제공해주신 문서들을 검토한 결과, 귀하께서는 Gen3 Lite 매니퓰레이터를 사용하여 실내 환경(책상 위)에서 일반 객체에 대한 자연어 기반 인식 및 파지 연구를 진행하고자 하는 것으로 이해했습니다. 두 가지 접근법(강화학습 기반과 VLM 기반)으로 연구를 진행하실 계획이신 것 같습니다.

다음은 귀하의 연구에 적합할 것으로 생각되는 5가지 영어 프로젝트/논문 제목 후보입니다:

1. "Real-time Vision-Language Model-based Object Recognition and Grasping System for Gen3 Lite Manipulator" (Gen3 Lite 매니퓰레이터를 위한 실시간 비전-언어 모델 기반 객체 인식 및 그래스핑 시스템)

2. "Zero-shot Object Grasping with Natural Language Commands: A Comparative Study of VLM and Reinforcement Learning Approaches" (자연어 명령을 통한 제로샷 객체 그래스핑: VLM과 강화학습 접근법의 비교 연구)

3. "Efficient General Object Manipulation using NanoOWL and NanoSAM with MoveIt2 Integration" (NanoOWL과 NanoSAM을 활용한 MoveIt2 통합 기반 효율적 일반 객체 조작)

4. "Reinforcement Learning vs. Vision-Language Models: A Comprehensive Analysis for Robotic Grasping with Gen3 Lite" (강화학습 vs. 비전-언어 모델: Gen3 Lite를 활용한 로봇 그래스핑을 위한 종합 분석)

5. "Language-Guided Robust Object Recognition and Manipulation System for Indoor Environments" (실내 환경을 위한 언어 기반 견고한 객체 인식 및 조작 시스템)

자연어 기반 실시간 객체 인식 및 그래스핑 시스템 개발 연구

1. 서론

1.1 연구 배경 및 필요성

로봇 기술의 급속한 발전에도 불구하고, 인간과 유사한 수준의 물체 인식 및 조작 능력을 구현하는 것은 여전히 중요한 과제로 남아있다. 특히 일반 가정이나 작업 환경에서 로봇이 자연어 명령을 이해하고 다양한 물체를 인식하여 파지할 수 있는 기술은 서비스 로봇, 산업용 로봇, 협동 로봇 등 다양한 분야에서 크게 요구되고 있다. 최근 컴퓨터 비전과 자연어 처리 기술의 발전으로 인해 이러한 과제를 해결하기 위한 새로운 접근법들이 등장하고 있다.

현재까지의 로봇 파지 시스템은 주로 특정 환경이나 사전에 학습된 물체에 대해서만 동작하는 경우가 많았다. 이러한 제한된 시스템은 새로운 환경이나 미지의 물체를 처리하는 데 어려움을 겪는다. 따라서 다양한 환경과 물체에 대응할 수 있는 유연한 파지 시스템의 개발이 필요하다. 특히 자연어 명령을 통해 로봇을 제어하는 방식은 전문 지식이 없는 일반 사용자도 로봇을 쉽게 활용할 수 있게 하여 로봇 기술의 접근성을 크게 향상시킬 수 있다.

최근 CLIP(Contrastive Language-Image Pre-training), SAM(Segment Anything Model)과 같은 비전-언어 모델(Vision-Language Models, VLMs)의 등장은 자연어와 시각 정보를 통합하여 처리하는 새로운 패러다임을 제시하고 있다. 이러한 모델들은 방대한 데이터셋에 대해 사전 학습되어 있어 제로샷(zero-shot) 또는 퓨샷(few-shot) 학습 능력을 보유하고 있으며, 이는 미지의 물체에 대한 인식 성능을 크게 향상시킬 수 있는 가능성을 제시한다. 또한 강화학습(Reinforcement Learning) 기반 접근법은 로봇이 환경과의 상호작용을 통해 최적의 파지 전략을 학습할 수 있는 가능성을 보여주고 있다.

본 연구에서는 이러한 최신 기술들을 활용하여 자연어 명령을 기반으로 일반 객체를 인식하고 파지할 수 있는 로봇 시스템을 개발하고자 한다. 특히 VLM 기반 접근법과 강화학습 기반 접근법을 비교 분석하여 각 방법론의 장단점을 파악하고, 최적의 시스템 구성을 도출하는 것을 목표로 한다. 이를 통해 실내 환경에서 다양한 일반 객체에 대해 실시간으로 동작할 수 있는 견고한 파지 시스템을 구현하고자 한다.

1.2 연구 목표 및 범위

본 연구의 주요 목표는 Gen3 Lite 로봇 팔을 활용하여 자연어 명령 기반의 일반 객체 인식 및 파지 시스템을 개발하는 것이다. 구체적인 목표는 다음과 같다:

1. **자연어 기반 객체 인식**: 사용자의 자연어 명령(예: "빨간 컵을 집어")에 따라 관련 객체를 정확하게 인식할 수 있는 시스템 개발
2. **실시간 성능 확보**: 최소 5FPS 이상의 처리 속도를 통해 실시간 성능 확보
3. **일반 객체 대응**: 사전에 학습되지 않은 다양한 일반 객체(unlabeled objects)에 대한 제로샷 인식 및 파지 가능
4. **정확한 위치 추정**: 객체의 3D 위치, 6D 포즈 또는 3D 바운딩 박스 정보를 정확하게 추정
5. **두 가지 접근법 비교**: VLM 기반 접근법과 강화학습 기반 접근법의 성능 비교 및 분석
6. **MoveIt2 통합**: ROS2 환경에서 MoveIt2와의 효과적인 통합을 통한 모션 계획 및 실행
7. **시스템 평가**: 다양한 실내 환경 조건(조명, 배경, 객체 배치 등)에서의 시스템 성능 및 견고성 평가

연구 범위는 다음과 같이 제한한다:

- **환경**: 실내 환경(주로 책상 위)으로 제한
- **로봇 하드웨어**: Gen3 Lite 로봇 팔 모델 사용
- **물체 유형**: 주로 강체(rigid body) 중심, 단순한 형태의 일상 물체에서 시작하여 점차 복잡한 객체로 확장
- **작업 유형**: 정적인 환경에서의 단일 객체 파지에 초점
- **모빌리티**: 현재 연구에서는 고정된 로봇 베이스 사용, 향후 모바일 베이스로 확장 가능성 탐색

1.3 접근 방법론

본 연구에서는 두 가지 주요 접근법을 병행하여 진행한다:

1. **VLM 기반 접근법**:
    - CLIP, SAM 등의 대규모 사전 학습된 비전-언어 모델을 활용
    - 자연어 명령을 처리하여 관련 객체 인식 및 분할
    - 인식 결과를 바탕으로 MoveIt2를 통한 파지 계획 및 실행
    - NanoOWL+NanoSAM 조합과 같은 경량화된 모델 활용 가능성 탐색
2. **강화학습 기반 접근법**:
    - 환경과의 상호작용을 통한 최적의 파지 전략 학습
    - SAC(Soft Actor-Critic), PPO(Proximal Policy Optimization) 등의 알고리즘 활용
    - 자연어 명령에서 추출한 정보를 상태 또는 보상 함수에 통합
    - 시뮬레이션 환경에서의 사전 학습 및 실제 환경으로의 전이(sim-to-real transfer)

두 접근법의 장단점을 분석하고, 가능한 경우 두 접근법의 장점을 결합한 하이브리드 방식도 탐색한다. 각 접근법의 성능은 다양한 실험 및 평가를 통해 검증하고, 최종적으로 가장 효과적인 시스템 구성을 도출한다.

1.4 보고서 구성

본 보고서의 구성은 다음과 같다:

- **1장 서론**: 연구 배경, 목표 및 접근 방법론 소개
- **2장 관련 연구**: 객체 인식, 파지 기술, VLM, 강화학습 관련 선행 연구 분석
- **3장 시스템 설계**: 전체 시스템 아키텍처 및 주요 구성 요소 설명
- **4장 VLM 기반 접근법**: CLIP, SAM 등을 활용한 객체 인식 및 파지 방법 상세 설명
- **5장 강화학습 기반 접근법**: 강화학습을 통한 파지 전략 학습 방법 상세 설명
- **6장 MoveIt2 통합**: 인식 결과와 MoveIt2 연동 방법 설명
- **7장 실험 및 평가**: 실험 환경, 평가 지표 및 결과 분석
- **8장 결론 및 향후 연구**: 연구 결과 요약 및 향후 연구 방향 제시

2. 관련 연구

2.1 비전-언어 모델 기반 객체 인식 기술

2.1.1 CLIP 및 파생 모델

CLIP(Contrastive Language-Image Pre-training)은 OpenAI에서 개발한 모델로, 이미지와 텍스트 간의 의미적 관계를 학습하기 위해 대규모 이미지-텍스트 쌍 데이터셋으로 사전 학습되었다. CLIP은 대조적 학습(contrastive learning) 방법을 사용하여 이미지와 그에 해당하는 텍스트 설명이 임베딩 공간에서 서로 가까워지도록 학습된다. 이 방식은 사전 학습 과정에서 명시적으로 보지 않은 클래스에 대해서도 일반화 능력을 갖추게 되어, 제로샷(zero-shot) 분류 성능이 매우 뛰어나다.

CLIP은 로봇 분야에서도 다양하게 활용되고 있는데, 예를 들어 Robotic-CLIP은 액션 데이터를 활용하여 CLIP 모델을 미세 조정함으로써 로봇의 인식 능력을 향상시키는 방법을 제시하였다. 또한 CLIP-RT는 자연어 감독을 통해 로봇 정책을 학습하는 VLA(Vision-Language-Action) 모델로, 사전 학습된 CLIP 모델을 로봇 학습 영역으로 확장하여 비전문가도 자연어만으로 로봇에게 새로운 조작 기술을 가르칠 수 있도록 한다.

