0

|  |
| --- |
|  |
|  |
|  |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
| <https://www.e4ds.com/sub_view.asp?ch=16&t=0&idx=16176>)  이렇게 투자되는 연구 자금 덕에, 기술력이 많이 향상되어 국내 사업분야들은 일반적으로 드론 하면 떠오르는 항공 뷰 촬영 및 단순 조종이 아니라 특수 목적 드론에 연구 초점이 많이 맞춰지고 있습니다.  좋은 상황이지만, 특수 목적 드론의 경우 (1)전문 교육을 받은 운용자들도 운용 시 어려움을 느끼는 점(특수 목적 드론 기술의 복잡성, 실전 경험 부족 등), (2)아직까지는 일부 자동화 기능의 성능이 아쉬운 점, (3)일반적으로 고정된 운용 거점을 활용하는 경우가 많아 유동적인 활용이 어려운 점 등의 문제점들이 있습니다.  (위 근거는 특수 목적 드론을 생산하고 특정 제품에서 전투용 적합 판정을 받았었던 중소업체 내부의 운용자와 개발자 시점에서 의견을 바탕으로 작성하였습니다)  따라서 위 문제점들을 상세히 파악하여 해결해보는 방향으로 과제 초점을 잡게 되었습니다.  <문제점 1 설명>  먼저 드론은 비행체이므로 일반 자동차처럼 4방향 조종이 아닌 Yaw,Roll,Pitch 제어가 필요합니다.  다만 이 제어를 위해서 4축 조이스틱이 2개 필요합니다.  <사진 1> 조이스틱 2개를 쓰고 있는 일반적인 드론 조종기 사진.  이러한 상황에서 특수 목적 드론의 경우 양 손의 엄지를 조이스틱에 둔 채로 미션 수행을 위해 지정된 버튼 등의 트리거 장치를 활성화해야 합니다. 그렇다 보니 운용자 입장에서는 해당 드론의 기능이 다양할수록 어려움을 느끼게 됩니다.  위 문제가 지속될 경우 운용자에게 아래 문제들을 초래할 수 있습니다.   1. 비행 시간이 누적될수록 사용자의 피로도가 더욱 누적될 수 있음. (어려울수록 집중해서) 2. 여러가지 상황을 보고 그 순간에 맞게 운용해야 하다 보니 정보 과부하가 올 수 있음. 3. 복잡한 조작이나 트리거 사용 중에 조작 실수가 일어날 수 있음.   <문제점 2 설명>  일반적인 드론의 자동화 기능 중에 가장 많이 쓰이고 중요한 것이 특정 위치에 착륙하는 기능이라 생각합니다. 해당 기능은 위급 상황(배터리 부족, 통신 두절 등)에서 사전에 지정된 위치로 복귀하여 사고를 방지할 수 있고 또한 착륙 시 지정된 위치에 착륙하기만 해도 여러가지 사고(충격 완화 재질로 외부가 구성된Station이 아닌 다른 위치에 착륙 시 드론에 손상이 일어남, 군사용 드론일 시 정찰 후 외부에 알려지지 않은 본대로 복귀하지 않을 경우 곤란한 상황이 일어남 등)를 미연에 방지한다고 생각합니다.  다만 일반적인 자동 랜딩 기능의 경우 GPS 좌표 데이터만을 사용하여GPS 값이 정교하지 않거나, 위성이 일정 개수 이상 잡히지 않을 경우 오차율이 심합니다.  이렇게 자동 랜딩 기능이 정교하지 않을 경우, 아래 문제들을 초래합니다.   1. 랜딩 스테이션을 벗어났기에 안정적인 랜딩이 불가능해지고, 랜딩 시 지면에 닿는 스키드, 바디 프레임 등이 데미지를 입어 내구도, 외부 손상으로 인해 드론의 균형이 깨져 이후 비행 시 안정적인 제어가 어려울 수 있음. 2. 완전한 복귀가 아닌, 미션 수행 중 경유지일 경우 목표 지점에 도착하지 않을 시 다음 운용에 어려움이 있음. 3. 단순 랜딩의 문제가 아닌, 해당 좌표로 제대로 갈 수 없음을 의미함.   <문제점 3 설명>  아래 상황에서 문제가 될 수 있습니다.   1. 배터리 교체, 등의 특수 상황에 매번 고정된 거점으로 이동하여야 함. 2. 거점이 고정되어 있으면 경유지가 많은 상황에서 효율적인 동선 설계가 불가능함. 3. 선박에서 운용 시, 선박에 고정되어 있더라도 배가 지속적으로 움직이게 되어   운용 거점이 의도치 않았지만 이동하는 상황이 있음.  결론적으로 위 문제점들을 해결하기 위해 아래 조건들을 충족해야 합니다.   1. 운용자는 자동화 기능이 필요하며, 전부 다 자동화되면 좋겠지만 특정 상황에 트리거를 활성화하는 것보다 드론의 움직임의 자동화를 우선 시 할 시 조작이 보다 용이함. 2. 자동화 기능의 성능이 향상되어야 하며, 그 중 자동 랜딩 기능의 위치가 정밀해야 함.   다만 기존 방식은 GPS 값 만을 쓰기에 보정을 위한 다른 데이터가 필요함.   1. 운용 거점이 이동 가능하여야 하며, 이동된 좌표를 사용자 또는 드론에게 주기적으로 알려주어야 함.   따라서 저는 좌표 데이터만으로 드론이 자동으로 해당 좌표로 이동하며 (1조건 충족)  이동 후, 특정 모듈을 통해 다른 데이터를 받아 랜딩 정밀도를 보정할 수 있으며 (2조건 충족)  이동하여도 착륙에 문제가 없도록 Landing Station이 드론에게 GPS 데이터를 지속적으로 주는(3조건 충족)  무인기 정밀 착륙 유도 장치의 개발이 필요하다 생각하였습니다. |

❑ **기대효과**

|  |
| --- |
| o 운송 드론의 경우 기존 방식보다 더 정확한 랜딩이 가능해져 좋습니다.  o 랜딩 지점의 정확도가 올라가게 되면, 랜딩 이후 후속 작업이 용이합니다. (예를 들자면 운송 드론 랜딩 이후 해당 물품을 자동으로 꺼내주는 장치의 설치 위치, 작업 연계성 등에서 강점이 있다고 생각합니다.)  o 의도되지 않은 착륙을 방지하여 착륙 중 발생할 수 있는 손상이나 사고가 방지됩니다.  o 이 외 GPS 데이터와 마커를 통해 보정하는 방식이 있으나, 그 방식보다는 비용측면에서 절감 가능.  o 도서 간 수송목적의 드론의 경우, 효율적인 waypoint 설정 및 비행이 가능합니다.  o 선박에서의 운용 시 기존 방식보다 효율적인 운용이 가능합니다.  o 이 외, 해당 자동화 기능을 사용하는 상황에서 이점을 볼 수 있습니다. |

❑ **개발 목차**

|  |
| --- |
