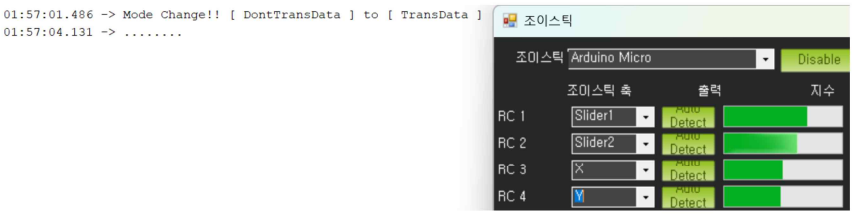
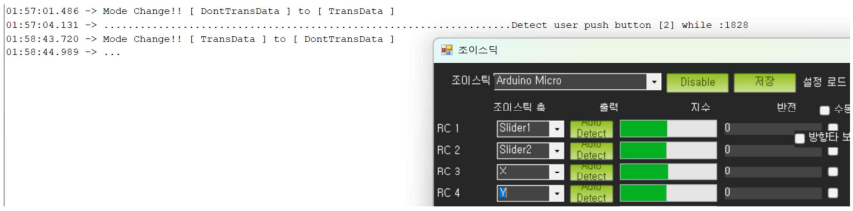
Mode Change - TransData 모드 일 시, MissionPlanner 앱에서 정상적으로 인식하는 모습



Mode Change - DontTransData 모드 일 시, MissionPlanner 앱에서 받지 않는 모습



- 조종기 F/W 개발 (3 차)

2 차 개발에 이어 추가 기능들을 개발하였습니다.

[개발 항목]

(1). 드론에 보낼 전, 후, 좌, 우, 정지 동작 구현

(2). LandingStation 으로부터 Byte 단위의 Command 명령을 I2C 로 받아 저장.

(3). 저장된 Command 에 따른 동작 수행.

Main Loop 동작

```
}else{      // CapstoneGCS.getStateMode() == DontTransData "AutoMode"
  CapstoneGCS.getSignalValues(command);
  CapstoneGCS.processSignal();
}
```

여기서 command 값은 RaspberryPi 에서 I2C 라인을 통해 받고, 받는 시점에 receiveEvent 를 통해 전역변수인 command 에 저장됩니다.

```
Wire.begin(I2C_ADDRESS);
Wire.onReceive(receiveEvent);
```

```
void receiveEvent(){
  command = Wire.read();
  // Serial.print("receiveEvent() : ");
  // Serial.println(command);
}
```

getSignalValues() 동작

```
void PSL_GCS_::getSignalValues(byte command){
    Value_Update_Signals = command;
}
```

말 그대로 전역변수 command 값을 class 변수안에 저장합니다.

processSignal() 동작

```
void PSL_GCS_::processSignal(){
    if(Value_Signals != Value_Update_Signals){
        Value_Signals = Value_Update_Signals;
        State_diffSignals = true;
    } else State_diffSignals = false;

    if(State_diffSignals == true ){
        Serial.print("Get Mission!! [ ");
        Serial.print(Value_Signals);
        Serial.println(" ]");

        switch(Value_Signals){
            case D_Foward:
                joystick.setYAxis(ANALOG_MIDDLE_VAL + ANALOG_MOVE_VAL);
                break;
            case D_Backward:
                joystick.setYAxis(ANALOG_MIDDLE_VAL - ANALOG_MOVE_VAL);
                break;
            case D_Left:
                joystick.setXAxis(ANALOG_MIDDLE_VAL - ANALOG_MOVE_VAL);
                break;
            case D_Right:
               joystick.setXAxis(ANALOG_MIDDLE_VAL + ANALOG_MOVE_VAL);
                break;
        }
        sendGcsData();
    }
}
```

수집한 command 값이 변경되면 신규 command 로 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.

