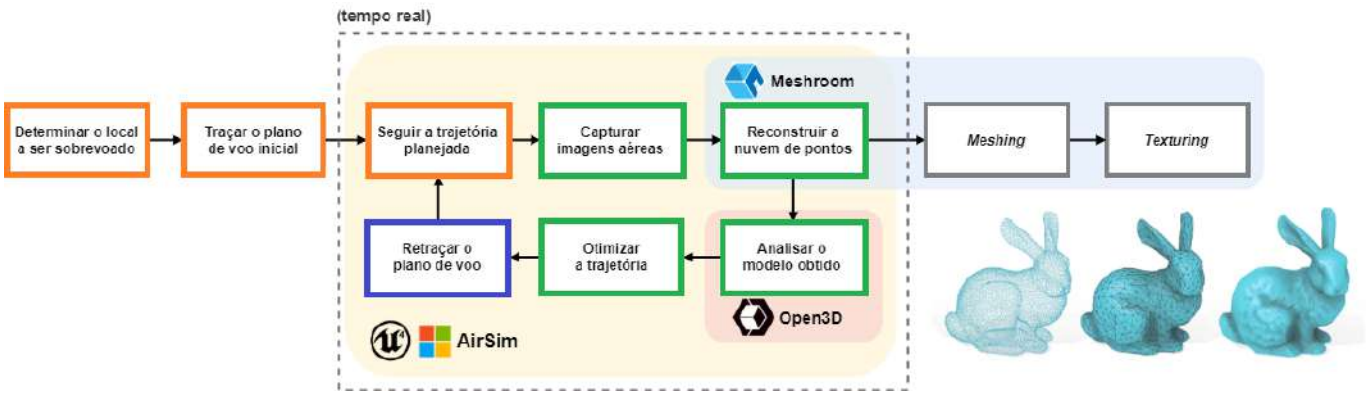


Figura 2: *Pipeline* de etapas envolvidas no processo de otimização da trajetória. Em laranja são destacadas as atividades desenvolvidas principalmente na primeira metade do projeto; em verde as que tiveram maior concentração na segunda metade; e em azul ressalta-se a etapa que é foco na discussão de trabalhos futuros. [2]

A Figura 2 apresenta os componentes do *framework* proposto, destacando em verde os blocos que foram abordados durante a segunda fase do projeto, enquanto os destacados em laranja concentraram-se principal-

[2] Ilustração esquemática das três fases finais da reconstrução 3D: *point cloud*, geração da malha triangular, e mapeamento de textura; utilizando imagens do coelho de Stanford modificadas a partir de [7].

mente na primeira fase (discutida no relatório parcial [8]). É importante esclarecer que:

1. Não faz parte do objetivo executar as etapas de *Meshing* e *Texturing* (destacadas em cinza) pois elas são demoradas e, assim, não passíveis de análise em tempo real. Entretanto, uma vez que dispomos das imagens e da reconstrução esparsa correspondente (etapa de *Structure from Motion* do processo de fotogrametria), nada impede que elas sejam utilizadas para a reconstrução posterior de uma malha texturizada (i.e. *offline*, em contrapartida com a criação *online* de *point clouds*);

2. Tratamos o controlador do *drone* como uma caixa preta. Por isso, pautamos nossa discussão em algoritmos que determinem como saída as poses (*viewpoints*, com posição e orientação) subsequentes que um *drone* deve alcançar para capturar imagens. Desta forma, objetivamos otimizar a aplicação dos algoritmos de controle em um nível mais alto do sistema, através da otimização da tarefa.

Para desenvolver o projeto, conforme apresentado na proposta [9], usamos o simulador de veículos autônomos AirSim [10] — que é executado na Unreal Engine 4 (UE4), uma *engine* de jogos escrita em C++ e reconhecida por sua fidelidade visual — desenvolvido pela Microsoft AI & Research, para realizar todos os experimentos com um *quadrotor* acoplado com cinco câmeras RGB, sensores cinemáticos, LIDAR e GPS.

Apesar de termos diversas câmeras disponíveis no simulador (três frontais, uma traseira e uma voltada para baixo), nos limitamos ao uso de apenas uma delas para cada experimento, visando aplicações do trabalho no mundo real com *drones* de baixo-custo.

Além disso, sabe-se que a navegação de *drones* em ambientes fechados ou sem sinal de GPS ainda é um grande desafio, porém, é um requisito básico para sistemas robóticos que devem operar no mundo real. Assim, decidimos nos restringir ao uso de sensores visuais e inerciais, como nos trabalhos [11, 12] de *Simultaneous Localization And Mapping* (SLAM [13]). A partir das imagens obtidas, utilizamos a *pipeline* de *Structure from Motion* (SfM [14]) do framework de fotogrametria AliceVision, com o software Meshroom [15, 16], para reconstruir iterativamente uma nuvem de pontos que representasse uma aproximação da estrutura tridimensional capturada, assim como as poses — posição e orientação — estimadas para a câmera a cada captura (que correspondem às posições do *drone*, onde as câmeras estão acopladas). Usamos, então, a biblioteca Open3D para processar e analisar esses dados tridimensionais, publicada em 2018 pelo Intel Intelligent Systems Lab [17].

A tabela 1 apresenta uma visão geral do cronograma de 12 meses do projeto proposto, dividindo-o em grandes etapas representativas das atividades desenvolvidas. Nas seções 3 e 4, abordamos dois importantes trabalhos relacionados diretamente à heurística que desenvolvemos. Então, descrevemos em maiores detalhes as principais atividades em que trabalhamos neste segundo período do projeto de iniciação científica, dando foco a cada elemento da *pipeline* apresentada na Figura 2. Concluímos o relatório na seção 10 com trabalhos futuros e considerações finais.

| Atividades | Meses | | | | | |
|---|---|---|---|---|---|---|
| | 1-2 | 3-4 | 5-6 | 7-8 | 9-10 | 11-12 |
| Revisão bibliográfica | ✓ | ✓ | ✓ | ✓ | ✓ | |
| Configuração do *pipeline* de experimentos | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Criação de uma *baseline* | ✓ | ✓ | ✓ | | ✓ | ✓ |
| Configuração de métricas de avaliação | | ✓ | ✓ | ✓ | ✓ | |
| Reconstrução da nuvem de pontos | | ✓ | ✓ | ✓ | ✓ | |
| Extensão da heurística de Smith et al. | | | | | ✓ | ✓ |
| Execução em tempo real | | | ✓ | | ✓ | ✓ |
| Disseminação de resultados | | | ✓ | | | ✓ |

Tabela 1: Cronograma de atividades do projeto proposto: ✓Atividades concluídas

**Revisão bibliográfica:** Esta etapa consiste em nos manter atualizados com o estado-da-arte em trabalhos relacionados ao projeto, como discutido na seção 3 (e também nas seções sobre trabalhos relacionados da proposta e do relatório anual [9, 8]).

**Configuração do *pipeline* de experimentos:** No primeiro período, preparamos as ferramentas utilizadas no projeto — AirSim (simulador), Meshroom (reconstrução 3D) e Open3D (análise dos dados tridimensionais) — para automatizar os experimentos, integrando suas APIs Python de forma a acelerar o processo de: desenvolvimento → teste → avaliação → desenvolvimento. No segundo período, essa etapa envolveu a compreensão e a execução do `uavmvs` — discutido na seção 4 — a criação de diversos *scripts* para visualização, usando as APIs do AirSim e Open3D, e o cálculo de métricas a partir de reconstruções feitas incrementalmente com o Meshroom.

**Criação de uma *baseline*:** Partimos de trajetórias simples, pré-programadas, comumente utilizadas (p. ex. em grade e em hélice, ilustradas na Figura 1), coletando imagens de um conjunto de cenas simuladas na Unreal Engine para podermos utilizá-las como base de referência para avaliar o problema tratado, comparando a qualidade de reconstruções feitas (conforme descrito na seção 9). Selecionamos, então, regiões de duas cenas disponíveis gratuitamente pelo Unreal Engine Marketplace [3] — baseados no uso de outros trabalhos da literatura [18, 19, 20, 5] — e computamos as métricas de erro para: a reconstrução 3D final de cada uma (avaliada conforme [4]) e o movimento estimado da câmera, comparado com a verdadeira trajetória do *drone* — avaliando *Absolute Trajectory Error* (ATE) e *Relative Pose Error* (RPE).

**Configuração de métricas de avaliação:** Esta etapa engloba: a implementação do procedimento estabelecido por Knapitsch et al. [4] para avaliar o desempenho da reconstrução da nuvem de pontos (comparada com o modelo 3D usado pelo simulador); a aplicação das métricas *ATE* e *RPE* usadas para SLAM, seguindo o *TUM dataset* [21, 22]; assim como as atividades envolvidas na extensão da heurística desenvolvida por [5] para uso em tempo real.

[3] https://www.unrealengine.com/marketplace

**Reconstrução da nuvem de pontos:** Na primeira fase do projeto, realizamos o *setup* necessário para geração e avaliação (*offline*) das nuvens de pontos reconstruídas pelo Meshroom a partir das imagens coletadas no AirSim. Nesta fase, focamos na análise em tempo real da nuvem de pontos reconstruída (incrementalmente), com o objetivo de possibilitar a identificação *online* de próximas regiões que devem ser capturadas da estrutura sobrevoada, de modo a aprimorar a reconstrução 3D final.

**Extensão da heurística de Smith et al.:** Investimos grande parte do tempo desta fase do projeto em estudar, reproduzir e estender o trabalho de Smith et al. [5] — como descrevemos em detalhes na seção 4 — por apresentar uma heurística estado-da-arte para a avaliação da contribuição que diferentes poses de câmeras (*viewpoints*) têm na qualidade do modelo reconstruído.