| 1. 시스템 구성도, 설계, 동작 흐름 2. 각 시스템 파트별 제품 선정 및 임의 견적 3. Git Repository 생성 4. 자재 수급 가능 여부 및 납기일 파악 5. 자재 선정 및 구매    1. HW    2. Module 6. 모듈 단위 검증    1. FC 검증 – MissionPlanner 연동 및 파라미터 세팅    2. GPS 모듈 - NMEA 형식 데이터 수신 확인    3. ESC (모터 구동, PWM 제어, Calibration)    4. Drone Power Board 전력 공급 확인    5. Telemetry 동작 확인 (datasheet 기준 통신 거리 등 검증)    6. 배터리 에이징 테스트 (충, 방전 5회 테스트 진행) 7. 드론 개발    1. 하드웨어 납땜 및 체결       1. FC - MainBoard 체결       2. MainBoard – ESC – Motor 체결       3. MainBoard – Telemetry 체결       4. MainBoard – GPS 체결       5. 모듈 - 드론 프레임 체결       6. Battery 체결    2. F/W 개발       1. FC F/W 개발 또는 기존 오픈소스 활용       2. IMU 센서 Calbration 코드 적용       3. 배터리 잔량 계산 코드 적용 (전압분배 회로 적용)       4. telemetry통해 원격 데이터 송 수신 packet 구조 설계 및 개발       5. GCS로부터 좌표 데이터 수신 기능 확인       6. 좌표 데이터 수신 후 좌표 설정하여 이동하는 기능 개발       7. 랜딩 자동화 기능 개발   7.3  7.4   1. GCS 개발    1. 구조 설계       1. F/W 개발          1. Switch          2. Joystick          3. USB 통신    2. 하드웨어 납땜 및 체결       1. GCS 프레임 (= 기구물)          1. Ctrl Board 연결          2. Joystick 연결          3. Mission Button 연결          4. Telemetry 연결       2. Ctrl Board 배선   8.1.2.1   * + 1. ㄴ     2. ㅇ   8.2.1  8.2.2   1. Station 개발    1. 구조 설계 및 개발 2. USB 통신 |

❑ **개발 내용 요약**

|  |
| --- |
| 1. 자동 랜딩 기능을 탑재한 드론 개발 2. 드론 운용이 가능한 조종장치 개발 3. 랜딩 (착륙) 을 도와줄 Station 개발 4. 수행 능력 평가 후 실제 효과가 있는지 검증 및 보완 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | | |  | |  |
|  |  | | | | | |
|  | | | | | | |
|  | |  |  | |  | |
|  | |  |  | |  | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | | |  | |  |
|  |  | | | | | |
|  | | | | | | |
|  | |  |  | |  | |
|  | |  |  | |  | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | | |  | |  |
|  |  | | | | | |
| <https://github.com/kimkyowon/Capstone_Design.git> (Repository 경로입니다)  <Folder Tree 이미지, 아래 설명의 (1)은 최상위 폴더부터 하위 폴더로 갈수록 ()안의 숫자가 1씩 늘어남>    (1) Capston\_Design : 가장 상위 폴더입니다.  (2) PrecisionLandingModule : Capstone\_Design의 프로젝트로, Capston\_Design의 하위 폴더입니다.  (3) ResearchNote : 연구 노트와 작성 시 활용된 자료들을 보관하는 곳입니다.  (3) Version\_1.0 : 실제 제작할 제품 관련한 자료들이 들어가는 곳입니다. 초안이기에 Version 1.0으로 지칭하였습니다.  (4) SystemConfiguration : Version1.0제품의 전체 SystemFlowChart등 전체 구성에 관한 자료가 들어갑니다.  (4) Drone : Version 1.0의 Drone에 대한 자료가 들어가는 폴더입니다.  (4) GroundControlStation : Version 1.0의 GroundControlStation (GCS)에 대한 자료가 들어가는 폴더입니다.  (4) LadingStation : Version 1.0의 LadingStation에 대한 자료가 들어가는 폴더입니다.  (5) Src : 개발 시 들어가는 FW src 가 들어가는 공간입니다. (Drone, GroundControlStation 도 동일 구성)  (5) Document : 개발 시 참고해야 할 DataSheet, SpecSheet 등의 자료가 들어가는 공간입니다.  (5) Release(BinaryOrHex) : 개발 후, 최종 FW Binary 또는 Hex 파일이 들어가는 공간입니다. | | | | | | |
| 기록자 | | 김교원 | 점검자 | | 박정은 | |
| 서명 | | (sign) | 서명 | | (sign) | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 프로젝트명 | 무인기 정밀 착륙 유도 장치 개발 | | | 연구기간 | | 2024.09.09~09.18 |
| 제목 | 1. 자재 수급 가능 여부 및 납기일 파악 | | | | | |
| **목적**  선정한 자재들의 수급 가능 여부를 파악하여 의도치 않은 상황을 방지한다.  수급 여부 파악과 동시에 납기일도 알 수 있어 추후 일정 계획을 세울 수 있다.  해당 과정을 통해 얻을 수 있는 이점은 아래와 같습니다.  1. 수급 가능 여부를 파악해두면 추후 나중에 문제가 생겨 교체품 구매시에 수급이 안되는 등의 문제점을 방지할 수 있습니다.  2. 납기일을 파악하여 추후 일정 계획 수립에 도움이 됩니다.  3. 전문 전자부품업체와 의논할 시 대치품 파악에도 도움이 됩니다. (수급이 어려우면 대치품을 찾아주는 업체가 있습니다.)  4. 일반적으로 해외 제품이 많은데, 이의 경우 납기일이 굉장히 김 (통관 절차가 복잡해짐, 수급 업체와 해외 업체의 휴무일을 모두 고려해야 함 등) 따라서 국내 업체에 미리 말해두거나 보유하고 있는 국내 업체를 찾아야 함.  [컨택할 업체 목록]   * ㈜플랜비넷 * DeviceMart * Digikey * 루나볼트 * 가치창조기술   [참고 자료]  Capstone\_Design\1.PrecisionLandingModule\1.ResearchNote의 ResearchNoteData.xlsx파일의 “제품선정자료” sheet참조 | | | | | | |
| 기록자 | | 김교원 | 점검자 | | 박정은 | |
| 서명 | | (sign) | 서명 | | (sign) | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 프로젝트명 | 무인기 정밀 착륙 유도 장치 개발 | | | 연구기간 | | 2024.10.01~10.07 |
| 제목 | 1. 자재 선정 및 구매 | | | | | |
