석사학위논문

다시점 영상에서 객체 추적 및 세그멘테이션을 위한 프레임워크

2026년

한 성 대 학 교 대 학 원

A I 응 용 학 과

A I 응 용 학 전 공

용 지 현

석사학위논문
지도교수 김명선

다시점 영상에서 객체 추적 및 세그멘테이션을 위한 프레임워크

Framework for Multi-view Object Tracking and Segmentation

2025년 12월 일

한 성 대 학 교 대 학 원

A I 응 용 학 과

A I 응 용 학 전 공

용 지 현

석사학위논문
지도교수 김명선

다시점 영상에서 객체 추적 및 세그멘테이션을 위한 프레임워크

Framework for Multi-view Object Tracking and Segmentation

위 논문을 공학 석사학위 논문으로 제출함

2025년 12월 일

한 성 대 학 교 대 학 원

A I 응 용 학 과

A I 응 용 학 전 공

용 지 현

용지현의 공학 석사학위 논문을 인준함

2025년 12월 일

심사위원장 강미선 (인)

심 사 위 원 이웅희 (인)

심 사 위 원 김명선 (인)

국 문 초 록

다시점 영상에서 객체 추적 및 세그멘테이션을 위한 프레임워크

한 성 대 학 교 대 학 원
A I 응 용 학 과
A I 응 용 학 전 공
용 지 현

단일 카메라 기반 영상은 시야 제한과 객체 가림(Occlusion)으로 인해 복잡한 장면에서의 객체 인식 및 추적 성능이 제한된다. 다시점 영상은 이러한 한계를 완화할 수 있으나, 한 프레임에서 처리해야 하는 이미지 수가 증가함에 따라 실시간 객체 추적 및 세그멘테이션에서는 높은 계산 복잡도와 방대한 데이터를 처리해야 하는 문제가 발생한다. 본 논문에서는 다시점 영상의 대표적 사례인 플렌옵틱 이미지와 멀티 카메라 이미지를 연구 대상으로 활용하여 이에 적합한 효율적인 객체 추적 및 세그멘테이션 프레임워크를 제안한다. 먼저, 플렌옵틱 이미지 환경에서는 기존 2D 비디오 기반 객체 추적기를 플렌옵틱 구조에 적합하도록 재구성하였다. 또한, 한 프레임 내 다수의 포컬 플레인 이미지 중 필수 정보만을 선별하는 포컬 플레인 이미지 선택 전략을 도입하고, 프레임워크 내부의 딥러닝 기반 특징 추출 모듈과 전처리 단계를 멀티코어로 구성된 CPU와 GPU 환경에서 병렬화하여 계산 효율을 극대화하였다. 또한 본 논문에서는 멀티 카메라 이미지 환경에서는 low-rank

projection matrix를 적용한 경량화된 Video Multi-Object Segmenter와 경량화된 Mask refiner를 원본 모델과 동적으로 조합하여 사용하는 효율적인 세그멘테이션 프레임워크를 제안한다. 연속된 프레임 간 코사인 유사도를 적용하여 현재 프레임의 이미지들의 경량화 정도를 적응적으로 조정함으로써, 더욱 fine-grained한 모델 적용을 가능하게 한다. 이때 다중 GPU 환경에서는 경량화 모델들과 원본 모델들이 혼재되어 수행된다. 이로 인해 발생하는 GPU 간 실행 시간 불균형은 프레임 단위 지연을 초래할 수 있다. 이를 해결하기 위하여 매 프레임마다 시스템 내부 GPU들의 하드웨어적인 연결 상태를 고려하여 GPU 간 데이터 이동을 최적으로 수행한다. 이를 통해 각 GPU의 프레임당 세그멘테이션 실행 시간을 균형적으로 유지함으로써, 전체 시스템의 평균 프레임 실행 시간을 최소화하도록 설계하였다. 실험 결과, 제안한 플렌옵틱 이미지 기반 추적 프레임워크는 기존 대비 81.7%의 실행 시간을 단축하였으며, 멀티 카메라 이미지 기반 세그멘테이션 프레임워크는 경량 모델 사용에 따른 IoU 감소를 2.86% 이내로 유지하면서 프레임당 실행 시간을 34.3% 절감하였다.

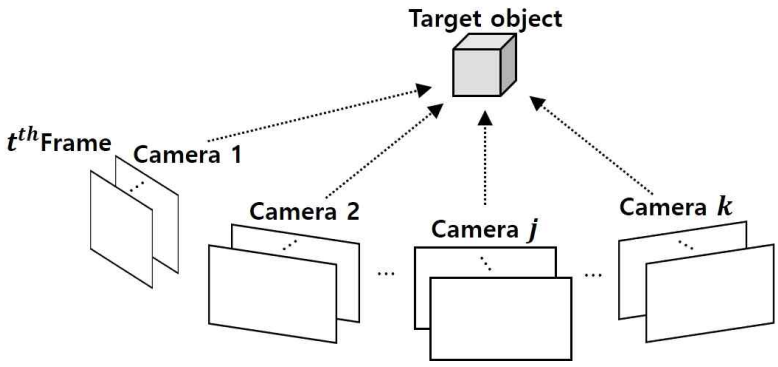
【주요어】 플렌옵틱, 스레드 풀, 멀티 스트림, 멀티뷰, low-rank 근사, 적응형 GPU 부하 재분배

목 차

표 목 차

그 림 목 차

제 1 장 서론



[그림 1-1] 멀티 카메라 이미지 예시

최근 영상 처리 및 컴퓨터 비전 분야에서는 단일 시점 영상이 가지는 한계를 극복하기 위한 다시점 영상을 활용한 연구가 진행되고 있다. 단일 카메라 기반 영상은 시야 제한과 객체의 가려짐(Occlusion)으로 인해 복잡한 장면에서의 객체의 정확한 인식 및 추적이 어렵다는 근본적인 제약을 지닌다. 이에 반해 다시점 영상은 동일한 장면을 복수의 시점에서 동시 관찰함으로써, 단일 시점 영상으로는 얻기 어려운 공간 및 깊이 정보를 얻을 수 있다. 이러한 특성을 통해 다시점 영상은 3차원 장면 이해를 위한 핵심 기술로 주목받고 있으며, 다양한 응용 분야의 기반이 되고 있다.

다시점 영상의 대표적인 예로는 플렌옵틱(Plenoptic) 이미지와 멀티 카메라(Multi-camera) 이미지가 있다. 첫 번째로 플렌옵틱 이미지는 복수의 카메라 또는 렌즈 어레이를 통해 서로 다른 초점을 갖는 2차원 이미지를 시퀀스 형태로 재구성한다(Oh, H., 2021; Son, W., et al, 2016; Pereira, F., et al, 2018). 이렇게 생성된 포컬 플레인 이미지(focal plane image)들은 포컬 스택(focal stack) 형태로 구성되며(Chalfoun, J., et al, 2024), 깊이 정보와 초점 정보를 동시에 포함한다. 이러한 특성은 일반적인 단일 시점 영상으로 얻기 어려운 공간적 표현을 가능하게 하며 최근에는 깊이 추정, 객체 추적 등 다양한 분야에서 사용되고 있다(Scagliola, A., et al, 2020).

두 번째로 멀티 카메라 이미지는 하나의 객체나 장면을 여러 대의 카메라로 서로 다른 시점에서 동시에 촬영한 이미지들의 집합을 의미한다(Vora, J., et al, 2023). [그림 1-1]은 동일한 객체를 서로 다른 각도에서 촬영한 멀티 카메라 이미지의 예시를 보여준다. 각 시점에서 획득된 영상은 단일 시점 영상으로는 얻을 수 없는 다각적인 시각 정보를 제공한다. 이러한 멀티 카메라 이미지 데이터는 깊이 정보와 결합될 경우(Park, J., et al, 2024; Son, H., et al, 2022), 3차원 재구성(3D reconstruction)(Dai, A., et al, 2018), 객체 인식, 추적, 증강현실(AR) 그리고 자율주행(Ouaknine, A., et al, 2021) 등 다양한 응용 분야에 활용될 수 있다. 이와 같이 플렌옵틱 및 멀티 카메라 이미지는 단일 영상이 지니는 시야 제한이나 부분 가림 문제를 완화함으로써 컴퓨터 비전 핵심 기술인 객체 추적 및 세그멘테이션의 성능을 향상시킬 수 있다.

객체 추적 기술은 일반적으로 VOT(Visual Object Tracking)(Javed, S., et al. 2022; Li, B., et al, 2019; Song, Z., et al, 2022)와 MOT(Multiple Object Tracking)(Manzoor, S., et al, 2022)로 구분된다. VOT는 연속된 비디오 프레임에서 첫 번째 프레임에 사용자가 지정한 객체를 별도의 검출 과정 없이 추적하는 방식이다. 반면, MOT는 사전에 정의된 클래스의 복수 객체를 각 프레임마다 검출하고, 프레임 간의 연관성을 분석하여 추적을 수행한다. 이러한 기술들은 교통 모니터링, 스포츠 경기 분석 등 다양한 실생활 응용에 활용되고 있다. 본 논문에서는 VOT 기반의 대표적인 추적기인 CSWinTT(Song, Z., et al, 2022)를 플렌옵틱 이미지 환경에 맞게 재구성하여, 사전에 정의되지 않은 객체를 사용자가 지정한 후 연속된 플렌옵틱 비디오 프레임에서 효율적으로 추적하는 것을 목표로 한다.

또한, 객체 세그멘테이션 분야는 VOS(Video Object Segmentation)(Caelles, S., et al. 2017)로 연속된 비디오 프레임에서 첫 프레임의 객체 위치를 기반으로 이후 프레임들에서 객체의 경계를 배경으로부터 정밀하게 분리하는 기술이다. 이는 VOT와 달리 픽셀 단위의 세그멘테이션 마스크를 예측하여 객체의 형상과 경계를 보다 정확하게 구분할 수 있다. 최근에는 시공간 정보를 효율적으로 저장하고 활용하는 STM(Space-Temporal

Memory)(Oh, S. W., et al, 2019; Wang, L., et al, 2017) 기반 모델이 높은 성능을 보이고 있으며, 특히 Transformer(Vaswani, A., et al, 2017) 기반의 HQTrack 프레임워크는 시공간적 특징 통합을 통해 VOS 성능을 크게 향상시켰다. HQTrack(Zhu, J., et al, 2023)은 VMOS(Video Multiple Object Segmentor)과 MR(Mask Refiner)로 구성되어 있으며, VMOS는 DeAOT(Yang, Z., et al, 2022) 구조를 기반으로 시공간 정보를 통합하고, MR은 HQSAM(Ke, L., et al, 2023)을 통해 최종 마스크의 품질을 향상시킨다.

그러나 이러한 VOT, VOS와 같은 기술은 대부분 단일 시점 비디오 환경을 전제로 설계되었기 때문에, 다시점 영상 환경에서는 여러 한계가 발생한다. 구체적으로는 각 시점의 영상 정보를 모두 처리해야 하므로, 구조적으로 높은 연산 복잡도와 데이터 처리량 증가 문제를 수반한다. 플렌옵틱 이미지의 경우, 한 프레임이 다수의 포컬 플레인 이미지를 포함하므로 객체 추적 시 연산량이 급격히 증가하며, 추적 대상 객체 수가 늘어날수록 계산량 또한 선형적으로 증가한다. 또한 멀티 카메라 이미지 환경에서는 한 프레임 내에서 k개의 카메라가 생성한 k개의 이미지를 모두 처리해야 하므로, 세그멘테이션 수행 시 연산 복잡도와 메모리 사용량이 k에 비례하여 증가한다. 특히 HQTrack 프레임워크의 구성 요소인 VMOS의 어텐션 연산은 시퀀스 길이 n에 대해 $O(n^2)$의 복잡도를 가지므로, 한 프레임에 다수의 이미지를 처리해야 하는 다시점 영상에서는 연산 및 메모리 비용이 급격히 증가한다.

따라서 본 논문은 플렌옵틱 이미지와 멀티 카메라 이미지 환경에서 발생하는 연산량 증가로 인한 전체 처리 지연 문제를 해결하기 위한 두 가지 프레임워크를 제안한다. 첫 번째로, 일반적인 2D 이미지 환경에서 널리 사용되는 VOT기반 추적기인 CSWinTT를 플렌옵틱 이미지 구조에 맞게 재구성하고 한 프레임 내 다수의 포컬 플레인 이미지 중 특정 포컬 플레인 이미지들의 정보만을 선별하는 포컬 플레인 이미지 선택 전략을 도입한 객체 추적 프레임워크를 설계하였다. 또한 프레임워크 내부의 일부 연산 과정들을 멀티코어로 구성된 CPU와 GPU 환경에서 병렬화하여 계산 효율을 극대화하고 높은 연산 복잡도로 인해 발생하는 데이터 처리 시간을 최소화하였다.

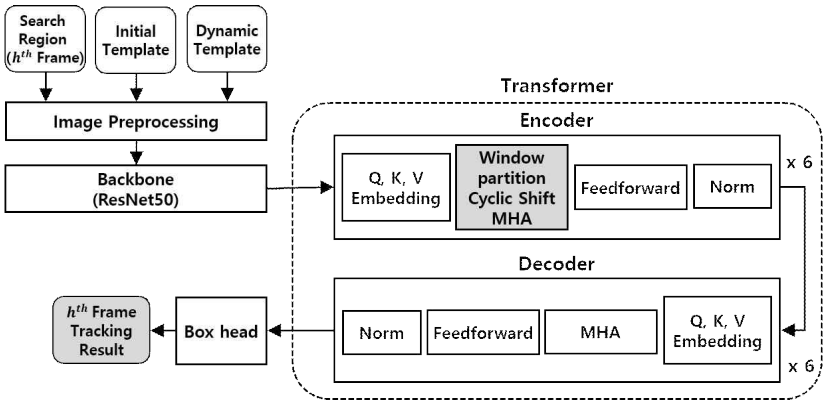
두 번째로, 멀티 카메라 이미지 환경에서 효율적인 객체 세그멘테이션을 수행하기 위해, 카메라 수 증가에 따른 연산 부담을 최소화할 수 있는 경량화된 모델을 동적으로 선택 및 적용할 수 있는 멀티 카메라 이미지 세그멘테이션 프레임워크를 설계하였다. 제안하는 세그멘테이션 프레임워크는 객체의 움직임 정도와 시각적 변화에 따라 연산량을 유연하게 조절하는 적응형 처리 전략을 적용한다. 또한, 다중 GPU 환경에서 원본 모델 조합과 경량화된 모델 조합 간 실행 시간 차이로 인해 발생하는 서로 다른 GPU 간 실행 시간 불균형을 완화하기 위해, 프레임 단위의 작업 부하 재분배 전략을 적용하였다. 이는 GPU 간 상대적인 실행 시간 비율과 연결성을 고려하여 트래커를 동적으로 재배치함으로써, 균형 잡힌 자원 활용을 보장하고, 프레임 단위의 지연을 최소화할 수 있다.

본 논문은 2장에서 연구에 필요한 배경지식과 내용과 앞서 간략히 설명하였던 다시점 영상 환경에서의 객체 추적 및 세그멘테이션 과정에서 발생하는 문제점을 설명한다. 3장에서는 제안하는 플렌옵틱 및 멀티 카메라 이미지 환경에서 프레임워크 구조와 구현 방법을 상세히 설명한다. 4장에서는 제안한 프레임워크에 대한 실험 및 분석 결과를 제시하며, 마지막으로 5장에서는 본 연구의 결론을 정리한다.

제 2 장 연구 배경 및 문제점

본 장에서는 대표적인 VOT 기반 객체 추적기인 CSWinTT의 구조와 객체 추적 방식, 그리고 VOS 기반 프레임워크인 HQTrack의 구조 및 세그멘테이션 방법을 설명한다. 이를 바탕으로 본 연구에서 해결해야 할 문제를 정의한다.

제 1 절 VOT 기반 객체 추적기의 구조 및 동작 방식



[그림 2-1] CSWinTT 실행 과정

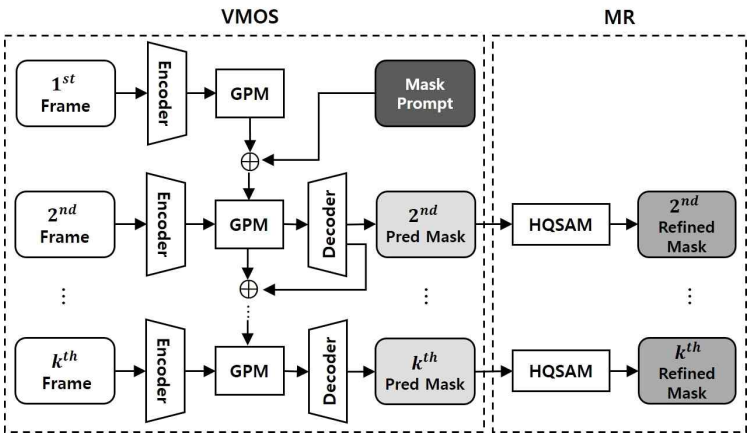
[그림 2-1]은 대표적인 VOT 기반 객체 추적기 중 하나인 CSWinTT가 일반적인 2D 이미지 환경에서 동작할 때, 임의의 h^{th}프레임에서 객체의 바운딩 박스 좌표를 추정하는 전체 과정을 나타낸 그림이다. CSWinTT는 초기 프레임에서 Template으로 주어지는 추적 대상 객체 정보와, 이를 탐색하기 위한 Search Region이 입력으로 구성된다. Template은 Initial Template과 Dynamic Template으로 구분된다. Initial Template은 첫 번째 프레임에서 사용자가 지정한 객체의 위치 정보를 기반으로 생성되며, 바운딩 박스는 상단 좌표 x, y와 그에 대응하는 너비(width) 및 높이(height)로 정의된다. Dynamic Template은 추적 과정에서 프레임 단위로 갱신되는 Template으로,

초기에는 Initial Template과 동일하게 설정된다. 이후 각 프레임마다 예측된 바운딩 박스의 Confidence Score를 계산하며, 이 값이 0.5 이상이면서 가장 높은 경우 해당 바운딩 박스를 새로운 Dynamic Template으로 갱신한다. Confidence Score는 추적된 영역 내에 객체가 존재할 확률을 나타내며, 0과 1 사이의 값을 갖는다. 값이 1에 가까울수록 해당 영역이 실제 객체를 포함할 가능성이 높음을 의미한다.

h^{th}프레임에서의 추론 과정은 먼저 Search Region의 전처리(Image Preprocessing) 단계로 시작된다. 이후 전처리가 완료된 Search Region은 초기 프레임에서 전처리된 Initial Template, 그리고 갱신 시점에 전처리가 수행된 Dynamic Template과 함께 ResNet50(He, K., et al, 2016)으로 구성된 백본 네트워크에 입력된다. 이러한 세 가지 입력은 백본 네트워크를 거쳐 피처맵(Feature Map)으로 추출된다. 추출된 피처맵은 이후 Transformer 모듈로 전달되며, 여섯 개의 레이어로 구성된 인코더(Encoder)와 디코더(Decoder)를 순차적으로 거친다. CSWinTT의 인코더는 입력 피처맵에 대해 임베딩을 수행한 뒤, Multi-Scale Window Partition과 Cyclic Shift를 적용한 멀티 헤드 어텐션(Multi-Head Attention, 이하 MHA)(Vaswani, A., et al, 2017)인 Window Partition Cyclic Shift MHA를 수행한다. 이는 CSWinTT의 핵심 과정으로, 각 헤드는 서로 다른 윈도우 크기를 사용하여 다중 스케일 어텐션을 계산한다. 또한, Cyclic Shift를 적용함으로써 고정된 윈도우 내에서뿐 아니라 다양한 위치적 관점에서 어텐션 계산을 가능하게 한다. 이렇게 생성된 어텐션 결과는 FFN(Feed-Forward Network)과 Normalization을 거쳐 디코더로 전달된다. 디코더는 Template과 Search Region 간 어텐션 관계를 파악하여, 최종적으로 Box Head로 전달할 특징 정보를 생성한다. Box Head는 다수의 Convolution 레이어로 구성되며, 이를 통해 최종적으로 h^{th}프레임에서의 객체 바운딩 박스 좌표를 산출한다.

