205 - Analyze Drone Log File

Team Information

Team Name : LuckyVicky

Team Member : Eungchang Lee, Hyun Yi, Juho Heo, Dongkyu Lee

Email Address : dfc_luckyvicky@googlegroups.com

Instructions

Description A drone has been discovered flying around a specific area. For an unknown reason, the drone fell to the ground. Upon analyzing the drone, it was found that the log files containing the flight path were encrypted. Analyze the log files and write the answers to the following questions.

Target	Hash (MD5)
00000005.BIN	3a8b11ee13957438c0257c6587ba2e40

Questions

1. Submit the GPS location where the drone returned. (50 points)
2. Submit the code used for decryption. (150 points)

Teams must:

- Develop and document the step-by-step approach used to solve this problem to allow another examiner to replicate team actions and results.
- Specify all tools used in deriving the conclusion(s).

Tools used:

Name:	HashTab	Publisher:	Implbits Software
Version:	6.0.0		
URL:	https://implbits.com		

Name:	HxD	Publisher:	Maël Hörz
Version:	2.5.0.0		
URL:	https://www.mh-nexus.de		

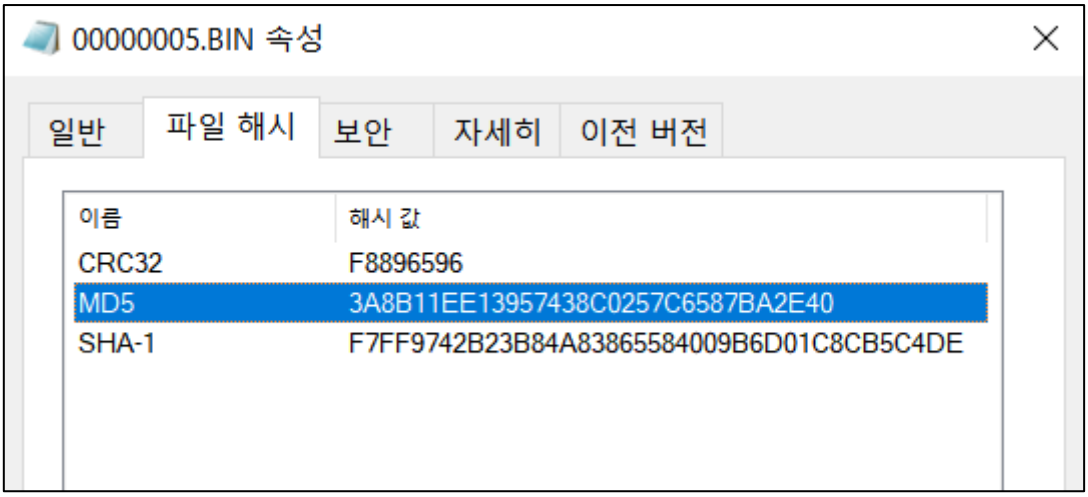
Name:	pymavlink	Publisher:	Ardupilot
Version:	2.4.41		
URL:	https://github.com/ArduPilot/pymavlink		

Name:	MissionPlanner	Publisher:	Ardupilot
Version:	1.3.82		
URL:	https://github.com/ArduPilot/MissionPlanner		

Name:	UAVLogReader	Publisher:	Ardupilot
Version:	1.0.1		
URL:	https://plot.ardupilot.org/ https://plotbeta.ardupilot.org/		

Name:	Visual Studio Code	Publisher:	Microsoft
Version:	1.93.1		
URL:	https://code.visualstudio.com/		

Step-by-step methodology:



[그림 1] 해시 값 확인

Target file인 00000005.BIN의 MD5 해시 값이 일치함을 확인하였습니다.

