



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 Anual (julho — dezembro)

Resumo

Reconstruções 3D precisas são fundamentais para aplicações em diversas áreas, desde robótica, computação gráfica e realidade virtual até medicina, geologia, agronegócio e arquitetura. Veículos aéreos não tripulados (VANTs) têm sido cada vez mais empregados para a captura de imagens aéreas à medida que modelos menores e mais acessíveis ficam disponíveis no mercado. Essas imagens podem ser utilizadas para 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. No geral, devido à falta de informações precisas (e atuais) sobre o ambiente, e também por questões de segurança, as soluções automatizadas com *drones* recorrem a voos em padrões regulares a uma distância aérea segura. No entanto, os pontos de vista capturados são, em muitos casos, insuficientes para uma reconstrução 3D de alta qualidade. Desse modo, a transição de sistemas automatizados para sistemas autônomos é fundamental. Neste cenário, este projeto propõe explorar uma forma de otimizar a trajetória de voo em tempo real, a partir, essencialmente, da visão adquirida através de câmeras RGB acopladas ao *drone*, obtendo uma reconstrução final mais precisa da área de interesse sobrevoada.

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]. Entretanto, e não por acaso, a reconstrução de cenas a partir de pontos de vista distintos continua sendo um problema não resolvido [2], 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).

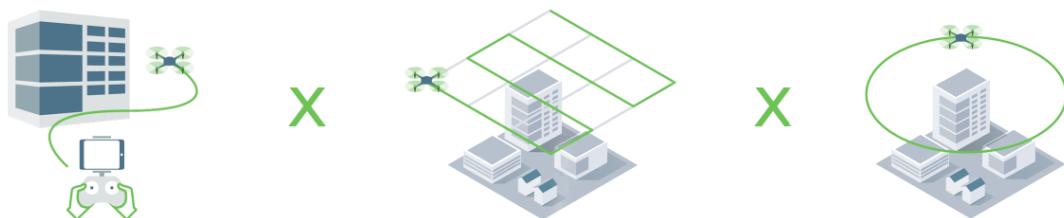


Figura 1: Ilustração esquemática do voo livre, controlado manualmente (esquerda), do voo automatizado com trajetória baseada em grade (centro), e do voo automatizado com trajetória em hélice/circular (direita)¹

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

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

algoritmos dependem de condições de cena estáveis para uma boa reconstrução (p. ex. iluminação constante), a obtenção de imagens não pode levar muito tempo, ou que ainda impossibilita que ela seja realizada em dias distintos (combinando dados).

Portanto, neste projeto estamos explorando formas de tornar o voo de *drones* para captura de imagens mais eficiente, de modo que os modelos 3D reconstruídos sejam superiores aos obtidos pela rota em grade (ou em hélice) utilizada atualmente para voos automatizados.

Assim, pretendemos desenvolver um algoritmo de planejamento de trajetória em tempo real (*real-time path planning* [3]), que iterativamente analise as nuvens de pontos que representam a reconstrução esparsa da estrutura sobrevoada, e assim determine as subsequentes poses do *drone*, para as quais ele deve se dirigir e capturar novas imagens.

Com isso, esperamos obter um modelo para o voo autônomo que otimize as reconstruções 3D realizadas. De posse destas reconstruções mais precisas é possível então diminuir custos, mas também — e principalmente — diminuir o risco associado a certas atividades de inspeção e manutenção da construção civil e do setor de energia (como em estruturas de pontes e edifícios, turbinas eólicas, refinarias de petróleo, postes de alta tensão, gasodutos, etc.), e também utilizar o modelo construído e as imagens capturadas para busca, resgate e monitoramento após desastres naturais [4, 5].

Assim, este projeto de iniciação científica tem como principais objetivos:

1. Construir uma *baseline*, avaliando a qualidade dos modelos reconstruídos a partir das estratégias mais comuns de voo autônomo utilizadas atualmente.
2. Desenvolver um algoritmo robusto a pequenos desvios de sua trajetória planejada ao sobrevoar uma estrutura.
3. Gerar incrementalmente uma nuvem de pontos da estrutura, em tempo real, conforme imagens são obtidas.
4. Analisar a nuvem de pontos para determinar que regiões devem ser capturadas a seguir, atualizando assim o plano de voo.
5. Automatizar o processo: “seguir a trajetória de voo capturando imagens da cena → reconstruir um modelo 3D → analisar a nuvem de pontos → aperfeiçoar e reformular a trajetória → seguir a trajetória de voo capturando imagens da cena”, reagindo a possíveis desvios de obstáculos.

6. Confrontar os resultados do planejador de trajetória otimizado frente à *baseline* estabelecida.

2 Realizações no Período

Para cumprir os objetivos do projeto, seguimos com o fluxo planejado para o desenvolvimento de uma solução autônoma para o problema — apresentado na Figura 2. Este fluxo prevê a definição automática de uma trajetória que se adapte à cena explorada, ao contrário das trajetórias pré-programadas oferecidas atualmente por soluções comerciais (como Pix4D, DroneDeploy e 3DR Site Scan).