제 2 절 VOS 기반 프레임워크의 구조 및 동작 방식

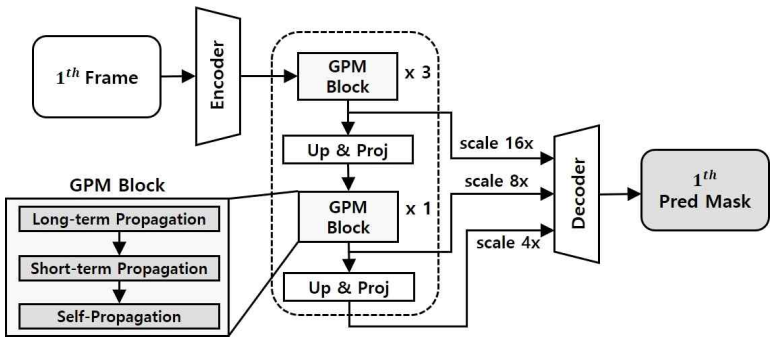
1) HQTrack



[그림 2-2] HQTrack 실행 과정

[그림 2-2]는 대표적인 VOS 프레임워크인 HQTrack의 동작 방식을 나타낸 것으로, 크게 VMOS와 MR로 구성된다. VMOS는 연속된 프레임에서 대상 객체를 세그멘테이션한다. 첫 번째 프레임에서는 이미지와 해당 마스크 프롬프트가 입력으로 주어지며, VMOS는 인코더를 통해 객체의 정보를 추출한다. 이를 DeAOT의 변형 구조인 GPM(Gated Propagation Module) 을 통해 장기 및 단기 메모리에 저장한다. 이후 두 번째 프레임부터 본격적인 세그멘테이션이 시작된다. 이미지가 입력되면 인코더는 이미지의 정보를 추출하고, GPM은 이전 프레임의 정보를 지속적으로 업테이트하며 다음 프레임으로 객체 정보를 전파한다. 이를 통해 긴 비디오에서도 일관적이고 정확한 세그멘테이션이 가능하다. 디코더는 이러한 정보를 기반으로 세그멘테이션 마스크를 생성한다. 이렇게 생성된 마스크가 VMOS의 출력으로 제공된다.

HQTrack의 MR은 HQSAM 모델을 사용한다. [그림 2-2]의 오른쪽에서 볼 수 있듯이, VMOS에서 생성된 세그멘테이션 마스크를 더욱 정교하게 개선한다. 구체적으로, VMOS에서 출력된 세그멘테이션 마스크의 바운딩 박스를 추출하고, 이를 HQSAM의 입력 프롬프트로 사용하여 더욱 정밀한 세그멘테이션 마스크를 생성한다. 이후 VMOS와 HQSAM의 출력 마스크 간 IoU(Intersection over Union)(Li, B., et al, 2019; Wu, Y., et al, 2013)를 비교한 뒤, 사용자가 사전에 설정한 임계값을 초과할 경우 HQSAM의 출력을, 그렇지 않으면 VMOS의 출력을 최종 마스크로 채택한다. 이러한 과정을



[그림 2-3] VMOS 구조

통해 MR은 VMOS에서 출력된 마스크를 정교하게 다듬으며, 본 연구에서는 HQSAM의 출력을 최종 마스크로 사용하였다.

2) VMOS

HQTrack의 VMOS는 InternImage-T(Wang, W., et al, 2023)를 인코더로 사용하며 기존 DeAOT를 확장한 VOS 모델이다. 멀티 스케일 Propagation 특징(Multi-Sacale Propagation features)을 통합하여 소형 객체 세그멘테이션 성능을 개선하였다(Li, J., et al, 2023). [그림 2-3]은 VMOS의 동작 과정을 나타낸다. VMOS는 4×, 8×, 16× 스케일의 Propagation 특징을 활용하며, 각 스케일은 linear projection이나 업샘플링(Upsampling)을 통해 생성된다. 구체적으로 8× 스케일의 특징은 16× 스케일의 특징을 기반으로 linear projection으로 변환되며, 4× 특징은 8×특징을 업샘플링하여 생성된다. Propagation 연산은 GPM 블록 내에서 수행되며, 16× 스케일은 3개의 GPM 블록, 8× 스케일은 1개의 GPM 블록으로 구성된다. 각 블록은 Long-term, Short-term, Self-Propagation 세 가지 방식으로 전파되며, 장단기 메모리를 활용해 객체의 외형 변화에 대응한다. 장기 메모리의 수는 최대 8개로 제한되며, 가장 오래된 메모리는 순차적으로 삭제된다. 이렇게 얻어진 Propagation 특징은 FPN(Lin, T. Y., et al, 2017)구조의 디코더를 거쳐 최종 마스크를 예측한다.

VMOS의 Propagation 연산은 Transformer 구조를 기반으로 하며, 하나의 세그멘테이션을 수행할 때 4개의 GPM 블록들이 각 세 가지의 Propagation

연산을 하므로 총 12회의 어텐션 연산이 반복적으로 실행된다. 이러한 구조는 객체 외형 변화에 대응할 수 있는 강력한 표현력을 제공하지만, 동시에 어텐션 메커니즘의 한계도 드러낸다. 특히 Self-Attention은 모든 토큰 쌍의 상호작용을 계산해야 하므로 시퀀스 길이 n에 대해 $O(n^2)$의 시간, 공간 복잡도를 가진다. 따라서 긴 시퀀스나 고해상도 입력을 처리할 때 연산 비용이 급격히 증가한다. 이는 연속된 프레임과 고해상도 영상을 다루는 비디오 세그멘테이션 작업에서 심각한 비효율성으로 이어질 수 있다.

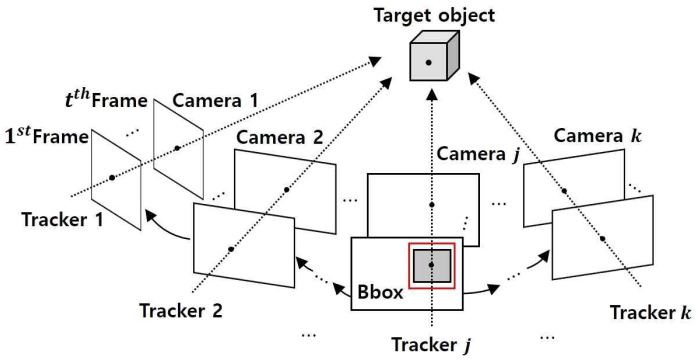
3) HQSAM

복잡한 장면에서는 VMOS가 생성하는 초기 세그멘테이션 마스크의 품질이 충분하지 않을 수 있다. 이를 보완하기 위해, 본 연구에서는 [그림 2-2]에 제시된 바와 같이 MR 단계에서 HQSAM을 활용하였다. 대규모 데이터셋으로 사전 학습된 HQSAM을 적용함으로써 초기 마스크를 정교하게 보정할 수 있다. HQSAM은 ViT-H(Dosovitskiy, A., et al, 2021) 기반의 이미지 인코더를 사용하며, 강력한 표현력을 바탕으로 매우 정확하고 안정적인 마스크 보정을 가능하게 한다. 그러나 이러한 높은 성능은 상당한 연산 부담을 수반하며, 이는 처리 속도를 저하시킬 수 있고 실시간 응용에서 제약으로 작용할 수 있다.

제 3 절 다시점 영상에서 객체 추적 및 세그멘테이션 문제점

다시점 영상 환경에서는 단일 시점 영상 처리와 비교해 데이터의 양과 복잡도가 기하급수적으로 증가한다. 하나의 프레임 내에 여러 이미지가 존재하므로, 한 프레임 내의 모든 이미지에 대해 순차적으로 딥러닝 기반 추적 및 세그멘테이션 모델을 수행할 경우, 연산량 급증으로 인한 처리 지연(latency)과 GPU 메모리 자원 부족 문제가 나타난다. 이러한 특성은 실시간 분석이 요구되는 응용 환경에서 심각한 성능 저하를 초래할 수 있다.

플렌옵틱 이미지의 경우, 하나의 프레임이 여러 장의 포컬 플레인으로 구성된다는 점에서 기존 2D 영상과 본질적으로 다르다. 본 연구에서 사용하는



[그림 2-4] 멀티 카메라 이미지 기반 세그멘테이션 과정

플렌옵틱 이미지는 한 프레임에 총 101장의 포컬 플레인으로 이루어져 있으며, 각 이미지는 서로 다른 초점 위치에서 재구성된다. 따라서 2D 기반 객체 추적기를 그대로 적용할 경우, 단일 프레임 내에서도 101회의 독립적인 추적 과정이 수행되어야 한다. 이러한 구조적 특성으로 인해, CSWinTT와 같은 대형 Transformer 기반 모델은 높은 GPU 메모리 사용량과 긴 추론 시간이 필요하며, 모든 포컬 플레인에 대해 순차적으로 추적을 수행할 경우 지연시간을 피하기 어렵다. 더 나아가, 객체의 수가 증가할수록 추적 연산량이 선형적으로 증가하므로 실시간 처리가 사실상 불가능하다. 이는 실시간 비디오 분석 응용에서 심각한 성능 저하로 직결된다.

멀티 카메라 이미지 환경에서도 유사한 문제가 발생한다. 본 논문에서는 기존의 단일 카메라에서 촬영된 이미지 기반 VOS 프레임워크인 HQTrack을 k개의 카메라에서 촬영된 멀티 카메라 이미지 환경에 맞게 재구성하여 사용한다(Yong, J., et al, 2025). 사용자는 전체 k개의 카메라 중 기준이 되는 카메라 j를 선택하고, 해당 카메라의 첫 프레임에서 객체의 바운딩 박스를 지정한다. HQTrack은 이 바운딩 박스를 프롬프트로 입력받아 각 카메라에 대응되는 마스크 프롬프트를 생성하며, 이를 기반으로 멀티 카메라 이미지 환경에서 각 카메라에 대해 독립적인 세그멘테이션이 수행된다. [그림 2-4]는 다양한 위치의 카메라들이 동일 객체를 촬영한 멀티 카메라 이미지에서의 세그멘테이션 과정을 시각적으로 보여준다. 한 프레임은 k개의 이미지로 구성되며, 이에 따라 연산량과 데이터 처리량은 카메라 수에 비례해 증가한다. 이는 기

존 단일 카메라 기반 환경과 비교해 연산 및 데이터 처리 부하가 급격히 증가함을 의미하며, 결과적으로 전체 시스템의 실행 속도 저하를 초래할 수 있다.

세그멘테이션 시, 각 카메라는 서로 다른 위치에서 동일한 객체를 촬영하므로, 카메라별로 독립적으로 세그멘테이션을 수행해야 한다. 이에 따라 k개의 카메라에 대해 GPM을 통한 개별적인 장기 및 단기 메모리 업데이트가 이루어지기 위해 k개의 트래커를 생성하여 각 카메라에서 독립적으로 객체 세그멘테이션을 수행한다. 그러나 이러한 구조에서는 시각적 변화가 거의 없는 프레임에서도 각 트래커가 동일한 수준의 연산을 반복적으로 수행한다. 이로 인해 실제 정보 변화량에 비해 과도한 연산이 이루어지는 비효율적인 상황이 초래된다.

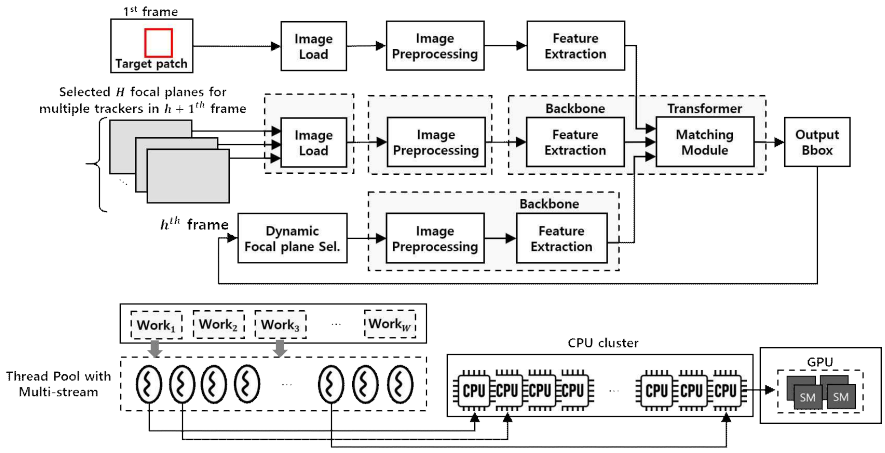
이러한 문제점을 해결하기 위하여, 본 논문에서는 플렌옵틱 및 멀티 카메라 이미지 환경에서 발생하는 과도한 연산량과 처리 지연 문제를 동시에 완화할 수 있는 새로운 객체 추적 및 세그멘테이션 프레임워크를 제안한다.

제 3 장 다시점 영상에서 객체 추적 및 세그멘테이션 프레임워크

본 장에서는 플렌옵틱 이미지 및 멀티 카메라 이미지 환경에서, 객체 추적과 세그멘테이션의 프레임 단위 처리 효율을 향상시키기 위한 새로운 프레임워크를 제안한다. 제안된 방법은 계산 복잡도를 줄이면서도 정확도 손실을 최소화하여 실시간 처리 성능을 개선하는 것을 목표로 한다.

제 1 절 플렌옵틱 이미지에서의 객체 추적 프레임워크

1) 제안한 객체 추적 프레임워크 구조



[그림 3-1] 제안하는 객체 추적 프레임워크의 구조와 전체 동작 과정

[그림 3-1]은 제안한 객체 추적 프레임워크의 전체 구조를 나타낸다. 본 프레임워크는 각 실행 단계에서 발생하는 작업들을 효율적으로 처리하기 위해 설계되었다. 각 작업은 현재 프레임에서의 포컬 플레인 처리 단위를 의미하며, 전체 파이프라인은 크게 이미지 로드(Image Load), 이미지 전처리(Image Preprocessing), 특징 추출(Feature Extraction), 매칭 모듈(Matching

Module), 그리고 결과 출력(Output Bbox)으로 구성된다. 이러한 모듈들은 기존 CSWinTT의 일반적인 처리 흐름을 유지하면서, 스레드 풀(thread pool)과 멀티 스트림(multi-stream)(Segura, A., et al, 2019) 구조를 도입하여 각 실행 단계의 작업을 병렬로 수행한다. 스레드 풀은 다수의 워커 스레드(worker thread)를 사전에 할당해 두고 요청 시 이를 재사용하는 방식으로, 스레드 생성과 소멸에 따르는 시스템 오버헤드를 효과적으로 배제한다. 나아가 각 워커 스레드를 독립적인 GPU 스트림에 매핑함으로써, 다수의 포컬 플레인에 대한 연산을 지연 없이 동시에 수행할 수 있는 환경을 구축하였다(Kim, M., et al, 2023; Plancher, B., et al, 2018). 이러한 구조는 GPU 자원의 활용 효율을 극대화하고, 전체 추적 지연 시간을 효과적으로 단축시킨다.

또한, 제안한 프레임워크는 동적 포컬 플레인 선택기(Dynamic Focal Plane Selector)를 포함한다(Yong, Y., et al, 2024). 이는 매 프레임마다 전체 포컬 플레인을 모두 탐색하는 대신, 이전 프레임의 추적 결과를 기반으로 현재 프레임에서 필요한 포컬 플레인 이미지 범위를 결정하여 처리한다. 이러한 동적 선택 메커니즘은 불필요한 연산을 줄이고, 실시간 추적 과정에서의 처리 효율과 속도를 향상시킨다.

2) 제안한 객체 추적 프레임워크의 동작 방법

플렌옵틱 이미지는 단일 객체라도 하나의 프레임 내에 다수의 포컬 플레인을 포함하는 구조적 특성을 가지므로, 본 연구에서는 포컬 플레인 단위로 연산을 병렬 수행할 수 있도록 전체 처리 파이프라인을 멀티스레드 기반으로 설계하였다. [그림 3-1]에서 점선으로 둘러싸인 회색 박스는 이러한 멀티스레딩 방식의 병렬 처리 블록을 나타내며, 이미지 로드, 이미지 전처리, 특징 추출, 매칭 모듈로 구성된 주요 연산 단계를 포함한다. 이미지 로드와 이미지 전처리는 계산 집약적인 딥러닝 모델을 사용하지 않는 CPU 기반 처리 단계이기 때문에, 각 포컬 플레인마다 서로 다른 CPU 스레드에 동일한 작업을 할당함으로써 병렬 처리를 극대화하였다. 이를 통해 초기 단계에서 발생하는 지연을 최소화하였다.

반면, 특징 추출과 매칭 모듈 단계는 ResNet50, Transformer와 같은 딥러닝 모델을 사용하는 고비용 연산으로 구성되어 있으며, GPU에서 수행되는 것이 효율적이다. 본 연구에서는 GPU 가속의 효율을 극대화하기 위해 CPU(호스트)가 GPU(디바이스)에 커널(kernel) 형태로 딥러닝 워크로드를 전달하는 과정 또한 멀티스레딩 방식으로 구성하였다. 각 스레드는 고유한 스트림에 매핑되어 비동기적으로 커널을 전달하며, 이를 통해 GPU 내부에서는 스트림 간 병렬 처리가 가능해진다. 이러한 설계는 GPU 내에서 딥러닝 연산이 최대한 중첩 및 병렬화되도록 하여 전체 계산 효율을 크게 향상시킨다. 이러한 설계를 바탕으로 제안하는 프레임워크는 아래와 같이 동작한다.