CLIP 외에도 SigLIP, MetaCLIP, AltCLIP, RemoteCLIP, BioCLIP, MobileCLIP, BLIP, BLIPv2, ALBEF, FastViT 등 다양한 파생 모델들이 개발되어 왔다. 이러한 모델들은 각각 특정 도메인이나 태스크에 더 적합하도록 설계되거나 경량화를 통해 계산 효율성을 개선하였다. 특히 MobileCLIP과 같은 경량 모델은 제한된 컴퓨팅 자원을 가진 로봇 플랫폼에서 실시간 처리가 가능하도록 최적화되었다.

2.1.2 SAM 및 파생 모델

SAM(Segment Anything Model)은 Meta AI에서 개발한 이미지 분할(segmentation) 모델로, 사용자가 제공한 프롬프트(점, 박스, 마스크 등)에 따라 이미지 내 객체를 정밀하게 분할할 수 있다. SAM은 1100만 개의 이미지와 11억 개의 마스크로 구성된 대규모 데이터셋인 SA-1B에서 학습되어, 다양한 객체와 장면에 대한 뛰어난 일반화 능력을 갖추고 있다.

최근 SAM2가 발표되어 이미지뿐만 아니라 비디오 분할까지 지원하는 통합 아키텍처를 제공한다. SAM2는 스트리밍 메모리 메커니즘을 통해 비디오 프레임을 순차적으로 처리하면서 객체의 문맥 정보를 유지하고, 초당 약 44프레임의 실시간 성능을 제공한다.

SAM은 계산 비용이 크기 때문에 임베디드 환경에서의 활용이 제한적이다. 이를 해결하기 위해 여러 경량화 모델들이 개발되었다:

- **MobileSAM**: 지식 증류(knowledge distillation) 기법으로 훈련된 경량 모델로, 원본 SAM과 유사한 품질을 유지하면서 크기와 연산량을 크게 줄였다. MobileSAMv2는 더 나아가 객체 인식 프롬프트 샘플링을

도입하여 분할 속도를 크게 향상시켰다.

- **NanoSAM**: NVIDIA에서 개발한 모델로, Jetson Orin 플랫폼에서 실시간 추론이 가능하도록 최적화되었다. ResNet18 기반의 경량 이미지 인코더와 MobileSAM의 마스크 디코더를 사용하여 Jetson AGX Orin에서 약 125 FPS의 성능을 달성하였다.

- **Efficient SAM(ESAM)**: 다양한 연구팀에서 개발한 경량화 버전으로, FastSAM, Efficient-SAM, EdgeSAM 등이 있다. 이들은 각각 다른 방식으로 SAM의 성능을 유지하면서 계산 효율성을 개선하였다.

2.1.3 객체 검출 및 분할 통합 모델

최근에는 객체 검출(detection)과 분할(segmentation)을 통합한 모델들이 등장하여 더욱 완전한 장면 이해를 가능하게 하고 있다.

- **Grounding DINO**: 텍스트 프롬프트를 사용하여 이미지에서 객체의 바운딩 박스를 검출하는 모델이다. DINO(Detection with Transformers)와 CLIP과 같은 언어 모델의 장점을 결합하여 개방형 어휘(open-vocabulary) 객체 검출이 가능하다. COCO 제로샷에서 52.5 AP를 달성하였으며, 다양한 도메인에서 우수한 일반화 성능을 보인다.

- **NanoOWL**: NVIDIA에서 개발한 모델로, OWL-ViT(Open-World Localization with Vision Transformers)를 Jetson 플랫폼에서 실시간으로 동작하도록 최적화하였다. 텍스트 프롬프트를 통해 객체를 검출할 수 있으며, Jetson AGX Orin에서 약 95 FPS의 성능을 보인다.

- **Grounded SAM**: Grounding DINO와 SAM을 결합한 파이프라인으로, 텍스트 프롬프트에 따라 객체를 검출하고 이를 SAM에 전달하여 정밀한 분할 마스크를 생성한다. Grounded SAM 2는 이를 비디오 도메인으로 확장하였으며, SAM2의 기능을 활용하여 비디오에서 객체를 추적할 수 있다.

- **Semantic-SAM**: 객체 분할과 의미 이해를 결합한 모델로, 클릭 한 번으로 객체를 분할하고 동시에 의미적 레이블을 제공한다. 다양한 수준의 세분성(granularity)을 지원하여 객체 수준부터 부품 수준까지 분할이 가능하다.

이러한 통합 모델들은 각각의 장점을 결합하여 보다 완전한 시각적 이해를 제공하지만, 계산 복잡성이 증가하는 단점이 있다. 따라서 실시간 성능이 요구되는 로봇 응용에서는 경량화 및 최적화 기술이 필수적이다.

2.2 로봇 파지 기술

2.2.1 전통적 파지 계획 방법

전통적인 로봇 파지 계획 방법은 주로 기하학적 접근법과 분석적 접근법으로 나눌 수 있다. 기하학적 접근법은 객체의 형상을 분석하여 안정적인 파지 지점을 찾는 방식이며, 분석적 접근법은 물리 모델을 활용하여 접촉점과 파지력을 최적화하는 방식이다.

기하학적 접근법은 객체의 3D 모델이나 점군 데이터에서 특징을 추출하여 파지 후보를 생성한다. 예를 들어, 객체의 주요 축을 찾아 그 방향으로 그리퍼를 정렬하거나, 객체의 기하학적 중심을 기반으로 파지 위치를 결정하는 방법 등이 있다. 이러한 방법은 계산 효율성이 높고 직관적이지만, 복잡한 형상의 객체나 부분적으로 가려진 객체에 대한 파지 성능이 제한적일 수 있다.

분석적 접근법은 접촉 역학(contact mechanics)과 힘 폐쇄(force closure) 개념을 기반으로 파지 안정성을 평가한다. Ferrari-Canny 메트릭과 같은 방법을 사용하여 파지 품질을 정량화하고, 최적의 파지 구성을 찾는다. 이러

한 방법은 이론적으로 견고하지만, 실제 환경의 불확실성을 완전히 고려하기 어렵고 계산 비용이 높은 단점이 있다.

2.2.2 데이터 기반 파지 방법

최근에는 딥러닝과 같은 데이터 기반 방법을 활용한 파지 기술이 크게 발전하였다. 이러한 방법은 대량의 데이터를 학습하여 다양한 객체와 환경에 적응할 수 있는 파지 모델을 구축한다.

GG-CNN(Generative Grasping Convolutional Neural Networks)은 RGBD 이미지로부터 직접 파지 포즈와 품질을 예측하는 네트워크로, 실시간으로 파지 지점을 생성할 수 있다. 이 네트워크는 각 픽셀 위치에서의 파지 품질, 각도, 그리퍼 너비를 동시에 예측하여 효율적인 파지 계획이 가능하다.

Grasping Rectangle 표현 방식은 2D 이미지에서 5차원 벡터(중심점 x, y, 방향 θ, 너비 w, 높이 h)로 파지를 표현하는 방법으로, 객체의 형태와 방향을 효과적으로 고려할 수 있다. 이를 기반으로 한 다양한 딥러닝 모델들이 개발되어 높은 성공률을 보이고 있다.

Affordance 기반 파지는 객체의 기능적 특성을 고려하여 파지 지점을 선택하는 방법이다. 예를 들어, 컵은 손잡이를 잡는 것이 가장 적절한 파지 방식이다. 이러한 방법은 객체의 의미적 이해를 필요로 하며, CLIP과 같은 비전-언어 모델을 활용하여 객체의 기능적 부분을 식별하는 연구가 진행되고 있다.

2.2.3 강화학습 기반 파지 방법

강화학습(Reinforcement Learning, RL)은 에이전트가 환경과의 상호작용을 통해 최적의 정책을 학습하는 방법으로, 로봇 파지 작업에도 널리 적용되고 있다. 강화학습의 주요 장점은 명시적인 프로그래밍 없이도 복잡한 환경에서 적응적인 파지 전략을 학습할 수 있다는 점이다.

SAC(Soft Actor-Critic)와 같은 off-policy 알고리즘은 탐색(exploration)과 활용(exploitation) 사이의 균형을 유지하면서 샘플 효율적인 학습이 가능하다. 이는 연속적인 행동 공간(continuous action space)을 가진 로봇 제어 작업에 특히 적합하다. SAC는 최대 엔트로피 강화학습 프레임워크를 사용하여 다양한 행동을 탐색하면서도 높은 보상을 얻는 정책을 학습한다.

PPO(Proximal Policy Optimization)는 on-policy 알고리즘으로, 정책의 안정적인 업데이트를 보장하면서도 성능을 지속적으로 향상시키는 특징을 가진다. 이 알고리즘은 구현이 간단하고 하이퍼파라미터 튜닝이 상대적으로 용이하여 실제 로봇 시스템에 적용하기 적합하다.

강화학습을 로봇 파지에 적용할 때의 주요 도전 과제 중 하나는 **시뮬레이션-실제 간 격차(sim-to-real gap)**이다. 이를 해결하기 위해 다음과 같은 방법들이 사용된다:

- **Domain Randomization**: 시뮬레이션 환경의 다양한 파라미터(물체 속성, 조명, 물리 특성 등)를 무작위화하여 모델이 다양한 조건에 적응할 수 있도록 한다.
- **System Identification**: 실제 로봇 시스템의 특성을 정밀하게 측정하고 시뮬레이션에 반영하여 격차를 줄인다.
- **Imitation Learning**: 인간 시연이나 전문가 정책을 모방하여 초기 정책을 학습한 후 강화학습으로 개선한다.

