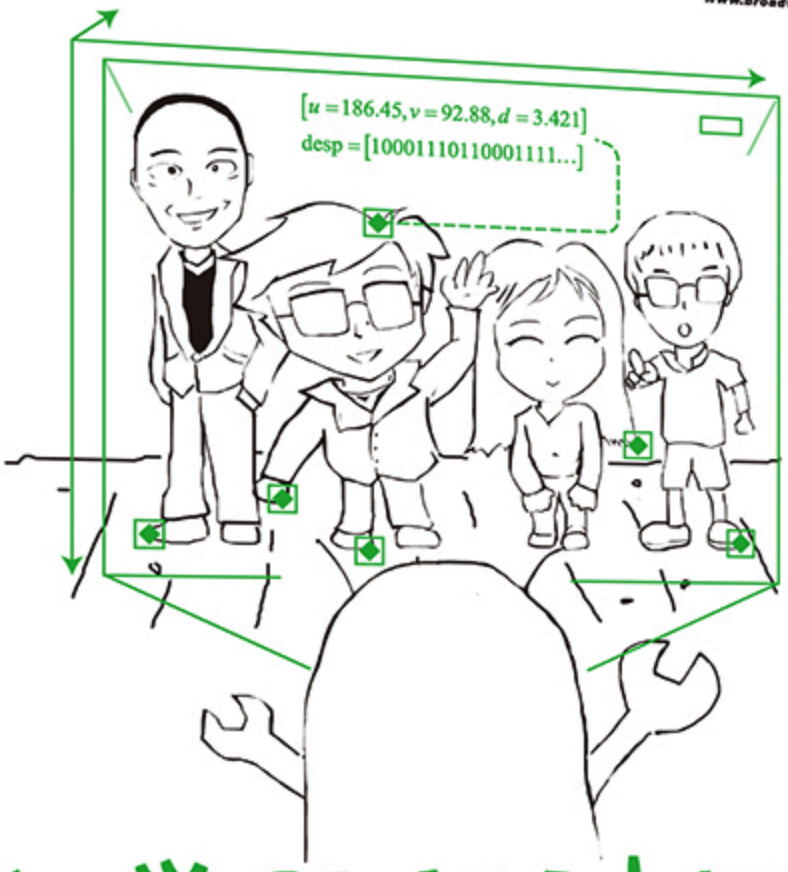
Broadview
www.broadview.com.cn



视觉 SLAM 十四讲

从理论到实践

高翔　张涛　等著

中国工信出版集团　電子工業出版社

视觉 SLAM 十四讲 从理论到实践

Preface

이 문서는 중국어 원서인 “입문 Visual SLAM 이론에서 연습까지 14 강(视觉SLAM十四讲 从理论到实践)” 책의 원저자로부터 한글 번역 허가를 받고 구글 번역기를 이용하여 작성된 문서입니다. 본 문서는 아래의 Contribution을 특징으로 합니다.

- 중국어 전공 서적을 구글 번역기를 이용해 한글로 초벌 번역했습니다.
- 초벌 번역 후 매끄럽지 않은 문장은 문맥에 맞게 수정되었습니다.
- 문서 내용 중 참고할만한 웹문서를 코멘트로 추가했습니다.
- SLAM 연구에서 주로 사용되는 용어는 한글로 번역된 용어보다 주로 사용되는 영어로 된 용어 그대로 표시하였습니다.

그럼에도 불구하고 부정확하거나 매끄럽지 않은 부분이 있을수 있습니다. 그런 부분은 코멘트로 제안해주시면 반영하도록 노력하겠습니다. 또한 읽으시다가 잘 이해가 가지 않는 부분도 코멘트로 질문해주시면 답변해드리도록 하겠습니다.

번역 참가자:

신동원 (광주과학기술원 박사과정)
김선호 (VIRNECT 선임연구원)
조원재 (일본국립농업기술혁신공학센터 연구원)
장형기 (Imperial College London 석사과정)
박준영 (광주과학기술원 석사과정)

2018년 10월 1일
신동원 드림

제13장. Mapping

주요 목표
1. 단안 SLAM에서 조밀 깊이 예측의 원리를 이해합니다.
2. 실험을 통해 우리는 단안의 조밀한 재구성 과정을 이해할 수 있습니다.
3. 여러 가지 RGB-D 재구성에서 지도 양식에 대해 배웁니다.

이 강의에서는 매핑 부분의 알고리즘을 소개하기 시작합니다. Frontend와 backend에서 우리는 카메라의 궤적과 특징점의 공간적 위치를 동시에 추정하는 문제에 초점을 맞추었습니다. 그러나 SLAM을 실제로 사용하는 경우 카메라를 배치하는 것 외에도 다른 많은 요구 사항이 있습니다. 예를 들어, 로봇 응용에서의 SLAM을 고려할 때, 우리는 위치, 탐색, 장애물 회피 및 상호 작용을 위해 맵을 사용하고자 하며, sparse한 피쳐 포인트 맵은 분명히 이러한 모든 요구를 충족시키지 못합니다. 따라서 이 강의에서는 다양한 형태의 지도에 대해보다 자세히 논의하고 현재 Visual SLAM지도의 단점을 지적합니다.

13.1 개요

매핑은 SLAM의 두 가지 주요 목표 중 하나로 간주됩니다. SLAM은 동시적 위치 추정 및 매핑이라고 불리기 때문입니다. 그러나 지금까지는 특징 지점의 위치 추정, Direct method의 위치 추정 및 Backend optimization를 포함하여 위치 추정 문제에 대해 논의했습니다. 그런데 이것은 SLAM에서 지도의 구성이 중요하지 않음을 의미할까요?

그렇지 않습니다. 사실, 고전적인 SLAM 모델에서 모든 랜드 마크 포인트의 집합인 맵을 호출합니다. 랜드 마크 포인트의 위치가 결정되면 매핑을 완료했다고 말할 수 있습니다. 위에서 언급한 Visual odometry와 BundleAdjustment는 실제로 점의 위치를 모델링하고 이를 최적화합니다. 이러한 관점에서 우리는 이미 매핑 문제를 탐구 해 왔습니다. 그렇다면 지도를 별도로 설명해야하는 이유는 무엇입니까?

이는 사람들이 맵을 그릴 필요가 있기 때문입니다. 저수준 기술로서 SLAM은 종종 상위 계층 응용 프로그램에 정보를 제공하는 데 사용됩니다. 상위 계층이 로봇 인 경우 응용 프로그램 계층의 개발자는 전역 위치 추정에 SLAM을 사용하고 로봇이 지도를 통해 탐색하도록 할 수 있습니다. 예를 들어 청소 로봇은 청소 작업을 완료해야 하며 계산이 전체지도를 포괄 할 수 있기를 바랍니다. 대안 적으로, 상위 계층이 증강 현실 장치 인 경우, 개발자는 가상 객체를 실제 객체에 중첩하고자하고, 특히 가상 객체와 실제 객체 간의 교합 관계를 처리 할 필요가있을 수있다.

응용 프로그램 수준에서 "위치 추정"과 비슷한 요구 사항이 있음을 알았습니다. SLAM은 카메라 또는 주 본체의 공간적 자세 정보를 제공하기를 원합니다. 지도의 경우 다양한 요구가 있습니다. Visual SLAM 관점에서 볼 때 매핑은 "포지셔닝"을 제공하는 것이지만 응용 프로그램 수준에서 매핑은 분명히 다른 많은 요구 사항을 가지고 있습니다. 지도의 유용성과 관련하여 다음을 대략적으로 요약합니다.

1. 포지셔닝. 위치 추정은 지도의 기본 기능입니다. 이전의 visual odometry 섹션에서는 로컬 지도를 사용하여 위치를 파악하는 방법에 대해 설명했습니다. Loop closure detection 섹션에서는 하위 정보에 대한 전체적인 설명이 있는 한 Loop closure detection을 통해 로봇의 위치를 결정할 수도 있습니다. 또한 로봇이 시작될 때마다 완전한 SLAM을 다시 수행하는 대신 맵을 한 번만 모델링하면되므로 다음 부팅 후 로봇을 맵에 배치 할 수 있도록 맵을 저장할 수 있기를 원합니다.
2. 탐색. 탐색은 로봇이 지도에서 경로를 계획하고 지도에서 두 점 사이의 경로를 찾은 다음 목표 지점으로 이동하는 것을 제어 할 수 있는 프로세스를 말합니다. 이 과정에서 최소한 지도가 도달 할 수 없는 곳과 받아 들여질 수 있는 곳을 알아야합니다. 이것은 sparse 피쳐 맵의 범위를 벗어나며 다른 형태의 맵을 가져야합니다. 우리는 나중에 이것이 최소한 dense 지도라고 말할 것입니다.
3. 장애물 회피. 장애물 회피는 또한 로봇이 종종 겪게되는 문제입니다. 이는 네비게이션과 비슷하지만, 보다 역동적인 장애물을 다루는데 중점을 둡니다. 마찬가지로, 특징점만으로는 특징점이 장애물인지 여부를 알 수 없으므로 밀도가 높은 지도가 필요합니다.
4. 3차원 복원. 때로는 주변 환경의 재구성 효과를 얻고 다른 사람들에게 보여주기 위해 SLAM을 사용하려고합니다. 이 지도는 주로 사람들을 보여주기 위해 사용되므로 편안하고 아름답게 보이기를 바랍니다. 또는 지도를 사용하여 다른 사람들이 재구성 한 3차원 개체 또는 장면 (예 : 3차원 화상 통화 또는 온라인

쇼핑)을 원격으로 볼 수 있습니다. 이지도는 또한 밀도가 높으며 모양에 대한 몇 가지 요구 사항이 있습니다. 조밀한 포인트 구름의 재구성에 만족하지 못할 수도 있으며, 비디오 게임의 3D 장면과 마찬가지로 텍스처 비행기를 제작할 수 있기를 원합니다.

5. 상호 작용. 상호 작용은 주로 사람과 지도 간의 상호 작용을 말합니다. 예를 들어, 증강 현실에서 우리는 가상의 물체를 방에 배치하고 그것들과 상호 작용합니다 (예를 들어 벽에있는 가상 웹 브라우저를 클릭하여 비디오를 보거나 벽에 물건을 던져서 (가상의) 물리적인 충돌이 있었으면 좋겠다. 반면에 로봇 애플리케이션은 사람과 지도와 상호 작용을합니다. 예를 들어, 로봇은 "테이블에서 신문을 가져가라"는 명령을받을 수 있습니다. 그런 다음 환경 맵과 함께 로봇은 "테이블", "위"및 " 신문. " 이를 위해서는 로봇이 지도에 대한 더 높은 수준의 인지도 (시맨틱 맵이라고도 함)가 있어야 합니다.

그림 13-1은 위에서 설명한 다양한 지도 유형과 그 사용법 간의 관계를 그래픽으로 보여줍니다. 우리의 이전 토론은 sparse 맵 섹션에 초점을 맞추었고 dense 맵을 아직 탐구하지 않았습니다. 이른바 dense 맵은 sparse 맵에 상대적인 개념입니다. sparse 지도는 특징 지점인 관심 대상 부분 만 모델링합니다. Dense 지도는 모든 부분을 모델링하는 것을 의미합니다. 동일한 테이블의 경우 sparse 맵은 테이블의 네 모퉁이만 모델링 할 수 있으며 dense 맵은 전체 데스크를 모델링합니다. 포지셔닝의 관점에서 볼 때 카메라를 배치하는 데 4개의 모서리 맵만 사용할 수 있지만 4 개 모서리에서 이 점 사이의 공간 구조를 추론 할 수 없으므로 탐색을 완료하는 데 4 개의 모서리 만 사용할 수는 없습니다.

위의 설명에서 알 수 있듯이 dense 맵은 매우 중요한 위치를 차지합니다. 나머지 질문은 다음과 같습니다. Visual SLAM으로 dense 맵을 만들 수 있습니까? 그렇다면 어떻게 구축 할 수 있습니까?

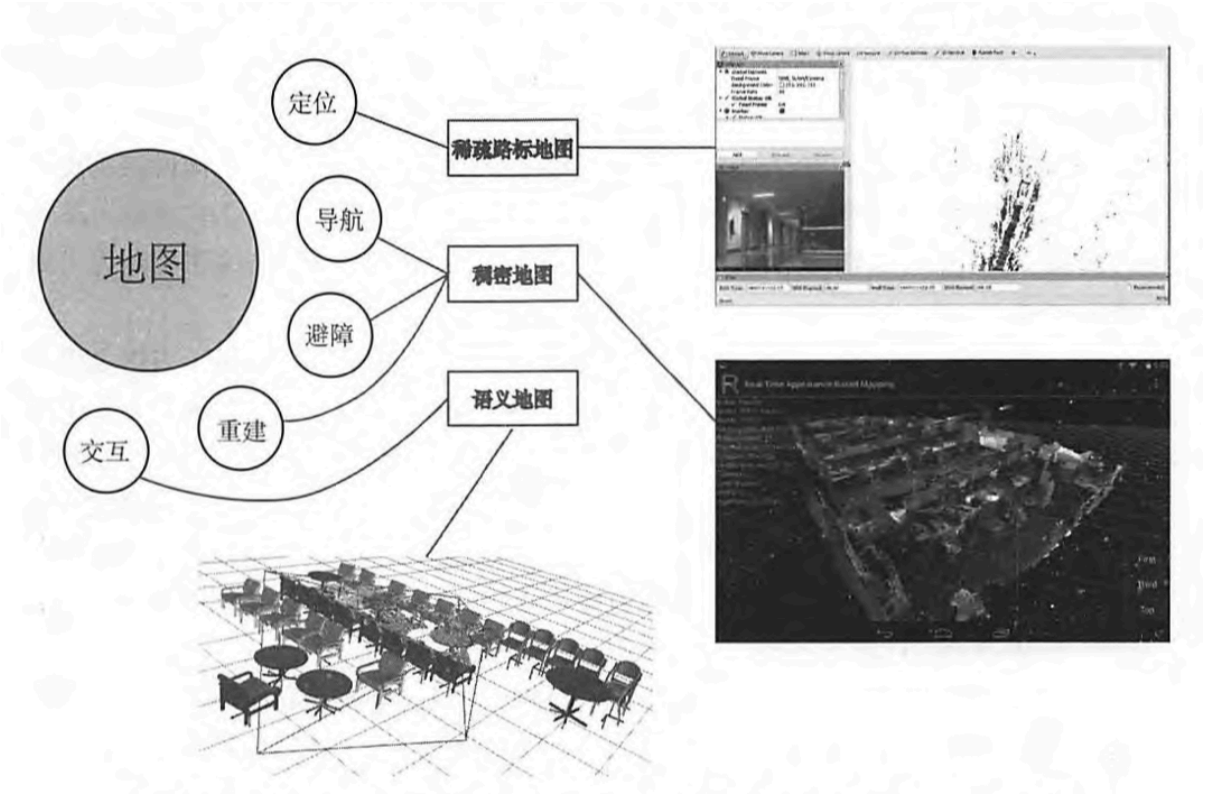


그림 13-1. 다양한 지도의 개략도.