먼저, 첫 번째 프레임 처리 단계에서는 플렌옵틱 영상의 기준 카메라 이미지로부터 타겟 패치를 생성한다. 이는 사용자가 [그림 3-1]과 같이 첫 번째 프레임 내에서 추적 대상 객체를 포함하는 영역을 직접 지정하는 과정이며, 해당 영역이 타겟 패치로 정의된다. 이후 두 번째 프레임부터는 본격적인 객체 추적 과정이 수행된다. 이 단계에서 메인 스레드는 두 번째 프레임에 포함된 모든 포컬 플레인을 스토리지로부터 메모리로 로드하는 작업을 워커 스레드에 할당한다. 각 워커 스레드는 할당된 포컬 플레인 이미지를 스토리지에서 메모리로 로드하는 작업을 병렬적으로 수행한다. 메인 스레드는 각 워커 스레드가 작업을 완료할 때까지 대기하며 모든 스레드의 작업 종료를 확인한 후 다음 단계를 수행한다.

이후 이미지 전처리 단계에서 메인 스레드는 메모리에 적재된 포컬 플레인들을 워커 스레드에 분산하여 전처리 작업을 병렬로 수행한다. 각 워커 스레드의 전처리 결과는 후속인 특징 추출의 입력으로 사용된다. 특징 추출 단계에서도 마찬가지로 메인 스레드가 타겟 패치와 각 포컬 플레인 이미지에 대한 피처맵을 추출하는 작업을 워커 스레드에 할당한다. 모든 워커 스레드의 연산이 완료되면 추출된 피처맵들은 다음 매칭 모듈의 입력으로 전달된다. 매칭 모듈 단계에서는 Transformer 연산을 각 워커 스레드에 할당하여 병렬적으로 수행함으로써, 타겟 패치와 각 포컬 플레인 간의 Confidence Score를 산출한다.

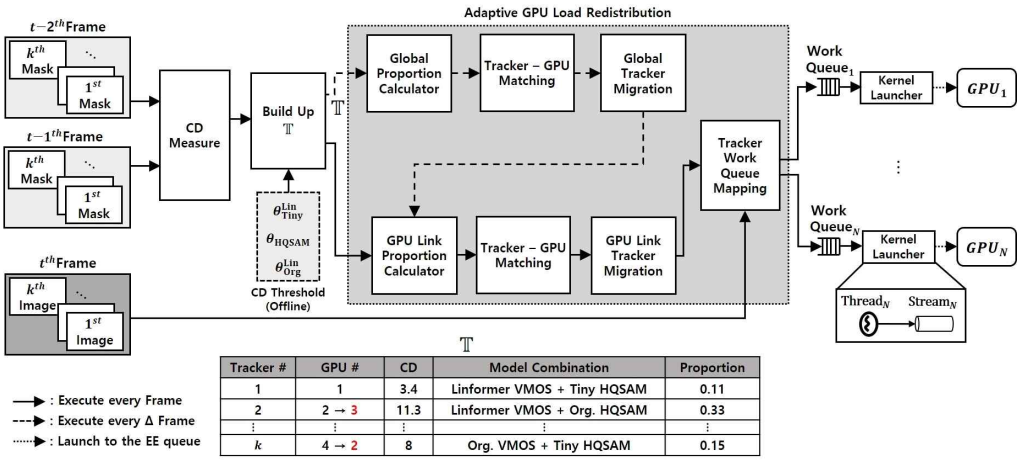
최종 출력 단계에서는 매칭 모듈로부터 계산된 유사도 점수를 기반으로

객체의 위치를 나타내는 바운딩 박스와 Confidence Score를 생성한다. 결과적으로, 제안한 프레임워크는 플렌옵틱 이미지 구조를 고려하여 이미지 로드, 이미지 전처리, 특징 추출, 매칭 모듈, 출력의 각 단계를 멀티스레드 기반 병렬 구조로 구현함으로써, 객체 추적의 처리 효율을 극대화하였다.

또한, 본 논문에서는 매 프레임마다 전체 포컬 플레인들을 사용하여 객체 추적을 수행할 경우 발생하는 과도한 연산량을 줄이기 위해, 동적 포컬 플레인 선택 전략을 적용하였다. 두 번째 프레임에서는 전체 101장의 포컬 플레인을 모두 활용하여 객체 추적이 이루어지며 이후 프레임부터 동적 포컬 플레인 선택 전략이 수행된다. 각 프레임에서 메인 스레드는 모든 포컬 플레인에 대해 Confidence Score를 계산하고, 이 중 가장 큰 값을 갖는 포컬 플레인 인덱스 l을 기준 포컬 플레인 인덱스로 결정한다. 이후 다음 프레임의 객체 추적 과정에서는 전체 포컬 플레인을 처리하는 대신, 기준 포컬 플레인 인덱스 l을 기준으로 특정 국소 범위 내의 포컬 플레인을 선택하여 추적에 활용한다.

선택되는 포컬 플레인 범위인 포컬 레인지(focal range) r은 이전 프레임에서 산출된 Confidence Score를 기반으로 동적으로 결정된다. 이전 프레임에서 Confidence Score가 0.5를 초과하는 경우 안정적인 추적 상태로 판단하여 좁은 범위인 $r = 3$을 적용한다. 반면, Confidence Score가 0.5 이하일 경우 추적 불확실성이 증가한 것으로 간주하여, 더 넓은 탐색 범위인 $r = 5$를 적용한다. 이와 같이 결정된 포컬 레인지 r에 따라, 이전 프레임에서 가장 높은 Confidence Score를 기록한 기준 포컬 플레인 인덱스 l을 중심으로 $l-r$부터 $l+r$까지 포컬 플레인 인덱스를 저장한다(Yong, Y., et al, 2024). 이러한 과정을 통해 이후 프레임은 해당 범위의 포컬 플레인을 탐색하며 객체 추적을 수행한다. 이러한 동적 포컬 플레인 선택기는 전체 포컬 플레인 처리로 인한 연산량 증가를 효과적으로 감소하는 동시에, 추적에 실질적으로 기여하는 포컬 플레인만을 활용함으로써 안정적이고 효율적인 객체 추적을 가능하게 한다.

제 2 절 멀티 카메라 이미지에서의 세그멘테이션 프레임워크



[그림 3-2] 제안한 멀티 카메라 이미지 세그멘테이션 프레임워크의 동작 과정

1) 제안한 멀티 카메라 이미지 세그멘테이션 프레임워크 구조

제안된 프레임워크는 멀티 카메라 이미지 환경에서의 효율적인 세그멘테이션을 위해 설계되었으며, 주요 절차는 다음과 같은 세 단계로 이루어진다: (1) CD Measure, (2) Build Up 𝕋, (3) Adaptive GPU Load Redistribution

[그림 3-2]는 제안된 세그멘테이션 시스템의 전체적인 동작 과정을 나타낸다. 각 과정을 통해 멀티 카메라 이미지 환경에서 발생하는 연산량 증가 문제를 해결하고, GPU 간 실행 시간을 균형 있게 조정하여 프레임 처리 속도를 향상시킨다. 매 프레임에서 GPU Link(NVLink[1], NVLink Switch[2])를 통한 메모리 복사를 통해 조정이 이루어지며, 일정 주기(△ 주기)마다 시스템 메모리를 거치는 메모리 복사를 수행하여 전체 GPU 실행 시간의 편차를 줄인다. [그림 3-2]의 실선은 매 프레임마다 이루어지는 GPU Link를 통한 메모리 복사 과정을 나타내며 점선은 일정 주기(△ 주기)마다 수행되는 시스템 메모리를 통한 메모리 복사 과정을 나타낸다. 이와 같은 절차를 통해 전체 세그멘테이션 효율이 향상되며, 본 프레임워크는 다양한 GPU 구성에서도 적용할 수 있도록 확장성과 유연성을 고려하여 설계되었다.

---

1) NVIDIA, Nvlink high-speed gpu interconnect, online: https://www.nvidia.com/en-us/products/workstations/nvlink-bridges/

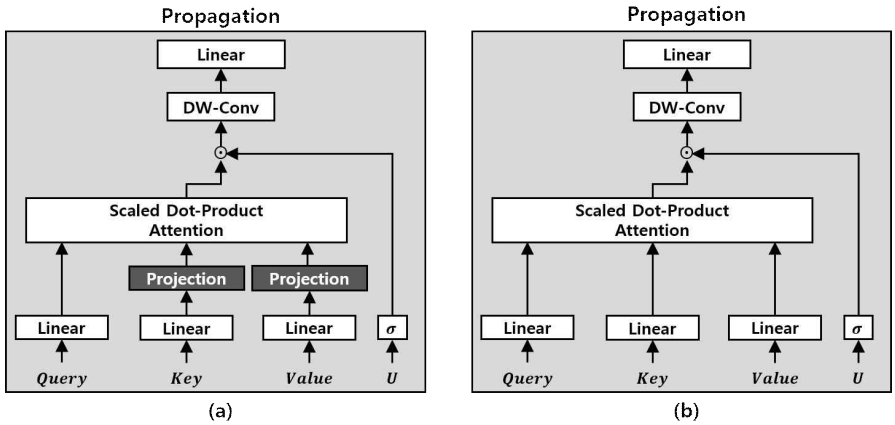
2) NVIDIA, Nvidia nvlink and nvlink switch, online: https://www.nvidia.com/en-us/data-center/nvlink/

CD Measure는 $t-2^{th}$프레임과 $t-1^{th}$프레임에서 각 세그멘테이션 마스크 간 객체의 CD를 계산한다. Build up $\mathbb{T}$는 사전에 정의된 임계값 θ_{Tiny}^{Lin}, θ_{HQSAM}, θ_{Org}^{Lin}과 계산된 CD 값을 기준으로, 각 트래커가 t^{th}프레임에서 사용할 모델 조합을 결정한다. 여기서 모델 조합이란 HQTrack의 VMOS와 MR의 HQSAM 각각에 대해 원본 버전과 경량화 버전 중 하나를 선택하는 것을 의미한다. 이렇게 결정된 모델 조합을 바탕으로 $\mathbb{T}$를 구성한다.

적응형 GPU 부하 재분배 기법은 매 프레임마다 GPU Link Proportion Calculator와 Tracker-GPU Matching을 통해 GPU Link로 연결된 GPU 간 실행 시간 편차를 줄인다. 이 과정에서 모델 조합별 상대적 실행 시간을 나타내는 Proportion 값을 활용하여, 각 트래커에 대해 t^{th}프레임에서 실행될 GPU가 재할당된다. 경량화 모델(Linformer VMOS + Tiny HQSAM)은 가장 작은 Proportion 값을, 원본 모델(Org. VMOS + Org. HQSAM)은 가장 큰 Proportion 값을 갖는다. 이후 GPU Link Tracker Migration 단계에서는 재할당 대상 트래커에 대해 세그멘테이션에 필요한 정보들을 GPU Link를 통해 메모리 복사하여, 해당 트래커가 세그멘테이션을 수행할 GPU의 위치가 재할당된다. 이러한 과정을 통해 GPU Link로 연결된 GPU 간 실행 시간 편차를 줄인다.

또한, 일정 주기마다 Global Proportion Calculator와 Tracker-GPU Matching을 수행해 GPU Link로 연결되지 않은 GPU 간 실행 시간 편차를 줄인다. 이어지는 Global Tracker Migration 단계에서는 시스템 메모리를 거치는 복사가 이루어진다. 이러한 메모리 복사를 통해 재할당 대상의 트래커가 세그멘테이션을 수행할 GPU의 위치가 조정된다. 이후 다시 GPU Link를 통해 메모리 복사함으로써 GPU Link로 연결된 GPU간 실행 시간 편차를 줄이는 과정을 이어서 수행한다. 이를 통해 전체 시스템 GPU의 실행 시간 편차를 줄일 수 있다.

마지막으로, 모든 메모리 복사가 끝나면 Tracker Work Queue Mapping 단계에서는 각 트래커와 t^{th}프레임의 입력 이미지를 재할당된 GPU의 작업 큐에 삽입하여 세그멘테이션을 수행한다. 이를 통해 GPU 간 실행 시간 편차를 줄이고, 멀티 카메라 세그멘테이션의 프레임당 실행 시간을 효과적으로 단축



[그림 3-3] GPM Block의 어텐션 기반 Propagation 연산 구조 (a) 기본 구조 (b) Linformer 기반 low-rank projection이 적용된 구조

할 수 있다.

2) HQTrack 내 모델 경량화

멀티 카메라 이미지 세그멘테이션 프레임워크는 HQTrack의 주요 구성 요소인 VMOS와 HQSAM을 경량화한 모델을 포함한다. VMOS에는 Linformer 기반 low-rank projection을 적용하여 경량화를 적용하였고, HQSAM에는 Tiny HQSAM을 적용하였다. 또한, 객체의 움직임을 분석하여 원본 모델과 경량화 모델 간 조합을 프레임 단위로 동적으로 선택하는 적응형 동적 모델 조합 선택 기법을 도입하였다. 더 나아가, 경량화 수준을 보다 정밀하게 조절하기 위해 프레임 간 코사인 유사도(Schütze, H., et al. 2008; Wojke, N., et al, 2017)를 활용하여 Linformer의 low-rank projection 차원을 조정하여 어텐션 연산의 근사 정도를 제어한다. 코사인 유사도가 높을 때는 투영 차원을 낮추어 근사화를 강화하고 실행 시간을 효과적으로 단축한다. 반대로 코사인 유사도가 낮을 경우, 높은 차원을 유지하여 세그멘테이션 정확도를 보존하면서도 실행 시간을 줄일 수 있다.

가) VMOS에 Linformer기반 low-rank projection matrix 적용

	Encoder	GPM	Decoder
Exe. Time(Sec.)	0.022	0.2	0.001
Ratio	9.73%	89.65%	0.62%

[표 3-1] VMOS 부분별 실행 시간 비율 분석

VMOS의 각 GPM 블록은 기존의 multi-head attention 대신, single-head attention을 사용하여 연산 복잡도를 줄이면서도 성능을 유지함으로써 효율적인 어텐션 기반 Propagation 구조를 사용한다. [그림 3-3]의 (a)는 이러한 GPM의 Propagation 연산 구조를 나타낸다. Propagation 연산에는 인코더로 출력된 이미지의 시각 정보와 게이팅 임베딩(gating embedding)이 입력된다. 이 임베딩은 비선형 게이트 함수 σ를 거쳐 처리되며, scaled dot-product attention의 결과에 element-wise 곱셈으로 적용되어 전파 강도를 조절한다. 어텐션 결과에는 depth-wise 2D convolution이 적용되어 효율적인 특징 추출이 가능하며, 마지막으로 linear projection을 통해 출력 특징으로 변환된다. 이러한 구조는 복잡도를 제어하면서도 정밀한 객체 세그멘테이션 성능을 유지하는 데 효과적이다.

[표 3-1]은 VMOS 구성 요소별 실행 시간 분석 결과를 제시한다. 전체 실행 시간의 약 89%가 GPM 단계에서 소요되며, 이 중 대부분은 어텐션 연산에 기인한다. 이는 총 12회의 Propagation 과정에서 반복적으로 어텐션 연산이 수행되기 때문에, 전체 실행 시간 증가의 주요 원인 중 하나로 분석된다. 따라서 GPM 내 어텐션 연산의 계산 효율을 향상시키는 것이 VMOS의 전반적인 성능 최적화를 위한 핵심 과제로 분석된다.

이러한 연산을 효율적으로 처리하기 위한 대표적인 기법 중 하나는 Linformer(Wang, S., et al, 2020)로, self-attention이 생성하는 attention matrix가 본질적으로 low-rank 구조를 가진다는 이론적, 실험적 관찰에 기반하여 설계된 경량화 모델이다. 이 구조는 query는 그대로 유지하되, key와 value 행렬을 저차원 공간으로 선형 투영되어 어텐션 연산을 low-rank 행렬로 근사한다. 이로써 기존의 어텐션 연산에서 $O(n^2)$의 복잡도를 $O(n)$으로 줄이면서도 성능을 효과적으로 유지할 수 있다(Wang, S., et al, 2020). 또한,

Linformer는 attention map의 표현력을 크게 해치지 않으면서도 메모리 사용량과 계산 비용을 효과적으로 감소시킬 수 있는 장점이 있다.

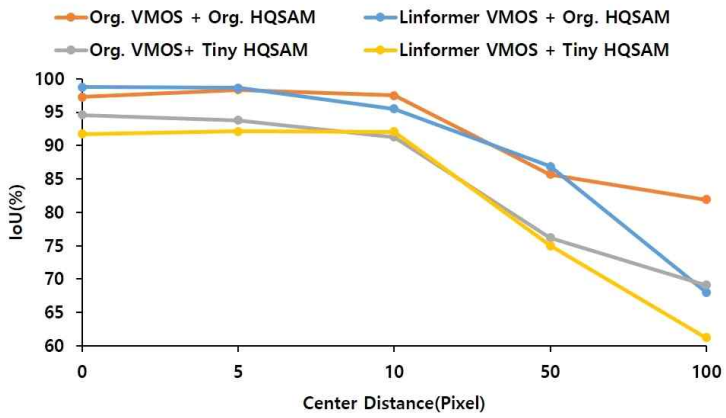
[그림 3-3]의 (b)는 Linformer 기반 low-rank projection이 적용된 Propagation 연산 과정을 나타낸다. 구현에 있어, Linformer 기반의 low-rank projection이 GPM 내 12개의 어텐션 연산에 적용된다. 프로젝션 행렬은 학습 가능한 파라미터로 초기화되며, 각 GPM 블록마다 별도로 정의되어 각 GPM 블록 간에 projection matrix를 공유하지 않고 low-rank 근사가 수행된다. 동일 블록 내에서도 key와 value에 대해 별도의 projection matrix를 적용함으로써 Propagation 방식별 구조적 특성을 보존할 수 있도록 설계하였다.

나) 효율적인 Mask Refiner를 위한 Tiny HQSAM 적용

HQSAM은 다양한 크기의 모델로 구성되어 있으며, 사용되는 이미지 인코더에 따라 구분된다. Heavy HQSAM은 ViT-H 기반의 이미지 인코더를 사용하며, Large HQSAM과 Base HQSAM은 각각 ViT-L 및 ViT-B와 같은 비교적 경량화된 인코더로 대체하여 재학습시킨 모델이다. Tiny HQSAM은 ViT-H 기반의 인코더로부터 지식 증류(knowledge distillation)(Hinton, G., et al, 2015)를 통해 경량 인코더를 학습하고, 이후 해당 인코더와의 호환성을 높이기 위해 마스크 디코더를 미세 조정(fine-tuning)한 구조를 갖는다.
기존의 HQTrack에서는 Heavy HQSAM을 사용하였으며 제안하는 멀티 카메라 이미지 세그멘테이션 프레임워크에서는 경량화된 Tiny HQSAM을 MR로 사용한다. Tiny HQSAM은 평균 IoU가 91% 이상으로 유지되어 높은 세그멘테이션 정확도를 안정적으로 보장하며, 프레임당 평균 실행 시간은 Heavy HQSAM 대비 약 73.46% 단축되어 실시간 처리 측면에서도 우수한 성능을 나타낸다. 보다 상세한 성능 비교 결과는 4장 2절 2)의 나)에 제시되어 있다.