Os blocos destacados em verde na Figura 2 foram abordados durante esta primeira fase do projeto. É importante esclarecer que: 1) não faz parte do escopo executar as etapas de *meshing* e *texturing* pois elas são demoradas e, portanto, não passíveis de análise em tempo real. Entretanto, uma vez que dispomos das imagens e a *Structure from Motion* (SfM [6]) correspondente, nada impede a utilização das mesmas para a reconstrução posterior de uma malha texturizada (i.e. *offline*); 2) tratamos o controlador do drone como uma caixa preta, de modo que nosso algoritmo irá ter como saída as poses seguintes que o drone deve alcançar. Desta forma, temos como objetivo otimizar a aplicação dos algoritmos de controle em um nível mais alto do sistema, através da otimização da tarefa.

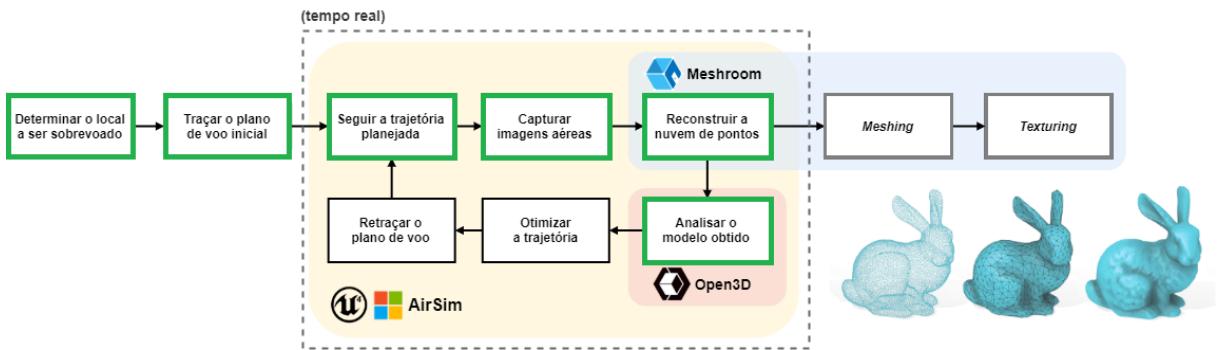


Figura 2: *Pipeline* de etapas envolvidas no processo de otimização da trajetória. Atividades iniciadas são destacadas em verde.²

Portanto, conforme apresentado na proposta do projeto [8], usamos o simulador de veículos autônomos AirSim [9], desenvolvido pelo grupo Microsoft AI & Research para realizar todos os experimentos com um *quadrotor* acoplado com cinco câmeras RGB,

²Ilustração esquemática das três fases finais da reconstrução 3D: *point cloud*, geração da malha triangular, e mapeamento de textura no modelo; utilizando imagens do coelho de Stanford (modificadas a partir de [7]).

sensores cinemáticos, LIDAR e GPS — sobre a Unreal Engine, uma *engine* de jogos reconhecida por sua fidelidade visual.

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, nos restringimos ao uso de sensores visuais e inerciais, como nos trabalhos [10, 11] de SLAM (*Simultaneous Localization And Mapping* [12]).

A partir das imagens obtidas, utilizamos a *pipeline* de *Structure from Motion* (SfM) do framework de fotogrametria AliceVision com o software Meshroom [13, 14] para iterativamente reconstruir uma nuvem de pontos que represente a estrutura tridimensional capturada, assim como as poses — posição e orientação — estimadas para a câmera a cada captura, que correspondem às posições do *drone*. Usamos então a biblioteca Open3D [15] para processar e analisar esses dados.

A seguir, descrevemos em maiores detalhes as principais atividades em que trabalhamos neste primeiro período do projeto de iniciação científica.

2.1 Análise de Trabalhos Relacionados

Continuamos acompanhando o que vem sendo desenvolvido em áreas relacionadas, e buscando estudar o que há de estado da arte nos tópicos que vamos abordando. Em particular, ressaltamos aqui alguns trabalhos desenvolvidos neste ano, pertinentes ao tema da iniciação científica.

Relevantes para o tópico de simulação, o laboratório de robótica e percepção conjunto da Universidade de Zurique e da ETH Zurique — coordenado pelo prof. Davide Scaramuzza — mostrou ser possível treinar multirotoretes em simulação e então implantar o algoritmo desenvolvido para o controle de voos acrobáticos diretamente em *drones* reais — sem *fine-tuning* (i.e. *zero-shot transfer*) — com o artigo *Deep Drone Acrobatics* [16]. Além disso, o grupo também começou a trabalhar em um novo simulador de *quadrotors*, cujo objetivo é ser altamente foto-realista e fisicamente preciso — usando a Unity, *engine* similar à Unreal Engine. Como o código desse simulador (nomeado Flightmare [17]) é aberto, nós exploramos sua implementação e fizemos pequenos testes, porém concluímos que ele ainda é muito novo e não contempla todas as *features* que o AirSim dispõe.

Já no que tange o planejamento de trajetória e movimento, dois estudos de destaque são: ATLAS [18], um método de reconstrução 3D *end-to-end* utilizando redes neurais (po-

rém não em tempo real) desenvolvido pela empresa Magic Leap; e um novo algoritmo de *path planning* em tempo real, criado por outro grupo da ETH Zurique [19]. Em particular, o método foi aplicado a dois casos relevantes para o nosso contexto: na exploração de ambiente internos desconhecidos usando *drones* e para a reconstrução de edifícios baseada na transformada de distância euclidiana com sinal truncada (*truncated signed distance function, TSDF*).