| **목적**  자재 최종 선정 및 실제 구매를 진행한다.  납기일이 다 다르기 때문에 우선순위 검증 모듈들 또는 다른 사출물을 주문하여 테스트 일정에 지장이 없도록 한다.  [서론]  먼저, 드론 제작 시 일정 무게가 넘어가게 되면 자격증이 필요합니다.  다만 2kg 이내로 제작 시에 온라인으로 취득 가능한 4종 자격증만 있으면 되므로, 2kg 이내 제품으로 선정하였습니다.  추가적으로 각 파츠별 자체적인 선정을 하려 했으나… 단가적인 측면에서 (낱개로 사는 게 더 비쌈) 어려움이 있었고 효율적인 측면에서 (Frame과 각 부품 체결 시 하네스 등이 필요한데 자체 선정 시 그 부분이 어려움)  결론적으로 2kg 이내이면서, 체결에 어려움이 없도록 Dev Kit 제품 (Set)으로 선정하였습니다.  [최종 선정 제품 – Drone dev kit]  <https://www.rcbank.co.kr/shop/goods/goods_view.php?goodsno=20749&category=064014002>  [구성품 - Image]    [구성품 목록 (spec)]  Frame - F450 (Fixhawk) [ size 450mm / weight 282g ]  Motor - S2312-920KV (14.8V -4 cell / 220W /920Kv )  ESC - 35A BL BLHeli\_S (cap 30A / 14.8V – 4cell / cont A 35A / Programed)  Propeller – DJI 9.4 \* 4.3 탄소 섬유 나일론 강화 프로펠러  Motor Cap - 2212 Motor Prop Adaptor (M6/CW/CCW/4pcs)  Battery - PT-B2200N-SP35 (14.8V, 4S1P, 35C+/JST-XT) 리튬폴리머 배터리 - XT60  Landing Skid - [Tarot] F450/F550 랜딩 스키드  GPS Mount - T-Type GPS Mount Set (High Grade)  Power connector - XT60 커넥터 전원잭 (10cm 실리콘 케이블 장착)  [선정 사유]  일반적인 Drone Frame kit는 Holybro에서 파는 10~20만원대에 구성되어 있습니다.  다만, 위 Frame Kit는 17만원에 Frame 뿐만 아닌, FC,GPS,Telemtry를 제외하고 나머지를 다 주고 있어 선정하였습니다.  또한 판매처에서 4종 온라인 교육만이 필요한 제품임을 보증해주기도 합니다.    [최종 선정 제품 – Telemetry]    <https://silvus.cafe24.com/product/detail.html?product_no=31&cate_no=26>  [구성품 - Image]    [구성품 목록 (spec)]  Harness -> Fc to Telemtry (JST 6pin connector)  Usb 5pin -> PC to Telemetry  Usb 5pin to Usb-A -> PC to Telemetry  Telemtry -> Holybro Sik V3 (100mw / 433mhz)  [선정 사유]  고출력 제품은 필요없지만, 적당한 중급 스펙의 RF 모듈이 필요하였는데  해당 제품이 저스펙도 아니고 고스펙도 아닌, 적당한 스펙 (100mw, Distance 300m) 를 보유중이어 선정하였습니다.  또한 제품 가격도 합리적이었습니다. (2개와 케이블이 총 8만원)  [참고 자료]  Capstone\_Design\1.PrecisionLandingModule\1.ReasearchNote\ResearchNoteData.xlsx 의 Sheet “제품선정자료” | | | | | | |
| 기록자 | | 김교원 | 점검자 | | 박정은 | |
| 서명 | | (sign) | 서명 | | (sign) | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 프로젝트명 | 무인기 정밀 착륙 유도 장치 개발 | | | 연구기간 | | 2024.09.09~09.18 |
| 제목 | 6-1. 모듈 단위 검증 – FC 검증 | | | | | |
| **목적**  드론의 핵심 부품인 FC를 중점으로 여러가지 모듈이 장착됩니다.  따라서 다른 모듈을 검증하기 전, FC의 검증이 선수되어야 하며  기본 동작이 검증된 시점에 initializing을 위한 파라미터 세팅을 진행합니다.   1. Mission Planner 연동  * 해당 PC앱을 연동함으로서, FC의 각종 모듈 값을 모니터링 할 수 있습니다.   (통신 속도, FC 내부의imu 등 센서 값 등)  V1.3.82 앱 다운로드 경로 (<https://ardupilot.org/planner/docs/mission-planner-installation.html>)   * 통신 방식은 RF 모듈을 활용해야 하지만, 아직 오지 않은 관계로 유선(Serial)방식을 사용해 검증하였습니다.     (PC와 FC를 유선 연걸 후, 좌측 빨간 박스에서 port 선택 후, 우측 연결 버튼을 누르면 연결됩니다)   * 주요 항목 간략한 소개      1. [데이터]: 지금 보이는 화면을 통해 드론의 데이터를 수집할 수 있습니다. 2. [계획]: 경로 지정을 통해 항로 설정이 가능합니다. 3. [설정]: 드론에 부착된 모듈등의 하드웨어를 보정하거나 수정할 수 있습니다. 4. [구성]: 이곳에서 FC의 설정값을 수정할 수 있습니다. (Initializing을 위한 Parameter 세팅 가능) 5. [모의 시험]: 실제 드론 없이 모의 비행이 가능합니다. 6. [비행화면]: FC에 부착된 IMU센서를 통해 기체 각도를 표현한 이미지, 실시간으로 변화합니다. 7. [개요]: 기본적인 비행 데이터를 제공해줍니다 (고도, 위도Yaw,등등)   (3) [명령]: 제공되는 API에 한해, 각종 모듈에 Command를 전송할 수 있습니다.  (3) [메시지]: PC앱과 FC간의 데이터 송수신을 모니터링 할 수 있습니다.  (3) [사전 점검]: 비행 전 상태를 알 수 있도록 사용자 지정 값 이상일 경우 비행이 가능합니다.  추후 모듈들이 도착하면 해당 기능을 통해 검증 예정입니다.     1. FC 데이터 모니터링     FC 자체에 IMU센서가 있는데, 해당 센서는 초기 부팅 시 FC의 수평 각에 맞춰 0으로 됩니다.  따라서 초기 부팅 시, 책상위에 수평을 유지한 채로 Mission planner에 연동 시 오차 없이 수평으로 나오면 정상입니다.  이후, 직접 FC를 기울여보며 테스트해보시면 됩니다.  테스트 영상은 아래 경로에 업로드 하였습니다. “Capstone\_Design\1.PrecisionLandingModule\2.Version\_1.0\1.Drone\2.Document\2.DataSheet\_SpecSheet\1.FC”   1. PWM 테스트   PWM는 Pulse width modulation의 약자로, 쉽게 말해서 신호의 길이를 조절하는 것입니다.  해당 OrangeCube FC는 해당 PWM출력 포트를 통해, 사용자가 mission trigger를 쓸 수 있도록 구성되어 있지만  사실 PWM 신호 자체는 정확하지 않기 때문에 (신호 길이가 ms단위로 딱 맞기도 어려울 뿐더러 일반적인 신호는 on,off 처럼 이분법적으로 나뉘어야 하지만 적게는 1ms ~ 2000ms처럼 나올 수 있는 신호 범위가 크기 때문)  일반적으로 선호하는 방식은 아니긴 합니다.  