3) 적응형 동적 모델 조합 선택 기법

앞서 VMOS의 GPM 블록 내 Propagation의 어텐션 연산에 Linformer



[그림 3-4] CD에 따른 네 가지 모델 조합의 IoU 비교

기반의 low-rank projection을 적용하고, HQSAM을 경량화한 Tiny HQSAM을 설계하여 효율적인 세그멘테이션을 수행할 수 있는 경량 모델을 구성하였다. 본 연구에서는 경량 모델과 원본 모델을 조합하여, 멀티 카메라 이미지 환경에서 매 프레임마다 최적의 모델을 동적으로 선택하는 적응형 동적 모델 조합 선택 기법을 제안한다. 모델 선택의 기준은 객체의 움직임 크기이며, 본 연구에서는 이를 프레임 간 객체 중심 좌표의 거리(Center Distance, 이하 CD)로 정의한다. 일반적으로 CD는 $t-1^{th}$프레임과 t^{th}프레임 간의 객체의 중심 거리를 계산한다. 그러나 본 논문에서는 $t-2^{th}$와 $t-1^{th}$프레임에서 세그멘테이션 마스크 간 중심 거리를 사용하여 객체의 움직임 정도를 파악한다.

제안된 기법은 다음 네 가지 모델 조합을 활용한다: Org. HQSAM은 기존 HQTrack에서 사용하는 Heavy HQSAM을 나타낸다. (1) Org. VMOS + Org. HQSAM, (2) Linformer VMOS + Org. HQSAM, (3) Org. VMOS + Tiny HQSAM, (4) Linformer VMOS + Tiny HQSAM.

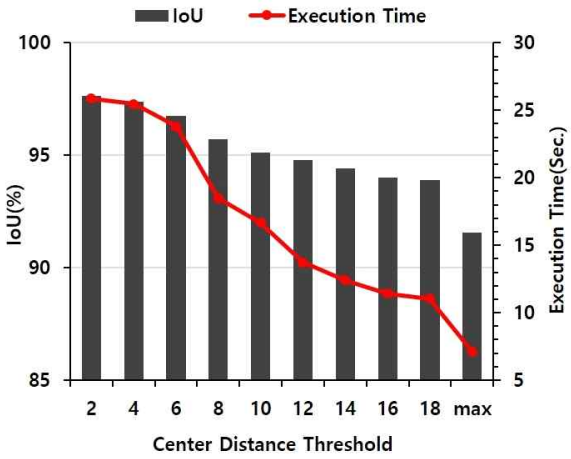
[그림 3-4]는 각 네 가지 모델이 75프레임에 대해 세그멘테이션을 수행하였을 때 CD에 따라 평균 IoU의 변화를 나타낸다. 실험 결과, 대부분의 모델 조합에서 CD 값이 10 이상인 이미지들의 IoU는 상대적으로 낮은 값을 가지며, 특히 Linformer VMOS + Tiny HQSAM 조합에서 성능 저하가 가장 두드러졌다. 반면, Org. VMOS + Org. HQSAM 조합은 대부분의 CD 값을

가진 이미지들의 IoU가 안정적인 성능을 유지했다. 이러한 결과는 객체 움직임이 큰 프레임에서 원본 모델 조합이 안정적인 세그멘테이션 성능을 제공할 수 있음을 나타낸다.

이를 바탕으로, 본 연구는 매 프레임마다 CD 값에 따라 모델 조합을 동적으로 선택하는 적응형 동적 모델 조합 선택 기법을 제안한다. 네 가지 사전 정의된 조합 중 하나를 선택해 세그멘테이션 정확도를 유지하면서 세그멘테이션 실행 시간을 단축하는 것이 목표다. 구체적으로, 각 트래커는 t^{th}프레임에서 세그멘테이션을 수행하기 전 $t-2^{th}$와 $t-1^{th}$프레임에서 세그멘테이션 마스크의 CD를 계산한다. 해당 CD 값이 설정된 임계값 이상일 경우, 객체의 움직임이 큰 구간으로 판단하여 Org. VMOS + Org. HQSAM 조합을 적용하여 세그멘테이션 정확도를 우선시한다. 반대로 CD 값이 임계값 미만이면, 움직임이 작아 경량 모델로도 안정적인 성능이 가능한 상황으로 판단하고, Linformer VMOS + Tiny HQSAM 등의 경량 모델 조합을 적용하여 성능 저하를 최소화하며 처리 속도를 높인다. 다중 GPU 환경에서 발생할 수 있는 메모리 복사 및 실행 속도 편차 등의 유동성을 배제하기 위하여, CD 임계값을 설정하기 위한 과정은 모두 단일 GPU에서 실행되었으며 자세한 설명은 뒤따르는 절에서 한다.

가) Stage 1: Org. HQSAM과 Tiny HQSAM 선택을 위한 임계값 θ_{HQSAM} 설정 방법

매 프레임마다 각 카메라가 세그멘테이션에 사용할 모델 조합을 적절히 선택하기 위해, 본 연구는 CD 임계값을 두 단계로 설정하였다. 1단계에서는 Org. HQSAM과 Tiny HQSAM 선택을 위한 임계값 θ_{HQSAM}을 정의한다. 구체적으로, CD가 θ_{HQSAM}이하면 상대적으로 실행 속도가 빠른 Tiny HQSAM을, 임계값을 초과하는 프레임에는 보다 높은 정확도를 제공하는 Org. HQSAM을 적용한다. θ_{HQSAM}을 높게 설정할수록 더 많은 프레임에서 경량화 모델이 적용되어 전체 시스템의 처리 속도가 향상될 수 있다. 그러나 경량화 모델의 핵심 목적은 실행 시간의 단축뿐만 아니라 성능 저하를 최소화하는



[그림 3-5] CD 임계값 변화에 따른 Tiny/Org. HQSAM 선택 시 IoU 및 실행 시간 비교

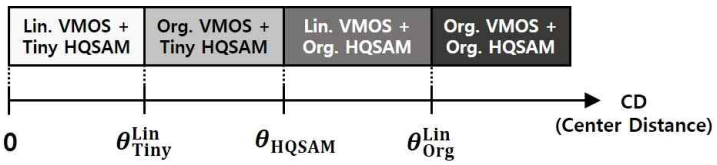
데 있으므로, 임계값 설정 시 처리 속도 개선과 함께 세그멘테이션 성능의 유지 또한 반드시 고려되어야 한다.

[그림 3-5]는 θ_{HQSAM}값을 변화시켰을 때 IoU와 프레임당 평균 실행 시간의 변화를 나타낸다. 여기서 x축은 θ_{HQSAM}값을 나타내며 max는 모든 프레임이 Tiny HQSAM을 실행하도록 한다. 실험은 VMOS를 Org. VMOS로 고정하고 θ_{HQSAM}을 기준으로 Org. HQSAM과 Tiny HQSAM 중 하나를 선택할 수 있도록 설정하였다. 결과적으로, θ_{HQSAM}이 지나치게 크면 IoU가 급격히 감소하는 것으로 나타났다. 따라서 실행 시간을 크게 단축하면서도 IoU를 최소 95% 이상 유지할 수 있는 기준으로 θ_{HQSAM}을 11로 설정하였다.

나) Stage 2: Org. VMOS와 Linformer VMOS 선택을 위한 임계값 θ_{Tiny}^{Lin}와 θ_{Org}^{Lin}설정 방법

앞서 제시한 1단계 기준 설정을 통해 HQSAM 선택을 위한 θ_{HQSAM}을 확정하였다. θ_{HQSAM}에 따라 전체 프레임은 각각 Tiny HQSAM 또는 Org. HQSAM이 적용되며, 이에 따라 두 가지 모델 조합이 구성된다:

CD $\leq \theta_{HQSAM}$: Org. VMOS + Tiny HQSAM



[그림 3-6] CD 임계값에 따른 모델 조합

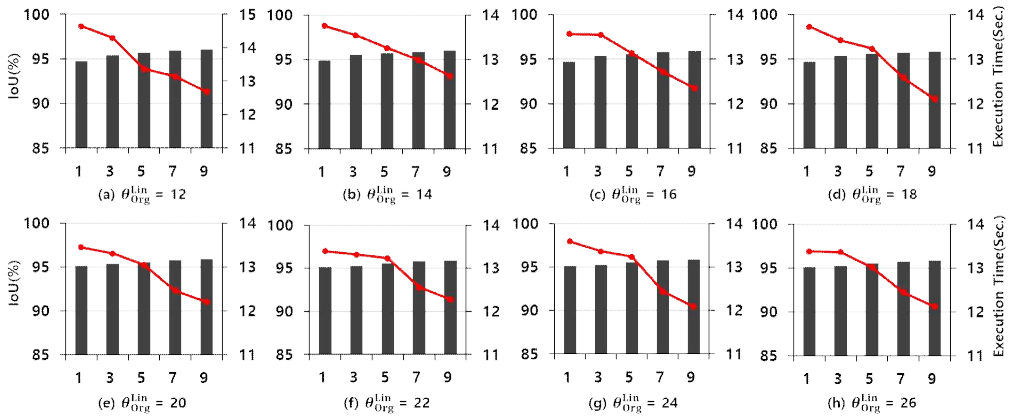
CD 〉 θ_{HQSAM} : Org. VMOS + Org. HQSAM

그러나 최종적으로 네 가지 모델 조합을 모두 활용하기 위해서는, 1단계에서 구분된 각 Tiny HQSAM, Org. HQSAM 적용 구간 내에서 Linformer 기반 VMOS 사용 여부를 추가로 판단하는 2단계 기준 설정이 필요하다.

[그림 3-6]은 각 CD 임계값을 기준으로 선택되는 모델 조합을 시각화한 것이다. θ^{Lin}_{Tiny}은 CD ≤ θ_{HQSAM} 구간에서 Linformer VMOS의 사용 여부를 판단하는 CD 임계값, θ^{Lin}_{Org}은 CD 〉 θ_{HQSAM} 구간에서 Linformer VMOS 사용 여부를 판단하는 임계값으로 정의된다. 이 두 CD 임계값 θ^{Lin}_{Tiny}, θ^{Lin}_{Org}은 각각 Tiny HQSAM과 Org. HQSAM 적용 구간에서 Linformer VMOS의 사용 여부를 결정하며, 본 연구에서는 이를 실험적으로 도출하였다. 이를 위해 각 카메라 데이터셋에 대해 네 가지 모델 조합을 선택하여 적용하면서, 프레임 단위 평균 IoU가 95% 이상 유지되면서 실행 시간이 최소화되는 θ^{Lin}_{Tiny}, θ^{Lin}_{Org}을 결정한다.

실험은 두 단계로 진행되었다. 먼저 1단계에서는 θ_{HQSAM}값을 11로 고정한 상태에서 θ^{Lin}_{Tiny}값을 일정 간격으로 증가시키며, 각 CD 임계값에 따라 선택된 모델 조합을 실행하여 IoU와 프레임당 평균 실행 시간을 측정하였다. 이어서 2단계에서는 θ^{Lin}_{Org}값을 변화시키면서 동일한 방식으로 θ^{Lin}_{Tiny}값을 조정하여 반복 실험을 수행하였다.

[그림 3-7]은 앞서 설명한 실험 절차에 따라 수행한 결과로, 다양한 임계값 설정에 따른 IoU와 프레임당 평균 실행 시간을 나타낸다. x축은 θ^{Lin}_{Tiny}값을 의미하며, 이에 따른 IoU와 평균 실행 시간의 변화를 확인할 수 있다. 각 그래프는 고정된 θ^{Lin}_{Org}값에 대한 θ^{Lin}_{Tiny}값 변화에 따른 결과를 나타내며, 모든 경



[그림 3-7] θ_{Tiny}^{Lin}와 θ_{Org}^{Lin}값의 변화에 따른 IoU 및 실행 시간 비교

우의 θ_{Tiny}^{Lin}과 θ_{Org}^{Lin}을 조합한 실험 결과를 나타낸다.

실험 결과, [그림 3-7]의 (g)에 제시된 바와 같이 θ_{Tiny}^{Lin}=9, θ_{Org}^{Lin}=24에서 최적의 조합이 도출되었다. 이러한 임계값을 기준으로 모델 조합을 선택하여 세그멘테이션을 진행하였을 때, IoU를 약 95% 이상 유지되면서 전체 실행 시간은 12.08초로 가장 많이 감소하였다. 이에 따라 각 HQSAM 구간 내에서 Linformer VMOS 적용 여부를 판단하는 CD 임계값이 확정되었으며, 이를 기반으로 1단계(HQSAM 선택)와 2단계(VMOS 선택)를 통합한 최종 네 가지 모델 조합에 따른 임계값을 결정하였다.

다) CD Measure와 Build Up $\mathbb{T}$의 동작 방식

본 절에서는 적응형 동적 모델 조합 선택 기법 기법의 핵심을 이루는 CD Measure와 Build Up $\mathbb{T}$의 세부 기능을 설명한다. 이 방법은 앞서 정의된 세 가지 CD 임계값을 활용하여 객체 움직임에 따라 모델을 동적으로 선택한다. 먼저, CD Measure는 $t-2^{th}$와 $t-1^{th}$프레임에서의 세그멘테이션 마스크간 CD를 계산한다. 이후 Build Up $\mathbb{T}$는 CD Measure로부터 계산된 CD 값을 기반으로, 프레임에서 각 트래커가 객체를 효과적으로 세그멘테이션하기 위해 필요한 정보인 $\mathbb{T}$를 출력한다.

$\mathbb{T}$는 다음과 같이 구성된다. 각 트래커에 대해 $t-2^{th}$와 $t-1^{th}$프레임 간의

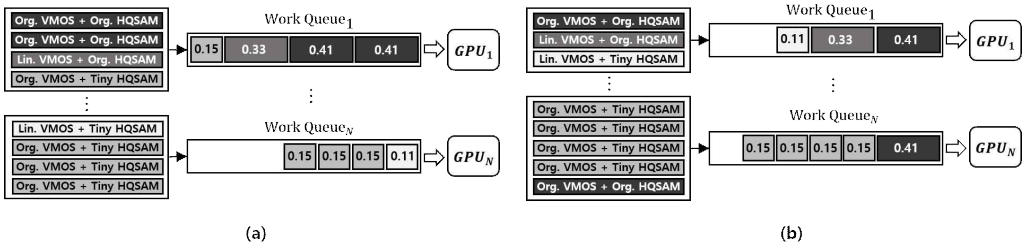
CD 값이 저장되며, 이는 사전에 정의된 세 가지 임계값을 기준으로 t^{th}프레임에서 적용할 모델 조합을 결정하는 데 활용된다. 또한 각 트래커가 수행할 모델 조합에 대응되는 Proportion 값이 함께 저장된다. 이 값은 네 가지 모델 조합 간의 상대적 실행 시간을 비교하는 지표로 사용된다.

아울러 각 트래커에 대해 t^{th}프레임에서 세그멘테이션에서 사용할 Long/Short-term memory와 encoder feature가 저장되어있는 GPU의 위치를 나타낸다. 이때, Long/Short-term memory와 encoder feature는 각 트래커가 t^{th}프레임에서 세그멘테이션을 수행할 때 필요한 정보이며 해당 정보가 위치한 GPU에서 트래커는 세그멘테이션이 실행되어야 한다. 이와 같은 정보를 포함하는 $\mathbb{T}$는 k개 트래커들의 집합을 포함하며, 전체 구조는 [그림 3-2] 하단에 제시되어 있다.

4) 코사인 유사도 기반 fine-grained 경량화 적용

앞선 내용에서는 CD를 기반으로 객체의 움직임 정도를 판단하여 모델 조합 선택을 수행하였다. 이를 기반으로 코사인 유사도를 활용한 fine-grained 경량화 기법을 추가적으로 제안한다. 해당 기법은 프레임 간 시각적 변화를 파악하고, 이를 바탕으로 Linformer 기반 projection으로는, 각 카메라에 대해 $t-1^{th}$프레임의 인코더 특징과 t^{th}프레임의 인코더 특징 간 코사인 유사도를 계산하였다. 코사인 유사도는 두 feature vector 간의 유사도를 정량화하는 지표로, 값이 1에 가까울수록 두 feature vector 간 시각적 변화가 적음을 의미한다.

이를 통해, 프레임 간 유사도가 높은 경우에는 projection dimension을 축소하여 연산량을 줄이고, 반대로 유사도가 낮은 경우에는 dimension을 증가시켜 정보를 보존한다. 이처럼 프레임 간 유사도에 따라 projection dimension을 조절하여 효율적인 추론을 가능하게 하였다. 본 논문에서는 코사인 유사도의 임계값을 0.8로 설정하였다. 이는 코사인 유사도 임계값을 0.6에서 0.9 사이의 값으로 조정하여 프레임당 실행 시간을 측정하였을 때 가장 짧은 실행 시간을 갖는 값으로 설정하였다. 이를 실험한 결과는 4장 2절 3)의 라)에서 자세히 나타난다. 따라서 유사도가 0.8 이상이면 projection



[그림 3-8] GPU 작업 분배 방식 비교: (a) 트래커 수 균등 분배, (b) Proportion 기반 트래커 분배

Model Combination	Avg. Execution Time	Proportion
Org. VMOS + Org. HQSAM	820ms	0.41
Linformer VMOS + Org. HQSAM	660ms	0.33
Org. VMOS + Tiny HQSAM	270ms	0.15
Linformer VMOS + Tiny HQSAM	210ms	0.11

[표 3-2] 각 모델 조합의 프레임당 평균 실행 시간과 Proportion 값

dimension을 64로 설정하여 경량화를 강화하고, 0.8 미만이면 dimension을 256으로 유지하여 정보 손실을 최소화하도록 설계하였다. dimension에 대한 자세한 내용은 4장 2절 2)의 가)에서 설명한다. 이러한 방식은 시각적 변화가 적은 구간에서는 연산량을 줄여 추론 속도를 개선하고, 변화가 큰 구간에서는 정보 보존을 우선시한다. 따라서, 전체 세그멘테이션 시스템의 추론 속도를 향상하는 동시에 성능 저하를 최소화하는 효과를 얻을 수 있다.

5) 적응형 GPU 부하 재분배 기법

다중 GPU 환경에서 트래커 수를 균등하게 분배하여 세그멘테이션을 수행할 경우, 각 트래커에 할당된 모델 조합의 실행 시간 차이로 인해 GPU 간 실제 처리 시간에 불균형이 발생할 수 있다. [그림 3-8]의 (a)에 제시된 그림은 다중 GPU 환경에서 각 GPU에 트래커 수를 동일하게 분배하여 실행하는 과정을 나타낸다. 이때, 실행 시간이 상대적으로 긴 모델 조합(e.g. Org. VMOS + Org. HQSAM)이 특정 GPU에 집중될 경우, 해당 GPU의 처리 시간이 지연된다. 그 결과, 일부 GPU가 더 일찍 작업을 완료하더라도 프레임

전체의 처리 시간은 여전히 길어질 수 있다.

