2021년 항공우주및소프트웨어공학 전공 종합설계 제안서

과제명: 전장 상황에서 정찰 임무 수행이 가능한 드론 개발
(Development of the UAV for reconnaissance in battlefield situations)

팀명: 천리안

팀원: 전병륜(팀장), 김강산, 김송섭, 박보근, 전동환

작성일: 2021년 5월 31일

지도교수: 이선아 (서명)

목 차

1. 개요

1.1. 과제의 개요

본 과제는 전장 상황과 같은 사전 정보가 없는 제한된 환경에서 표적탐지와 지형탐색과 같은 정찰 임무 수행이 가능한 드론을 개발하는 것이다. 카메라를 이용한 군인, 표지판, 활자 등의 표적탐지, 2D LiDAR 센서를 이용한 지형지도 생성, 탐색한 지형정보를 활용한 경로 탐색 기술을 구현한다.

1.2. 추진배경 및 필요성

1.2.1. 군용 드론 시장 현황 및 전망

군사용 무인기(UAV)의 저렴한 가격과 조종자의 안전 확보 등 전장 상황에서의 강점이 두드러지면서 군용 드론 시장 규모가 커지고 있다. 그림 1을 통해, 전 세계 군용 드론 시장은 2018년 121억 3,000만 달러에서 연평균 성장률 12.00%로 증가하여, 2025년에는 268억 2,000만 달러에 이를 것으로 전망된다. 그림 2에 따르면 우리나라의 군용 드론 시장은 2018년 3억 2,910만 달러에서 연평균 성장률 11.72%로 증가하여, 2025년에는 7억 1,500만 달러에 이를 것으로 전망된다[1]. 특히 그림 3과 같이 정보·감시·정찰·표적획득용 드론은 2018년 88억 7,120만 달러에서 연평균 성장률 11.78%로 증가하여 2025년에는 193억 4,540만 달러에 이를 것으로 추산된다[2].

그림 1 글로벌 군용 드론 시장 규모 및 전망(출처: “군용 드론 시장”, 연구개발특구진흥재단(2019))

그림 2 우리나라 군용 드론 시장 규모 및 전망(출처: “군용 드론 시장”, 연구개발특구진흥재단(2019))

그림 3 글로벌 군용 드론 시장의 용도별 시장 규모 및 전망(출처: “군용 드론 시장”, 연구개발특구진흥재단(2019))

1.2.2. 임무 수행 드론에 관한 관심

육군은 2017년 ‘5대 게임 체인저(Game Changer)’라는 새로운 군사력 건설개념이 처음으로 공개했다. 5대 게임 체인저는 군 병력 감축과 복무기간 단축 등 안보 환경 변화, 북한 핵·미사일 위협의 고도화, 전쟁 패러다임의 변화 등에 대응하기 위해 건설할 5대 핵심 전력이다. 이 중 드론봇 전투단은 병력 감축 태풍을 맞은 육군이 가장 중점을 두고 있는 분야 중 하나이다. 중대부터 군단에 이르기까지 다양한 제대에서 다양한 목적을 가진 소형 무인기와 민간 상용 드론 등을 활용해 핵심 표적에 대한 정찰 능력과 타격 수단을 연동하는 등 드론 전투단을 편성하겠다는 것이다[3].

드론봇 군사연구센터와 드론 교육센터를 설립하고 드론봇 전투단을 창설했지만, 정책과 비교하면 국내 무인 전투기술 수준은 다소 부족한 실정이다. 지상 로봇과 항공 로봇의 기술 수준은 선진국에 대비 세계 7위 수준이고, 드론봇 업체도 대부분 영세하여 연구 인력이나 투자 예산도 부족하며, 주요 부품을 수입하여 활용하고 있다[4].

임무 수행이 가능한 드론 경진대회가 매년 개최되고 있다. MathWorks에서 주최하는 ‘미니드론 자율 비행 경진대회’, 육군본부에서 주최하는 ‘드론봇 챌린지 대회’, 방위사업청에서 주최하는 ‘전장 상황에서의 자율 비행 기술 경진대회’ 등이 있다. 드론봇 챌린지 대회의 경우 6개의 종목(원거리 정찰, 건물 내부 정찰, 공격, 다수비행체 제어, 수송, 수풀 지역 극복)에 대해 평가하며, 전장 상황에서의 자율 비행 기술 경진대회의 경우 자율 비행과 표적탐지, SLAM이 실시간으로 가능한 드론을 요구한다.

1.2.3. 과제의 필요성

드론봇 전투단 편성을 위해선 드론봇에 관한 기술 수준이 뒷받침되어야 하지만, 국내 무인 전투기술은 다소 부족하다. 그림4와 같이 드론은 악천후 시 운용, 적 전자 공격 회피/전자전 수행, 적R/D 피탐 및 회피, 침투하는 적 드론 탐지, 장기 체공 가능/이·착륙 공간 최소화, 저소음, 시설 내부 및 정밀 감시 표적 정보획득, 드론 통합관제/조종, 실시간 수집 영상 전송/분석체계 등 9가지 측면에서 제한사항이 있다 [5].

9가지 제한사항 중 시장 규모가 가장 큰 정보·감시·정찰·표적획득과 관련이 있는 '시설 내부 및 정밀 감시 표적 정보획득'을 해결하기 위해서는 GPS 및 통신이 불가능한 전장 상황에서 표적탐지 및 지형탐색이라는 정찰 임무가 수행 가능한 자율 비행 드론개발이 필요하다.

구 분	현재 제한사항	미래 요구능력
① 악천후시 운용	• 강우 5mm/h, 풍속 10m/s 미만시 운용	• 악천후시 운용 가능(강우 : 10mm/h, 풍속 : 24m/s) * 프로펠러 없는 전천후 드론 개발중
② 적 전자공격 회피/전자전 수행	• 항재밍안테나 미장착, 상용주파수 사용 • 전자전 수행능력 제한	• 소형 항재밍안테나 장착, 드론 전용 주파수 확보 • GPS없이 영상 및 센서 기반으로 자율비행 • 전자전드론 운용
③ 적 R/D 피탐 및 회피	• 적지종심작전지역에서 적 R/D 탐지에 취약 • 개전초 적 방공체계 표적식별 제한	• (고정익)스텔스 기술 적용(형태, 재질, 도색) • 기만드론을 운용하여 적 방공체계 표적 식별
④ 침투하는 적 드론 탐지	• 아군 저고도탐지레이더로는 적 소형드론 탐지 제한	• 적 소형드론(0.6×0.6m) 탐지 가능 • 드론 통합관제체계와 연동하여 피·아 드론 탐지 및 식별 가능
⑤ 장기 체공 가능 /이·착륙 공간 최소화	• (회전익드론)평균 30~50분 비행 * 잦은 배터리 교체로 간단없는 표적 감시 제한 • (고정익드론)이·착륙 공간 부족	• 하이브리드엔진(12시간 이상), 수소연료전지 (4시간 이상) 사용 • 수직 이·착륙형 드론 운용
⑥ 저소음	• 고도 100m 이견된 상태에서 전화벨 소리 크기로 인지(60db) * 적 인원에게 노출 가능	• 프로펠러 없는 저소음드론 운용 * 고도 100m 이격된 상태에서 소음청취 제한 (40db 이하)
⑦ 시설내부 및 정밀 감시표적 정보 획득	• 시설내부에서 통신 및 GPS수신 제한 • 시설내 장애물 산재시 비행이 제한 • 정밀감시표적에 근접접근 및 은밀감시 제한	• 시설내부 자율비행 및 3차원 지리공간정보 활용 • 비행 중 필요시 지상 이동(차륜, 궤도)하면서 운용 가능 • 조류·곤충 등을 모방한 생체모방형 초소형드론 운용
⑧ 드론통합관제/ 조종	• 드론 통합관제 제한 * 제대별 다양한 드론을 동시 운용시 충돌 가능성 내재 • 1:1 드론 조종	• 드론용 중앙방공통제소 구축 • 군집드론 알고리즘을 적용한 1:다수 드론 조종 가능
⑨ 실시간 수집영상 전송/분석체계	• 장거리·대용량 영상정보 전송 제한 • 전 출처 영상 분석체계 부재	• 통신중계드론을 운용하여 실시간 대용량 영상정보 송·수신 • 전 출처 영상을 실시간 융합·분석·처리체계 구축

그림 4 제한사항 및 미래 요구 능력

1.2.4. 과제의 창의성 혹은 실용성

본 과제의 창의성은 드론을 이용한 자율 임무 수행이다. 전 세계적으로 드론에 관한 관심은 증가하고 있지만, 드론을 이용한 자율 비행을 통한 임무 수행이라는 주제에 대한 접근 장벽은 높다. 시중에 판매되는 서적을 살펴보면 드론 제작과 관련된 서적은 다양하지만, 드론을 이용한 자율 비행, SLAM, 사물 인식 등을 동시에 다

루고 있는 서적은 전혀 없다. 본 활동의 보고서는 서적을 대신하여 드론을 이용한 임무 수행에 대한 기초지식을 제공할 수 있으며, 이러한 캡스톤 주제는 도전적이라고 할 수 있다.

본 과제의 실용성은 국방과학기술연구소(ADD)에서 주최하는 "전장 상황에서의 자율 비행 기술 경진대회"에서 같은 주제로 대회를 마련했다는 점에서 증명된다. 해당 대회에서 평가 지표로 다루는 '자율 비행 및 장애물 회피 실적 및 수행 경험' 등은 국방과학기술연구소에서 이러한 기술을 필요로 한다는 점을 보여준다.

2. 과제의 목표 및 내용

2.1. 목표

본 과제의 목표는 전장 상황에서 정찰 임무 수행이 가능한 드론개발이다. 본 과제의 전장 상황은 도심지 내의 전투로, GPS 및 통신이 불가하고 사전 정보가 없는 실내 환경이다. 이를 위해서 **1) 카메라를 통해 군인, 표지판, 활자와 같은 표적을 탐지**하고, **2) LiDAR 센서를 이용한 지형을 탐색**한다. 표적과 지형 데이터를 이용해 **3) 표적의 정보가 나타난 2차원 통합지도를 생성**하는 **4) 자율비행 드론**을 개발한다.

그림 5는 드론의 운용 개념도이다. 먼저 사용자가 조종기(TX)를 이용해 드론을 임무 수행 모드로 전환한다. 임무 수행 명령은 수신기(RX)-FC(Flight Controller)-PC 순으로 이동하며, PC는 임무 수행 프로그램을 실행한다. 드론은 1D LiDAR 센서를 이용해 얻은 고도 정보를 이용하여, 임무 고도까지 상승한다. 정찰 임무를 수행하면서 배터리 잔량 및 탐색 가능성을 바탕으로 임무 종료를 판단한다. 임무 종료를 판단하면, 최단 거리로 이륙 지점으로 복귀/착륙한다. 정지한 드론으로부터 USB로 통합 지도 확보/검증하여 사전 정보가 없는 실내 환경에 대한 정보를 얻을 수 있다.

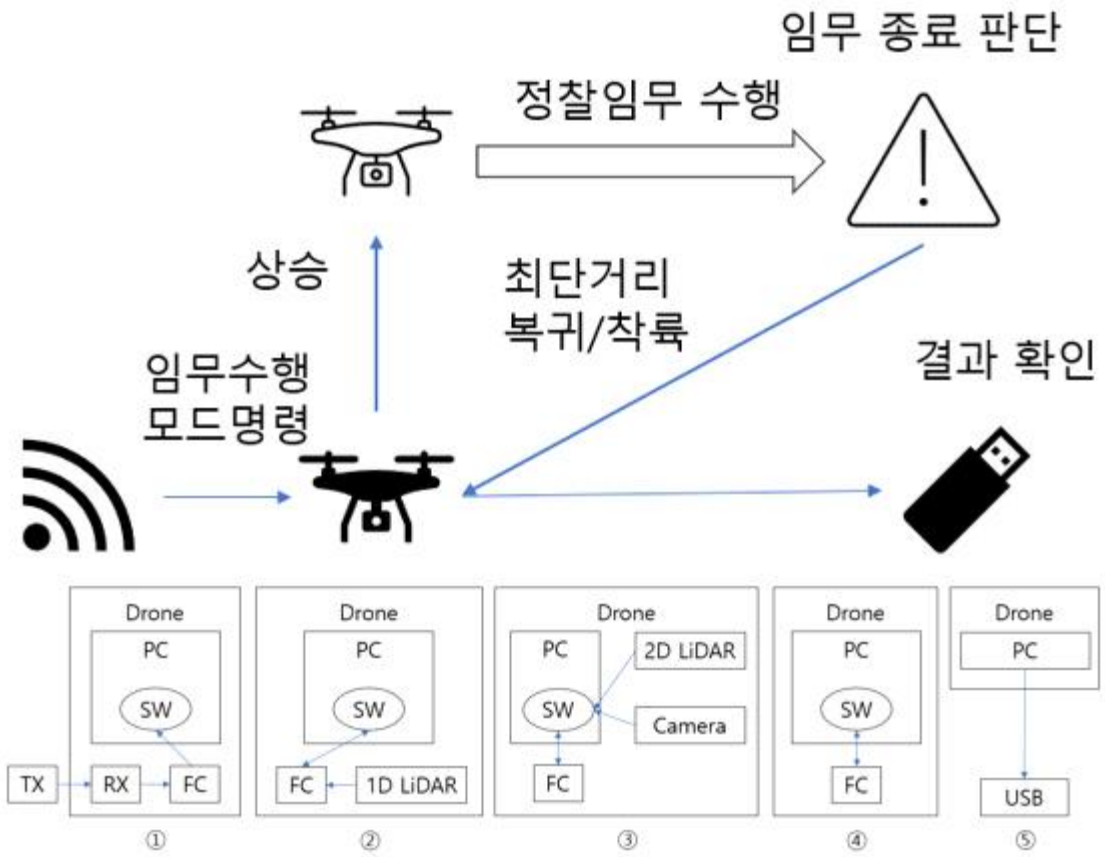


그림 5 운용 개념도

그림 6은 임무를 Context Diagram으로 표현한 것이다. 조종사는 TX(수신기)를 이용해 비행 모드를 전달한다. 비행 모드는 수동 비행 모드와 자율 비행 모드로 나뉜다. 자율 비행 모드로 임무를 수행하며, 예상치 못한 상황에 대비하기 위해 수동으로 드론을 조작할 수 있다. FC는 자율 비행 모드의 경우, 동작 정보(활성화)를 경로 비행 SW로 전달한다. 활성화 동작 정보는 Mapping SW를 걸쳐 Camera, 2D LiDAR까지 전달되어 센서와 SW들을 활성화한다. 수동 비행 모드의 경우, FC는 동작 정보(비활성화)를 전달하여 Mini PC와 Camera, 2D LiDAR를 동작시키지 않는다.

Mapping SW에서 VO(Visual Odometry)와 OD(Object Detection)은 Camera로부터 사진 정보를 받아, 궤적 정보와 표적정보를 Main 스레드에 전달한다. SLAM(Simultaneous Localization and Mapping)은 2D LiDAR로부터 지형정보를 전달받아, 지도 정보를 생성하고 Main 스레드에 전달한다. Main 스레드는 지도 정보를 실시간으로 경로 비행 SW의 Main 스레드로 전달한다.

경로 비행 SW에서 Main 스레드는 Mapping SW와 FC로부터 실시간 지도와 센서 데이터를 받는다. 경로 계획 스레드는 실시간 지도에서 탐색 영역과 미 탐색 영역을 구분하고, 미 탐색 영역에 대한 경로를 계획한다. 계획한 경로 데이터는 Main 스레드를 거쳐 제어모델로 전달된다. 제어모델은 경로 좌표와 센서 데이터를 바탕으로 FC로 제어 명령을 전달한다. FC는 제어 명령을 Motors로 전달하고, 1D LiDAR에서 받은 고도 정보와 내장된 IMU 센서로부터 얻은 센서 데이터를 경로 비행 SW로 전달한다. 경로 계획 스레드는 탐색 가능성과 비행시간을 비교하여 임무 종료를 결정하고, 출발점으로 최단 비행경로를 계획한다.

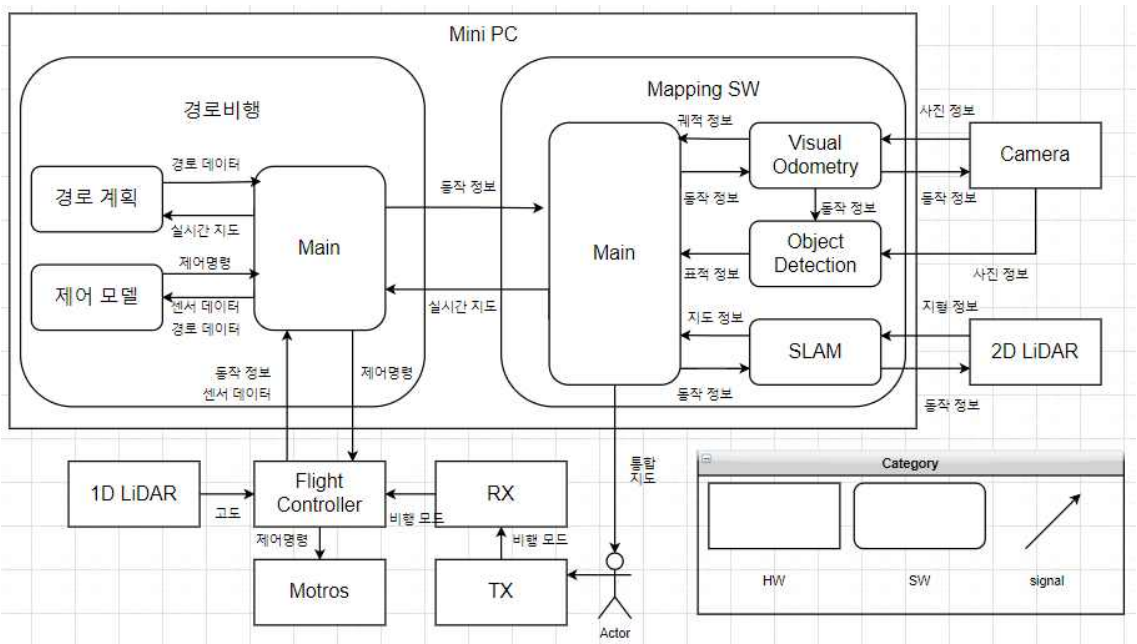


그림 6 Context Diagram

2.2. 연구/개발 내용

● **표적탐지**

객체 인식기술(Object Detection)이란, 수집된 이미지 또는 제공된 이미지에서 사전에 지정된 클래스의 객체 존재 여부 및 위치를 판단하는 기술로 그림 7과 같이 Object Classification, Generic Object Detection, Segmantic Segmentation, Object Instance Segmentation 등으로 분류된다[6]. 본 과제에서는 Bounding Box를 이용하는 Generic Object Detection을 목표로 한다.

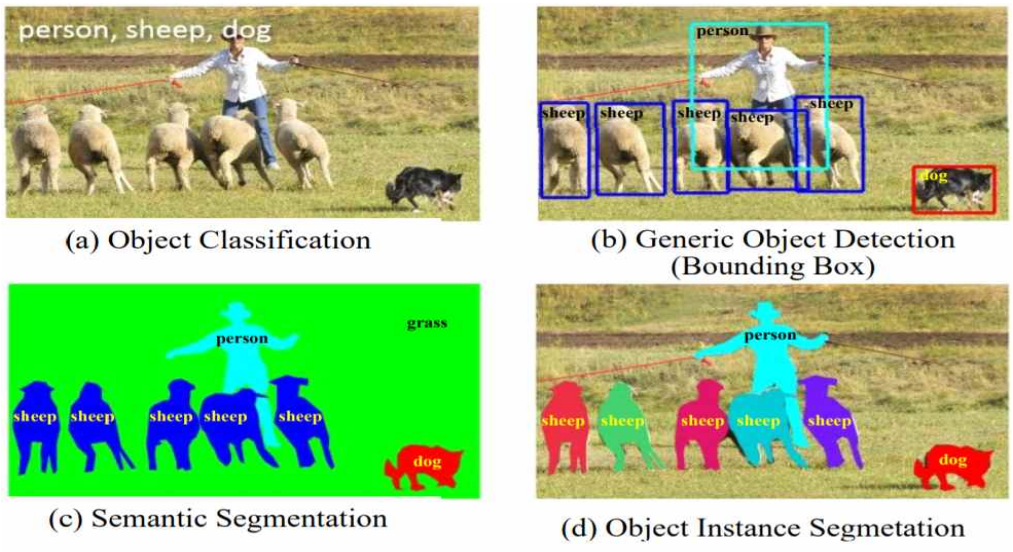


그림 7 객체 인식기술의 종류

객체 인식은 Deep learning 모델을 이용한다. 사전에 객체 정보와 Bounding Box가 Label 처리된 Dataset을 이용하여 학습한다. 그림 8와 같이 카메라 센서를 이용해 수집한 영상정보에서 군인, 표식, 활자 등의 객체 정보와 Bounding Box의 좌표를 인식한다. 스테레오 카메라 특성을 이용하여, Boundary Box의 Depth를 추출한다. 추출한 Depth를 이용해 표적의 객체 정보와 위치정보를 획득한다.

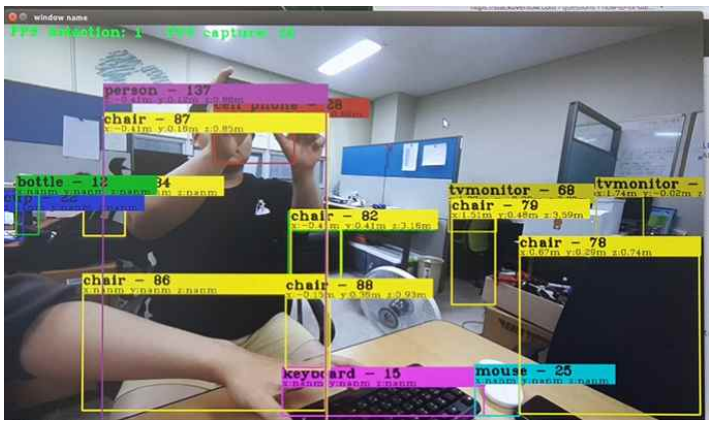


그림 8 Bounding Box, Depth, 객체 정보가 나타난 표적탐지 결과

● **지형탐색**

지형탐색이란 SLAM을 바탕으로 지형정보를 데이터화 하는 기술로, Odometry 와 Mapping으로 구분할 수 있다. Odometry란 드론의 궤적을 추정하는 기술이며, Mapping이란 지형정보와 추정궤적을 기반으로 공간에 대한 지도를 생성하는 기술이다.

본 과제에서는 카메라 센서를 이용한 Visual Odometry와 2D LiDAR 센서를 이용한 Mapping 기술을 구현한다. Visual Odometry는 사진의 특징점을 이용하여 위치를 파악하는 기술로, 드론의 출발점으로부터 드론의 이동 궤적을 계산한다. 2D LiDAR 센서를 이용해 드론에 대한 장애물의 상대위치를 계산하고 Odometry와 합하여 절대 위치에 지형정보를 표시한다. 이러한 과정을 통해서 만들어진 지도는 그림 10과 같이 나타난다. 생성된 지형정보는 자율 비행과 2차원 통합 지도생성에 사용된다.

● **통합지도 작성**

표적탐지와 지형탐색을 이용해 얻은 표적정보와 지형정보를 그림 10과 같은 통합한 지도를 작성한다. 지도에는 장애물, 궤적 그리고 표적이 나타난다. Visual Odometry를 이용해 드론의 궤적을 측정하고, Odometry를 기준으로 Stereo Camera로 반환받은 표적의 거리 데이터를 Mapping을 구현한다. Odometry에서 얻은 드론의 좌표 및 자세를 고려해, 표적탐지 기술에서 얻은 표적의 위치정보를 Mapping 지도에 표시한다.

동일 표적이 오차로 인해 인근 위치에 다수 표시될 가능성이 있다. 이 문제를 해결하기 위해 라벨이 같고 비슷한 위치에 있는 표적은 동일 객체로 판단하고 하나만 표시한다. 지형 데이터와 객체 탐지 데이터를 통합한 파일을 저장하고 사용자는 이 파일을 USB 혹은 네트워크를 통해 자신의 컴퓨터로 가져온다.

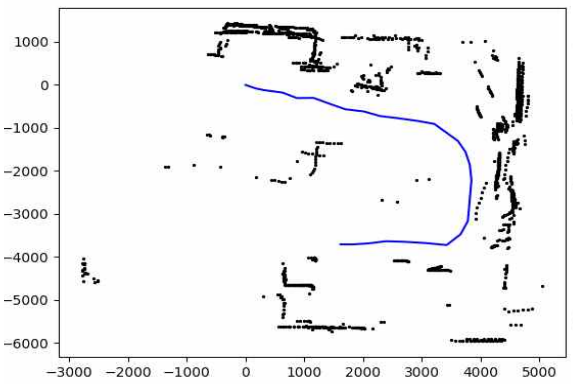


그림 9 지형지도(경상대학교 407-315)

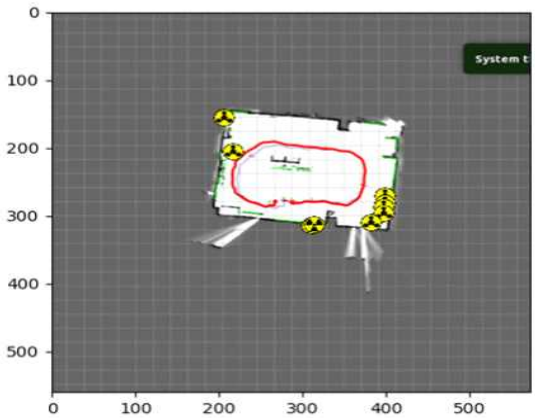


그림 10 통합지도(경상대학교 407-602)

● **자율 비행**

드론의 자율 비행 레벨은 그림 11과 같이 Level 0부터 Level 5까지 총 6단계로 구분할 수 있다. 본 과제에서 목표로 하는 자율 비행 레벨은 Level 2단계로 비행 임무는 드론이 제어하되, 예상치 못한 발생 시 수동제어로 조종사가 제어한다[7].

지형탐색 결과를 바탕으로 비행경로를 계획한다. 경로 계획은 깊이 우선 탐색 기법을 사용한다. 먼저, SLAM 알고리즘을 통해 탐색 완료된 지역과 탐색이 필요한 지역을 나누어 탐색이 요구되는 지역의 좌표값을 스택에 저장한다. 그 이후 스택에서 좌표값을 꺼내와 목표 지점으로 잡고, Flight Controller(FC)는 계획된 비행경로를 바탕으로 드론을 제어하여 경로를 따라 주행한다. 주행의 종료는 비행 성능에 기반을 둔 비행시간과 탐색 가능성을 바탕으로 드론이 결정한다.

만약 응급상황 발생 시, 사전에 설정한 수동제어 모드를 통해 드론의 비행을 운용자가 제어할 수 있도록 한다.