13.2 단안의 dense 재구성

13.2.1 스테레오 비전

Visual SLAM의 dense 재구성은 이 강의의 첫 번째 중요한 주제입니다. 카메라는 오랫동안 시각 전용 센서로 간주되어 왔습니다. 단일 이미지의 픽셀은 카메라의 영상 평면 및 객체의 밝기에만 객체의 각도를 제공 할 수 있지만 객체의 거리를 제공 할 수는 없습니다. Dense 재구성에서는 각 픽셀 (또는 대부분의 픽셀)의 거리를 알아야합니다. 대략 다음과 같은 솔루션이 있습니다.

1. 단안 카메라를 사용하여 카메라를 이동시킨 후 카메라 시점 간의 삼각화를 통해 픽셀 거리를 측정합니다.
2. 양안 카메라를 사용하여 왼쪽 및 오른쪽 대상 시차를 사용하여 픽셀의 거리를 계산합니다.
3. 픽셀 거리를 직접 얻기 위해 RGB-D 카메라를 사용합니다.

처음 두 방법은 StereoVision이라고 하는데, 특히 첫번째 방법은 움직이는 단일 개체는 Moving View Stereo라고도합니다. RGB-D 카메라로 얻은 깊이와 비교할 때, 단안 및 양안 깊이 획득은 보통 정확하지 않습니다. 많은 계산량이 필요하고 덜 안정적인 깊이 추정을 수행합니다. 물론 RGB-D에는 범위, 적용 범위 및 조명 제한이 있지만 RGB-D를 사용한 dense 재구성은 단안 및 양안 깊이 획득보다 더 흔한 선택입니다. 단안 및 양안 깊이 획득 방법의

장점은 RGB-D가 여전히 잘 적용되지 않는 야외 및 Large-scale 에서 입체감으로 깊이 정보를 추정 할 수 있다는 것입니다.

가장 간단한 경우부터 시작하겠습니다. 주어진 카메라 궤도를 기반으로 일정 기간 동안 비디오 시퀀스를 기반으로 이미지의 깊이를 추정하는 방법입니다. 즉, SLAM을 고려하지 않고 약간 간단한 매핑 문제를 고려해 보겠습니다.

비디오 시퀀스를 가정 할 때, 우리는 각 프레임에 해당하는 궤적을 얻습니다. (물론 이것은 Visual odometry Frontend에 의해 추정됩니다.) 이제 첫 번째 이미지를 참조 프레임으로 사용하여 참조 프레임의 각 픽셀의 깊이 (또는 거리)를 계산합니다.

1. 먼저, 이미지에서 피쳐를 추출하고 디스크립터에 기반한 피쳐 들간의 매치를 만든다. 즉,이 피쳐들을 통해 특정 공간 지점을 추적하고 이미지 사이의 위치를 파악합니다.
2. 하나의 이미지만으로 특징점의 위치를 결정할 수 없기 때문에 다른 관점(이미지)에서 관측하여 깊이를 추정해야 합니다. 원리는 이전에 설명한 삼각 측량입니다.

Dense 깊이 맵 추정의 차이점은 각 픽셀을 특징점 계산 디스크립터로 계산할 수 없다는 점입니다. 그러므로, Dense 깊이 추정 문제에서, 매칭은 중요한 부분이된다 : 다른 그림들에서 첫 번째 그림의 픽셀의 위치를 어떻게 결정 하는가? 이를 위해서는 epipolar line search 및 block matching 기술이 필요합니다. 그런 다음 각 영상에서 픽셀의 위치를 알면 삼각측량을 사용하여 특징점과 같은 깊이를 결정할 수 있습니다. 그러나 차이점은 여기서 한 번만이 아니라 깊이 추정을 수렴하기 위해 많은 삼각 측량을 사용해야한다는 것입니다. 깊이 측정치가 증가함에 따라 깊이 측정치가 매우 불확실한 cost에서 점진적으로 안정된 값으로 수렴되기를 바랍니다. 이것은 깊이 필터 기술입니다. 따라서 다음 내용은 주로 이 항목에 중점을 둡니다.

13.2.2 Epipolar line search 및 Block matching

서로 다른 시점에서 같은 점을 관찰함으로써 생성된 기하학적 관계를 먼저 살펴봅시다. 이것은 7.3절에서 논의 된 epipolar geometry와 매우 흡사합니다. 그림 13-2를 참조하십시오. 왼쪽의 카메라는 픽셀 $\boldsymbol{p}_1$을 관찰합니다. 이것은 단안의 카메라이므로 깊이를 알 수 없으므로이 깊이가 특정 영역 내에 있다고 가정하면 특정 최소값 d에서 무한대 사이라고 말할 수 있습니다 $(d_{\min}, +\infty)$. 따라서 픽셀에 해당하는 공간 지점은 특정 선분 (이 경우 광선)에 분산됩니다. 다른 시점 (오른쪽 카메라)에서는 이 선분의 투영도 이미지 평면에서 선을 형성합니다. 이 선은 epipolar line이라고합니다. 이 epipolar line은 두 카메라 사이의 움직임을 알면 결정할 수 있습니다. 그래서 질문은 : 우리가 방금 본 $\boldsymbol{p}_1$ 지점은 epipolar line상에서 어느 점입니까?

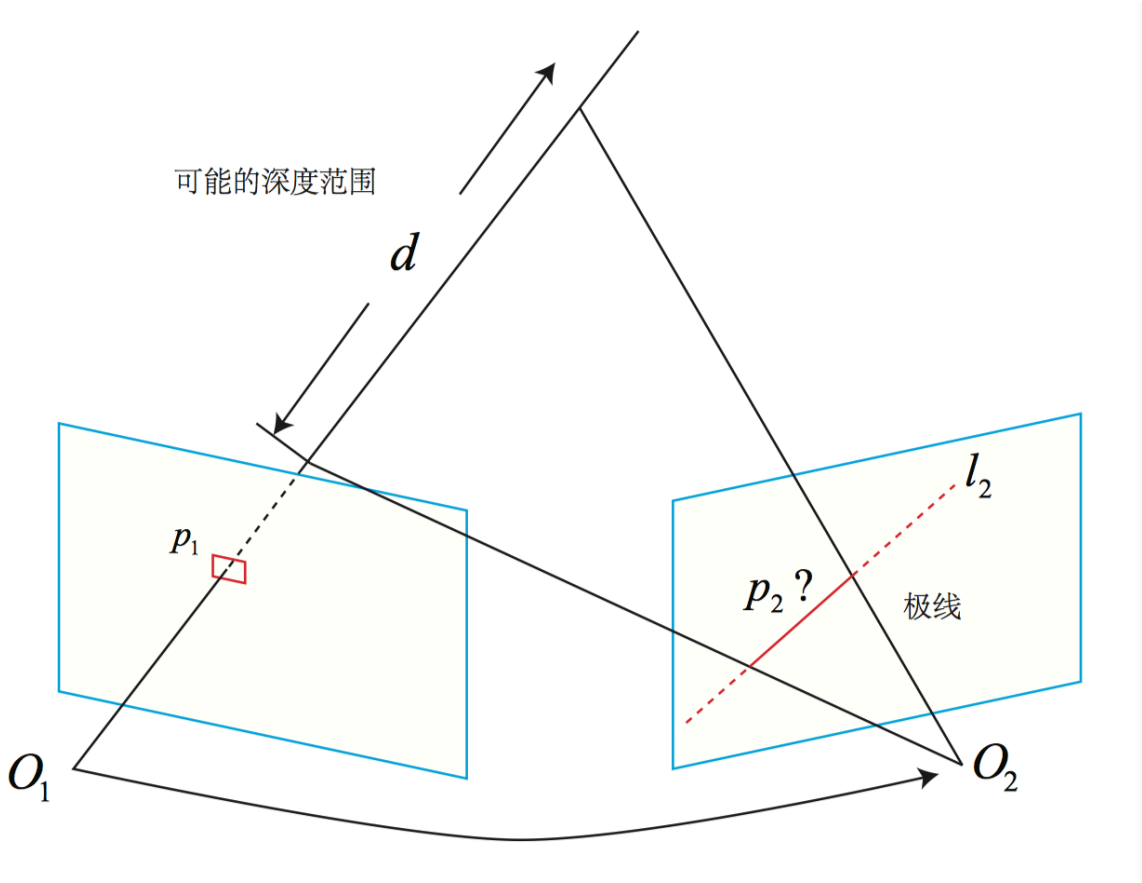


그림 13-2 epipolar line search의 개략도

반복하여, 특징점 방법에서 피쳐 매칭에 의한 p_2의 위치를 찾는다. 그러나 이제는 디스크립터가 없으므로 p_1과 유사한 epipolar line의 점만 검색 할 수 있습니다. 더 구체적으로, 우리는 제2 이미지에서 epipolar line의 한쪽 끝을 따라 다른 끝으로 갈 수 있고, 각 픽셀을 p_1의 유사성과 하나씩 비교할 수있다. 픽셀을 직접 비교하는 관점에서이 방법은 Direct method과 비슷합니다.

Direct method에 대한 논의에서 개별 픽셀의 밝기 값을 비교하는 것이 반드시 안정적이고 신뢰할만한 것은 아니라는 것을 알고 있습니다. 한 가지 분명한 사실은 다음과 같습니다. epipolar line에 p_1과 비슷한 점이 많이있는 경우 어떤 점을 true로 결정합니까? 이것은 우리가 Loop closure detection에서 말한 것으로 돌아갈 것 같습니다 : 두 이미지 (또는 두 점) 사이의 유사성을 어떻게 결정합니까? Loopback 감지는 bag-of-visual-word로 해결되지만 다른 방법을 찾아야합니다.

직관적인 아이디어는 다음과 같습니다. 단일 픽셀의 밝기는 구별 할 수 없으므로 픽셀 블록을 비교할 수 있습니다. 우리는 p_1 주위에 크기 $w \times w$ 의 작은 블록을 취한 다음, epipolar line 상에 같은 크기의 많은 작은 블록을 비교하여 어느 정도 개선합니다. 이를 블록 매칭이라고합니다. 이 프로세스에서 이 비교는 작은 이미지 전체에서 작은 블록 전체의 그레이 스케일 값이 다르다고 가정 할 때만 의미가 있습니다. 따라서 알고리즘의

가정은 픽셀의 그레이스케일 값 불변성에서 이미지 블록의 그레이스케일 불변량까지 어느 정도 강하게됩니다.

자, 이제 우리는 $\boldsymbol{p}_1$ 주위의 작은 조각을 가지고 epipolar line상에 많은 작은 블록들을 가져갑니다. $\boldsymbol{p}_1$ 주위의 작은 블록을 A ∈ R w × w로 그리고 epipolar line상의 n 개의 작은 블록을 $\boldsymbol{B}_i, i = 1, \cdots, n$으로 기록합시다. 그러면 블록과 블록의 차이를 어떻게 계산합니까?
몇 가지 계산 방법이 있습니다.

1. SAD (Sum of Absolute Distance, 절대 차이의 합):

$$S(\boldsymbol{A}, \boldsymbol{B})_{\mathrm{SAD}} = \sum_{i,j} |\boldsymbol{A}(i,j) - \boldsymbol{B}(i,j)|. \tag{13.1}$$

2. SSD (Sum of Squared Distance, 차이의 제곱의 합):

$$S(\boldsymbol{A}, \boldsymbol{B})_{\mathrm{SSD}} = \sum_{i,j} (\boldsymbol{A}(i,j) - \boldsymbol{B}(i,j))^2. \tag{13.2}$$

3. NCC (Normalized Cross Correlation). 이 방법은 두 개의 작은 블록의 상관 관계를 계산하는 방법으로 처음 두 개보다 더 복잡합니다.

$$S(\boldsymbol{A}, \boldsymbol{B})_{\mathrm{NCC}} = \frac{\sum_{i,j} \boldsymbol{A}(i,j)\boldsymbol{B}(i,j)}{\sqrt{\sum_{i,j} \boldsymbol{A}(i,j)^2 \sum_{i,j} \boldsymbol{B}(i,j)^2}}. \tag{13.3}$$

여기서는 상관 관계가 사용되므로 0에 가까운 상관 관계는 두 이미지가 서로 다르다는 것을 의미하고 1에 가까우면 비슷 함을 의미합니다.

우리가 직면한 많은 상황에서와 마찬가지로, 이러한 계산은 종종 정확도 - 효율성 사이의 모순을 가지고 있습니다. 좋은 정확도를 가진 방법은 종종 복잡한 계산을 필요로하는 반면, 간단한 빠른 알고리즘은 계산적으로 비효율적인 경향이 있습니다. 이를 위해서는 실제 프로젝트에서 트레이드 오프를 고려해야 합니다. 또한 이 간단한 버전 외에 평균의 SSD, 평균의 NCC 등 각 작은 블록의 평균을 제거 할 수 있습니다. 평균을 제거한 후에 "작은 블록 B가 A보다 밝지 만 여전히 매우 유사"한 상황을 허용하므로 이전보다 훨씬 안정적입니다. 독자가 더 많은 블록 일치 측정 항목에 관심이있는 경우 [88, 89]를 살펴보는 것이 좋습니다.

이제 우리는 epipolar line에서 $\boldsymbol{A}$와 각 $\boldsymbol{B}_i$ 사이의 유사성 척도를 계산합니다. 편의상, 우리가 NCC를 사용한다고 가정하고, epipolar line을 따라 NCC 분배를 얻을 것입니다. 이 분포의 모양은 그림 13-3과 같이 이미지 자체의 모양에 크게 의존합니다. 긴 탐색 범위의 경우 우리는 일반적으로 볼록하지 않은 함수를 얻습니다. 이 분포에는 많은 피크가 있지만 단 하나의 참된 대응점이 있어야합니다. 이 경우 깊이를 설명하기 위해 단일 값을 사용하는 대신 확률 분포를 사용하여 깊이 값을 설명하는 경향이 있습니다.