**Execução em tempo real:** Esta etapa envolve o nosso foco em extensões do estudo desenvolvido durante o projeto para abordagens do problema de reconstrução 3D pelo aprendizado por reforço, e para a elaboração de soluções que considerem ajustes da trajetória em tempo real. Para isso, é necessário que as métricas de avaliação sejam computáveis em pouco tempo, o que impede o uso da malha triangular na análise, por exemplo, já que é uma fase da *pipeline* de fotogrametria bem mais demorada do que a construção das *point clouds*.

**Disseminação de resultados:** Submetemos os relatórios técnicos do projeto, nos correspondemos com autores dos principais trabalhos que referenciamos, documentamos os algoritmos desenvolvidos e divulgamos o projeto publicando-o de forma aberta[4] — o que levou parte da estrutura construída para usar o simulador a ser utilizada na equipe de sistemas autônomos da Unicamp E-Racing. Também pretendemos submeter os resultados desse projeto na forma de artigo, para conferências relacionadas à área de robótica durante os próximos meses.

## 3 Análise de Trabalhos Relacionados

Diferentemente da discussão sobre trabalhos anteriores na proposta do projeto [9] na qual apresentamos a literatura sobre os três principais sistemas que precisam operar juntos para possibilitar a otimização de trajetórias com *drones* autônomos — visão, mapeamento e planejamento — e também da coleção de avanços recentes que abordamos no relatório anual [8] mostrando a crescente tendência de uso de simuladores foto-realistas e de abordagens para problemas relacionadas usando técnicas de aprendizado de máquina, neste relatório daremos destaque específico a dois trabalhos que tratam diretamente do problema de planejamento de trajetória para a reconstrução de cenas.

Em 2018, Smith et al. [5] publicaram um trabalho propondo um novo método de otimização para o planejamento da captura de imagens em cenas urbanas usando *drones*, apoiado pela proposição de uma heurística baseada nos fatores que influenciam a qualidade de reconstruções a partir de diversas imagens (*multi-view stereo reconstruction*). Para avaliar seu método, eles também usam a Unreal Engine para criar

[4] https://github.com/laurelkeys/ff

uma *benchmark* sintética, por facilitar a captura de imagens e a comparação de reconstruções com os modelos geométricos simulados.

A partir da heurística descrita por Smith et al., no fim de 2020 Huang et al. [6] propuseram uma forma de tratar o problema abordado pelo ponto de vista da otimização de dois objetivos conflitantes: a maximização da qualidade da reconstrução final e a minimização do comprimento do caminho e da quantidade de imagens capturadas durante o voo. Para tanto, eles adotaram a mesma heurística proposta por Smith et al., e fizeram experimentos na mesma *benchmark*, mostrando resultados equiparáveis (com qualidade semelhante, mesmo usando menos imagens).

Além disso, uma contribuição importante feita pelo trabalho é que, enquanto diversas abordagens encontradas na literatura que propõem métodos de otimização acabam definindo o problema de forma NP-difícil, Huang et al. mostram que é possível tratar os dois objetivos isoladamente, resolvendo-os de modo guloso e sequencial. Assim, como destcado no último relatório, tais trabalhos mostram que seguimos uma linha de estudo ativa, e que vem mostrando importantes avanços atualmente. Por isso, focamos no segundo período do projeto na viabilização do cálculo da heurística proposta por Smith et al. em tempo real.

# 4 Extensão da Heurística de Smith et al.

Grande parte do tempo desse segundo período de vigência do projeto foi relacionado a compreensão, execução e modificação do trabalho apresentado por Smith et al. [5]. Como mencionado na seção 3, apesar do trabalho mais recente ligado à nossa proposta de estudo ser o desenvolvido por Huang et al. [6] em 2020, ele utiliza a heurística proposta por Smith et al. para avaliar a contribuição que diferentes *viewpoints* têm na reconstrução de cenas, sem fazer nenhuma modificação nos valores de seus parâmetros — além de [6] não ter disponibilizado código ou executáveis. Assim, podemos dizer que o trabalho de Smith et al. de 2018 ainda é estado-da-arte quando se trata de métricas de otimização que podem ser utilizadas para a solução do problema que exploramos.

Logo, concluímos que seria importante entender e conseguir executar a implementação publicada junto com esse trabalho: "`uavmvs` - UAV capture planning for MVS reconstructions" [23].

## 4.1 Execução e Estudo

Apesar do código ser aberto, não há nenhuma forma de documentação [5] — nem mesmo por meio de comentários no código, explicando nem mesmo os parâmetros das funções mais importantes, já que vários deles não são abordados pelo artigo com a mesma nomenclatura — sua última atualização foi em 2018, e as instruções

[5] Veja por exemplo os diversos *issues* abertos no repositório do `uavmvs` com dúvidas sobre como utilizá-lo: https://github.com/nmoehrle/uavmvs/issues

de compilação não funcionavam com esta última versão[6]. Assim, para conseguirmos executá-lo foi necessário investigar como corrigir esses problemas de compilação.

Outra complicação foi que algumas das bibliotecas externas, das quais o `uavmvs` depende, só são suportadas oficialmente para Linux — enquanto nós usamos um ambiente Windows para desenvolver o projeto, por ser a plataforma com melhor suporte e informações de referência para a Unreal Engine. Portanto, também investimos tempo criando uma imagem Docker[7]com um ambiente onde pudéssemos facilmente alterar e recompilar o código (já que o `uavmvs` usa versões antigas específicas de algumas dependências, p.ex. as relacionadas ao NVIDIA CUDA Compiler [24]).

Com isso, depois de garantirmos que a compilação estava funcionando de maneira reprodutível, nos restou ler e depurar o código disponível para conseguir entender alguns dos pontos que não foram esclarecidos pelo artigo [5], devido à falta de documentação.

### 4.2 Algoritmo e Alterações

A heurística proposta por Smith et al. tem como objetivo estimar a influência que um par de câmeras tem na qualidade da reconstrução final de cada *sample* da nuvem de pontos, para poder ser utilizada na otimização do problema de planejamento de trajetória selecionando um conjunto mínimo de pontos de vista necessários. A Figura 3 apresenta um esquema das medidas empregadas no cômputo da heurística.

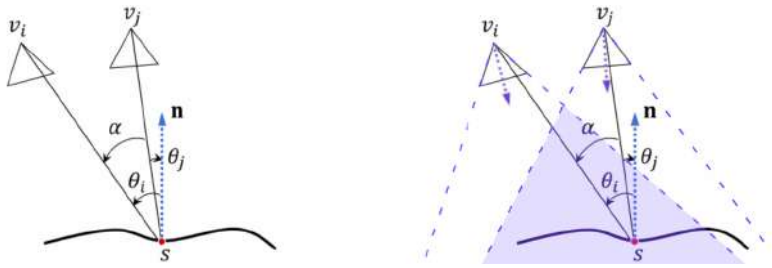


Figura 3: Dois *viewpoints* observando um *sample* da nuvem de pontos reconstruída, com as medidas utilizadas pela heurística no cálculo da *reconstrutibilidade* de cada ponto. Na direita, destacamos em roxo o *view frustum* e o vetor de orientação das câmeras.

Dado um par de *viewpoints* $(v_i, v_j)$ para as câmeras, a posição de um ponto $s$ e seu vetor normal de superfície $n$, a contribuição de $(v_i, v_j)$ para a qualidade da reconstrução de $s$ (sua *reconstrutibilidade*) é calculada por:

$$q(s, v_i, v_j) = w_1(\alpha) \cdot w_2(d_m) \cdot w_3(\alpha) \cdot cos(\theta_m), \text{ onde:} \quad (1)$$

- $\alpha$ é o ângulo de paralaxe entre os vetores $s\vec{v}_i$ e $s\vec{v}_j$, que vão de $s$ aos *viewpoints*
- $\theta_m = max(\theta_i, \theta_j)$ é o maior dos dois ângulos entre $n$ e $s\vec{v}_i$ e $s\vec{v}_j$

[6]Enquanto finalizávamos este relatório, algumas das correções que fizemos foram incorporadas ao `uavmvs`: https://github.com/nmoehrle/uavmvs/pull/17

[7]O *Dockerfile* utilizado, notas relacionadas ao processo que seguimos para conseguir executar o `uavmvs` e um *script* feito para testar seus diferentes componentes podem ser vistos em: https://gist.github.com/laurelkeys/0010bbd0409bcaef221003e4dfdca60f#file-uavmvs-md

- $d_m = max(||s\vec{v}_i||, ||s\vec{v}_j||)$ é a distância de $s$ até a câmera mais distante

As funções $w_1$, $w_2$ e $w_3$ são definidas por Smith et al. como:

$$w_1(\alpha) = (1 + \exp -k_1 \cdot (\alpha - \alpha_1))^{-1},$$
$$w_2(d_m) = 1 - min(d_m/d_{max}, 1),$$
$$w_3(\alpha) = 1 - (1 + \exp -k_3 \cdot (\alpha - \alpha_3))^{-1},$$

com constantes $k_1 = 32$, $\alpha_1 = \frac{\pi}{16}$, $k_3 = \frac{k_1}{4}$, $\alpha_3 = 4 \cdot \alpha_1$, e onde o parâmetro $d_{max}$ deve ser escolhido com base na relação entre os pixels das imagens capturadas e a distância real entre as posições representadas por eles (Huang et al. sugerem inicializá-lo como o dobro do valor de *ground sample distance* [25]).