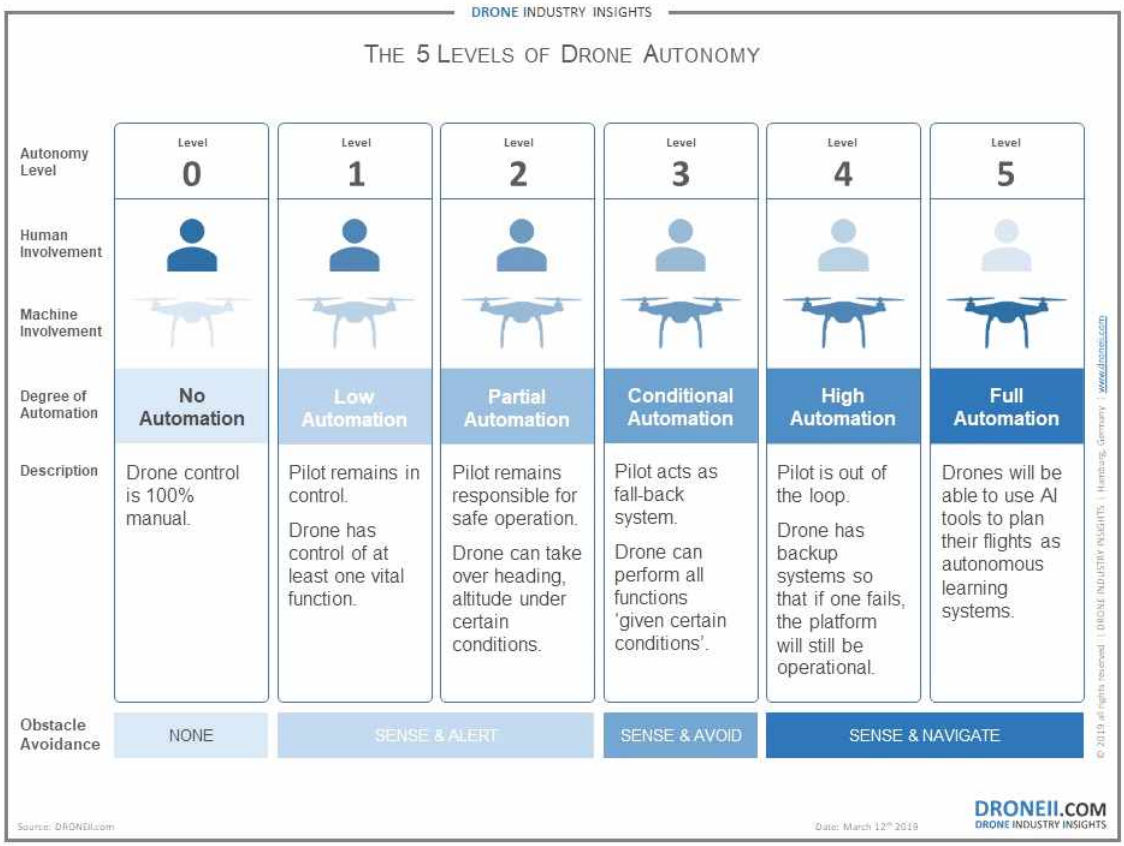


그림 11 드론 자율 비행 레벨

2.3. 과제의 기대 결과물

항목	예상 결과물	결과물에 대한 설명 요약
설계	드론 HW 설계	Requirement
		장비 규격 및 구성
		CATIA 3D 모델링
	SW 아키텍처 설계	Usecase Diagram
		Sequence Diagram
		Class Diagram
수학적 분석	드론 Specification	드론 비행 성능 계산
		동역학 모델링
	Mapping	지도 작성에 필요한 좌표 통일
	표적탐지	표적탐지기술 검증에 필요한 이론
컴퓨터 해석	표적탐지기술 검증	표적탐지 기술 평가
	제어기 설계	Matlab을 이용한 드론 PID 제어기
실험 및 제작	추력 시험	추력 시험 장치 제작 및 추력 성능 측정
	드론 제작	프로토 타입 제작
	초도 비행	비행 안정성 확인
	3D 프린팅	센서 설치를 위한 Mount 제작
	관성 모멘트 측정	Bifilar Pendulum을 이용한 실험
	표적탐지기술 구현	Darknet 플랫폼으로 모델학습
	지도작성 기술 구현	Cartographer 기술을 이용한 지도 작성
	Mapping SW 개발	표적탐지 기술과 지도작성 기술 융합
	경로 계획 SW 개발	지도작성 기술을 이용한 경로 계획 SW 개발
	프로토타입 시연	데모 시나리오에 따른 프로토타입 시연

표 1 예상 결과물

2.3.1. 설계

- **드론 HW 설계**
  시나리오 분석을 바탕으로 임무 수행에 필요한 드론의 Requirement를 작성한다. Requirement를 바탕으로 드론의 규격, 형상 및 사용 장비들을 결정한다.
- **SW 아키텍처 설계**
  HW 설계에서 작성한 임무 요구도를 바탕으로 Usecase Diagram, Sequence Diagram, Class Diagram을 작성하고, 아키텍처 구조를 완성한다.

2.3.2. 수학적 분석

- **드론 Specification**
  Requirement에서 결정한 장비들을 바탕으로 드론의 비행성능을 계산한다. Bifilar Pendulum 실험을 통해 관성모멘트를 측정하고, 드론에 대한 동역학 모델링 및 제어기 설계를 진행한다.
- **Mapping**
  2D LiDAR와 카메라를 이용해 드론에 대한 표적 및 장애물의 상대위치를 파악하고를 파악하고, 좌표계 변환을 통해 지도에 대한 절대 위치로 표시한다.

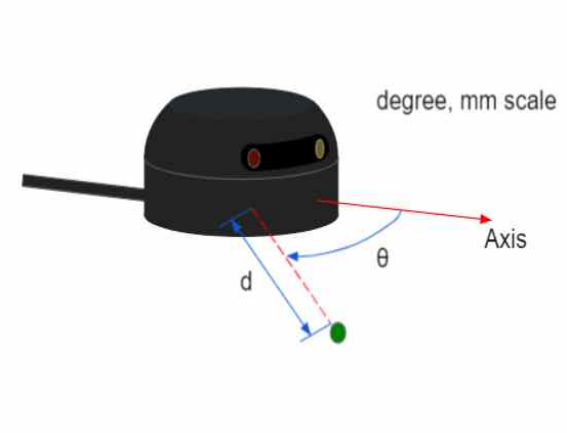


그림 12 2D LiDAR의 Cylindrical 좌표계

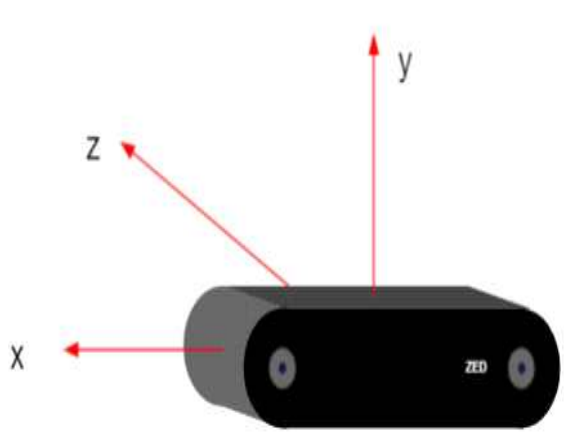


그림 13 Camera의 Cartesian 좌표계

- **표적탐지**
  평가 지표를 확인할 수 있는 도구인 Tensorboard를 사용하기 이전에 평가 지표들을 수식적인 접근으로 이해하며 분석한다. 특히 precision, recall 지표를 동시에 반영하는 mAP, F1 Score를 평가한다[16][17].

2.3.3. 시뮬레이션

- **표적탐지기술 검증**

  (1) 국방과학기술연구소(ADD)에서 제공한 Dataset을 이용하여 모델을 학습시킨다.

  (2) ADD에서 제공한 Validation Dataset을 이용해 모델의 표적탐지율을 mAP와 Macro F1 Score를 평가한다.

- **제어기 설계**

  수학적 분석에서 진행한 PID 제어모델을 바탕으로 Matlab Simulink를 이용하여 제어기를 설계한다. FC의 IMU data를 입력받아 FC로 명령할 수 있도록 설계한다.

2.3.4. 실험 및 제작

- **추력 시험**

  모터-프로펠러 조합의 정확한 추력을 알기 위해 추력 시험을 진행한다. 추력 시험대를 제작하여, Throttle(%)에 따른 추력 및 소모전류를 측정한다. 확보한 데이터를 비행 성능과 비교하여 장비를 보완한다.

- **드론 제작**

  드론 HW 설계에서 선정한 장비들을 바탕으로 드론을 제작하고, FC 플랫폼을 활용하여 드론의 Calibration 및 비행 모드를 설정한다. 초도 비행 이후 Mini PC, 2D LiDAR, Camera 센서를 부착한다.

- **초도 비행**

  Payload를 부착하지 않은 상태에서 드론 동작을 확인하기 위해 초도 비행을 수행한다. 무인항공기제어실험실에서 진행하며, Pitch, Roll, Yaw 방향 제어와 비행 모드(수동 조작, 호버링, 착륙) 기능이 정상 수행함을 확인한다.

- **3D 프린팅**

  CATIA를 바탕으로 3D 프린팅을 이용해 2D LiDAR, Mini PC, Camera를 부착할 수 있는 Mount를 제작한다.

- **관성모멘트 측정**

  드론의 PID 제어에 필요한 관성모멘트의 값을 얻기 위해, Bifilar Pendulum을 이용하여 관성모멘트를 측정한다. 측정한 관성모멘트를 수식 및 CATIA를 이용해 얻은 관성모멘트와 비교한다.

- **표적탐지 기술 구현**

  특정 표적의 클래스를 구분하기 위해 딥러닝을 이용하는 객체 인식 모델을 사용하여 표적탐지 기술을 구현한다. ADD에서 제공하는 데이터 셋과 직접 수집한 자료들을 사용하여 객체 인식 모델을 학습시킨다.

- **지도작성 기술 구현**

LiDAR 센서를 통해 지형 데이터를 얻고 구글 Cartographer 기술을 이용하여 지도작성 기술을 구현한다.

- **Mapping SW 개발**
  Cartographer를 통해 작성된 지도 데이터를 표적탐지 기술을 통해 반환받은 표적의 클래스 및 거리 정보를 표시한다.
- **경로 계획 SW 개발**
  실시간으로 생성되는 지도를 이용한다. 지도에 대해 탐색 가능한 미 탐색 영역으로 비행경로를 계획하고, 제어기 모델로 전달한다. 탐색 가능성과 비행 가능 시간을 고려하여, 임무 종료 시 출발점으로 최단 경로를 계획한다.
- **프로토타입 시연**
  제작한 프로토 타입을 이용하여 시나리오를 수행하고 영상을 촬영한다. 드론의 비행 영상, 드론의 촬영 영상, 통합지도 데이터를 확보한다.

2.4. 기대효과 및 활용방안

2.4.1. 기대효과

- **기술적 측면**
  본 개발 과제에서 표적탐지 기술을 통해 전장 상황에서 위험 요소인 군인(군인 특공, 파병군인, 여군, 여군특공 등), 표지(시작, 출구, 끝), 표지(방사능, 생화학)등을 식별한다. 지형탐색기술은 사전 정보가 없는 위험지역에 대하여 2차원 지도를 작성한다.
- **사회적 측면**
  기존의 드론은 운용을 위해 전문 인력이 요구되므로 부대 여건에 따라 사용이 제한된다. 본 드론은 운용을 위한 교육을 간소화하므로 운용 편의성을 살려 전투체계에 다양성을 추가할 수 있다. 또한 경진대회 및 학술회에 출전하여 기술 소개 및 교류를 통한 드론 기술 분야의 발전을 촉진한다.

2.4.2 활용방안

- SLAM, 표적탐지 각각의 기술들이 다양한 기술들과 융합하기에 용이하기 때문에 전체적인 프로젝트를 코드와 명세를 Github에 올려 형상관리 및 배포를 함으로써 향후 후배들의 캡스톤, 교내외 대회 및 KCC 학술대회 등 다양한 프로젝트에 활용되는 것을 기대한다.
- 전장 상황 이외에도 건물 혹은 터널의 붕괴, 테러상황과 복잡하고 위험한 선체 내부, 발전소, 동굴 탐사 등의 상황에서 인명 피해에 대한 부담 없이 정보 수집을 할 수 있다.

3. 연구 내용 및 결과

3.1. 주요 연구 내용 및 결과 요약

항목		진행내용	달성률(%)
설계	드론 HW 설계	Requirement	100
		장비 규격 및 구성	100
		CATIA 3D 모델링	100
	SW 아키텍처 설계	Usecase Diagram	100
		Sequence Diagram	100
		Class Diagram	100
수학적 분석	드론 Specification	드론 비행 성능 계산	100
	Mapping	지도 작성에 필요한 좌표 통일	
컴퓨터 해석	제어기 설계	동역학 모델링 및 제어기 설계	50
	표적탐지기술 검증	표적탐지 기술 분석 및 평가	100
제작 및 실험	추력 시험	추력 시험 장치 제작 및 추력 성능 측정	80
	드론 제작	프로토타입 제작	100
	초도 비행	비행 안정성 확인	100
	3D 프린팅	센서 설치를 위한 Mount 제작	100
	관성 모멘트 측정	Bifilar Pendulum을 이용한 실험	0
	Jetson Xavier 보드 부팅	기술 구현을 위한 보드 세팅	100
	표적탐지기술 구현	Darknet 플랫폼으로 모델학습	100
	지도작성 기술 구현	Cartographer 기술을 이용한 지도 작성	100
	Mapping SW 개발	표적탐지 기술과 지도 작성 기술 융합	87
	경로 계획 SW 개발	지도 작성 기술을 이용한 경로 계획 SW 개발	0

표 2 연구 내용 달성률

3.1.1. 설계

● 드론 HW 설계

(1) 요구사항

시나리오를 바탕으로 설계한 Requirement는 표 3과 같다

HW		SW	
총 중량	3000g	**운용환경**	arm64
형상	쿼드콥터, X자 형	OS	Jetpack Ubuntu 18.04
크기	600급	**Memory**	4GB 이상
운용고도	지면 기준 100cm	**Stroage**	SD Card 64GB
운용시간	5 min	**사용 언어**	Python3, C++
탐색거리	20 m		

표 3 Requirement

드론은 멀티콥터의 한 종류로 모터의 수에 따라 트라이콥터, 쿼드콥터, 헥사콥터 등으로 나뉜다. 멀티콥터 중 가장 흔한 형상은 쿼드콥터이다. 모터 수가 홀수인 경우, 작용 반작용으로 발생하는 기체 회전을 고려한 설계가 필요하다. 모터가 짝수인 경우, 작용 반작용 문제가 해결된다. 모터가 2개인 경우는 작용 반작용을 해결하지만 자세 제어 문제가 남는다. 쿼드콥터부터는 자세 제어 문제 또한 해결된다. 헥사 등 모터 수가 많은 경우 안정적인 비행이 가능하지만, 모터 수에 따른 비용증가와 소모전류 증가 등의 문제가 발생한다. 따라서 설계 구조가 가장 간단하고, 부수적인 문제가 가장 적은 쿼드콥터 형상으로 설계했다.

드론은 그림 14과 같이 운동 축 방향에 따라 X 형과 + 형으로 구분할 수 있다. X 형 드론은 피치/롤 조종 중 하나만 조종하여도 모터 전체의 출력이 변화하여 빠른 반응속도와 안정적인 비행이 가능하다. 반면에 +드론의 경우 피치/롤 중 하나만 조종하는 경우, 한 쌍의 모터 출력만 변화하여 반응이 X 형보다 느리다. 비행 제어의 안정성을 확보하기 위해 X 형으로 설계했다.

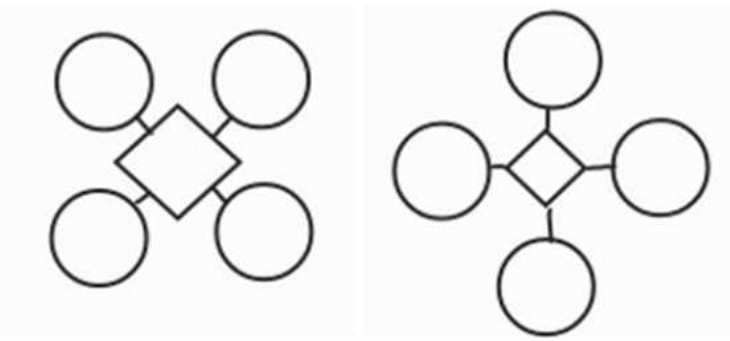


그림 14 X 형 드론(좌측), + 형 드론(우측)

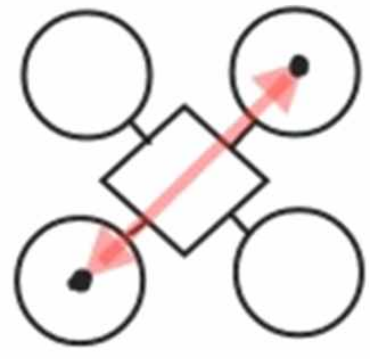


그림 15 Wheel Base Length

Wheel Base Length(WBL)은 그림 15과 같이 대각선상에 있는 모터의 축간거리로 크기를 가늠하는 척도이다. 드론의 중량이 클수록 긴 프로펠러를 사용해야 하고, 긴

프로펠러는 WBL의 길이를 증가시킨다.

본 드론은 총 중량이 3,000g으로 출력이 50~70% 상태에서 모터 하나당 750g의 추력을 담당해야 한다. 그림 16은 드론용 모터의 추력 vs 프로펠러 길이 통계이다. 본 자료를 바탕으로 750g의 추력을 위해 필요한 프로펠러는 12-15inch임을 확인했다. 프로펠러와 드론 부품 및 payload의 충돌을 방지하고, payload의 배치 공간을 확보하기 위해 프레임을 600급 이상으로 설계했다.

쿼드콥터 형상, X형, 600급 이상이라는 조건을 만족하는 드론 프레임으로 Tarot IRON-MAN Sport 650 선정하였다.

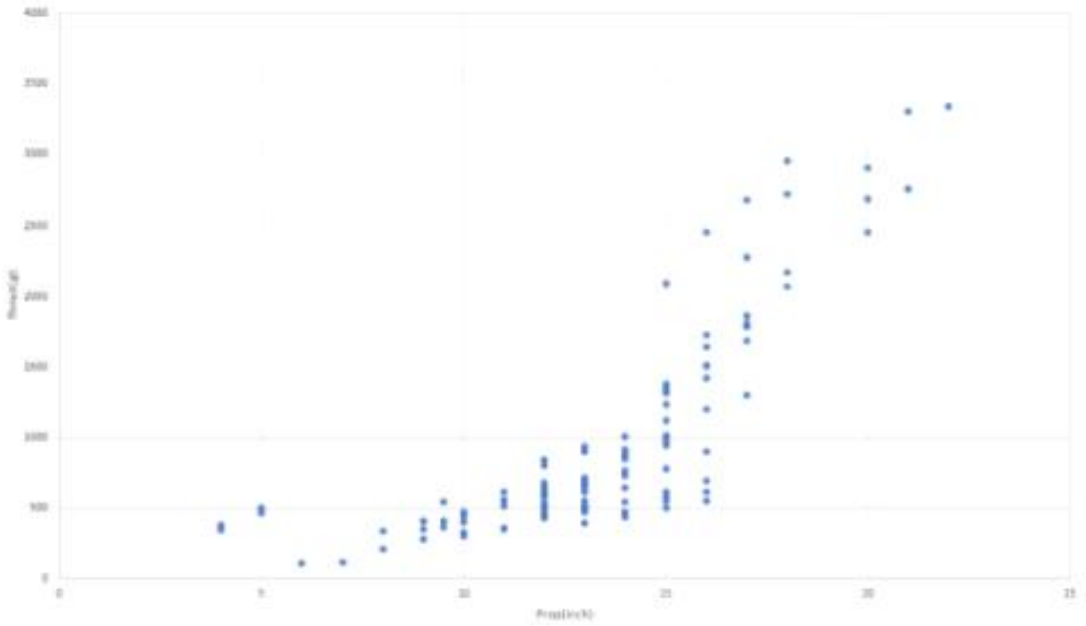


그림 16 Thrust-inch diagram

(2) 장비 선정 및 3D 모델링 - Payload

드론에서 payload란 기체에 탑재 가능한 센서 혹은 화물 등을 의미한다. 본 기체에 임무 수행을 위해 탑재하는 payload는 카메라 센서, 2D LiDAR 센서, 1D LiDAR 센서, Flight Controller, Mini PC 등이 있다.

표적탐지 임무 수행을 위해 사용하는 카메라 센서로 ZED 2 Stereo Camera를 선정하였다. 카메라는 드론의 정면에 배치되어 비행 동안 촬영을 진행한다. 중량은 Data Sheet 기준 166g으로 무게 중심에 위치하지 않기 때문에 모멘트를 유발한다. 이는 배터리 위치 조정을 통해 바로잡을 계획이다. 표적탐지를 위한 YOLO V5 모델을 이용해 촬영 영상 속 표적을 탐지하여 표적정보와 위치정보를 추출하며, 추출된 정보는 통합지도 작성 시 사용된다. ZED 카메라의 3D 모델은 업체에서 제공되는 파일 그림 17을 이용했다[8].

지형탐색 임무 수행을 위해 사용되는 2D LiDAR 센서로 RPLIDAR A3를 선정하였다. 2D LiDAR 센서는 드론의 중앙, 정상에 위치하여 내부 구조물 및 프로펠러로 인한

시야가 가려짐을 방지한다. 이 같은 위치에 배치하기 위해, 3D 프린팅 구조물 설계를 통한 마운트 설계가 필요하다. 2D LiDAR 센서로 수집한 정보를 바탕으로 Cartographer라는 2차원 SLAM 모델을 이용하여 지도를 생성한다. 생성한 지도는 드론의 비행경로 계획과 통합지도 작성에 사용된다. RPLIDAR A3의 3D 모델은 제조사에서 제공하는 파일 그림 18를 이용했다[9].

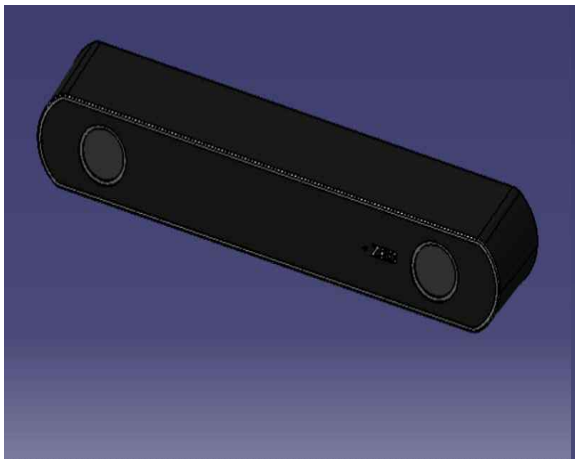


그림 17 ZED 2 3D Model

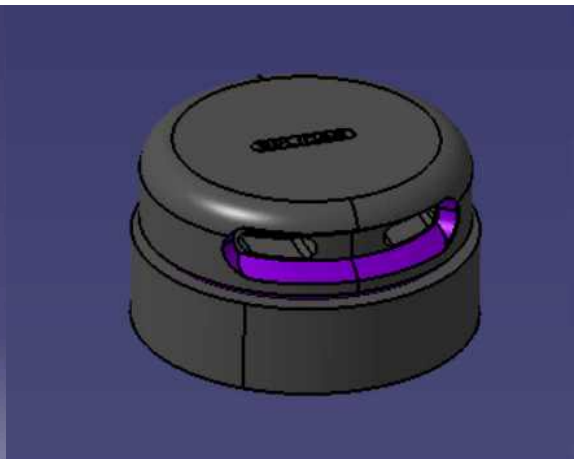


그림 18 RPLiDAR A3 3D Model

1D LiDAR 센서는 드론의 고도 측정에 사용하기 위해 드론의 하단에 배치된다. 드론의 하단 중심부에는 배터리가 위치하므로, 드론의 중심부가 아닌 배터리 Panel과 Chassis 사이, 배터리 옆에 위치한다. 1D LiDAR는 Pixhawk와 호환되는 센서로, 수집한 데이터는 Pixhawk로 전달되어 PID 제어에 필요한 Z축 데이터로 사용된다. LiDAR lite v3의 3D 모델은 제조사에서 오픈 소스 그림 19을 사용하였다[10].

Mini PC는 NVIDIA Xavier NX를 사용한다. 형상은 103mm x 90.5mm x 31mm로 모든 장비 중 가장 많은 면적을 차지한다. 고려할 형상으로 포트와 방열팬이 있다. 카메라 센서와 LiDAR센서의 케이블이 가로막히지 않도록 포트 앞을 다른 장비로 가로막아서는 안 된다. 방열팬이 상단을 향해 위치해 있어, 제 기능을 발휘하기 위해 2D LiDAR 센서는 방열팬과 거리를 유지해야 한다. NVIDIA Xavier NX의 3D 모델은 NVIDIA에서 제공하는 파일 그림 20을 사용했다[11].

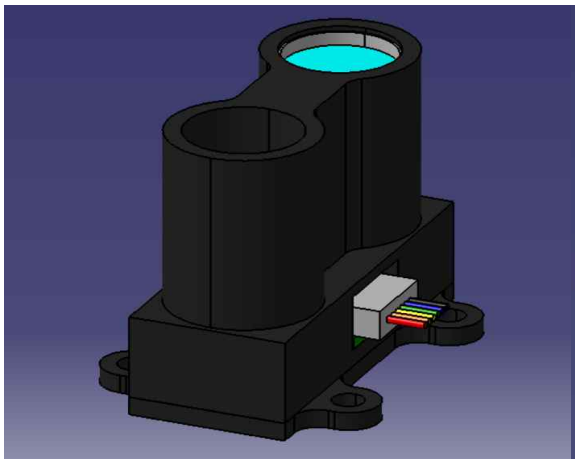


그림 19 LiDAR lite V3 3D Model

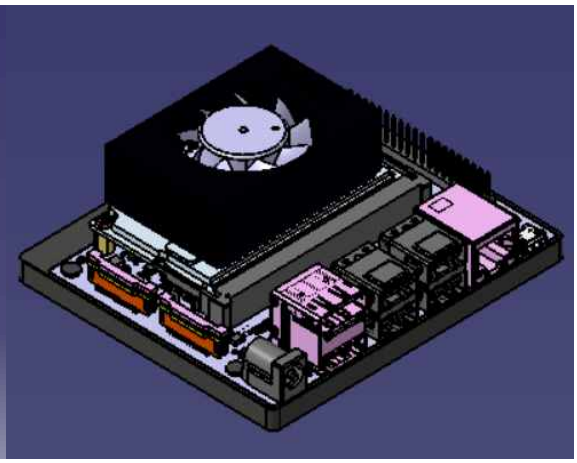


그림 20 NVIDIA Xavier NX 3D Model

Mount는 그림 21와 같이 Mini PC와 카메라, 2D LiDAR를 장착할 수 있도록 설계한다. Camera와 LiDAR의 도면, Mini PC의 3D 모델을 이용했다. 1층과 2층을 구분하여 설계하며, 1층은 Mini PC와 카메라, 2층은 2D LiDAR 센서를 설치한다. 설치를 위해, M3 볼트와 너트를 사용하였다. Mount는 그림 22와 같이 3D 프린팅으로 제작한다. 3D 프린팅의 방법은 FDM(Fused deposition modeling), SLA(Stereolithography), DLP(Digital Light Processing) 방식 등이 있다. FDM 방식으로 제작하기 위해 공차는 0.2~0.3mm로 설정하였고, 무게 및 재료 절감을 위해 불필요한 부위는 제거하였다.