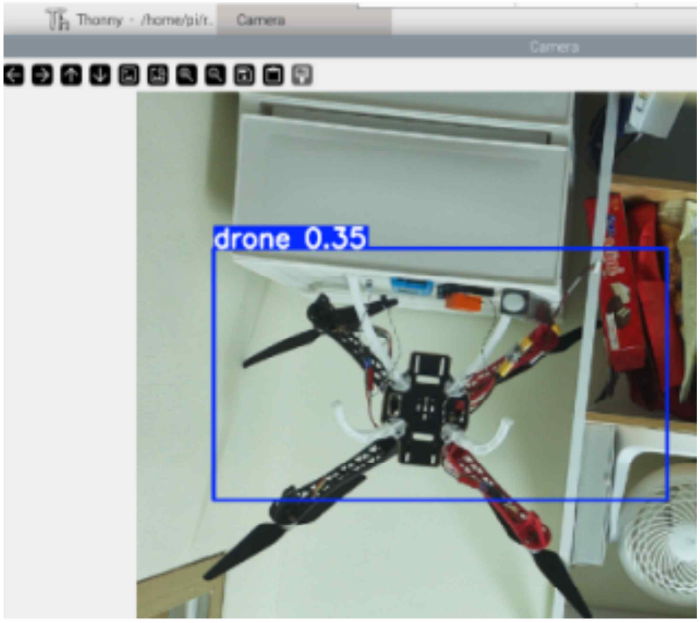
```
void PSL_GCS_::setDroneStop(){
    setJoystickValueToMiddle();
    sendGcsData();
}
```

[참고 자료]

Capstone_Design\1.PrecisionLandingModule\2.Version_1.0\2.GroudControlStation\1.Src\joystick 의 joystick.ino 참조

Capstone_Design\1.PrecisionLandingModule\2.Version_1.0\2.GroudControlStation\1.Src\PSL_GCS 폴더 참조

기록자	김교원	점검자	박정은
서명	(sign)	서명	(sign)



=> 10 -> 30 에 비해 30 -> 100 은 큰 변화량이 없어 보임.

< 4 차 학습 - epochs 185 / 모델 학습 소요시간 : 10 시간 >

변경안

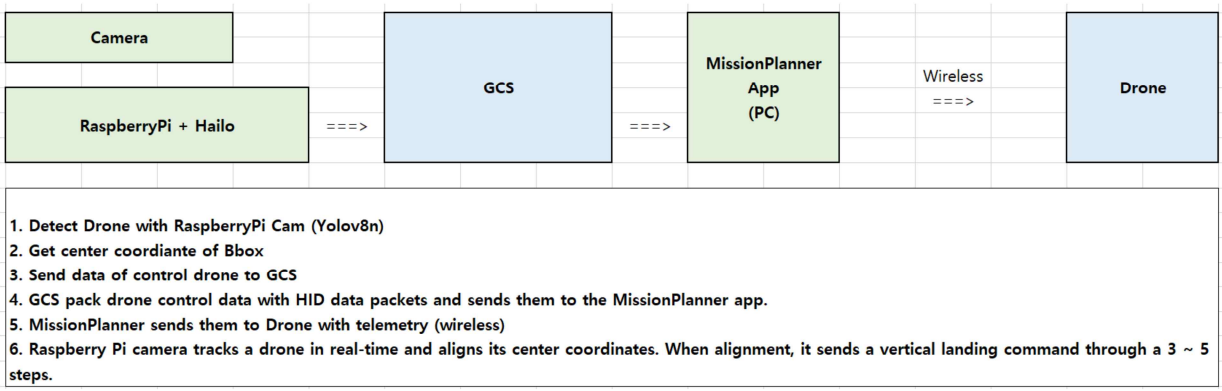
- 최대 epoch 600 까지 증가. (단 정확도 차이 없을 시, 학습 조기 중단 기능이 있어 185 까지 학습함)
- autobatch 활성화로 그래픽인 T4 메모리 최적화
- 학습률 조정 (0.01)
- 가중치 감쇠 적용 (0.0001 ~ 0.01 범위 중, 0.0005 로 설정)

결과적으론 과적합 현상이 있었는지 epoch 100 의 모델보다 인식율이 현저히 떨어졌습니다.

(화면상에 10 번 중 3 번 정도만이 인식됨)

6. 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 BBox 의 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 코드에서는 명령을 단순 print 시켰으나, 이제 실제로 GCS 상에 이동 명령을 보내기 위해 I2C 라인과 Interrupt 신호 라인 각각 하나를 활용하여 전달하기로 결정하였습니다.

원래는 무선 신호를 주는 것이 맞으나.. 테스트 환경에서 이미 drone 과 MissionPlanner 가 datalink 로 무선통신을 하는 중에 wifi 또는 BT 등의 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, print 만 하던 명령을 실제로 gcs 상에 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

테스트 결과 MissionPlanner App 상의 조이스틱 값에
카메라 화면상에 잡힌 드론에 위치에 따른 command 명령어가 정확하게 갱신되는 것을 확인하였습니다.

7. a
8. a
9. a
10. a
11. a
12. a
13. a
14. a
15.