이러한 문제를 완화하기 위해, [그림 3-8]의 (b)와 같이 매 프레임마다 각 GPU에 할당된 트래커들의 Proportion 총합을 계산하여 GPU별 예상 실행 시간 차이를 추정한다. Proportion 값은 각 모델 조합에 따라 정의되며, 단일

Symbol	Description
$\mathbb{T}$	Collection of trackers including their state information
k	Number of trackers
$Tracker$	Set of trackers, i.e., $\{\tau_1, \tau_2, \cdots, \tau_k\}$
$\triangle$	Period of memory copy via the system memory
j	Tracker index; $j \in \{1,2,\cdots,k\}$
τ_j	j^{th} tracker; $\tau_j \in Tracker$
p_j	Proportion value assigned to the j^{th} tracker
gpu_j	GPU ID used by the j^{th} tracker
t	Frame index
$Frame_t$	A list of k images in the t^{th} frame, i.e., $[img_1, img_2, \cdots, img_k]$
img_j	Input image of the j^{th} tracker; $img_j \in Frame_t$
$\mathcal{G}$	Number of GPU groups
m, c	GPU group index; $m, c \in \{1,2,\cdots,\mathcal{G}\}$
gpu_link_set	Set of GPU groups, i.e., $\{G_1, G_2, \cdots, G_{\mathcal{G}}\}$
G_m	m^{th} GPU group in gpu_link_set, i.e., $\{g_m^1, g_m^2, \cdots, g_m^q\}$
q	Number of GPUs included in group G_m
o, f	GPU indices in group G_m; $o, f \in \{1,2,\cdots,q\}$
g_m^o, g_m^f	GPU ID selected from G_m; $g_m^o, g_m^f \in G_m$
$mig_{\max}$	Maximum number of migrations between g_m^o and g_m^f
src	ID of the source GPU(migration origin)
dst	ID of the destination GPU(migration target)
P_{src}	Sum of proportion values of trackers assigned to the src GPU
P_{dst}	Sum of proportion values of trackers assigned to the dst GPU
I_{src}	Set of tracker indices assigned to the src GPU; $I_{src} \subseteq \{1,2,\cdots,k\}$
S^*	Set of indices corresponding to trackers for migration; $S^* \subseteq \{1,2,\cdots,k\}$

[표 3-3] 표기법

이미지를 처리하는 데 필요한 평균 실행 시간을 정규화한 값이다. 이는 프레임당 실행 시간을 단축시키는 핵심 지표로 활용된다. [표 3-2]는 모든 모델 조합에 대해 정의된 Proportion 값을 나타낸다. 각 트래커는 CD 값에 따라 네 가지 모델 조합 중 하나를 선택하고, 해당 모델 조합에 대응하는 Proportion을 할당받는다. 이를 통해 트래커의 작업이 GPU 간 균등하게 분배되도록 설계함으로써 프레임당 실행 시간을 단축한다.

본 연구에서는 다중 GPU 환경에서 GPU 간 실행 시간 불균형으로 인해 발생하는 프레임 단위 지연 문제를 해결하기 위해 적응형 GPU 부하 재분배 기법을 제안한다. 제안된 기법은 각 트래커의 Proportion 값을 기반으로 세그멘테이션이 수행될 GPU를 재할당하여 작업을 효과적으로 조정한다. 이를 통해 각 GPU에서 수행되는 세그멘테이션의 실행 시간이 균등하게 분배된다. GPU Link를 이용한 메모리 복사 방식은 GPU Link로 연결된 GPU들로 이루어진 GPU 그룹 내부의 실행 시간 편차를 줄일 수 있으나, 서로 다른 그룹 간 편차는 해소하지 못한다. 이를 보완하기 위해, 본 연구는 시스템 메모리를 거쳐 대상 GPU로 메모리 복사하는 과정을 통해 전역적으로 GPU 간 실행 시간 균형 조정을 수행한다. 이를 통해 서로 다른 GPU 그룹 간 객체 세그멘테이션 실행 시간 편차를 줄이고, 시스템 전체의 처리 효율을 향상시킨다. 또한, 본 기법의 이해를 돕기 위해 자주 사용되는 주요 표기법을 [표 3-3]에 정리하였다.

[그림 3-9]는 적응형 GPU 부하 재분배 기법의 핵심 동작을 나타내는 수도코드이다. AdaptiveLoadDistributor 함수의 입력으로는 이전 단계(Build Up $\mathbb{T}$)에서 생성된 $\mathbb{T}$, GPU 그룹들의 집합 $gpu\_link\_set = \{G_1, G_2, \cdots, G_\mathcal{G}\}$, 그리고 트래커의 최대 이동 개수 $mig_{\max}$가 주어진다. 이때 각 GPU 그룹 $G_m = \{g_m^1, g_m^2, \cdots, g_m^q\}$은 GPU Link로 연결된 GPU들의 집합인 m^{th} GPU 그룹을 의미한다.

먼저 GPU Link로 연결되지 않은 서로 다른 GPU 그룹 간 실행 시간을 최소화하기 위한 작업 재할당 탐색 과정에 대하여 설명한다. 서로 다른 GPU 그룹 간에는 GPU Link로 연결되어있지 않아 시스템 메모리를 거쳐 대상 GPU로 작업의 메모리 복사가 이루어진다. 시스템 메모리를 거치는 메모리

Algorithm 1 adaptive GPU load redistribution

```
1: function AdaptiveLoadDistributor(𝕋, gpu_link_set, mig_max)
2:     if t mod Δ = 0 then
3:         for m ← 1 to 𝒢 − 1 do
4:             for c ← m + 1 to 𝒢 do
5:                 (src, dst, P_src, P_dst) ← GlobalPropCalc(𝕋, G_m, G_c)
6:                 S*, src, dst ← MatchTrackersToGPUs(𝕋, src, dst, P_src, P_dst, mig_max)
7:                 GlobalTrackerMigration(𝕋, S*, src, dst)
8:             end for
9:         end for
10:    end if
11:    for m ← 1 to 𝒢 do
12:        for o ← 1 to q − 1 do
13:            for f ← o + 1 to q do
14:                (src, dst, P_src, P_dst) ← GPULinkPropCalc(𝕋, g_m^o, g_m^f)
15:                S*, src, dst ← MatchTrackersToGPUs(𝕋, src, dst, P_src, P_dst, mig_max)
16:                GPULinkTrackerMigration(𝕋, S*, src, dst)
17:            end for
18:        end for
19:    end for
20:    MapTrackersToWorkQueue(Tracker)
21: end function
```

[그림 3-9] 적응형 GPU 부하 재분배 기법의 동작을 나타내는 수도코드

복사 방식은 GPU Link를 통한 메모리 복사보다 상대적으로 높은 전송 지연을 수반한다. 만약 이를 매 프레임마다 수행한다면 전체 실행 시간이 증가하여 성능 저하를 유발할 수 있다. 이러한 점을 고려하여 본 연구에서는 △ 프레임 주기로 시스템 메모리 기반 전역 균형 조정을 수행하도록 설정하였다([그림 3-9], 라인 2). 실제 실험에는 △를 30프레임으로 설정하였으며 이 값은 실험에 근거하며, 이에 대한 상세한 설명은 4장 2절 3)의 다)에 기술되어 있다.

GlobalPropCalc 함수는 전체 GPU 그룹을 대상으로 𝕋와 두 그룹 G_m과 G_c를 입력으로 사용한다. 이 함수는 각 그룹에서 하나의 GPU를 선택하여 source GPU와 destination GPU를 결정하고, 이들의 총 Proportion 값 P_{src}와 P_{dst}를 반환한다([그림 3-9], 라인 3~5). 이어서 MatchTrackersToGPUs 함수가 호출되어, $mig_{\max}$ 조건을 고려하여 두 GPU 간 Proportion 차이를 최소화할 수 있는 재할당 대상의 트래커 인덱스를 탐색한다. 그 결과, 전체 GPU 그룹을 대상으로 시스템 메모리를 거쳐 src에서 dst로 복사될 트래커

인덱스 집합 S^*과 해당 src, dst가 반환된다([그림 3-9], 라인 6). 이 값을 입력으로 하는 GlobalTrackerMigration 함수는 실제로 src에서 dst로 이동 대상 트래커들의 정보를 시스템 메모리를 거쳐 메모리 복사한다. 이후 해당 트래커들에 대해 t^{th}프레임에서 세그멘테이션이 수행될 GPU의 위치를 dst로 업데이트한다([그림 3-9], 라인 7). 이를 통해 GPU Link로 연결되지 않은 서로 다른 GPU 그룹 간 실행 시간 편차를 최소화한다. GlobalPropCalc 함수와 MatchTrackersToGPUs 함수는 각각 3장 2절 5)의 가) [그림 3-10]과 3장 2절 5)의 다) [그림 3-12]에서, GlobalTrackerMigration 함수는 3장 2절 5)의 라) [그림 3-13]에서 상세히 설명한다.

이후 GPU Link로 연결된 GPU 그룹 내에서 GPU 간 실행 시간 편차를 최소화하기 위해, gpu_link_set 내의 $\mathcal{G}$개의 GPU 그룹을 순차적으로 탐색한다([그림 3-9], 라인 11~13). 이 과정은 매 프레임마다 수행되며, △프레임 주기에는 서로 다른 GPU 그룹 간 균형화를 수행한 뒤, 이어서 진행된다. 구체적으로, 그룹 내 모든 GPU 쌍 (g_m^o, g_m^f)에 대해 GPULinkPropCalc 함수를 수행한다([그림 3-9], 라인 14). 이 함수는 두 개의 GPU 그룹을 입력받는 GlobalPropCalc 함수와 다르게, 같은 그룹 내에 있는 두 GPU에 할당된 트래커들의 Proportion 합을 계산한다. 이후, source GPU(src)와 destination GPU(dst)를 결정하고, 이들의 총 Proportion 값 P_{src}, P_{dst}과 함께 반환된다. 이에 대한 자세한 내용은 3장 2절 5)의 나) [그림 3-11]에서 설명한다.

이후, GPULinkPropCalc 함수가 수행되면 이전과 동일하게 MatchTrackersToGPUs 함수가 호출되고, src에서 dst로 복사될 트래커 인덱스 집합 S^*과 src, dst가 반환된다([그림 3-9], 라인 15). 이 값을 입력으로 하는 GPULinkTrackerMigration 함수는 GPU Link를 통해 src에서 dst로 메모리 복사를 수행하며, 해당 트래커에 대해 세그멘테이션이 수행될 GPU의 위치를 dst로 업데이트한다([그림 3-9], 라인 16). 이를 통해 GPU Link로 연결된 GPU 그룹 내 실행 시간 불균형이 줄어든다. 최종적으로 시스템 메모리를 거치는 메모리 복사 및 GPU Link를 통한 메모리 복사 과정이 모두 완료되면 MapTrackersToWorkQueue 함수에서 각 k개의 트래커는 재할당된 GPU에서 세그멘테이션을 수행하게 된다([그림 3-9], 라인 20).

Algorithm 2 Global Proportion Calculator

function GLOBALPROPCALC($\mathbb{T}, G_m, G_c$)
  $P\_list_m \leftarrow [0, 0, \cdots, 0] \in \mathbb{R}^{|G_m|}$
  $P\_list_c \leftarrow [0, 0, \cdots, 0] \in \mathbb{R}^{|G_c|}$
  for all $g_m^o \in G_m$ **do**
    $T_m^o \leftarrow \{ \tau_j \in \text{Tracker} \mid \text{gpu}_j == g_m^o \}$
    $P_m^o \leftarrow \sum_{\tau_j \in T_m^o} p_j$
    $P\_list_m[o] \leftarrow P_m^o$
  end for
  for all $g_c^f \in G_c$ **do**
    $T_c^f \leftarrow \{ \tau_j \in \text{Tracker} \mid \text{gpu}_j == g_c^f \}$
    $P_c^f \leftarrow \sum_{\tau_j \in T_c^f} p_j$
    $P\_list_c[f] \leftarrow P_c^f$
  end for
  $\text{avg}_m \leftarrow \frac{1}{|G_m|} \sum_{o \in \{1,\cdots,q\}} P\_list_m[o]$
  $\text{avg}_c \leftarrow \frac{1}{|G_c|} \sum_{f \in \{1,\cdots,q\}} P\_list_c[f]$
  if $\text{avg}_m > \text{avg}_c$ **then**
    $idx\_src \leftarrow \arg\max_{o \in \{1,\cdots,q\}} P\_list_m[o]$
    $idx\_dst \leftarrow \arg\min_{f \in \{1,\cdots,q\}} P\_list_c[f]$
    $src \leftarrow g_m^{idx\_src}, \;\; dst \leftarrow g_c^{idx\_dst}$
    $P_{src} \leftarrow P\_list_m[idx\_src], \;\; P_{dst} \leftarrow P\_list_c[idx\_dst]$
  else
    $idx\_src \leftarrow \arg\max_{f \in \{1,\cdots,q\}} P\_list_c[f]$
    $idx\_dst \leftarrow \arg\min_{o \in \{1,\cdots,q\}} P\_list_m[o]$
    $src \leftarrow g_c^{idx\_src}, \;\; dst \leftarrow g_m^{idx\_dst}$
    $P_{src} \leftarrow P\_list_c[idx\_src], \;\; P_{dst} \leftarrow P\_list_m[idx\_dst]$
  end if
  return $(src, dst, P_{src}, P_{dst})$
end function

[그림 3-10] GlobalPropCalc 함수의 동작을 나타내는 수도코드

MapTrackersToWorkQueue 함수에 대한 과정은 3장 2절 5)의 라) [그림 3-13]에서 자세히 설명한다.

한편, 수도코드상에서는 반복문이 다중 중첩 구조를 가지므로 연산 비용이 커 보일 수 있다. 그러나 일반적인 실제 서버 환경에서는 GPU Link 개수와 GPU Link로 연결된 GPU의 숫자는 유한적이다(많아야 10개 혹은 20개 이내). 따라서 탐색해야 할 범위는 매우 제한적이고 중첩 반복문 형태의 알고리즘으로 인한 연산량 증가는 거의 나타나지 않는다.

가) Global Proportion Calculator

[그림 3-10]는 서로 다른 GPU 그룹 간의 실행 시간 편차를 줄이기 위한

GlobalPropCalc 함수의 동작 과정을 나타낸다. 함수는 $\mathbb{T}$와 두 GPU 그룹(G_m, G_c)을 입력으로 받아, 서로 다른 GPU 간의 Proportion 차이를 최소화할 수 있는 src, dst GPU를 결정한다.

먼저, 주어진 $\mathbb{T}$에서 j^{th}트래커가 $t-1^{th}$프레임에서 사용한 GPU, 즉 세그멘테이션 수행에 필요한 Long/Short-term memory와 encoder feature 데이터가 위치한 GPU ID를 gpu_j라 정의한다. 각 그룹 G_m과 G_c의 GPU에 대해, G_m에서 gpu_j == g_m^o인 경우 해당 트래커 집합을, G_c에서 gpu_j == g_c^f인 경우 해당 트래커 집합을 각각 추출한다([그림 3-10], 라인 4~5, 9~10). 이후, 각 집합 내 트래커의 Proportion 값을 누적하여 GPU별 Proportion 합을 계산하고, 이를 P_list_m과 P_list_c에 저장한다([그림 3-10], 라인 6~7, 11~12). 이렇게 얻어진 리스트는 각 그룹 내부 GPU들에 해당하는 트래커들의 Proportion 누적값들을 나타낸다.

GPU Link를 통한 메모리 복사를 수행하면, GPU 그룹 내부의 GPU 간 실행 시간 편차가 줄어들며, 이로 인해 각 GPU의 Proportion 누적값은 점차 해당 GPU 그룹의 평균 Proportion 값에 수렴하게 된다. 따라서 서로 다른 그룹 간 평균 Proportion 값을 비교하면, GPU Link로 연결되지 않은 그룹 간 실행 시간 편차를 파악할 수 있다. 이를 위해 GPU Link로 연결되지 않은 서로 다른 그룹 G_m과 G_c 각각에 대해 평균 Proportion 값을 계산한다([그림 3-10], 라인 14~15). 만약 G_m의 평균 Proportion이 G_c 보다 크다면, G_m이 상대적으로 실행 시간이 긴 그룹을 의미하므로 G_m내부에서 Proportion이 가장 큰 GPU를 source로, G_c내부에서 Proportion이 가장 작은 GPU를 destination으로 선택한다([그림 3-10], 라인 16~20). 반대로, G_c의 평균 Proportion이 더 클 경우 G_c에서 가장 큰 Proportion 값을 갖는 GPU를 source로, G_m에서 Proportion이 가장 작은 GPU를 destination으로 지정한다([그림 3-10], 라인 21~26). 이후 각 src와 dst에 대해 누적된 Proportion 값은 각각 P_{src}와 P_{dst}로 저장되며, 이 결과는 이후 단계인 Tracker-GPU Matching에서 사용된다.

이는 서로 다른 GPU 그룹 간 실제 실행 시간의 병목은 여전히 가장 실행

시간이 긴 GPU에 의해 결정되기 때문에 서로 다른 GPU 그룹에서 가장 Proportion 값이 큰 GPU와 Proportion 값이 가장 작은 GPU 간 실행 시간 편차가 최소화되도록 하는 전략이다. 이러한 과정을 통해 GPU Link로 연결되지 않은 GPU 간의 간 편차를 최소화할 수 있는 균형 조정 방향을 결정한다.

나) GPU Link Proportion Calculator

Algorithm 3 GPU Link Proportion Calculator

1: **function** GPULINKPROPCALC($\mathbb{T}, g_m^o, g_m^f$)
2: $T_u \leftarrow \{ \tau_j \in Tracker \mid gpu_j == g_m^o \}$
3: $P_u \leftarrow \sum_{\tau_j \in T_u} p_j$
4: $T_v \leftarrow \{ \tau_j \in Tracker \mid gpu_j == g_m^f \}$
5: $P_v \leftarrow \sum_{\tau_j \in T_v} p_j$
6: **if** $P_u > P_v$ **then**
7: $(src, dst) \leftarrow (g_m^o, g_m^f)$
8: $(P_{src}, P_{dst}) \leftarrow (P_u, P_v)$
9: **else**
10: $(src, dst) \leftarrow (g_m^f, g_m^o)$
11: $(P_{src}, P_{dst}) \leftarrow (P_v, P_u)$
12: **end if**
13: **return** $(src, dst, P_{src}, P_{dst})$
14: **end function**

[그림 3-11] GPULinkPropCalc 함수의 동작을 나타내는 수도코드

[그림 3-11]은 Adaptive GPU Load Redistribution([그림 3-9]) 내에서 동작하는 GPULinkPropCalc 함수의 수도코드를 나타낸다. 이 함수는 t^{th}프레임에서 각 k개의 트래커가 세그멘테이션에 필요한 정보와 이들의 집합으로 구성된 $\mathbb{T}$, GPU Link로 연결된 두 GPU (g_m^o, g_m^f)를 입력으로 받는다.