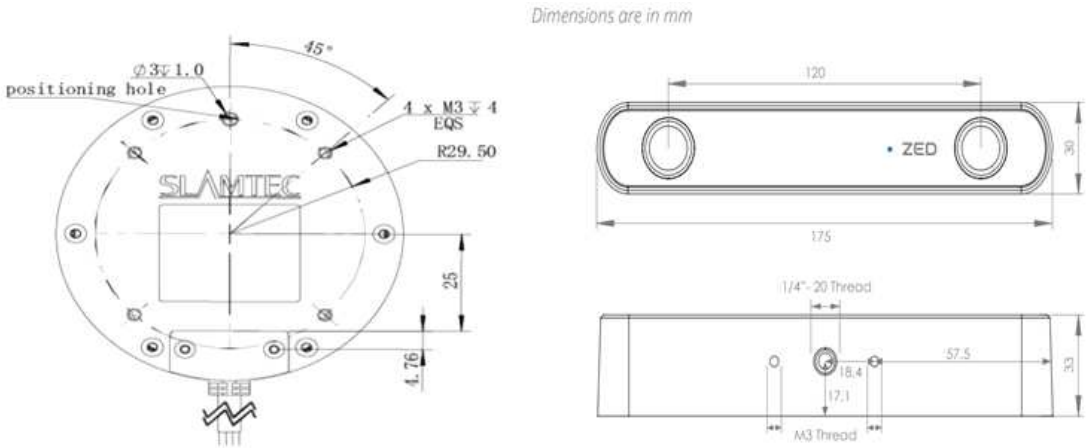


그림 21 2D LiDAR와 Camera의 도면

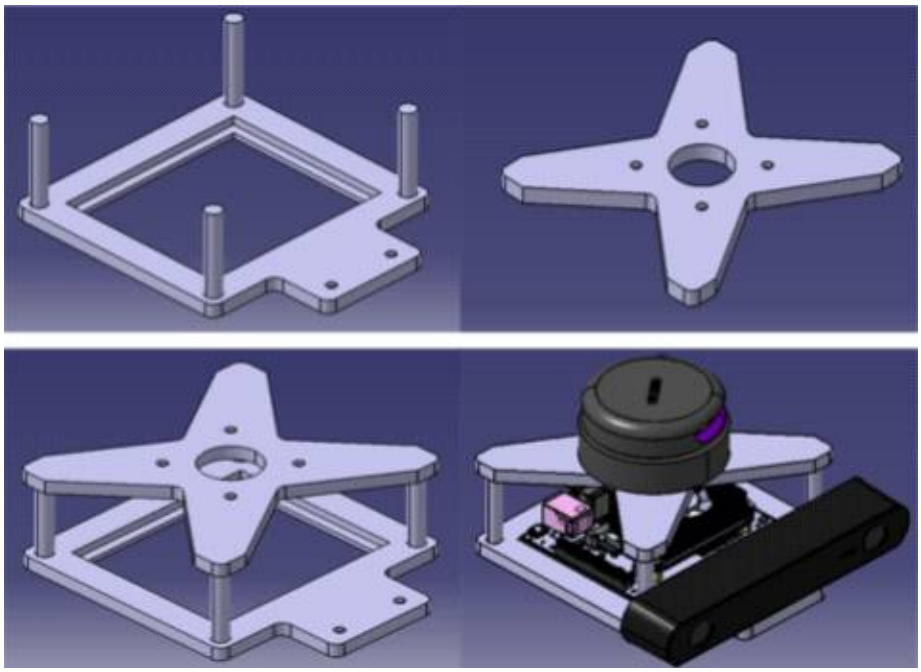


그림 22 Mount

드론에서 Empty Weight이란 Payload를 제외한 무게로 Flight Controller, 수신기, 모터, ESC, 배터리, Frame 등이 있다. FC는 Pixhawk 4 Mini를 사용한다. FC는 Mini PC에서 얻은 비행경로를 바탕으로 드론의 자세 및 동작을 제어한다. 사용하는 Pixhawk 4 Mini는 기본형인 Pixhawk 4에서 불필요한 포트들을 제거한 장치로, 더 가볍고 크기가 작다. FC는 PID제어에 이용될 데이터들이 수집되는 IMU센서가 내장되어 있으며, 드론의 앞과 뒤를 구분하는 장치이다. 센서값의 오차를 최소화하기 위해 드론의 중앙에 배치해야 하며, 일반적인 드론은 Hub Plate 위에 배치한다. 본 과제에서는 Mount 등의 장비들이 부착되므로 Chassis의 하단에 양면테이프를 이용해 부착한다. Chassis 하단에 부착하는 경우, Pixhawk의 설정을 위한 USB 포트 사용이 공간적으로 제한된다. PC와 연결하기 위해 외장형 USB 또는 Telemetry를 사용하는 방법이 있으며 본 과제에서는 Telemetry를 사용하였다.

(3) 장비 선정 및 3D 모델링 – 구조물

Pixhwak 4 Mini는 SD카드 돌출부를 제외한다면 직육면체 모양이므로, 직육면체 형상으로 설계하였다. Pixhawk의 3D 모델은 제공되지 않기 때문에 그림 23의 도면을 바탕으로 CATIA를 이용해 3D 모델링을 진행하였다[12]. Pixhawk와 함께 고려하는 장비로 Pixhawk Power Module이 있다. Power Module은 이후 언급할 ESC의 BEC(Battery Elimination Circuit) 기능과 관련이 있다. ESC는 OPTO Type과 BEC Type으로 구분할 수 있다. BEC는 모터의 전압이 수신기가 사용할 수 있도록 낮춰주는 기능이며, OPTO는 BEC기능을 지원하지 않는 변속기이다. 드론의 경우 BEC 타입의 변속기를 사용하는 경우, 변속기가 4개가 부착되기 때문에 고장을 방지하기 위해 3개의 변속기는 BEC를 제거해주거나 OPTO 타입의 변속기와 함께 외장 BEC를

따로 장착한다. Power Module은 언급한 외장 BEC 기능을 수행하므로 수신기를 위한 별도의 BEC가 필요하지 않다[13].

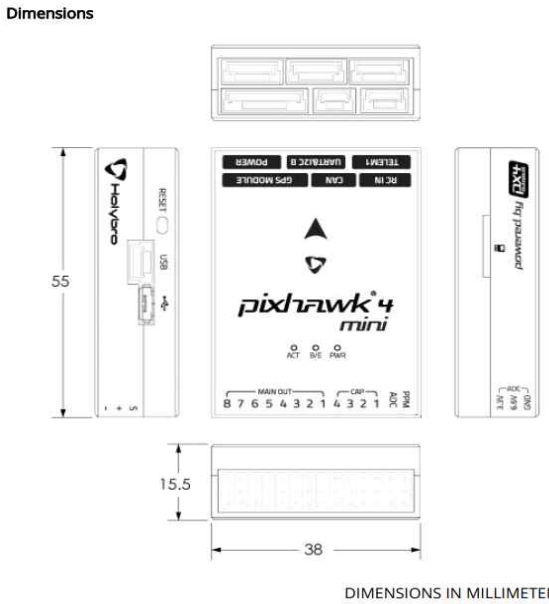


그림 23 Pixhwak 4 Mini 도면

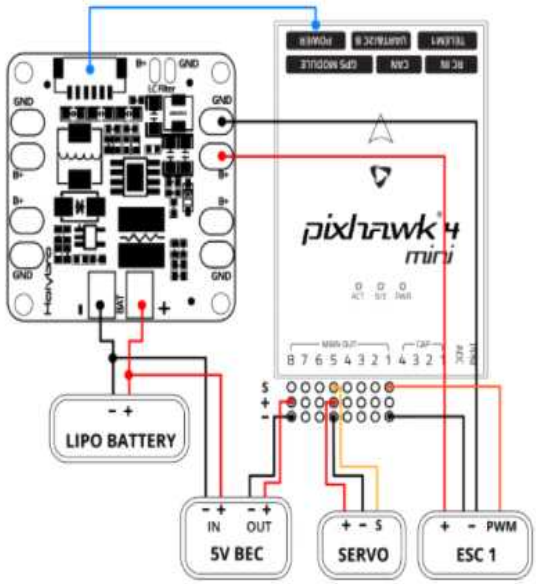


그림 24 Pixhawk 4 Mini 배선도

수신기는 RX601을 선정하였다. 쿼드콥터에 대해 Pixhawk와 수신기가 연결되기 위해서는 5개 이상의 채널이 지원돼야 한다. 본 수신기는 6개의 채널을 지원하며, 보유하고 있는 조종기 devo 7과 호환되는 수신기이다. 수신기는 드론의 Hub Plate와 Chassis 사이에 배치된다. RX601의 3D 모델은 제공되지 않기 때문에, 실측값을 바탕으로 CATIA를 이용해 3D 모델링을 진행하였다. Pixhawk 4 Mini와 호환되는 수신기의 종류로는 Spektrum/DSM 수신기와 PPM 수신기가 있다. RX601은 지원되지 않는 PWM 수신기이므로 PPM Encoder가 필요하다. PWM(Pulse Width Modulation)과 PPM(Pulse Position Modulation)은 수신기와 FC 사이의 무선 프로토콜 방식이다. PWM은 일정 간격 동안 들어온 신호의 Width를 검출하는 방식으로, 선 하나당 하나의 채널만 이용할 수 있다. PPM은 일정 간격 동안 들어온 신호의 위치를 검출하는 방식으로, 하나의 선으로 여러 개의 채널을 사용할 수 있다. 그림 25은 PWM 신호와 PPM신호가 어떻게 전달되는지 보여준다[14]. RX601의 채널 ELEV, AILE, THRO, RUDD, GEAR, AUX1을 채널 1번부터 6번으로 정의하자. PWM 프로토콜을 사용하는 RX601과 모터를 직접 연결하는 경우, 1번부터 4번까지 총 4개의 채널을 사용하게 된다. 이 상태에서 조종기를 조작하는 경우, Throttle 신호에 대해 모터 4개가 모두 동작하는 대신, Throttle 채널에 해당하는 모터 1개만 동작한다. 이는 단발기 형태의 항공기나 헬기에 사용할 수 있지만, 드론에 사용하는 데 무리가 있다. Pixhawk용 플랫폼은 Pixhawk 운용을 위해 5개 이상의 채널이 필요하다. RX601의 AUX1을 제외한 모든 채널에 케이블을 연결하여 5개의 채널을 확보했다. 그림 26은 이를

설명한다.

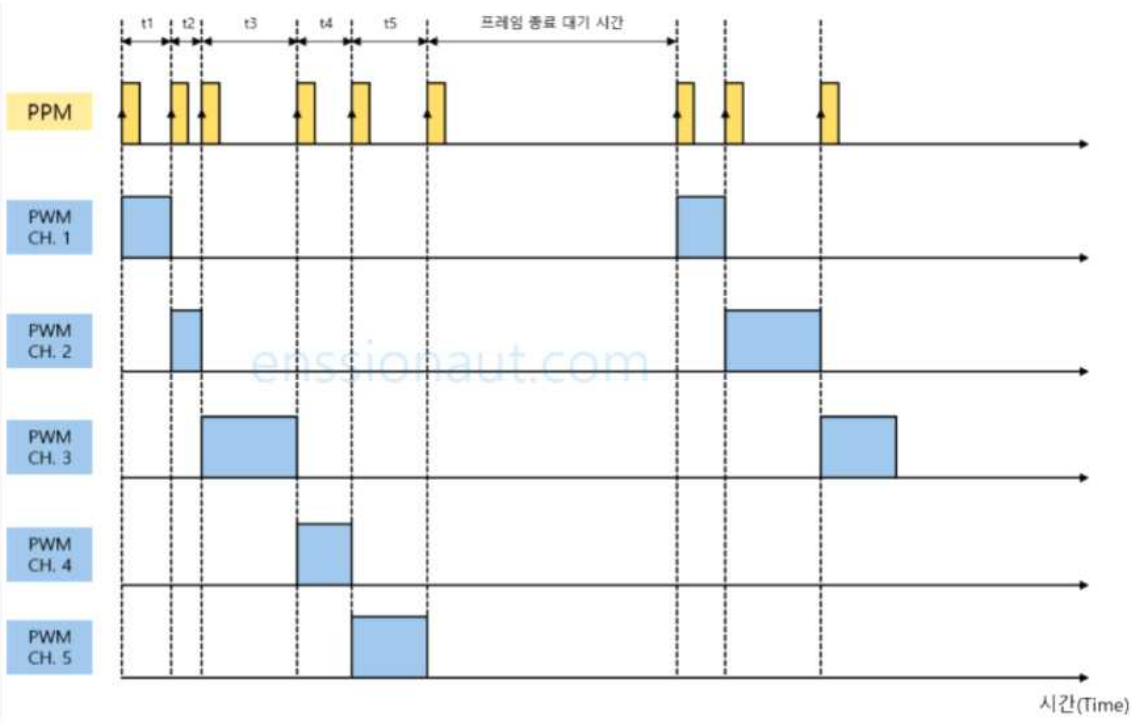


그림 25 PWM 신호와 PPM 신호

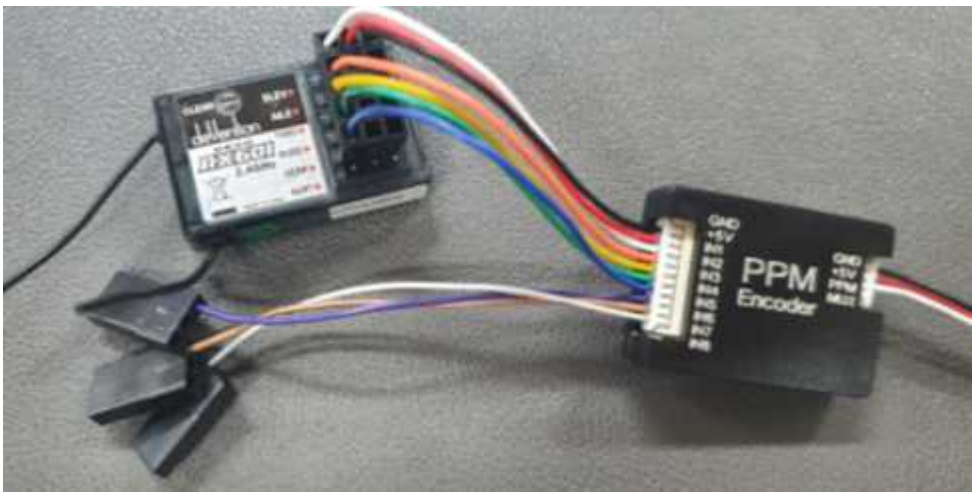


그림 26 PPM Encoder와 RX601

모터는 Requirements를 만족시키는 T-motor 사의 MN3508 700kV로 선정하였다. 프로펠러를 샤프트에 고정하는 방식은 어댑터 방식과 스크류 고정 방식을 지원한다. 어댑터 방식이란, 샤프트에 어댑터와 프로펠러를 고정한 뒤 어댑터를 조여 고정하는 방식이며 프로펠러의 중심부를 두껍게 설계하는 플라스틱 프로펠러에 많이 쓰인다. 스크류 고정 방식은 모터에 나사를 이용해 고정하는 방식으로 카본 재질의 얇은 프로펠러에 많이 쓰인다. 플라스틱 프로펠러는 비행 중 파손이 잦아 여유분이 필요하므로, 카본 프로펠러를 선정하였다. 그림 27는 MN3508의 Testing Data이다

[15]. 14.8V의 배터리를 사용하는 경우 12인치의 프로펠러를 사용한다면 출력의 50~65% 사이에서 호버링 조건을 만족함을 알 수 있다. 모터의 추력은 Testing Data와 오차가 크므로 추력 시험이 필요하다. 추력 시험은 Load Cell을 이용하여 시험을 진행한다. MN3508 700kv의 3D 모델은 오픈소스 그림 28을 사용했다[16].

Voltage (V)	Prop	Throttle	Current (A)	Power (W)	Thrust (G)	RPM	Efficiency (G/W)	Operating Temperature (°C)
14.8	T-MOTOR 11*3.7CF	50%	3.1	45.88	380	5300	8.28	41
		65%	5.6	82.88	630	6500	7.60	
		75%	7.9	116.92	780	7200	6.67	
		85%	10.5	155.40	960	8000	6.18	
		100%	12.7	187.96	1110	8500	5.91	
	T-MOTOR 12*4CF	50%	3.8	56.24	460	4700	8.18	48
		65%	7.4	109.52	800	6300	7.30	
		75%	10	148.00	1000	6900	6.76	
		85%	13.5	199.80	1200	7500	6.01	
		100%	16.1	238.28	1360	8100	5.71	

그림 27 MN3508 700kv Testing Data

ESC는 XRotor Pro 40A를 선정하였다. OPTO 타입의 변속기로 BEC기능을 지원하지 않는다. OPTO 변속기의 경우 BEC기능을 하는 선이 접지선 역할을 하게 되는데, 이는 신호의 잡음을 제거하는 역할을 한다. 본 변속기는 Version A와 Version B가 존재한다. Version A는 소켓이 연장된 전선에 달린 형태이고, Version B는 소켓이 변속기에 직접 붙어있는 형태이며, 본 장치는 Version A이다. ESC는 드론 Arm Pipe 하단에 배치하고 케이블타이를 이용해 고정한다. 변속기의 3D 모델은 오픈소스 그림 29를 활용했다.

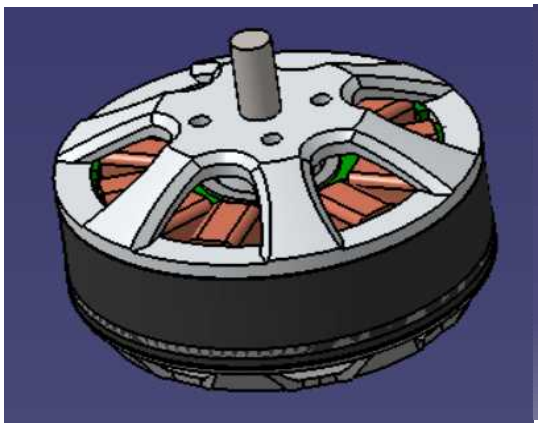


그림 28 MN3508 3D Model

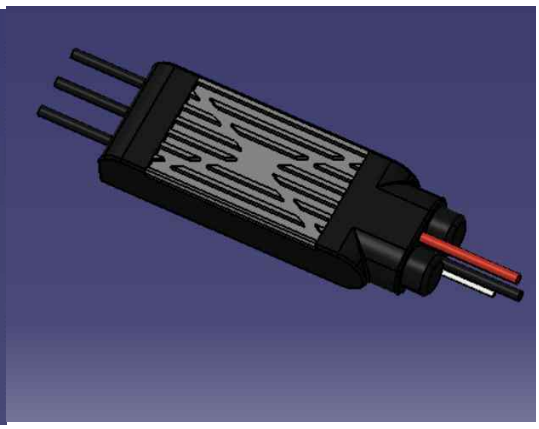


그림 29 XRotor Pro 40A 3D Model

배터리는 Dinogy Graphene 14.8V 7200mAh 4s 80c로 선정하였다. 배터리는 드론의 무게 중심 위치를 결정하는 데 가장 중요한 장비이다. 위치 조절이 가능한 장비 중 가장 무겁고, 가장 아래에 배치되므로 드론의 무게 중심 및 안정성을 조절을 함께 담당한다. 배터리는 Specification에 따르면 136mm x 45mm x 42mm, 528g의

직육면체로 3D 모델링을 진행하였다. 본 배터리는 트랙사스 타입의 커넥터를 지원하고, Power Module은 XT60 커넥터를 지원하므로 그림 30의 XT60 컨넥터 → 트랙사스 컨넥터 변환 잭을 사용한다. 배터리가 있어야 하는 장비는 FC와 Mini PC이다. 본 과제는 하나의 배터리로 모든 전원을 공급하기 때문에, 하나의 전원을 두 장치에 공급할 수 있는 그림 31의 XT60 병렬 컨넥터 케이블 Type 2를 사용한다. NX는 전원 입력 타입이 M5.5 x 2.5이므로 DC 플러그 그림 32의 NT10-AF35와 DC잭 변환 젠더를 사용한다. DC 플러그는 F5.5 x 2.1 타입의 잭을 갖고 있으므로 M5.5 x 2.5를 사용한다. 병렬 커넥터 케이블 중 하나를 DC 플러그와 결합하여 사용한다.

그림 30 XT60 컨넥터 → 트랙사스 컨넥터

그림 31 XT60 병렬 컨넥터 케이블 Type 2 변환 잭

그림 32 DC잭 변환 젠더, NT10-AF35, 결합 모습

선정한 Frame은 Hub Plate, Chassis, Arm, Panel, 착륙장치로 나뉜다. Hub Plate는 Payload와 같은 장비들이 배치되는 공간으로 2D LiDAR 센서, 카메라 센서, Mini PC가 부착된다. Chassis는 Hub Plate 아래층으로, Hub Plate와 함께 Arm을 고정한다. Chassis 와 Hub Plate 사이에는 Power Module이 배치되고, Chassis 밑에는 FC, Telemetry, 수신기 조합이 배치된다. Arm은 Arm Copter Attachment와 Arm Pipe, 모터마운트로 구성된다. ESC가 모터 마운트 아래에 배치된다. Panel은 배터리가 배치되는 Panel과 Panel-Chassis를 연결해주는 Pendant 및 Pipe, Mount로 구성된다. Mount는 Pipe를 Chassis 밑에 고정하고, Pipe는 Pendant를, Pendant에 Panel이 고정된다. 드론 프레임 제품들은 도면이 제공되지 않기 때문에 실측값을 바탕으로 3D

모델링을 진행하였다. 그림 33은 3D 모델 데이터를 어셈블리 기능을 이용해 조립한 형태이다.

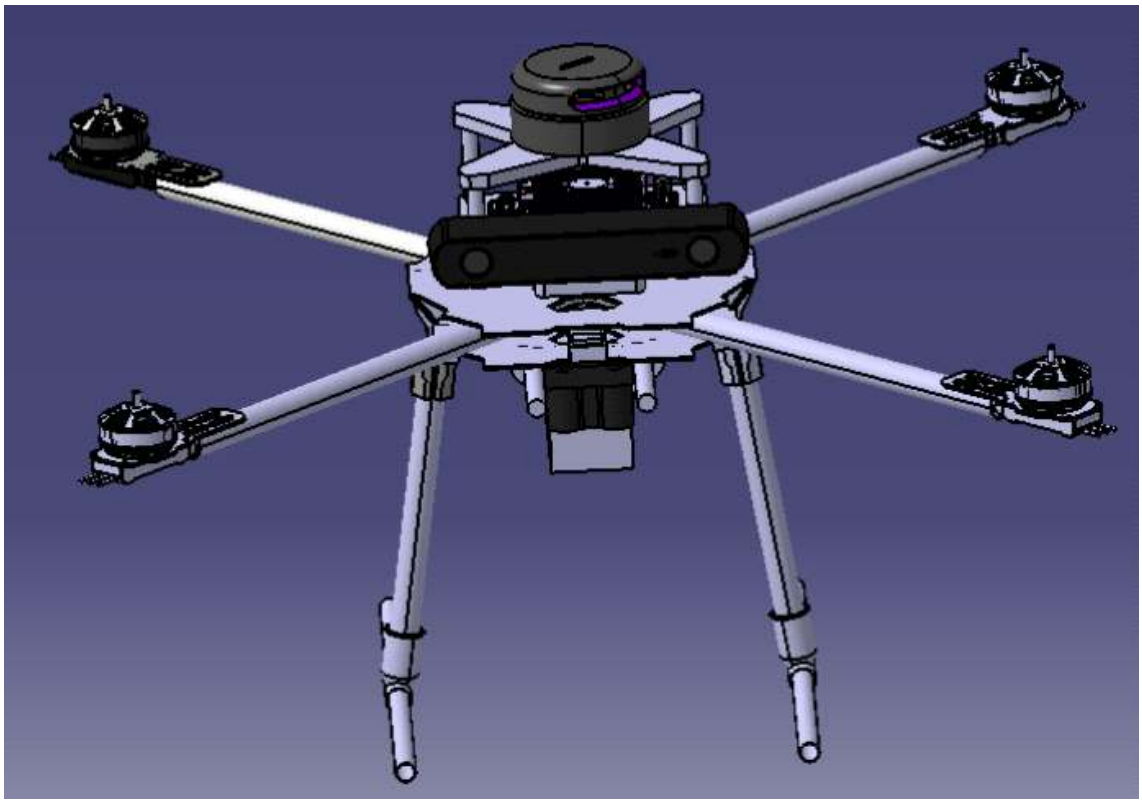


그림 33 최종 형상

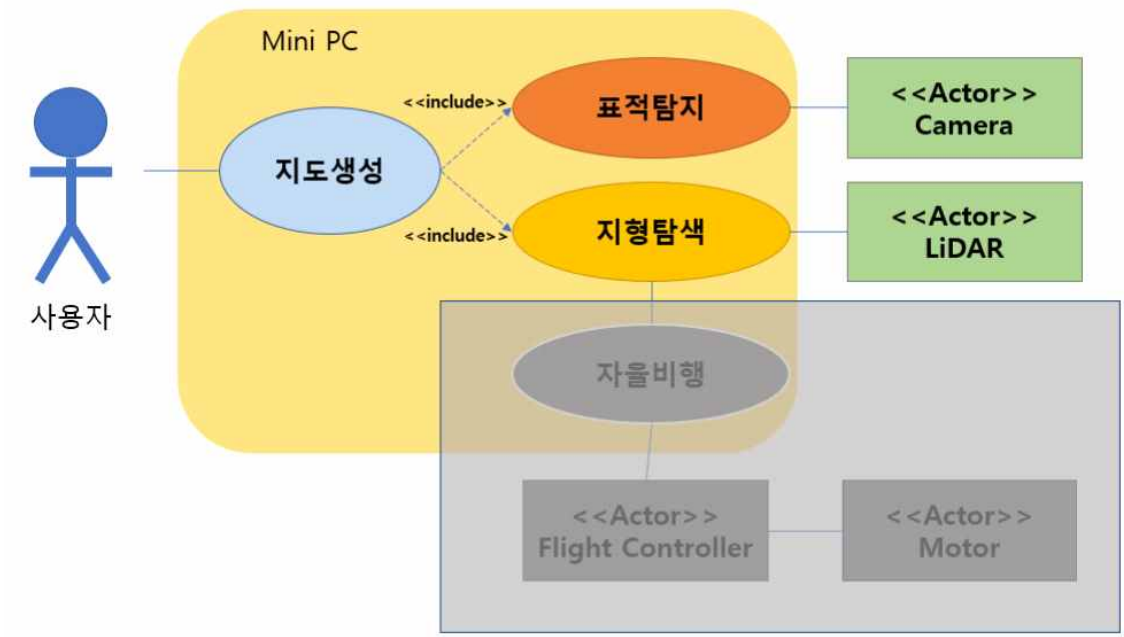


그림 34 유스케이스 다이어그램

S/W 아키텍처 설계

그림 34는 유스케이스 다이어그램으로서 시스템에서 제공해야 하는 기능이나 서비스를 나타낸 다이어그램이다. 이에 대한 명세는 표 4, 5, 6에서 명시했다.