여튼, 이를 검증하기 위해 이전에 작성하여둔 MyPwmFilter 코드를 활용해 FC가 출력하는 PWM값이 정확한지 측정하였습니다.  (아래는 직접 작성한Pwm Filter library)    Capstone\_Design\1.PrecisionLandingModule\2.Version\_1.0\1.Drone\2.Document\2.DataSheet\_SpecSheet\1.FC 경로에서 “PwmTestCode\_Arduino.zip” 을 확인해보시면 됩니다.  (아래는 테스트 영상의 일부 사진)    테스트 코드 설명: “**우측 사진”**을 보면 “저, 중, 고”, “전환”, “숫자”, “pwm최소값”, “pwm최대값” 형태인데.  각 파트를 설명하자면 아래와 같습니다.   1. “저, 중, 고”는 Pwm의 최소, 중간, 최대 값 출력버튼을 의미합니다. 2. “전환”은 자동으로 “저, 중, 고”를 변환해줍니다. 3. “숫자”는 FC의 포트 번호입니다. 여기서 9는 “AUX OUT”포트의 1번입니다. 4. “pwm 최소값”은 말 그대로 pwm의 최소값입니다. 사진처럼 1000이 최소값이며 “저”를 누르게 되면 1000이 출력됩니다. 5. “pwm 최대값”은 말 그대로 pwm의 최대값입니다. 사진처럼 2000이 최대값이며 “고”를 누르게 되면 2000이 출력됩니다.   동작에 대한 설명은 아래와 같습니다.   1. 최상단 사진인 VS code를 보시면, 핀 할당 표가 있습니다. 말 그대로 arduino의 pin 할당 표입니다. 2. pin\_switch가 1000이면 pin\_a로 들어오는 pwm수치를 파악해 변화하면 pin\_a의 pwm의 값을 출력해줍니다. 3. pin\_switch가 1500이면 pin\_b로 들어오는 pwm수치를 파악해 변화하면 pin\_b의 pwm의 값을 출력해줍니다. 4. pin\_switch가 2000이면 pin\_fc로 들어오는 pwm수치를 파악해 변화하면 pin\_fc의 pwm의 값을 출력해줍니다. 5. Pwm 수치가 기존과 다를 시, 좌 하단 사진의 Arduino Serial Monitor에 출력되도록 하였으며, 데이터 형태는 아래와 같습니다.     [시간] -> [핀번호] [이전값] ===> [현재값] [날값]  **시간은** 말 그대로 serial print가 출력된 시간입니다.  **핀 번호는** 변화된 pwm값이 들어온 번호입니다. (예시 : sw값이 변하면 sw값인 0이)  **이전 값은** 말 그대로 이전 pwm값입니다.  **현재 값은** 말 그대로 현재 변화되어 들어온 pwm값입니다.  **날 값은** 앞서 설명드린 것 처럼, pwm값은 딱 맞게 들어오지 않기 때문에, 보정되지 않은 값을 의미합니다.  (현재 코드에서 보정 범위는 100ms입니다)  테스트 영상은 아래 경로의 “TestVideo.zip”폴더에 존재합니다.  Capstone\_Design\1.PrecisionLandingModule\2.Version\_1.0\1.Drone\2.Document\2.DataSheet\_SpecSheet\1.FC  테스트 과정이 모두 기재되진 않았지만, 위 테스트를 통해 얻은 결론은 아래와 같습니다.   1. 현재 보유중인 FC는 AUX OUT의 1,2,3,4만 동작함. 2. PWM Filtering을 위한 오차범위는 100이면 충분함. 3. 장시간의 테스트에도 FC의 PWM출력은 정상적으로 동작함.   [참고 자료]  Capstone\_Design\1.PrecisionLandingModule\2.Version\_1.0\1.Drone\2.Document\2.DataSheet\_SpecSheet\1.FC 경로의  “TestVideo.zip” 과 “PwmTestCode\_Arduino.zip” | | | | | | |
| 기록자 | | 김교원 | 점검자 | | 박정은 | |
| 서명 | | (sign) | 서명 | | (sign) | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 프로젝트명 | 무인기 정밀 착륙 유도 장치 개발 | | | 연구기간 | | 2024.10.14~11.01 |
| 제목 | 7. 드론 개발 | | | | | |
| **목적**  전체 시스템 중, 파트별로 구성하여 단계별 개발을 진행하기 위해 구분하였음.  또한 드론의 규모가 어느정도 있는 상황이라 세세하게 단계별 점검이 필요하며 이를 사전 구성 시 추후 점검이 용이함.   1. 하드웨어 납땜 및 연결   이전에 구매하였던 자재들을 모아 제작 진행하였음.  (조립 가이드라인은 다음 링크를 참조하였음 : <https://www.youtube.com/watch?v=IdpUYPuSYhE> )  => MainBoard - ESC 납땜  (좌측은 MainBoard, 우측은 ESC / ESC의 납땜 포인트는 2곳이고 각 VCC와 GND임)    => ESC - Motor 체결 및 납땜  (ESC - Motor의 경우, 납땜 포인트가 3곳인데 방향성이 있어 진행의 원활을 위해 커넥터 형식으로 구매하였습니다. 따라서 납땜을 생략 후 체결만 진행하였습니다.)  => MainFrame 체결 및 납땜  (왼쪽부터 조립 전, 조립 후 입니다. 내부 회로가 노출되어 있기에 추후 절연 작업이 필요합니다.)    => Motor - Propellar 체결  기존 패키지에 있던 프로펠러는 체결방식이 맞지 않았고, 테스트용이므로  조금 더 강성이 좋은 상품으로 재선정하였습니다.    <https://www.rcbank.co.kr/m2/goods/view.php?goodsno=68702>  => FC - Battery 연결  FC와 Battery를 연결하기 전, 배터리의 상태를 봐줄 셀 체커기가 필요했습니다.  (위 제품은 2S ~ 8S 배터리 체커기이고 위 제품을 구매하여 잔량 및 상태체크를 할 수 있었습니다)  이후 Battery를 FC에 직결할 수 없으므로, PowerModule (인터페이스 제품)을 구매해야 했습니다.  결론적으로 실제 연결은 아래 4 파트로 나뉘어집니다.    좌측부터 FC - Battery - PowerModule - Drone 입니다.  => 기본 동작 검증  - 역전압 현상 확인 : 배터리 직결 시, 역전압 현상으로 보이는 외부 증상은 없었습니다.  - FC 연결 확인 : 배터리 연결 시, FC에 정상적으로 파워가 공급되며 접근 가능했습니다.  - ESC 비프음 확인 : VCC가 안정적으로 공급되어 초기 부팅 시 나오는 비프음을 확인했습니다.  <= 안정적으로 모두 연결된 모습  [참고 자료]  Capstone\_Design\1.PrecisionLandingModule\1.ResearchNote의 ResearchNoteData.xlsx파일의 “제품선정자료” sheet참조 | | | | | | |