1. Submit the GPS location where the drone returned. (50 points)

00000005.BIN

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	A3	95	80	80	59	46	4D	54	00	42	42	6E	4E	5A	00	00	£•€€YFMT.BBnNZ..
00000010	00	00	00	00	00	00	00	00	00	54	79	70	65	2C	4C	65	.........Type,Le
00000020	6E	67	74	68	2C	4E	61	6D	65	2C	46	6F	72	6D	61	74	ngth,Name,Format
00000030	2C	43	6F	6C	75	6D	6E	73	00	00	00	00	00	00	00	00	,Columns........
00000040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000050	00	00	00	00	00	00	00	00	00	A3	95	80	74	4C	55	4E	.........£•€tLUN
00000060	49	54	51	62	5A	00	00	00	00	00	00	00	00	00	00	00	ITQbZ...........
00000070	00	00	54	69	6D	65	55	53	2C	49	64	2C	4C	61	62	65	..TimeUS,Id,Labe
00000080	6C	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	l...............
00000090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000000A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000000B0	00	00	A3	95	80	73	2C	46	4D	54	55	51	42	4E	4E	00	..£•€s,FMTUQBNN.
000000C0	00	00	00	00	00	00	00	00	00	00	00	54	69	6D	65	55	...........TimeU
000000D0	53	2C	46	6D	74	54	79	70	65	2C	55	6E	69	74	49	64	S,FmtType,UnitId
000000E0	73	2C	4D	75	6C	74	49	64	73	00	00	00	00	00	00	00	s,MultIds.......