시스템	천리안 드론
유스케이스명	지형탐색
액터명	LiDAR
개요	LiDAR는 센서를 통해 지형 Data를 수집하여 시스템으로 보낸다.
사전 조건	-
기본 흐름	1. LiDAR는 센서를 통해서 주변 지형의 Data를 수집한다. 2. LiDAR는 수집한 Data를 시스템으로 보낸다. 3. 시스템은 받은 지형 Data를 저장한다.
대체 흐름	
예외 흐름	
사후 조건	

표 4 유스케이스 명세서 (지형탐색)

시스템	천리안 드론
유스케이스명	표적탐지
액터명	사용자, Camera
개요	Camera는 관찰한 Data를 Mini PC로 보내서 표적을 탐지한다.
사전 조건	시스템에 표적인식 모델이 학습 돼 있어야 한다.
기본 흐름	1. 사용자는 시스템에서 표적탐지 기능을 실행한다. 2. 시스템은 Camera를 통해 Data를 수집하고 저장한다. 3. 시스템은 저장된 Data에 접근하여 표적탐지를 수행한다.
대체 흐름	
예외 흐름	기본흐름2에서 Precision이 설정해 놓은 임계값 이하면 저장하지 않는다.
사후 조건	

표 5 유스케이스 명세서 (표적탐지)

시스템	천리안 드론
유스케이스명	통합지도생성
액터명	사용자
개요	사용자는 표적탐지 Data, 지형탐색 Data을 통해 만들어진 통합지도를 확인한다.
사전 조건	시스템은 지형탐색 Data와 표적탐지 Data를 가지고 있어야한다.
기본 흐름	1. 사용자는 시스템에 cartographer기술을 실행한다. 2. 시스템은 작성된 지도를 사용자에게 보여준다. 3. 사용자는 cartographer기술을 통해 작성된 지도를 저장한다. 4. 시스템은 저장된 지도를 확인하고 표적탐지 Data와 통합한다. 5. 사용자는 USB를 시스템에 연결한다. 6. 사용자는 시스템에서 작성된 통합지도를 얻는다.
대체 흐름	
예외 흐름	
사후 조건	사용자는 USB를 통해 생성된 지도를 확인한다.

표 6 유스케이스 명세서 (통합지도생성)

시스템	천리안 드론
유스케이스명	자율비행
액터명	Flight Controller, Motor
개요	Flight Controller는 시스템에서 정해준 경로에 맞게 Motor를 제어하여 자율비행을 한다
사전 조건	시스템은 지형 Data를 가지고 있어야 한다.
기본 흐름	1. 시스템은 지형 Data를 확인한다. 2. 시스템은 저장 돼 있는 지형 Data(SLAM Data)와 경로탐색 기법을 통해 진행경로를 결정한다. 3. 시스템은 결정한 진행경로를 Flight Controller에 보낸다. 4. Flight Controller는 결정된 진행 경로에 맞게 Motor를 조절한다.
대체 흐름	
예외 흐름	
사후 조건	

표 7 유스케이스 명세서 (자율비행)

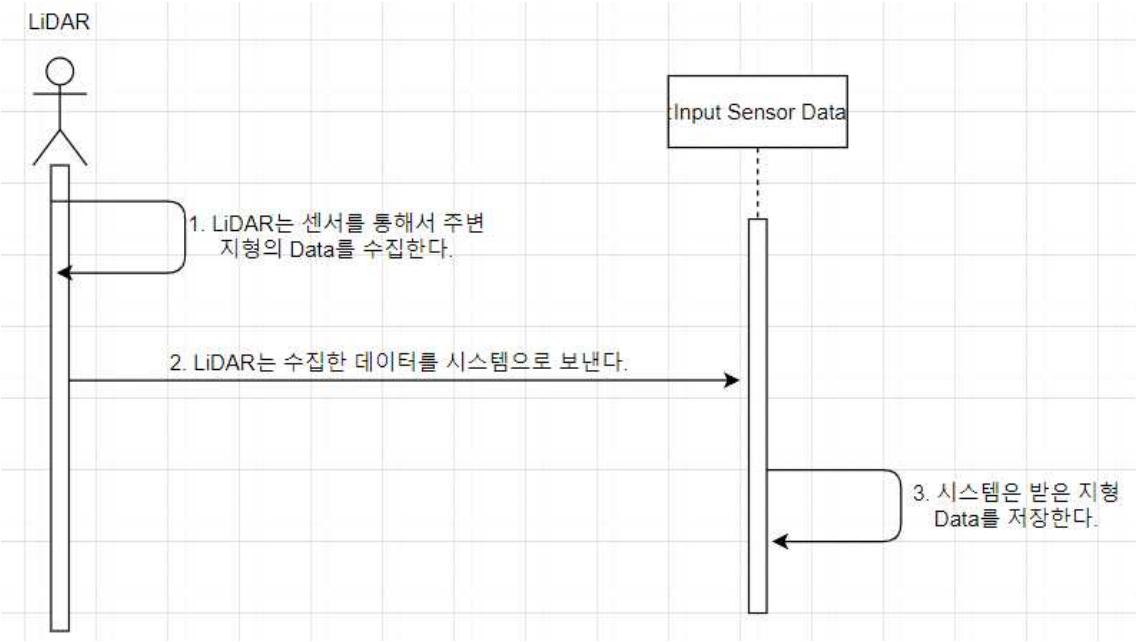


그림 35 시퀀스 다이어그램 (지형탐색)

그림 35는 지형 탐색에 대한 시퀀스 다이어그램이다. 먼저 LiDAR는 센서를 통해서 주변 지형의 Cylindrical 좌표계에 대응하는 좌표값을 수집한다. 이후 LiDAR는 수집한 데이터를 시스템에 전송한다. 마지막으로 시스템은 받은 지형 데이터를 저장한다.

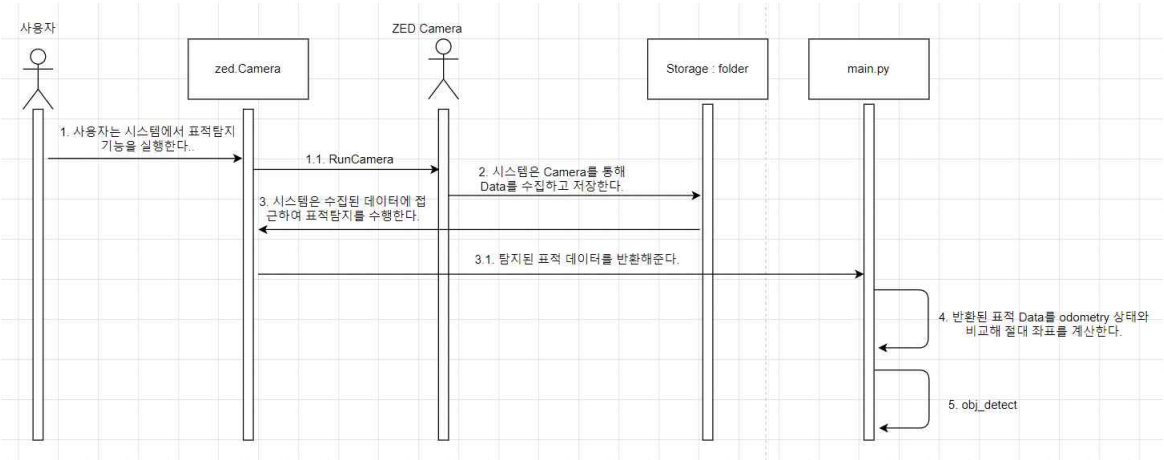


그림 36 시퀀스 다이어그램 (표적탐지)

그림 36은 표적탐지에 대한 시퀀스 다이어그램이다. 사용자는 시스템에서 표적탐지 기술을 실행하면 zed.py의 Camera가 실행된다. 시스템은 Camera를 통해 Data를 수집하고 저장한다. 시스템은 수집된 Data에 접근하여 표적탐지를 수행한다. Camera는 탐지된 표적 Data를 반환해준다. 반환된 표식 Data를 Odometry 상태와 비교해 절대좌표를 계산한다. 시스템은 이를 반복하여 수행한다.

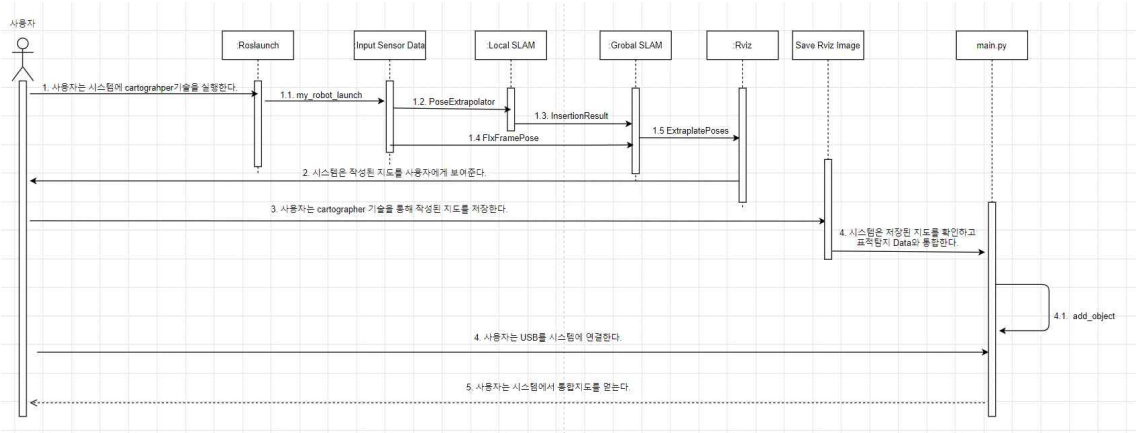


그림 37 시퀀스 다이어그램 (통합지도 생성)

그림 37은 통합지도 생성에 대한 시퀀스 다이어그램을 나타낸 것이다. 사용자가 시스템에 Cartographer 기술을 실행하면 Roslaunch는 SLAM을 실행하고 이 Data를 Rviz에 전달한다. 여기서 Cartograhper는 goolge에서 진행한 프로젝트로서 LiDAR를 통해 받은 지형 Data를 수학적으로 최적화하여 Mapping하는 기술이다. 여기서 Rviz는 ROS(Robot OS)에서 제공해 주는 기능으로, bag파일 형식으로 저장된 Data를 시각화 해주는 역할을 한다. 따라서 Rviz는 전송된 Data를 토대로 사용자에게 지도를 보여준다. 사용자는 Rviz를 통해서 나타난 지도를 이미지로 저장한다. 시스템은 저장된 지도를 확인하고 표적탐지 Data와 통합한다.

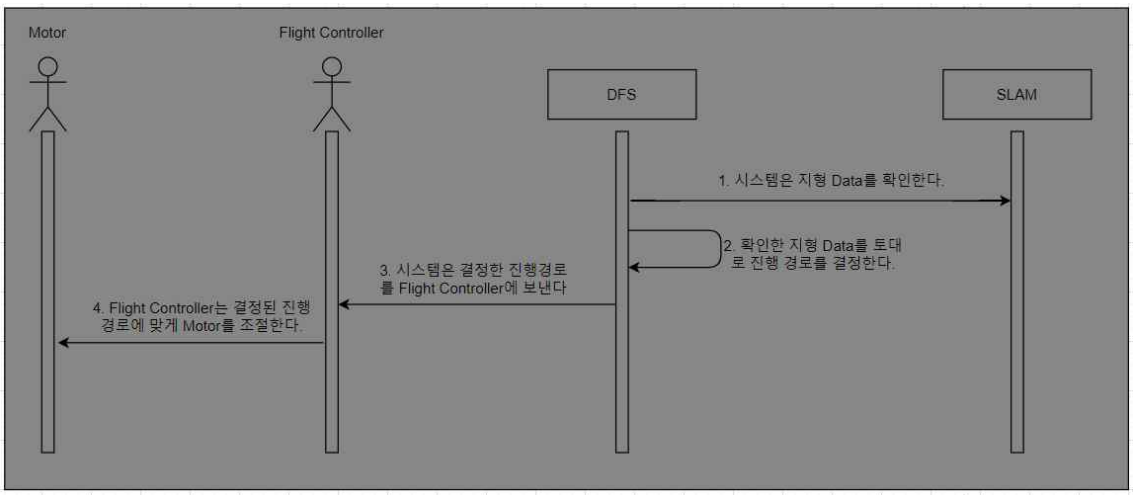


그림 38 시퀀스 다이어그램 (자율비행)

그림 38은 시스템은 지형 Data를 확인한다. 시스템은 확인한 지형 Data를 토대로 DFS를 통해 진행 경로를 결정한다. 시스템은 결정한 진행경로를 Flight Controller에 보낸다. Flight Controller는 결정된 진행 경로에 맞게 Motor를 조절한다.

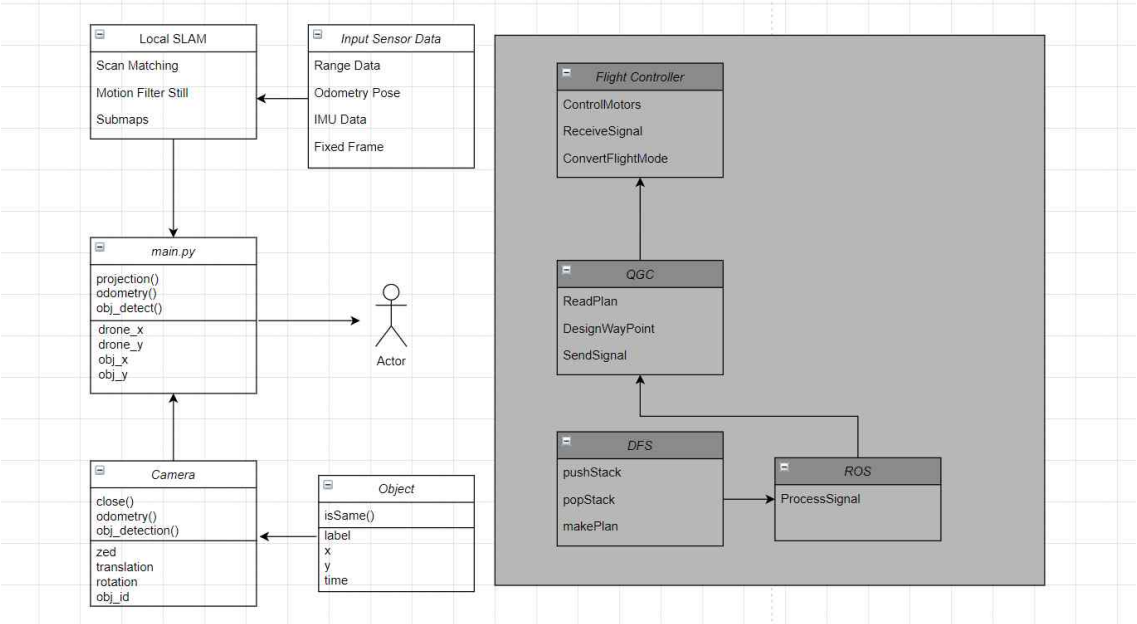


그림 39 클래스다이어그램

그림 39는 전체 SW의 대략적인 구조를 나타낸 클래스 다이어그램이다. Input Sensor Data 부분은 SLAM을 위해 각 데이터들을 센서를 통해 받아온다. 여기서 받아온 데이터를 바탕으로 Local SLAM에서 부분적인 지도를 작성을 한다. Camera 부분에서는 zed 카메라를 통해 객체를 탐지하고 object의 정보를 통해서 객체에 대한 최적화를 진행한다. main에서는 SLAM 부분과 Camera 부분이 지도에서 통합되는 부분은 main.py에서 실행되고 Actor인 사용자에게 통합된 지도가 전달된다.

3.1.2. 수학적 분석

- **드론 Specification**

  MN3508의 Testing Data를 따르면, 14.8V의 배터리와 12인치의 프로펠러를 사용한다면 출력의 50~65% 사이에서 호버링 조건(1개당 추력 = 750g)을 만족함을 알 수 있다. 변속기의 전류는 모터 최대전류의 1.2배 이상으로 선택한다. 모터의 최대전류는 실험값 기준 16.1A로 최대전류는 19.32A이다. 변속기의 전류는 20A 이상을 요구한다. 모터의 전류 소모는 호버링 상태에 대한 전류 소모를 바탕으로 선형보간법을 이용하여 예측했다.

  (800g-460g)/(7.4A-3.8A)*(소모전류-3.8A)+460g=750g
  소모전류 = 6.87A

  모터 4개의 소모전류는 27.48A이며, payload 및 FC의 전류 소모의 합은 4.525A 이므로, 드론의 전류 소모를 32A로 예측했다.

비행시간 = 배터리 용량 x 배터리 방전 / 소모 전류 * 60min / 1h

배터리 방전 = 0.8(20% 이하로 사용하는 경우, 배터리 고장 위험이 있음)

비행시간 = 10.8min

예측한 비행시간은 10.8분으로 Requirements에서 요구하는 비행시간 10분을 만족한다.

3.1.3. 컴퓨터 해석

- **제어기 설계**

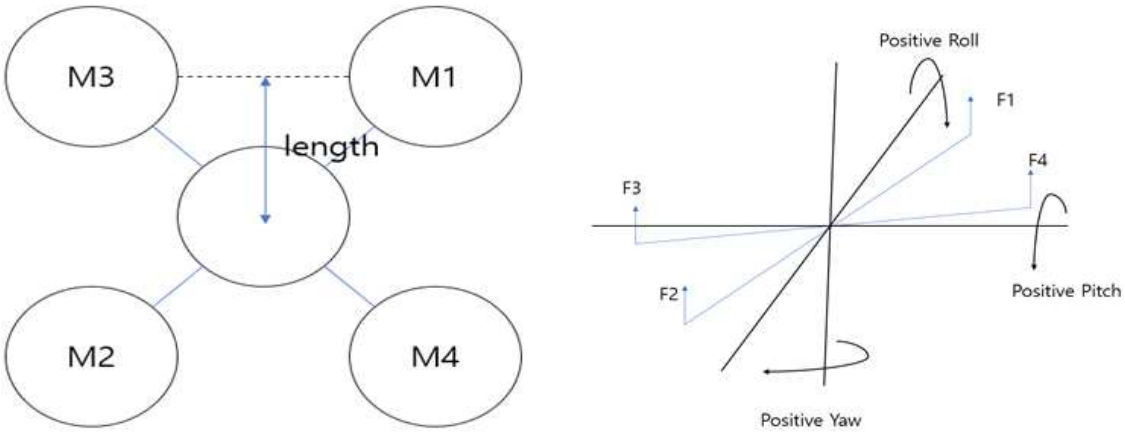


그림 40

드론의 힘과 모멘트는 그림 40과 같다. F는 추력, τ는 토크, δ는 신호, k는 상수라고 하자. $F_* = k_1\delta_*,\ \tau_* = k_2\delta_*$ 일 때, 드론의 힘과 토크를 나타내는 행렬, 신호를 나타내는 행렬은 아래와 같다.

$$\begin{bmatrix} F \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} k_1 & k_1 & k_1 & k_1 \\ -lk_1 & lk_1 & lk_1 & -kk_1 \\ lk_1 & -lk_1 & lk_1 & -lk_1 \\ -k_2 & k_2 & -k_2 & k_2 \end{bmatrix} \begin{bmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \\ \delta_4 \end{bmatrix} = M \begin{bmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \\ \delta_4 \end{bmatrix}, \quad \begin{bmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \\ \delta_4 \end{bmatrix} = M^{-1} \begin{bmatrix} F \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix}$$

드론 좌표계를 지면 좌표계로 변환시키는 좌표변환행렬은 다음과 같이 정의한다.

$$C^T = \begin{pmatrix} \cos\psi\cos\theta - \sin\psi\cos\phi + \cos\psi\sin\theta\sin\phi & \sin\psi\sin\phi + \cos\psi\sin\cos\phi \\ \sin\psi\cos\theta \quad \cos\psi\cos\phi + \sin\psi\sin\theta\sin\phi & -\cos\psi\sin\phi + \sin\psi\sin\theta\cos\phi \\ -\sin\theta \qquad \cos\theta\sin\phi & \cos\theta\cos\phi \end{pmatrix}$$

드론에 대해 ψ각을 무시할 수 있으므로, 드론 좌표계의 속도를 지면 좌표계로 변환시키는 식은 다음과 같다.

$$\begin{pmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{p}_z \end{pmatrix} = C^T \begin{pmatrix} U \\ V \\ W \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta\sin\phi & \sin\cos\phi \\ 0 & \cos\phi & -\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{pmatrix} \begin{pmatrix} U \\ V \\ W \end{pmatrix},$$

드론의 각속도를 오일러 각의 시간 변화율로 변환하는 식에 대해 선형화 과정을 ϕ와 θ가 몹시 작다고 가정하면 다음과 같다.

$$\begin{pmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{pmatrix}=\begin{pmatrix}1 & \sin\phi\tan\theta & \cos\phi\tan\theta\\ 0 & \cos\phi & -\sin\phi\\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta\end{pmatrix}\begin{pmatrix}p\\ q\\ r\end{pmatrix}=\begin{pmatrix}p\\ q\\ r\end{pmatrix}$$

드론의 6 자유도 운동방정식에 대해 p, q, r을 작은 값이라고 가정하면 다음과 같다.

$$\begin{pmatrix}\dot{u}\\ \dot{v}\\ \dot{w}\end{pmatrix}=\begin{pmatrix}rv-qw\\ pw-ru\\ qu-pv\end{pmatrix}+g\begin{pmatrix}-\sin\theta\\ \cos\theta\sin\phi\\ \cos\theta\cos\phi-\frac{F}{mg}\end{pmatrix}=g\begin{pmatrix}-\sin\theta\\ \cos\theta\sin\phi\\ \cos\theta\cos\phi-\frac{F}{mg}\end{pmatrix}$$: 선형 운동량 보존 법칙

$$\begin{pmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{pmatrix}=\begin{pmatrix}\frac{J_y-J_z}{J_x}qr\\ \frac{J_z-J_x}{J_y}pr\\ \frac{J_x-J_y}{J_z}pq\end{pmatrix}+\begin{pmatrix}\frac{\tau_\phi}{J_x}\\ \frac{\tau_\theta}{J_y}\\ \frac{\tau_\psi}{J_z}\end{pmatrix}=\begin{pmatrix}\frac{\tau_\phi}{J_x}\\ \frac{\tau_\theta}{J_y}\\ \frac{\tau_\psi}{J_z}\end{pmatrix}$$: 회전 운동량 보존 법칙

선형 운동량 보존 법칙과 좌표변환행렬을 합해 지면 좌표계에서의 가속도에 대한 식, 드론의 각속도를 오일러 각의 시간 변화율로 변환하는 식을 회전 운동량 보존 법칙과 합하면 다음과 같은 식을 얻는다.

$$\begin{pmatrix}\ddot{p}_x\\ \ddot{p}_y\\ \ddot{p}_z\end{pmatrix}=\begin{pmatrix}0\\ 0\\ g\end{pmatrix}+C^T\begin{pmatrix}0\\ 0\\ -\frac{F}{m}\end{pmatrix}=\begin{pmatrix}0\\ 0\\ g\end{pmatrix}+\begin{pmatrix}-\cos\phi\sin\theta\\ \sin\phi\\ -\cos\phi\cos\theta\end{pmatrix}\frac{F}{m},\ \begin{pmatrix}\ddot{\phi}\\ \ddot{\theta}\\ \ddot{\psi}\end{pmatrix}=\begin{pmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{pmatrix}=\begin{pmatrix}\frac{\tau_\phi}{J_x}\\ \frac{\tau_\theta}{J_y}\\ \frac{\tau_\psi}{J_z}\end{pmatrix}$$

$\ddot{p}_x, \ddot{p}_y, \ddot{p}_z$ 선형화를 마지막으로 다음 두 행렬을 얻는다.

$$\begin{pmatrix}\ddot{p}_x\\ \ddot{p}_y\\ \ddot{p}_z\end{pmatrix}=\begin{pmatrix}-g\theta\\ g\phi\\ -\frac{F}{m}\end{pmatrix},\ \begin{pmatrix}\ddot{\phi}\\ \ddot{\theta}\\ \ddot{\psi}\end{pmatrix}=\begin{pmatrix}\frac{\tau_\phi}{J_x}\\ \frac{\tau_\theta}{J_y}\\ \frac{\tau_\psi}{J_z}\end{pmatrix}$$

위 결과들을 바탕으로 Matlab Simulink를 통해 그림 41의 제어기를 설계한다.

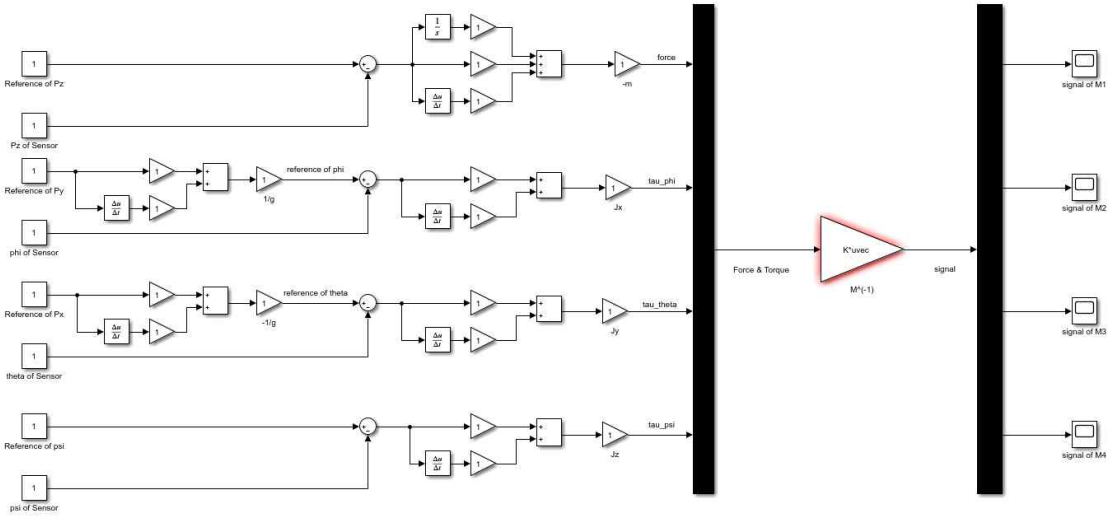


그림 41 PID 제어기

● 표적탐지 기술 검증

(1) IoU(Interaction of Union): 그림 42는 IoU의 정의이다. 물체 탐지에서 Bounding Box를 얼마나 잘 예측했는지를 측정하는 수치이며 교집합을 합집합으로 나눈 값으로 정의한다. 임계값 이하의 Bounding Box는 제거한다.