| 기록자 | | 김교원 | 점검자 | | 박정은 | |
| 서명 | | (sign) | 서명 | | (sign) | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 프로젝트명 | 무인기 정밀 착륙 유도 장치 개발 | | | 연구기간 | | 2024.11.25~11.29 |
| 제목 | 8. GCS 개발 | | | | | |
| **목적**  드론을 제어할 지상통제기를 개발한다.  실제 HW 완제품 개발 전 기능 요구사항, 시스템 설계, 동작 검증 등을 진행하여  실물 제품을 만드는 과정에서 오류가 발생하지 않도록 방지할 수 있다.   1. 구조 설계  * GCS 와 Drone 의 일반적인 통신 구조     기본적으로 Command를 주고 받으며 해당 Command 구조는 Mavlink Protocol로 수행됩니다  또한, 원격 통신이기에 중간 매개체로 Datalink를 사용하게 되므로 GCS는 원하는 Command를 Datalink측에 MavlinkProtocol로 패킷화하여 전송 후 Drone이 수신하게 되고. 이후 Drone은 다시 잘 받았다는 Ack메시지를 GCS에게 전달하게 됩니다.   * 현재 프로젝트 상 통신 구조     다만, 현재의 경우 일반적인 통신 구조와 달리 중간에 MissionPlanner가 존재하여 사용자는 Joystick만 구현을 하고  Raw Data 만을 전송해주면 됩니다 (나머지 Packet화 등은 MissionPlanner에서 동작).  따라서 저의 경우 Joystick을 구현 후 MissionPlanner에 체결 시GCS에 대한 개발이 완료될 것으로 보입니다.   * 조종기 기능 설계   조종기 기능 설계 이전, 조종 방식에 대해 먼저 보여드리자면 아래와 같습니다.    총 조종모드는 2가지가 있는데, 저의 경우 위 그림처럼 조종모드 2를 사용할 예정입니다.  따라서 기본적인 Drone의 조종을 위해 Joystick은 총 2개가 필요하고  Arming/Disarming 을 위한 PushLock 버튼 1개,  자동 Landing 기능을 위한 PushLock 버튼 1개가 필요합니다.     * **조종기 F/W 개발 (1차)**   조종기의 경우 단순히 Joystick과 button의 값만 주기적 샘플링을 하는게 아니라 HID 장비로서 동작해야 하는데요, 해당 구조를 Arduino 버전으로 만든 Reference가 존재합니다.  <https://github.com/MHeironimus/ArduinoJoystickLibrary> (Reference 주소)  따라서 해당 Reference 기준으로 아래 라이브러리를 활용하였고    해당 예제 중 FlightControllerTest.ino 메인 파일을 분석하였고  Arduino main 동작은 크게 2가지인 Setup (1회성) 과 Loop (무한반복)이 있기에  해당 파트를 중점적으로 분석하였습니다.  < Setup >    필수적인 내용만 넣어둔 듯 한데 분석하자면  본인이 쓸 Joystick의 min max 값을 매핑하고, HID 장치로서 연결된 device에 전송하기 위해서 Joystick.begin()을 실행해야 합니다, 마지막으로 핀 설정을 해주면 필수적인 부분은 마루리 되는 것 같습니다.  저의 경우에도 동일하게 구성하였습니다.  < Loop >  현재 Reference 기준으로는 TestCode여서 그런지, 실제 조이스틱 값을 주기적으로 샘플링하여  Joystick 객체에서 값 설정 후 업데이트의 반복은 하지 않고 있습니다.  따라서 자체적으로 루프를 구현하였고  1. 데이터 Sampling / 2. 이전과 값이 다를경우 값 세팅 / 3. 데이터 자동 전송 기능이 비활성화 시 수동 전송  을 반복하도록 작성하였습니다.  (완성된 루프구조)  추후 검증이 필요하나 8.2 의 과정을 통해 실물 GCS 제작 후 가능한 것으로 보입니다.  **- 조종기 F/W 개발 (2차)**  이전 제작한 F/W 구조에서는 샘플용 코드여서 수정 및 운용에서 어려운 부분이 많았습니다.  따라서 저는 기존의 샘플 코드를 참고하여 자체적으로 새로운 F/W 구조를 개발하기로 하였습니다.  먼저 폴더 구조는 아래와 같습니다.    해당 폴더 구조는 PSL\_GCS라는 아두이노 코드 (확장자 명 : .ino)파일을 동작시키기 위한 라이브러리 폴더 (src)를 가진 형태이며. 라이브러리 폴더 안에는 자체 개발한 라이브러리 (PSL\_GCS, Button)과 기존 라이브러리 (DynamicHID, Joystick) 을 포함하고 있습니다.  main코드와 라이브러리 항목에 대해 설명드리기 위해 **main 코드 먼저 설명**드리자면 아래와 같습니다.    PSL\_GCS의 Class 생성 후 버튼 2개에 대한 인터럽트 서비스 루틴 설정을 추가하였습니다.  위 인터럽트 서비스 루틴을 통해 버튼에 들어오는 Noise를 디바운싱 할 수 있습니다.          => HW 날 값 추출 (getJoystickValues, getButtonValues)  RaspberryPi로부터 오는 Signal값은 모델 학습 최종 종료 후 진행 예정  => 미션 트리거가 되는 “특정 시간 이상 버튼 누르기” 과정에서 디바운싱 처리를 합니다 (debounceButton)  => 객체 감지 후, 자동화 조종을 위한 HW 값 차단 모드인지 차단 해제 모드인지에 따라 다르게 동작합니다.  만약 자동화 조종 모드 일 시, RaspberryPi로부터 받은 Signal을 토대로 정해진 축 값을 통해 이동합니다.    자동화 조종 모드가 아닐 시, 앞서 추출한 HW 값을 바탕으로 데이터를 Window상으로 전송합니다.  이 동작은 HW 값에 변동이 있을 시에만 동작합니다. < if(CapstoneGCS.getDiffState()) >  => HW 값에 변동이 없다면 Arduino 동작이 원활하게 되는지 외적인 변동이 없기에  HeartBeat를 추가하여 print 되도록 하였습니다.  => delay(1)은 위 동작을 너무 빨리 반복할 필요는 없기에 추가하였습니다.  **라이브러리 하위 항목 별로 설명** 드리자면 아래와 같습니다.  - DynamicHID : HID 통신을 위한 기본적인 코드가 들어가 있습니다. Joystick Library가 이를 활용해 윈도우 상에서 HID 장치로 인식할 수 있게끔 합니다.    위 Class 구조에서 Joystick은  customHidReportDescriptor을 만들어 sendReport를 통해 데이터를 전송한다고 보시면 됩니다.  - Joystick : Joystick의 기본적인 코드가 들어가 있습니다. 