또한 강화학습에 **언어 이해 능력을 통합**하는 연구도 진행되고 있다. 자연어 명령을 강화학습의 목표나 보상 함수에 반영하여 사용자의 의도에 따라 적응적으로 파지 전략을 학습할 수 있다. 이러한 방법은 CLIP과 같은 비전-언어 모델의 표현력과 강화학습의 적응적 특성을 결합하여 더욱 유연한 파지 시스템을 구현할 수 있다.

2.3 비전-언어-액션 모델

최근에는 비전, 언어, 그리고 로봇 행동을 통합적으로 처리하는 비전-언어-액션(Vision-Language-Action, VLA) 모델이 큰 주목을 받고 있다. 이러한 모델들은 시각적 입력과 자연어 명령을 함께 처리하여 로봇의 행동을 직접 생성하는 end-to-end 학습 방식을 채택한다.

2.3.1 VLA 모델의 기본 원리

VLA 모델은 크게 세 가지 주요 구성 요소로 이루어진다:

1. **비전 인코더**: 이미지나 비디오에서 시각적 특징을 추출
2. **언어 인코더**: 자연어 명령에서 텍스트 특징을 추출
3. **액션 디코더**: 비전 및 언어 특징을 통합하여 로봇 행동 생성

이러한 모델들은 일반적으로 대규모 인터넷 데이터 또는 시뮬레이션 데이터로 사전 학습된 후, 실제 로봇 태스크에 대해 미세 조정(fine-tuning)된다. 이 과정에서 비전-언어 표현 학습과 로봇 행동 생성이 통합적으로 이루어진다.

2.3.2 주요 VLA 모델

RT-2(Robotics Transformer 2)는 Google DeepMind에서 개발한 모델로, 비전-언어 기초 모델에 로봇 행동 생성 능력을 추가하였다. RT-2는 실제 로봇 데이터와 웹에서 수집한 이미지-텍스트 데이터를 함께 훈련하여, 다양한 로봇 태스크에서 뛰어난 일반화 능력을 보여주었다.

CLIP-RT는 OpenAI의 CLIP을 로봇 학습으로 확장한 모델로, 자연어 감독을 통해 로봇 정책을 학습한다. 이미지와 언어 명령이 주어졌을 때 자연어로 표현된 로봇 동작을 예측하는 대조적 모방 학습(contrastive imitation learning) 방식을 사용한다. CLIP-RT는 매개변수가 7배 적으면서도 최첨단 VLA 모델보다 성능이 뛰어나다는 특징이 있다.

OpenVLA는 로봇 조작을 위한 오픈소스 VLA 모델로, 비전 인코더(SigLIP 및 DinoV2)와 언어 모델(Llama 2)을 결합하였다. OpenVLA는 Open X-Embodiment 데이터셋의 970K 로봇 조작 에피소드로 훈련되었으며, Jetson AGX Orin에서도 최적화된 미세 조정(OFT)을 통해 성능 향상이 가능하다.

DexGraspVLA는 사전 학습된 VLM을 고수준 작업 계획자로 활용하고, diffusion 기반 정책을 저수준 행동 제어기로 사용하는 계층적 프레임워크이다. 이 모델은 다양한 시각적 및 언어적 입력을 도메인 불변 표현으로 변환하여 모방 학습의 효율성을 높이고, 보이지 않은 다양한 객체에 대해서도 높은 파지 성공률을 보인다.

2.3.3 VLA 모델의 장단점

장점:

- 인식, 작업 이해, 제어를 단일 모델에서 통합적으로 처리하여 시스템 복잡도 감소
- 자연어를 통한 직관적인 로봇 제어 인터페이스 제공
- 대규모 사전 학습을 통한 뛰어난 일반화 능력

단점:

- 대규모 모델로 인한 높은 계산 요구량과 느린 추론 속도
- 로봇 특화 데이터의 부족으로 인한 학습의 어려움
- 장기 계획이 필요한 복잡한 작업 처리의 한계
- 모델 내부 작동 방식의 해석이 어려워 안전성 보장의 어려움

현재 VLA 모델들은 대부분 연구 단계에 있으며, 실제 로봇 시스템에 적용하기 위해서는 모델 경량화, 추론 최적화, 안전성 보장 등의 과제를 해결해야 한다. 하지만 비전, 언어, 행동의 통합적 처리는 로봇 지능을 한 단계 발전시킬 수 있는 유망한 방향으로 평가된다.

2.4 MoveIt2 및 ROS2 통합

ROS2(Robot Operating System 2)는 로봇 응용 프로그램 개발을 위한 오픈소스 미들웨어 플랫폼이며, MoveIt2는 ROS2 환경에서 로봇 매니퓰레이션을 위한 주요 프레임워크이다. 객체 인식 및 파지 시스템을 구현하기 위해서는 이러한 도구들과의 효과적인 통합이 필수적이다.

2.4.1 MoveIt2 개요

MoveIt2는 운동학, 동역학, 충돌 검사, 모션 계획, 궤적 처리 등 로봇 매니퓰레이션에 필요한 다양한 기능을 제공하는 통합 프레임워크이다. 주요 기능으로는 다음과 같은 것들이 있다:

- **모션 계획(Motion Planning)**: 로봇의 현재 상태에서 목표 상태까지의 충돌 없는 경로를 계획
- **충돌 감지(Collision Detection)**: 로봇과 환경 간의 충돌을 감지하고 회피
- **역기구학(Inverse Kinematics)**: 엔드 이펙터의 목표 위치에 대한 로봇 조인트 값을 계산
- **파지 계획(Grasp Planning)**: 객체 파지를 위한 최적의 접근 경로와 그리퍼 구성을 계획

MoveIt2는 플러그인 기반 아키텍처를 채택하여 다양한 알고리즘을 유연하게 통합할 수 있다. 예를 들어, OMPL(Open Motion Planning Library), CHOMP(Covariant Hamiltonian Optimization for Motion Planning), TrajOpt 등 다양한 모션 계획 알고리즘을 플러그인으로 사용할 수 있다.

2.4.2 객체 인식 결과의 MoveIt2 통합

객체 인식 시스템에서 얻은 결과를 MoveIt2와 통합하기 위한 여러 방법이 있다:

Planning Scene 업데이트:

- 인식된 객체를 collision object로 추가하여 모션 계획 시 충돌을 방지
- 객체의 3D 모델이나 단순화된 기하학적 표현(상자, 실린더 등)을 사용
- `moveit_msgs::CollisionObject` 메시지를 통해 객체 정보를 전달

파지 포즈 생성:

- 객체의 위치, 방향, 형태 정보를 바탕으로 가능한 파지 포즈 후보를 생성
- `moveit_msgs::Grasp` 메시지를 사용하여 파지 정보를 표현
- 파지 전/후의 그리퍼 상태, 접근/후퇴 방향, 허용 오차 등을 지정

Grasp Generator Plugin:

- 외부에서 개발된 파지 계획 알고리즘(GG-CNN, Grasp Pose Detection 등)을 MoveIt2의 플러그인으로 통합
- 특정 객체 유형이나 태스크에 최적화된 파지 전략을 쉽게 적용 가능

Perception Pipeline:

- MoveIt2의 Perception Pipeline을 활용하여 센서 데이터(RGBD 이미지 등)를 처리하고 인식된 객체 정보를 Planning Scene에 반영
- 실시간으로 변화하는 환경에 대응하기 위한 지속적인 환경 인식 가능

2.4.3 ROS2 통신 메커니즘

ROS2는 다양한 통신 메커니즘을 제공하여 객체 인식 모듈과 로봇 제어 모듈 간의 효과적인 데이터 교환을 지원한다:

토픽(Topics):

- 퍼블리셔-서브스크라이버 패턴을 기반으로 한 비동기 통신 방식
- 지속적인 센서 데이터 스트림이나 인식 결과를 전달하는 데 적합
- QoS(Quality of Service) 설정을 통해 통신의 신뢰성, 지연 시간, 대역폭 등을 제어 가능

서비스(Services):

- 요청-응답 패턴을 기반으로 한 동기 통신 방식
- 객체 인식이나 파지 포즈 계산과 같은 단발성 작업에 적합
- 클라이언트가 서비스 호출 결과를 기다려야 하므로 실시간 시스템에서는 주의 필요

액션(Actions):

- 장기 실행 태스크를 위한 비동기 통신 방식으로, 중간 피드백과 취소 기능을 제공
- 파지 동작과 같이 시간이 오래 걸리는 작업에 적합
- 태스크 상태를 모니터링하고 필요시 중단할 수 있는 유연성 제공

파라미터(Parameters):

- 시스템 구성과 동작을 제어하기 위한 키-값 쌍 저장소
- 알고리즘 설정, 모델 경로, 성능 임계값 등을 관리하는 데 사용
- 런타임 중에 동적으로 변경 가능하여 시스템 조정이 용이

이러한 통신 메커니즘을 효과적으로 활용하여 객체 인식 모듈과 로봇 제어 모듈 간의 원활한 상호작용을 구현할 수 있다. 특히 실시간성이 요구되는 파지 시스템에서는 통신 지연과 오버헤드를 최소화하기 위한 메시지 설계와 QoS 설정이 중요하다.

3. 시스템 설계

3.1 시스템 요구사항 및 제약조건

본 연구에서 개발하고자 하는 자연어 기반 객체 인식 및 그래스핑 시스템의 요구사항과 제약조건을 명확히 정의하여 시스템 설계의 기준을 수립한다.