Por fim, destacamos que na última versão do Open3D, lançada em outubro deste ano, foi introduzido um novo módulo com foco em aprendizado de máquina (Open3D-ML), que oferece suporte para operadores, modelos, algoritmos e conjuntos de dados específicos de domínio 3D nesse contexto. Isso também deve facilitar os experimentos que realizaremos na segunda metade do projeto, agora que o Open3D provê suporte oficial para Pytorch e TensorFlow (as duas bibliotecas de *machine learning* mais populares atualmente).

Tais trabalhos mostram que estamos seguindo uma linha de estudo importante e ativa.

2.2 Determinar o Local a Ser Sobrevoado

Após termos uma prova de conceito para os métodos que usaremos para avaliar o erro na localização do *drone* e na reconstrução da cena — que discutiremos na seção 2.7 — outro fator importante é a reproduzibilidade dos experimentos.

Para isso, delimitamos a área que será sobrevoada por um retângulo (*bounding box*) e salvamos suas coordenadas, associadas ao *environment* simulado, para cada experimento (veja Figura 3).



Figura 3: Diferentes áreas de interesse, geradas interativamente pela alteração de um retângulo delimitador (em vermelho).

Com o objetivo de facilitar a definição e customização dessas regiões, desenvolvemos um programa com o qual podemos interativamente modificá-las (a partir de translação e escala) usando apenas o teclado, e em tempo real visualizar tais modificações e guardar arquivos que as descrevem. Logo, podemos rapidamente salvar os dados necessários, assim como efetuar alterações após cada experimento.

Isso foi feito com novas funções da API do AirSim que permitem a visualização (*plotting*) de pontos e curvas tridimensionais programaticamente, que chamamos conforme os controles do teclado são acionados.



Figura 4: Exemplo de área de interesse delimitada (esquerda) sobre a qual o *drone* voa trajetórias em grade (direita).

2.3 Traçar o Plano de Voo Inicial

A partir da descrição das áreas de interesse, implementamos um *script* para que o *drone* sobrevoe a região com uma trajetória em grade, 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 4), para efetuar diferentes experimentos — e então compará-los para podermos utilizar os melhores resultados como as *baselines* que esperamos sobrepassar com a metodologia proposta.

Relembramos que, conforme definido na proposta [8], enxergamos o controlador de voo como uma caixa-preta — no caso, utilizamos as APIs de controle multirotor do AirSim para comunicar velocidades ou (listas de) posições para o *drone* — de modo que o algoritmo proposto irá ter como saída uma lista iterativa de “próximo melhor alvo”.

Ou seja, nosso objetivo não é controlar individualmente cada motor, mas sim apontar a melhor posição seguinte para o *drone* e orientação da câmera (o próximo *viewpoint*).

2.4 Seguir a Trajetória Planejada

Além de salvarmos arquivos que descrevem diferentes regiões de interesse para serem sobrevoadas, nós armazenamos as coordenadas que representam as rotas de voo planejadas, utilizadas como *baseline*.

Porém, ao efetuarmos os primeiros experimentos, percebemos que o controlador de voo do AirSim desviava muito do trajeto esperado, principalmente quando temos curvas abruptas no caminho — o que fica visível no voo em grade (Figura 5).

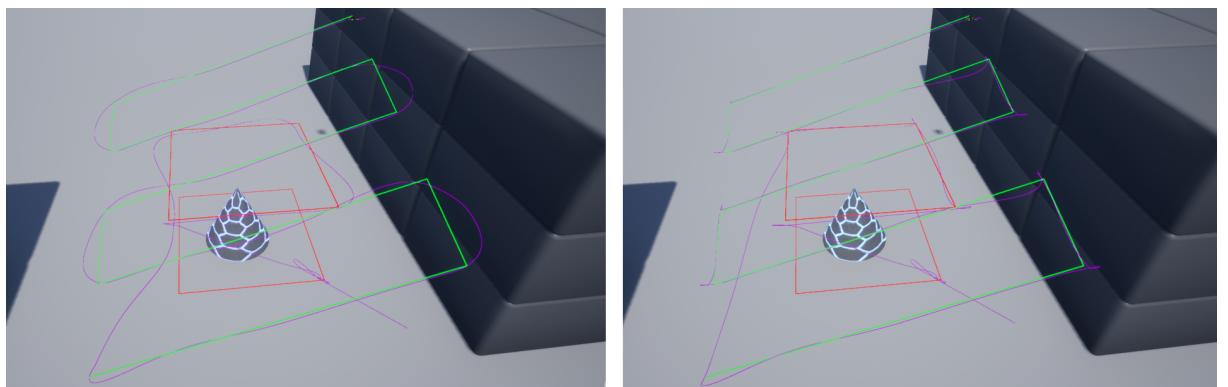


Figura 5: Comparação entre o controlador padrão do AirSim (esquerda) e o controle de voo aprimorado (direita). A linha roxa representa os caminhos traçados pelo *drone* sobre um trajeto em grade inclinada (verde), acima de uma região delimitada em vermelho.