[그림 1] Raw값 확인 - 1

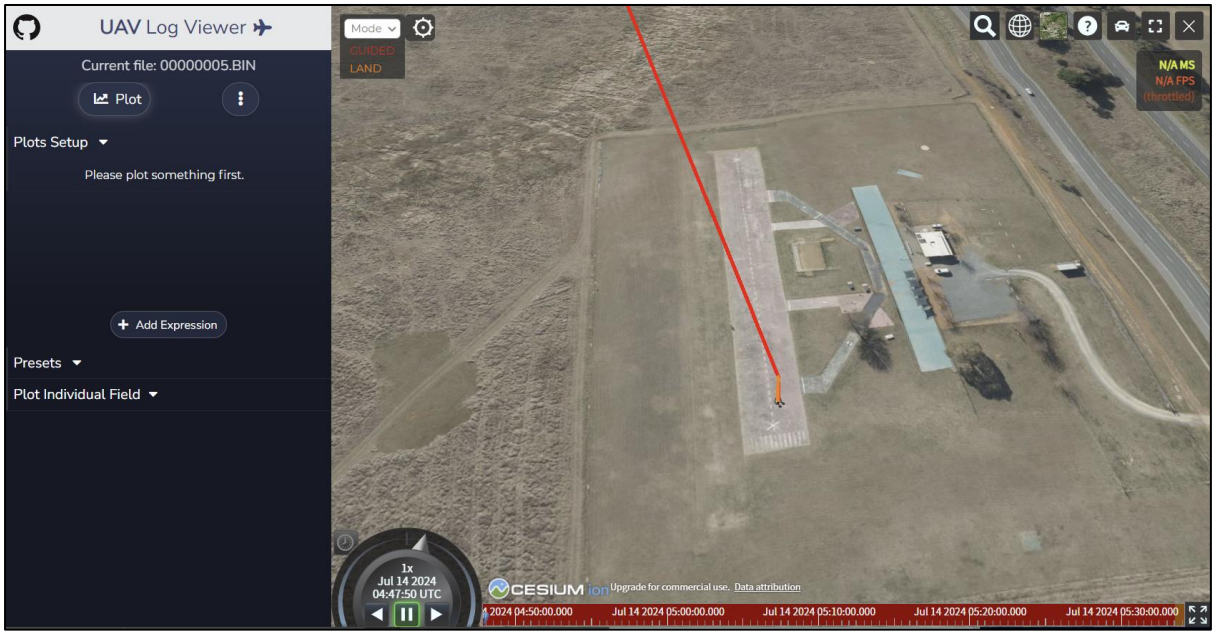
00000005.BIN

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00018C00	CD	CC	4C	3E	CD	CC	4C	3E	A3	95	3A	8B	DE	93	02	00	ÍÌL>ÍÌL>£•:‹Þ"‥
00018C10	00	00	00	01	10	C4	92	3B	83	29	09	3B	C8	C2	50	BB	.....Ä';ƒ).;ÈÂP»
00018C20	7C	17	85	BB	79	00	31	3A	EA	62	43	BA	12	09	84	3B	|.…»y.1:êbC°..„;
00018C30	24	0D	54	3B	14	04	05	BB	3E	21	93	BB	A3	95	2C	8B	$.T;...»>!"»£•,‹
00018C40	DE	93	02	00	00	00	00	01	09	00	08	00	CC	8A	EC	29	Þ"..........ÌŠì)
00018C50	35	BC	40	F5	57	3C	FE	90	0B	BA	2E	1C	5D	B2	08	5C	5¼@õW<þ..°..]².\
00018C60	D9	BA	6C	87	0F	3B	91	E1	B8	3B	01	00	02	00	EE	FF	Ù°l‡.;'á¸;....îÿ
00018C70	28	E4	00	00	A3	95	2D	8B	DE	93	02	00	00	00	00	01	(ä..£•-‹Þ"......
00018C80	00	00	00	00	FF	FF	00	00	00	00	E8	00	34	00	F0	FD	....ÿÿ....è.4.ðý
00018C90	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00018CA0	00	00	A3	95	2E	8B	DE	93	02	00	00	00	00	01	FF	FF	..£•.‹Þ"......ÿÿ
00018CB0	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....