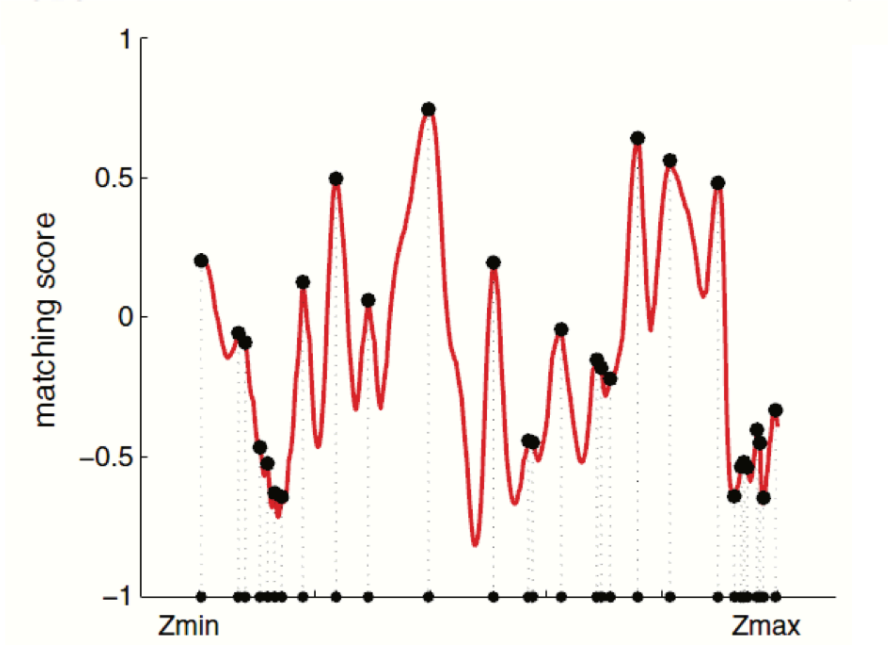


그림 13-3 거리에 따른 매칭 점수의 분포, 이미지는 문헌 [90]에서 참조했습니다.

13.2.3 가우시안 분포 깊이 필터

픽셀 깊이의 추정은 또한 상태 추정 문제로 모델링 될 수 있으므로 앞에서 살펴본 필터 및 비선형 최적화 방법으로서의 두 가지 솔루션이 있습니다. 비선형 최적화 효과는 더 좋지만, Frontend가 이미 많은 계산량을 차지하고 있다는 것을 고려하여, SLAM의 실시간 처리가 필요하면, 일반적으로 계산량이 더 적은 필터 방법을 채택한다. 깊이 필터에 대해서도 설명합니다.

깊이 분포의 가정에 대한 몇 가지 다른 접근 방법이 있습니다. 첫째, 상대적으로 간단한 가정 하에서 깊이 값이 가우시안 분포를 따르고 칼만과 같은 접근법을 취한다고 가정 할 수 있습니다. 반면, 문헌 [34, 90]에서는 균일한 Gaussian Mixture Model의 가정이 또한 채택되고, 더 복잡한 깊이 필터의 또 다른 형태가 도출된다. 단순성과 사용의 용이성 원칙에 입각하여 먼저 가우시안 분포 가정 하에서 깊이 필터를 소개하고 시연한 다음 균일한 가우시안 분포의 필터를 연습문제로 사용합니다.

픽셀의 깊이 d를 다음과 같이 설정합니다.

$$P(d) = N(\mu, \sigma^2). \tag{13.4}$$

그리고 새로운 데이터가 도착할 때마다 우리는 그 깊이를 관찰 할 것입니다. 다시 이 관측치가 가우스 분포라고 가정합니다.

$$P(d_{\text{obs}}) = N(\mu_{\text{obs}}, \sigma^2_{\text{obs}}). \tag{13.5}$$

따라서 우리의 질문은 관찰된 정보를 사용하여 원본 d의 분포를 어떻게 갱신 할 것인가입니다. 이것은 정보 융합의 문제입니다. 부록 A에 따르면 두 가우스 분포의 곱은

여전히 가우스 분포입니다. 퓨전 후 d의 분포를 $N(\mu_{\text{fuse}}, \sigma^2_{\text{fuse}})$ 이라고하면, 가우시안 분포의 곱에 따라 다음과 같습니다.

$$\mu_{\text{fuse}} = \frac{\sigma^2_{\text{obs}}\mu + \sigma^2\mu_{\text{obs}}}{\sigma^2 + \sigma^2_{\text{obs}}}, \quad \sigma^2_{\text{fuse}} = \frac{\sigma^2\sigma^2_{\text{obs}}}{\sigma^2 + \sigma^2_{\text{obs}}}. \tag{13.6}$$

관측 방정식과 운동 방정식만 있기 때문에 여기서의 깊이는 정보 융합 부분을 사용하기 때문에 완전한 칼만처럼 예측하고 업데이트 할 필요가 없습니다. 융합 방정식은 이해하기 쉽지만, 여전히 존재하는 문제, 즉 우리가 관찰하는 깊이 분포를 결정하는 방법이 있다는 것을 알 수 있습니다. 즉, $\mu_{\text{obs}}, \sigma_{\text{obs}}$ 를 계산하는 방법은?

$\mu_{\text{obs}}, \sigma_{\text{obs}}$ 를 다룰 수 있는 몇 가지 다른 방법이 있습니다. 예를 들어, 문헌 [37]은 기하학적 불확도를 고려하는 반면, 문헌 [90]은 기하학적 불확도와 광도 불확도의 합을 고려한다. 당분간은 기하학적 관계로 인한 불확실성만을 고려해야합니다. 이제 우리는 epipolar line 탐색과 블록 매칭을 통해 현재 프레임에서 참조 프레임의 픽셀의 투영 위치를 결정한다고 가정 해 보겠습니다.

그림 13-4를 예로 들어 보겠습니다. epipolar line search를 고려하면 $\boldsymbol{p}_1$ 에 해당하는 $\boldsymbol{p}_2$ 점을 찾고 $\boldsymbol{p}_1$ 의 깊이 값을 관측하고 $\boldsymbol{p}_1$ 에 해당하는 3차원 점을 P라고 생각합시다. 따라서, $\boldsymbol{O}_1\boldsymbol{P}$ 는 p이고, $\boldsymbol{O}_1\boldsymbol{O}_2$ 는 카메라의 이동 거리 t이고, $\boldsymbol{O}_2\boldsymbol{P}$ 는 a로 표시된다. 또한,이 삼각형의 하단 두 모서리는 α, β로 표시됩니다. 이제는 β 각이 β '가되고 p가 p'가되고 상각은 γ가되도록 epipolar line l_2 에 픽셀 크기 오차가 있다고 가정합니다. 우리가 알아야 할 것은,이 픽셀의 오차로 인해 발생하는 p '와 p의 차이는 무엇입니까?
이것은 전형적인 기하학적 문제입니다. 이 양 사이의 기하학적 관계를 작성해 봅시다. 분명히 있습니다 :

$$\begin{aligned} \boldsymbol{a} &= \boldsymbol{p} - \boldsymbol{t} \\ \alpha &= \arccos\langle \boldsymbol{p}, \boldsymbol{t} \rangle \\ \beta &= \arccos\langle \boldsymbol{a}, -\boldsymbol{t} \rangle . \end{aligned} \tag{13.7}$$

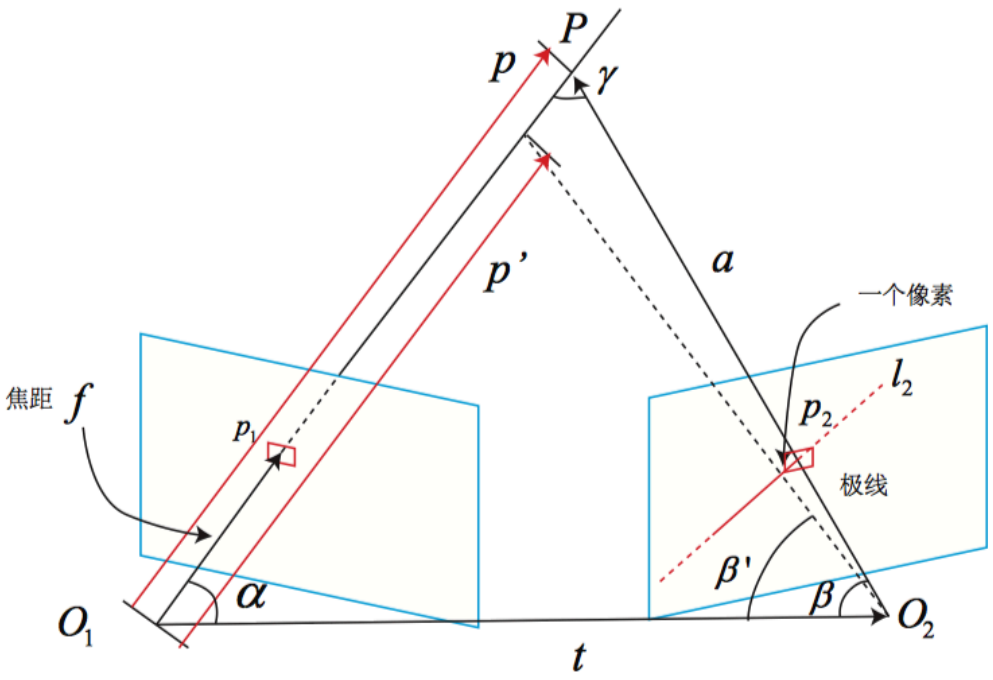


그림 13-4 불확도 분석

p_2 에 대해 픽셀을 이동시키는 것은 카메라의 초점 거리가 f이므로 β가 델타 δβ를 생성하게합니다.

$$\delta\beta = \arctan\frac{1}{f}. \tag{13.8}$$

그러므로:

$$\begin{aligned} \beta' &= \beta + \delta\beta \\ \gamma &= \pi - \alpha - \beta'. \end{aligned} \tag{13.9}$$

따라서, Sin 정리로부터, p '의 크기를 얻을 수있다 :

$$\|\boldsymbol{p}'\| = \|\boldsymbol{t}\|\frac{\sin\beta'}{\sin\gamma}. \tag{13.10}$$

따라서 단일 픽셀의 불확실성으로 인한 깊이 불확실성을 결정했습니다. Epipolar line상의 블록 검색에 픽셀 오류가 하나만 있다고 생각되면 다음을 설정할 수 있습니다.

$$\sigma_{\text{obs}} = \|\boldsymbol{p}\| - \|\boldsymbol{p}'\|. \tag{13.11}$$

물론 epipolar line search의 불확실성이 한 픽셀보다 큰 경우 이 유도에 따라 이 불확실성을 증폭 할 수도 있습니다. 뒤따르는 심층적인 데이터 융합은 이미 이전에 소개되었습니다. 실제 엔지니어링에서 불확실성이 특정 임계 값보다 작으면 깊이 데이터가 수렴 된 것으로 간주 할 수 있습니다.

요약하면, 우리는 치밀한 깊이를 추정하기 위한 완전한 과정을 제시한다 :

1. 모든 픽셀의 깊이가 초기 가우스 분포를 만족한다고 가정한다.

2. 새로운 데이터가 생성되면, 투영 점 위치는 극선 검색과 블록 매칭에 의해 결정됩니다.
3. 기하학적 관계를 기반으로 삼각 측량 후의 깊이와 불확도를 계산합니다.
4. 현재 관측치를 이전 추정치에 통합합니다. 수렴하면 계산이 중지되고, 그렇지 않으면 2 단계로 돌아갑니다.

이 단계들은 실행 가능한 깊이 추정치를 형성합니다. 실제로 작동하는 방식을 연습 섹션에서 보여줍니다.

13.3 실습 : 단안의 조밀한 재구성

이 절의 샘플 프로그램은 REMODE [87, 91]에 대한 테스트 데이터 세트를 사용할 것입니다. 무인 항공기의 단안 정면을 제공하며 총 200 개의 이미지를 제공하면서 각 이미지의 실제 포즈를 제공합니다. 이들 데이터에 기초하여, 제 1 프레임 이미지의 각 픽셀에 대응하는 깊이 값, 즉 단안 dense 재구성을 고려해봅시다.