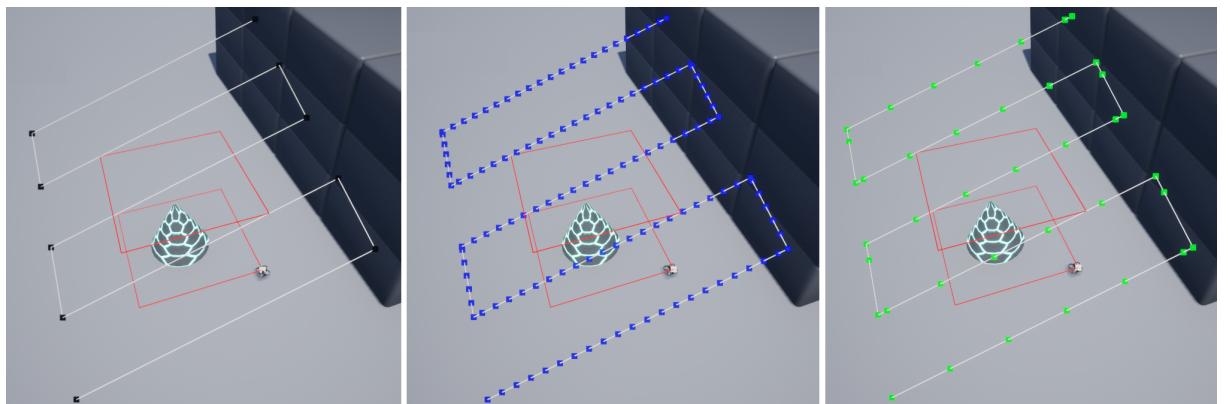


Figura 6: Pontos que definem uma trajetória em grade (esquerda), e a trajetória discretizada, adicionando diversos pontos intermediários (centro e direita).

Portanto, optamos por discretizar as trajetórias antes de comunicá-las para o controlador (Figura 6). Ou seja, embora guardemos apenas as coordenadas necessárias para

definir a trajetória — i.e. os vértices que compõem os segmentos de reta que formam a grade — criamos novos *waypoints* intermediários sobre ela, a uma distância constante entre si (a qual podemos customizar e otimizar, também programaticamente).

Isso resultou em voos mais próximos do planejado, ou seja, com menos *undershoot* e *overshoot*, e também trará mais oportunidades de reforço quando estivermos treinando um agente — já que teremos mais pontos para *rewards* (possivelmente diminuindo o tempo de treinamento).

2.5 Capturar Imagens Aéreas

Conforme o *drone* obtém imagens no simulador, nós às 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.

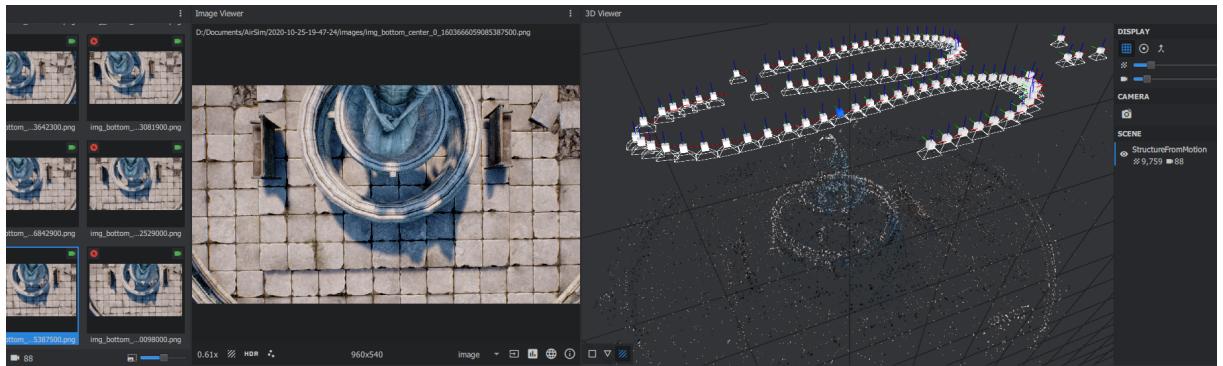


Figura 7: Reconstrução 3D efetuada pelo Meshroom, com imagens obtidas no AirSim.

Como queremos aumentar a representação da cena de forma *online* (isto é, conforme obtemos mais imagens durante o voo), 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*.

2.6 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 7).

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 se quisermos estender a solução desenvolvida para o mundo real, em condições adversas (p.ex. em ambientes internos) e usando principalmente a visão estéreo (i.e. sem GPS ou LIDAR).

O processo de alinhamento consiste inicialmente em guardar as posições obtidas pelo AirSim no formato PLY [20] (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* [21]), 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 9).