Com $q(s, v_i, v_j)$ e a função binária $\delta(s, v)$, usada para avaliação da visibilidade do *sample* a partir de um ponto de vista $v$ — que é igual a 1 se $s$ é visível, e 0 caso contrário — a heurística é definida para um conjunto $V$ de *viewpoints* como:

$$h(s, V) = \sum_{\substack{i=1...|V| \\ j=i+1...|V|}} \delta(s, v_i) \cdot \delta(s, v_j) \cdot q(s, v_i, v_j). \tag{2}$$

Atentamos dois fatores usados na heurística que inviabilizam o cálculo em tempo real:

1. Para cada *sample*, o vetor normal é obtido a partir das faces triangulares dos modelos 3D usados nas cenas sintéticas (também simuladas com a Unreal Engine 4).

2. A função de visibilidade $\delta(s, v)$ é determinada a partir da técnica de *ray casting* [26], usando estruturas de aceleração para calcular interseções "raio-triângulo" [27].

Como mencionamos no início do relatório (seção 2), a construção de uma malha triangular que represente a superfície da cena a partir da nuvem de pontos gerada pela etapa de *Structure from Motion* é um processo consideravelmente mais demorado. Logo, só é viável calcular a heurística em simulação — onde já temos acesso às estruturas *ground-truth* usadas pela Unreal Engine, por exemplo — ou então, no mundo real, usando uma reconstrução inicial aproximada como nas estratégias de *explore-then-exploit* — mas que implica em dois voos sobre a cena, sendo que o segundo só pode começar após a *pipeline* completa de fotogrametria ser finalizada para as imagens capturadas no voo inicial.

Assim, propomos métodos de estimar os fatores 1 e 2 usando apenas nuvens de pontos — o que possibilita o uso das *point clouds* que são geradas incrementalmente durante o voo para a aproximação da heurística — e mostramos sua viabilidade usando implementações de algoritmos geométricos bem estabelecidos pela biblioteca Open3D ( ver Figuras 4 e 5): 1) É possível calcular uma aproximação para os vetores normais de cada *sample* com base no ajuste de um plano tridimensional a um conjunto de pontos em sua vizinhança (o que gera dois resultados válidos, com direções opostas, que então comparamos com a posição das câmeras para escolher a orientação adequada); 2) Usamos o algoritmo de remoção de pontos escondidos (*Hidden Point*

*Removal*) proposto por Katz et al. [28] que tenta determinar a visibilidade de uma nuvem de pontos relativa a um dado ponto no espaço, sem a reconstrução da superfície (ou mesmo a estimativa dos vetores normais).

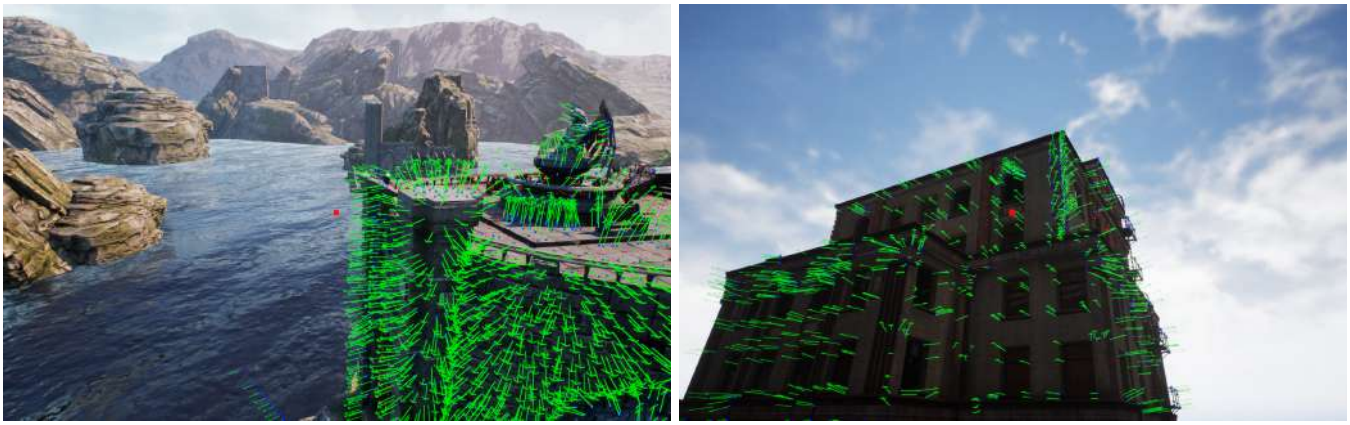

Figura 4: Em verde são representados os vetores normais estimados para *samples* visíveis a partir do ponto de vista da câmera (nas cenas dos modelos `Estátua` e `Prédio 3C`).

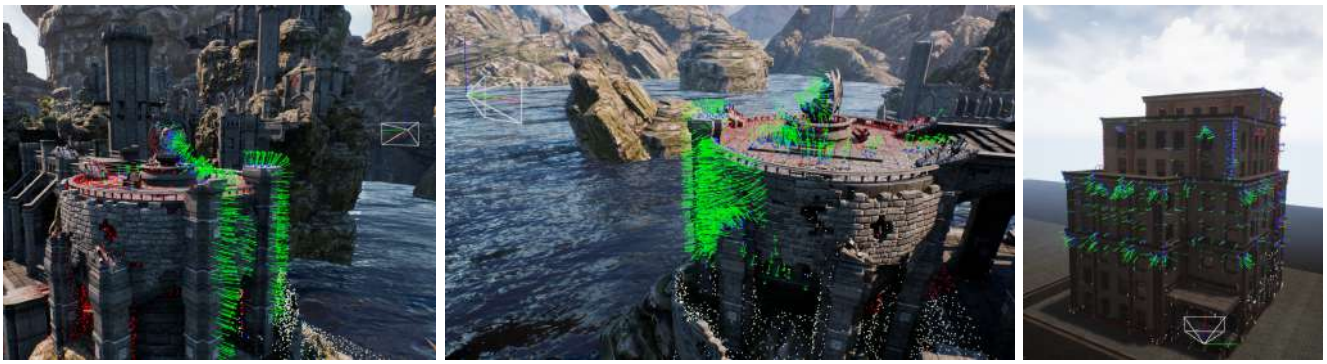

Figura 5: Com base nos *viewpoints* da Figura 4, pontos coloridos em branco estão fora do *view frustum* ilustrado nas imagens, enquanto os destacados em vermelho estão dentro dele mas não foram considerados visíveis pelo algoritmo de *Hidden Point Removal* [28] (na esquerda, note a parte que fica escondida da câmera, e na direita, as janelas dos andares superiores do prédio que não são observáveis a partir da posição baixa da câmera).

Além disso, notamos que os ângulos $\alpha$ e $\theta_m$ no cálculo de $q(s, v_i, v_j)$ usam apenas os vetores que ligam as posições das câmeras aos pontos, desconsiderando o alinhamento deles com a orientação de cada câmera (i.e. para onde elas apontam de fato). Por isso, consideramos que a adição de um fator a mais na função $q$, ponderando-a pelo grau de alinhamento entre os dois vetores, seria apropriada:

$$q_\gamma(s, v_i, v_j) = w_1(\alpha) \cdot w_2(d_m) \cdot w_3(\alpha) \cdot cos(\theta_m) \cdot max(0, cos(\phi_m))^\gamma, \text{ onde:} \tag{3}$$

- $\phi_m = max(\phi_i, \phi_j)$, sendo $\phi_i$ o ângulo entre $s\vec{v}_i$ e o vetor de orientação da câmera $i$, e $\phi_j$ o ângulo entre $s\vec{v}_j$ e o vetor de orientação da câmera $j$
- $\gamma \in [0, 1]$, tal que $max(0, cos(\phi_m))^{\gamma=0} = 1$, e assim: $q_{\gamma=0}(s, v_i, v_j) = q(s, v_i, v_j)$

A constante $\gamma$ nos dá controle sobre a influência que a similaridade por cossenos [29] tem no valor da

contribuição de cada par de câmeras para a *reconstrutibilidade* dos pontos. Usamos também a função $max(0, \cdot)$ para garantir que $q_{\gamma>0} = 0$ quando $|\phi_m| > 90$.

# 5 Determinar o Local a Ser Sobrevoado & Traçar o Plano de Voo Inicial

Um fator importante a considerar é a reprodutibilidade dos experimentos. Para isso, delimitamos a área que será sobrevoada com um retângulo (*bounding box*) e salvamos suas coordenadas, associadas ao *environment* simulado para cada experimento. Como precisamos dos modelos *ground-truth* de cada cena para comparar as reconstruções feitas nos experimentos, exportamos a malha poligonal dos objetos contidos nesses *bounding boxes* pelo próprio editor da Unreal Engine 4 — como os dados só podem ser exportados no formato FBX [30], algumas etapas de pré-processamento são necessárias antes de podermos usá-los com as métricas de avaliação (que descreveremos melhor na seção 9).

A partir da descrição das áreas de interesse, implementamos um *script* para que o *drone* sobrevoe a região com trajetórias em grade ou em hélice, e que nos dá a possibilidade de variar a altura de voo e a quantidade de zigue-zagues (*lanes* percorridas para completar o trajeto, vide Figura 9) ou o raio da hélice circular (Figura 6), para podermos efetuar diferentes experimentos. Para os voos em grade, também usamos a planejamento de *viewpoints* gerado pelo `uavmvs`, usado por Smith et al. como uma trajetória inicial, já que adotam uma estratégia de *explore-then-exploit* em seu trabalho.

Figura 6: Trajetórias em hélice com o *view frustum* ilustrado para as poses de captura.

Uma detalhe importante é que, em seu trabalho de 2020, Huang et al. mencionam que dentre as quatro cenas publicadas por Smith et al. eles não conseguiram reproduzir os resultados reportados para duas delas, nem aproximadamente (veja a nota de rodapé na página 7 do artigo [6]).