먼저 샘플 프로그램에서 사용하는 데이터를 http://rpg.ifi.uzh.ch/datasets/remode_test_data.zip에서 다운로드 할 것을 독자에게 요청하십시오. 웹 브라우저 또는 다운로드 도구를 사용하여 다운로드 할 수 있습니다. 압축을 풀면 test_data/Images에 0에서 200까지의 모든 이미지를 확인할 수 있고 각 이미지의 포즈를 기록하는 test_data 디렉토리에 텍스트 파일이 표시됩니다.

```
scene_000.png 1.086410 4.766730 -1.449960 0.789455 0.051299 -0.000779 0.611661
scene_001.png 1.086390 4.766370 -1.449530 0.789180 0.051881 -0.001131 0.611966
scene_002.png 1.086120 4.765520 -1.449090 0.788982 0.052159 -0.000735 0.612198
......
```

그림 13-5는 여러 순간의 이미지를 보여줍니다. 이 장면은 주로 지상, 책상으로 구성되어있는 것을 볼 수 있습니다. 깊이 추정이 대략 정확하다면, 우리는 적어도 테이블의 깊이와 지면의 차이를 볼 수 있습니다. 아래에서 우리는 이전 설명을 따라 치밀한 깊이 추정 절차를 작성합니다. 이해를 돕기 위해 이 프로그램은 C 언어 스타일로 작성되어 하나의 파일에 저장됩니다. 이 프로그램은 약간 길기 때문에 이 책의 몇 가지 중요한 기능에 중점을 두고 나머지 내용은 독자가 GitHub의 소스 코드와 비교하여 읽어야합니다.

slambook/ch13/dense_monocular/dense_mapping.cpp

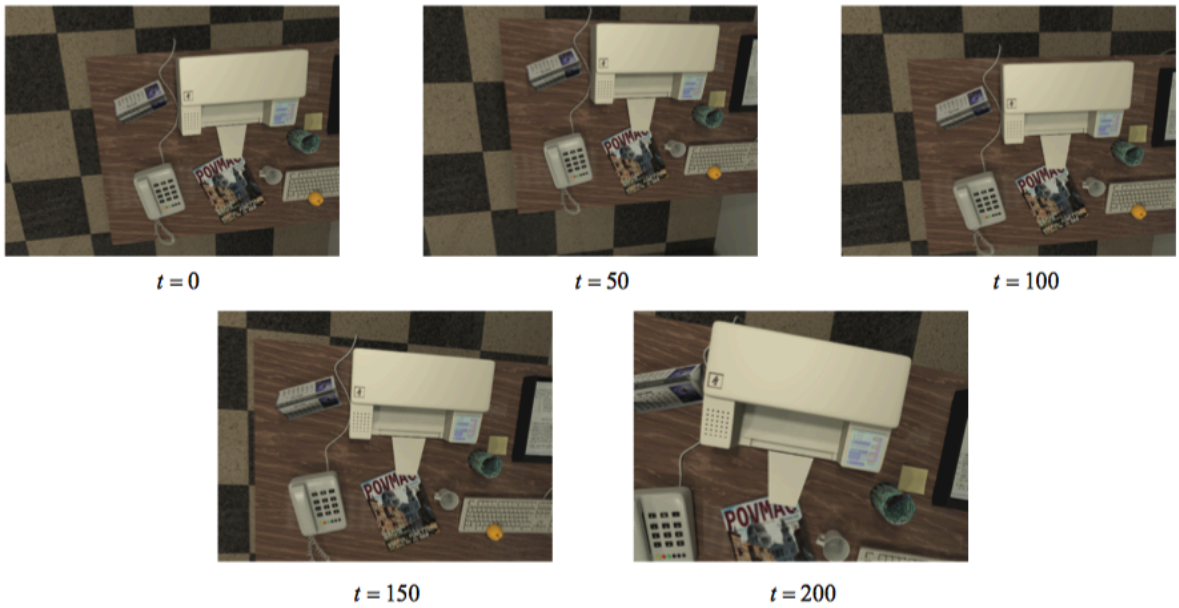


그림 13-5 데이터 세트 아이콘

독자가 이전 섹션을 이해한다면 여기서 소스 코드를 읽는 것이 어렵지 않다고 생각합니다. 그러나 몇 가지 주요 기능을 설명합니다.

1. 주요 기능은 매우 간단합니다. 데이터 세트에서 이미지를 읽은 다음 업데이트 함수에 전달하여 깊이 맵을 업데이트해야합니다.

2. 업데이트 기능에서 참조 프레임의 각 픽셀을 탐색하고 현재 프레임에서 epipolar line 일치를 찾은 다음 일치하는 경우 epipolar line 일치의 결과를 사용하여 깊이 지도 추정을 업데이트합니다.

3. epipolar line 탐색의 원리는 이전 섹션에서 설명한 것과 거의 동일하지만 구현시 세부 사항이 추가됩니다. 깊이 값이 가우스 분포를 따르기 때문에 평균값을 중심으로 반경으로 ± 3σ를 취한 다음 현재 프레임을 취합니다. epipolar line의 투영을 찾으십시오. 그런 다음 이 epipolar line에서 픽셀을 탐색합니다 (단계 크기는 sqrt(2)/2에 대해 0.7의 근사치를 취합니다). 가장 높은 NCC가있는 점을 일치 지점으로 찾습니다. 가장 높은 NCC가 임계 값 (여기서는 0.85)보다 낮으면 일치가 실패한 것으로 간주됩니다.

4. NCC의 계산은 이미지 블록 A, B에 대해 평균 제거 방법을 사용합니다.

$$\mathrm{NCC}_z(\boldsymbol{A},\boldsymbol{B}) = \frac{\sum_{i,j}\left(\boldsymbol{A}(i,j)-\bar{\boldsymbol{A}}(i,j)\right)\left(\boldsymbol{B}(i,j)-\bar{\boldsymbol{B}}(i,j)\right)}{\sqrt{\sum_{i,j}\left(\boldsymbol{A}(i,j)-\bar{\boldsymbol{A}}(i,j)\right)^2\sum_{i,j}\left(\boldsymbol{B}(i,j)-\bar{\boldsymbol{B}}(i,j)\right)^2}}. \tag{13.12}$$

5. 삼각형 분할은 7.5 절에서와 같은 방식으로 계산됩니다. 불확실성 및 가우스 융합 방법의 계산은 이전 섹션과 일치합니다. 프로그램은 조금 길지만, 독자는 위의 팁에 따라 읽을 수 있다고 생각합니다. 실제 실행 효과를 살펴 보겠습니다.

실험 결과

이 프로그램을 컴파일 한 후에는 데이터 셋 디렉토리를 매개 변수로 사용하여 실행하십시오.

```
$ build/dense_mapping ~/dataset/test_data
read total 202 files.
*** loop 1 ***
*** loop 2 ***
......
```

프로그램에 의해 출력된 정보는 상대적으로 간단하며 반복 횟수, 현재 이미지 및 깊이 맵만 표시됩니다. 깊이 맵과 관련하여 깊이 값에 0.4를 곱한 결과, 즉 순수한 흰색 점의 깊이 (값이 1.0 임)가 약 2.5 미터임을 보여줍니다. 색상이 어두울수록 깊이 값이 작아집니다. 즉 물체가 더 가까워집니다. 프로그램이 실제로 실행중인 경우, 깊이 평가는 동적 프로세스입니다. 즉, 덜 구체적인 초기 값에서 안정된 값으로 수렴하는 프로세스입니다. 초기 값은 평균과 분산이 3.0 인 분포를 사용합니다. 물론 초기 분포를 수정하여 결과에 어떤 영향을 주는지 확인할 수도 있습니다.
그림 13-6에서 반복 횟수가 특정 값을 초과하면 깊이 맵이 안정된 경향이 있고 새 데이터가 변경되지 않는다는 것을 알 수 있습니다. 안정화 후 깊이 맵을 보면 마루와 테이블의 차이를 대략적으로 볼 수 있지만 테이블상의 오브젝트 깊이는 테이블에 가깝습니다. 전체 견적은 대부분 정확하지만 잘못 계산한 값도 많습니다. 너무 크거나 작은 추정치 인 심도 맵의 주변 데이터와 불일치로 나타납니다. 또한, 이동하는 동안 관측치가 적기 때문에 모서리의 위치가 정확하게 예측되지 않았습니다. 요약하면, 우리는 이 깊이 맵의 대부분이 정확하다고 믿지만 원하는 효과를 얻지는 못합니다. 다음 절에서 이러한 상황의 원인을 분석하고 향상시킬 수있는 것에 대해 논의 할 것입니다.

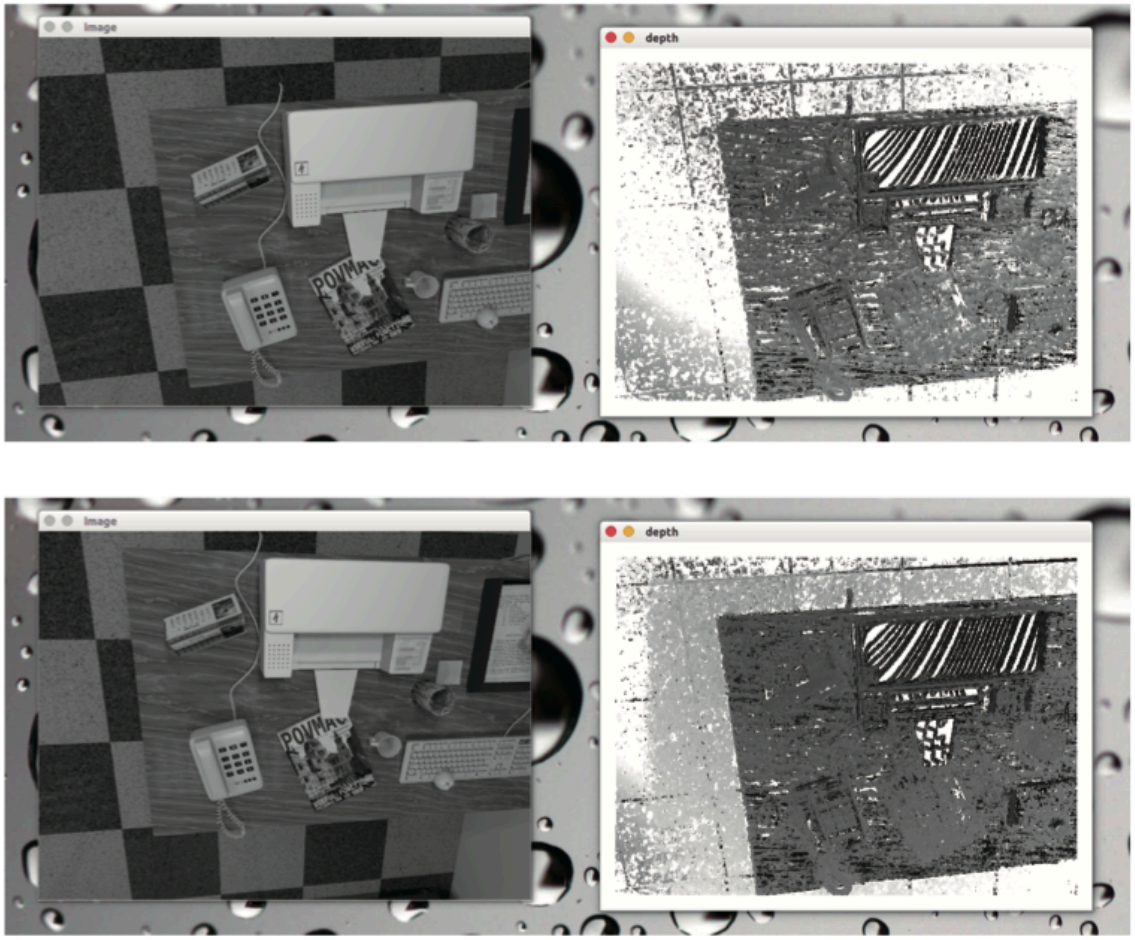


그림 13-6 데모 프로그램 런타임의 스크린 샷. 두 수치는 각각 반복 10과 30의 결과입니다.

13.4 실험 분석 및 토론

이전 섹션에서 우리는 참조 프레임의 각 픽셀의 깊이를 추정하면서 움직이는 단안 카메라의 dense 매핑을 시연했습니다. 우리의 코드는 비교적 간단하고 많은 트릭을 사용하지 않으므로 실제 엔지니어링에서 흔히 발생하는 상황이 있습니다. 간단한 것들은 종종 가장 효과적이지 않습니다.

실제 데이터의 복잡성으로 인해 실제 환경에서 작동 할 수 있는 프로그램은 신중하게 고려해야하고 많은 엔지니어링 기술이 필요하기 때문에 각 실제 코드가 매우 복잡해지므로 초보자에게 설명하기가 어렵습니다. 우리는 덜 효율적이지만 비교적 읽기 쉽고 구현하기 쉬운 구현을 사용해야했습니다. 우리는 데모 프로그램에 대한 몇 가지 개선점을 확실히 제안 할 수 있지만 수정 된 (매우 복잡한) 프로그램을 독자에게 직접 제시하기위한 것이 아닙니다.

아래에서 우리는 이전 실험의 결과에 대한 예비 분석을 수행 할 것입니다. 컴퓨터 비전과 필터 관점에서 데모 실험의 결과를 분석합니다.

13.4.1 픽셀 그래디언트 문제

깊이 이미지를 보면 분명히 알 수 있습니다. 블록 일치의 정확성은 이미지 블록이 정확한 매칭을 가졌는지 여부에 달려 있습니다. 분명히 이미지 블록이 흑백이거나 유효한 정보가 부족한 경우 NCC 계산에서 실수로 이미지 블록을 주변 픽셀과 일치시킬 수 있습니다. 독자는 데모에서 프린터의 표면을 관찰해야 합니다. 균일한 흰색이기 때문에 불일치가 발생하기 쉽습니다. 따라서 프린터 표면의 깊이 정보는 대부분 잘못되었습니다. 샘플 프로그램의 공간 표면에는 분명히있을 것으로 예상되지 않는 스트라이프 깊이 추정치가 있으며, 우리의 시각적 판단에 따르면, 프린터 표면이 매끄러워야 합니다.

우리가 Direct method에서 보았던 여기에 관련된 문제가 있습니다. 블록 매칭 (그리고 NCC 계산)을 할 때, 우리는 작은 블록이 변경되지 않았다고 가정하고 그 작은 블록을 다른 작은 블록과 비교해야합니다. 이 때, 명확한 그라디언트를 가진 작은 블록은 다른 블록과 확연히 구별되는 특징 값을 가지며 불일치를 유발하기 쉽지 않습니다. 눈에 띄지 않는 기울기를 가진 픽셀의 경우, 블록 매칭에서 차별이 없기 때문에 깊이를 효과적으로 추정하기가 어렵습니다. 반대로 픽셀 그래디언트가 더 명확한 곳에서는 잡지, 전화 및 데스크톱의 질감이 뚜렷한 다른 객체와 같이 우리가 얻는 깊이 정보가 상대적으로 정확합니다. 따라서 데모 프로그램은 스테레오 비전에서 가장 일반적인 문제인 대상의 질감에 대한 의존성을 반영합니다. 이 문제는 스테레오 비전의 재구성 품질이 환경 질감에 크게 의존한다는 것을 구현하는 양안 시력에서도 매우 일반적입니다.

데모 프로그램은 의도적으로 바둑판 같은 바닥, 나뭇결 모양의 바탕 화면과 같은 질감 친화적인 환경을 사용하여 겉으로는 좋은 결과를 얻을 수 있습니다. 그러나 실제로는 벽과 매끄러운 표면과 같은 밝기가 균일한 영역이 종종 표시되어 깊이 추정에 영향을 미칩니다. 어떤면에서 이 문제는 기존의 알고리즘 흐름에서 개선되고 해결 될 수 없습니다. 픽셀 주변의 작은 영역 (작은 블록)에 대해서만 신경을 써야합니다.