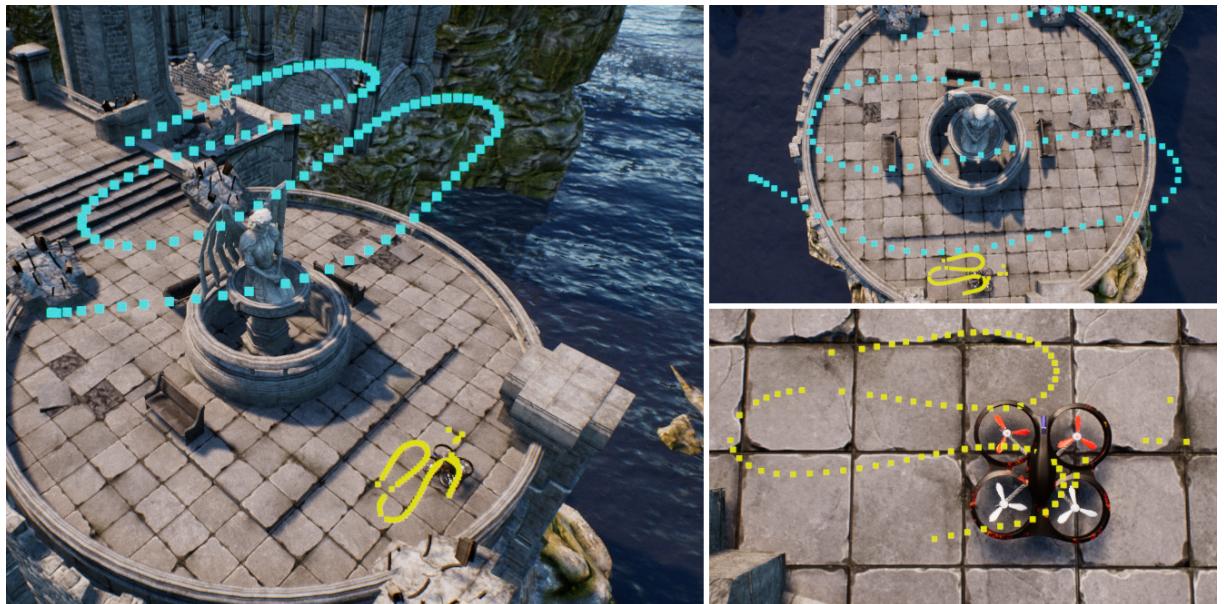


Figura 8: Trajetória real do *drone* em azul, e trajetória reconstruída — antes do alinhamento — em amarelo.

Em seguida, podemos utilizar as métricas: ATE para medir diretamente a diferença



Figura 9: Alinhamento dos valores reais extraídos do simulador com a trajetória reconstruída, para a cena da Figura 7. As posições estimadas pelo Meshroom (após alinhadas) são mostradas em amarelo, e o *ground truth* é mostrado em azul.

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). Mais detalhes serão apresentados nas seções 2.7 e 3, a seguir.

2.7 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 [22, 23], 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, uma das grandes contribuições deste projeto será a elaboração de um conjunto de áreas de interesse bem definidas em *environments* (i.e. as regiões reconstruídas), juntamente com a análise dos resultados obtidos pelas soluções mais comuns atualmente (vide Figura 1), para possibilitar comparações futuras com essa *baseline*.

Portanto, formas de análise quantitativas são essenciais para que possamos contrastar diferentes resultados, de modo objetivo. Para isso, escolhemos três métricas principais para a *baseline*:

- A qualidade final das trajetórias será avaliada segundo o procedimento proposto por Knapitsch et al. [24, p. 9] para o cálculo de precisão (*precision*) e revocação (*recall*) de reconstruções 3D baseadas em imagens.
- *Absolute trajectory error* (ATE).
- *Relative pose error* (RPE).

Os dois últimos métodos (ATE e RPE [25]) são métricas comumente utilizadas para a avaliação do erro associado à trajetória estimada na literatura de SLAM e odometria visual [26, 27, 28] — 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.

Para evitar problemas posteriores — quando fossemos avaliar os resultados da *baseline* — decidimos testar as métricas propostas logo no começo do projeto, para garantir que conseguíramos rodar as implementações que escolhemos como referência.

Examinando a implementação do método descrito em [24], usado por Knapitsch et al. no desenvolvimento da *benchmark Tanks and Temples*, tivemos que realizar algumas alterações pois eles trabalharam com outro software de fotogrametria (COLMAP [29]). Por exemplo, foi necessário adaptar o tratamento da entrada para o formato de arquivo específico utilizado pelo Meshroom para descrição dos dados (Figura 10).

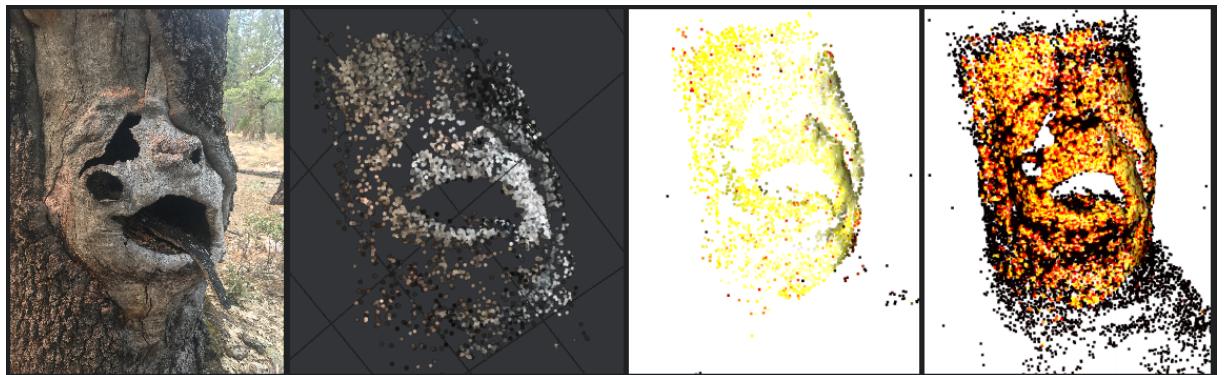


Figura 10: Exemplo de imagem e *point cloud* reconstruída no Meshroom (esquerda), usada para o cálculo de *precision* e *recall* proposto por Knapitsch et al. [24].