3.1.1 기능적 요구사항

1. **자연어 명령 처리**:
    - 사용자의 자연어 명령을 입력으로 받아 처리할 수 있어야 함
    - 다양한 형태의 명령(예: "빨간 컵을 집어", "왼쪽에 있는 상자를 들어")을 이해할 수 있어야 함
    - 명령에 담긴 목표 객체와 관련된 속성(색상, 크기, 형태, 위치 등)을 추출할 수 있어야 함
2. **객체 인식 및 위치 추정**:
    - 명령에 해당하는 객체를 카메라 영상에서 식별할 수 있어야 함
    - 사전에 학습되지 않은 객체(unlabeled objects)도 명령에 따라 인식할 수 있어야 함

- 인식된 객체의 3D 위치 또는 6D 포즈를 정확하게 추정할 수 있어야 함
- 다중 객체가 존재할 때 명령에 가장 적합한 객체를 선택할 수 있어야 함

3. **파지 계획 및 실행**:

- 인식된 객체의 형태와 위치에 적합한 파지 전략을 계획할 수 있어야 함
- 충돌 없는 접근 경로를 생성하고 실행할 수 있어야 함
- 객체의 특성(깨지기 쉬움, 무거움 등)에 맞게 파지력을 조절할 수 있어야 함
- 파지 실패 시 재시도 또는 대안 전략을 수행할 수 있어야 함

4. **피드백 및 상호작용**:

- 시스템 상태와 인식 결과를 사용자에게 시각적으로 표시할 수 있어야 함
- 명령이 모호한 경우 사용자에게 추가 정보를 요청할 수 있어야 함
- 작업 진행 상태 및 완료 여부를 알릴 수 있어야 함

3.1.2 성능 요구사항

1. **실시간 처리**:

- 전체 파이프라인의 처리 속도는 최소 5FPS 이상이어야 함
- 명령 입력에서 로봇 동작 시작까지의 지연 시간이 2초 이내여야 함
- 객체 인식 및 위치 추정의 정확도를 유지하면서 실시간성을 확보해야 함

2. **인식 정확도**:

- 다양한 조명 조건에서도 안정적인 객체 인식 성능을 제공해야 함
- 부분적으로 가려진 객체도 인식할 수 있어야 함
- 객체 위치 추정의 평균 오차가 1cm 이내여야 함

3. **파지 성공률**:

- 단순한 형태의 객체에 대해 90% 이상의 파지 성공률을 달성해야 함
- 복잡한 형태의 객체에 대해 70% 이상의 파지 성공률을 달성해야 함
- 다양한 크기와 재질의 객체에 대해 적응적으로 파지할 수 있어야 함

3.1.3 제약조건

1. **하드웨어 제약**:

- Gen3 Lite 로봇 팔 모델 사용
- 고정된 베이스 사용 (현재 단계에서는 모바일 베이스 미사용)
- RGBD 카메라를 통한 객체 인식 및 위치 추정
- 제한된 계산 자원 (온보드 컴퓨터 또는 Jetson AGX Orin 사용)

2. **환경 제약**:

- 실내 환경(특히 책상 위)으로 한정
- 정적인 환경에서의 작업 (물체와 로봇 베이스가 움직이지 않음)
- 적절한 조명 조건 (극단적인 조명 상황은 제외)

3. **소프트웨어 제약**:

- ROS2 Humble 및 MoveIt2 기반 시스템 구현
- 오픈소스 컴포넌트 활용 (소스 코드 수정 및 확장 가능)
- 실시간 성능을 위한 최적화 필요

4. **객체 제약**:

- 주로 강체(rigid body) 대상
- 극단적으로 반사가 심하거나 투명한 객체는 제외
- 매우 작은 객체(1cm 이하)는 제외
- 매우 무거운 객체(로봇 팔의 페이로드 초과)는 제외

3.2 시스템 아키텍처

본 시스템의 전체 아키텍처는 모듈화된 설계를 채택하여 유연성과 확장성을 확보하였다. 각 모듈은 특정 기능을 담당하며, ROS2의 통신 메커니즘을 통해 상호작용한다.

3.2.1 전체 시스템 구성

시스템의 전체 구성은 다음과 같은 핵심 모듈로 이루어진다:

1. **자연어 처리 모듈**:

- 사용자의 자연어 명령을 입력받고 해석
- 명령에서 객체 특성(종류, 색상, 위치 등) 추출
- 명령의 의도 파악 및 작업 계획 생성

2. **객체 인식 모듈**:

- RGBD 카메라로부터 이미지 및 깊이 데이터 획득
- VLM 또는 강화학습 기반의 객체 검출 및 분할
- 인식된 객체의 3D 위치 및 포즈 추정

3. **파지 계획 모듈**:

- 인식된 객체의 특성에 따른 최적 파지 전략 선택
- 목표 객체에 접근하기 위한 충돌 없는 경로 계획
- 파지 동작 시퀀스 생성

4. **로봇 제어 모듈**:

- 계획된 경로를 따라 로봇 팔 제어
- 그리퍼 동작 제어 (개폐 및 파지력 조절)
- 센서 피드백 기반 실시간 제어 조정

5. **시각화 및 사용자 인터페이스 모듈**:

- 시스템 상태 및 인식 결과 시각화
- 사용자 명령 입력 인터페이스 제공
- 작업 진행 상태 및 결과 알림

3.2.2 데이터 흐름

시스템 내의 데이터 흐름은 다음과 같은 순서로 이루어진다:

1. 사용자가 자연어 명령을 입력하면, 자연어 처리 모듈이 이를 해석하여 객체 특성과 작업 정보를 추출한다.
2. 추출된 객체 특성은 객체 인식 모듈로 전달되어, RGBD 카메라 데이터에서 해당 객체를 검색한다.
3. 객체 인식 모듈은 인식된 객체의 위치 및 포즈 정보를 파지 계획 모듈로 전달한다.
4. 파지 계획 모듈은 객체 정보를 기반으로 최적의 파지 전략과 접근 경로를 계획한다.
5. 계획된 동작은 로봇 제어 모듈로 전달되어 실제 로봇 팔 움직임으로 실행된다.
6. 전체 과정의 상태와 결과는 시각화 및 사용자 인터페이스 모듈을 통해 사용자에게 제공된다.

3.2.3 ROS2 노드 구성

전체 시스템은 다음과 같은 ROS2 노드들로 구현된다:

1. **command_interpreter_node**:
    - 자연어 명령을 해석하고 작업 계획을 생성
    - 구독 토픽: `/speech_input`, `/text_input`
    - 발행 토픽: `/task_specification`
2. **object_detection_node**:
    - VLM 기반 객체 검출 및 분할 수행
    - 구독 토픽: `/camera/rgb/image_raw`, `/camera/depth/image_raw`, `/task_specification`
    - 발행 토픽: `/detected_objects`, `/object_masks`
3. **pose_estimation_node**:
    - 인식된 객체의 3D 위치 및 포즈 추정
    - 구독 토픽: `/detected_objects`, `/object_masks`, `/camera/depth/image_raw`
    - 발행 토픽: `/object_poses`
4. **grasp_planning_node**:
    - 객체 특성에 따른 최적 파지 전략 선택 및 경로 계획
    - 구독 토픽: `/object_poses`, `/task_specification`
    - 발행 토픽: `/grasp_poses`, `/motion_plan`
5. **robot_control_node**:
    - MoveIt2 인터페이스를 통한 로봇 제어
    - 구독 토픽: `/motion_plan`, `/grasp_poses`
    - 발행 토픽: `/robot_status`, `/grasp_result`
6. **visualization_node**:
    - 시스템 상태 및 인식 결과 시각화
    - 구독 토픽: `/detected_objects`, `/object_masks`, `/object_poses`, `/grasp_poses`, `/robot_status`
    - 발행 토픽: `/visualization_marker_array`

3.2.4 접근 방법별 시스템 구성

본 연구에서는 VLM 기반 접근법과 강화학습 기반 접근법 두 가지를 병행하여 구현하고 비교 분석한다.

VLM 기반 접근법 구성:

- **object_detection_node**: NanoOWL 또는 Grounding DINO를 사용하여 텍스트 프롬프트로 객체 검출
- **segmentation_node**: NanoSAM 또는 MobileSAMv2를 사용하여 객체 분할
- **grasp_planning_node**: 분할된 객체의 기하학적 특성에 기반한 파지 포즈 계산

강화학습 기반 접근법 구성:

- **policy_network_node**: 사전 학습된 강화학습 정책 네트워크 실행
- **feature_extraction_node**: RGBD 이미지에서 상태 표현 추출
- **action_execution_node**: 정책 네트워크의 출력을 로봇 액션으로 변환 및 실행

두 접근법의 비교를 위해 동일한 입력과 평가 지표를 사용하며, 필요에 따라 하이브리드 접근법(VLM으로 객체 식별 후 RL로 파지 실행 등)도 탐색할 수 있다.

3.3 하드웨어 구성 및 센서 설정

3.3.1 로봇 하드웨어

본 연구에서 사용하는 Gen3 Lite 로봇 팔은 협동 로봇으로, 가볍고 콤팩트한 설계로 연구 및 교육 환경에 적합한 모델이다. 주요 특징은 다음과 같다:

- **자유도(DoF)**: 6자유도
- **페이로드**: 약 0.5kg
- **작업 반경**: 약 700mm
- **반복 정밀도**: ±0.1mm
- **그리퍼**: 2지 그리퍼 (선택적으로 다른 엔드 이펙터로 교체 가능)
- **중량**: 약 5kg
- **제어 인터페이스**: ROS2 지원

Gen3 Lite는 가벼운 무게와 저전력 요구사항으로 인해 이동이 용이하고 설치가 간단하다는 장점이 있다. 그러나 페이로드가 제한적이기 때문에 무거운 객체를 다루는 데에는 제약이 있다.