주로 실제 조종기에서 모두 조이스틱을 쓰지만 어떤 것은 Rudder, Throttle, Rx,Ry 등등 명칭이 다 다르기에 HW에서 추출한 값에 명칭을 붙여 전달하기 위한 용도로 쓰입니다.    class 자체가 좀 커서 모두 담긴 어려워 핵심 부분만 담았습니다.  위 함수들을 사용하여 X축 값, Y축 값, Rudder축 값, 버튼 값 등의 데이터를 특정 배열에 셋업 후 sendState를 통해 window에 전송하게 됩니다.  - PSL\_GCS : 이번 PrecisionLanding 을 위한 Class를 제작한 라이브러리입니다.  총 구성해야 하는 기능은 아래와 같습니다  1. HW 값 추출 후 저장( Joystick, Button, Signal - from RaspberryPi )  2. HW 데이터 가공 (Invert)  3. 특수 기능 추가 ( ModeChange , autoMode )  4. 디버깅 기능 추가 ( printCurrentValues - 상태값 출력 )    - Button : HW 장치 중 Button에 대한 Class를 작성하였습니다.  디바운싱, 상태 값 출력, 미션을 위한 버튼 플래그 생성 등의 기능이 있습니다.    아래는 부가 기능들에 대한 자료입니다.  <HW 구성에 따른 값 Mapping 기능>  - HW 장비들의 방향에 따라 값이 달라지므로 이를 고려해 값을 매핑해줄 수 있는 기능을 제작하였습니다.  (동일 제품 조이스틱이 HW 구성에서 0도 돌아가서 들어간 것과 180도 돌아서 삽입된 것은 상하좌우 값을 반전시킴)      <PSL\_GCS 생성자 하위 파라미터 자동화>    생성자에 joystick과 Button 생성자를 추가하여 자동으로 하위 항목도 생성되도록 설정하였습니다.  <미션 트리거가 되는 버튼 동작>  먼저 Interrupt 함수로 버튼의 상태값이 반전 될 시, 상태값을 받아옵니다.    이후 main loop에서 동작하는 디바운스 함수로 실제 사람이 버튼을 길게 누른 것인지 노이즈인지 판별합니다.    판별이 되었다면 미션 수행을 해도 된다는 Flag인 State\_isButtonPushed를 활성화 해줍니다.  마지막으로 미션 수행 후 해당 State\_isButtonPushed를 비활성화 해줍니다.    Mode Change - TransData 모드 일 시, MissionPlanner 앱에서 정상적으로 인식하는 모습    Mode Change - DontTransData모드 일 시, MissionPlanner 앱에서 받지 않는 모습    **- 조종기 F/W 개발 (3차)**  **2차 개발에 이어 추가 기능들을 개발하였습니다.**  [ 개발 항목 ]  (1). 드론에 보낼 전, 후, 좌, 우, 정지 동작 구현  (2). LandingStation 으로부터 Byte 단위의 Command 명령을 I2C로 받아 저장.  (3). 저장된 Command에 따른 동작 수행.  **Main Loop 동작**    여기서 command 값은 RaspberryPi에서I2C라인을 통해 받고, 받는 시점에 receiveEvent를 통해 전역변수인 command에 저장됩니다.      **getSignalValues() 동작**    말 그대로 전역변수 command값을 class 변수안에 저장합니다.  **processSignal() 동작**    수집한 command 값이 변경되면 신규 command로 판단하여 MissionPlanner app 으로 명령을 전송합니다.  **안정화 작업**  생길 수 있는 예외 케이스 (드론 인식이 중간에 끊김 등 )을 고려하여 드론을 멈출 동작 또한 구현하여  Interrupt 신호를 받을 시, 드론이 정지할 수 있는 기능도 추가하였습니다.    [참고 자료]  Capstone\_Design\1.PrecisionLandingModule\2.Version\_1.0\2.GroudControlStation\1.Src\joystick의 joystick.ino참조  Capstone\_Design\1.PrecisionLandingModule\2.Version\_1.0\2.GroudControlStation\1.Src\PSL\_GCS 폴더 참조 | | | | | | |
| 기록자 | | 김교원 | 점검자 | | 박정은 | |
| 서명 | | (sign) | 서명 | | (sign) | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 프로젝트명 | 무인기 정밀 착륙 유도 장치 개발 | | | 연구기간 | | 2025.03.10~03.15 |
| 제목 | 9.1 구조 설계 및 개발 | | | | | |
| **목적**  기존 설계한 Landing System의 문제점을 수정하고 대안을 마련하기 위한 작업.  구조 설계를 통해 시행착오 감소, 효율적 설계 등의 이점을 얻을 수 있음.   1. 기존 구조의 문제점   개발 설계 당시에는 몰랐으나 OrangeCube 시스템 구조와 소모 전류량 그리고 단가 때문에 3가지 문제점들이 있었습니다.  **1.** 세밀한 FC 제어 구조를 가지고 있지 않다면. (FC 동작 코드를 모두 꿰뚫고 있지 않는 이상, FC에 장착된 모듈이 Mavlink Packet을 통한 부드러운 제어가 어려움)  **2.** IR Beacon의 높은 전력 소모량  **3.** IR Beacon 및 IR Sensor 와 FC가 호환되는 제품이 한정적이며 단가가 매우 높음 (약 180만원 )  따라서 이를 해결할 대안책이 필요했습니다.   1. 구조 개선   각 문제점을 개선하기 위해 약 몇 일 고려해본 결과 아래와 같은 결과를 도출해 낼 수 있었습니다.  **1.** Joystick 의 경우 내부적으로 제공되는 SW를 거쳐 매끄러운 동작 제어가 가능하므로 Joystick의 동작과 똑같이 제어 ( = Joysitck이 보내는 Packet과 동일하게 보내기 )  **2.** 전력을 공급받을 수 있도록 GCS와 유선 연결 및 전력 수급  **3.** Beacon형태가 아닌, 다른 방식의 유도 장치로 변경.  ( Raspberry Pi를 통해 YoloV8 딥러닝 모델을 사용하여 드론 인식 후 카메라 화면 중앙에 드론이 배치될 수 있도록 제어 )     1. 딥러닝 모델 개발   - 사용할 보드 스펙 : Raspberry Pi 3 model B+  - 딥러닝 모델 : YoloV3  3.1 데이터 셋 탐색  신규 구조의 경우, Pi Camera 단에서 드론을 인식하는 과정이 필요하기에  드론 이미지와 해당 이미지에 대한 바운더리 박스, 좌표, 클래스 정보가 담긴 파일이 필요합니다.      <https://www.kaggle.com/datasets/sshikamaru/drone-yolo-detection>  찾아보니 4000장 이상의 드론 이미지, 이미지에 1대1로 대응하는 객체 정보 파일, cfg 파일을 제공해주는 곳이 존재했습니다.  