Além disso, duas outras informações necessárias para a avaliação da reconstrução 3D são: a nuvem de pontos *ground truth* (i.e. definida pelas estruturas usadas no ambiente de simulação), e a matriz de transformação que alinha a nuvem de pontos reconstruída a partir das imagens obtidas à essa *ground truth* (como mostramos nas Figuras 8 e 9).

Tendo os resultados das métricas de alinhamento das trajetórias discutidas na seção 2.6, passaremos a analisar também o alinhamento entre a nuvem de pontos gerada pelo Meshroom — que é uma estimativa da estrutura capturada — e o modelo real do simulador. Entretanto, como a Unreal Engine representa tais modelos como malhas triangulares (*triangular meshes*) — e não na forma de *point clouds* — já começamos a extrair as áreas que usaremos para a *baseline* no formato de dados utilizado pelo simulador, e então a

convertê-las para nuvens de pontos. Realizamos isso a partir da amostragem (*sampling*) uniforme de pontos sobre a superfície dos triângulos que compõem a malha (Figura 11).

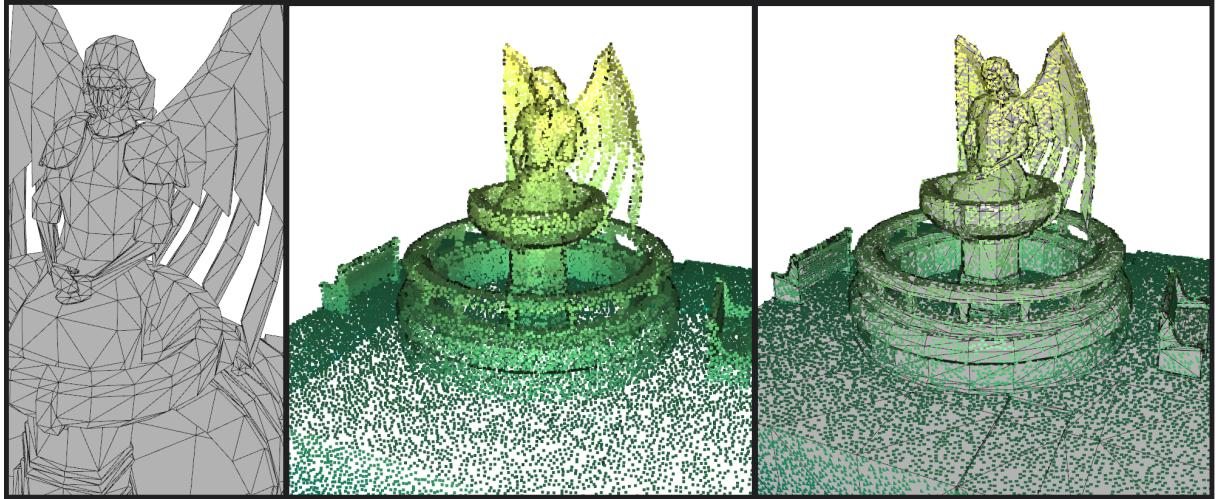


Figura 11: Malha triangular extraída da Unreal Engine (esquerda) para a estátua apresentada nas Figuras 7, 8 e 9, nuvem de pontos gerada por *sampling* uniforme (centro), e sobreposição das duas representações (direita).

Já para a avaliação das métricas de SLAM, consideramos as implementações de ATE e RPE que foram empregadas no *dataset* para SLAM visual TartanAir [27] — 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.

2.8 Melhorias no *Pipeline* de Experimentos

Conforme desenvolvíamos diferentes programas para testes com o *drone*, percebemos que muito do *setup* necessário para realizar os experimentos era repetitivo e/ou manual, de uma forma que estava nos atrasando mais do que o necessário.

Assim, decidimos investir tempo dessa primeira metade do projeto na automatização das diversas etapas envolvidas no processo de experimentação.

Dois fatores que desaceleravam este processo, em especial, são: o fato de que a API do AirSim espera que um *environment* da Unreal Engine já tenha sido lançado e esteja

executando, e que as configurações (p.ex. as câmeras utilizadas, o veículo simulado, sua posição inicial, etc.) são feitas em arquivos no formato JSON, que é lido de um diretório e arquivo padrão (nomeado `settings.json`).

Essas pressuposições feitas pelo AirSim faziam com que, para cada ambiente diferente que quiséssemos simular (ou mesmo se quiséssemos explorá-lo no modo interativo que o simulador dispõe, ao invés do modo *multirotor*), precisássemos lembrar qual o *environment* correto que deveríamos executar, (re)iniciá-lo manualmente, e combiná-lo com as configurações corretas — o que também envolvia manter diferentes arquivos JSON para cada teste, para podermos reproduzí-los. Além de tornar todo o desenvolvimento mais lento, isso também era bastante sujeito a erros.