그림 13-7을 예로 들어 두 가지 극단적 인 경우를 생각해 봅시다 : 픽셀 그라디언트가 epipolar line과 평행한 경우와 epipolar line에 수직인 경우입니다. 먼저 수직적인 상황을 살펴 보겠습니다. 수직적인 예제에서, 비록 작은 블록이 분명한 그라디언트를 가지고 있다고 하더라도, 우리가 epipolar line을 따라 블록 매칭을 할 때, 매칭이 동일하다는 것을 알게 될 것이므로, 효과적인 매치가 없습니다. 반대로, 수평적인 예에서 우리는 가장 높은 일치가 발생한 곳을 정확하게 결정할 수있었습니다. 실제로 그라디언트와 epipolar line은 완전히 수직이 아니거나 완전히 평행하지 않은 어딘가에 있을 것입니다. 이때 픽셀 기울기와 epipolar line 사이의 각도가 클 때 epipolar line 일치의 불확실성이 크고 각도가 작으면 일치의 불확실성이 작다고합니다. 데모 프로그램에서 우리는 이러한 상황을 픽셀 오류로 일률적으로 처리합니다. 실제로는 충분하지 않습니다. epipolar line과 픽셀 그래디언트의 관계를 고려할 때보다 정확한 불확도 모델을 사용해야합니다. 특정 조정 및 개선은 연습문제에 있습니다.

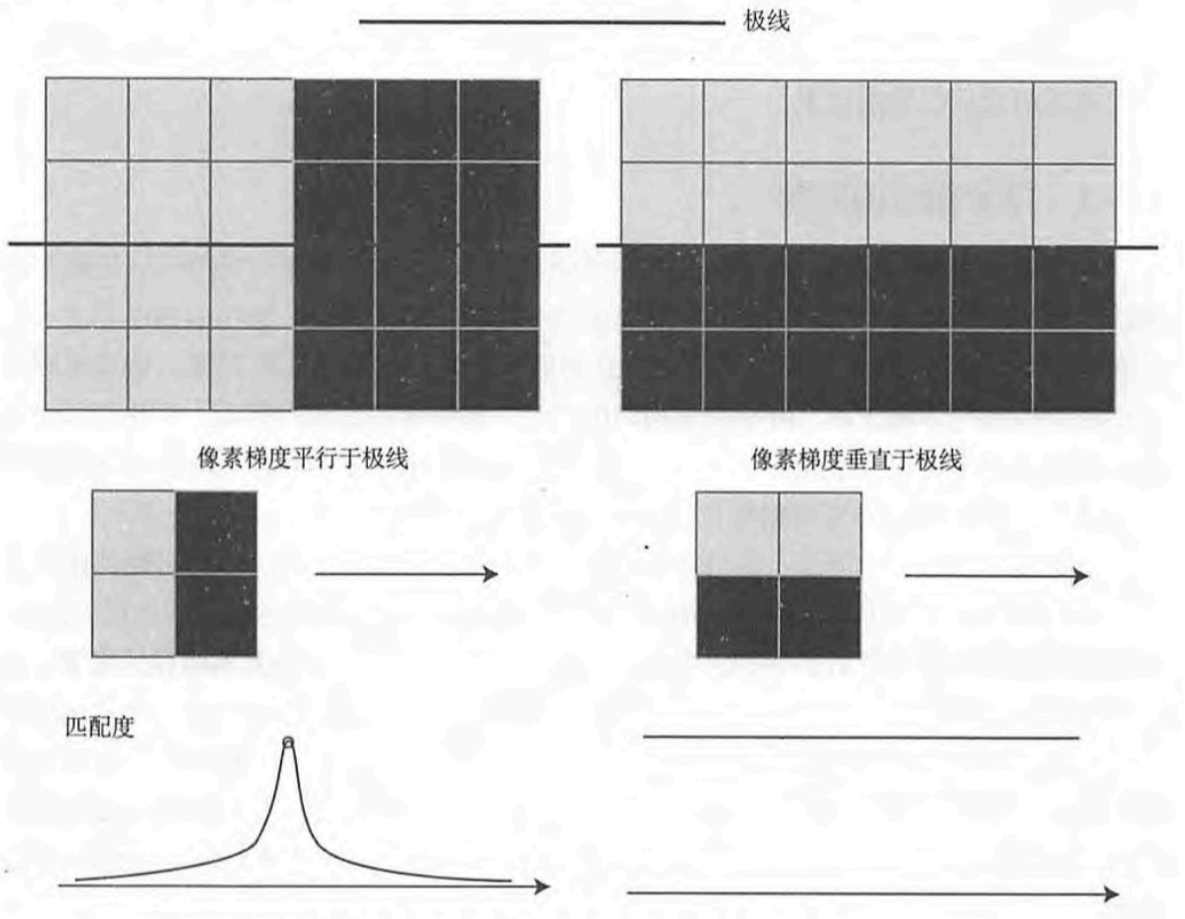


그림 13-7 픽셀 그래디언트와 epipolar line 간의 관계에 대한 개략도.

13.4.2 Inverse depth

다른 관점에서 우리는 다음과 같이 질문 할 수 있습니다. 픽셀 깊이를 가우스 분포로 가정하는 것이 적절합니까? 이것은 매개 변수화 문제 (매개 변수화)와 관련이 있습니다.

이전 섹션에서 우리는 파라메트릭 양식인 점의 x, y 및 z 좌표로 월드 좌표를 설명했습니다. 3개의 양 x, y, z는 무작위로 생각되며 (3차원) 가우시안 분포를 따릅니다. 그러나 이 강의에서는 이미지 좌표와 깊이 값 d를 사용하여 공간 지점 (즉, dense 매핑)을 설명합니다. 우리는 u, v가 움직이지 않으며, 또 다른 매개 변수 형식인 d (1차원) 가우시안 분포를 따른다고 생각합니다. 그런 다음 두 가지 매개 변수가 있는 형식의 차이점은 무엇입니까? 가우시안 분포를 따르고 또 다른 파라 메트릭 형태를 형성한다고 가정 할 수 있습니까?

다른 파라 메트릭 양식은 실제로 동일한 양, 즉 특정 3차원 공간 점을 나타냅니다. 카메라의 한 점을 볼 때, 그 이미지 좌표 u, v는 상대적으로 결정적이며, 깊이 값 d는 매우 불확실하다는 것을 고려하면. 이때 세계 좌표 x, y, z를 사용하여이 점을 설명하면 카메라의 현재 포즈에 따라 x, y 및 z의 세 가지 양간에 중요한 상관 관계가있을 수 있습니다. 공분산 행렬에 반영된 대각선이 아닌 요소는 0이 아닙니다. u, v, d가 점을 매개 변수화하기 위해

사용된다면 u, v, d는 적어도 거의 독립적이며 u, v도 독립적이라고 생각할 수도 있습니다 - 그래서 공분산 행렬은 대각선으로 근사됩니다.

Inverse depth는 최근 몇 년간 SLAM 연구에서 널리 사용되는 매개변수화 기법입니다[114,115]. . 데모에서는 깊이 값이 가우스 분포 $d \sim N(\mu, \sigma^2)$를 만족한다고 가정했습니다. 그러나 이것은 불합리한가? 깊이는 정말로 정규 분포와 비슷합니까? 생각해 보면, 정규 분포에는 몇 가지 문제가 있습니다.

1. 우리가 실제로 말하고 싶은 것은 장면의 깊이는 약 5 ~ 10 미터이고, 더 먼 지점이있을 수 있지만 근접 거리는 확실히 카메라 초점 거리보다 작지 않습니다 (또는 깊이가 0보다 작지 않음). 이 분포는 가우시안 분포와 같은 대칭 모양을 형성하지 않습니다. 그 꼬리는 약간 길 수도 있고, 음수 영역은 0 일 수도 있습니다.
2. 일부 야외 응용 프로그램에서는 매우 멀리 떨어져 있거나 심지어 무한대인 지점이있을 수 있습니다. 초기 값에서 이러한 점을 다루는 것은 어렵습니다. 정규 분포로 설명하는 것은 몇 가지 어려움이 있습니다.

따라서 Inverse depth가 생겨났습니다. 시뮬레이션에서, 깊이의 역함수, 즉 Inverse depth가 가우스 분포에 더 효과적이라는 것을 알게되었다. 후속적으로, 실제적인 적용에서 Inverse depth는 더 나은 수치 안정성을 가지며 점차적으로 일반적인 기법이되며 기존의 SLAM 기법 [56,57,73]에서 표준 기법에 존재한다.

데모 프로그램을 양의 깊이값에서 Inverse depth로 변경하는 것은 복잡하지 않습니다. 깊이 도출이 정면에 나타나면 d를 Inverse depth d^{-1}로 바꾸면 충분합니다. 우리는 또한 이 변환을 연습문제으로 남겨두고 그것을 독자에게 건네줍니다.

13.4.3 이미지 간의 변환

이미지 대 이미지 변환을 수행하는 것은 블록 일치 이전의 일반적인 전처리 방법이기도합니다. 이는 이미지 타일이 카메라가 움직이는 것과 동일한 것으로 가정하고 카메라가 패닝할 때(회전 변환만 수행할 때) 이 가정을 유지할 수 있지만 카메라가 크게 회전하면, 그것을 지키기가 어렵습니다. 특히, 카메라가 광학 중심을 중심으로 회전하면 흑백 이미지가 흑백 블록이되어 여전히 동일한 블록 임에도 불구하고 상관 관계가 음수가됩니다.

이러한 일이 발생하지 않도록 하려면 일반적으로 블록이 일치하기 전에 참조 프레임과 현재 프레임 간의 모션을 고려해야합니다. 카메라 모델에 따르면, 기준 프레임상의 하나의 픽셀 $\boldsymbol{P}_R$은 실제 3차원 포인트 세계 좌표 $\boldsymbol{P}_W$ 와 다음과 같은 관계를 갖는다 :

$$d_R \boldsymbol{P}_R = \boldsymbol{K} \left(\boldsymbol{R}_{\mathrm{RW}} \boldsymbol{P}_W + \boldsymbol{t}_{\mathrm{RW}} \right). \tag{13.13}$$

마찬가지로 현재 프레임의 경우 $\boldsymbol{P}_C$ 로 표시된 $\boldsymbol{P}_W$ 의 투영도 있습니다.

$$d_C \boldsymbol{P}_C = \boldsymbol{K} \left(\boldsymbol{R}_{\mathrm{CW}} \boldsymbol{P}_W + \boldsymbol{t}_{\mathrm{CW}} \right). \tag{13.14}$$

$\boldsymbol{P}_W$ 를 대입하고 제거하면 두 이미지 사이에 픽셀 관계가 생깁니다.

$$d_C \boldsymbol{P}_C = d_R \boldsymbol{K} \boldsymbol{R}_{\mathrm{CW}} \boldsymbol{R}_{\mathrm{RW}}^{\mathrm{T}} \boldsymbol{K}^{-1} \boldsymbol{P}_R + \boldsymbol{K} \boldsymbol{t}_{\mathrm{CW}} - \boldsymbol{K} \boldsymbol{R}_{\mathrm{CW}} \boldsymbol{R}_{\mathrm{RW}}^{\mathrm{T}} \boldsymbol{K} \boldsymbol{t}_{\mathrm{RW}}. \tag{13.15}$$

$d_R, \boldsymbol{P}_R$이 알려지면 $\boldsymbol{P}_C$의 투영 위치를 계산할 수 있습니다. 이 시점에서 $\boldsymbol{P}_R$의 두 구성 요소 각각에 $\mathrm{d}u, \mathrm{d}v$를 증가시킴으로써 $\boldsymbol{P}_C$, $\mathrm{d}u_c, \mathrm{d}v_c$의 Incremental을 얻을 수 있습니다. 이러한 방식으로, 로컬 영역에서의 현재 프레임 이미지 좌표 변환과 기준 프레임 간의 선형 관계가 계산되어 아핀 변환을 구성한다 :

$$\begin{bmatrix} \mathrm{d}u_c \\ \mathrm{d}v_c \end{bmatrix} = \begin{bmatrix} \frac{\mathrm{d}u_c}{\mathrm{d}u} & \frac{\mathrm{d}u_c}{\mathrm{d}v} \\ \frac{\mathrm{d}v_c}{\mathrm{d}u} & \frac{\mathrm{d}v_c}{\mathrm{d}v} \end{bmatrix} \begin{bmatrix} \mathrm{d}u \\ \mathrm{d}v \end{bmatrix} \tag{13.16}$$

아핀 변환 행렬에 따르면, 현재 프레임 (또는 참조 프레임)의 픽셀을 변환 한 다음 블록 일치를 수행하여 회전에 더 나은 효과를 얻을 수 있습니다.

13.4.4 병렬화 : 효율성 문제

실험에서 우리가 추정하고자하는 지점이 수백 개 정도의 특징점에서 수십만 픽셀로 바뀌었기 때문에 (깊이 영상을 활용하므로) 밀도가 높은 깊이지도의 추정이 매우 시간 소모적이라는 것을 알았습니다. 이러한 많은 양을 실시간으로 계산할 수는 없습니다. 그러나 이 문제에는 또 다른 특성이 있습니다. 수십만 픽셀의 깊이 추정치는 서로 독립적입니다! 따라서 병렬 처리가 유용합니다.

샘플 프로그램에서 이중 루프로 모든 픽셀을 순회하며 하나씩 Epipolar line 탐색을 수행합니다. CPU를 사용할 때 프로세스는 순차적으로 수행됩니다. 마지막 픽셀을 계산 한 후 다음 픽셀을 계산해야합니다. 그러나 실제로는 이전 픽셀의 계산이 끝날 때까지 다음 픽셀을 기다릴 필요가 없습니다. 두 픽셀 사이에 명확한 연결이 없기 때문에 여러 스레드를 사용하여 각 픽셀을 개별적으로 계산 한 다음 결과를 통합 할 수 있습니다. 이론상으로, (300,000 개의 픽셀을 계산해야할 때) 우리가 300,000 개의 쓰레드를 가지고 있다면, 이 문제의 계산 시간은 한 픽셀을 계산하는 것과 같습니다.

GPU의 병렬 컴퓨팅 아키텍처는 이러한 유형의 문제에 매우 적합하므로 dense reconstruction에서 병렬 가속에 GPU가 사용되는 경우가 종종 있습니다. 물론 이 책은 GPU 프로그래밍과 관련이 없으므로 여기에서는 GPU 가속화의 가능성을 지적하고 특정 연습은 검증을 위해 독자에게 맡깁니다. 유사한 작업을 토대로, GPU의 고밀도 깊이 추정은 메인 스트림 GPU에서 실시간으로 수행됩니다.