3.3.2 센서 구성

주요 센서 구성은 다음과 같다:

1. **RGBD 카메라**:
    - 모델: Intel RealSense D435 또는 유사한 RGBD 카메라
    - 해상도: RGB 1920x1080, 깊이 1280x720
    - 프레임 레이트: 30fps
    - 시야각(FOV): 약 87° x 58°
    - 작동 범위: 0.2m ~ 10m
    - 설치 위치: 로봇 베이스 주변 또는 그리퍼 근처
2. **그리퍼 센서**:

- 그리퍼에 내장된 힘 감지 센서
- 파지력 측정 및 제어에 활용
- 접촉 감지 및 슬립 방지 기능 지원

3. **추가 센서 (선택적)**:

- 마이크: 음성 명령 수신용
- 추가 카메라: 다른 각도에서의 관측을 위한 보조 카메라
- 힘/토크 센서: 그리퍼에 가해지는 힘과 토크를 정밀하게 측정

3.3.3 센서 배치 및 캘리브레이션

센서의 효과적인 활용을 위해 다음과 같은 배치 및 캘리브레이션 절차가 필요하다:

1. **카메라 배치**:

- 작업 공간(책상 위)을 충분히 관측할 수 있는 위치에 배치
- 로봇 팔의 움직임에 의한 카메라 시야 가림을 최소화
- 안정적인 마운팅 구조로 진동 및 흔들림 방지

2. **카메라-로봇 캘리브레이션**:

- 카메라 좌표계와 로봇 베이스 좌표계 간의 변환 행렬 계산
- 체커보드 패턴을 이용한 외부 캘리브레이션 수행
- 주기적인 재캘리브레이션을 통한 정확도 유지

3. **깊이 이미지 보정**:

- 깊이 카메라의 내재적 오차 보정
- 표면 반사율에 따른 깊이 측정 오차 보정
- 이상치(outlier) 필터링 및 노이즈 감소

4. **작업 공간 설정**:

- 로봇 팔의 작업 영역 정의
- 안전 경계 설정 (작업 공간 제한)
- 충돌 가능 영역 매핑

3.4 소프트웨어 프레임워크

3.4.1 개발 환경

시스템 개발을 위한 소프트웨어 환경은 다음과 같이 구성된다:

1. **운영체제**:

- Ubuntu 22.04 LTS (Jammy Jellyfish)

2. **미들웨어**:

- ROS2 Humble Hawksbill
- MoveIt2 (Humble 호환 버전)

3. **프로그래밍 언어**:
    - Python 3.10 (주요 개발 언어)
    - C++ 17 (성능이 중요한 모듈용)
4. **딥러닝 프레임워크**:
    - PyTorch 2.0 이상
    - ONNX Runtime
    - TensorRT (Jetson 플랫폼용)
5. **컴퓨터 비전 라이브러리**:
    - OpenCV 4.5 이상
    - Open3D (점군 처리용)
    - PCL (Point Cloud Library)
6. **개발 도구**:
    - Visual Studio Code
    - Git (버전 관리)
    - Docker (환경 격리 및 배포)

3.4.2 주요 소프트웨어 컴포넌트

VLM 기반 시스템의 주요 컴포넌트:

1. **NanoOWL**:
    - 자연어 프롬프트 기반 객체 검출
    - TensorRT 최적화를 통한 실시간 성능 확보
    - ROS2 인터페이스 구현
2. **NanoSAM** 또는 **MobileSAMv2**:
    - 객체 분할 및 마스크 생성
    - TensorRT로 최적화된 추론 엔진
    - ROS2 인터페이스 구현
3. **Pose Estimation Module**:
    - 깊이 이미지와 분할 마스크를 기반으로 3D 포즈 추정
    - Point Cloud Processing을 통한 6D 포즈 계산
    - 위치 정보의 시간적 필터링 및 안정화
4. **Grasp Planning Module**:
    - 객체 형상 분석 및 최적 파지점 결정
    - Force Closure 및 안정성 평가
    - 접근 경로 및 파지 전략 생성
5. **MoveIt2 Integration**:

- 파지 포즈를 MoveIt2와 연동
- 충돌 회피 경로 계획
- 실시간 궤적 실행 및 모니터링

강화학습 기반 시스템의 주요 컴포넌트:

1. **Feature Extractor**:
   - RGBD 이미지에서 상태 표현 추출
   - 객체 특성 및 로봇 상태 정보 통합
   - 텍스트 명령에서 목표 정보 추출
2. **Policy Network**:
   - 사전 학습된 SAC 또는 PPO 모델
   - 상태 입력에 따른 액션 출력
   - ONNX 또는 TensorRT로 최적화
3. **Reward Estimator**:
   - 파지 성공 가능성 예측
   - 실시간 보상 함수 계산
   - 액션 성공률 평가
4. **Action Executor**:
   - 정책 네트워크 출력을 로봇 명령으로 변환
   - 실시간 액션 실행 및 조정
   - 피드백 기반 액션 수정
5. **Training Environment**:
   - 시뮬레이션 환경 (Isaac Sim 또는 Gazebo)
   - 실제 로봇과의 도메인 갭 최소화
   - 실시간 학습 및 적응

3.4.3 모델 최적화 전략

실시간 성능을 위한 모델 최적화 전략은 다음과 같다:

1. **모델 경량화**:
   - 네트워크 아키텍처 최적화 (MobileNet, EfficientNet 등 경량 백본 사용)
   - 지식 증류(Knowledge Distillation)를 통한 모델 크기 축소
   - 불필요한 레이어 및 파라미터 제거
2. **양자화(Quantization)**:
   - FP32에서 FP16 또는 INT8로 양자화
   - 정확도 손실을 최소화하는 최적 양자화 레벨 선택
   - 모델별 양자화 민감도 분석 및 조정

3. **가속 라이브러리 활용**:
    - NVIDIA TensorRT를 사용한 추론 최적화
    - CUDA 및 cuDNN 활용
    - OpenVINO 또는 ONNX Runtime 최적화
4. **병렬 처리 및 파이프라이닝**:
    - 여러 처리 단계의 병렬화
    - GPU 스트리밍을 활용한 파이프라인 구성
    - 작업 분산 및 스케줄링 최적화
5. **메모리 최적화**:
    - 공유 메모리 활용으로 데이터 복사 최소화
    - 배치 처리를 통한 메모리 액세스 효율화
    - 메모리 누수 방지 및 가비지 컬렉션 최적화

이러한 최적화 전략은 특히 제한된 컴퓨팅 자원을 가진 환경(예: Jetson 플랫폼)에서 실시간 성능을 달성하는 데 필수적이다.

4. VLM 기반 접근법

4.1 자연어 명령 처리

VLM 기반 접근법에서는 사용자의 자연어 명령을 처리하여 객체 인식 모듈에 필요한 정보를 추출하는 것이 첫 번째 단계이다.

4.1.1 자연어 명령 파싱

사용자의 자연어 명령을 효과적으로 처리하기 위해 다음과 같은 파싱 과정을 거친다:

1. **전처리**:
    - 텍스트 정규화 (대/소문자 변환, 특수 문자 처리 등)
    - 불용어(stopwords) 제거
    - 철자 오류 교정 및 문장 정규화
2. **구문 분석**:
    - 명령문 구조 파악 (주어, 동사, 목적어 등 식별)
    - 의존성 파싱을 통한 문장 구조 분석
    - 명령의 의도(intent) 추출
3. **개체 인식**:
    - 목표 객체 식별
    - 객체의 속성(색상, 크기, 형태 등) 추출
    - 위치 정보(왼쪽, 오른쪽, 위, 아래 등) 추출
4. **관계 분석**:

- 여러 객체 간의 관계 파악
- 공간적 관계(~위에, ~옆에 등) 해석
- 모호성 해결을 위한 문맥 분석

이러한 파싱 과정은 간단한 규칙 기반 방식부터 심층 언어 모델(예: BERT, GPT)까지 다양한 방법으로 구현할 수 있다. 본 연구에서는 실시간 성능과 정확성을 고려하여 경량화된 언어 모델을 사용하되, 로봇 파지 작업에 특화된 명령 패턴을 효과적으로 처리할 수 있도록 설계한다.

4.1.2 명령-CLIP 프롬프트 변환

파싱된 자연어 명령은 CLIP 모델이 이해할 수 있는 형태의 프롬프트로 변환되어야 한다. 이를 위한 주요 전략은 다음과 같다:

1. **템플릿 기반 프롬프트 생성**:
   - 미리 정의된 템플릿을 사용하여 자연어 명령을 CLIP 프롬프트로 변환
   - 예: "빨간 컵을 집어" → "a red cup", "a cup that is red"
   - 객체 속성에 따른 다양한 템플릿 활용

2. **앙상블 프롬프트 생성**:
   - 동일한 명령에 대해 여러 변형의 프롬프트를 생성
   - 다양한 표현을 조합하여 인식 성능 향상
   - 예: "큰 파란 상자" → ["a large blue box", "a big blue container", "a blue box that is large"]

3. **텍스트 증강(Text Augmentation)**:
   - 유사어 및 관련 개념을 포함한 확장 프롬프트 생성
   - 객체의 다양한 특성 및 시각적 변형 고려
   - 예: "사과" → "a red apple", "a green apple", "a yellow apple"

4. **부정 프롬프트(Negative Prompting)**:
   - 명령과 관련 없는 객체를 명시적으로 배제하는 프롬프트 추가
   - 비슷한 객체 간 구분을 강화
   - 예: "파란색이 아닌 빨간 컵", "접시가 아닌 컵"

4.1.3 모호성 해결 전략

자연어 명령은 종종 모호성을 포함할 수 있으며, 이를 해결하기 위한 전략이 필요하다:

1. **다중 객체 처리**:
   - 동일한 설명에 일치하는 여러 객체가 발견될 경우
   - 상대적 크기, 로봇과의 거리 등 추가 기준으로 선택
   - 필요시 사용자에게 명확화 요청

2. **문맥 기반 해석**:
   - 이전 명령과의 연계성 고려
   - 작업 환경의 상황 인식
   - 대화 흐름에 따른 참조 해결

3. **능동적 인식**:
    - 여러 각도에서의 관측을 통한 정보 보완
    - 모호한 객체의 추가 특징 탐색
    - 사용자와의 상호작용을 통한 정보 보완

4. **신뢰도 기반 결정**:
    - 인식 결과의 신뢰도 점수 활용
    - 임계값 이하의 낮은 신뢰도 결과에 대한 처리 전략
    - 앙상블 프롬프트의 일관성 평가

4.2 VLM 기반 객체 인식 구현

VLM 기반 접근법의 핵심은 자연어 명령에 따라 관련 객체를 시각적으로 인식하는 것이다. 이를 위해 최신 비전-언어 모델들을 활용한다.