Portanto, implementamos uma nova camada em cima da API do AirSim para que pudéssemos disparar programaticamente os ambientes corretos para cada *script* de teste, usando chamadas para a interface de linha de comando da Unreal Engine, em conjunto com o conteúdo do arquivo `settings.json`. Para isso, criamos uma classe que representa todos os valores que customizamos em nossos testes, e que automaticamente gera o conteúdo do arquivo (veja Figura 12) — logo, não precisamos mais manter diferentes JSONs, porque isso também faz com que todo o *setup* já fique contido em um único arquivo, que pode ser facilmente versionado.

```

30  if args.env_name is not None:
31      ff.launch_env(
32          *ff.LaunchEnvArgs(args),
33          settings=ff.settings_str_from_dict(
34              AirSimSettings(
35                  clock_speed=args.clock,
36                  sim_mode=ff.SimMode.Multirotor,
37                  view_mode=ff.ViewMode.SpringArmChase,
38                  subwindows=[
39                      Subwindow(0, ff.CameraName.front_center),
40                      Subwindow(1, ff.CameraName.back_center),
41                      Subwindow(2, ff.CameraName.bottom_center),
42                  ],
43                  ).as_dict()
44          ),
45      )
46      ff.input_or_exit("\nPress [enter] to connect to AirSim ")

```

```

{
    "SettingsVersion": 1.2,
    "SimMode": "Multirotor",
    "ViewMode": "SpringArmChase",
    "SubWindows": [
        {
            "WindowID": 0, "ImageType": 0,
            "CameraName": "front_center"
        },
        {
            "WindowID": 1, "ImageType": 0,
            "CameraName": "back_center"
        },
        {
            "WindowID": 2, "ImageType": 0,
            "CameraName": "bottom_center"
        }
    ]
}

```

Figura 12: Exemplo de uso da API que criamos para automatizar a execução de *environments* da Unreal Engine e a gerar o arquivo `settings.json` (mostrado na direita).

Com isso, esperamos recuperar esse tempo na segunda metade do projeto, agora que muitos processos já estão automatizados e facilmente repetíveis, reduzindo significativamente o tempo necessário para configurar o treinamento para a aprendizagem por reforço da solução.

3 Plano de Atividades

3.1 Cronograma

Na Tabela 1 apresentamos o cronograma de 12 meses proposto no início do projeto, seguido do planejamento atualizado na Tabela 2.

Atividades	Meses					
	1-2	3-4	5-6	7-8	9-10	11-12
Revisão bibliográfica	★	★	★	★	★	
Configuração do <i>pipeline</i> de experimentos	★	★		★		
Criação de uma <i>baseline</i>	●	●				
Prevenção de obstáculos		●	●	●		
Atualização de trajetória a partir da nuvem de pontos		●	●	●	●	
Voo livre				●	●	●
Disseminação de resultados					●	●

Tabela 1: Cronograma inicial do projeto. (atividades: ★ iniciadas; ● futuras)

Atividades	Meses					
	1-2	3-4	5-6	7-8	9-10	11-12
Revisão bibliográfica	✓	✓	★	★	★	
Configuração do <i>pipeline</i> de experimentos	✓	✓	★	★		
Criação de uma <i>baseline</i>	✓	✓	★			
Prevenção de obstáculos		★	★	●	●	
Atualização de trajetória a partir da nuvem de pontos		✓	★	●	●	
Voo livre				★	●	●
Disseminação de resultados					●	●

Tabela 2: Cronograma atualizado. (atividades: ★ iniciadas; ● futuras; ✓ concluídas)

Revisão bibliográfica: Esta etapa consiste em nos manter atualizados com o estado-da-arte em trabalhos relacionados ao projeto, como discutido na seção 2.1.

Configuração do *pipeline* de experimentos: No primeiro período, preparamos as ferramentas utilizadas no projeto — AirSim (simulador), Meshroom (reconstrução 3D), Open3D (análise dos dados tridimensionais) — para automatizar os experimentos, integrando as APIs Python providas de forma a acelerar o processo de: desenvolvimento → teste → avaliação → desenvolvimento. Esta etapa também envolve o uso e integração de bibliotecas de aprendizado de máquina, no segundo período do projeto.

Criação de uma *baseline*: Partimos de trajetórias simples, pré-programadas, comumente utilizadas (p. ex. em grade e em hélice, conforme Figura 1), coletando imagens de um conjunto de cenas simuladas na Unreal Engine para podermos utilizá-las como base de referência para avaliar nosso trabalho, comparando a qualidade das reconstruções feitas (conforme descrito na seção 2.7). Agora, como apresentaremos no início da seção 3.2 a seguir, concluiremos esta etapa estabelecendo de 4 a 5 áreas em diferentes *environments*, e as métricas de erro para: a reconstrução 3D final de cada uma (avaliada conforme [24]) e o movimento estimado da câmera contra a verdadeira trajetória do *drone* (avaliando os valores de ATE e RPE).

Prevenção de obstáculos: Em conjunto com a etapa seguinte, passaremos a trabalhar no desenvolvimento de um algoritmo inteligente (i.e. com técnicas de aprendizado de máquina) para atualizar o plano de voo, de modo a otimizar a reconstrução da cena sobrevoada. Usando a modelagem do aprendizado por reforço — e ainda seguindo um plano de voo predeterminado como base — introduziremos obstáculos de modo que será necessário que o agente passe a desviar do caminho planejado para evitá-los.

Atualização da trajetória a partir da nuvem de pontos: Nesta etapa iniciaremos a análise em tempo real da nuvem de pontos reconstruída — pelo Meshroom a partir das imagens coletadas no AirSim — com um algoritmo de aprendizado por reforço, cujo objetivo é determinar as próximas regiões da estrutura sobrevoada que devem ser capturadas (de modo a aprimorar a reconstrução 3D final). Para tanto, já concluímos neste primeiro período o *setup* para geração e avaliação das nuvens de ponto, como descrito em seções anteriores, que será utilizada para otimizar a rede neural que irá determinar as ações o agente (*drone*).