13.4.5 기타 개선 사항

사실, 우리는 또한 다음과 같이 이 루틴을 개선 할 수있는 많은 솔루션을 제안 할 수 있습니다 :

1. 각 픽셀은 완전히 독립적이므로, 픽셀의 깊이가 작고 에지가 큰 경우가 있을 수 있습니다. 우리는 깊이 맵에서 인접한 깊이 변화가 너무 크지 않아서 깊이 추정에 공간 정규항을 추가한다고 가정할 수있다. 이 접근법은 결과 깊이 맵을 더 매끄럽게 만듭니다.
2. 우리는 Outlier를 명시적으로 처리하지 않았습니다. 실제로, 폐색, 조명 및 모션 블러와 같은 다양한 요인으로 인해 각 픽셀에 대해 성공적인 일치를 유지하는 것은 불가능합니다. 데모 프로그램의 실행에서 NCC가 특정 값보다 크면 성공적인 일치가 고려되며 불일치의 경우는 고려되지 않습니다.

오류 정합을 처리하는 데는 여러 가지 방법이 있습니다. 예를 들어, [90]에서 제안 된 uniform-Gaussian mixture model 하의 깊이 필터는 내부 inlier와 outlier를 명시 적으로 구별하고 확률적 모델링을 수행하여 outlier 데이터를 보다 잘 처리합니다. 그러나 이런 유형의 필터 이론은 더 복잡하며,이 책은 너무 많이 참여하기를 원하지 않습니다. 독자는 원고를 읽을 수 있습니다.

위의 논의에서 알 수 있듯이 많은 개선이있을 수 있습니다. 접근 방식의 각 단계를 신중하게 개선한다면 dense 매핑을 위한 좋은 계획을 얻을 수 있습니다. 그러나 우리가 논의한 바와 같이, 픽셀 그라데이션과 epipolar line 방향 (및 병렬 경우)의 상관 관계와 같은 환경 텍스처에 대한 의존성과 같은 이론적인 어려움에는 몇 가지 문제가 있습니다. 이러한 문제는 코드 구현을 조정하여 해결하기가 매우 어렵습니다. 그래서 지금까지 스테레오 카메라와 이동하는 단안 카메라를 이용하여 dense 지도를 만들 수 있지만 우리는 일반적으로 환경 질감과 조명에 너무 의존적이라고 생각합니다.

13.5 RGB-D dense 매핑

단안 및 양안 렌즈를 사용하는 고밀도 재구성 외에도 적용 가능한 경우 RGB-D 카메라가 더 나은 선택입니다. 이전 강의에서 자세히 설명한 깊이 추정 문제는 추정을 위해 많은 양의 계산 자원을 소모하지 않고 RGB-D 카메라의 센서에서 하드웨어로 완벽하게 측정 할 수 있습니다. 또한 RGB-D의 Structured light 또는 Time-of-Flight 원리는 텍스처로부터 깊이 데이터의 독립성을 보장합니다. 단단한 물체를 향하더라도 빛을 반사 할 수 있는 한 깊이를 측정 할 수 있습니다. 이것은 RGB-D 센서의 큰 장점이기도합니다.

Dense 매핑을 위해 RGB-D를 사용하는 것은 상대적으로 쉽습니다. 그러나 지도의 형식에 따라 기본 스트림을 구성하는 여러 가지 다른 방법이 있습니다. 가장 직관적이고 간단한 방법은 추정된 카메라 포즈를 기반으로 RGB-D 데이터를 포인트 클라우드로 변환한 다음 함께 결합하고 마지막으로 이산 포인트로 구성된 점 구름 맵을 얻는 것입니다. 이를 바탕으로 외형에 대한 추가 요구 사항이 있고 객체의 표면을 평가하려는 경우 삼각형 메쉬(Mesh)와 표면 요소(Surfel)을 사용하여 구조를 만들 수 있습니다. 반면에 지도의 장애물 정보를 알고 지도를 탐색하려면 Voxel을 사용하여 점유지도를 만들 수 있습니다.

우리는 많은 새로운 개념을 소개하는 것 같습니다. 걱정하지 마십시오. 천천히 하나씩 소개 할 것입니다. 실험에 적합한 일부 실험의 경우 평소와 같이 몇 가지 데모 프로그램을 제공합니다. RGB-D 매핑에 관련된 이론적 지식은 그다지 크지 않기 때문에 다음 섹션을 실용적인 부분에서 직접 소개합니다. GPU 매핑은 이 책의 범위를 벗어나므로 간단히 원리를 설명하고 설명하지는 않겠습니다.

13.5.1 실습 : 포인트 클라우드지도

먼저, 가장 단순한 포인트 클라우드에 대해 이야기 해 봅시다. 소위 포인트 클라우드는 일련의 이산 포인트로 표현되는 지도입니다. 가장 기본적인 포인트는 x, y, z 3차원 좌표를 포함하며 r, g, b의 색상 정보도 가질 수 있습니다. RGB-D 카메라는 색상 및 깊이 맵을 제공하므로 카메라의 내부 파라미터에서 RGB-D 포인트 클라우드를 쉽게 계산할 수 있습니다. 어떤 방법으로 카메라 포즈를 취하면 단순히 포인트 클라우드를 직접 추가하여 전역 점 구름을 얻을 수 있습니다. 이 책의 5.4.2 절에는 카메라의 내부 및 외부 매개 변수를 통해 점 구름을 연결하는 예제가 나와 있습니다. 그러나 이 예제는 주로 카메라의 내부 및 외부 매개 변수를 이해하는 독자를위한 것이며, 실제 구축시 더 나은 시각 효과를 얻기

위해 포인트 클라우드에 일부 필터링을 추가합니다. 이 프로그램에서는 주로 외부 포인트 제거 필터와 다운 샘플링 필터라는 두 종류의 필터를 사용합니다. 샘플 프로그램의 코드는 다음과 같습니다.

slambook/ch13/dense_RGBD/pointcloud_mapping.cpp

우리의 생각은 많이 변하지 않았습니다. 주요 차이점은 다음과 같습니다.

1. 각 프레임에 포인트 클라우드를 생성 할 때 심도 값이 너무 크거나 유효하지 않은 지점을 제거하십시오. 이는 주로 Kinect의 유효 범위에 기인하며 초과 범위 이후의 깊이 값에는 큰 오차가 있습니다.
2. 통계 필터 방법을 사용하여 특이치를 제거하십시오. 필터는 가장 가까운 N 점에서 각 점의 거리 값 분포를 계산하여 평균 거리가 너무 큰 점을 제거합니다. 이러한 방식으로, 우리는 "서로 붙어있는"지점을 유지하고 격리 된 잡음 지점을 제거합니다.
3. 마지막으로, 다운 샘플링은 복셀 필터를 사용하여 수행됩니다. 다수의 시야각의 중첩 된 뷰가 있기 때문에, 중첩 영역에는 많은 수의 밀집된 점이있다. 이것은 많은 메모리 공간을 차지할 것입니다. 복셀 필터링은 특정 크기 큐브 (또는 복셀)에 단 하나의 점이 있음을 보장합니다. 이는 3차원 공간을 다운 샘플링하는 것과 동일하며 많은 저장 공간을 절약합니다.

그림 13-8은 필터링 전후의 비교를 보여줍니다. 왼쪽에는 다섯 번째 프로그램에서 생성 된 점 구름 맵이 있고 오른쪽에는 필터링 된 점 구름 맵이 있습니다. 흰색 상자 섹션을 보면 필터링하기 전에 노이즈에 의해 생성 된 많은 고립 된 포인트가 있음을 알 수 있습니다. 통계가 제거 된 후, 우리는 노이즈을 제거하고 전체지도를 더 깨끗하게 만들었습니다. 반면에 복셀 필터에서는 해상도를 0.01로 조정합니다. 즉, 1 입방 센티미터 당 1 포인트가 있음을 의미합니다. 이것은 상대적으로 높은 해상도이기 때문에 스크린 샷에서 맵의 차이를 느낄 수는 없지만 프로그램 출력에서 점의 수는 크게 줄어 듭니다 (900,000 포인트에서 440,000 포인트로 즉, 반은 제거됨)

그림 13-8 필터링 전후의 비교

포인트 클라우드 맵은 우리에게 환경 개요를 제공하는 보다 기본적인 시각적 맵을 제공합니다. 3D로 저장되므로 장면의 구석을 빠르게 탐색하고 장면을 돌아 다닐 수

있습니다. 포인트 클라우드의 큰 장점은 추가 처리없이 RGB-D 이미지에서 효율적으로 직접 생성 할 수 있다는 것입니다. 필터링 작업도 매우 직관적이며 처리 효율성이 허용됩니다.
그러나 점 구름을 사용하여 지도를 표현하는 것은 여전히 매우 기본적인 작업이며 지도의 필요에 따라 점 구름 지도를 충족시킬 수 있는지 알아볼 수 있습니다.

1. 위치 요구 사항: frontend, visual odometry가 처리되는 방식에 따라 다릅니다. 포인트 클라우드로 인해 특징 지점 기반 Visual odometry인 경우 특징점 정보는 저장되지 않으므로 특징점 기반 위치 추정 방법에 사용할 수 없습니다.
   Frontend가 포인트 클라우드 ICP인 경우 로컬 포인트 클라우드를 글로벌 포인트 클라우드 ICP로 간주하여 포즈를 추정합니다. 그러나 이를 위해서는 보다 정확도가 높은 글로벌 포인트 클라우드가 필요합니다. 포인트 클라우드를 다루는 우리는 포인트 클라우드 자체를 최적화하지 않았으므로 충분하지 않습니다.
2. 탐색 및 장애물 회피 요구 사항 : 점군은 탐색 및 장애물 회피에 직접 사용되지 않습니다. 순수 점 구름은 "장애물이 있는지 여부"라는 메시지를 표현할 수 없으며 탐색 및 장애물 회피의 기본 요구 사항인 포인트 클라우드에서 "공간 점이 사용되었는지 여부"를 쿼리 할 수 없습니다. 그러나 탐색 및 장애물 회피에 더 적합한 지도 양식을 얻기 위해 점 구름 기준으로 처리 할 수 있습니다.
3. 시각화 및 상호 작용 : 기본적인 시각화 및 상호 작용 기능. 장면의 모습을 볼 수 있으며 장면을 탐색 할 수도 있습니다. 시각적인 관점에서 볼 때 포인트 클라우드는 불연속 점만 포함하고 객체 표면 정보 (예 : 정상)가 없으므로 사람의 일반적인 시각 체계와 일치하지 않습니다. 예를 들어, 포인트 클라우드 맵의 객체는 앞면과 뒷면에서 동일하며 객체의 뒷편을 볼 수 있습니다. 객체의 표면에 대한 정보가 없기 때문에 이것들은 우리의 일상 경험과 일치하지 않습니다.

요약하자면 포인트 클라우드 맵은 "기본"으로, 이는 센서가 읽은 원시 데이터에 더 가깝다는 것을 의미합니다. 몇 가지 기본 기능이 있지만 일반적으로 디버깅 및 기본 디스플레이에 사용되며 응용 프로그램에 적합하지 않습니다. 지도에 고급 기능을 추가하려면 포인트 클라우드 맵을 활용하여 다른 형태의 맵을 만드는 것이 좋습니다. 예를 네비게이션 알고리즘을 위해 포인트 클라우드에서 그리드 맵 (Occupancy Grid)을 생성하여 포인트가 통과 할 수 있는지 여부를 쿼리 할 수 있습니다. 다른 예를 들어, SfM에서 일반적으로 사용되는 Poisson reconstruction [94] 방법은 객체의 표면 정보를 얻기 위해 기본 점 구름을 통해 객체 그리드 맵을 재구성 할 수 있습니다. 포아송 (Poisson) 재건 외에도 Surfel은 패싯(facet, 작은 표면)을 지도의 기본 단위로 사용하여 물체의 표면을 표현하는 방법으로 아름다운 시각적 지도를 만들 수 있습니다.

그림 13-9는 Poisson 재구성과 Surfel의 예를 보여줍니다.이 점은 순수 점 구름 매핑보다 훨씬 뛰어날 수 있으며 점 구름으로 모두 빌드 될 수 있습니다. 포인트 클라우드으로부터 변환 된 대부분의지도 형태는 PCL 라이브러리에 제공되며 관심있는 독자는 PCL 라이브러리의 내용을 탐색 할 수 있습니다. 이 책의 출발점으로 각 형태의 지도를 자세히 다루지는 않을 것입니다.

그림 13-9 포아송 재구성 및 Surfel.

13.5.2 옥트리지도

다음은 네비게이션에서 일반적으로 사용되며 더 나은 압축 성능을 가진 맵 양식을 설명합니다.
포인트 클라우드 맵에서 우리는 3차원 구조를 가지고 있고 복셀 필터링을 수행하여 해상도를 조정하지만 포인트 클라우드에는 몇 가지 명백한 결함이 있습니다.

- 점 구름 맵은 대개 매우 커서 pcd 파일이 커집니다. 640 x 480 픽셀 이미지는 300,000 개의 공간 지점을 생성하며 많은 저장 공간이 필요합니다. 일부 필터링 후에도 pcd 파일은 매우 큽니다. 점 구름 맵은 많은 불필요한 세부 정보를 제공합니다. 우리는 특히 카펫의 주름과 그림자의 그림자에 대해 이러한 사항에 신경 쓰지 않습니다. 그것들을 지도에 두는 것은 공간 낭비입니다. 이러한 공간의 점유로 인해, 우리는 해상도를 줄이지 않는 한 큰 환경을 제한된 메모리로 모델링 할 수 없습니다. 그러나 해상도를 줄이면 맵 품질이 저하됩니다. 지도를 압축하고 중복 된 정보를 버릴 수있는 방법이 있습니까?
- 점 구름 맵은 움직이는 물체를 처리 할 수 없습니다. 우리는 접근 방식에 "포인트 추가"기능을 가지고 있기 때문에 "포인트가 사라지면 제거하십시오." 실제 환경에서 움직이는 물체의 편재성은 포인트 클라우드 맵을 덜 실용적으로 만듭니다.