4.2.1 NanoOWL과 NanoSAM 통합

NanoOWL과 NanoSAM은 NVIDIA에서 개발한 경량화된 모델로, Jetson 플랫폼에서의 실시간 성능을 위해 최적화되었다. 이 두 모델을 통합하여 객체 검출 및 분할 파이프라인을 구축하는 방법은 다음과 같다:

1. **NanoOWL 설치 및 구성**:
    - GitHub 저장소에서 NanoOWL 코드 클론
    - TensorRT 엔진 빌드 및 최적화
    - ROS2 패키지로 래핑

2. **NanoSAM 설치 및 구성**:
    - GitHub 저장소에서 NanoSAM 코드 클론
    - TensorRT 엔진 빌드 및 최적화
    - ROS2 패키지로 래핑

3. **파이프라인 구성**:
    - NanoOWL을 사용하여 자연어 프롬프트에 따른 객체 검출
    - 검출된 객체의 바운딩 박스를 NanoSAM의 프롬프트로 사용
    - NanoSAM을 통해 정밀한 객체 마스크 생성

4. **성능 최적화**:
    - 두 모델의 추론 파이프라인 병렬화
    - 결과 캐싱을 통한 중복 계산 방지
    - 프레임 간 정보 재활용을 통한 계산량 감소

NanoOWL은 텍스트 프롬프트를 입력으로 받아 이미지에서 관련 객체의 바운딩 박스를 생성한다. 이 정보는 NanoSAM에 전달되어 정밀한 객체 마스크를 생성한다. 이러한 과정은 ROS2 노드로 구현되어 객체 인식 파이프라인의 핵심을 형성한다.

```
# NanoOWL-NanoSAM 통합 파이프라인의 의사코드 예시
def process_frame(rgb_image, depth_image, text_prompt):
    # NanoOWL로 텍스트 프롬프트에 따른 객체 검출
    bounding_boxes, confidence_scores = nanoowl_predictor.predict(
        rgb_image, text=text_prompt
    )

    results = []
    for box, score in zip(bounding_boxes, confidence_scores):
        if score < CONFIDENCE_THRESHOLD:
            continue

        # NanoSAM에 바운딩 박스를 프롬프트로 전달하여 마스크 생성
        mask = nanosam_predictor.predict(rgb_image, box=box)

        # 마스크와 깊이 이미지를 사용하여 3D 위치 추정
        pose_3d = estimate_3d_pose(mask, depth_image, camera_intrinsics)

        results.append({
            'box': box,
            'mask': mask,
            'confidence': score,
            'pose_3d': pose_3d
        })

    return results
```

4.2.2 Grounding DINO와 MobileSAMv2 통합

NanoOWL과 NanoSAM의 대안으로 Grounding DINO와 MobileSAMv2를 통합한 파이프라인도 구현한다. 이 조합은 더 높은 정확도를 제공할 수 있지만, 계산 요구량이 더 많을 수 있다.

1. **Grounding DINO 설치 및 구성**:
    - GitHub 저장소에서 Grounding DINO 코드 클론
    - 모델 다운로드 및 ONNX 변환
    - TensorRT 최적화 (가능한 경우)
2. **MobileSAMv2 설치 및 구성**:
    - GitHub 저장소에서 MobileSAMv2 코드 클론
    - 모델 다운로드 및 최적화
    - ROS2 인터페이스 구현
3. **파이프라인 구성**:
    - Grounding DINO를 사용하여 자연어 프롬프트에 따른 객체 검출
    - 검출된 객체의 바운딩 박스를 MobileSAMv2의 입력으로 사용
    - MobileSAMv2를 통해 객체 마스크 생성
4. **최적화 전략**:

- Grounding DINO의 에지 버전 사용 고려
- 낮은 해상도에서 초기 검출 후 필요시 고해상도 처리
- 시간적 일관성을 활용한 추적 및 계산량 감소

```
# Grounding DINO-MobileSAMv2 통합 파이프라인의 의사코드 예시
def process_frame(rgb_image, depth_image, text_prompt):
    # Grounding DINO로 텍스트 프롬프트에 따른 객체 검출
    bounding_boxes, confidence_scores, labels = groundingdino_predictor.predict(
        image=rgb_image, text=text_prompt
    )

    results = []
    for box, score, label in zip(bounding_boxes, confidence_scores, labels):
        if score < CONFIDENCE_THRESHOLD:
            continue

        # MobileSAMv2에 바운딩 박스를 전달하여 마스크 생성
        mask = mobilesamv2_predictor.predict(rgb_image, box=box)

        # 마스크와 깊이 이미지를 사용하여 3D 위치 추정
        pose_3d = estimate_3d_pose(mask, depth_image, camera_intrinsics)

        results.append({
            'box': box,
            'mask': mask,
            'label': label,
            'confidence': score,
            'pose_3d': pose_3d
        })

    return results
```

4.2.3 3D 위치 및 포즈 추정

인식된 객체의 2D 마스크와 깊이 이미지를 결합하여 3D 위치 및 포즈를 추정하는 방법을 구현한다.

1. **마스크 기반 점군 생성**:
   - 객체 마스크와 깊이 이미지를 사용하여 객체의 3D 점군 추출
   - 카메라 내부 매개변수를 사용한 픽셀 좌표의 3D 변환
   - 이상치(outlier) 필터링 및 노이즈 제거
2. **기하학적 특징 추출**:
   - 점군의 무게 중심(centroid) 계산
   - 주성분 분석(PCA)을 통한 주요 축 및 방향 추정
   - 객체의 바운딩 박스(3D) 또는 바운딩 실린더 계산
3. **6D 포즈 추출**:
   - 주성분 축을 기반으로 한 객체의 방향(orientation) 추정

- 객체 형상에 따른 최적 파지 축 결정
- 다양한 형태의 객체에 대한 특화된 포즈 추정 전략

4. **시간적 필터링**:

- 칼만 필터를 사용한 포즈 추정값의 안정화
- 갑작스러운 변화 감지 및 이상치 처리
- 다중 프레임에 걸친 추정 결과의 통합

```
# 마스크 및 깊이 이미지를 사용한 3D 포즈 추정 의사코드
def estimate_3d_pose(mask, depth_image, camera_intrinsics):
    # 마스크 영역 내의 깊이 픽셀 추출
    masked_depth = depth_image * mask

    # 2D 픽셀 좌표를 3D 점군으로 변환
    points_3d = []
    for y in range(mask.shape[0]):
        for x in range(mask.shape[1]):
            if mask[y, x] > 0 and depth_image[y, x] > 0:
                z = depth_image[y, x]
                # 카메라 내부 매개변수를 사용한 3D 좌표 계산
                x_3d = (x - camera_intrinsics.cx) * z / camera_intrinsics.fx
                y_3d = (y - camera_intrinsics.cy) * z / camera_intrinsics.fy
                points_3d.append((x_3d, y_3d, z))

    # 점군 클러스터링 및 필터링
    points_3d = filter_outliers(points_3d)

    # 무게 중심 계산
    centroid = compute_centroid(points_3d)

    # 주성분 분석(PCA)을 통한 주 축 계산
    axes, eigenvalues = compute_principal_axes(points_3d)

    # 6D 포즈 (위치 + 방향) 구성
    position = centroid
    orientation = axes_to_quaternion(axes)

    return {
        'position': position,
        'orientation': orientation,
        'size': eigenvalues,  # 객체의 대략적인 크기
        'points': points_3d   # 필터링된 3D 점군
    }
```

4.3 VLM 기반 파지 계획

인식된 객체의 위치와 형태 정보를 바탕으로 최적의 파지 전략을 계획하는 방법을 설명한다.