Notamos problemas parecidos, principalmente após a otimização dos *viewpoints*. Enquanto na trajetória inicial todos os pontos apontam a câmera totalmente para baixo — i.e. com linha de visão ortogonal ao chão — o resultado da otimização da trajetória por vezes apresentou orientações incoerentes, que simplesmente não faziam sentido. Por outras, a otimização manteve quase todas as câmeras na mesma pose e mudou a

posição ou orientação de algumas também de forma incompreensível, como foi o caso do modelo `Estátua` (ilustrado na Figura 7).

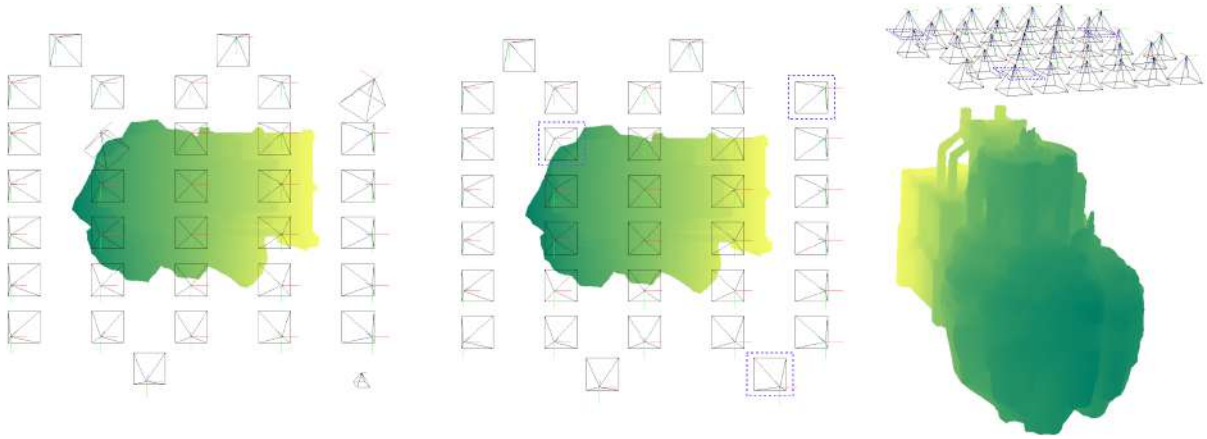

Figura 7: Exemplo de otimização incoerente gerada pelo `uavmvs` (esquerda) a partir da trajetória inicial em grade (centro). Note que apenas três *viewpoints* são alterados (com *view frustums* destacados em roxo).

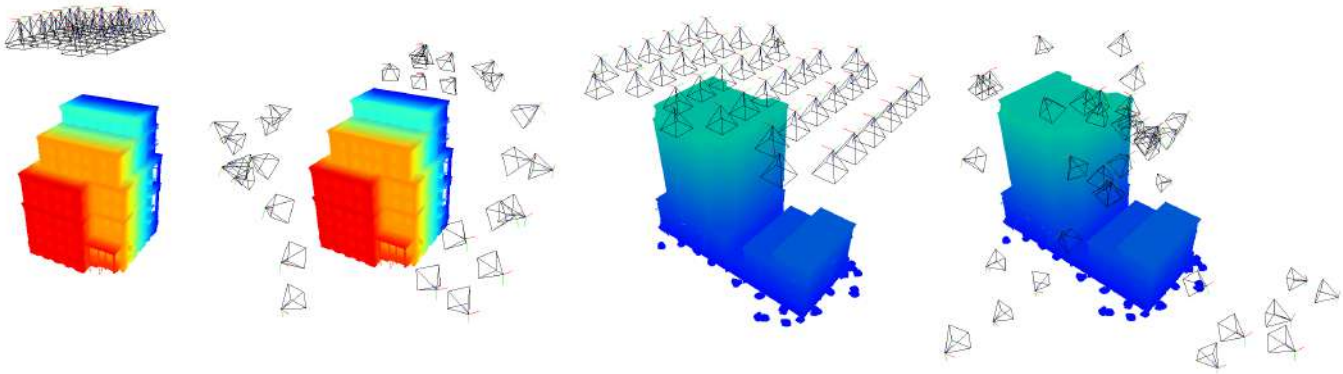

Figura 8: Trajetórias iniciais e otimizadas para os modelos `Prédio 3C` e `Prédio 7A`.

## 6 Seguir a Trajetória Planejada

Como descrito no relatório parcial [8], criamos trajetórias em grade e em hélice gerando coordenadas no sistema de referência usado pelo AirSim (que é diferente do adotado pela Unreal Engine, ilustrado na Figura 10), a partir de um sistema que desenvolvemos para controlar a região de interesse delimitando interativamente a área sobrevoada.

Já para as trajetórias produzidas pela implementação de Smith et al. (tanto as iniciais em grade como as otimizadas, conforme Figura 8) alguns passos de processamento tiveram que ser implementados. Primeiro, o `uavmvs` não tem um formato padrão para escrever as coordenadas em arquivos — na verdade, três formatos diferentes são utilizados dependendo do *output*, um deles é armazenado como CSV [31], mas os outros dois são específicos — portanto, precisamos escrever mais de um programa para fazer o *parsing* das trajetórias.

Segundo, o sistema de coordenadas adotado pelo `uavmvs` é diferente da convenção de eixos que o AirSim segue — e que, por sua vez, é diferente do usado na UE4, que também não é igual ao dos arquivos FBX

Figura 9: Exemplo de área de interesse delimitada (esquerda) sobre a qual o *drone* voa trajetórias em grade (direita).

exportados ou do Blender, que usamos para converter o formato FBX para PLY [32].

Portanto, ressaltamos que essa etapa de processamento dos dados não é simples e nem direta, pois, a cada passo, precisamos verificar se todas as nossas suposições estão corretas. Isso envolve muitas vezes a criação de formas de inspeção visual dos dados. Enquanto alguns *scripts* de visualização podem ser escritos rapidamente com a biblioteca Open3D, como sempre precisamos garantir que temos tudo funcionando de acordo com o AirSim, inevitavelmente precisamos validar dentro do próprio simulador todas as transformações que aplicamos aos dados geométricos.

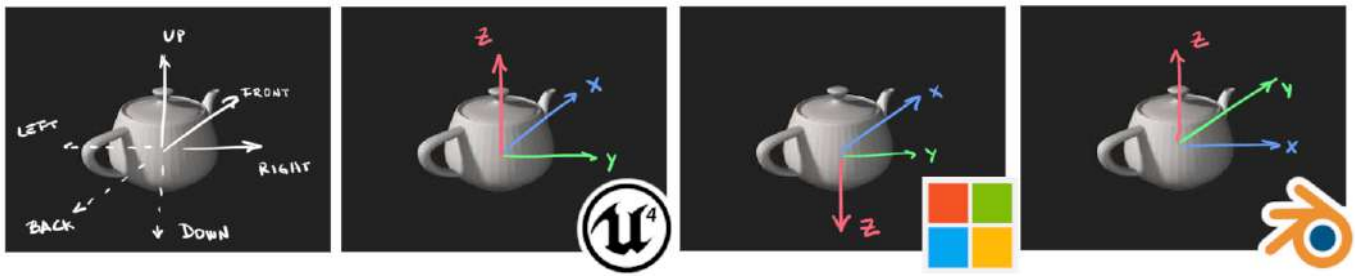


Figura 10: Diferentes convenções de eixos para os sistemas de coordenadas usados pela Unreal Engine 4, AirSim e Blender.

Por exemplo, ao mesmo tempo que a Unreal Engine usa centímetros como unidade padrão e tem um sistema de coordenadas *left-handed* (i.e. que segue a "regra da mão esquerda" [33]), o AirSim usa metros, seguindo medidas do Sistema Internacional de Unidades, e adota o quadro de referência comum aeroespacial *North-East-Down* (que é *right-handed*, com o eixo $+Z$ apontando para baixo e o eixo $+X$ para frente). Já o Blender, que usamos para converter os modelos exportados da UE4, também tem um sistema *right-handed*, mas com $+Z$ apontando para cima (o que também é pressuposto pelo `uavmvs` para *meshes*, de modo que não podemos usar um mesmo arquivo nele e no AirSim, sem antes fazer transformações). A Figura 11 apresenta o sistema NED usado pelo AirSim.

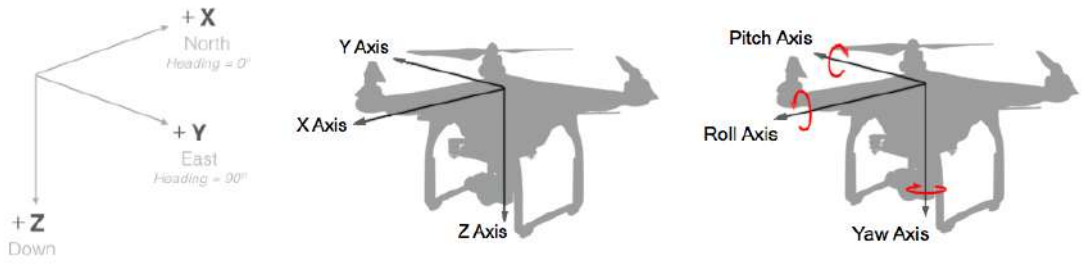


Figura 11: Sistema NED usado pelo AirSim (esquerda e centro) e ângulos de Euler (direita), que podem descrever a orientação de um *drone* em um dado sistema de coordenadas. [8]

Isso se torna ainda mais complicado devido às diferentes formas que o AirSim e o `uavmvs` usam para descrever a orientação representada por cada pose. Enquanto o AirSim usa quatérnios para expressar tanto orientações como rotações (e possui funções auxiliares para convertê-las para ângulos de Euler [34]), o `uavmvs` usa matrizes $3\times3$ para guardar orientações no formato de arquivo usado para as trajetórias inicial e otimizada.