(2) Confidence: 검출한 것에 대해 알고리즘이 얼마나 확신이 있는지를 나타내는 지표이다. 임곗값을 기준으로 임곗값 이하의 경계 박스 후보는 배경으로 간주해서 제거함으로써 recall과는 반비례 관계이고 precision과는 비례관계이다.

(3) Threshold: IoU, Confidence의 참 거짓 여부를 판단하는 임계값이다.

(4) Confusion Matrix: 그림 43은 Confusion Matrix로 TP, FP, FN, TN을 정의한다. 이는 다음 Precision, Recall 등에 사용된다.

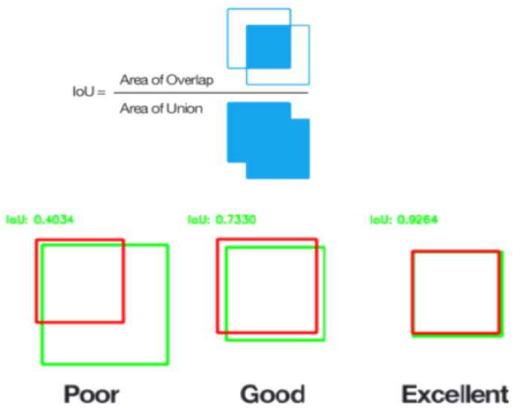


그림 42 IoU 정의

실제 상황 (ground truth)	예측 결과 (predict result)	
	Positive	Negative
Positive	TP(true positive) 옳은 검출	FN(false negative) 검출되어야 할 것이 검출되지 않았음
Negative	FP(false positive) 틀린 검출	TN(true negative) 검출되지 말아야 할 것이 검출되지 않았음

그림 43 Confusion Matrix

(5) $\mathrm{Precision} = \frac{TP}{TP+FP} = \frac{TP}{all\,\mathrm{detections}}$; 정밀도

탐지된 객체의 정확도, 0.0 ~ 1.0 사이의 값을 가지며 높을수록 좋다.

(6) $Recall = \frac{TP}{TP+FN} = \frac{TP}{all\,ground\,truths}$; 재현율

전체 객체 중에서 탐지한 비율, 0.0 ~ 1.0사이의 값을 가지며 높을수록 좋다.

(7) $F1\,Score = \frac{2}{\frac{1}{\mathrm{Precision}} + \frac{1}{Recall}}$

precision과 recall의 조화평균이며 0.0 ~ 1.0 사이의 값을 가지며 높을수록 좋다. 정밀도와 재현율이 모두 높으면 F1 Score가 높으므로 하나의 값으로 2가지 정보를 유추할 수 있다.

(8) AP (Average Precision): 변화하는 recall에 따라 대응되는 precision을 그림 44와 같은 PR Curve를 적분하여 얻은 값이다.

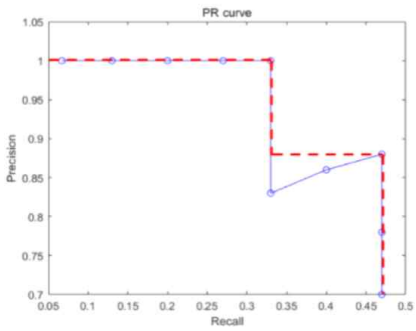


그림 44 PR Curve

(9) mAP (mean Average Precision): AP를 Class 수로 나눈 값이다.

(10) FPS(frame/second): 실시간성을 평가하는 지표이다.

- yolov5

그림 45는 Precision/confidence 그래프이며 Confidence가 0.91일 때 최대의 Precision을 보인다. 그림 46은 PR그래프 이다 mAP 지표는 0.992로 나타나 있다. 모든 지표가 상당히 높은 수치를 나타내고 있지만 ADD에서 제공해주는 데이터이다. 같은 표적을 가지고 각도, 고도, 조도에 대한 변화만을 반영하여 train set과 test, valid set이 거의 동일한 데이터 셋으로 이루어져 있기 때문에 높은 수치로 측정되었다고 예측한다.

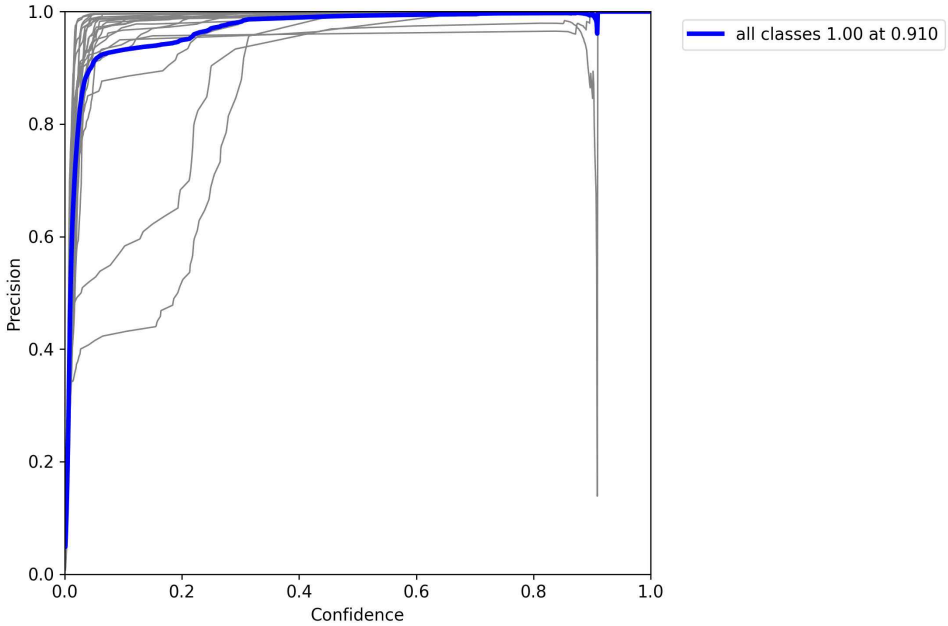


그림 45 Precision/confidence 그래프

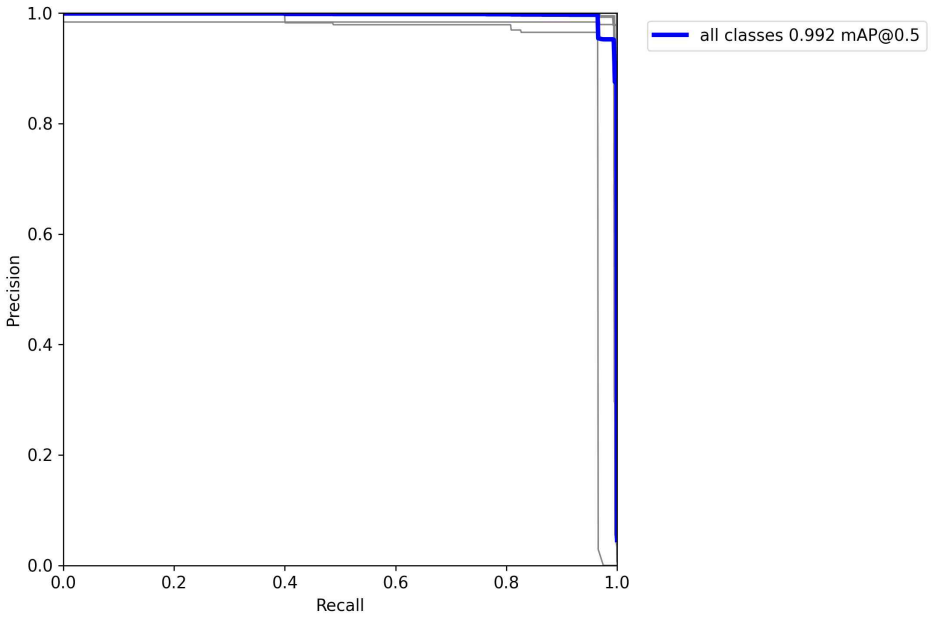


그림 46 PR 그래프

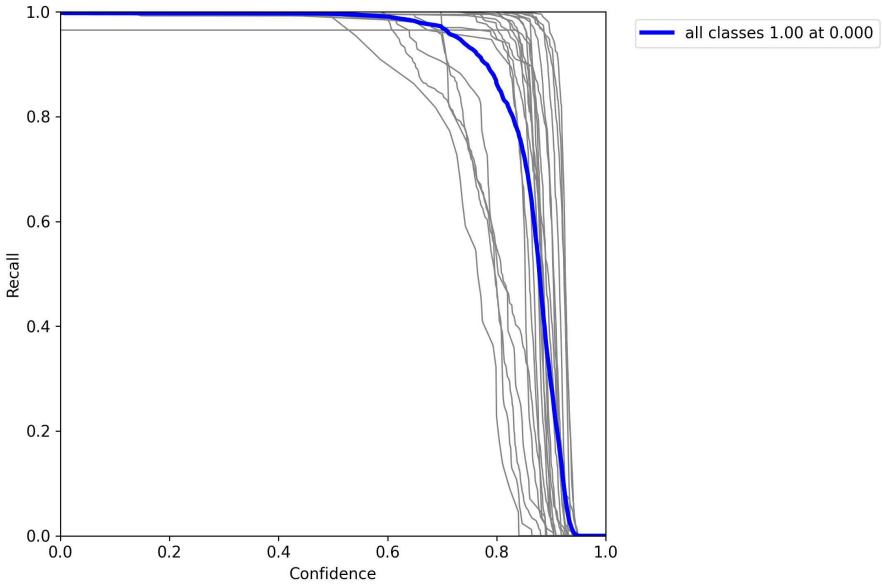


그림 47 Recall/Confidence

- yolov4

본 과제에서는 zed와 yolo 모델을 통합하여야 함으로 output으로 .weights, .cfg을 얻을 수 있는 Darknet 프레임워크를 사용하였고 Darknet 프레임워크에서 학습이 가능한 yolov4 모델을 사용하였다. yolov4의 모델학습을 위해서 yolov5에서 진행한 것과 마찬가지로 GPU에 맞는 드라이버 설치와 cuda환경을 설정해주었다. yolov5와는 다르게 방사능 표지와 군인으로 두 가지 클래스만을 학습 시켰으며 데이터셋은 크롤링 수집방법과 팀원이 군복을 입고 직접 촬영하여 데이터 셋을 생성하였다. 데이터셋은 Labeling 도구를 활용하여 데이터셋 내의 클래스 위치를 표시해 주었고 이 데이터를 roboflow라는 데이터는 저장하고 내려받는 것을 지원하는 사이트에 업로드 하여 자동으로 Annotate 파일을 생성해주는 사이트를 활용하였다.(그림 48 참고)

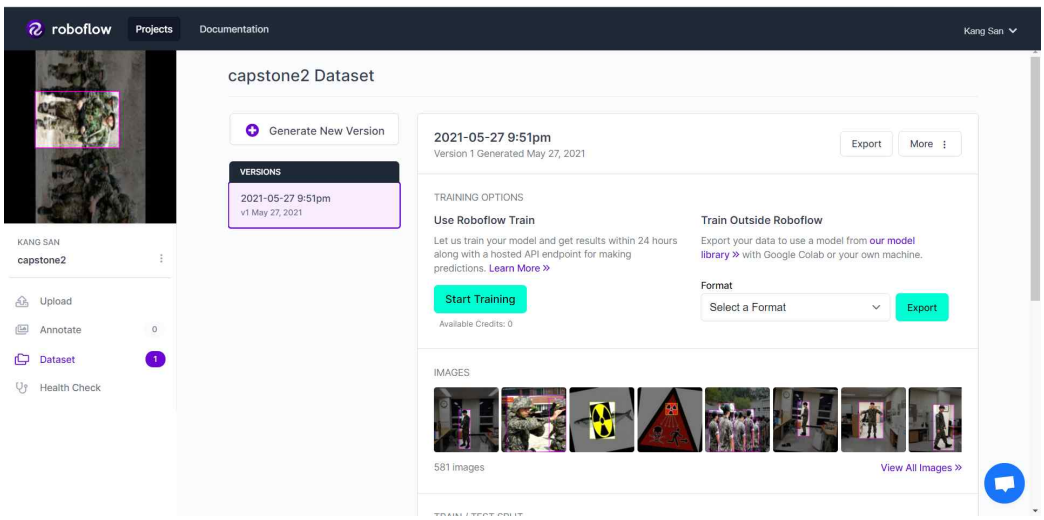


그림 48 roboflow

위와 같은 데이터 셋을 활용하여 학습한 결과는 다음과 같다. 그림 49는 train 셋 506장 valid셋 50장 test 셋 25개의 이미지로 4000회 반복 학습한 결과에서 mAP와 손실률을 나타낸 그림이다. 이를 통해서 약 18000번의 iteration에서 최고의 mAP지표를 나타내는 것을 확인 할 수 있었으며 이때 .weights 파일을 zed 카메라와 yolov4를 활용한 표적탐지 기술구현에 사용하여 구현 하였다.

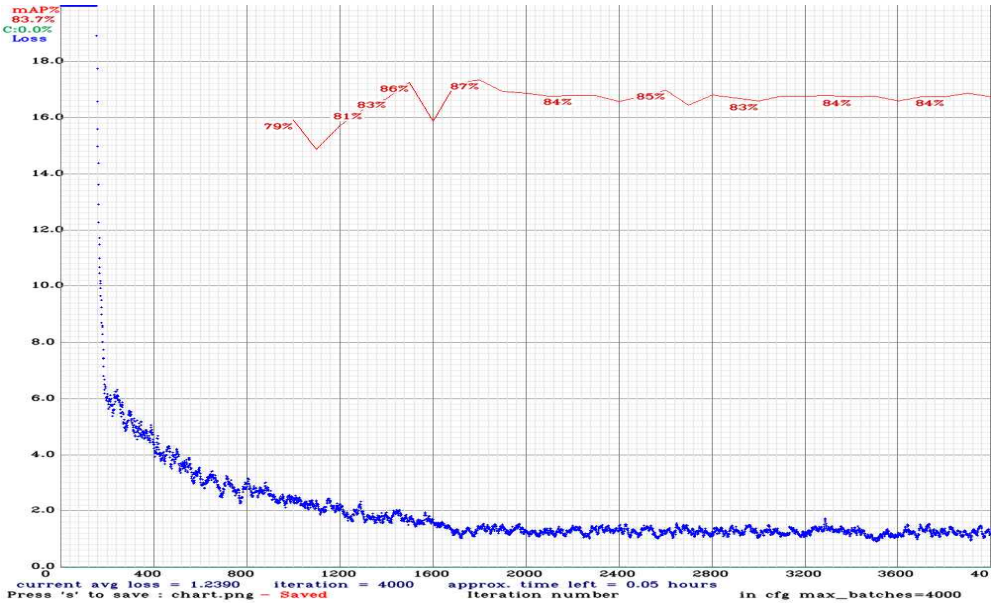


그림 49 mAP, 손실률

3.2. 제작과정 및 결과물 평가

3.2.1. 제작과정

- 추력 시험

  모터-프로펠러 조합의 추력을 검증하기 위해 추력 시험을 진행한다. 추력 시험을 통해 모터와 프로펠러 조합의 추력은 표 8로 확인 할 수 있었다. 추력 시험 결과에 따르면 예상 중량 3000g에 대해 추중비는 1.533으로 2보다 낮다. 이는 여유 추력이 낮고, 출력의 70~80% 사이에서 호버링이 이루어지는 불안정한 상태이다.

Voltage(V)	Prop	Throttle(%)	Thrust(g)
14.8	12*5.5	50	280
		60	450
		70	620
		80	800
		90	100
		100	1150

표 8 추력 시험 결과

● **드론 제작**

(1) Pixhawk 펌웨어 설치

FC의 종류로는 Pixhawk, NAZA 등이 있으며, 본 과제는 펌웨어가 공개된 Pixhawk를 사용한다. 사용한 FC Pixhawk 4 Mini는 펌웨어로 Ardupilot을 사용했다. 펌웨어의 종료는 PX4와 Ardupilot이 있다. 펌웨어 설치를 지원하는 플랫폼으로 QGroundControl과 Mission Planner가 있다. QGC를 이용하여 PX4를 설치할 때, Pixhawk 4 Mini의 센서가 +Roll 방향으로 기울어져 있다고 인식한다. 초도 비행을 통해 –Roll 운동이 반복됨을 확인하여, Mission Planner를 이용해 Ardupilot을 설치했다. Ardupilot의 경우 센서를 정확히 인식하여 문제가 발생하지 않았다. HW 조립 이후, Pixhawk의 기본 설정은 Youtube의 '[팰콘샵] 드론 제작 가이드 2-2편 - 픽스호크 셋팅에서 비행까지 완벽 따라하기'를 따라 진행했다[18]

(2) HW 조립

Motor와 Mount를 결합한다. Carbon plate와 Motor 결합 시, 배선을 유의하여 결합한다. Motor base와 Carbon plate 결합 시, 그림과 같이 가장 끝 나사만 조인다.

그림 50 Motor – Mount 결합

Motor base와 Pipe, Pipe와 Buckle과 Attachment를 결합하고, 나머지 나사를 조인다. 배선을 Pipe 내부에 넣고, Attachment와 Pipe를 연결할 때, 나사 구멍에 맞추어 장착하며 Motor base와 지면의 수평을 유지한다. 총 4개의 Arm을 조립한다.

그림 51 Arm 조립

Power Module과 ESC를 납땜한다. 납땜 완료한 Power Module과 Chassis, Arm을 조립한다. 드론의 앞/뒤를 구분하기 위해 정면 Arm은 검은색, 후면 Arm은 적색으로 조립한다. Power Module의 Power 선은 드론 후방을 향하도록 장착한다. 조립 시 배선을 정리하며 진행한다.

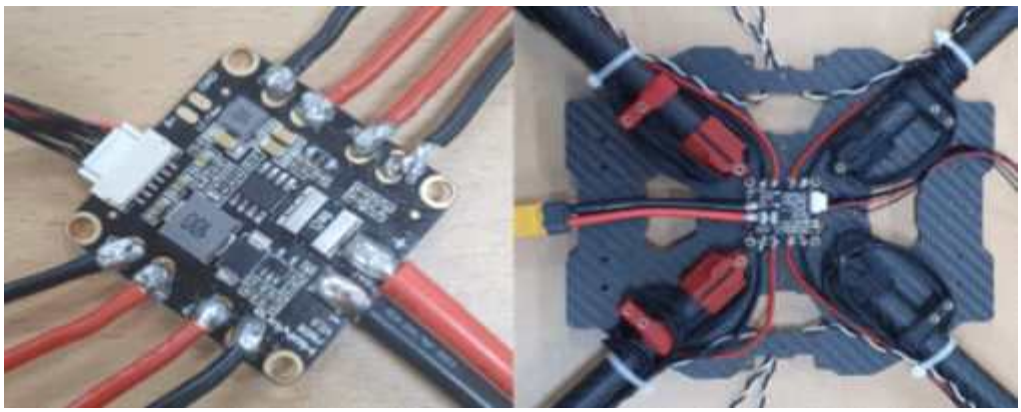


그림 52 Chassis 조립

Pixhawk를 Chassis 배면에 부착한다. Pixhawk에 그려진 화살표가 정면을 향하도록, 위/아래를 구분하여 부착하며 센서값 오차를 줄이기 위해 드론의 중심부에 부착한다. Pixhawk의 PPM 포트에는 PPM Encoder를 연결하여 RX601의 신호를 받을 수 있도록 한다. Telemetry는 TELEM1 포트에, LiDAR lite v3는 UART*I2C B 포트에 연결한다. Power Module과 연결하기 위해 RC IN 포트를 Power Module 선과 연결한다. ESC의 선은 Pixhawk의 MAIN OUT의 1, 2, 3, 4번 포트를 연결한다. Hub Plate

기준 M3를 1번, M4를 2번, M1을 3번, M4를 4번에 연결한다.

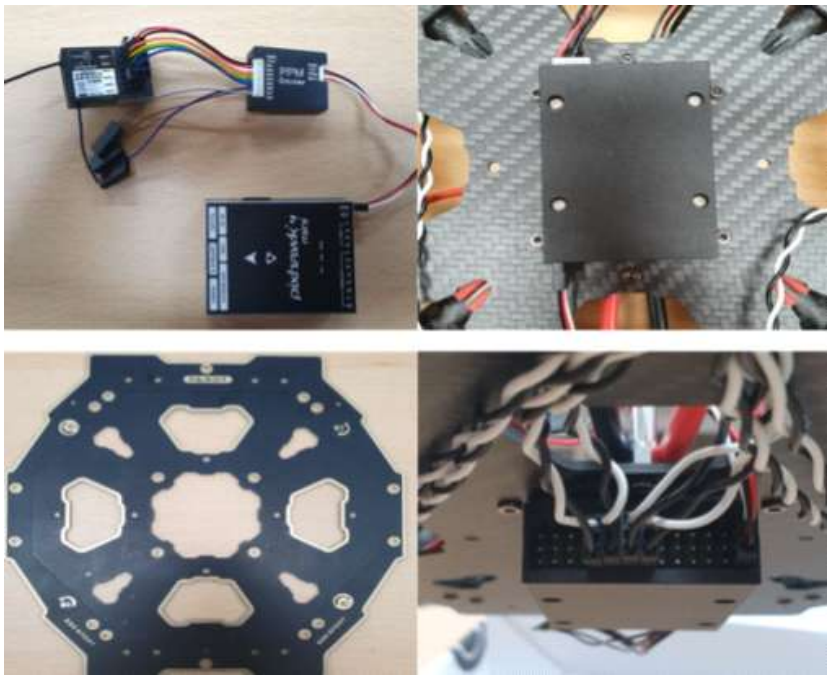


그림 53 Pixhawk 부착

다리와 Panel을 드론에 부착한다. 배터리를 Panel에 부착할 수 있도록 벨크로를 사용한다. 마지막으로 Hub Plate를 덮어서 조립을 마무리한다.

그림 54 다리, Panel, Hub Plate 부착

- **초도 비행**
  Payload를 부착하지 않은 상태에서 드론 동작을 확인하기 위해 초도 비행을 했다. '무인 항공기 제어 실험실'에서 실험했으며, 초도 비행 시 드론이 무게는 2043g, Prop은 12*5.5를 사용했다. 시험을 통해 Pitch, Roll, Yaw 방향과 비행 모드(수동, 호버링, 착륙) 기능이 정상 수행함을 확인했다.
- **3D 프린팅**
  HW 설계에 따라 3D 프린팅을 이용해 Mount를 설계했다.

그림 55 Mount 출력물 및 조립 결과

- **Jetson Xavier 보드 부팅**

  과제에 사용된 보드는 "Jeton Xazier nx"라는 모델이다. 처음 빈 보드에 OS(Operating System)를 탑재하기 위해 개인 desktop에 nvidia에서 제공해주는 이미지 파일(SDK for Jetpack 4.5)을 다운 받는다. Jetpack 4.5에서는 Ubuntu18.04를 제공해주고 딥러닝(deep-learning) 추론 최적화 프로그램을 포함한 TensorRT 7.1.3, 딥러닝 프레임워크를 위한 라이브러리인 cuDNN 8.0, GPU 가속 애플리케이션을 구축을 도와주는 CUDA 10.2

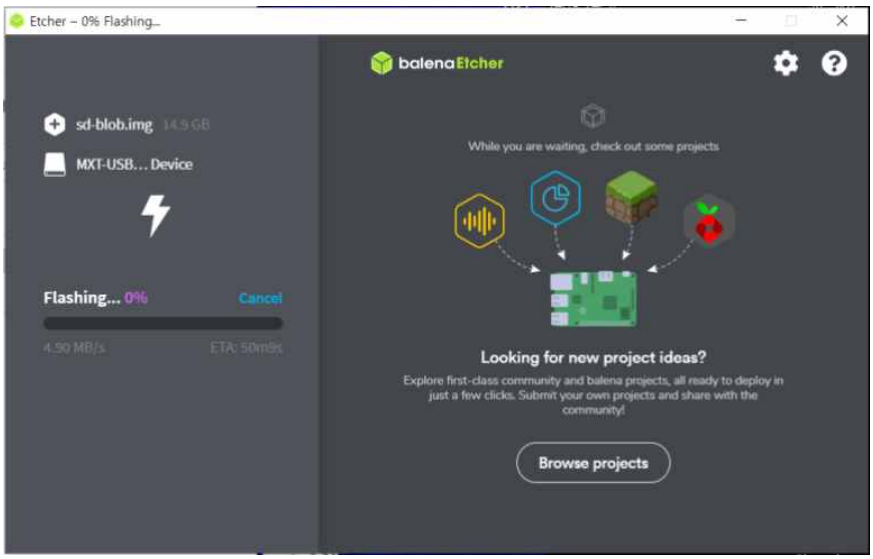


그림 56 balenaEtcher

등을 포함하고 있다. 다운로드가 완료된 이미지 파일은 SD card를 통해서 보드에 설치 되어야한다. 이미지 파일을 SD card로 적재해주는 프로그램은 "balenaEtcher"를 이용하였다. 그림 56는 balenaEtcher를 이용해서 SD card에 Jetpack 4.5를 적재하고 있는 그림이다. 이후 적재가 완료된 SD card를 Jetson Xavier에 삽입하고 부팅을 하면 Jetson Xavier 보드 이용이 가능하다.

- **표적탐지 기술 구현**

  표적탐지를 하기 위한 카메라로는 ZED 카메라를 사용한다. ZED 카메라를 사용하기 위해서는 ZED SDK의 설치가 필요하다. (https://www.stereolabs.com/developers/release/) 링크를 통해 다운 받은 파일에는 확장자가 .run인 ZED 이름 의 파일을 실행시키면 설치가 진행된다.

  $ chmod a+x 파일명.run

  $ sudo ./파일명.run

  설치가 진행되는 과정에서 "Do you want to install the Python API ?" 라는 옵션에서 Y를 선택하면 생기는 get_python_api.py를 실행시킴으로써 Python API도 함께 설치가 가능하다. 모든 설치가 완료되면 /usr/local/zed/tools/에 실행파일들이 생성되어 있다. ZED 카메라는 $ ZED_Explorer 명령을 통해서 그림 57처럼 실행할 수 있다. ZED 카메라는 stereo 카메라이기 때문에 양안으로 촬영된다.

그림 57 ZED 카메라를 실행시킨 모습

Yolov5모델을 학습을 시켰을 때는 대표적인 딥러닝 프레임워크인 pytorch를 이용했기 때문에 가중치 파일의 확장자는 .pt로 Configuration 파일은 .yaml로 생성됐다. 하지만 표적탐지 모델과 ZED 카메라의 통합을 위해 사용한 github repository에서는 확장자가 .weights인 가중치 파일과 확장자가 .cfg인 Configuration 파일이 필요하다. 따라서 위 두가지 파일을 output으로 가지는 Darknet 플랫폼 사용하였다. 표적탐지 모델의 학습을 위해 사용한 데이터는 웹 크롤링을 통해 수집하여 Labeling 도구를 사용하여 생성하였고 학습의 속도를 고려하여 Google Colab을 사용하여 학습을 진행 하였다. batch 사이즈는 colab의 GPU의 RAM를 고려하여 32로, epoch은 통상적으로 클래스의 수 * 2000 고려하여 4000으로 결정하였다. 이렇게 생성된 weights 파일과 cfg 파일을 사용하여 zed-yolo reposit

ory를 사용하여 표적탐지 기술을 구현하였다.