Voo livre: Exploraremos como tornar o modelo implementado mais geral, removendo a limitação imposta por termos que prover um plano de voo inicial e usando ambientes mais desafiadores (p.ex. como os tratados pela etapa “Prevenção de obstáculos”). Assim, pretendemos ter ao final desta fase um algoritmo que seja inteiramente responsável por traçar a rota de voo do *drone*, tendo conhecimento apenas de uma caixa delimitadora (*bounding box*) da área de interesse.

Disseminação de resultados: Na etapa final do projeto submeteremos artigos científicos

cos (e relatórios técnicos) para conferências de relevância na área, documentaremos os algoritmos desenvolvidos, e divulgaremos o trabalho desenvolvido — já publicado de forma aberta (*open-source*).

3.2 Próximos passos

A segunda metade do projeto será focada principalmente no desenvolvimento da solução inteligente para o planejamento de trajetórias. Abordaremos o problema de otimização das reconstruções 3D pela ótica do aprendizado por reforço.

Assim, após finalizarmos as avaliações do erro de localização da câmera na cena, e da modelagem das estruturas (vide seção 2.7), fixaremos as cenas que irão compor nossa *baseline*.

Sob essa base de referência, executaremos todos os testes futuros e compararemos os resultados do nosso agente (i.e. o *drone*, no contexto de *reinforcement learning*), explorando como aprimorá-lo para que ele cumpra os objetivos apresentados na seção 1.

Referências Bibliográficas

- [1] Wikipedia contributors, “3D reconstruction — Wikipedia, the free encyclopedia.” [2](#)
- [2] 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](#)
- [3] Wikipedia contributors, “Real-time path planning — Wikipedia, the free encyclopedia.” [3](#)
- [4] M. Erdelj, E. Natalizio, K. Chowdhury, and I. Akyildiz, “Help from the Sky: Leveraging UAVs for Disaster Management,” *IEEE Pervasive Computing*, vol. 16, pp. 24–32, 01 2017. [3](#)
- [5] A. Restas, “Drone Applications for Supporting Disaster Management,” *World Journal of Engineering and Technology*, vol. 03, pp. 316–321, 01 2015. [3](#)
- [6] Wikipedia contributors, “Structure from motion — Wikipedia, the free encyclopedia.” [4](#)

- [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. [4](#)
- [8] T. L. Chaves and E. L. Colombini, “Planejamento de trajetória de *Drones* para reconstrução 3d,” 2020. [4](#), [7](#)
- [9] “AirSim.” <https://github.com/microsoft/AirSim>, 2017-2019. [4](#)
- [10] I. Alzugaray, L. Teixeira, and M. Chli, “Short-term UAV path-planning with monocular-inertial SLAM in the loop,” pp. 2739–2746, 05 2017. [5](#)
- [11] K. Celik, S.-J. Chung, M. Clausman, and A. Somani, “Monocular Vision SLAM for Indoor Aerial Vehicles,” vol. 2013, pp. 1566–1573, 10 2009. [5](#)
- [12] Wikipedia contributors, “Simultaneous localization and mapping — Wikipedia, the free encyclopedia.” [5](#)
- [13] 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)*, pp. 257–270, Springer Berlin Heidelberg, 2012. [5](#)
- [14] M. Jancosek and T. Pajdla, “Multi-view reconstruction preserving weakly-supported surfaces,” in *CVPR 2011*, IEEE, jun 2011. [5](#)
- [15] Q.-Y. Zhou, J. Park, and V. Koltun, “Open3D: A modern library for 3D data processing,” *arXiv:1801.09847*, 2018. [5](#)
- [16] E. Kaufmann, A. Loquercio, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, “Deep drone acrobatics,” *RSS: Robotics, Science, and Systems*, 2020. [5](#)
- [17] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, “Flightmare: A flexible quadrotor simulator,” 2020. [5](#)
- [18] Z. Murez, T. van As, J. Bartolozzi, A. Sinha, V. Badrinarayanan, and A. Rabinovich, “Atlas: End-to-end 3d scene reconstruction from posed images,” in *ECCV*, 2020. [5](#)
- [19] L. Schmid, M. Pantic, R. Khanna, L. Ott, R. Siegwart, and J. Nieto, “An efficient sampling-based method for online informative path planning in unknown environments,” *IEEE Robotics and Automation Letters*, vol. 5, pp. 1500–1507, April 2020. [6](#)
- [20] Wikipedia contributors, “Ply (file format) — Wikipedia, the free encyclopedia,” 2019. [10](#)
- [21] Wikipedia contributors, “Random sample consensus — Wikipedia, the free encyclopedia,” 2020. [10](#)

- [22] C. Peng and V. Isler, “Adaptive view planning for aerial 3d reconstruction,” 05 2018. [11](#)
- [23] 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. [11](#)
- [24] 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. [11, 12, 16](#)
- [25] TUM Department of Informatics Computer Vision Group, “RGB-D SLAM Dataset and Benchmark.” <https://vision.in.tum.de/data/datasets/rgbd-dataset/tools#evaluation>. [12](#)
- [26] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets Robotics: The KITTI Dataset,” *International Journal of Robotics Research (IJRR)*, 2013. [12](#)
- [27] 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. [12, 13](#)
- [28] 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. [12](#)
- [29] J. L. Schönberger and J.-M. Frahm, “Structure-from-motion revisited,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. [12](#)

Links acessados em Dezembro de 2020.