Uma forma de entender a distinção entre orientação e rotação é que a orientação caracteriza um *destino* angular e a rotação caracteriza a *rota angular* para esse destino. Explicando melhor essa diferença: ao passo que rotações são aplicadas a vetores tridimensionais (que descrevem posições de objetos na cena, por exemplo) para alterar suas coordenadas, as orientações se comportam de forma matematicamente equivalente, porém, conceitualmente elas representam o deslocamento angular de um objeto *em relação a um quadro de referência* específico (ou seja, devem ser aplicadas aos vetores da base do espaço vetorial, ilustrados na Figura 10).

Desse modo, precisamos considerar as suposições implícitas feitas a respeito dos sistemas de coordenadas quando queremos converter uma orientação de um *software* para o outro (fora as conversões matemáticas necessárias para mudar de uma representação para outra, por exemplo, de matrizes para quatérnios e vice-versa).

## 7 Capturar Imagens Aéreas

Conforme o *drone* obtém imagens no simulador, nós as transmitimos para o Meshroom, que cria uma representação 3D da cena capturada como uma nuvem de pontos (*point cloud*). Para isso, é realizada a extração de *features* das diferentes imagens para combinar as que apresentam áreas próximas na estrutura modelada, e então as poses nas quais a câmera (acoplada ao *drone*) estava ao tirar cada foto são estimadas (Figura 12).

Como queremos aumentar a representação da cena de forma *online* (isto é, conforme obtemos mais imagens durante o voo, ilustrado na Figura 13), reconstruímos apenas a *point cloud* já que a criação de uma malha triangular com textura é muito mais custosa computacionalmente (referente às etapas destacadas em cinza no diagrama da *pipeline*, Figura 2). Porém, isso não impede que um modelo seja gerado *offline*, após o voo, passando pelas etapas de *Meshing* e *Texturing*.

[8] Figura adaptada a partir de: https://developer.dji.com/mobile-sdk/documentation/introduction/flightController_concepts.html

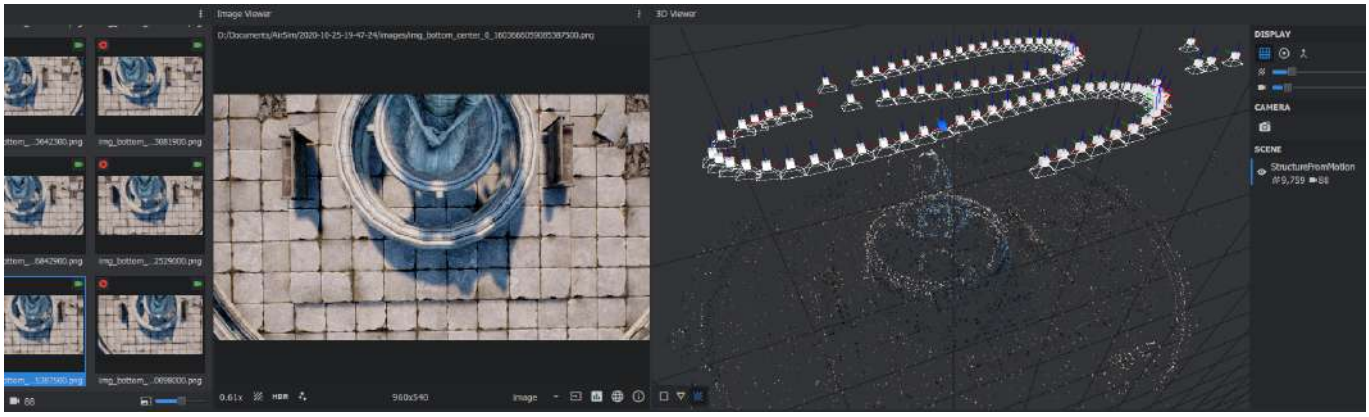


Figura 12: Reconstrução 3D efetuada pelo Meshroom, com imagens obtidas no AirSim.

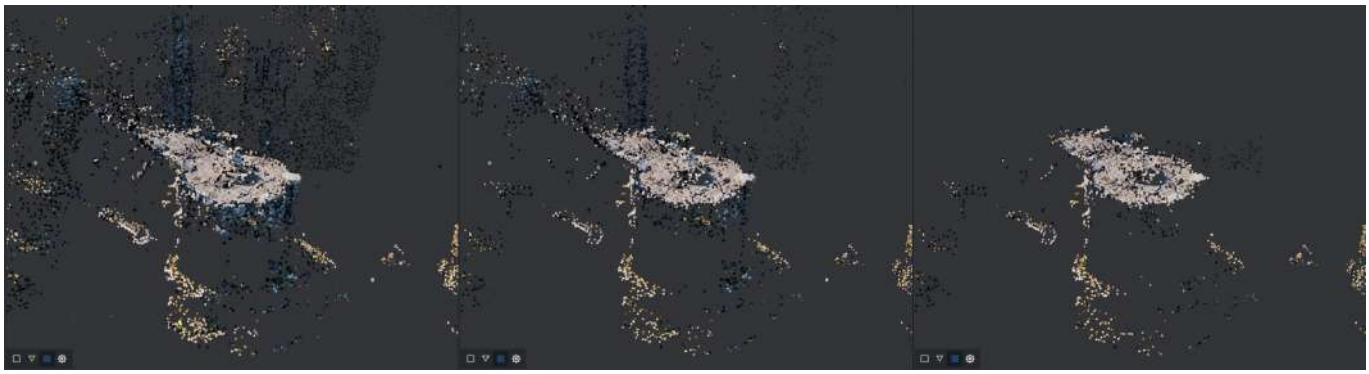

Figura 13: Reconstrução 3D efetuada incrementalmente pelo Meshroom a partir de capturas do modelo `Estátua`, em diferentes estágios (as *point clouds* à direita foram geradas com menos fotos, portanto, possuem menos pontos e não cobrem todo o modelo).

## 8 Reconstruir a Nuvem de Pontos

Uma vez que sabemos a trajetória real (*ground truth*) do *drone*, já que obtemos sua posição e orientação diretamente da simulação, podemos compará-la com a trajetória reconstruída pelo Meshroom (mostrada na direita da Figura 12).

Ressaltamos a importância desta validação pois, para que o algoritmo determine os próximos *viewpoints*, é necessário ter uma boa estimativa da pose atual do *drone*. Porém, conforme mencionamos, não podemos depender do conhecimento de valores reais para estender a solução desenvolvida para o mundo real, em condições adversas (como ambientes fechados) e usando principalmente a visão estéreo (sem GPS ou LIDAR).

O processo de alinhamento consiste inicialmente em guardar as posições obtidas pelo AirSim no formato PLY [32] (utilizado para armazenar dados tridimensionais, e para a análise com o Open3D), e também em converter a representação de dados do Meshroom. Usamos então o Open3D para fazer uma estimativa inicial da relação global entre os pontos com o algoritmo RANSAC (*RANdom SAmple Consensus* [35]), seguido do método local ICP (*Iterative Closest Point*), que tenta minimizar a diferença quadrática entre as duas trajetórias representadas como nuvens de pontos (veja a Figura 14). Ainda assim, para algumas trajetórias o resultado ainda não era satisfatório. Entretanto, uma vez que sabemos quais *viewpoints* obtidos pelo AirSim

correspondem a quais poses estimadas para as câmeras após a etapa de *Structure from Motion* executada pelo Meshroom (que gera valores em um sistema de coordenadas arbitrário), podemos usar essa informação para melhorar o alinhamento.

Já que esperamos uma transformação afim entre os sistemas, que possa ser definida por um vetor de translação, por uma rotação dos eixos, e por um único fator de escala (i.e. podemos supor que a escala será igual para os três eixos), buscamos uma transformação de Helmert [36]: $\vec{X}_{AirSim} = \vec{C} + \mu \cdot \mathbb{R} \cdot \vec{X}_{Meshroom}$. Portanto, usamos o método dos mínimos quadrados para estimar a matriz $4 \times 4$ que representa tal transformação, e por fim a refinamos aplicando ICP novamente.

Figura 14: Alinhamento dos valores reais extraídos do simulador com a trajetória reconstruída, para a cena da Figura 12. As posições estimadas pelo Meshroom (após alinhadas) são mostradas em amarelo, e o *ground truth* é mostrado em azul.

Em seguida, podemos utilizar as métricas de SLAM: ATE para medir diretamente a diferença entre os pontos estimados e os reais, e RPE para avaliar a precisão local da trajetória em um dado intervalo de tempo (o que corresponde ao desvio da trajetória, por unidade de tempo).

## 9 Analisar o Modelo Obtido

Um dos grandes desafios do projeto é o fato de não termos uma base de referência para o problema que buscamos tratar. Mesmo havendo trabalhos relacionados, e que também utilizaram ambientes simulados pela Unreal Engine [20, 18], eles não compartilham ou disponibilizam o conjunto de ambientes (muitas vezes por utilizarem *environments* pagos), suas reconstruções, ou as coordenadas em que obtiveram cada imagem.

Assim, para compor nossa *baseline* de experimentos, escolhemos um conjunto de áreas de interesse (i.e. as regiões reconstruídas) bem definidas em dois *environments* disponíveis gratuitamente pelo Unreal Engine Marketplace: *Infinity Blade: Grass Lands* [9] e *Modular Building Set* [10].