4.3.1 객체 특성 기반 파지점 선정

객체의 기하학적 특성과 의미적 정보를 활용하여 최적의 파지점을 선정하는 방법은 다음과 같다:

1. **기하학적 분석**:
    - 객체의 주축(principal axis)을 따라 가능한 파지 방향 탐색
    - 객체의 대칭성 및 안정성 고려
    - 곡률 분석을 통한 적합한 접촉 지점 식별
2. **의미적 분석**:
    - 객체의 종류에 따른 적절한 파지 방식 선택
    - 특정 부분(예: 컵의 손잡이)에 대한 파지 우선순위 설정
    - 취약성이나 기능적 중요성을 고려한 파지점 제외
3. **접근성 평가**:
    - 로봇 팔의 작업 공간 내에서 접근 가능한 파지점 필터링
    - 충돌 가능성이 낮은 파지 방향 우선 고려
    - 다른 객체에 의한 가림(occlusion) 여부 확인
4. **파지 품질 예측**:
    - 각 파지 후보의 안정성 및 성공 가능성 평가
    - 접촉 지점의 마찰 및 미끄러짐 가능성 고려
    - 객체 무게와 로봇 팔의 힘 제한 고려

4.3.2 파지 접근 전략

선정된 파지점까지 로봇 팔을 안전하게 이동시키기 위한 접근 전략은 다음과 같다:

1. **사전 위치 계획**:
    - 파지 시작 전 안전한 사전 위치로 이동
    - 작업 공간의 전반적인 상황 고려
    - 파지 동작을 위한 최적 시작 자세 선정
2. **접근 경로 생성**:
    - 장애물 회피를 고려한 접근 경로 계획
    - 부드러운 궤적을 위한 최적 경로 생성
    - 객체 변위 가능성을 최소화하는 접근 방향 선택
3. **속도 및 가속도 프로파일**:
    - 상황에 적합한 움직임 속도 프로파일 설정
    - 급격한 가속/감속 회피를 통한 진동 최소화
    - 파지 직전 정밀한 위치 조정을 위한 속도 감소
4. **적응적 접근**:
    - 접근 과정 중 실시간 센서 피드백 활용
    - 예상치 못한 장애물 발견 시 경로 재계획

- 객체 위치 변화에 따른 접근 전략 조정

4.3.3 MoveIt2 통합 파지 계획

MoveIt2를 활용하여 파지 동작을 계획하고 실행하는 방법은 다음과 같다:

1. **파지 포즈 정의**:
   - `moveit_msgs::Grasp` 메시지를 사용하여 파지 정보 구성
   - 파지 위치, 방향, 접근 방향, 그리퍼 폭 등 설정
   - 다중 파지 후보를 우선순위에 따라 구성
2. **Planning Scene 설정**:
   - 객체를 collision object로 추가
   - 환경 내 다른 장애물 정보 업데이트
   - 필요시 허용 충돌(allowed collision) 설정
3. **모션 계획 수행**:
   - MoveIt의 `plan` 및 `execute` API 호출
   - 다양한 계획 알고리즘(RRT, RRTConnect 등) 활용
   - 계획 실패 시 대안 전략 시도
4. **그리퍼 제어**:
   - 객체 특성에 맞는 적절한 그리퍼 폭 계산
   - 파지력 동적 조절을 위한 피드백 제어
   - 파지 성공 여부 감지 및 대응

```
# MoveIt2를 활용한 파지 계획 의사코드
def plan_and_execute_grasp(pose_3d, object_properties, move_group):
    # 파지 메시지 생성
    grasp = moveit_msgs.msg.Grasp()

    # 파지 포즈 설정
    grasp.grasp_pose.header.frame_id = "base_link"
    grasp.grasp_pose.pose.position = pose_3d.position
    grasp.grasp_pose.pose.orientation = pose_3d.orientation

    # 접근 방향 설정
    grasp.pre_grasp_approach.direction.header.frame_id = "wrist_frame"
    grasp.pre_grasp_approach.direction.vector.z = 1.0
    grasp.pre_grasp_approach.min_distance = 0.05
    grasp.pre_grasp_approach.desired_distance = 0.15

    # 후퇴 방향 설정
    grasp.post_grasp_retreat.direction.header.frame_id = "wrist_frame"
    grasp.post_grasp_retreat.direction.vector.z = -1.0
    grasp.post_grasp_retreat.min_distance = 0.05
    grasp.post_grasp_retreat.desired_distance = 0.15
```

```
    # 그리퍼 상태 설정
    grasp.pre_grasp_posture = create_gripper_posture("open")
    grasp.grasp_posture = create_gripper_posture("closed")

    # 객체를 Planning Scene에 추가
    add_object_to_planning_scene(object_properties)

    # 파지 계획 실행
    success = move_group.pick("object_name", [grasp])

    if not success:
        # 대안 파지 전략 시도
        alternative_grasps = generate_alternative_grasps(pose_3d,
object_properties)
        success = move_group.pick("object_name", alternative_grasps)

    return success
```

4.4 VLM 기반 접근법의 성능 최적화

VLM 기반 접근법의 실시간 성능을 확보하기 위한 최적화 전략을 설명한다.

4.4.1 모델 경량화 및 양자화

계산 효율성을 높이기 위한 모델 경량화 및 양자화 전략은 다음과 같다:

1. **모델 구조 최적화**:
    - 불필요한 레이어 제거 또는 간소화
    - 효율적인 백본 네트워크로 대체 (MobileNet, EfficientNet 등)
    - 특화된 모델 아키텍처 활용 (NanoOWL, NanoSAM 등)
2. **양자화 기법**:
    - FP32에서 FP16 또는 INT8로 정밀도 감소
    - 하드웨어 특화 양자화 (TensorRT 최적화)
    - 양자화 교정(calibration)을 통한 정확도 손실 최소화
3. **지식 증류**:
    - 큰 교사(teacher) 모델에서 작은 학생(student) 모델로 지식 전이
    - 모델 크기 감소와 정확도 유지 간의 균형 탐색
    - 특정 태스크에 특화된 증류 전략 개발
4. **가지치기(Pruning)**:
    - 중요도가 낮은 가중치 또는 필터 제거
    - 구조적 가지치기를 통한 병렬 계산 효율 향상
    - 희소성(sparsity) 활용 최적화

4.4.2 파이프라인 최적화

전체 파이프라인의 계산 효율성을 높이기 위한 전략은 다음과 같다:

1. **병렬 처리**:
    - 독립적인 처리 단계의 병렬화
    - GPU 연산과 CPU 연산의 병렬 수행
    - 다중 스레드를 통한 작업 분산
2. **메모리 최적화**:
    - 중간 결과의 재사용을 통한 중복 계산 방지
    - 제로 카피(zero-copy) 기법을 통한 메모리 전송 최소화
    - 메모리 풀링을 통한 동적 할당 오버헤드 감소
3. **시간적 일관성 활용**:
    - 이전 프레임의 결과를 활용한 계산량 감소
    - 객체 추적을 통한 반복 검출 회피
    - 정적 장면에서의 계산 결과 캐싱
4. **해상도 및 처리 빈도 조정**:
    - 필요에 따른 이미지 해상도 동적 조정
    - 상황에 따른 처리 프레임 레이트 조절
    - 중요도에 따른 처리 우선순위 부여

4.4.3 하드웨어 가속 활용

하드웨어 특성을 최대한 활용하기 위한 전략은 다음과 같다:

1. **TensorRT 최적화**:
    - NVIDIA TensorRT를 사용한 모델 최적화
    - 그래프 최적화 및 융합(fusion)을 통한 성능 향상
    - 배치 크기 및 실행 정밀도 최적화
2. **CUDA 및 cuDNN 활용**:
    - GPU 연산을 위한 CUDA 커널 최적화
    - cuDNN 라이브러리를 통한 딥러닝 연산 가속
    - 스트림 기반 병렬 처리 및 메모리 관리
3. **하드웨어 특화 최적화**:
    - Jetson 플랫폼의 하드웨어 특성 활용
    - CPU, GPU, DLA 등 다양한 계산 리소스의 효율적 분배
    - 전력 소비와 성능 간의 균형을 위한 동적 조절
4. **분산 처리**:
    - 필요시 외부 컴퓨팅 리소스 활용 고려
    - 에지와 클라우드의 하이브리드 처리 전략

- 네트워크 지연과 계산 이점 간의 트레이드오프 분석

5. 강화학습 기반 접근법

5.1 강화학습 기반 파지 시스템 설계

5.1.1 MDP 정식화

강화학습(Reinforcement Learning, RL) 문제를 해결하기 위해서는 먼저 로봇 파지 작업을 마르코프 결정 프로세스(Markov Decision Process, MDP)로 정식화해야 한다. MDP는 상태, 행동, 상태 전이 함수, 보상 함수, 할인 인자로 구성된다.

1. **상태 공간(State Space) 설계**:
   - RGBD 이미지에서 추출한 특징 벡터
   - 로봇 팔의 현재 자세 및 그리퍼 상태
   - 자연어 명령에서 추출한 목표 정보
   - 객체와 로봇의 상대적 위치 관계

2. **행동 공간(Action Space) 설계**:
   - 연속적인(continuous) 행동 공간 사용
   - 엔드 이펙터의 상대적 이동 (delta x, delta y, delta z)
   - 그리퍼의 방향 변화 (delta roll, delta pitch, delta yaw)
   - 그리퍼 개폐 제어 (gripper width)

3. **상태 전이 함수**:
   - 로봇 동역학 모델에 따른 상태 변화
   - 환경과의 상호작용 (객체 이동, 충돌 등)
   - 센서 노이즈 및 불확실성 모델링

4. **보상 함수(Reward Function)**:
   - 파지 성공 여부에 따른 주요 보상
   - 목표 객체와의 거리에 따른 중간 보상
   - 충돌 및 부적절한 행동에 대한 페널티
   - 효율적인 동작 경로에 대한 추가 보상

5. **할인 인자(Discount Factor)**:
   - 미래 보상에 대한 가중치 설정
   - 장기적 목표와 단기적 행동 간의 균형 조정