우리가 다음에 소개 할 것은 유연하고 압축 된 최신 맵 형식입니다 : Octomap [96]. 3D 공간을 여러 개의 작은 정사각형 (또는 복셀)으로 모델링하는 것이 일반적입니다. 작은 사각형의 각면을 평균하여 두 조각으로 자르면 작은 사각형은 같은 크기의 8 개의 작은 사각형이됩니다. 이 단계는 최종 블록 크기가 최고 수준의 모델링에 도달 할 때까지 계속 반복 될 수 있습니다. 이 과정에서 "작은 정사각형을 같은 크기의 8 개로 분할하는"과정은 "한 노드에서 여덟 개의 자식 노드로 확장"된 것으로 간주되며 가장 큰 공간에서 가장 작은 공간으로 세분하는 전체 프로세스가 Octo tree를 나타냅니다.

그림 13-10에서 볼 수 있듯이 왼쪽에는 작은 사각형이 될 때까지 8 개의 조각으로 균등하게 나뉘어진 큰 큐브가 표시됩니다. 따라서 전체 큰 블록은 루트 노드로 간주 될 수 있으며 가장 작은 블록은 "리프 노드"로 간주 될 수 있습니다. 따라서 octree에서 다음 레벨의 노드에서 한 수준 위로 이동할 때지도의 볼륨을 8 배로 확장 할 수 있습니다. 간단한

계산을 해보 죠 : 리프 노드의 블록 크기가 $1\ \mathrm{cm}^3$ 이면 옥트 트리를 10개의 레이어로 제한하면 전체 모델 가능한 볼륨은 약 $8^{10}\mathrm{cm}^3 = 1,073\mathrm{m}^3$ 입니다. 방을 모델링하기에 충분합니다. 볼륨은 깊이와 기하 급수적으로 관련되어 있으므로 모델링 된 볼륨은 깊이를 더 많이 사용하면 매우 빠르게 증가합니다.

독자는 포인트 클라우드의 복셀 필터에서 복셀의 한 점만 제한 할 수 있을지 궁금해 할 것입니다. 왜 우리는 포인트 클라우드가 공간을 차지하고 옥토 트리가 공간을 절약한다고 말합니까? 이것은 옥트리에 노드에 노드가 있는지 여부에 대한 정보를 저장하기 때문입니다. 그러나 차이점은 블록의 모든 하위 노드가 점유되었거나 점유되지 않았을 때 노드를 확장 할 필요가 없다는 것입니다. 예를 들어 지도가 처음에는 비어있는 경우 풀 트리가 아닌 루트 노드 하나만 있으면됩니다. 지도에 정보를 추가 할 때 실제 객체는 종종 함께 연결되기 때문에 공백은 종종 함께 연결되므로 대부분의 옥트 트리 노드를 리프 수준까지 확장 할 필요가 없습니다. 따라서 옥트리는 포인트 클라우드보다 많은 저장 공간을 절약합니다.

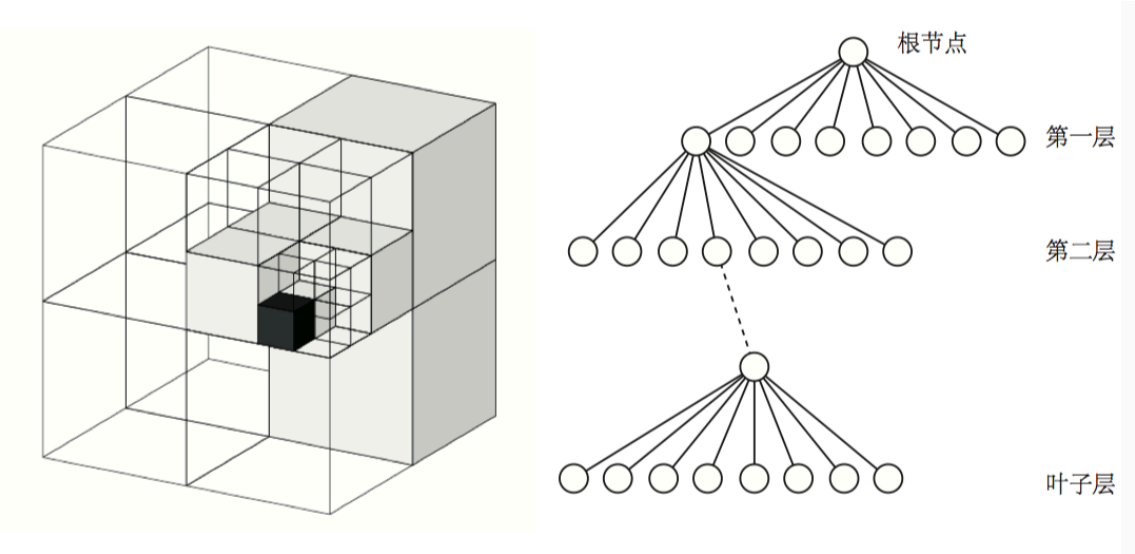


그림 13-10 octree 다이어그램.

octree의 노드는 점유 여부에 대한 정보를 저장한다고 합니다. 포인트 클라우드 레벨에서 자연스럽게 empty공간을 0으로, occupied 공간을 1로 사용할 수 있습니다. 이 0-1 표현은 1 비트에 저장되어 공간을 절약 할 수 있지만 너무 간단합니다. 노이즈의 영향으로 어떤 지점은 잠시 동안 0, 잠시 동안 1, 대부분 시간 동안 0, 시간의 작은 부분에 대해 1 또는 "예"및 "아니오"이외에 볼 수 있습니다. "알 수없는"상태 등의 값을 가질 수 있습니다. 우리는 노드가 점유되었는지 여부를 확률적으로 표현할 것입니다. 예를 들어, 표현하기 위해 부동 소수점 $x \in [0, 1]$을 사용합니다. 이 x는 초기 값 0.5로 시작합니다. 계속 점유하고 있다면, 이 값을 증가 시키십시오. 반대로, 그것이 공백임을 계속 관찰한다면, 계속 감소 시키십시오.

이 방법으로 지도에서 장애물 정보를 동적으로 모델링합니다. 그러나 현재 접근법은 작은 문제가 있습니다. 즉, x가 지속적으로 증가하거나 감소하면 [0, 1] 범위를 벗어나 처리에 불편을 초래할 수 있습니다. 그래서 노드가 점령 될 확률을 사용하는 대신 확률 로그

(Log-odds)를 사용하여 노드를 설명합니다. y ∈ R을 확률 대수로, x를 0 ~ 1의 확률로 놓고, 이들 사이의 변환은 로짓 변환에 의해 기술된다 :

$$y = \text{logit}(x) = \log\left(\frac{x}{1-x}\right). \tag{13.17}$$

거꾸로

$$x = \text{logit}^{-1}(y) = \frac{\exp(y)}{\exp(y)+1}. \tag{13.18}$$

y가 -∞에서 + ∞로 변하면 x는 0에서 1로 변한다는 것을 알 수있다. y가 0이면 x는 0.5가됩니다. 따라서 y를 저장하여 노드가 점유되었는지 여부를 나타낼 수 있습니다. 계속 관찰하면서 y를 1 만큼 증가시키고, 그렇지 않으면 y를 1만큼 감소시킵니다. 확률을 쿼리할 때 역 로그 변환을 사용하여 y를 확률로 변환합니다. 수학적 용어로, 노드를 n, 관측 된 데이터를 z 라하자. 그리고, 처음부터 카메라 t까지의 노드의 확률 로그를 $L(n|z_{1:t})$, 시각 t + 1을

$$L(n|z_{1:t+1}) = L(n|z_{1:t-1}) + L(n|z_t). \tag{13.19}$$

확률론적 형태가 아닌 확률적 형태로 쓰여진다면, 그것은 좀 더 복잡해질 것입니다 :

$$P(n|z_{1:T}) = \left[1 + \frac{1-P(n|z_T)}{P(n|z_T)}\frac{1-P(n|z_{1:T-1})}{P(n|z_{1:T-1})}\frac{P(n)}{1-P(n)}\right]^{-1}. \tag{13.20}$$

로그 확률을 사용하여 RGB-D 데이터를 기반으로 전체 옥트리 트리 맵을 업데이트 할 수 있습니다. RGB-D 이미지에서 깊이 d가 있는 픽셀을 관찰한다고 가정하면, 깊이 값에 해당하는 공간 지점에서 점령된 데이터를 관찰하고 카메라의 광학 중심에서 이 점까지의 데이터를 관찰했습니다 선 세그먼트에는 객체가 없어야합니다 (그렇지 않으면 차단됩니다). 이 정보를 이용하여 octree 맵을 매우 잘 업데이트하고 모션 구조를 처리 할 수 있습니다.

13.5.3 연습 : Octomap

다음은 Octomap 매핑 프로세스의 데모입니다. 먼저 독자에게 Octomap 라이브러리 (https://github.com/OctoMap/octomap)를 설치하도록 요청하십시오. Octomap 라이브러리는 주로 octmap 맵과 octovis (비주얼 라이저)를 포함하며, 둘 다 cmake 프로젝트입니다. 주로 doxygen을 사용하며 다음 명령으로 설치할 수 있습니다.

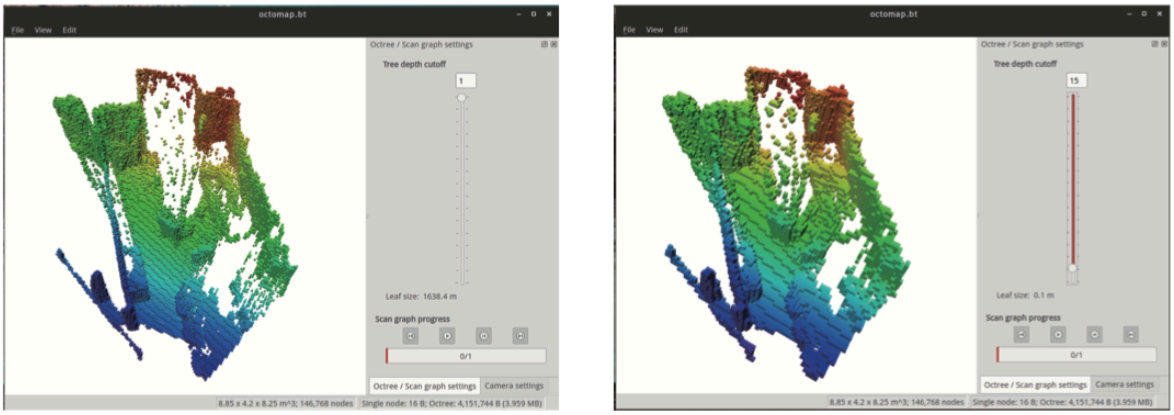
```
sudo apt-get install doxygen
```

처음 5 개의 이미지로부터 octree 맵을 생성하는 방법을 직접 설명하고 그려 봅시다.

slambook/ch13/dense_RGBD/octomap_mapping.cpp

octomap :: OcTree를 사용하여 전체 맵을 작성했습니다. 사실, octomap은 다양한 octree를 제공합니다. 맵이 있으며 각 노드가 수행해야하는 변수를 정의 할 수 있습니다. 단순화를 위해, 우리는 색 정보없이 가장 기본적인 옥트리 트리 맵을 사용했다.

Ocotmap은 내부적으로 포인트 클라우드 구조를 제공합니다. PCL의 점 구름보다 약간 간단하며 점의 공간 위치 정보 만 전달합니다. RGB-D 이미지와 카메라 자세 정보를 바탕으로, 우리는 먼저 포인트의 좌표를 월드 좌표로 옮긴 다음 옥토 맵의 포인트 클라우드에 넣고 마지막으로 그것을 옥트리 트리 맵에 넘깁니다. 그 후, octomap은 이전에 소개 된 투영 정보를 기반으로 내부 점유 확률을 업데이트하고 마지막으로 이를 압축 된 octree 맵으로 저장합니다. 생성된 맵을 octomap.bt 파일로 저장합니다. 이전에 octovis를 컴파일 할 때 실제로 octovis라는 시각화 프로그램을 설치했습니다. 이제지도 파일을 열어 호출하면지도의 모습을 볼 수 있습니다.
그림 13-11은 우리가 만든 지도의 결과를 보여줍니다. 지도에 색상 정보를 추가하지 않았으므로 처음 지도를 열면 회색으로 표시됩니다. 독자는 지도보기, 회전, 크기 조정 등을 포함하여 octovis 인터페이스에 익숙해 질 수 있습니다.



八叉树地图（0.05米分辨率）　　八叉树地图（0.1米分辨率）

그림 13-11 다른 해상도의 octree 맵 결과 표시

오른쪽에는 옥트리에 대한 심도 한계 막대가 있으며,지도의 해상도를 조정할 수 있습니다. 우리의 구조에 사용 된 기본 깊이가 16 층이기 때문에, 여기서는 16 개의 레이어, 즉 가장 높은 해상도를 보여줍니다. 즉, 각각의 작은 블록의 측면 길이는 0.05 미터입니다. 깊이를 한 층 씩 줄이면 팔진의 리프 노드가 한 층 올라가고 각 작은 조각의 길이는 두 배가 되어 0.1m가 됩니다. 보시다시피, 우리는 다른 경우에 맞게 지도 해상도를 쉽게 조정할 수 있습니다.
Octomap은 탐색 할 수있는 곳이 있습니다. 예를 들어 지도에서 탐색하는 방법을 설계하기 위해 어느 지점의 점유 확률을 쉽게 쿼리 할 수 있습니다 [97]. 또한 독자는 포인트 클라우드 맵과 팔 트리 맵의 파일 크기를 비교할 수 있습니다. 이전 섹션에서 생성 된 포인트 클라우드 맵은 디스크 파일 크기가 6.9MB이고 옥토 맵은 56KB 밖에 없으며 포인트 클라우드 맵의 1 %에도 미치지 못합니다. 더 큰 장면을 효과적으로 모델링 할 수 있습니다.

13.6 TSDF 지도 및 퓨전 시리즈

참고자료: https://youtu.be/0xOniUK89v8?list=PLTBdjV_4f-EKBCUs1HmMtsnXv4JUoFrzg

이 강의 마지막에는 SLAM과 매우 유사하지만 약간 다른 관점인 실시간 3D 재구성에 대한 연구 방향을 소개합니다. 이 섹션에서는 GPU 프로그래밍에 대해 설명하고 참조 예제를 제공하지 않으므로 선택적 읽기 자료입니다.

이전 지도 모델에서는 위치 추정이 주요 본체입니다. 맵의 스티칭은 후속 처리 단계로서 SLAM 프레임 워크에 배치됩니다. 이 프레임 워크가 주류가 되는 이유는 위치 알고리즘이 실시간 요구 사항을 충족 할 수 있고 지도의 처리가 실시간 응답없이 키 프레임에서 처리 될 수 있기 때문입니다.