Apresentamos a região reconstruída para o *environment Grass Lands* na Figura 15 — que também foi usada em diversos trabalhos anteriores [18, 19, 20] — e nas Figuras 16 e 17, dois modelos que selecionamos do *Modular Building Set*, já que muitos trabalhos da literatura usam cenas urbanas. O conjunto desses três

[9] https://www.unrealengine.com/marketplace/infinity-blade-plain-lands
[10] https://www.unrealengine.com/marketplace/modular-building-set

modelos selecionados (`Estátua`, `Prédio 3C` e `Prédio 7A`) engloba diversos aspectos interessantes: há diversas regiões bem detalhadas e, ao mesmo tempo, de fácil acesso no mapa *Grass Lands*; em contraste, os edifícios disponíveis no *Modular Building Set* possuem uma geometria mais simples, porém, exigem que a trajetória planejada considere como o *drone* vai capturar imagens do modelo inteiro (p.ex. das fachadas internas dos edifícios no `Prédio 7A`), e nos permitem analisar os problemas que acontecem quando modelos têm diversas superfícies planas (o que contribui negativamente para o resultado da *pipeline* de fotogrametria, já que a etapa de extração de *features* não se comporta bem com fotos que possuem poucos detalhes).

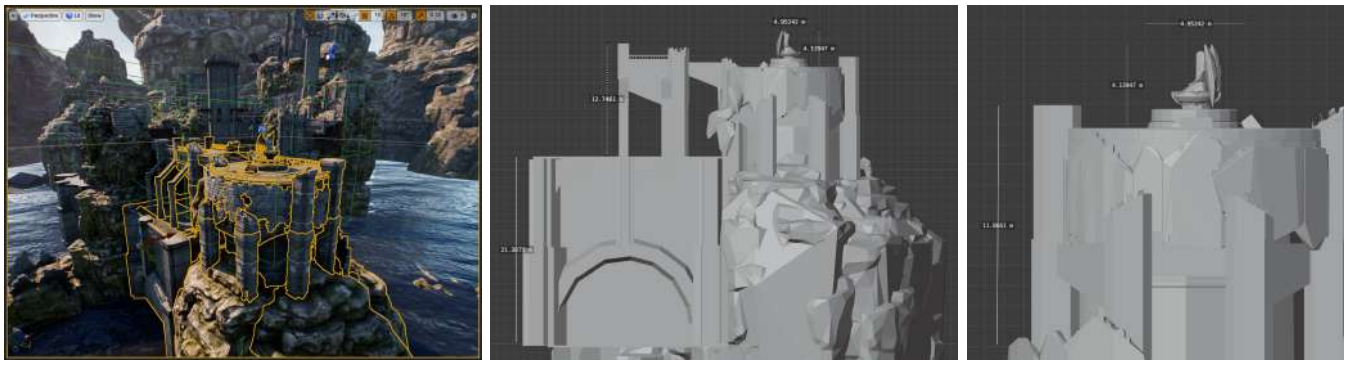

Figura 15: Modelo `Estátua` do mapa *Grasslands* na Unreal Engine 4 (esquerda) com medições feitas com o software Blender (centro e direita).

Figura 16: Modelo `Prédio 3C` na UE4 (esquerda) com medições (centro e direita).

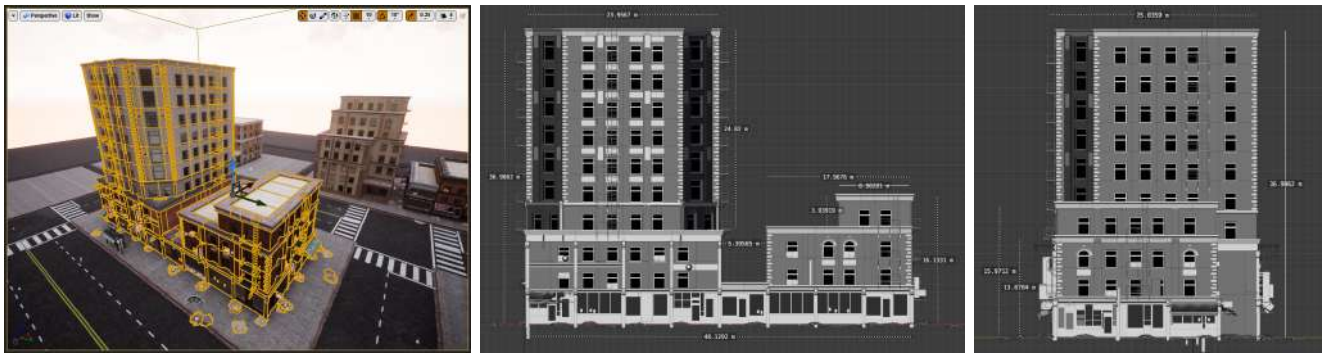

Figura 17: Modelo `Prédio 7A` na UE4 (esquerda) com medições (centro e direita).

Como não há uma metodologia bem estabelecida para a avaliação do problema estudado — tanto que boa parte da discussão encontrada na literatura pode ser considerada subjetiva, por basear-se em fatores qualitativos — buscar formas de análise quantitativas é essencial para podermos contrastar diferentes resultados de modo objetivo. Para isso, escolhemos três métricas principais:

- A qualidade final das reconstruções é avaliada segundo o procedimento proposto por Knapitsch et al. [4, p. 9] para o cálculo de precisão (*precision*) e completude (ou revocação, *recall*) de reconstruções 3D baseadas em imagens.
- *Absolute trajectory error* mede a diferença euclidiana entre os pontos da trajetória verdadeira (simulada) e da trajetória estimada (pelo Meshroom) [37].
- *Relative pose error* também pode ser usado para avaliar a qualidade da localização, determinando o erro relativo entre posição e orientação de poses correspondentes.

As duas últimas métricas — ATE e RPE [21] — são comumente utilizadas na literatura de SLAM e odometria visual [38, 22, 39] — problemas que efetivamente abordamos ao termos que reconstruir a cena e, ao mesmo tempo, localizar o *drone* nela para determinar o melhor ponto (*viewpoint*) ao qual devemos dirigi-lo para capturar imagens seguintes.

Examinando a implementação do método descrito em [4], usado por Knapitsch et al. no desenvolvimento da *benchmark Tanks and Temples*, tivemos que realizar algumas alterações para adaptá-la às ferramentas que usamos (p.ex. Meshroom), as quais descrevemos no relatório anterior [8]. Além disso, duas informações são necessárias para a avaliação da reconstrução 3D: a nuvem de pontos *ground truth* (definida pelas estruturas usadas no ambiente de simulação) e a matriz de transformação que a usa como referência para alinhar a nuvem de pontos reconstruída a partir das imagens obtidas (veja a Figura 18).

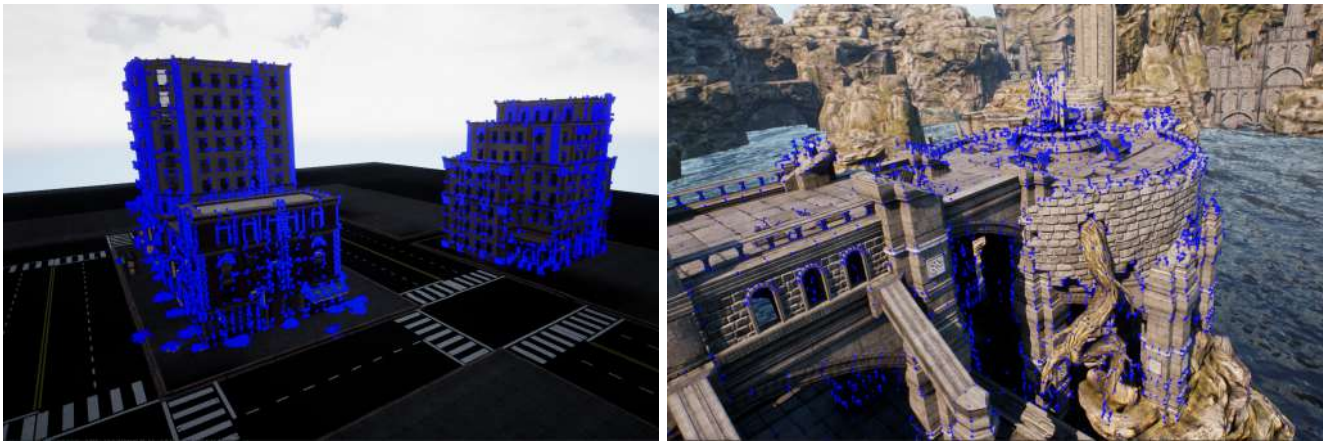

Figura 18: *Point clouds* sobrepostas aos modelos `Prédio 7A` e `Prédio 3C` (esquerda) e `Estátua` (direita). Alguns dos pontos ficam escondidos por trás da malha triangular.

Para isso, criamos as *point clouds* de referência a partir da amostragem de pontos na superfície das malhas triangulares exportadas pela UE4 e seguimos os procedimentos já descritos nas seções 6 e 8 para calcular a

transformação de alinhamento.