이러한 MDP 정식화를 통해 로봇 파지 작업이 강화학습 프레임워크 내에서 잘 정의된 문제로 표현될 수 있다.

```
# 로봇 파지를 위한 MDP 정식화 예시
class GraspingMDP:
    def __init__(self, rgb_camera, depth_camera, robot_arm, text_command):
        self.rgb_camera = rgb_camera
        self.depth_camera = depth_camera
```

```
        self.robot_arm = robot_arm
        self.text_command = text_command
        self.target_object_features = extract_target_features(text_command)

    def get_state(self):
        # 현재 상태 관측
        rgb_image = self.rgb_camera.capture()
        depth_image = self.depth_camera.capture()
        robot_state = self.robot_arm.get_state()

        # 상태 특징 추출
        visual_features = extract_visual_features(rgb_image, depth_image)
        text_features = extract_text_features(self.text_command)

        return {
            'visual_features': visual_features,
            'robot_state': robot_state,
            'text_features': text_features
        }

    def step(self, action):
        # 행동 실행
        delta_position = action[:3]  # dx, dy, dz
        delta_orientation = action[3:6]  # droll, dpitch, dyaw
        gripper_width = action[6]

        self.robot_arm.move_relative(delta_position, delta_orientation)
        self.robot_arm.set_gripper_width(gripper_width)

        # 새로운 상태 관측
        new_state = self.get_state()

        # 보상 계산
        reward = self._compute_reward(new_state)

        # 종료 조건 확인
        done = self._check_termination()

        return new_state, reward, done

    def _compute_reward(self, state):
        # 보상 함수 구현
        reward = 0.0

        # 목표 객체와의 거리에 따른 보상
        target_distance = compute_distance_to_target(state,
self.target_object_features)
        reward -= target_distance * DISTANCE_PENALTY

        # 파지 성공 여부에 따른 보상
        if is_grasp_successful(state):
            reward += SUCCESS_REWARD

        # 충돌 페널티
```

```
        if is_collision_detected(state):
            reward -= COLLISION_PENALTY

        return reward

    def _check_termination(self):
        # 종료 조건 확인
        if is_grasp_successful(self.get_state()):
            return True

        if is_collision_detected(self.get_state()):
            return True

        if self.steps_taken > MAX_STEPS:
            return True

        return False
```

5.1.2 상태 표현 및 특징 추출

강화학습의 효율성과 성능은 상태 표현의 질에 크게 의존한다. 로봇 파지 작업에 적합한 상태 표현 및 특징 추출 방법은 다음과 같다:

1. **시각적 특징 추출**:
   - CNN 기반 특징 추출기 사용
   - 사전 학습된 모델(ResNet, MobileNet 등) 활용
   - CLIP의 이미지 인코더를 특징 추출기로 활용
   - 깊이 정보를 활용한 3D 구조적 특징 추출

2. **언어적 특징 추출**:
   - BERT, RoBERTa 등의 언어 모델 활용
   - 자연어 명령에서 목표 객체 및 속성 추출
   - CLIP의 텍스트 인코더를 특징 추출기로 활용
   - 명령의 의미적 임베딩 생성

3. **로봇 상태 표현**:
   - 로봇 팔의 조인트 각도 및 속도
   - 엔드 이펙터의 현재 포즈(위치 및 방향)
   - 그리퍼 상태(개폐 정도, 접촉 감지 등)
   - 현재 수행 중인 작업 단계 정보

4. **관계적 특징 추출**:
   - 로봇과 객체 간의 상대적 위치 및 방향
   - 객체 간의 공간적 관계 정보
   - 현재 상태와 목표 상태 간의 차이
   - 파지 가능성 예측 정보

이러한 특징들은 개별적으로 추출된 후 적절하게 결합되어 강화학습 알고리즘에 입력된다. 특히 CNN과 같은 딥 네트워크를 통해 추출된 특징은 차원이 높고 복잡할 수 있으므로, 필요에 따라 차원 축소 기법(PCA, t-SNE 등)이나 특징 선택 알고리즘을 적용할 수 있다.

5.1.3 보상 함수 설계

효과적인 강화학습을 위해서는 적절한 보상 함수의 설계가 중요하다. 로봇 파지 작업에 적합한 보상 함수 설계 방법은 다음과 같다:

1. **스파스 보상과 밀집 보상**:
    - 스파스 보상: 파지 성공 시에만 큰 양의 보상 제공
    - 밀집 보상: 목표를 향한 점진적 진행에 따라 중간 보상 제공
    - 두 보상의 적절한 조합을 통한 학습 효율성 향상

2. **다중 목표 보상**:
    - 목표 객체 접근: 그리퍼와 목표 객체 간 거리 감소에 따른 보상
    - 적절한 방향 정렬: 그리퍼 방향과 최적 파지 방향 간 정렬에 따른 보상
    - 안정적 파지: 성공적인 파지 후 객체 안정성에 따른 보상
    - 임무 완수: 파지 후 목표 위치로 이동 성공에 따른 보상

3. **페널티 및 제약 조건**:
    - 충돌 페널티: 로봇 팔이 장애물과 충돌 시 음의 보상
    - 비효율적 움직임 페널티: 불필요한 움직임에 대한 작은 음의 보상
    - 작업 영역 이탈 페널티: 로봇 팔이 작업 영역을 벗어날 때 페널티
    - 시간 제한 페널티: 너무 오래 걸리는 작업에 대한 페널티

4. **언어 지향적 보상**:
    - 올바른 객체 선택: 자연어 명령에 명시된 객체를 파지할 때 추가 보상
    - 속성 일치: 명령에 명시된 속성(색상, 크기 등)과 일치하는 객체 선택 시 보상
    - 의미적 정확성: 명령의 의도와 행동의 일치도에 따른 보상

보상 함수는 학습 중 정책의 행동을 직접적으로 유도하므로, 원하는 행동을 명확하게 장려하면서도 불필요한 행동을 억제하도록 신중하게 설계해야 한다. 또한, 개별 보상 요소의 가중치를 적절하게 조정하여 바람직한 학습 경로를 유도해야 한다.

```
# 로봇 파지를 위한 보상 함수 설계 예시
def compute_reward(current_state, action, next_state, target_object_features):
    reward = 0.0

    # 1. 거리 기반 보상 (목표 객체에 접근할수록 높은 보상)
    prev_distance = compute_distance_to_target(current_state,
target_object_features)
    curr_distance = compute_distance_to_target(next_state, target_object_features)
    distance_reward = DISTANCE_SCALE * (prev_distance - curr_distance)
    reward += distance_reward

    # 2. 방향 정렬 보상 (그리퍼가 최적 파지 방향에 정렬될수록 높은 보상)
```

```
    alignment_score = compute_gripper_alignment(next_state,
target_object_features)
    alignment_reward = ALIGNMENT_SCALE * alignment_score
    reward += alignment_reward

    # 3. 파지 성공 보상 (성공적인 파지에 큰 보상)
    if is_grasp_successful(next_state):
        grasp_stability = compute_grasp_stability(next_state)
        success_reward = SUCCESS_REWARD * grasp_stability
        reward += success_reward

    # 4. 페널티
    # 4.1. 충돌 페널티
    if detect_collision(next_state):
        reward -= COLLISION_PENALTY

    # 4.2. 작업 영역 이탈 페널티
    if is_out_of_workspace(next_state):
        reward -= WORKSPACE_PENALTY

    # 4.3. 에너지 효율성 페널티 (불필요한 움직임 억제)
    energy_cost = compute_action_energy(action)
    reward -= ENERGY_SCALE * energy_cost

    # 5. 언어 지향적 보상 (명령과 일치하는 객체를 처리할 때 추가 보상)
    language_alignment = compute_language_alignment(next_state,
target_object_features)
    reward += LANGUAGE_SCALE * language_alignment

    return reward
```

5.2 강화학습 알고리즘 선택 및 구현

5.2.1 SAC (Soft Actor-Critic) 알고리즘

SAC는 최대 엔트로피 강화학습 프레임워크를 기반으로 한 off-policy 알고리즘으로, 탐색(exploration)과 활용(exploitation) 간의 균형을 효과적으로 유지하며 샘플 효율적인 학습이 가능하다. 이는 연속적인 행동 공간을 가진 로봇 파지 작업에 특히 적합하다.

1. **SAC의 주요 특징**:
    - 최대 엔트로피 원칙: 보상 최대화와 함께 행동 엔트로피 최대화 추구
    - 이중 Q-함수: 과대 추정 편향을 줄이기 위한 이중 Q-함수 사용
    - 오토튜닝: 엔트로피 목표를 자동으로 조정하는 메커니즘
    - Off-policy 특성: 경험 재사용을 통한 샘플 효율성 향상
2. **네트워크 구조**:
    - 정책 네트워크: 상태를 입력으로 받아 행동 분포(평균 및 표준편차) 출력
    - 가치 네트워크: 두 개의 Q-함수 네트워크로 구성
    - 특징 추출기: CNN, CLIP 등을 활용한 상태 특징 추출

3. **구현 세부사항**:

   - 경험 리플레이 버퍼: 다양한 경험 샘플 저장 및 재사용
   - 점진적 업데이트: 타겟 네트워크의 소프트 업데이트를 통한 안정성 확보
   - 배치 정규화: 학습 안정성 향상을 위한 정규화 기법 적용
   - 그래디언트 클리핑: 그래디언트 폭발 방지를 위한 제한

4. **하이퍼파라미터 튜닝**:

   - 학습률: 정책 및 가치 네트워크의 적절한 학습률 설정
   - 할인 인자: 미래 보상의 중요도 조절
   - 업데이트 비율: 타겟 네트워크 업데이트 속도 조절
   - 배치 크기: 한 번에 처리하는 경험 샘플 수 설정

```
# SAC 구현 의사코드
class SAC:
    def __init__(self, state_dim, action_dim, hidden_dim=256, lr=3e-4, gamma=0.99,
tau=0.005):
        # 네트워크 초기화
        self.actor = ActorNetwork(state_dim, action_dim, hidden_dim)
        self.critic1 = CriticNetwork(state_dim, action_dim, hidden_dim)
        self.critic2 = CriticNetwork(state_dim, action_dim, hidden_dim)
        self.target_critic1 = CriticNetwork(state_dim, action_dim, hidden_dim)
        self.target_critic2 = CriticNetwork(state_dim, action_dim, hidden_dim)

        # 타겟 네트워크 초기화
        self.update_target_networks(tau=1.0)  # 완전 복사

        # 옵티마이저
```