포지셔닝은 일반적으로 가볍습니다. 특히 sparse 피쳐 또는 direct method를 사용하는 경우에는 맵의 표현과 저장이 매우 중요합니다. 크기와 컴퓨팅 요구 사항이 크기 때문에 실시간 처리에 도움이 되지 않습니다. 특히 조밀한 지도는 종종 키 프레임 수준에서만 계산할 수 있습니다.

그러나 현재의 관행에서는 dense 맵을 최적화하지 않았습니다. 예를 들어, 두 이미지에서 같은 의자가 관찰되면, 두 이미지의 포즈에 따라 두 점 구름을 겹쳐서 맵을 생성합니다. 자세 추정은 대개 오류가 발생하기 쉽기 때문에 직접적인 스티칭은 종종 정확하지 않습니다. 예를 들어 같은 의자의 점 구름을 완벽하게 겹칠 수는 없습니다. 이 때, 의자에 있는 두 개의 유령(완벽하게 겹쳐지지 않아서 보이는 잔상)이 지도에 나타납니다. 이 현상을 때로는 "고스트 (ghosting)"라고합니다.

이 현상은 분명히 우리가 원하는 것이 아닙니다. 우리는 지도에 대한 이해와 일치하여 재구성 결과가 부드럽고 완벽하기를 바랍니다. 3D dense 재구성은 정확한 지도의 재구성을 주요 대상으로하기 때문에 기본적으로 가속을 위해 GPU를 사용해야하며 병렬 가속을 위해 매우 진보 된 GPU 또는 다중 GPU가 필요합니다. 보통 더 무거운 컴퓨팅 장치가 필요합니다. 반대로 SLAM은 경량화 및 소형화로 나아가고 있으며 일부 프로그램은 매핑 및 Loop closure detection 섹션을 포기하여 시각적 주행 거리만 남겨 둡니다. 실시간 재구성은 대규모의 동적 장면을 재구성하는 쪽으로 이동하고 있습니다.

RGB-D 센서의 출현 이후, RGB-D 이미지를 사용하는 실시간 재구성은 차례로 생산되는 중요한 개발 방향을 형성했습니다. KinectFusion, DynamicFusion, ElasticFusion, Fusion4D, VolumnDeform. 그 중에서도 KinectFusion은 기본 모델 재구성을 완료하지만 작은 장면에 대해서만 수행하고 이후의 작업은 크고 움직이고 변형된 장면으로 확장하는 것입니다. 우리는 그것들을 대규모 작업의 실시간 재구성으로 간주하지만, 다양성으로 인해 각각의 작동 방식을 자세하게 논의하는 것은 불가능합니다. 그림 13-12는 재구성 결과 중 일부를 보여주는데,이 모델링 결과는 간단히 점을 연결하는 것보다 매우 상세하고 훨씬 더 섬세한 것을 알 수 있습니다.

우리는 대표적인 TSDF 지도를 소개 할 것입니다. TSDF는 Truncated Signed Distance Function의 약어로, 절단된 기호 거리 함수로 변환 할 수 있습니다.

Octree와 마찬가지로 TSDF 맵은 그림 13-13과 같이 블록 맵이기도합니다. 먼저 모델링 할 3D 공간 (예 : 3x 3x 3m3)을 선택하고 공간을 특정 해상도에 따라 많은 작은 블록으로 분할한 다음 각 작은 블록 내부에 정보를 저장합니다. 차이점은 TSDF 맵은 메모리가 아닌 GPU메모리에 저장된다는 것입니다. GPU의 병렬 특성으로 인해 메모리 영역을 가로지르고 직렬화되어야하는 CPU와 달리 각 복셀을 병렬로 계산하고 업데이트 할 수 있습니다.

각 TSDF 복셀 내에서 가장 가까운 객체의 표면에서 해당 복셀까지의 거리가 저장됩니다. 작은 조각이 대상 표면 앞에 있으면 양수 값을 가지며, 그렇지 않으면 작은 조각이 표면 뒤에있는 경우 음수입니다. 물체의 표면은 대개 매우 얇은 층이기 때문에 너무 크고 작은 값은 1과 -1로 취해 지므로 절단 이후의 거리가 얻어지며이를 TSDF라고합니다. 정의에 따르면, TSDF 값이 0 인 위치는 표면 그 자체입니다. 또는 수치 오류가 있기 때문에 TSDF가

음수 부호에서 양수 부호로 바뀌는 위치가 지표 자체입니다. 그림 13-13의 아래 부분에서 TSDF 변경 기호에 나타나는 얼굴과 비슷한 표면을 볼 수 있습니다.

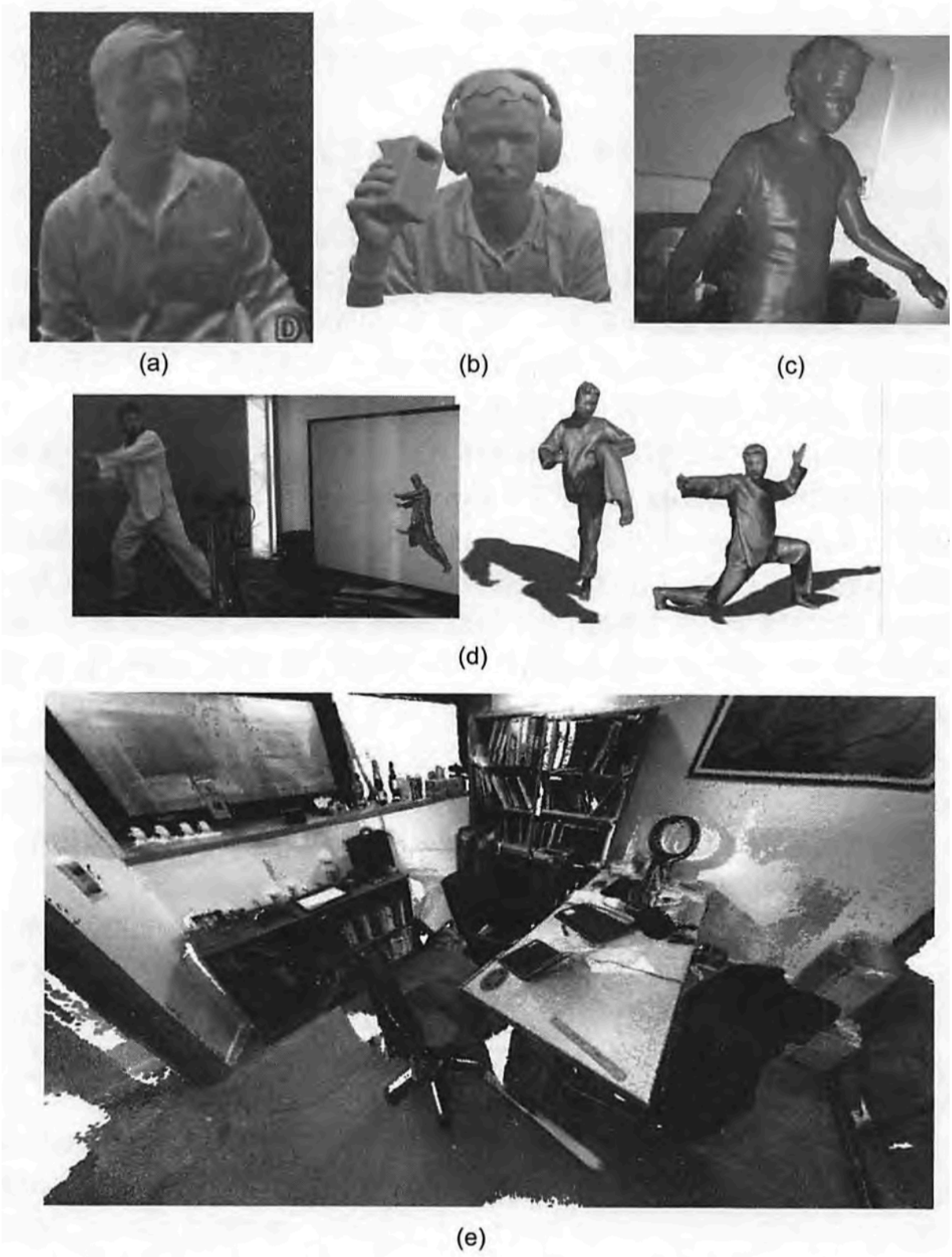
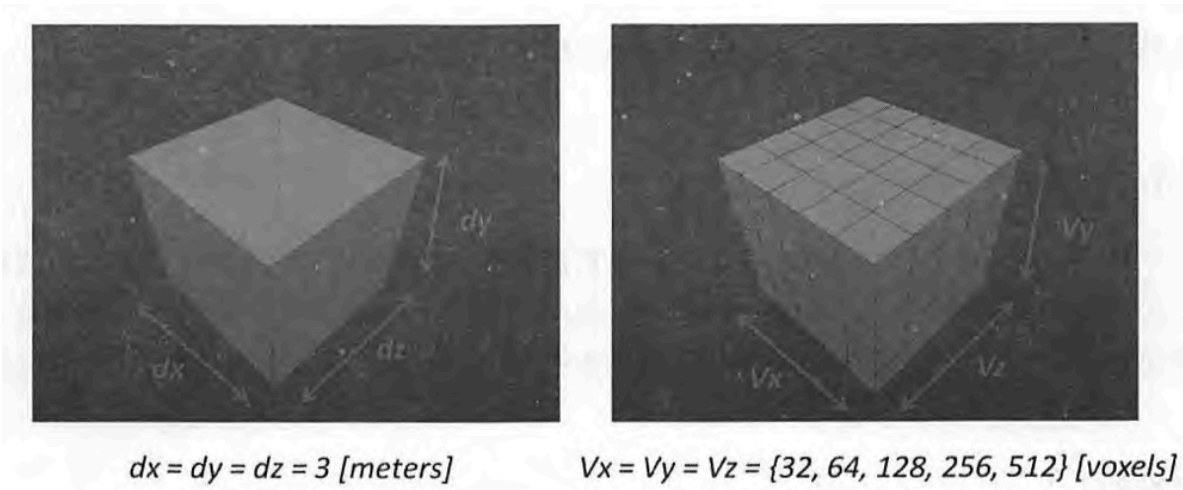


그림 13-12다양한 실시간 3D 재구성 모델. (a) KinectFusion (b) DynamicFusion (c) Volumn Deform (d) Fusion4D (e) ElasticFusion



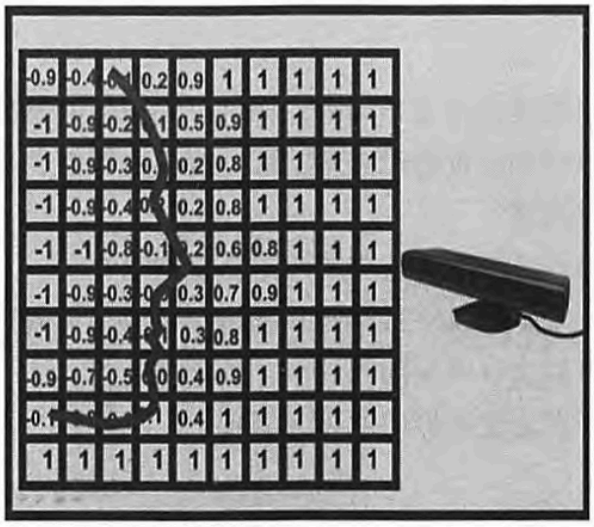


그림 13-13 TSDF 볼륨

TSDF에는 SLAM과 매우 유사한 "위치 추정"및 "구축"의 두 가지 문제점이 있지만이 책의 이전 양식과는 약간 다른 양식이 있습니다. 여기서, 위치 추정 문제는 주로 GPU의 현재 RGB-D 이미지를 TSDF 맵과 비교하여 카메라 포즈를 추정하는 방법을 나타냅니다. 매핑 문제는 추정 된 카메라 포즈를 기반으로 TSDF 맵을 업데이트하는 방법입니다. 전통적으로 RGB-D 이미지에서 베이지안 필터를 수행하여 깊이 맵에서 노이즈를 제거합니다.

TSDF의 위치는 위에서 설명한 ICP와 유사하지만 GPU의 병렬화로 인해 전통적인 Visual odometry에서와 같이 특징점을 계산할 필요없이 전체 깊이 맵과 TSDF 맵에서 ICP 계산을 수행 할 수 있습니다. 동시에 TSDF에는 색 정보가 없으므로 색상 맵을 사용하지 않고 자세 추정을 완료하기 위해 깊이 맵만 사용할 수 있습니다. 이는 시각적 주행 거리 계산 방법이 조명과 텍스처에 어느 정도 의존하지 않으므로 RGB-D 재건은 더욱 견고합니다. 반면에

매핑 부분은 TSDF의 값을 병렬로 업데이트하여 추정된 표면을 더 부드럽고 안정적으로 만드는 프로세스입니다. 우리는 GPU 관련 내용을 너무 많이 소개하지 않으므로 구체적인 방법은 정교하지 않습니다. 관심있는 독자들은 해당 논문을 참고하십시오.

13.7 요약

이 강연에서는 몇 가지 일반적인 유형의 지도, 특히 dense 지도를 소개합니다. 단안 또는 양안을 기반으로 한 dense 지도를 볼 수 있지만 RGB-D지도는 더 쉽고 안정적입니다. 이 강의의 지도는 메트릭 지도에 초점을 두고 있으며 토폴로지 지도 양식은 SLAM 연구와 매우 다르기 때문에 자세히 논의하지 않습니다.

연습문제

1. (13.6)을 유도해봅시다.
2. 이 강좌의 치밀도 추정치를 반 밀집도로 변경합니다. 먼저 분명한 그라디언트를 스크리닝 할 수 있습니다.
3.이 프리젠테이션의 단안 조밀 재구성 코드를 양수의 깊이에서 Inverse depth로 변경하고 아핀 변환을 추가합니다. 실험이 개선 되었습니까?
4. octree에서 탐색 또는 경로 계획을 보여줄 수 있습니까?
5. 연구 논문 [98]은 TSDF지도가 포즈 추정 및 업데이트를 수행하는 방법을 탐구한다. 이전에 얘기 한 위치 추정 알고리즘과 다른 점은 무엇입니까?
6. 연구 균일 성 - 가우시안 혼합물 필터의 원리와 구현.