- **지도작성 기술 구현**

  Cartographer 기술을 구현하기 위해서는 ROS Melodic을 필요로 한다. 여기서 ROS Melodic이란 로봇 응용 프로그램을 개발할 때 필요한 라이브러리와 다양한 개발 및 디버깅 도구를 제공하는 응용 프로그램이다. ROS Melodic은 다음 링크를 참고하여 설치 가능하다[19]. 본 과제를 진행할 때에는 위 링크의 1.4 Installation 부분에서는 Desktop-full install을 진행하였고, 1.5 Environment setup 부분에서는 ~/.bashrc를 사용하는 방법으로 진행하였다. RPLiDAR를 운용하기 위해서는 마찬가지로 SDK가 필요하고 이를 ROS와 통합하기 위해 SLAMTEC에서 제공하는 rplidar_ros 레파지토리를 내려받아 빌드를 진행했다[20]. rplidar_ros를 내려받고 빌드하는 부분은 아래와 같은 명령어를 사용하여 진행할 수 있다.

  $ git clone https://github.com/Slamtec/rplidar_ros.git
  $ catkin_make

  위와 같이 Cartographer를 구현할 준비가 되면 Cartographer 레파지토리를 내려받아 설치를 진행한다[21]. 본 과제에서는 ROS Melodic으로 진행하기 때문에 wstool, rosdep과 Ninja를 설치할 때 아래의 명령어를 사용한다.

  $ sudo apt update
  $ sudo apt install –y python-wstool python-rosdep ninja-build stow

  wstool은 프로젝트의 작업공간을 관리해 주는 도구이고 rosdep은 ROS상의 중요한 요소들을 구동하는데 필요한 시스템 의존성을 찾아서 설치를 도와주는 도구이다. 마지막으로 Ninja는 빌드를 더욱 빠르게 할 수 있도록 도와준다. 도구들의 설치가 완료되면 Cartographer를 구현하기 위한 작업공간인 catkin_ws_ros 디렉토리를 만들어 주었다. 그 후 wstool을 이용해서 src 디렉토리를 만들어서 Cartographer 레파지토리를 받아와준다.

  $ mkdir catkin_ws_ros
  $ cd catkin_ws_ros
  $ wstool init src
  $ wstool merge –t src https://raw,githubusercontent.com/cartographer-project/cartographer_ros/master/cartographer_ros.rosinstall
  $ wstool update –t src

  다음으로는 cartographer_ros의 dependencies를 설치하기 위해 위에서 설치했던

rosdep을 사용한다.

$ sudo rosdep init
$ rosdep update
$ rosdep install --from-paths src —ignore-src —rosdistro=${ROS_DISTRO} -y

Cartographer는 abseil의 C++코드를 사용하기 위해서 abseil 라이브러리를 설치한다. abseil이란 구글에서 내부적으로 사용되는 C++코드들을 모아서 라이브러리로 만들어 놓은 것이다. ROS의 abseil과의 충돌을 방지하기 위해서 기존 ROS abseil은 제거해준다.

$ src/cartographer/scripts/install_abseil.sh
$ sudo apt remove ros-${ROS_DISTRO}-abseil-cpp

마지막으로 빌드와 설치를 진행한다.

$ catkin_make_isolated --install --use-ninja

설치가 완료되면 launch 파일, lua 파일, urdf 파일 그리고 rviz 파일을 수정한다(수정된 파일들은 부록에서 확인할 수 있다). launch 파일은 여러 node를 한 번에 실행시키거나 같은 node를 여러 개 쓰는 작업을 한다. lua 파일은 프로그램에 내장될 수 있는 스크립트로 임베디드 스크립트라고 부르기도 한다. urdf 파일은 robot 모델에 대해서 물리적 특성을 XML언어로 정의 해놓은 것이다. 수정된 파일들은 레파지토리를 빌드 했던 폴더의 하위 폴더 "cartographer_ros"에서 my_robot.lua는 cofiguration 폴더 안에, head_2d.urdf는 urdf 폴더 안에, my_robot.launch, visualization.launch는 launch 폴더 안에 저장하고 demo.rviz 파일은 rviz 폴더를 만든 후 안에 저장한다.

실행을 할 때에는 LiDAR 작동과 지형 데이터를 수집하여 cartographer를 수행하는 터미널1과 실시간으로 시각화를 해주는 터미널2까지 총 두 개의 터미널을 실행했다. 각각의 터미널에서 명령어는 아래와 같다.

터미널1.
$ roslaunch cartographer_ros my_robot.launch
터미널2.
$ roslaunch cartographer_ros visualization.launch

실행을 하면 rplidar가 작동되고 지도 작성이 시작된다. 이에 대한 시각화는 Rviz를 통해서 확인하였다. 그림 58는 Rviz를 통해 시각화된 지도의 모습을 보여준다.

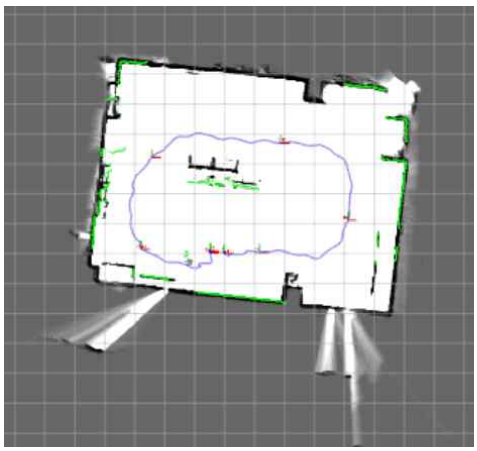


그림 58 Cartographer Data를 Rviz를 이용해 시각화한 지도

- **Mapping SW 개발**

  통합 지도 작성을 위해 실시간 Odometry 값을 알아야 한다. 실시간 Odometry 값을 추정하기 위해 ZED API의 소스코드 및 예제를 다루는 Github 레파지토리를 참고했다[22][23]. 먼저 ZED API를 통해 드론의 위치와 회전각을 반환받을 수 있다. 이 레파지토리 예제에서 다루는 좌표계는 RIGHT_HANDED_Y_UP 좌표계와 m단위를 사용한다. 여기에서 m 단위를 mm 단위로 반환 받도록 수정하였다. 이를 고려해 좌표 및 척도를 필요에 따라 조정해 저장한다.

  Odometry 값을 알게 되면 객체를 탐지해 3D 좌표를 반환해야 한다. 이는 ZED API와 YOLO V4를 사용해 객체를 탐지하고 탐지한 객체의 상대 좌표와 클래스를 반환하는 Github 레파지토리의 예제를 참고했다[24]. 하나의 객체를 여러 번 반환해 지도에 표시하면 올바르지 않은 정보가 지도에 나타나기 때문에 이 경우를 줄이기 위해 해당 객체마다 ID를 부여해 같은 ID를 가지는 객체를 반환하지 않도록 설계한다. 우리는 객체가 탐지된 시간과 위치가 비슷하면 같은 ID로 판단하도록 구현하였다.

  main에서 Odometry 값과 객체의 상대좌표를 반환받아 객체의 절대좌표를 계산한다. 이 때 반환 받는 상대좌표는 회전 및 이동한 Cartesian 좌표계 이므로 좌표계의 회전변환 공식을 사용한다. 그림 59는 좌표계 회전변환 공식을 이용해 객체의 위치를 보정한 결과이다.

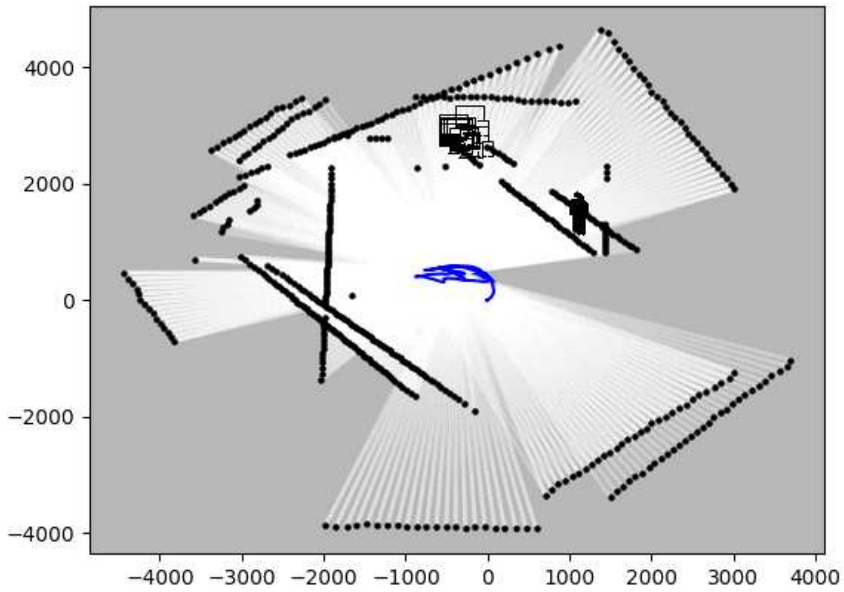


그림 59 좌표축 회전변환을 이용한 객체 위치 보정 결과. 이 예시에서는 ID 중복을 허용해 주었다.

ID를 통해 객체 반환을 줄여주더라도 같은 객체를 여러 번 인식할 우려가 있다. 예를 들어 다른 장소로 이동한 뒤 다시 이전의 위치로 이동했을 경우 다시 같은 위치에 있는 같은 객체를 서로 다른 객체로 판단해 여러 번 객체를 표시할 우려가 있다. 이는 객체의 절대좌표를 계산하는 main에서 같은 클래스의 객체를 대상으로 거리를 측정해 1m 이내에 존재하는 객체를 동일 객체로 판단해 제거하였다.

객체 탐지 결과를 Cartographer와 통합하기 위해 Rviz 지도가 png 파일로 저장되면 그 사진을 matplotlib을 이용해 불러온다. 불러온 지도 사진에 저장된 객체를 표시한다. 이때 불러온 이미지는 좌측 상단을 원점으로 하는 px단위 이미지이므로 객체의 위치는 이 점을 고려해 계산해야한다. Cartographer에서는 기본적으로 Lidar Odometry를 지원하는데, 이 Odometry값과 카메라 Odometry 값이 다르면 탐지된 객체의 위치를 신뢰할 수 없다. 카메라 Odometry와 Lidar Odometry를 비교할 수 있도록 카메라 Odometry도 지도에 표시해 이미지를 저장한다.

- **프로토타입 시연**

실내 좁은 환경에 드론 주행의 위험성으로 인해 숙련된 드론 조종사가 필요하나, 보고서 제출 기한까지 섭외하지 못하여 본 실험은 드론에 부착 없이 그림60과 같이 진행하였다.

본 실험은 공과대학 407동 5층에서 진행하였으며 목표 인식 물체는 군인과 방사능 표식이다. 아래의 데모 시나리오 표에 따라 시험을 진행하였다.

사전조건	군복 입은 사람이 5층 홀에, 방사능 마크를 복도에 배치한다.
#1	501호 뒷문 앞에서 출발한다.
#2	사람이 Lidar, 카메라 mini PC, 배터리를 들고 정해진 이동 경로를 따라 이동하며 지도 작성 및 영상 정보를 얻는다.
#3	PC를 들고 있는 사람은 시작점으로 돌아오며 SLAM 결과를 사진으로 저장한다.(그림 61)
#4	시스템은 저장된 SLAM 결과 사진을 불러와 ZED Odometry 결과와 표적 인식 결과를 지도에 표시한 뒤 저장한다.
#5	팀원은 mini PC에서 완성된 지도를 확인한다.
OUTPUT	통합지도 (그림 62. 표적에 대한 정보가 포함된 2D 지도) https://www.youtube.com/playlist?list=PL-FO1b2-7fJa25fnTEd72a5VR85sku3cH

표 9 데모 시나리오

그림 60 Payload

그림 61 Lidar SLAM

그림 62 통합지도

3.2.2. 실험결과 및 분석

프로토타입 시연을 통해 수행 능력을 확인했다. 방사능 표식 및 군인의 객체 정보 및 위치정보를 지도상에 표시했다. 표적을 탐지하는 모델은 yolov5 모델을 pytorch 프레임워크에서 학습하여 사용하려고 하였으나 pytorch에서 학습하여 생성된 파일인 .pt, .yaml 파일이 ZED 카메라를 사용하여 표적을 탐지하는 repository에 알맞지 않았다. 따라서 zed-yolo repository에 적용할 수 .weight 파일과 .cfg 파일을 얻기 위해 Darknet 프레임워크를 사용하여 yolov4 모델을 학습시켜서 표적탐지를 진행하였다. Darknet에서 학습 후 반환하는 손실률 및 mAP 그래프와 .weights, .cfg, .names 파일을 활용하여 mAP와 손실률을 통해 최적의 성능을 낼 수 있는 모델의 weights 파일을 선정하였다. 아래의 그림63은 mAP와 손실률을 나타는 그래프이며 약 18000반복 하였을 때 최적의 mAP를 나타내는 것을 확인 할 수 있다. 기술 통합과 관련된 내용으로는 동일 객체를 서로 다른 객체로 인식하고 반환하는 것을 막기 위해 비슷한 시간에 카메라 프레임 상 가까이 있는 물체를 동일 객체로 가정하고, 카메라가 이동하면 객체가 프레임 상에서 반대로 이동하는 것을 표현하기 위해 그 객체의 카메라 좌표를 갱신하며 이동하여 주변부에서 관측되는 동일 객체를 같은 ID로 부여해 복수 인식률을 낮추었다. 같은 객체에 대해 다른 시간에 탐지하거나 센서 오차로 발생하는 단일 물체에 대한 복수 표식 문제를 해결하기 위해 main에서도 비슷한 절대좌표에 위치하는 객체를 동일 객체로 가정하고 이를 제거하는 방식으로 동일 객체 복수 표시 문제를 해결하였다. 프로토타입 제작을 통해 개발하는 드론의 안정성을 확인했다. 추력 시험을 통해 드론의 프로펠러-모터 조합 성능을 확인했다. 소모전류 측정 실험을 병행하여 계획했으나, 장비 미확보로 측정하지 못했다. 추력 시험을 통해 확인한 성능은 유효한 결과값을 갖지 못했다. 확보한 데이터에 따르면 추중비가 2보다 작아, 안정적인 비행이 어렵다고 예상했으나, 초도 비행을 통해 확인한 결과에 따르면 안정적인 비행 수행이 가능했다. 자율 비행 임무 수행을 다음 해결과제로 남겨둠에 따라, 자율 비행에 필요한 제어기 설계와 관성 모멘트 측정 실험, 경로 비행 SW 개발을 진행하지 않았다.

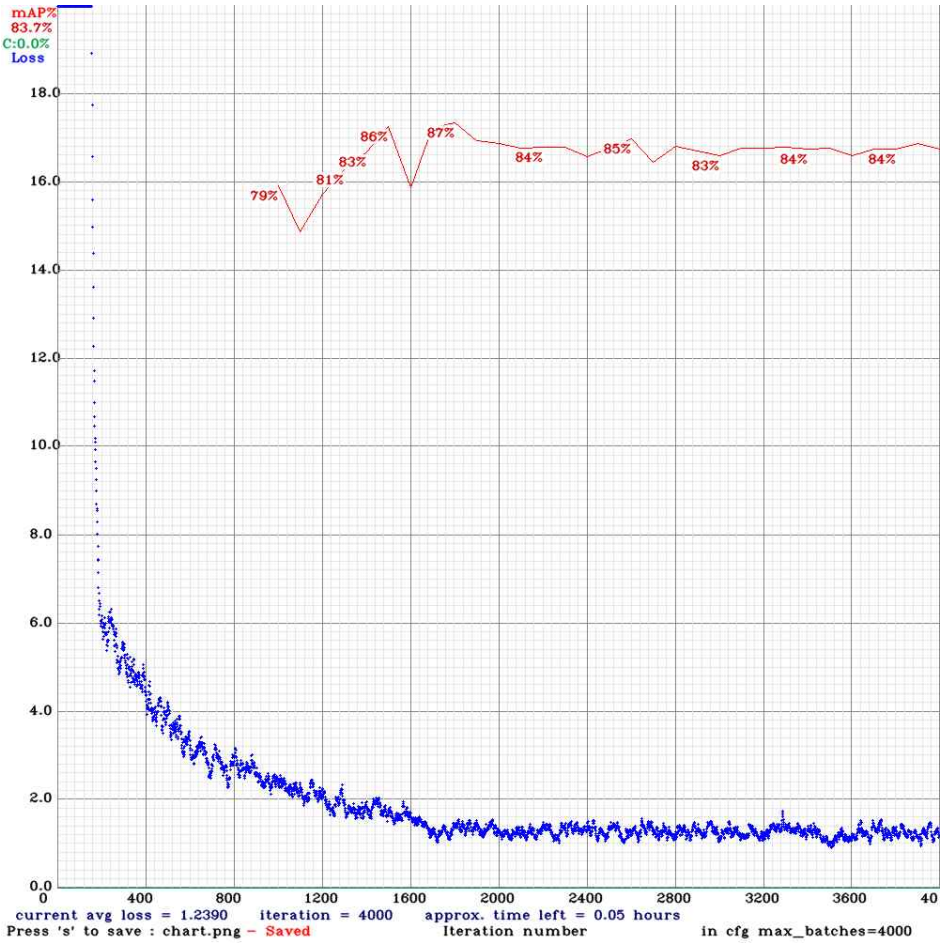


그림 63 mAP, 손실률

프로토타입 실험 평가항목		1	2	3	4	5
HW 항목	추력 시험이 성공적이었는가?			✔		
	초도 비행이 안정적이었는가?					✔
	관성모멘트 측정이 정확했는가?	✔				
	3D 모델링이 성공적이었는가?					✔
	제어기가 안정적으로 동작했는가?		✔			
	비행시간이 적절했는가?			✔		
SW 항목	SLAM 기술은 잘 구현되었는가?					✔
	Odometry는 SLAM의 Odometry와 일치하는가?				✔	
	표적인식의 위치와 개수는 일치하는가?					✔
	자율주행은 잘 구현되었는가?	✔				
	지도 통합은 성공적이었는가?					✔
	앞으로 개선해야할 점은?	표적이 거울, 유리 등에 비치는 경우의 판단능력 자율주행 및 경로비행				

표 10 프로토타입 실험 점검표

3.3. 최종 결론

본 과제의 주제는 전장상황에서 정찰임무가 가능한 자율비행 드론 개발이었다. 주제를 두 파트, 정찰임무와 자율주행 임무로 구분하였다. 정찰임무란 정찰을 통해 표적을 탐지하고 지도를 작성해서 정찰한 지역에 대해 표적정보가 포함된 2D 지형 지도를 생성하는 것이다. 자율주행 임무란 작성한 지도에 대해 탐색영역과 미 탐색영역을 구분하여 비행경로를 계획하고 주행하는 임무이다. 본 과제 수행을 통해 수동조작을 바탕으로 정찰임무를 수행 가능한 드론을 개발하고, 목표 주제에 대한 실현 가능성을 확인했다.

정찰임무 수행을 위해 아래 그림 64을 출력하는 mapping SW를 개발했다. Yolo v4 모델을 이용해 객체탐지 기능을 구현했고, Google Cartographer를 이용해 지형 지도 작성을 구현했다. 이 두 기능을 통합하기 위해 서로 다른 스레드에서 카메라 Odometry를 측정하며 객체를 탐지해 탐지된 객체의 상대 좌표를 Lidar 및 카메라의 동작 시작점을 기준으로 한 절대 좌표로 계산해 메모리에 저장했다. 이후 프로그램 종료 신호를 받으면 Cartographer의 결과지도 이미지를 불러와 절대 좌표로 계산된 객체의 좌표를 Cartographer 지도 위에 표시했다. 이때 객체의 위치가 신뢰성이 있는지 판단하기 위해 카메라 Odometry도 지도에 표시해 Cartographer의 Lidar Odometry와 비교할 수 있게 했고, 이 완성된 지도를 디스크에 저장해 비휘발성으로 저장해 언제든 다시 결과 사진을 확인해 정찰 결과를 알 수 있게 했다.
자율 주행 임무를 수행하기 위해 Pixhawk를 이용해 아래 그림 65과 같은 드론을 제작했다. Pixhawk에 내장된 센서를 이용해 지원되는 SW를 이용하여 PID 제어가 가능함을 초도 비행을 통해 확인했다. 동역학 및 비행제어 지식을 바탕으로 자율 주행에 필요로 하는 제어 모델을 Matlab을 바탕으로 설계했으나, Pixhawk와 제어기 통합을 구현하지 못했다. 경로 주행 SW 개발에 대해, 깊이 우선 탐색 기법을 이용해 제어 모델을 바탕으로 비행경로 계획법의 기초 개념을 확립했으며, 실시간 지도를 이용해 경로를 생성하는 기술은 구현하지 못했다.

그림 64 통합지도

그림 65 완성된 드론

4. 프로젝트 팀 구성 및 역할 분담

팀원	역할
전병륜(팀장)	드론 HW 설계, 비행성능 계산
김강산	표적탐지기술 구현 및 가상환경 구축
김송섭	지형탐색 기술 구현 및 통합
박보근	드론 동역학 모델링 및 제어기 모델링
전동환	통합지도 생성 SW 구현

표 11 역할 분담

5. 과제 예산 및 내역

- **예산: 1,050,000₩**

항목	비용(원)	
	학과	링크 사업단
기체프레임	186,120	
Flight Controller	217,800	
Pro.peller	20,800	69,920
ESC	56,240	56,240
Telemetry		74,950
3D 프린팅 비용		59,200
회의비		87,500
배터리		대여
모터		대여
카메라		대여
Mini PC		대여
2D LiDAR		대여
1D LiDAR		대여
기타 소모품		200,160
총합	480,960	547,970
	1,028,930	

표 12 예산

Mini PC와 같은 고가의 장비들은 '소프트웨어 구조 및 진화 연구실'에서 대여함

6. 과제 추진 일정

항목	3월				4월				5월				6월
주	1	2	3	4	1	2	3	4	1	2	3	4	1
설계													
	-드론 HW 설계 -SW 아키텍쳐 설계												
수학적 분석													
	-드론 specification 작성 -기자재 주문												
시뮬레이션													
	-구조해석 -표적탐지기술 구현 -지도생성기술 구현												
실험 및 검증													
	-프로토타입 제작 및 시험 -시험결과 평가												
보고서 작성 및 발표													
	-영상자료 촬영 및 중간발표 -보고서 발표												

7. 참고 문헌

연번	종류	제목	저자	발행년도	발행자
1	보고서	군용 드론 시장		2019	연구개발특구진흥재단
2	보고서	군용 드론 시장		2019	연구개발특구진흥재단
3	논문	육군의 5대 게임 체인저 전쟁 판도 바꾼다	유용원	2018	주간조선
4	논문	드론봇 전투부대 편성 및 운용개념에 관한 연구	류창수, 김명환, 정영진	2019	국방과기술
5	논문	드론봇 전투부대 편성 및 운용개념에 관한 연구	류창수, 김명환, 정영진	2019	국방과기술
6	논문	Deep Learning for Generic Object Detection: A Survey		2018	IJCV
7	웹사이트	DRONEII: Tech Talk – Unraveling 5 Levels of Drone Autonomy (https://dronelife.com/2019/03/11/droneii-tech-talk-unraveling-5-levels-of-drone-autonomy/)		2019	dronelife
8	웹사이트	Can I get 3D models and dimensions of my camera? (https://support.stereolabs.com/hc/en-us/articles/360007494333-Can-I-get-3D-models-and-dimensions-of-my-camera-)			Stereo LABS
9	웹사이트	https://www.slamtec.com/en/Support			SLAMTEC
10	웹사이트	LiDAR Lite V3 (https://grabcad.com/library/garmin-lidar-lite-v3-1/details?folder_id=9906189)		2021	GrabCAD
11	웹사이트	Jetson Download Center (https://developer.nvidia.com/embedded/downloads)			NVIDIA Developer
12	웹사이트	Pixhawk 4 Mini (https://docs.px4.io/master/ko/flight_controller/pixhawk4_mini.html)		2021	PX4 autopilot
13	웹사이트	Pixhawk 4 Mini Wiring Quick Start (https://docs.px4.io/v1.9.0/en/assembly/quick_start_pixhawk4_mini.html)		2020	PX4 autopilot

14	웹사이트	비행 전자 제어 - 1. RC 기자재의 신호 체계(https://enssionaut.com/board_aerospace/1391)		2016	Another Name for Engineering Passion
15	웹사이트	MN3508 KV700 (https://store-en.tmotor.com/goods.php?id=356)			T-Motor
16	웹사이트	Tiger MN3508 Brushless Motor (https://grabcad.com/library/tiger-mn3508-brushless-motor-1)		2014	GrabCAD
17	블로그	물체 검출 알고리즘 성능 평가방법 AP(Average Precision)의 이해 (https://bskyvision.com/465)	-	2019	-
18	웹사이트	드론 제작 가이드 2-2편 - 픽스호크 셋팅에서 비행까지 완벽 따라하기 (https://www.youtube.com/watch?v=2z6hNdNp8I0)	Korea Falcon	2020	팰콘샵
19	웹사이트	Ubuntu install of ROS Melodic (http://wiki.ros.org/melodic/Installation/Ubuntu)	ROS.org	2020	Tully Foote
20	웹사이트	rplidar_ros (https://github.com/Slamtec/rplidar_ros)	Slamtec	2019	Slamtec
21	웹사이트	Compiling Cartographer ROS (https://google-cartographer-ros.readthedocs.io/en/latest/compilation.html)	Cartographer ROS	2021	Google
22	웹사이트	zed-python-api (https://github.com/stereolabs/zed-python-api)	stereolabs	2021	stereolabs
23	웹사이트	zed-example/positional tracking (https://github.com/stereolabs/zed-examples/tree/master/positional%20tracking)	stereolabs	2021	stereolabs
24	웹사이트	zed-yolo (https://github.com/stereolabs/zed-yolo)	stereolabs	2021	stereolabs

8. 자체 평가

구분	자체평가서	
과제수행에 관한 전반적인 평가	전장 상황에서 정찰 임무 수행이 가능한 자율 비행 드론개발이라는 주제로 과제를 시작했습니다. 과제의 목표 중 정찰 임무 수행 영역은 해결했지만, 자율 비행이라는 영역을 해결치 못하고 다음 과제로 남겨둔 점이 아쉬움으로 남습니다.	**(83)/100점**
개선할 수 있었던 점	- Mapping SW개발에 있어 완전한 자동화가 이루어지지 않은 점이 아쉽습니다. 최초 목표했던 과제는 자율 비행이 함께 이루어져야 했지만, 기한에 맞추어 성과를 내기 위해 포기한 특징들이 아쉽습니다.	
종합설계에 대한 발전적 제언	- 종합설계 과제수행을 지원할 수 있는 공간이 부족합니다. 전년도의 경우, 공구가 구비된 '조나단' 동아리 방에서 진행하여 복잡한 절차를 거치지 않고 과제를 진행할 수 있었습니다. 올해는 동아리 방이 철거되면서, 제작에 필요한 공구를 구매하고 매일 장소를 옮겨가며 진행하는 불편함이 있었습니다. 작업용 책상과 공구함이 갖추어진 공용 공간 지원을 희망합니다. - 드론을 실제로 날려보기 위한 환경이 부족합니다. 대부분의 학생들이 드론 조종 경험이 부족할뿐더러 직접 만든 드론이나 자율주행을 목표로 하는 드론의 테스트는 꽤 위험할 수 있다고 생각됩니다. 하지만 이러한 테스트를 할 수 있는 안전한 환경은 아직 협소하다고 생각됩니다. 실험동의 무인항공기 제어 실험실에 대한 안내 혹은, 다른 테스트 환경의 제공이 있었으면 좋겠습니다.	