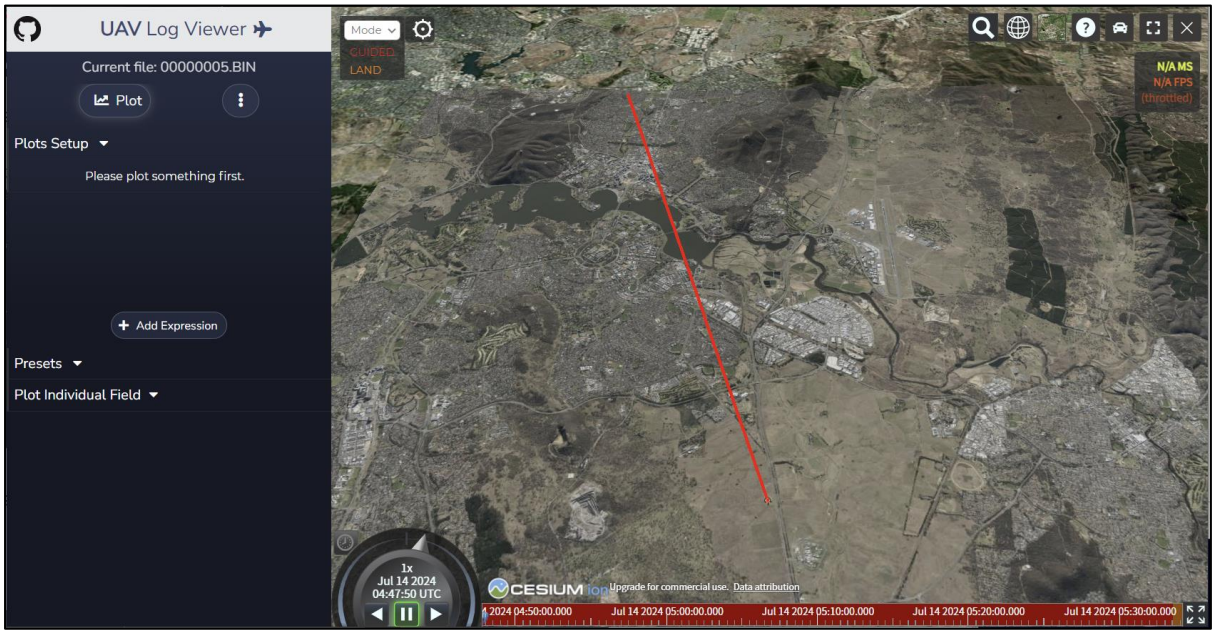
[그림 2] Raw값 확인 - 2

Target File로 주어진 .bin 파일은 https://ardupilot.org/copter/docs/common-downloading-and-analyzing-data-logs-in-mission-planner.html 에서 dataflash log라는 것을 짐작할 수 있습니다. 해당 파일을 위 두 그림과 같이 HxD를 통해 Raw값을 확인해보았을 때, 첫 그림처럼 로그 타입 내 파라미터 데이터들을 확인할 수 있는 부분도 존재하고 두번째 그림처럼 확인할 수 없는 부분도 존재했습니다.

UavLogViewer 상에서 해당 00000005.BIN 파일을 로드해보면, 다음과 같은 비행 경로들을 확인할 수 있었습니다.

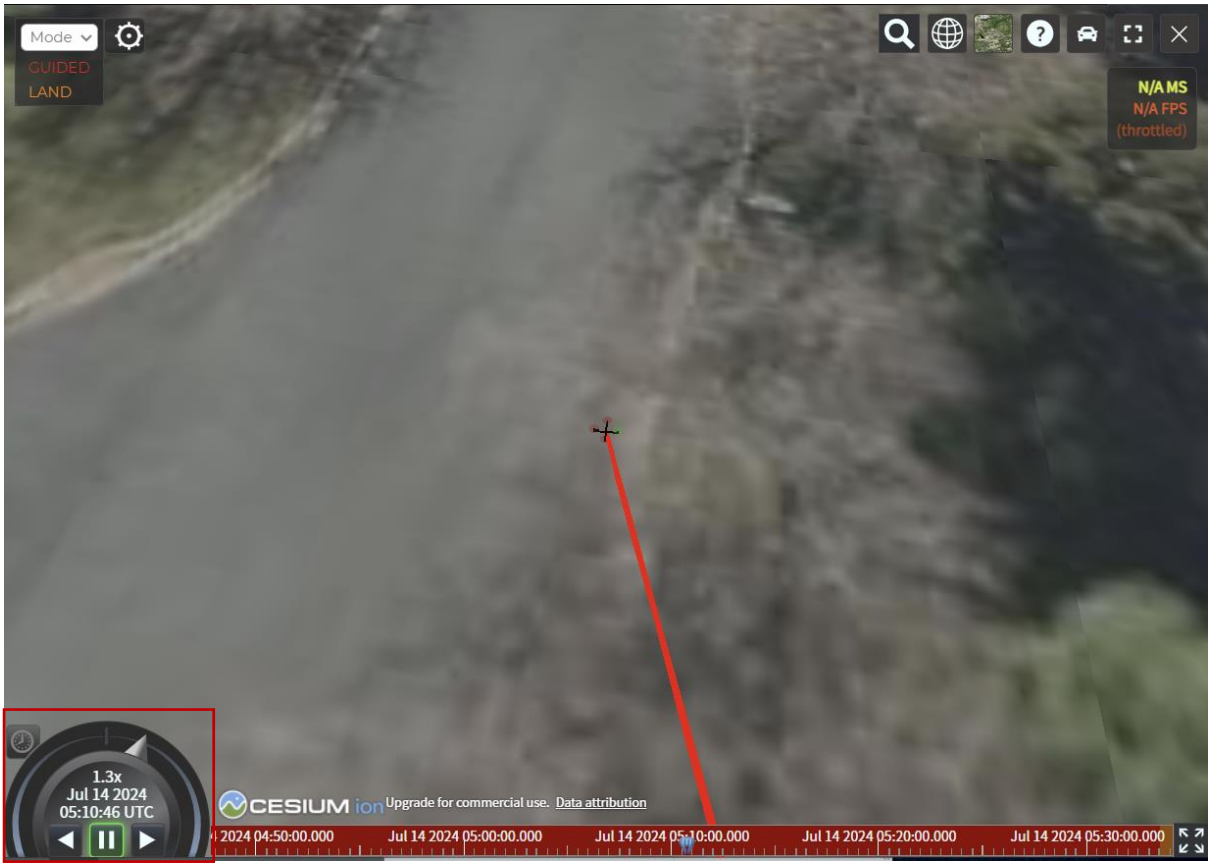


[그림 3] target file의 비행 경로 확인(시작 지점)