3.2 모델 학습  해당 데이터 셋 이용 시, yoloV2-tiny를 주 타겟으로 잡았지만 저의 경우 라즈베리파이로의 이식을 위해 TFLite파일 변환 과정이 필요하기에 YoloV3으로 진행하기로 하였습니다. (데이터셋 설명 시 YoloV3도 사용된다고 명시되어 있었음)  단, 아직 Raspberry Pi Cam 선정이 되지 않아 ( 가속기 없이 어느정도 퍼포먼스가 나오는지 잘 모르는 상태)  Colab 상에서 해당 데이터셋으로 모델 학습과 실제 훈련,학습 데이터 이외의 이미지를 구분할 수 있는 상태인지 확인하기 위한 정도의 테스트를 진행했습니다.  여러가지 시행착오가 있었으나 (Make file 수정, Cuda 버전 호환성 문제, cfg 파일 수정 등)  결론적으로는 해당 데이터셋을 통해 학습이 가능했고 다른 이미지를 테스트하는 코드까지 정상적으로 돌아가는 것을 알 수 있었습니다.    ( 위 사진은 외부 이미지를 테스트했을 시 출력되는 아웃풋 )  다만 라벨링이 제대로 되었는지에 대한 평가를 위해 이미지를 출력하려했으나 Colab상에서 강제된 드라이버 버전과 Cuda 버전과의 차이로 인해 OpenCV 단에서 문제가 일어나 확인이 어려웠습니다.  해당 검증 부분은 빠른 시일 내 Pi cam 선정 후 실제 화면을 통해 테스트해보겠습니다.   1. 2차 검증   4.1 테스트 환경 구축  - 3.2 모델 학습 과정에서 YoloV3을 통해 예측해보았으나 **호환성 문제**로 결과 이미지가 생성되지 않고, GPU가 있음에도 정적인 이미지에서 **객체 하나 인식하는데 60ms 가 소모**되는 것으로 보아 실시간 처리 용도로는 어려움이 있을 것으로 판단되었습니다.  따라서 기존 라즈베리파이 모델 단일로 하지 않고, 신규 라즈베리파이 모델 중에서 개발보드+가속기+카메라 등이 하나로 판매될정도로 호환성이 높은 제품을 탐색하였고 아래 제품으로 선정하였습니다.    해당 제품으로 테스트하기 위해 OS 설치, SSH 환경 구축, 모듈 검증 과정을 진행하였습니다.  OS 설치 - Raspberry Pi Imager 사용. 저의 경우에 Mini hdmi cable이 없어서 OS 설치와 SSH, VNC, 언어 세팅 등을  동시에 진행하였습니다.  SSH 환경 구축 - RealVNCViewer 사용 (아래 이미지는 직접 동작시킨 모습)    Camera 모듈 검증 - 관련 라이브러리 설치 후 동작 확인해보았습니다.     1. 학습 모델 변경 및 구조 개선   5.1 YoloV8n  - 이전 테스트 버전에서는 데이터 셋과의 호환성을 위해 원본 저작자가 시행한 YoloV3 모델 또는 V2 tiny를 그대로 기용하려 했으나, 기존 모델 성능이 낮았건 점과 개발보드를 활용한 레퍼런스들이 모두 YoloV8을 사용하여  YoloV8 모델 중에서 경량화된 n 모델을 사용하기로 하였습니다.  그에 따라, 기존 Colab에서 테스트하던 코드 구조와 소스들을 변경해야 했습니다.  먼저 코드 구조를 이해하기 위해 레퍼런스들을 찾아보던 중, 제 상황과 거의 동일한 레퍼런스를 찾았습니다.  <https://colab.research.google.com/drive/1qrTS6c5M8TgtbVh7pWITo-2LnrQuFXAh?usp=sharing>  해당 코드 구조는 복잡한 것 없이  사용할 그래픽카드 정보를 보고 필요 라이브러리 설치 후 데이터 셋 구조만 맞춰서 바로 훈련시킨 뒤 ONNX 파일로 변환하는 것으로 보였습니다.  따라서 저 또한 기존 데이터셋을 YoloV8n 환경에 맞추기 위해 폴더 구조를 변경하였으며  훈련용 데이터와 평가용 데이터를 구분하여 코드 작성하였습니다.  정리하자면 기존 YoloV3 를 학습시켜봤던 테스트코드에서 YoloV8n 모델을 학습시키기 위한 코드로 변경하기 위해  - 데이터 셋 파일 구조 재생성  - 이미지, 라벨 파일 구분 작업  - 이미지, 라벨 1대1 매칭 여부에 대한 전수 검사  - .cfg 파일이 아닌, .yaml 파일 생성  - 이에 따라 현재 사용되지 않는 .ipynb , .cfg , Makefile파일들 제거  등을 진행하였습니다.  5.2 학습 시작 및 라즈베리파이 운용 세팅  학습 시작 시 많은 어려움이 있었는데 그 중 대표적인 게 아래 두 가지였습니다.  - Colab에서 사용자 인터페이스 단의 상호작용이 일정 시간 없을 경우 자원을 회수해버리는 점  - epochs 를 50으로 설정했더니 학습 시간이 너무 오래 걸리는 점  다행히 실행중인 코드에 영향을 주지 않으면서 자원 회수를 막을 방법으로  console 에서 리소스를 일정 주기로 확인하는 방법이 있었고  <https://sjkoding.tistory.com/79>  epochs 는 모델 성능을 재차 확인하면서 추가 학습을 늘려나가도 되지 않을까 싶어 초기 10으로 조정하였습니다.  또한 학습 중간중간 산출되는 데이터를 기반으로 시각화 작업 또한 진행했습니다.  [ 학습 모델 샘플 - Colab ]  - 이미지 : 4010 장  - 훈련 모델 파라미터    - 샘플 결과 ( 8번 학습 시. )    확실히 학습량이 누적될 수록 인식률이 좋아지는 것을 확인할 수 있었습니다.  다만 극적인 그래프 변화율이 없어서 모델이 완전히 학습 되었을 시 검증을 해보아야 합니다.  [ 샘플 테스트 - 라즈베리파이 ]  라즈베리파이 상에서 운용하기 위해 몇 가지 셋업이 필요합니다.  - HW : 27W 충전기, 카메라, hailo, 네트워크 연결  - SW : 가상환경, 관련 라이브러리 설치, Thonny 인터프리터 변경, 샘플 코드 작성  HW 의 경우 이전 카메라 검증 단계에서 모두 세팅을 완료하였으므로 바로 SW 세팅을 진행하였습니다.  1. 가상환경 설치 및 실행 ( Pi OS와 별도로 진행하는 것, 손상 위험 없이 진행가능 )    => 가상환경 관련 패키지 설치 및 경로 설정    => 가상환경 활성화    => 가상환경은 기존 환경과 별도의 개념이므로 라이브러리 최신화 진행    => 컴퓨터 비전 작업을 지원해주는 패키지 설치    => Picamera2 모듈 설치    => 재부팅 진행    => 샘플 코드를 실행시킬 “Thonny” 실행    => “Configuration Interpreter...” 실행 후, 가상환경에서의 인터프리터 경로로 재설정    => 샘플 코드 작성 ( picam2 라이브러리 활용하여 실행 )  만약 라이브러리 설치가 잘 되지 않은 경우, 미처 설치하지 못한 라이브러리가 있을 수 있으므로  가상환경과 Pi os 상의 라이브러리 동기화 작업을 진행해주면 될 것 같습니다.  [ 실행 화면 ]    => 정상적으로 실행되는 모습, 단 샘플 모델이어서 인식율이 저조함  [ 학습 테스트 진행 ]  본 학습 테스트는 단순히 epochs를 많이 두었을 시 과적합 현상이 발현할 수 있으므로  epochs 수를 점점 늘려가며 이에 비례한 인식율을 확인 후 어느정도의 학습량이 적절한지 파악하기 위해서 진행  테스트 과정은 아래와 같습니다.  1. epochs 양 조절 후 학습  2. 