9. 감사의 글 및 후기

팀원	감사의 글 및 후기
전병륜	종합설계 과목을 수강하면서, 프로젝트의 팀장을 담당했습니다. 팀장의 임무를 수행하면서 개인적으로는 제 단점을 파악할 값진 기회였습니다. 가장 두드러진 단점은 커뮤니케이션의 부재였습니다. 팀원들에게 생각을 명확히 전달하지 못해, 프로젝트 진행 중간마다, 생각하는 구성이 달라 수정하는 시간을 많이 소비 했습니다. 또한 발표에 생각보다 훨씬 취약하다는 점도 체감했습니다. 팀원으로서는 임무를 수행하면서 대학 생활을 통해 얼마나 성장했는지 알 기회였습니다. 과제수행에 필요한 학문적인 지식을 갖춤은 물론이고, 필요로 하는 지식 탐구 능력도 갖추었음을 느낄 기회였습니다. 마지막으로, 과제의 방향이 어긋날 때마다, 올바른 방향으로 지도해주신 이선아 지도 교수님께 감사드립니다.
김강산	올해 1월부터 우연히 접하게 된 전장상황에서의 자율비행 경진대회를 시작으로 참 많은 경험을 하게 된 것 같습니다. 리눅스 컴퓨터의 터미널 명령어도 제대로 몰라서 필요할 때 마다 찾아서 쓰던 저는 참 많은 것을 배워가는 것 같습니다. 이번 캡스톤에 필요한 SLAM, 표적탐지 기술 등을 구현하면서 다양한 오픈소스를 찾아서 구현하는 방법, 표적탐지 모델학습에 필요한 cuda 환경 설정 및 pytorch, darknet 프레임워크를 사용할 수 있게 되었고 SLAM의 한 종류인 cartographer를 구현하면서 알게 된 ros와 ros에 사용되는 명령어 및 여러 파일들의 속성을 알게 되었습니다. 이처럼 교내 수업에서 다 배우지 못한 것들을 캡스톤을 통해서 배움으로서 단기간에 정말 많이 성장했다고 느낍니다. 마지막으로 저희 캡스톤 주제에 필요한 대부분의 장비가 고가의 장비였는데 이를 지원해주신 OPPAV 교수님과 참여 박사, 석사, 학부생 분들에게 감사합니다.
김송섭	기술을 구현하고 통합하는 부분에서 생각대로 잘 안 돼서 지체되고 있던 상황에서 팀원들이 잘 이끌어주어서 끝까지 잘 마무리할 수 있었습니다. 오픈소스를 쓰거나 특정 라이브러리를 사용할 때 공식 사이트를 통해 정보를 학습하고 지원하는 환경을 맞추어 주는 것과 구현을 시작하기에 앞서 아키텍처를 통해 생각을 다 같이 정리하는 과정이 얼마나 소중한지를 알게 되었습니다. 이성진 교수님과 이선아 교수님의 피드백 덕분에 부족한 점도 채워가며 결과물의 품질을 더욱 향상시킬 수 있었습니다.
박보근	먼저, 한 학기를 넘어서 1월 팀 구성부터 열심히 달린 모든 팀원들에게 감사를 표합니다. 자율주행, 표적탐지, 지도생성이라는 어려운 주제를 가지고 여기까지 힘들고 지치는 경우도 많았지만 끝까지 노력하였기에 결과가 잘 나왔다고 생각합니다. 5명 팀원 모두가 서로에게 의지하고 도와가며 잘 마무리 해주어 다시 감사합니다. 그리고 매번 저희 캡스톤 주제를 피드백해주시고 장비 대여를 흔쾌히 허락해주신 이선아 교수님 감사합니다. 이번 캡스톤이 앞으로의 공부와 다양한 프로젝트를 수행할 때 좋은 경험으로 쌓여 많은 도움이 될 것입니다.
전동환	저는 함수의 내부 구성을 잘 모르면 응용을 잘 못하는데 오픈소스들이 다들 다른 사람들이 작성한 코드여서 내부 구조를 몰라 오픈소스를 잘 활용하지 못했습니다. 이번 캡스톤 활동을 하며 오픈소스를 분석하며 자세히 읽어보는 시간이 많아서 오픈소스를 토대로 제가 원하는 코드를 작성할 수 있는 능력이 많이 향상된 것 같습니다. 그리고 이런 오픈소스를 분석하면서 몰랐던 파이썬의 코딩 기법들을 배우고 이용해 코딩 능력이 한걸음 발전한 것 같습니다. 제가 오픈소스의 벽에 막혀 방황하고 있을 때 다양한 활동을 하면서 저를 이끌어준 팀원들과 항상 격려해주시며 피드백 해주신 이선아 교수님과 이성진 교수님 정말 감사합니다.

부록 A. 소스코드

A.1. Stack.py(스택 기능 구현)

```
class Stack:
    def __init__(self):
        self.top=[]
    def __len__(self):
        return len(self.top)
    def push(self,item):
        self.top.append(item)
    def pop(self):
        if not self.isEmpty():
            return self.top.pop(-1)
        else:
            print("Stack underflow")
            exit()
    def clear(self):
        self.top=[]
    def isEmpty(self):
        return len(self.top) == 0
```

A.2. Maze.py (미로 생성 및 벽 정보 전달)

```
import numpy as np
class Maze:
    def __init__(self):
        self.maze = []
        self.startX = 0
        self.startY = 0
        self.setDefault()
    # 기본지도
    def setDefault(self):
        self.maze = [[1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
                     [1, 0, 0, 0, 1, 1, 1, 0, 0, 1],
                     [1, 0, 1, 0, 0, 0, 1, 0, 1, 1],
                     [1, 0, 1, 0, 1, 0, 0, 0, 1, 1],
                     [1, 0, 0, 0, 0, 0, 1, 1, 1, 1],
                     [1, 0, 1, 0, 1, 0, 1, 1, 1, 1],
                     [1, 0, 1, 1, 1, 0, 0, 1, 0, 1],
                     [1, 0, 0, 0, 1, 1, 0, 1, 0, 1],
                     [1, 1, 1, 0, 1, 1, 0, 0, 0, 1],
                     [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]
        self.maze = np.array(self.maze, dtype = np.int8)
        self.startX = 1
        self.startY = 1
    def setPosition(self):
        return self.startX, self.startY
    # SLAM; 주변 지역 벽을 탐지해 반환
    # {좌표 : 벽}
    def getWall(self, x, y):
        dic = {}
        for i in (-1, 1):
            if(0 <= x+i and x+i < self.maze.shape[0]):
```

```
                dic[ (x+i, y) ] = self.maze[x+i, y]
        for j in (-1,1):
            if(0 <= y+j and y+i < self.maze.shape[1]):
                dic[ (x, y+i) ] = self.maze[x, y+i]
        return dic
```

A.3. DFS.py (깊이 우선 탐색을 이용한 지도 작성)

```
import numpy as np
from Stack import Stack
from Maze import Maze
# 초기화
stack = Stack()
maze = Maze() # 실제 지도
map_ = np.ones((2, 2), dtype=np.int8) # 드론이 그리는 지도
odom = np.zeros((2, 2), dtype=np.int8) # 드론의 이동 경로
dic = {}
x, y = maze.setPosition() # 초기 위치
odom[x, y] = 1
map_[x, y] = 0
while True:
    dic = maze.getWall(x,y) # SLAM; 주변의 벽을 탐지(4방향)
    for i in dic.keys():
        # 현재 가지고 있는 지도 밖의 벽 혹은 길이 나왔을 때
        # 지도의 크기를 키운다.
        if(map_.shape[0] <= i[0]):
            map_2 = np.ones((i[0] + 1, map_.shape[1]), dtype=np.int8)
            odom2 = np.zeros((i[0] + 1, map_.shape[1]), dtype=np.int8)
            map_2[:map_.shape[0], :map_.shape[1]] = map_
            odom2[:map_.shape[0], :map_.shape[1]] = odom
            map_ = map_2
            odom = odom2
        if(map_.shape[1] <= i[1]):
            map_2 = np.ones((map_.shape[0], i[1] + 1), dtype=np.int8)
            odom2 = np.zeros((map_.shape[0],i[1] + 1), dtype=np.int8)
            map_2[:map_.shape[0], :map_.shape[1]] = map_
            odom2[:map_.shape[0], :map_.shape[1]] = odom
            map_ = map_2
            odom = odom2
        # SLAM에서 얻은 데이터에서 길을 지도에 표시 후 스택에 저장
        if dic[i] == 0 and odom[i] != 1:
            map_[i] = dic[i]
            stack.push(i)
    # 갈 수 있는 모든 길을 이동했을 때 알고리즘 종료
    if stack.isEmpty():
        break
    x, y = stack.pop()
    odom[x, y] = 1 # x,y로 이동한 것 기록
    # 미로에서 탈출 했을 때 알고리즘 종료
    if y == maze.maze.shape[1] - 1:
        break
# 결과 출력
print("odometry\n", odom)
print("map\n", map_)
```

```
print("position : (" + str(x) + ', ' + str(y) + ')')
print('ground_truth\n', maze.maze)
print('sucess')
```

A.4. reset.py (비정상적 종료로 인해 Lidar가 켜져 있을 경우)

```
from rplidar import RPLidar

def reset():
    lidar = RPLidar(port="/dev/ttyUSB0", baudrate=256000, timeout=3)
    # Linux     : "/dev/ttyUSB0"
    # MacOS     : "/dev/cu.SLAB_USBtoUART"
    # Windows   : "COM5"
    lidar.stop()
    lidar.stop_motor()
    lidar.disconnect()

if __name__ == "__main__":
    reset()
```

A.5. zed.py (카메라 Odometry 및 표적탐지 결과 조율)

```
import sys
import os.path
import glob
import pyzed.sl as sl
import darknet_zed as dark
import math
import time

# ID 구분을 위한 객체
class Object:
    def __init__(self, label, x, y):
        self.label = label
        self.x = x; self.y = y
        self.time = time.time()

    # 비슷한 시간, 비슷한 위치에서 같은 객체가 감지될 경우 같은 ID로 간주
    def isSameID(self, label, x, y):
        if label != self.label: returnFalse
        if time.time() - self.time > 3: returnFalse
        if (self.x-x)**2 + (self.y-y)**2 > 100000: returnFalse;
        self.x = x
        self.y = y
        self.time = time.time()
        returnTrue

# 카메라 제어를 위한 객체
class Camera:
    def __init__(self):
        init = sl.InitParameters(camera_resolution=sl.RESOLUTION.HD720,
                                 coordinate_units=sl.UNIT.MILLIMETER,
```

```
coordinate_system=sl.COORDINATE_SYSTEM.RIGHT_HANDED_Y_UP)
        self.zed = sl.Camera()
        status = self.zed.open(init)
        if status != sl.ERROR_CODE.SUCCESS:
            print(repr(status))
            exit()

        # for Odomeytry
        self.tracking_params = sl.PositionalTrackingParameters()
        self.zed.enable_positional_tracking(self.tracking_params)

        self.runtime = sl.RuntimeParameters()
        self.camera_pose = sl.Pose()

        self.camera_info = self.zed.get_camera_information()

        self.py_translation = sl.Translation()
        self.pose_data = sl.Transform()

        self.translation = [0, 0, 0]
        self.rotation = [0, 0, 0]

        # for Object Detection
        self.thresh = 0.4

        self.darknet_path="./Darknet/darknet/"
        self.config_path          =          self.darknet_path          +
"cfg/custom-yolov4-detector.cfg"
        self.weight_path                                                  =
"./Darknet/darknet/custom-yolov4-detector_best.weights"
        self.meta_path = self.darknet_path + "data/obj.data"

        self.svo_path = None
        self.zed_id = 0

        self.obj_id = []

        self.mat = sl.Mat()
        self.point_cloud_mat = sl.Mat()
        if dark.netMain isNone:
            dark.netMain = dark.load_net_custom(
                    self.config_path.encode("ascii"),
                    self.weight_path.encode("ascii"),
                    0, 1) # batch size = 1
        if dark.metaMain isNone:
            dark.metaMain = dark.load_meta(
                self.meta_path.encode("ascii"))
        if dark.altNames isNone:
```

```
            try:
                with open(self.meta_path) as meta_fh:
                    meta_contents = meta_fh.read()
                    import re
                    match = re.search("names *= *(.*)$", meta_contents,
                                      re.IGNORECASE | re.MULTILINE)
                    if match:
                        result = match.group(1)
                    else:
                        result = None
                    try:
                        if os.path.exists(result):
                            with open(result) as names_fh:
                                names_list                                  =
names_fh.read().strip().split("\n")
                                altNames    =    [x.strip()    for    x    in
names_list]
                    except TypeError:
                        pass
            except Exception:
                pass

    # 카메라 연결 해제 및 카메라 종료
    def close(self):
        self.zed.disable_object_detection()
        self.zed.disable_positional_tracking()
        self.zed.close()

    # Odometry 측정 및 반환( [x, y, z], [roll, pitch, yaw] )
    def odometry(self):
        if self.zed.grab(self.runtime) == sl.ERROR_CODE.SUCCESS:
            tracking_state = self.zed.get_position(self.camera_pose)
            if tracking_state == sl.POSITIONAL_TRACKING_STATE.OK:
                rotation = self.camera_pose.get_rotation_vector()
                translation                                                 =
self.camera_pose.get_translation(self.py_translation)

                # [x_r, y_r, z_r] in aero coordinate system
                self.rotation    =    [-rotation[2],    rotation    [0],
-rotation[1]]
                # [x, y, z] in aero coordinate system
                self.translation = [-translation.get()[2],
                        translation.get()[0],
                        -translation.get()[1]]

                self.pose_data                                              =
self.camera_pose.pose_data(sl.Transform())

        yield self.translation, self.rotation
```

```
    # 객체 탐지 결과 반환( (x, y), label )
    def obj_detection(self):
        if self.zed.grab(self.runtime) == sl.ERROR_CODE.SUCCESS:
            self.zed.retrieve_image(self.mat, sl.VIEW.LEFT)
            image = self.mat.get_data()

            self.zed.retrieve_measure(
                self.point_cloud_mat, sl.MEASURE.XYZRGBA)
            depth = self.point_cloud_mat.get_data()

            # Do the detection
            detections    =    dark.detect(dark.netMain,    dark.metaMain,
image, self.thresh)

            for detection in detections:
                label = detection[0]
                confidence = detection[1] # correct rate
                bounds = detection[2]
                flag = True
                for i in self.obj_id:
                    x=bounds[0]-bounds[2]/2
                    y=bounds[1]-bounds[3]/2
                    if i.isSameID(label, bounds[0], bounds[1]):
                        flag = False
                        continue

                    else:
                        continue
                if flag:
                    self.obj_id.append(Object(label,           bounds[0],
bounds[1]))
                    x, y, z = dark.get_object_depth(depth, bounds)
                    distance = math.sqrt(x * x + y * y + z * z)
                    print("label  =  {},  distance  =  {}".format(label,
distance))
                    if z isnot -1 and confidence > .7:
                        yield (-z, -x), label
                else: continue
```

A.6. main.py (Cartographer와 ZED 카메라 Odometry, 표적탐지 결과 통합)

```
import os
import glob
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as img
import threading
import pyzed.sl as sl

from matplotlib.offsetbox import OffsetImage, AnnotationBbox
```

```
from zed import Camera
from math import pi, sin, cos

global cam
global lidar

cam = Camera()
print('Camera Ready')

global pose
global obj_det

# Odometry 좌표 및 상태
global drone_x
global drone_y
global drone_rot
drone_x = []
drone_y = []
drone_rot = []

global isFinished
global isOdom
isFinished = False
isOdom = False

global exit_event
exit_event = threading.Event()

# Odometry 실행용 스레드
def odometry():
    global isOdom
    global drone_x
    global drone_y
    global drone_rot

    try:
        while not isFinished:
            if exit_event.is_set():
                break
            pose = cam.odometry()
            for translation, rotation in pose:
                drone_x.append(translation[0])
                drone_y.append(translation[1])
                drone_rot = [rotation[0], rotation[1], rotation[2]]
                isOdom = True

    finally:
        print('Odometry Finish')
```

```
global x
global y
global x_now
global y_now
global x_val
global y_val

x = []
y = []
x_now = []
y_now = []
x_val = {}
y_val = {}

# 객체 좌표
global obj_x
global obj_y
obj_x = {}
obj_y = {}

# 객체 이미지 및 변환 비율
obj_img = {
        "b'radioactivity'"   :   './img/Radiation.png',
        "b'soldier'"         :   './img/Soldier.png'}
obj_zoom = {
        "b'radioactivity'" :   .02,
        "b'soldier'"       :   .1}

# 객체 탐지용 스레드
def obj_detect():
    global obj_x
    global obj_y
    while not isFinished:
        if isOdom:
            obj_det = cam.obj_detection()
            for pos, label in obj_det:
                flag = False

                if label isNone: continue
                if label in obj_x:
                  for was in zip(obj_x[label], obj_y[label]):
                    if (pos[0] - was[0])**2 + (pos[1] - was[1])**2 <
4000000:
                        flag = True
                        break
                if flag: continue

                if label in obj_x:
                    print(label)
```

```
                    obj_x[label].append(drone_x[-1]                    +
pos[0]*cos(drone_rot[2]) + pos[1]*sin(drone_rot[2]))
                    obj_y[label].append(drone_y[-1]                    -
pos[0]*sin(drone_rot[2]) + pos[1]*cos(drone_rot[2]))
                    print(obj_x[label], obj_y[label])
                else:
                    print(label)
                    obj_x[label]         =         [drone_x[-1]         +
pos[0]*cos(drone_rot[2]) + pos[1]*sin(drone_rot[2])]
                    obj_y[label]         =         [drone_y[-1]         -
pos[0]*sin(drone_rot[2]) + pos[1]*cos(drone_rot[2])]
                    print(obj_x[label], obj_y[label])
    print('Object Detection Finish')

# ZED 카메라 영상 저장
def video_recode():
    path_output = './video/test.svo'
    recording_param        =        sl.RecordingParameters(path_output,
sl.SVO_COMPRESSION_MODE.H264)
    err = cam.zed.enable_recording(recording_param)
    if err != sl.ERROR_CODE.SUCCESS:
            print('Video Recording Error')
            exit(1)
    frames_recorded = 0
    whilenot isFinished:
        continue

iteration = 0

# Threading
try:
  pose = cam.odometry()
  obj_det = cam.obj_detection()

  odom_thread = threading.Thread(target=odometry, args=())
  obj_detect_thread = threading.Thread(target=obj_detect)
  video_recode_thread = threading.Thread(target=video_recode)
  isFinished = False

  odom_thread.start()
  obj_detect_thread.start()
  video_recode_thread.start()
  while True:
      continue

except KeyboardInterrupt:
    print('interrupt')
    isFinished = True
    odom_thread.join()
```

```
    obj_detect_thread.join()
    video_recode_thread.join()
    isFinished = True

# 병합
finally:
    if not isFinished:
        isFinished = True

    cam.close()

    fig, sub = plt.subplots()
    car_img = "./car_img/test.png“

    # Cartographer 결과 이미지가 저장되길 기다림
    while not os.path.isfile(car_img):
        continue
    print("Merge start")
    image = img.imread(car_img)
    plt.imshow(image)

    map_x = image.shape[1] / 2
    map_y = image.shape[0] / 2

    # 좌표 계산 및 통합 지도 생성
    drone_x_ar = np.array(drone_x)
    drone_y_ar = np.array(drone_y)
    drone_x_arr = map_x + drone_x_ar * .02059
    drone_y_arr = map_y + drone_y_ar * .02059
    plt.plot(drone_x_arr, drone_y_arr, 'red')
    for label in obj_x:
        image = plt.imread(obj_img[str(label)])
        img_offset = OffsetImage(image, zoom=obj_zoom[str(label)])
        for img_x, img_y in zip(obj_x[label], obj_y[label]):
            img = AnnotationBbox(img_offset, (map_x + img_x * .02059,
map_y + img_y * .02059), frameon=False)
            sub.add_artist(img)

    # 지도 저장 및 프로그램 종료
    plt.savefig('./result/result.png')
    print("Successful Completion")
```

부록 B. SW아키텍처

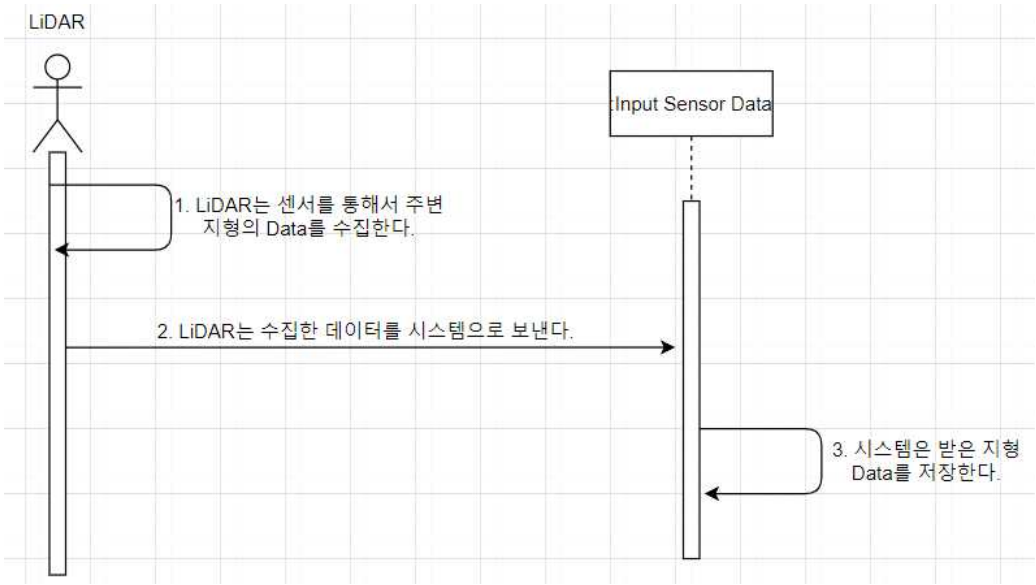


그림 66 시퀀스 다이어그램 (지형탐색)

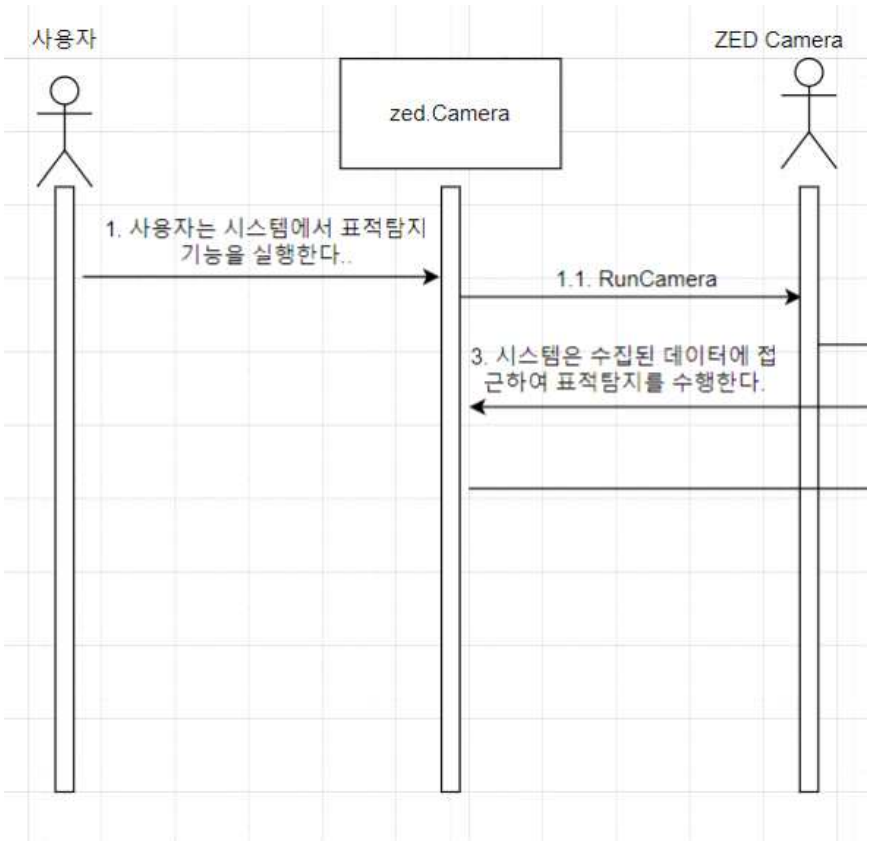


그림 67 시퀀스 다이어그램 (표적탐지1)

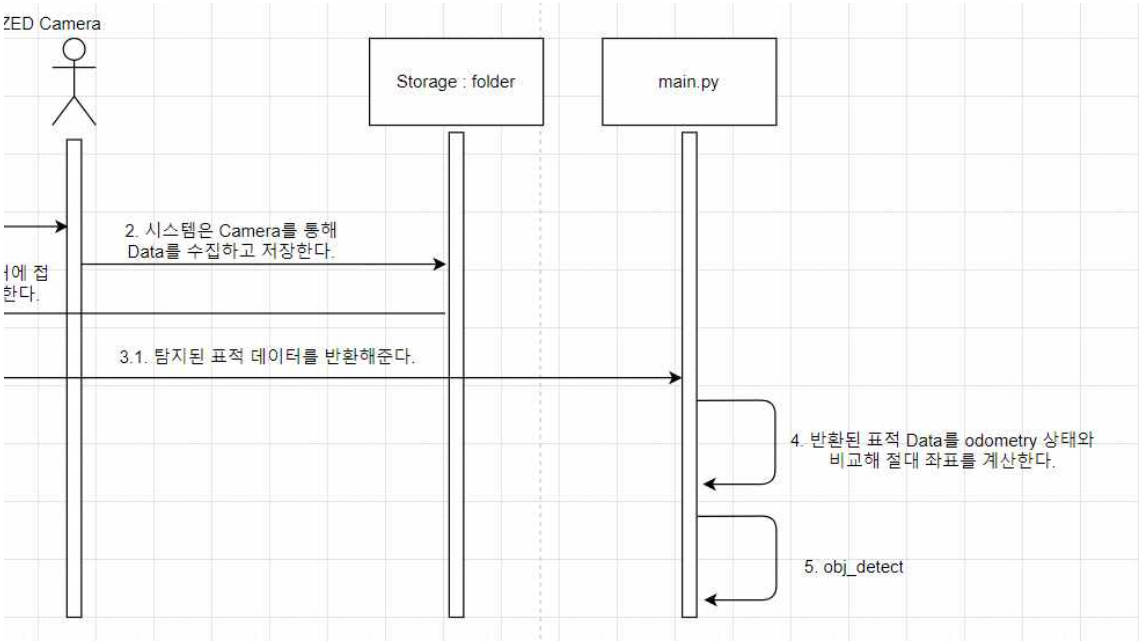


그림 68 시퀀스 다이어그램 (표적탐지2)