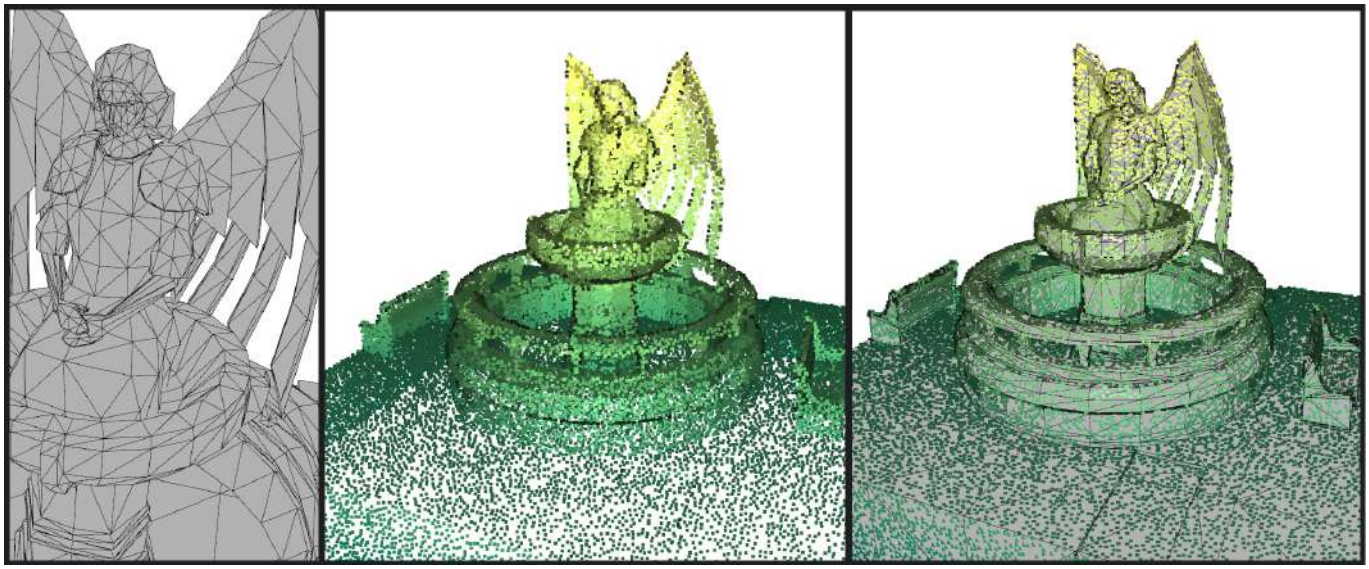

Figura 19: Malha triangular extraída da Unreal Engine (esquerda), nuvem de pontos gerada por *sampling* uniforme (centro), e sobreposição das duas representações (direita).

Já para a avaliação das métricas de SLAM, consideramos as implementações de ATE e RPE que foram empregadas no *dataset* para SLAM visual TartanAir [22] — desenvolvido por um grupo da Carnegie Mellon University — por ele ser composto de imagens também coletadas pelo AirSim. Novamente, modificações foram necessárias, porque os *scripts* publicados junto com o TartanAir eram específicos para a forma como os dados foram organizados no *dataset*. Apesar disso, conseguimos obter as trajetórias efetuadas pelo *drone* a partir do simulador e da reconstrução realizada pelo Meshroom, e então convertê-las para o formato esperado, assim obtendo valores de ATE e RPE.

## 10 Otimizar a Trajetória & Retraçar o Plano de Voo

Concluímos o relatório agora com uma apresentação dos desafios relacionados à otimização da trajetória durante o voo — com o objetivo de refinar a posição e orientação dos *viewpoints* planejados em tempo real — e também com uma discussão de como nosso projeto contribui para isso e para possíveis trabalhos futuros.

Grande parte dos trabalhos com foco nesta área aborda o problema de modo *explore-then-exploit*, ou seja, dois voos são necessários para obter uma reconstrução final. O voo inicial (fase de *exploration*) é feito seguindo uma das trajetórias comumente utilizadas por soluções comerciais (p.ex. as em grade e em hélice que exploramos), com o objetivo de capturar imagens para a criação de uma reconstrução aproximada (denominada *proxy* por alguns autores, como Smith et al. [5]). Uma vez criada, gera-se uma solução para o problema de otimização de *viewpoints* tendo essa *proxy* como base, e realiza-se então um segundo voo na cena a ser capturada, agora seguindo a trajetória que estima-se reconstruir a *proxy* de modo ótimo (fase de *exploitation*). Portanto, tanto o planejamento de voo para a trajetória de *exploration* quanto para a de *exploitation* são efetivamente realizados *offline*. O que alguns trabalhos trazem é uma forma de desvio de

obstáculos feita de modo *online*, porém a trajetória que o *drone* deve seguir é raramente alterada durante o voo. Logo, novas informações que poderiam ser extraídas das imagens que o *drone* captura durante esse segundo voo — e que possibilitariam um refinamento do modelo aproximado que se constrói da cena após o primeiro voo — são ignoradas no planejamento da trajetória.

Notamos nisso uma margem de pesquisa que podia ser aprofundada: por que não fazer ajustes à trajetória planejada em tempo real? Para isso, seria necessário calcular e extrair métricas a partir das novas imagens conforme elas são capturadas (para que elas possam influenciar o plano de voo). Outra vertente de pesquisa importante para a qual isso contribuiria é na abordagem do problema pela ótica do aprendizado por reforço, uma vez que avaliações mais eficazes (e frequentes, se executadas de forma incremental) da qualidade da reconstrução podem acelerar consideravelmente o tempo de treinamento.

Assim, nosso projeto propôs não só extensões a uma heurística estado-da-arte para que ela pudesse ser usada para a avaliação incremental (e em tempo real) da contribuição de diferentes *viewpoints* para a qualidade final das reconstruções geradas, mas também, sugerimos um conjunto de cenas e modelos que podem ser empregados por diversos trabalhos futuros (junto com métricas quantitativas para avaliar tanto a reconstrução em si, quanto a localização estimada do *drone*, que é essencial para aplicações em tempo real).

Por fim, ressaltamos a importância de considerar que a dinâmica do *drone* faz com que as poses planejadas não sejam atingidas de forma precisa (com variáveis níveis de erro, dependendo do *drone* utilizado, das condições de vento, da acurácia da localização, etc.). Ilustramos isso na Tabela 2 e nas Figuras 20 e 21, mostrando que a qualidade da reconstrução é significativamente menor quando usamos as imagens capturadas pelo *drone* ao invés das capturadas nos *viewpoints* planejados pelo `uavmvs`.

Portanto, ao propormos um método de estimar a qualidade da reconstrução em tempo real (usando *point clouds* ao invés de *meshes*), possibilitamos que ajustes sejam feitos baseados nas condições físicas reais do momento de captura.

| | **Trajetória Inicial** | **Trajetória Otimizada** |
|---|---|---|
| **Sem dinâmica de voo (ideal)** | Precision : 0.6945 | Precision : 0.7692 |
| | Recall : 0.1918 | Recall : 0.1480 |
| | F1-Score : 0.3006 | F1-Score : 0.2482 |
| **Com dinâmica de voo (*drone*)** | Precision : 0.6047 | Precision : 0.7217 |
| | Recall : 0.1517 | Recall : 0.1337 |
| | F1-Score : 0.2426 | F1-Score : 0.2256 |

Tabela 2: Valores de precisão e completude (*precision* e *recall*) calculados de acordo com o trabalho de Knapitsch et al. [4] para avaliar reconstruções feitas a partir de imagens capturadas exatamente nos *viewpoints* planejados (ideais), e compará-las com os resultados obtido usando imagens capturadas pelo *drone* (cuja dinâmica de voo leva a divergências de posição e orientação em relação às poses ideais). O valor *F1-Score* combina as métricas de precisão e completude: $\textit{F1-Score} = (2 \cdot \textit{Precision} \cdot \textit{Recall}) \,/\, (\textit{Precision} + \textit{Recall})$.

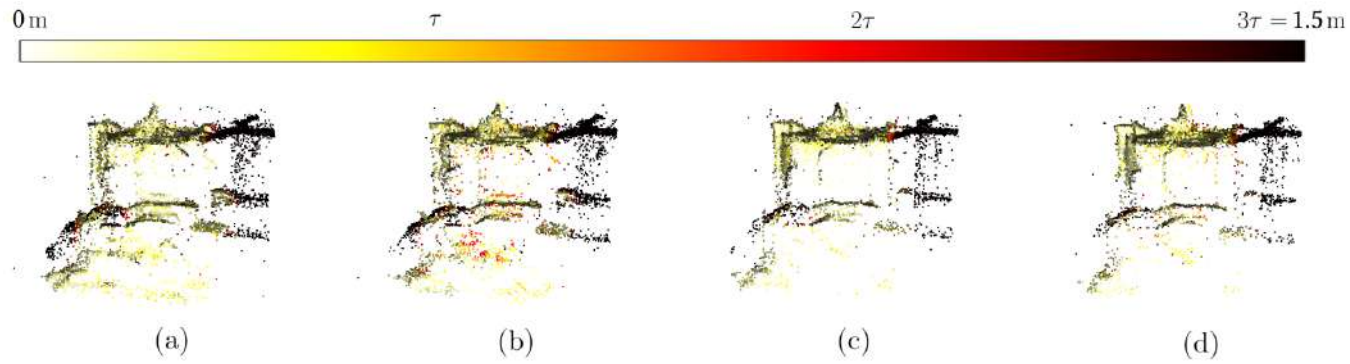


Figura 20: Mostramos o conjunto de pontos reconstruídos com sua distância em relação ao modelo *ground-truth* (i.e. *precision*) identificada pelo código de cores. As imagens correspondem aos dados apresentados na Tabela 2: (a) trajetória inicial ideal (b) trajetória inicial com *drone* (c) trajetória otimizada ideal (d) trajetória otimizada com *drone*. Note que os valores absolutos de *precision* são maiores que os de *recall* uma vez que as *point clouds* de referência possuem muito mais pontos do que as reconstruídas.