먼저, 주어진 T에서 gpu_j == g_m^o이면 해당 트래커의 Proportion 값을 누적하여 P_u를 계산하고, gpu_j == g_m^f이면 해당 트래커의 Proportion 값을 누적하여 P_v를 계산한다([그림 3-11], 라인 2~5). 이후 P_u와 P_v를 비교하여 더 큰 값을 가지는 GPU를 src, 더 작은 값을 가지는 GPU를 dst로 설정한다([그림 3-11], 라인 6~7, 9~10). 각 src와 dst에 대해 누적된 Proportion 값은 각각 P_{src}와 P_{dst}로 저장되며([그림 3-11], 라인 8, 11), 이 결과는 이후

Algorithm 4 Tracker–GPU Matching

1: **function** MATCHTRACKERSTOGPUS($\mathbb{T}, src, dst, P_{src}, P_{dst}, mig_{max}$)
2: $I_{src} \leftarrow \{ j \in \{1, \cdots, k\} \mid gpu_j == src \}$
3: $\delta_{\min} \leftarrow P_{src} - P_{dst}$
4: $S^* \leftarrow \emptyset$
5: **for** each $S \in \{ S \subseteq I_{src} \mid 1 \leq |S| \leq mig_{max} \}$ **do**
6: $T_S \leftarrow \{ \tau_j \in Tracker \mid j \in S \}$
7: $P_S \leftarrow \sum_{\tau_j \in T_S} p_j$
8: $\delta_{\text{trial}} \leftarrow |(P_{src} - P_S) - (P_{dst} + P_S)|$
9: **if** $\delta_{\text{trial}} < \delta_{\min}$ **then**
10: $\delta_{\min} \leftarrow \delta_{\text{trial}}$
11: $S^* \leftarrow S$
12: **if** $\delta_{\text{trial}} == 0$ **then**
13: **break**
14: **end if**
15: **end if**
16: **end for**
17: **return** S^*, src, dst
18: **end function**

[그림 3-12] Tracker-GPU Matching 함수의 동작을 나타내는 수도코드

단계인 Tracker-GPU Matching의 입력으로 사용된다([그림 3-11], 라인 13). 결과적으로, GPULinkPropCalc는 GPU Link로 연결된 GPU 쌍 (g_m^o, g_m^f)의 Proportion 합을 비교하여 GPU 간 편차를 최소화할 수 있는 균형 조정 방향을 결정한다. 이를 통해 Proportion 합이 더 큰, 즉 예상 실행 시간이 긴 GPU에서 더 작은 GPU로 작업이 재할당되도록 한다.

다) Tracker-GPU Matching

[그림 3-12]는 함수 MatchTrackersToGPUs로, Tracker-GPU Matching 과정을 나타낸다. 이 함수는 t^{th}프레임에서 GlobalPropCalc 함수 및 GPULinkPropCalc 함수가 계산한 Proportion 편차를 줄이는 방향으로 최적의 GPU 할당을 결정한다. 해당 과정은 GPU 그룹 내부의 실행 시간 균형화 뿐만 아니라, GPU Link로 연결되지 않은 서로 다른 GPU 그룹 간 실행 시간 균형화에도 동일하게 적용된다. 이 함수의 입력으로는 src, dst에 대한 누적 Proportion 값 P_{src}와 P_{dst}, 그리고 GPU 간 트래커 재할당 시 이동할 최대 개수 $mig_{\max}$가 주어진다.

먼저, I_{src}는 src GPU와 동일한 gpu_j에 해당하는 트래커들의 인덱스로 이루어진 집합이다([그림 3-12], 라인 2). src와 dst간의 Proportion 편차 $\delta_{\min}$은 $P_{src} - P_{dst}$로 정의된다([그림 3-12], 라인 3). I_{src}내에서 크기가 1 이상 $mig_{\max}$이하인 모든 부분집합 S를 대상으로 탐색을 진행한다([그림 3-12], 라인 5). 이는 전체 집합에 대한 완전 탐색으로 인해 발생할 수 있는 과도한 지연을 방지하기 위함이다. 이동할 트래커 인덱스 집합 S에 대해, S에 속한 트래커 인덱스에 해당하는 트래커들의 Proportion 합 P_S를 계산하고 ([그림 3-12], 라인 6~7), S를 src에서 dst로 이동시켰을 때의 예상 Proportion 편차 δ_{trial}를 구한다([그림 3-12], 라인 8). 해당 과정은 δ_{trial}값을 최소화하는 부분집합을 찾기 위해 반복을 수행한다. 만약 $\delta_{trial} < \delta_{\min}$이면 $\delta_{\min}$을 δ_{trial}로 갱신하고, S를 최적의 인덱스 집합 S^*로 기록한다([그림 3-12], 라인 9~11). 또한 δ_{trial}이 0인 경우에는 이상적인 균형 상태가 달성된 것으로 판단하고 탐색을 즉시 종료한다([그림 3-12], 라인 12-14).

최종적으로 최적의 트래커 인덱스 집합인 S^*와 src, dst가 반환된다. 즉, Tracker-GPU Matching은 Proportion 기반 각 GPU 간의 실행 시간 불균형을 최소화할 수 있도록 트래커의 GPU 할당을 탐색하고 재구성하여, 전체 시스템의 실행 시간 균형을 달성하는 데 기여한다.

라) Tracker Migration and Tracker Work Queue Mapping

[그림 3-13]은 이전 MatchTrackersToGPUs 결과에 따라 실제 트래커가 세그멘테이션에 필요한 정보를 GPU 간에 메모리 복사하는 GlobalTrackerMigration 함수와 GPULinkTrackerMigration 함수의 동작 과정을 나타낸다. 두 함수는 ($\mathbb{T}$, S^*, src, dst)를 입력으로 받아 각각 시스템 메모리를 거치는 복사와 GPU Link를 통한 메모리 복사를 수행한다.

일정 주기($\triangle$)마다 수행되는 GlobalTrackerMigration 함수는 S^*내 인덱스에 해당하는 트래커들([그림 3-13], 라인 2)을 대상으로 다음과 같은 2단계 메모리 복사가 수행된다. 먼저, D2H(Device-to-Host) 방식을 통해 src에서 Long/Short-term memory와 encoder feature를 시스템 메모리(Host

Algorithm 5 Tracker Migration, Tracker Work Queue Mapping

```
1: function GlobalTrackerMigration(𝕋, S*, src, dst)
2:     for each τ_j ∈ { τ_j ∈ Tracker | j ∈ S* } do
3:         D2H(long/short-term memory, system memory, src)
4:         H2D(long/short-term memory, dst, system memory)
5:         D2H(Encoder features, system memory, src)
6:         H2D(Encoder features, dst, system memory)
7:         gpu_j ← dst                          ▷ Update gpu_j within 𝕋
8:     end for
9: end function
10: function GPULinkTrackerMigration(𝕋, S*, src, dst)
11:     for each τ_j ∈ { τ_j ∈ Tracker | j ∈ S* } do
12:         D2D(long/short-term memory, dst, src)
13:         D2D(Encoder features, dst, src)
14:         gpu_j ← dst                          ▷ Update gpu_j within 𝕋
15:     end for
16: end function
17: function MapTrackersToWorkQueue(Tracker, Frame_t)
18:     for each τ_j ∈ Tracker do
19:         img_j ← Frame_t[j]
20:         InsertQ(τ_j, img_j)
21:     end for
22: end function
```

[그림 3-13] 시스템 메모리를 거치는 복사와 GPU Link를 통한 메모리 복사의 동작 과정을 나타내며 세그멘테이션이 수행되는 과정을 나타내는 수도코드

Memory)로 전송한다([그림 3-13], 라인 3, 5). 이후, H2D(Host-to-Device) 방식을 통해 시스템 메모리에 저장된 데이터를 dst로 다시 복사한다([그림 3-13], 라인 4, 6). 메모리 복사가 완료된 해당 트래커의 dst를 gpu_j로 갱신하여, 다음 GPU 간 실행 시간 편차 최소화 과정은 해당 업데이트된 gpu_j를 바탕으로 수행된다.

반면, GPULinkTrackerMigration 함수는 매 프레임마다 동일한 GPU 그룹 내에서 GPU Link로 직접 연결된 src와 dst를 대상으로 실행된다. 따라서, Long/Short-term memory와 encoder feature 데이터가 src에서 dst로 D2D(Device-to-Device)를 통해 직접 복사된다([그림 3-13], 라인 10~13). 이후, 세그멘테이션에 필요한 정보들의 메모리 복사가 완료되면 해당 트래커의 gpu_j를 dst로 갱신한다. 이를 통해 gpu_j를 지속적으로 업데이트하여, 해당 시스템의 GPU 환경에 맞게 Adaptive GPU Load Redistribution([그림 3-9])

과정을 반복적으로 수행한다.

MapTrackersToWorkQueue는 각 트래커들을 각 GPU의 작업 큐에 배치하여, 세그멘테이션을 수행할 수 있도록 하는 단계이다. 앞선 과정을 통해 각 k개의 트래커는 t^{th}프레임에서 어느 GPU에서 실행될지 결정되며, 이에 따라 각 트래커는 할당된 GPU인 gpu_j에서 실행된다. 각 트래커와 해당 트래커에 대응하는 t^{th}프레임의 입력 이미지를 함께 gpu_j의 작업 큐에 삽입함으로써, 각 GPU는 자신에게 할당된 큐의 작업을 모든 GPU에 대하여 병렬로 세그멘테이션을 수행하게 된다([그림 3-13], 라인 17-22).

제안된 프레임워크는 GPU 간 연결 상태를 고려하여 수행하며 메모리 복사 시간을 최소화하는 동시에 복사 시간이 전체 실행 시간 편차를 유발하지 않도록 보장하는 데 목적이 있다. 또한, 트래커가 세그멘테이션에 필요한 정보를 GPU 간 복사할 수 있도록 하며, 이를 통해 각 GPU의 프레임당 세그멘테이션 실행 시간을 균형 있게 조정한다. 결과적으로 프레임당 실행 시간이 단축되며, 이를 증명하기 위한 실험 결과는 4장 2절 2)의 바)에서 자세히 확인할 수 있다.

제 4 장 실험 결과 및 분석

본 장에서는 3장에서 제안한 플렌옵틱 이미지 및 멀티 카메라 이미지 환경을 대상으로, 객체 추적과 세그멘테이션의 프레임 단위 처리 효율을 향상시키기 위해 설계된 각 프레임워크의 효과를 검증하고 실험 결과를 분석한다. 각 절에서는 제안된 프레임워크의 효용성을 평가하기 위해 수행된 실험 절차와 방법을 상세히 기술한다.

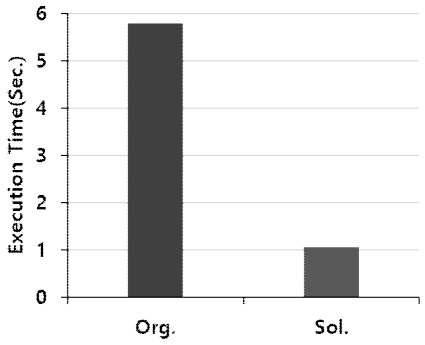
제 1 절 플렌옵틱 이미지에서의 객체 추적 프레임워크 실험 결과

1) 실험 환경 및 구현

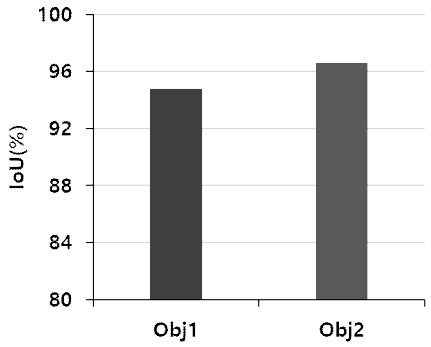
본 절에서는 제안한 프레임워크의 성능을 검증하기 위해 구성된 실험 환경과 절차를 기술한다. 모든 실험은 AMD Ryzen Threadripper PRO 3955WX(16코어) CPU와 48GB GPU 메모리를 갖춘 NVIDIA RTX A6000 GPU로 이루어진 서버에서 수행하였다. 먼저, 기존 방식(Org.)과 제안 방식(Sol.)의 출력 결과를 Ground Truth와 각각 비교하여 두 방식 간 성능 차이가 통계적으로 유의미한 결과를 갖는지 분석하였다. 이후, 프레임 처리 시간을 측정하여 Sol.의 실행 효율이 향상되었는지를 평가하였다. 또한 Sol.에서는 총 8개의 워커 스레드를 활용하여 작업이 병렬적으로 수행되도록 설정하였다.

입력 데이터는 플렌옵틱 이미지를 사용하였으며, 실험은 65프레임부터 120프레임 구간에서 진행하였다. 각 프레임은 101장의 포컬 플레인 이미지로 구성되어 있다. 또한 두 개의 객체를 지정하여 다중 객체 추적을 수행하였다. 평가 지표로는 객체 추적 정확도를 나타내는 IoU와 프레임당 평균 실행 시간을 사용하였다. IoU는 추적된 객체의 세그멘테이션 마스크와 Ground Truth 간 교집합 영역을 합집합 영역으로 나눈 비율로 정의되며, 값이 높을수록 객체 세그멘테이션 정확도가 우수함을 의미한다.

2) 실험 결과 및 분석



[그림 4-1] 프레임당 평균 객체 추적 실행 시간



[그림 4-2] 객체 별 IoU 측정 결과

가) 객체 추적 실행 시간

[그림 4-1]은 65프레임부터 120프레임을 대상으로 수행된 실행 시간 측정 결과를 나타낸다. 각 프레임은 101장의 포컬 플레인 이미지로 구성되었으며, 제안한 객체 추적 프레임워크의 성능 향상을 정량적으로 확인하기 위해 프레임당 평균 실행 시간을 비교하였다.

먼저 기존 Org.의 경우 프레임당 평균 5.79초가 소요되었다. 반면, 제안한 Sol.에서는 1.06초의 평균 실행 시간이 측정되었다. 이는 제안 방식이 Org. 대비 약 81.7%의 실행 시간을 단축한 것으로, 전체 처리 파이프라인의 효율성이 크게 향상되었음을 시사한다. 이러한 결과는 제안 프레임워크가 포컬 플레인 이미지로 구성된 프레임 기반 처리에서도 높은 연산 효율성을 제공하며, 실시간 처리가 필요한 응용 환경에 충분히 적용 가능함을 보여준다.

나) 객체 추적 정확도

제안된 프레임워크의 실행 시간 향상 효과뿐만 아니라, 세그멘테이션 정확도 측면의 성능 변화 또한 평가하였다. 이를 위해 동일한 데이터셋을 대상으로 객체별 IoU을 측정하였다. [그림 4-2]는 전체 프레임에 대해 Ground Truth의 바운딩 박스와 제안한 프레임워크의 바운딩 박스 결과 간 IoU를 프레임 단위로 측정한 뒤, 이를 평균하여 얻은 성능 지표를 나타낸 것이다. 그 결과, Obj 1의 IoU는 94.79%, Obj 2의 IoU는 96.59%로 나타나 전반적으로 높은 정확도를 유지함을 확인할 수 있었다.

특히 주목할 점은, 제안된 Sol. 프레임워크가 기존 Org. 방식 대비 실행 시간을 크게 단축함에도 불구하고 세그멘테이션 정확도에서는 거의 손실이 발생하지 않았다는 것이다. 이는 연산 병렬화 및 처리 파이프라인 최적화를 통해 GPU 간 실행 시간 편차를 효과적으로 줄인 결과로 해석할 수 있으며, 전체 프레임 처리의 효율성이 향상되었음을 시사한다.

제 2 절 멀티 카메라 이미지에서의 세그멘테이션 프레임워크 실험 결과

1) 실험 환경 및 구현

본 실험은 제안한 기법의 효용성을 검증하고 기존 방식 대비 성능 향상 여부를 평가하기 위해 수행되었다. 각 기법의 출력 결과를 Ground Truth와 비교하여 정확도를 분석하고, 실행 시간을 측정함으로써 제안 기법의 실질적인 성능 개선 효과를 확인하였다.

실험은 앞선 절에서 사용한 플렌옵틱 이미지 기반 객체 추적 프레임워크와 동일한 AMD Ryzen Threadripper PRO 3955WX(16코어) CPU와 48GB GPU 메모리를 갖춘 NVIDIA RTX A6000 GPU로 이루어진 서버에서 수행하였다. 해당 시스템의 GPU Link는 최대 2-way NVLink 구성을 지원하며, 2-slot 및 3-slot low-profile 브리지를 통해 두 개의 GPU를 연결할 수 있다. 이때 NVIDIA NVLink의 양방향(bidirectional) 대역폭은 112.5 GB/s이

다. 이에 따라, 실험에 사용된 gpu_link_set은 G_1과 G_2의 두 그룹으로 구성되며, 각 그룹은 G_1={g_1^1, g_1^2}={1번 GPU, 4번 GPU}, G_2={g_2^1, g_2^2}={2번 GPU, 3번 GPU}로 정의된다. 또한 제안한 프레임워크에서 $mig_{\max}$는 3으로 설정하였다.

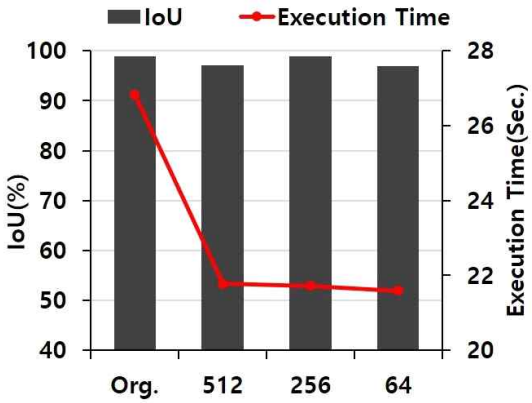
입력 데이터로는 32대의 카메라로 동시 촬영된 멀티 카메라 비디오가 사용되었다. 각 프레임은 32개의 카메라로부터 수집된 총 32장의 이미지로 구성되며 총 75프레임을 사용하였다. 평가 지표로는 세그멘테이션 정확도를 측정하는 IoU와 프레임당 평균 실행 시간을 사용하였다.

실험은 다음 네 가지 구성으로 진행되었다. Case 1은 적응형 동적 모델 조합 선택 기법만 적용한 구성으로, Linformer의 projection dimension을 256으로 설정하였다. Case 2는 Case 1에 코사인 유사도 기반의 fine-grained 경량화 기법을 추가하여 수행한 실험이다. 구체적으로, 연속된 프레임 간 인코더 특징 간 코사인 유사도가 0.8 이상일 경우 projection dimension을 실행 시간이 가장 짧은 64로 축소하고, 그렇지 않으면 IoU가 가장 높은 256을 유지하도록 구성하였다. 이러한 증명은 4장 2절 2)의 가)를 통해 확인할 수 있다. Case 3은 Case 2와 동일한 방법으로 세그멘테이션을 진행하며 추가로 매 프레임마다 GPU Link Proportion Calculator를 활용하여 GPU 그룹 내부의 각 GPU 실행 시간 편차를 최소화한 경우이다. 마지막으로, 제안한 기법(Sol.)은 Case 3 와 동일하게 매 프레임마다 GPU 그룹 내부의 실행 시간 균형을 달성하며 동시에, GPU Link로 연결되지 않은 GPU 그룹 간에는 Global Proportion Calculator를 통해 30프레임 주기마다 시스템 전반의 GPU 실행 시간 편차를 최소화하는 방식을 적용하였다.