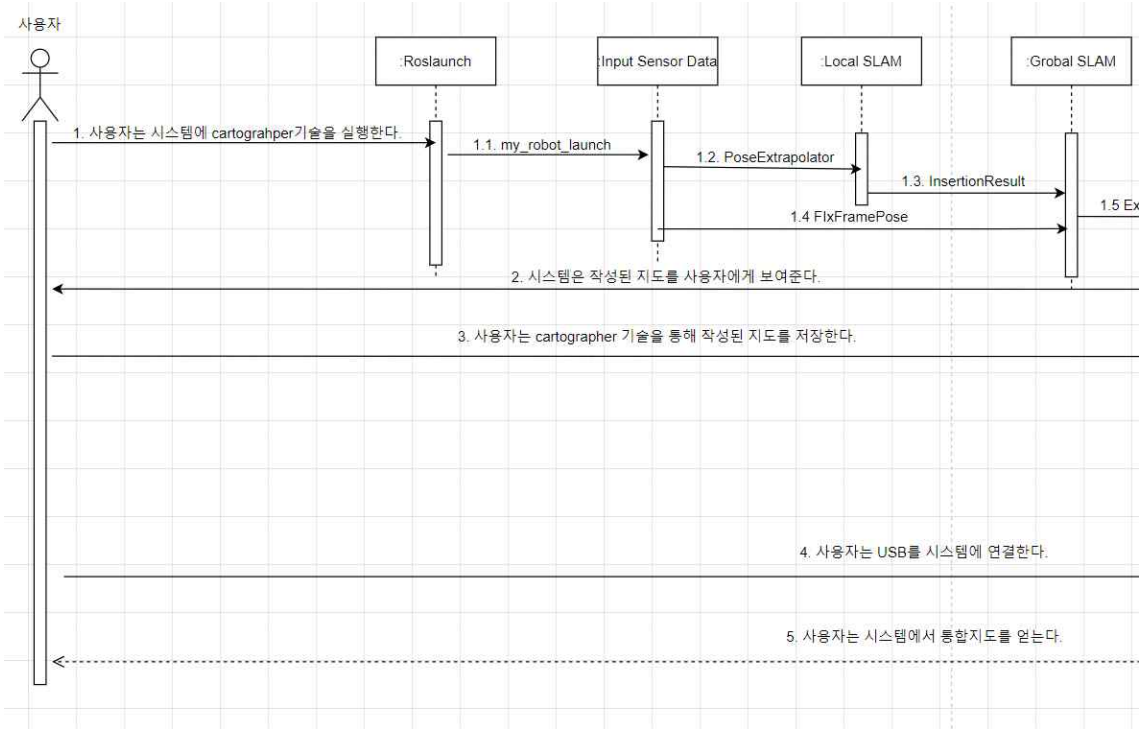


그림 69 시퀀스 다이어그램 (통합지도작성1)

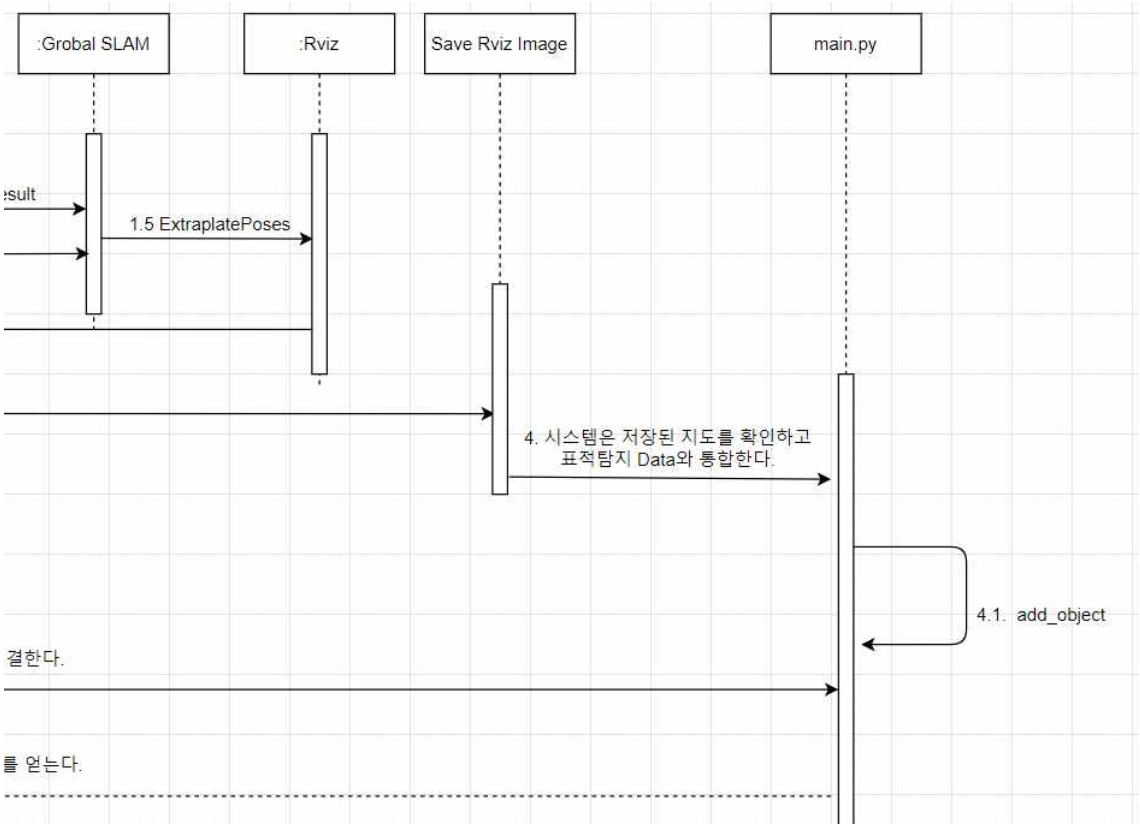


그림 70 시퀀스 다이어그램 (통합지도작성2)

부록 C. 추력 시험 진행 과정

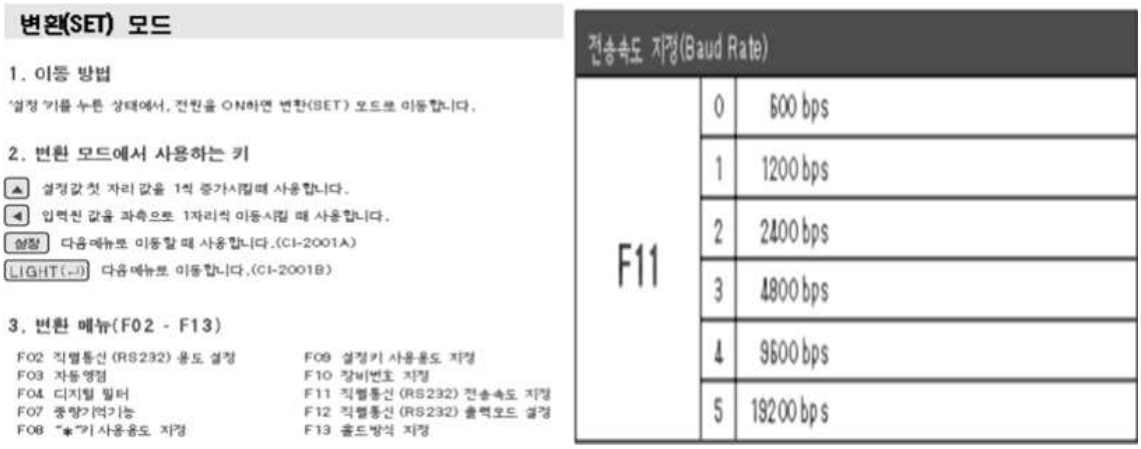
준비물	
HW	로드셀(CBCL-50L), 알루미늄 프로파일, 모터마운트, 모터, 프로펠러(12*5.5), Indicator(CI-2001A), 조종기, 수신기, 변속기, 배터리
SW	comportmaster

표 17 추력 시험 준비물

그림 71과 같이 알루미늄 프로파일을 이용하여 모터 추력에 따라 시험대가 흔들리지 않도록 제작하였다. 추력은 Load Cell을 이용해 측정했으며, 수집한 데이터는 Indicator를 거쳐 comportmaster에서 확인할 수 있다.

그림 71 추력 시험대, Load Cell, Indicator, comportmaster

PC와 Indicator의 연결은 포트 번호와 Baud Rate 동기화가 필요하다. Indicator의 설명서 그림 72을 따라 Baud Rate를 설정하고, comportmaster에서 포트 번호와 Baudrate를 동기화한다. 그림 73에 따라 테스트 모드를 통해 컴퓨터와의 연결을 테스트를 진행하고, 이상이 없다면 실험을 진행한다. 조종기는 Throttle(%)를 조종기 화면에 표시해준다. 10% 단위로 측정한다.



변환(SET) 모드

1. 이동 방법

'설정'키를 누른 상태에서, 전원을 ON하면 변환(SET) 모드로 이동합니다.

2. 변환 모드에서 사용하는 키

▲ 설정값 첫 자리 값을 1씩 증가시킬때 사용합니다.
◀ 입력된 값을 좌측으로 1자리씩 이동시킬 때 사용합니다.
설정 다음메뉴로 이동할 때 사용합니다.(CI-2001A)
LIGHT(↵) 다음메뉴로 이동합니다.(CI-2001B)

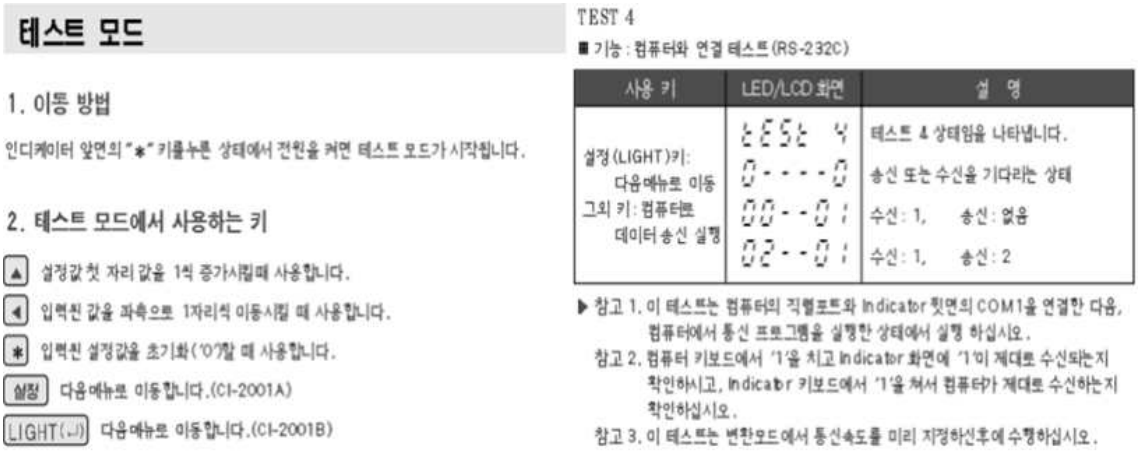
3. 변환 메뉴(F02 - F13)

F02 직렬통신 (RS232) 용도 설정
F03 자동영점
F04 디지털 필터
F07 중량기억기능
F08 "*"키 사용용도 지정
F09 설정키 사용용도 지정
F10 장비번호 지정
F11 직렬통신 (RS232) 전송속도 지정
F12 직렬통신 (RS232) 출력모드 설정
F13 홀드방식 지정

전송속도 지정(Baud Rate)

F11	0	600 bps
	1	1200 bps
	2	2400 bps
	3	4800 bps
	4	9600 bps
	5	19200 bps

그림 72 CI-2001A 설명서에 따른 Baud Rate 설정 방법



테스트 모드

1. 이동 방법

인디케이터 앞면의 "*" 키를누른 상태에서 전원을 켜면 테스트 모드가 시작됩니다.

2. 테스트 모드에서 사용하는 키

▲ 설정값 첫 자리 값을 1씩 증가시킬때 사용합니다.
◀ 입력된 값을 좌측으로 1자리씩 이동시킬 때 사용합니다.
* 입력된 설정값을 초기화('0')할 때 사용합니다.
설정 다음메뉴로 이동합니다.(CI-2001A)
LIGHT(↵) 다음메뉴로 이동합니다.(CI-2001B)

TEST 4

■ 기능 : 컴퓨터와 연결 테스트(RS-232C)

사용 키	LED/LCD 화면	설 명
설정(LIGHT)키: 다음메뉴로 이동 그외 키: 컴퓨터로 데이터송신 실행	tESt 4	테스트 4 상태임을 나타냅니다.
	0----0	송신 또는 수신을 기다리는 상태
	00--01	수신 : 1, 송신 : 없음
	02--01	수신 : 1, 송신 : 2

▶ 참고 1. 이 테스트는 컴퓨터의 직렬포트와 Indicator 뒷면의 COM1을 연결한 다음, 컴퓨터에서 통신 프로그램을 실행한 상태에서 실행 하십시오.
참고 2. 컴퓨터 키보드에서 '1'을 치고 Indicator 화면에 '1'이 제대로 수신되는지 확인하시고, Indicator 키보드에서 '1'을 쳐서 컴퓨터가 제대로 수신하는지 확인하십시오.
참고 3. 이 테스트는 변환모드에서 통신속도를 미리 지정하신후에 수행하십시오.

그림 73 설명서에 따른 컴퓨터와 연결 확인

부록 D. Cartographer 수정 파일

*** my_robot.launch**

```
<!--
  Copyright 2016 The Cartographer Authors
  Licensed under the Apache License, Version 2.0 (the "License");
  you may not use this file except in compliance with the License.
  You may obtain a copy of the License at
       http://www.apache.org/licenses/LICENSE-2.0
  Unless required by applicable law or agreed to in writing, software
  distributed under the License is distributed on an "AS IS" BASIS,
  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or imp
lied.
  See the License for the specific language governing permissions and
  limitations under the License.
-->
<launch>
  <!-- Load robot description and start state publisher-->
    <param name="robot_description" textfile="$(find cartographer_ros)/urdf/head
_2d.urdf" />
<node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_sta
te_publisher" />


  <!-- Start RPLIDAR sensor node which provides LaserScan data  -->
    <node name="rplidarNode" pkg="rplidar_ros" type="rplidarNode" output="sc
reen">
        <param name="serial_port" type="string" value="/dev/ttyUSB0"/>
        <param name="serial_baudrate" type="int" value="256000"/>
        <param name="frame_id" type="string" value="laser"/>
        <param name="inverted" type="bool" value="false"/>
        <param name="angle_compensate" type="bool" value="true"/>
    </node>


  <!-- Start Google Cartographer node with custom configuration file-->
    <node name="cartographer_node" pkg="cartographer_ros" type="cartographe
r_node" args="
          -configuration_directory
             $(find cartographer_ros)/configuration_files
          -configuration_basename my_robot.lua" output="screen">
    </node>
<!-- Additional node which converts Cartographer map into ROS occupancy grid
map. Not used and can be skipped in this case -->
    <node name="cartographer_occupancy_grid_node" pkg="cartographer_ros" ty
pe="cartographer_occupancy_grid_node" args="-resolution 0.05" />
```

```
</launch>
```

* visualization.launch

```
<launch>
  <!-- Start RViz with custom view -->
  <node pkg="rviz" type="rviz" name="show_rviz" args="-d $(find cartographer_r
os)/rviz/demo.rviz"/>
</launch>
```

* my_robot.lua

```
-- Copyright 2016 The Cartographer Authors
--
-- Licensed under the Apache License, Version 2.0 (the "License");
-- you may not use this file except in compliance with the License.
-- You may obtain a copy of the License at
--
--         http://www.apache.org/licenses/LICENSE-2.0
--
-- Unless required by applicable law or agreed to in writing, software
-- distributed under the License is distributed on an "AS IS" BASIS,
-- WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or im
plied.
-- See the License for the specific language governing permissions and
-- limitations under the License.
include "map_builder.lua"
include "trajectory_builder.lua"
options = {
  map_builder = MAP_BUILDER,
  trajectory_builder = TRAJECTORY_BUILDER,
  map_frame = "map",
  tracking_frame = "base_link",
  published_frame = "base_link",
  odom_frame = "odom",
  provide_odom_frame = true,
  publish_frame_projected_to_2d = true,
  use_odometry = false,
  use_nav_sat = false,
  use_landmarks = false,
  num_laser_scans = 1,
  num_multi_echo_laser_scans = 0,
  num_subdivisions_per_laser_scan = 1,
```

```
  num_point_clouds = 0,
  lookup_transform_timeout_sec = 0.2,
  submap_publish_period_sec = 0.3,
  pose_publish_period_sec = 5e-3,
  trajectory_publish_period_sec = 30e-3,
  rangefinder_sampling_ratio = 1.,
  odometry_sampling_ratio = 1.,
  fixed_frame_pose_sampling_ratio = 1.,
  imu_sampling_ratio = 1.,
  landmarks_sampling_ratio = 1.,
}MAP_BUILDER.use_trajectory_builder_2d = true
TRAJECTORY_BUILDER_2D.min_range = 0.5
TRAJECTORY_BUILDER_2D.max_range = 8.
TRAJECTORY_BUILDER_2D.missing_data_ray_length = 8.5
TRAJECTORY_BUILDER_2D.use_imu_data = false
TRAJECTORY_BUILDER_2D.use_online_correlative_scan_matching = true
TRAJECTORY_BUILDER_2D.real_time_correlative_scan_matcher.linear_search_window
= 0.1
TRAJECTORY_BUILDER_2D.real_time_correlative_scan_matcher.translation_delta_cost_
weight = 10.
TRAJECTORY_BUILDER_2D.real_time_correlative_scan_matcher.rotation_delta_cost_wei
ght = 1e-1
TRAJECTORY_BUILDER_2D.motion_filter.max_angle_radians = math.rad(0.2)
-- for current lidar only 1 is good value
TRAJECTORY_BUILDER_2D.num_accumulated_range_data = 1
POSE_GRAPH.constraint_builder.min_score = 0.65
POSE_GRAPH.constraint_builder.global_localization_min_score = 0.65
POSE_GRAPH.optimization_problem.huber_scale = 1e2
POSE_GRAPH.optimize_every_n_nodes = 35
return options
```

* head_2d.urdf

```
<robot name="head_2d">
  <material name="orange">
    <color rgba="1.0 0.5 0.2 1" />
  </material>
  <material name="gray">
    <color rgba="0.2 0.2 0.2 1" />
  </material>
  <link name="laser">
    <visual>
      <origin xyz="0 0 0" />
      <geometry>
        <cylinder length="0.03" radius="0.03" />
      </geometry>
      <material name="gray" />
```

```
    </visual>
  </link>
  <link name="base_link">
    <visual>
      <origin xyz="0.01 0 0.015" />
      <geometry>
        <box size="0.11 0.065 0.052" />
      </geometry>
      <material name="orange" />
    </visual>
  </link>
  <joint name="laser_joint" type="fixed">
    <parent link="base_link" />
    <child link="laser" />
    <origin rpy="0 0 3.1415926" xyz="0 0 0.05" />
  </joint>
</robot>
```

*** demo.rviz**

```
Panels:
  - Class: rviz/Displays
    Help Height: 78
    Name: Displays
    Property Tree Widget:
      Expanded:
        - /TF1/Frames1
      Splitter Ratio: 0.6006709933280945
    Tree Height: 728
  - Class: rviz/Selection
    Name: Selection
  - Class: rviz/Tool Properties
    Expanded:
      - /2D Pose Estimate1
      - /2D Nav Goal1
      - /Publish Point1
    Name: Tool Properties
    Splitter Ratio: 0.5886790156364441
  - Class: rviz/Views
    Expanded:
      - /Current View1
    Name: Views
    Splitter Ratio: 0.5
  - Class: rviz/Time
    Experimental: false
    Name: Time
    SyncMode: 0
    SyncSource: PointCloud2
Preferences:
  PromptSaveOnExit: true
Toolbars:
```

```
  toolButtonStyle: 2
Visualization Manager:
  Class: ""
  Displays:
    - Alpha: 0.5
      Cell Size: 1
      Class: rviz/Grid
      Color: 160; 160; 164
      Enabled: true
      Line Style:
        Line Width: 0.029999999329447746
        Value: Lines
      Name: Grid
      Normal Cell Count: 0
      Offset:
        X: 0
        Y: 0
        Z: 0
      Plane: XY
      Plane Cell Count: 100
      Reference Frame: <Fixed Frame>
      Value: true
    - Class: rviz/TF
      Enabled: true
      Frame Timeout: 15
      Frames:
        All Enabled: false
        base_link:
          Value: true
        laser:
          Value: false
        map:
          Value: true
        odom:
          Value: true
      Marker Scale: 1
      Name: TF
      Show Arrows: true
      Show Axes: true
      Show Names: true
      Tree:
        map:
          odom:
            base_link:
              laser:
                {}
      Update Interval: 0
      Value: true
    - Class: Submaps
      Enabled: true
      Fade-out distance: 1
      High Resolution: true
      Low Resolution: false
      Name: Submaps
      Submap query service: /submap_query
      Submaps:
```

```
      All: true
      All Submap Pose Markers: true
      Trajectory 0:
        0.180: true
        1.180: true
        10.180: true
        11.180: true
        12.180: true
        13.180: true
        14.180: true
        15.180: true
        16.180: true
        17.180: true
        18.180: true
        19.180: true
        2.180: true
        20.180: true
        21.180: true
        22.180: true
        23.180: true
        24.180: true
        25.180: true
        26.180: true
        27.113: true
        28.23: true
        3.180: true
        4.180: true
        5.180: true
        6.180: true
        7.180: true
        8.180: true
        9.180: true
        Submap Pose Markers: true
        Value: true
    Topic: /submap_list
    Tracking frame: base_link
    Unreliable: false
    Value: true
  - Alpha: 1
    Autocompute Intensity Bounds: true
    Autocompute Value Bounds:
      Max Value: 10
      Min Value: -10
      Value: true
    Axis: Z
    Channel Name: intensity
    Class: rviz/PointCloud2
    Color: 0; 255; 0
    Color Transformer: FlatColor
    Decay Time: 0
    Enabled: true
    Invert Rainbow: false
    Max Color: 255; 255; 255
    Max Intensity: 4096
    Min Color: 0; 0; 0
    Min Intensity: 0
```

```
    Name: PointCloud2
    Position Transformer: XYZ
    Queue Size: 10
    Selectable: true
    Size (Pixels): 3
    Size (m): 0.05000000074505806
    Style: Flat Squares
    Topic: /scan_matched_points2
    Unreliable: false
    Use Fixed Frame: true
    Use rainbow: true
    Value: true
  - Class: rviz/MarkerArray
    Enabled: true
    Marker Topic: /trajectory_node_list
    Name: MarkerArray
    Namespaces:
      Trajectory 0: true
    Queue Size: 100
    Value: true
  - Alpha: 0.699999988079071
    Class: rviz/Map
    Color Scheme: map
    Draw Behind: false
    Enabled: false
    Name: Map
    Topic: /map
    Unreliable: false
    Use Timestamp: false
    Value: false
  - Alpha: 1
    Class: rviz/RobotModel
    Collision Enabled: false
    Enabled: true
    Links:
      All Links Enabled: true
      Expand Joint Details: false
      Expand Link Details: false
      Expand Tree: false
      Link Tree Style: Links in Alphabetic Order
      base_link:
        Alpha: 1
        Show Axes: false
        Show Trail: false
        Value: true
      laser:
        Alpha: 1
        Show Axes: false
        Show Trail: false
        Value: true
    Name: RobotModel
    Robot Description: robot_description
    TF Prefix: ""
    Update Interval: 0
    Value: true
    Visual Enabled: true
```

```
  Enabled: true
  Global Options:
    Background Color: 100; 100; 100
    Default Light: true
    Fixed Frame: map
    Frame Rate: 30
  Name: root
  Tools:
    - Class: rviz/Interact
      Hide Inactive Objects: true
    - Class: rviz/MoveCamera
    - Class: rviz/Select
    - Class: rviz/FocusCamera
    - Class: rviz/Measure
    - Class: rviz/SetInitialPose
      Theta std deviation: 0.2617993950843811
      Topic: /initialpose
      X std deviation: 0.5
      Y std deviation: 0.5
    - Class: rviz/SetGoal
      Topic: /move_base_simple/goal
    - Class: rviz/PublishPoint
      Single click: true
      Topic: /clicked_point
  Value: true
  Views:
    Current:
      Angle: 0
      Class: rviz/TopDownOrtho
      Enable Stereo Rendering:
        Stereo Eye Separation: 0.05999999865889549
        Stereo Focal Distance: 1
        Swap Stereo Eyes: false
        Value: false
      Invert Z Axis: false
      Name: Current View
      Near Clip Distance: 0.009999999776482582
      Scale: 20.697040557861328
      Target Frame: <Fixed Frame>
      Value: TopDownOrtho (rviz)
      X: 0
      Y: 0
    Saved: ~
Window Geometry:
  Displays:
    collapsed: false
  Height: 1025
  Hide Left Dock: false
  Hide Right Dock: false
  QMainWindow State: 000000ff00000000fd00000004000000000000015600000363f
c0200000008fb0000001200530065006c00650063007400690065006e00000001e10000
009b0000005c00fffffffb0000001e0054006f006f006c00200050007200650072
00740069006500730200000 1ed000001df000001850000000a3fb0000001200560069006
5007700730020005400 6f006f02000001df0000021100000185000000122fb0000002000
54006f006f006c00200050007200650072007400690065007300320300000028
8000001 1d000002210000017afb0000001000440069007300700006c006100790073010
000003d000003630000000c900ffffffb0000002000730065006c00650063007400690065006f
006e00200062007500660066006500720200000138000000aa0000023a00000294fb00
0000140057006900640065005300740065007200650 6f02000000e60000000d2000003
```

ee0000030bfb0000000c004b0069006e00650063007402000001860000010600000030c
0000026100000001000001000000363fc02000000003fb0000001e0054006f006f006c00
2000500072006f00700065007200740069006500730100000041000000780000000000
000000fb00000000a00560069006500770073010000003d00000363000000a400ffffffb0
0000012005300650006c00650063007400690006f006e010000025a000000b2000000000
0000000000000020000049000000a9fc0100000001fb0000000a00560069006500770073
03000000004e00000080000002e1000001970000000030000073d00000003efc010000000
02fb0000000800540069006d00650100000000000000073d000002eb00fffffffb000000080
054006900 6d00650100000000000004500000000000000000000004db00000363000000
0040000000400000008000000008fc00000001000000020000000100000000a0054006f00
6f006c00730100000000ffffffff0000000000000000
  Selection:
    collapsed: false
  Time:
    collapsed: false
  Tool Properties:
    collapsed: false
  Views:
    collapsed: false
  Width: 1853
  X: 67
  Y: 27