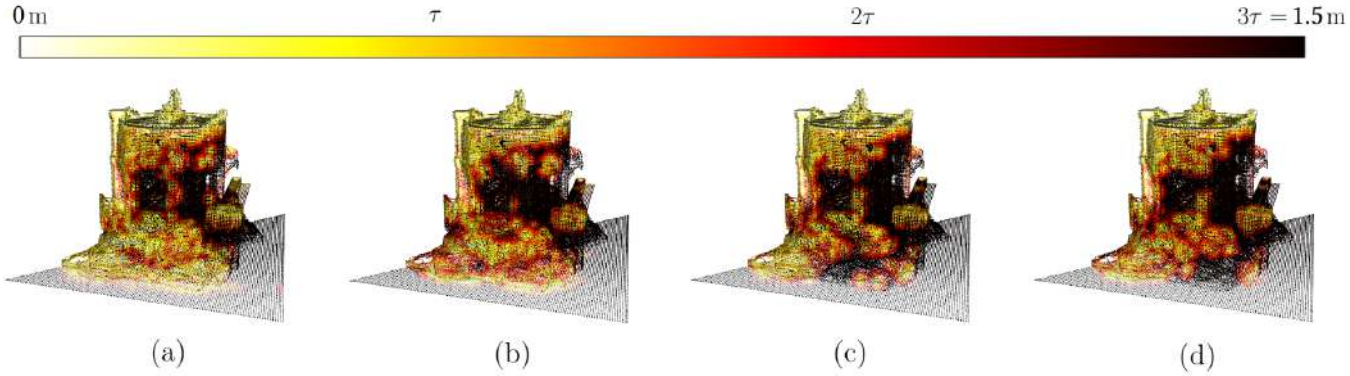


Figura 21: Mostramos o conjunto de pontos *ground-truth* com sua distância em relação ao modelo reconstruído (i.e. *recall*) identificada pelo código de cores. As imagens correspondem aos dados apresentados na Tabela 2: (a) trajetória inicial ideal (b) trajetória inicial com *drone* (c) trajetória otimizada ideal (d) trajetória otimizada com *drone*. Note uma maior concentração de pontos escuros na direita das imagens (c) e (d), que leva a menores valores de *recall* nas trajetórias otimizadas pelo `uavmvs`. Ou seja, apesar da trajetória otimizada ter resultados com maior precisão, ela não captura a cena de forma tão uniforme quanto a trajetória inicial em grade (que, portanto, apresenta maior completude, com valores superiores de *recall*).

# Referências Bibliográficas

[1] Wikipedia contributors, "3D reconstruction — Wikipedia, the free encyclopedia." [Online]. Available: https://en.wikipedia.org/wiki/3D_reconstruction 2

[2] ——, "Computer vision — Wikipedia, the free encyclopedia." [Online]. Available: https://en.wikipedia.org/wiki/Computer_vision#History 2

[3] X.-F. Han, H. Laga, and M. Bennamoun, "Image-based 3D Object Reconstruction: State-of-the-Art and Trends in the Deep Learning Era," *IEEE transactions on pattern analysis and machine intelligence*,

2019. 2

[4] A. Knapitsch, J. Park, Q.-Y. Zhou, and V. Koltun, "Tanks and temples: benchmarking large-scale scene reconstruction," *ACM Trans. Graph.*, vol. 36, pp. 78:1–78:13, 2017. 3, 5, 18, 20

[5] N. Smith, N. Moehrle, M. Goesele, and W. Heidrich, "Aerial path planning for urban scene reconstruction: A continuous optimization method and benchmark," vol. 37, 12 2018, pp. 1–15. 3, 5, 6, 7, 8, 19

[6] X. Zhou, K. Xie, K. Huang, Y. Liu, Y. Zhou, M. Gong, and H. Huang, "Offsite aerial path planning for efficient urban scene reconstruction," *ACM Transactions on Graphics*, vol. 39, pp. 1–16, 11 2020. 3, 7, 11

[7] G. Barill, N. Dickson, R. Schmidt, D. I. Levin, and A. Jacobson, "Fast Winding Numbers for Soups and Clouds," *ACM Transactions on Graphics*, 2018. 3

[8] T. L. Chaves and E. L. Colombini, "Planejamento de Trajetória de *Drones* para Reconstrução 3D – Relatório Anual," 2020. 4, 5, 6, 12, 18

[9] ——, "Planejamento de Trajetória de *Drones* para Reconstrução 3D – Proposta de Projeto," 2020. 4, 5, 6

[10] "AirSim," https://github.com/microsoft/AirSim, 2017-2019. 4

[11] I. Alzugaray, L. Teixeira, and M. Chli, "Short-term UAV path-planning with monocular-inertial SLAM in the loop," 05 2017, pp. 2739–2746. 4

[12] K. Celik, S.-J. Chung, M. Clausman, and A. Somani, "Monocular Vision SLAM for Indoor Aerial Vehicles," vol. 2013, 10 2009, pp. 1566–1573. 4

[13] Wikipedia contributors, "Simultaneous localization and mapping — Wikipedia, the free encyclopedia." [Online]. Available: https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping 4

[14] ——, "Structure from motion — Wikipedia, the free encyclopedia." [Online]. Available: https://en.wikipedia.org/wiki/Structure_from_motion 4

[15] P. Moulon, P. Monasse, and R. Marlet, "Adaptive Structure from Motion with a Contrario Model Estimation," in *Proceedings of the Asian Computer Vision Conference (ACCV 2012)*. Springer Berlin Heidelberg, 2012, pp. 257–270. 4

[16] M. Jancosek and T. Pajdla, "Multi-view reconstruction preserving weakly-supported surfaces," in *CVPR 2011*. IEEE, jun 2011. [Online]. Available: https://doi.org/10.1109/cvpr.2011.5995693 4

[17] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018. 4

[18] B. Hepp, M. Nießner, and O. Hilliges, "Plan3D: Viewpoint and Trajectory Optimization for Aerial Multi-View Stereo Reconstruction," *ACM Trans. Graph.*, vol. 38, pp. 4:1–4:17, 2017. 5, 16

[19] M. Roberts, D. Dey, A. Truong, S. Sinha, S. Shah, A. Kapoor, P. Hanrahan, and N. Joshi, "Submodular trajectory optimization for aerial 3d scanning," in *International Conference on Computer Vision (ICCV) 2017*, 2017. 5, 16

[20] C. Peng and V. Isler, "Adaptive view planning for aerial 3d reconstruction," 05 2018. 5, 16

[21] TUM Department of Informatics Computer Vision Group, "RGB-D SLAM Dataset and Benchmark," https://vision.in.tum.de/data/datasets/rgbd-dataset/tools#evaluation. 5, 18

[22] W. Wang, D. Zhu, X. Wang, Y. Hu, Y. Qiu, C. Wang, Y. Hu, A. Kapoor, and S. Scherer, "TartanAir: A Dataset to Push the Limits of Visual SLAM," 2020. 5, 18, 19

[23] N. Moehrle, "UAVMVS - UAV capture planning for MVS reconstructions," 2017. [Online]. Available: https://github.com/nmoehrle/uavmvs 7

[24] Wikipedia contributors, "Nvidia CUDA Compiler - Wikipedia, the free encyclopedia." [Online]. Available: http://en.wikipedia.org/w/index.php?title=Nvidia%20CUDA%20Compiler&oldid=1006119784 8

[25] Wikipedia, "Ground sample distance — Wikipedia, the free encyclopedia." [Online]. Available: http://en.wikipedia.org/w/index.php?title=Ground%20sample%20distance&oldid=975377013 9

[26] ——, "Ray casting — Wikipedia, the free encyclopedia." [Online]. Available: http://en.wikipedia.org/w/index.php?title=Ray%20casting&oldid=1030890302 9

[27] *Physically Based Rendering*. Elsevier. [Online]. Available: https://doi.org/10.1016/c2009-0-30446-8 9

[28] S. Katz, A. Tal, and R. Basri, "Direct visibility of point sets," vol. 26, 07 2007. 10

[29] Wikipedia, "Cosine similarity — Wikipedia, the free encyclopedia." [Online]. Available: http://en.wikipedia.org/w/index.php?title=Cosine%20similarity&oldid=1020457433 10

[30] Wikipedia contributors, "FBX — Wikipedia, the free encyclopedia." [Online]. Available: http://en.wikipedia.org/w/index.php?title=FBX&oldid=986594903 11

[31] Wikipedia, "Comma-separated values — Wikipedia, the free encyclopedia." [Online]. Available: http://en.wikipedia.org/w/index.php?title=Comma-separated%20values&oldid=1030399511 12

[32] Wikipedia contributors, "Ply (file format) — Wikipedia, the free encyclopedia," 2019. 13, 15

[33] Wikipedia, "Right-hand rule — Wikipedia, the free encyclopedia." [Online]. Available: http://en.wikipedia.org/w/index.php?title=Right-hand%20rule&oldid=1022155205 13

[34] ——, "Euler angles — Wikipedia, the free encyclopedia." [Online]. Available: http://en.wikipedia.org/w/index.php?title=Euler%20angles&oldid=1029550074 14

[35] Wikipedia contributors, "Random sample consensus — Wikipedia, the free encyclopedia," 2020. 15

[36] Wikipedia, "Helmert transformation — Wikipedia, the free encyclopedia." [Online]. Available: http://en.wikipedia.org/w/index.php?title=Helmert%20transformation&oldid=976227807 16

[37] D. Prokhorov, D. Zhukov, O. Barinova, A. Vorontsova, and A. Konushin, "Measuring robustness of visual slam," 10 2019. 18

[38] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets Robotics: The KITTI Dataset," *INT J ROBOT RES*, 2013. 18

[39] Z. Zhang and D. Scaramuzza, "A Tutorial on Quantitative Trajectory Evaluation for Visual(-Inertial) Odometry," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2018. 18

Links acessados em Junho de 2021.