2) 실험 결과 및 분석

가) VMOS에서 Linformer 기반 projection matrix의 dimension에 따른 성능 비교

[그림 4-3]은 Linformer가 적용된 VMOS에서 projection matrix dimension의 크기 변화가 전체 시스템 성능에 미치는 영향을 보여준다. 본



[그림 4-3] Linformer projection matrix dimension 크기별 IoU 및 실행 시간 비교

실험에서는 HQSAM은 Heavy HQSAM 모델로 고정한 상태에서, Linformer의 projection matrix dimension을 512, 256, 64로 점차 축소하며 각 설정에 대해 성능을 비교하였다. 평가에는 32개 카메라 영상에 대해 총 75프레임의 평균 IoU와 프레임당 평균 실행 시간이 사용되었다.

실험 결과, 모든 설정에서 IoU는 95% 이상으로 유지되어 높은 정확도를 보였다. Org. 설정(즉, Linformer 기반 low-rank projection matrix차원 축소를 적용하지 않은 경우)에서는 평균 IoU가 98.82%로 가장 높았으며, 프레임당 평균 실행 시간은 약 26.82초로 확인되었다. 반면, 차원을 축소하면 실행 시간이 크게 단축되었다. 구체적으로, dimension을 512로 축소했을 때 실행 시간은 21.78초로 감소하였으며, 256의 경우는 21.72초, 64의 경우 21.59초로 가장 짧은 시간을 기록하였다. 특히 dimension을 64로 설정했을 때 Org. 대비 약 19.5%의 실행 시간 감소를 달성하였다.

IoU는 dimension 축소에 따른 약간의 성능 저하가 관찰되었다. 가장 큰 차원 축소(64)에서도 IoU는 96.85%로, Org. 대비 약 1.99% 감소에 불과하였다. 이러한 결과는 Linformer 구조 내에서 projection matrix dimension을 축소하는 것이 계산량을 효과적으로 줄여 처리 속도를 개선하면서도 IoU 성능을 96% 이상으로 대부분 유지할 수 있음을 시사한다. 즉, VMOS의 GPM 내부 어텐션 연산에서 Linformer 기반 low-rank projection matrix 기법을 적용하는 것이 성능 최적화를 위한 효과적인 전략임을 확인하였다.

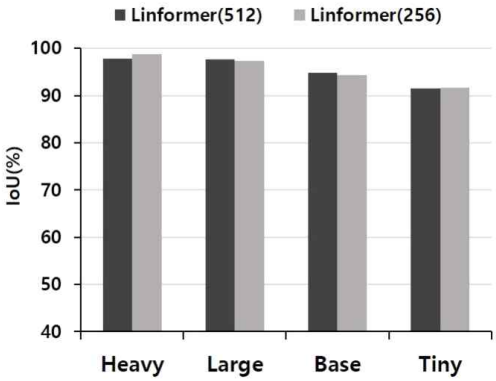
나) HQSAM의 경량화 수준별 성능 평가

[그림 4-4] HQSAM 모델에 따른 IoU 및 실행 시간 비교

본 실험은 VMOS의 GPM 내부 어텐션 연산에 Linformer 기반의 low-rank projection 기법을 적용하지 않은 Org. VMOS 구성을 고정한 상태에서, HQSAM의 경량화 수준(Heavy, Large, Base, Tiny)에 따른 성능 차이를 평가하였다. 성능 평가는 평균 IoU와 프레임당 평균 실행 시간을 기준으로 수행되었으며, 각 HQSAM 모델에 대한 정확도와 처리 속도를 비교, 분석하였다.

[그림 4-4]는 HQSAM 모델 크기에 따른 IoU와 평균 실행 시간 변화를 나타낸다. 실험 결과, Heavy HQSAM은 평균 IoU 98.82%로 가장 높은 정확도를 기록하였으나, 프레임당 평균 실행 시간이 26.82초로 가장 느린 처리 속도를 보였다. 반대로 가장 경량화된 Tiny HQSAM은 평균 7.12초의 실행 속도를 달성하여 Heavy 대비 73.47% 빠른 성능을 보였다. 그러나 이 경우 IoU는 91.69%로 감소하여 Heavy와 비교해 약 7.13% 정확도 손실이 발생하였다. 중간 크기 모델인 Large와 Base는 성능과 정확도의 균형을 보였다. Large HQSAM은 IoU 97.30%와 실행 시간 17.60초를 기록하여 정확도를 크게 손상시키지 않으면서 실행 속도를 약 34.4% 개선하였다. Base HQSAM은 IoU 94.33%, 실행 시간 14.00초로 확인되었다.

이러한 결과는 HQSAM의 경량화가 정확도 손실을 최소화하면서도 처리 속도를 크게 개선할 수 있음을 보여준다. 특히 Tiny HQSAM은 IoU가 90%



[그림 4-5] VMOS의 Linformer projection dimension (512, 256)에서 Tiny HQTrack을 사용한 HQSAM 모델 크기(Heavy, Large, Base, Tiny)별 IoU 비교

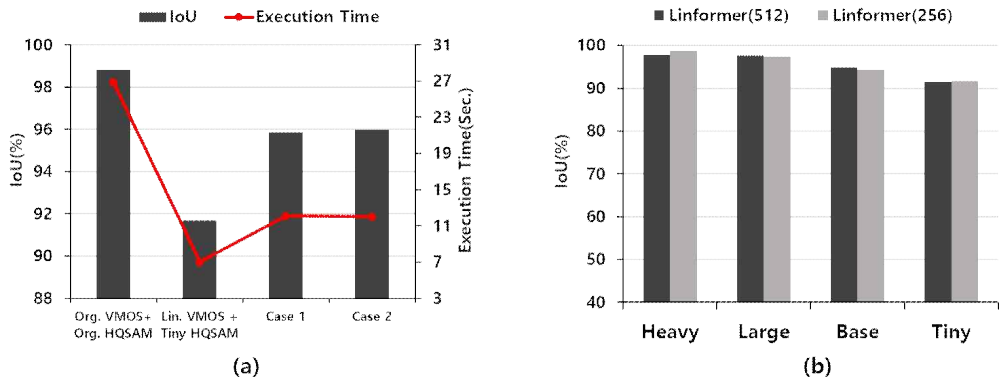
이상 유지되면서도 가장 빠른 속도를 보였기에, 본 연구에서는 실시간 처리 요구에 적합한 모델로 Tiny HQSAM을 채택하였다.

다) Linforme VMOS 적용 시 HQSAM 모델별 IoU 비교

본 실험은 Linformer VMOS를 사용하는 HQTrack에서, HQSAM 모델 크기(Heavy, Large, Base, Tiny)에 따른 IoU 성능 변화를 분석한 것이다. 특히 projection matrix dimension을 512와 256로 설정하여, 각 차원 축소 수준에 대해 HQSAM 모델별 성능을 비교하였다. [그림 4-5]는 이 두 설정에서 각 HQSAM 모델의 평균 IoU 결과를 나타낸다.

실험 결과, Heavy 및 Large 모델에서는 97.79%와 98.82%로 가장 높은 정확도를 보였으며, Base 모델과 Tiny 모델에서는 Linformer(512)는 94.87%, 91.56%, Linformer(256)는 94.32%와 91.69%를 기록하였다. 모든 설정에서 IoU가 일관되게 90% 이상을 유지하여, HQSAM 모델 경량화와 Linformer기반 어텐션 연산 최적화를 동시에 적용하더라도 세그멘테이션 정확도를 실용적인 수준 이상으로 유지할 수 있음을 확인하였다. 이는 실행 시간을 줄이면서 정확도 저하를 최소화는 데 있어 경량화 전략의 효과성을 입증한다.

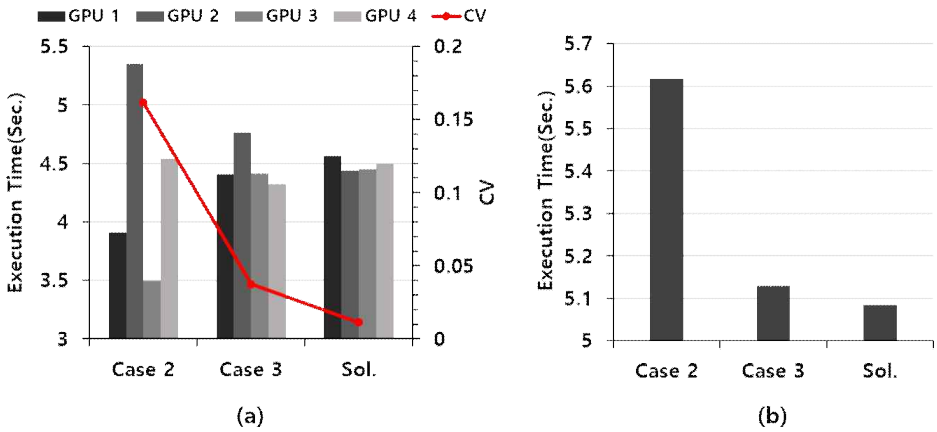
라) 적응형 동적 모델 조합 선택 기법과 코사인 유사도 기반 fine-grained 경량화의 성능 평가



[그림 4-6] (a) Case 1, Case 2에 대한 IoU 및 프레임당 실행 시간, (b) (a)의 Case 1과 Case 2를 확대한 그림

본 실험은 적응형 동적 모델 조합 선택 기법과 코사인 유사도 기반의 fine-grained 경량화 기법이 미치는 영향을 분석하였다. [그림 4-6]의 (a)는 단일 GPU 환경에서 Case 1, Case 2 에서의 평균 IoU와 프레임당 평균 실행 시간을 시각화한 결과를 보여준다. 실험 결과, 두 Case 모두 Org. VMOS + Org. HQSAM 모델을 전체 75프레임에 일관되게 사용한 경우보다 실행 시간이 크게 단축되었으며, Linformer VMOS + Tiny HQSAM을 전체 프레임에 사용하였을 때와 비교하여 IoU 성능이 유의미하게 향상되었다. [그림 4-6]의 (b)는 (a)에서 Case 1과 Case 2를 확대하여 나타낸 결과로, Case 1과 Case 2는 각각 95.84%와 95.96%의 평균 IoU를 기록하였다. 이는 Org. VMOS + Org. HQSAM 대비 단 2.86% 감소한 결과를 나타내며 Linformer VMOS + Tiny HQSAM 대비 약 4.53% 및 4.66%의 성능 향상을 보였다. 프레임당 평균 실행 시간은 각각 12.08초 및 12.00초로 측정되어, Org. VMOS + Org. HQSAM 대비 약 54.97% 및 55.27%의 시간 감소를 나타냈다.

이러한 결과는 객체의 움직임 특성에 따라 원본 모델과 경량화 모델을 조합하여 선택적으로 적용하는 것이, 정확도의 손실을 최소화하며 세그멘테이션 처리 효율을 향상시키는 데 효과적임을 보여준다. 특히 Case 2에서 확인된 바와 같이, 제안된 코사인 유사도 기반 fine-grained 경량화 방법은 효율적인 전략임이 입증되었다. 프레임 간 유사도가 높은 경우 어텐션 연산에서



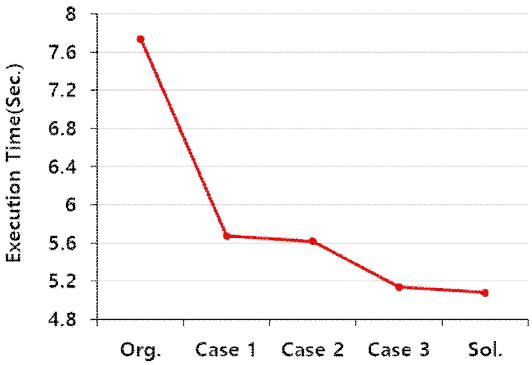
[그림 4-7] (a) Case 2, Case 3, and Sol.에 대한 CV와 각 GPU의 프레임당 평균 실행 시간 비교, (b) Case 2, Case 3, and Sol.에 대한 프레임당 평균 실행 시간

Linformer 기반 projection matrix의 차원을 더욱 축소하여 실행 시간을 추가로 단축하면서도 성능 저하를 최소화할 수 있는 효율적인 전략임을 보여준다.

마) 적응형 GPU 부하 재분배 기법을 통한 GPU 간 평균 실행 시간 및 성능 향상 평가

본 실험은 제안한 프레임워크가 GPU 간 처리 시간 불균형을 완화하고 프레임당 평균 실행 시간을 줄일 수 있음을 검증하였다. 각 실험은 모두 다중 GPU 환경에서 실행되었으며 Case 2의 경우에는 32개의 트래커를 4개의 GPU에 균등하게 할당한다. 이에 따라 모든 프레임에서 각 GPU는 고정적으로 8개의 트래커에 대한 세그멘테이션을 수행하는 방식이다. 지표로는 GPU 간 실행 시간의 상대적 분산을 나타내는 CV(Coefficient of Variation)를 사용하였다.

[그림 4-7]의 (a)는 각 실험 조건에서의 CV 값과 GPU별 프레임당 평균 실행 시간을 보여준다. Case 2의 경우, GPU 2 에는 연산량이 큰 모델 조합이, GPU 3 에는 연산량이 작은 모델 조합이 집중되면서 GPU별 실행 시간 불균형이 발생하였고, 이에 따라 CV 값이 0.16으로 가장 큰 불균형을 보였다. 반면, Case 3과 Sol.에서는 작업이 GPU 간에 균형적으로 분산되어 CV 값이 각각 0.03과 0.01로 감소하였다. 이를 통해 GPU 간 처리 불균형이 효



[그림 4-8] 다중 GPU 환경에서 각 Case 간 프레임당 평균 실행 시간 비교

과적으로 완화되었음을 확인할 수 있다.

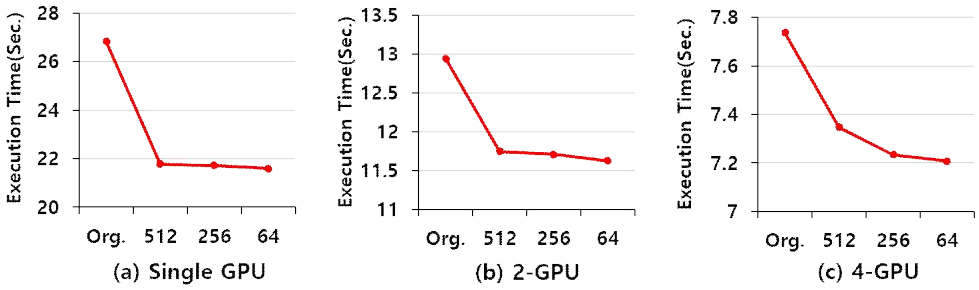
[그림 4-7]의 (b)는 각 조건에서의 프레임당 평균 실행 시간을 나타낸다. Case 2는 5.61초로 측정된 반면, Case 3과 Sol.은 각각 5.12초와 5.08초로, 약 8.73% 및 9.45%의 실행 시간 단축 효과를 보였다. 이러한 결과는 제안한 프레임워크가 GPU 간 실행 시간 편차를 최소화하는 동시에, 전체 시스템의 처리 효율을 향상시킴을 입증한다.

바) 제안한 세그멘테이션 프레임워크의 각 케이스별 성능 비교

제안한 프레임워크의 성능을 정량적으로 검증하기 위해 다중 GPU 환경에서 실험을 수행하였다. 우선, Org. 은 기존의 HQTrack의 모델 조합인 Org. VMOS와 Org. HQSAM을 모든 프레임에 동일하게 적용하였다. 이때 32개의 트래커가 4개의 GPU에 균등하게 분배되며, 각 GPU는 고정적으로 8개의 트래커에 대한 세그멘테이션 연산을 수행하였다.

실험 결과는 [그림 4-8]에 나타난 바와 같이, Org.의 프레임당 평균 실행 시간은 7.74초로 나타났다. 반면, Case 1, Case 2, Case 3은 각각 5.67초, 5.62초, 5.13초의 실행 시간을 기록하며 점진적인 개선을 달성하였다. 특히 Case 3에서는 Org. 대비 약 33.6%의 실행 시간 단축이 이루어졌음을 확인할 수 있다. 최종적으로 제안한 프레임워크(Sol.)는 평균 5.08초로 측정되었으며, 이는 Org. 대비 34.3%의 성능 향상을 달성하였다.

이러한 결과는 제안한 적응형 동적 모델 조합 선택 기법과 적응형 GPU



[그림 4-9] Single / 2-GPU / 4-GPU 환경에서 Linformer 기반 projection matrix dimension 변화에 따른 프레임당 평균 실행 시간 비교

부하 재분배 기법이 실행 시간을 효과적으로 줄일 수 있음을 실증적으로 보여준다. 결론적으로, 본 연구에서 제안한 솔루션은 멀티 카메라 이미지 환경에서의 효과적인 Segmentation 효율성을 크게 향상시킬 수 있음을 확인하였다.

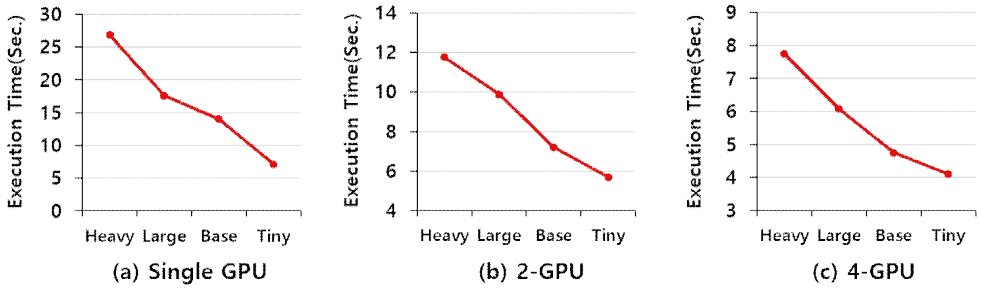
3) Trade-off Analysis

가) GPU 환경에 따른 VMOS 내 Linformer 기반 projection matrix dimension의 성능 평가

[그림 4-9]는 Linformer 기반 low-rank projection matrix dimension 변화에 따른 실행 시간의 변화를 GPU 환경별로 비교한 결과를 나타낸다. 실험은 Single GPU, GPU 2개, GPU 4개의 환경에서 각각 수행되었으며 HQSAM은 모든 실험 조건에서 동일하게 Org. HQSAM을 사용하였다. 결과적으로, Linformer의 projection matrix 차원이 작아질수록 실행 시간이 점진적으로 감소하는 경향을 보였다.

Single GPU 환경에서는 Linformer 기반의 projection을 적용하지 않은 Org.에서 26.83초였던 실행 시간이, 64로 설정했을 때 21.59초로 줄어들어 약 19.5%의 실행 시간 단축 효과가 나타났다. GPU 2개로 실험한 환경에서는 12.94초에서 11.62초로 약 10.2% 감소하였고, GPU 4개로 실험한 환경에서도 7.74초에서 7.2초로 약 7.0%의 실행 시간 단축이 확인되었다.