산출된 학습 데이터 시각화 자료 생성  3. 라즈베리파이로 파일 전송 후, 샘플 코드 기반 테스트  (setup) 이전 샘플 테스트로 진행하였던 코드 기반으로, 반복적으로 실행하여 학습시킬 수 있는 코드 작성    (setup) Colab 환경에서의 학습 시, 산출되는 데이터 수집 후 시각화 자료 작업    저는 위 데이터 중 ( box\_loss , cls\_loss , dfl\_loss , Box(P) , mAP50-95 ) 들을 선정하여 시각화 하였습니다.  (setup) 이 데이터들을  “Capstone\_Design\1.PrecisionLandingModule\1.ReasearchNote\ResearchNoteData.xlsx” 의  “Test YoloV8n in RaspberryPi” sheet에 정리하였습니다.  < 1차 학습 - epochs 10 / 모델 학습 소요시간 : 10분 >  => 학습 정확도가 급격히 변화하는 모습 ( 횟수 자체가 적어서 그런듯 합니다 )    => 실제 인식 사진  < 2차 학습 - epochs 30 / 모델 학습 소요시간 : 40분 >  => 횟수가 늘어갈 수록 인식율이 점점 좋아지긴 하나 누적 될 수록 변화량이 작은 것이 보임    => 바운더리 박스와 전체적인 인식율이 많이 올라간 모습  < 3차 학습 - epochs 100 / 모델 학습 소요시간 : 4시간 >      => 10 -> 30 에 비해 30 -> 100은 큰 변화량이 없어 보임.  < 4차 학습 - epochs 185 / 모델 학습 소요시간 : 10시간 >  변경안  - 최대 epoch 600 까지 증가. (단 정확도 차이 없을 시, 학습 조기 중단 기능이 있어 185까지 학습함)  - autobatch 활성화로 그래픽인 T4 메모리 최적화  - 학습률 조정 (0,01)  - 가중치 감쇠 적용 (0.0001 ~ 0.01 범위 중, 0.0005로 설정)  결과적으론 과적합 현상이 있었는지 epoch 100 의 모델보다 인식율이 현저히 떨어졌습니다.  ( 화면상에 10번 중 3번 정도만이 인식됨)   1. 정밀 착륙 유도 시스템 SW 개발   6.1 System Flow    System Flow 를 풀어쓰자면 다음과 같기에, 모델 학습을 병행하며 위 순서대로 개발하게 될 것 같습니다.  6.2 Get center coordinate of BBox  - 위 기능을 위해 실제 BBox의 좌표값이 필요합니다. 다행히 추가적인 구현없이 객체 인식 결과물에서  산출되는 BBox 의 x,y 최소 최대 좌표값들이 있어 해당 class 함수를 통해 값을 반환받을 수 있었습니다.        => 테스트 코드 실행 결과    6.3 Send data of control drone to GCS  - 위 기능을 구현하기 위해, BBox에서 산출된 중심 좌표가 카메라 화면의 중심부에 올 수 있도록  신호를 줄 수 있도록 구현할 예정이며. 신호는 특정 digital pin에 output을 주거나, Packet 형태로 Serial라인을 거쳐 전달할 예정입니다.  다만 현재 GCS HW를 완벽히 구성하지 못해 임시로 화면 좌측 상단에 어디로 움직여야 하는지에 대한 정보를 출력하도록 설정하였습니다.        => 테스트 코드 실행 결과 (좌측 상단에 상하, 좌우 기준 어디로 가야할지 적혀있는 상태)  6.4 GCS pack drone control data with HID data packets and sends them to the MissionPlanner app  => 30쪽의 2차 FW 개발 중, “PSL\_GCS.h”의 sendGcsData() 참조.  6.5 MissionPlanner sends them to Drone with telemetry (wireless)  => 테스트 예정  6.6 Raspberry Pi camera tracks a drone in real-time and aligns its center coordinates. When alignment, it sends a vertical landing command through a 3 ~ 5 steps.  - 현재 수직 착륙 신호는 구성하지 않았으나 중심 좌표 도달 시, 프로그램이 종료될 수 있도록 구성하였습니다.      => 테스트 코드 실행 결과 (좌측 상단의 좌표 값은 더이상 무의미하므로 센터에 위치시켰다는 메시지만 출력)  테스트 코드에서는 명령을 단순 print 시켰으나, 이제 실제로 GCS 상에 이동 명령을 보내기 위해 I2C라인과 Interrupt 신호 라인 각각 하나를 활용하여 전달하기로 결정하였습니다.  **원래는 무선 신호를 주는 것이 맞으나.. 테스트 환경에서 이미 drone과 MissionPlanner가 datalink 로 무선통신을 하는 중에 wifi 또는 BT등의 추가 무선통신을 활용할 시 변수로 작용할 수 있어 유선으로 대체하였습니다.**  **< I2C Command Table >**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Command** | **I2C Address (7bit)** | **I2C Address (8bit)** | **Data (Byte)** | **Description** | | Forward | 0x04 | 0x08 | 0x01 | Make drone moving to forward | | Backward | 0x04 | 0x08 | 0x02 | Make drone moving to backward | | Left | 0x04 | 0x08 | 0x03 | Make drone moving to left | | Right | 0x04 | 0x08 | 0x04 | Make drone moving to right |   위 명령 표를 기반으로, print만 하던 명령을 실제로 gcs상에 데이터를 보내게 변경하였습니다.  추가적으로 안정화를 위해 interrupt 핀 하나로 드론 정지 명령을 아래 상황에 보내기로 하였습니다.  - 정 중앙에 위치하였을 경우  - 화면상의 드론을 감지하여 이동 명령 중 감지를 실패하여 위치를 놓친 경우   1. a 2. a 3. a 4. a 5. a 6. a 7. a 8. a   [참고 자료]  Capstone\_Design\1.PrecisionLandingModule\2.Version\_1.0\3.LandingStation\1.Src\1. YOLOv8\_detect\_drones\_model파일의 “Detect\_drones\_Capstone.ipynb”, 이 외 산출물 참조 | | | | | | |
| 기록자 | | 김교원 | 점검자 | | 박정은 | |
| 서명 | | (sign) | 서명 | | (sign) | |