모든 실험 환경에서 일관되게 감소 추세가 확인되어 Linformer 기반 차원 축



[그림 4-10] Single / 2-GPU / 4-GPU 환경에서 HQSAM 모델에 따른 프레임당 평균 실행 시간 비교

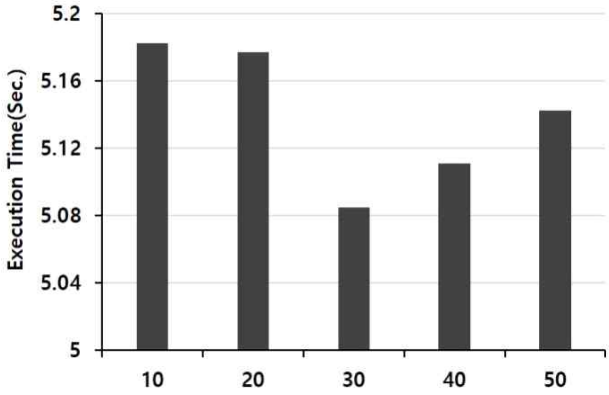
소가 GPU 환경 전반에서 모델의 연산 효율성을 개선하는 데 기여함을 입증하였다.

나) GPU 개수와 HQSAM 모델 종류에 따른 성능 평가

[그림 4-10]은 다양한 HQSAM 모델 구성(Heavy, Large, Base, Tiny)에 따른 프레임당 평균 실행 시간의 변화를 GPU 환경별로 비교한 결과를 보여준다. 실험은 Single GPU, GPU 2개, GPU 4개의 환경에서 각각 수행되었으며, VMOS 구성은 모든 조건에서 동일하게 Org. VMOS를 사용하였다.

실험 결과, HQSAM의 모델 크기에 따라 모든 GPU 환경에서 추론 시간이 일관되게 감소하는 경향이 관측되었다. Single GPU 환경에서는 Heavy HQSAM에서 프레임당 평균 실행 시간이 26.82초였던 반면, Tiny HQSAM에서는 7.12초로 약 73.47%의 실행 시간 단축이 이루어졌다. GPU 2개를 사용한 환경에서는 Heavy HQSAM의 평균 실행 시간이 약 11.74초였던 반면, Tiny HQSAM에서는 약 5.70초로, 약 51.4%의 실행 시간 단축이 이루어졌다. GPU 4개를 사용한 환경에서도 유사한 경향이 나타났으며, Heavy HQSAM에서는 7.74초, Tiny HQSAM에서는 4.10초로 측정되어 약 47.0%의 성능 개선이 확인되었다.

이러한 결과는 멀티 카메라 이미지 기반 세그멘테이션 환경에서, 단일 이미지 처리에 비해 훨씬 많은 이미지 처리가 요구되는 점을 고려할 때, 경량화된 HQSAM 모델이 연산자원의 효율적 활용 측면에서 더욱 적합하다는 가능성을 시사한다.



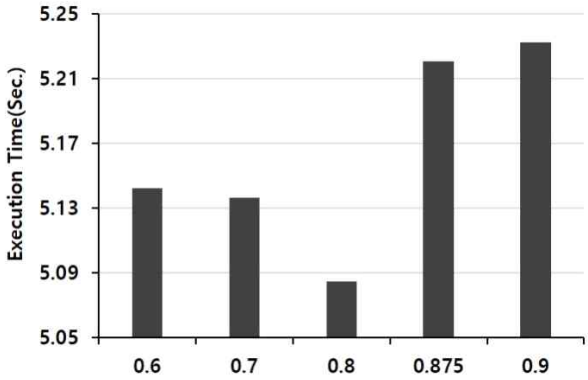
[그림 4-11] 시스템 메모리를 거치는 메모리 복사 주기 조정에 따른 실행 시간

다) △에 따른 적응형 GPU 부하 재분배 기법의 효과

[그림 4-11]은 제안한 멀티 카메라 이미지 세그멘테이션 프레임워크에서 시스템 메모리 기반 트래커 정보 메모리 복사 주기를 10, 20, 30, 40, 50프레임 단위로 변경했을 때의 평균 실행 시간 변화를 보여준다. 본 실험은 GPU 간 실행 시간 균형을 위한 시스템 메모리를 통한 메모리 복사 주기가 전체 시스템 성능에 미치는 영향을 분석하기 위해 수행되었으며, 동작 방식은 Sol. 방법과 동일하다.

실험 결과, 30프레임 주기에서 평균 실행 시간이 5.08초로 가장 낮게 나타나, 실험 조건 중 가장 효율적인 성능 균형을 보였다. 반면, 10프레임 및 20프레임과 같은 짧은 주기는 메모리 복사 횟수 증가로 인한 오버헤드 때문에 평균 실행 시간이 각각 5.18초, 5.17초로 소폭 증가하였다. 또한, 50프레임과 같은 긴 주기는 그사이에 발생한 부하 불균형을 제때 해소하지 못해 실행 시간 균형 효과가 감소하게 된다. 즉, 프레임 주기가 길어질수록 GPU 간 부하 불균형이 이미 심화된 상태에서 뒤늦게 조정이 이뤄지게 되며, 결과적으로 평균 실행 시간은 5.14초로 증가하며 성능 저하가 발생하였다.

결과적으로, 메모리 복사 주기가 지나치게 짧으면 시스템 메모리를 통한 복사 과정에서 오버헤드가 발생하고, 반대로 주기가 지나치게 길면 GPU 간 실행 시간 불균형이 심화하여 프레임당 평균 실행 시간이 증가한다. 따라서 적절한 메모리 복사 주기 설정이 중요하다. 본 실험에서는 30프레임 주기에



[그림 4-12] 코사인 유사도 임계값에 따라 변화하는 프레임당 평균 실행 시간 비교

서 가장 효율적인 성능을 보임을 확인하였다.

라) 코사인 유사도 임계값 변화가 프레임워크의 효율성(실행 시간)에 미치는 영향

[그림 4-12]는 제안한 멀티 카메라 이미지 세그멘테이션 프레임워크에서 코사인 유사도 임계값 변화에 따른 프레임당 평균 실행 시간을 보여준다. 본 실험은 코사인 유사도 임계값이 프레임워크의 효율성에 미치는 영향을 분석하고, 최적의 임계값을 설정하기 위한 목적으로 수행되었으며 동작 방식은 Sol.과 같다.

실험 결과, 임계값이 0.8일 때 프레임당 평균 실행 시간이 5.08초로 가장 낮게 측정되었다. 반대로 0.9일 때는 5.23초로 가장 높은 실행 시간이 기록되었다. 또한 0.6일 경우에는 실행 시간이 5.14초로 측정되었는데, 이는 0.8보다는 다소 길지만 0.9보다는 짧은 수준으로 나타났다. 이러한 결과는 임계값 설정이 지나치게 낮을 경우에도 효율성이 저하될 수 있음을 보여준다.

따라서 본 연구에서는 최적의 코사인 유사도 임계값을 0.8로 확인하였다. 이는 제안된 프레임워크가 코사인 유사도를 활용한 fine-grained 경량화 기법을 통해 Linformer 기반 projection matrix의 dimension을 동적으로 조절함으로써, 불필요한 연산을 줄이고 효율적인 세그멘테이션을 가능하게 함을 의미한다.

제 5 장 결론

본 논문에서는 객체 추적 및 세그멘테이션 과정에서 프레임당 처리해야 하는 이미지 수가 많은 다시점 영상 환경에서 발생하는 높은 연산 부담과 처리 지연 문제를 효과적으로 완화하기 위한 두 가지 프레임워크를 제안하였다. 제안된 프레임워크들은 연산 효율성을 극대화하는 동시에 정확도 손실을 최소화하고 프레임 단위 처리 속도를 향상시키는 것을 목표로 한다.

첫 번째로, 플렌옵틱 이미지 환경에서 프레임별 처리 속도를 효과적으로 향상시키기 위한 객체 추적 프레임워크를 제안하였다. 본 프레임워크는 프레임당 101장의 포컬 이미지를 모두 처리하던 기존 방식과 달리, 필수적인 포컬 레인지로 범위를 제한하여 7~11장의 이미지만을 처리하도록 설계함으로써, 솔루션 적용 전과 비교해 IoU 성능 저하를 거의 발생시키지 않으면서도 연산량을 크게 감소시켰다. 또한 GPU 자원을 지속적으로 활용하여 처리 지연을 최소화하기 위해, 제안한 시스템은 멀티 스레딩 기반의 멀티 스트림 병렬 처리 기법을 도입하여 전체 처리 파이프라인의 효율성을 극대화하였다.

두 번째로 멀티 카메라 이미지 세그멘테이션 프레임워크를 제안하였다. 우선, 각 트래커에 대해 세그멘테이션 마스크 간 객체의 Center Distance를 계산하고, 이를 기반으로 HQTrack의 VMOS와 HQSAM에서 원본 모델과 경량화된 모델 중 적절한 조합을 선택하여 활용한다. 또한, 프레임 간 코사인 유사도를 계산하여 Linformer 기반 projection matrix dimension을 동적으로 조절함으로써 시각적 변화가 적은 프레임에서는 projection matrix의 차원을 축소해 추론 속도를 높이고, 변화가 큰 프레임에서는 차원을 유지해 정보를 보존한다. 또한, 매 프레임마다 시스템 내부 GPU들의 하드웨어 연결 상태를 고려하여 GPU 간 데이터 이동을 최적으로 수행한다. 이를 통하여 경량화 모델과 원본 모델을 혼재해서 실행해도 각 GPU마다 프레임 바운더리 내에서 전체 segmentation 끝나는 시간을 비슷하게 유지하여 시스템 전체적인 처리 속도를 최대가 되게 한다. 4-GPU 환경에서 실험한 결과, Baseline HQTrack 대비 통해 IoU는 단 2.86% 하락했고 실행 시간은 34.3% 단축되었다.

참 고 문 헌

1. 국외문헌

Caelles, S., Maninis, K. K., Pont-Tuset, J., Leal-Taixé, L., Cremers, D., & Van Gool, L. (2017). One-shot video object segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition(CVPR), 221-230.

Chalfoun, J., Lund, S. P., Ling, C., Peskin, A., Pierce, L., Halter, M., Elliott, J., & Sarkar, S. (2024). Establishing a reference focal plane using convolutional neural networks and beads for brightfield imaging. Scientific Reports, 14(1), 7768.

Dai, A., & Nießner, M. (2018). 3dmv: Joint 3d-multi-view prediction for 3d semantic scene segmentation. In Proceedings of the European conference on computer vision(ECCV), 452-468.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In Proceedings of 9th International Conference on Learning Representations(ICLR), 3-7.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition(CVPR), 770-778.

Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531.

Javed, S., Danelljan, M., Khan, F. S., Khan, M. H., Felsberg, M., & Matas, J. (2022). Visual object tracking with discriminative filters and siamese networks: a survey and outlook. IEEE transactions on

pattern analysis and machine intelligence, 45(5), 6552-6574.

Ke, L., Ye, M., Danelljan, M., Tai, Y. W., Tang, C. K., & Yu, F. (2023). Segment anything in high quality. Advances in Neural Information Processing Systems(NeurIPS), 36, 29914-29934.

Kim, M., Kim, I., Yong, J., & Kim, H. (2023). Scheduling framework for accelerating multiple detection-free object trackers. Sensors, 23(7), 3432.

Li, B., Wu, W., Wang, Q., Zhang, F., Xing, J., & Yan, J. (2019). Siamrpn++: Evolution of siamese visual tracking with very deep networks. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 4282-4291.

Li, J., Xu, Y., Yang, Z., Yang, Y., & Zhuang, Y. (2023). ZJU ReLER Submission for EPIC-KITCHEN Challenge 2023: Semi-Supervised Video Object Segmentation. arXiv preprint arXiv:2307.02010.

Lin, T. Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017). Feature pyramid networks for object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition(CVPR), 2117-2125.

Manzoor, S., Sung, K. H., Zhang, Y., An, Y. C., & Kuc, T. Y. (2022). Qualitative analysis of single object and multi object tracking models. In 2022 22nd International Conference on Control, Automation and Systems(ICCAS), 1539-1545.

Oh, H. (2021). Single Object Tracking in Plenoptic Sequences via Similarity Estimation. Journal of the Institute of Electronics and Information Engineers, 58(2), 33-42. 10.5573/ieie.2021.58.2.33

Oh, S. W., Lee, J. Y., Xu, N., & Kim, S. J. (2019). Video object segmentation using space-time memory networks. In Proceedings of the IEEE/CVF international conference on computer vision, 9226-9235.

Ouaknine, A., Newson, A., Pérez, P., Tupin, F., & Rebut, J. (2021). Multi-view radar semantic segmentation. In Proceedings of the IEEE/CVF International Conference on Computer Vision, 15671-15680.

Park, J., Jeong, J., Choi, J., Kim, Y. G., & Ryu, E. (2024). Depth Prediction Transformer-Based Multi-View Video Depth Map Generation Technique for 6-DoF Immersive Video Synthesis. In Proceedings of the Korean Institute of Broadcast and Media Engineers(KIBME), 6, 713-716.

Pereira, F., da Silva, E. A., & Lafruit, G. (2018). Plenoptic imaging: Representation and processing. Academic Press Library in Signal Processing, 6, 75-111.

Plancher, B., & Kuindersma, S. (2018). A performance analysis of parallel differential dynamic programming on a gpu. In International Workshop on the Algorithmic Foundations of Robotics, 656-672.

Scagliola, A., Di Lena, F., Garuccio, A., D'Angelo, M., & Pepe, F. V. (2020). Correlation plenoptic imaging for microscopy applications. Physics Letters A, 384(19), 126472.

Schütze, H., Manning, C. D., & Raghavan, P. (2008). Introduction to information retrieval, 39, 234-265. Cambridge: Cambridge University Press.

Segura, A., Arnau, J. M., & González, A. (2019). SCU: a GPU stream compaction unit for graph processing. In Proceedings of the 46th international symposium on computer architecture, 424-435.

Son, H., Shin, M., Kim, J., Yun, K., Cheong, W., Lee, H., & Kang, S. (2022). Deep learning-based Multi-view Depth Estimation Methodology of Contents' Characteristics. In Proceedings of the Korean Institute of Broadcast and Media Engineers(KIBME), 6, 4-7.

Son, W., Jang, H., Bae, S., Park, S., Kim, J., & Kim, D. (2016).

Plenoptic Image Processing Technology Trends. Electronics and Telecommunications Research Institute, 31(4), 1-12.

Song, Z., Yu, J., Chen, Y. P. P., & Yang, W. (2022). Transformer tracking with cyclic shifting window attention. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition(CVPR), 8791-8800.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems(NeurIPS), 30.

Vora, J., Dutta, S., Jain, K., Karthik, S., & Gandhi, V. (2023). Bringing generalization to deep multi-view pedestrian detection. In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision(WACV),110-119.

Wang, L., Lu, H., Wang, Y., Feng, M., Wang, D., Yin, B., & Ruan, X. (2017). Learning to detect salient objects with image-level supervision. In Proceedings of the IEEE conference on computer vision and pattern recognition(CVPR), 136-145.

Wang, S., Li, B. Z., Khabsa, M., Fang, H., & Ma, H. (2020). Linformer: Self-attention with linear complexity. arXiv preprint arXiv:2006.04768.

Wang, W., Dai, J., Chen, Z., Huang, Z., Li, Z., Zhu, X., Hu, X., Lu, T., Lu, L., Li. H., Wang, X., & Qiao, Y. (2023). Internimage: Exploring large-scale vision foundation models with deformable convolutions. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition(CVPR), 14408-14419.

Wojke, N., Bewley, A., & Paulus, D. (2017). Simple online and realtime tracking with a deep association metric. In 2017 IEEE international conference on image processing(ICIP), 3645-3649.

Wu, Y., Lim, J., & Yang, M. H. (2013). Online object tracking: A benchmark. In Proceedings of the IEEE conference on computer vision

and pattern recognition(CVPR), 2411-2418.

Yang, Z., & Yang, Y. (2022). Decoupling features in hierarchical propagation for video object segmentation. Advances in Neural Information Processing Systems(NeurIPS), 35, 36324-36336.

Yong, J., Hwang, H., & Kim, M. (2025). Object segmentation and tracking in multi-view video. Journal of the Korea Institute of Information and Communication Engineering(JKIICE), 29(3), 358-364.

Yong, Y., Kang, J., & Oh, H. (2024). Detection-Free Object Tracking for Multiple Occluded Targets in Plenoptic Video. Electronics, 13(3), 590.

Zhu, J., Chen, Z., Hao, Z., Chang, S., Zhang, L., Wang, D., Lu, H., Luo, B., He, J., Lan, J., Chen, H., & Li, C. (2023). Tracking anything in high quality. arXiv preprint arXiv:2307.13974.

ABSTRACT

Framework for Multi-view Object Tracking and Segmentation

Yong, Ji-Hyeon

Major in Applied Artificial Intelligence

Dept. of Applied Artificial Intelligence

The Graduate School

Hansung University

Single-camera-based video suffers from limited object recognition and tracking performance in complex scenes due to restricted field of view and occlusion. Multi-view video can mitigate these limitations, but the increased number of images to process per frame leads to high computational complexity and the need to handle massive data volumes in real-time object tracking and segmentation. This paper proposes efficient object tracking and segmentation frameworks for two representative types of multi-view video: plenoptic imaging and multi-camera imaging. First, in the plenoptic imaging setting, we restructure existing 2D video-based object trackers to better align with the characteristics of plenoptic images. In addition, we introduce an image selection strategy that extracts only the essential focal plane images from the numerous ones available in each frame, and we maximize

computational efficiency by parallelizing the deep learning-based feature extraction module and preprocessing stages across a multi-core CPU and GPU environment. Second, in the multi-camera imaging setting, we propose an efficient segmentation framework that dynamically switches between lightweight and original models, and uses Video Multi-Object Segmenter with a low-rank projection matrix and a lightweight Mask Refiner. Furthermore, cosine similarity between consecutive frames is used to accurately determine the extent of motion or variation of target objects. This information enables adaptive adjustment of the lightweight level for the current frame, allowing fine-grained model selection. In a multi-GPU setting, the coexistence of lightweight and original models can lead to execution time imbalance across GPUs, causing frame-level latency. To mitigate this, the proposed framework optimally manages inter-GPU data transfers at each frame by considering the hardware connectivity of GPUs. As a result, the segmentation execution time per frame is balanced across GPUs, minimizing the overall average per-frame execution time. Experimental results demonstrate that the proposed framework maintains the IoU drop within 2.86% due to lightweight model usage, while achieving a 34.3% reduction in average per-frame execution time.

【Keywords】 Plenoptic, Thread pool, Multi-stream, Multi-view, low-rank approximation, Adaptive GPU load redistribution