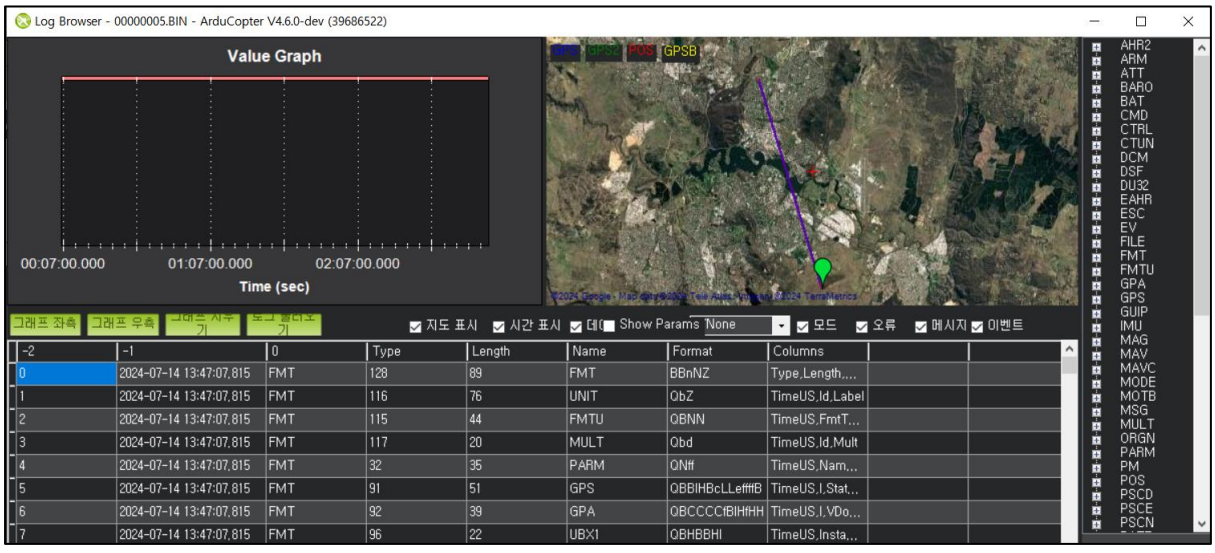
[그림 4] target file의 비행 경로 확인(전체)

위 두 그림을 통해서 드론의 대략적인 비행 경로를 알 수 있었습니다.



[그림 5] 드론의 비행 경로 중 반환점

위 그림은 비행 경로에서 확대해보았을 때, 드론이 해당 지점에서 돌아오는 것을 확인할 수 있었습니다.



-2	-1	0	Type	Length	Name	Format	Columns
0	2024-07-14 13:47:07.815	FMT	128	89	FMT	BBnNZ	Type,Length,...
1	2024-07-14 13:47:07.815	FMT	116	76	UNIT	QbZ	TimeUS,Id,Label
2	2024-07-14 13:47:07.815	FMT	115	44	FMTU	QBNN	TimeUS,FmtT,...
3	2024-07-14 13:47:07.815	FMT	117	20	MULT	Qbd	TimeUS,Id,Mult
4	2024-07-14 13:47:07.815	FMT	32	35	PARM	QNf	TimeUS,Nam,...
5	2024-07-14 13:47:07.815	FMT	91	51	GPS	QBBIHBcLLefffB	TimeUS,I,Stat,...
6	2024-07-14 13:47:07.815	FMT	92	39	GPA	QBCCCCfBIHfHH	TimeUS,I,VDo,...
7	2024-07-14 13:47:07.815	FMT	96	22	UBX1	QBHBBHI	TimeUS,Insta,...

[그림 6] Mission Planner로 확인한 00000005.BIN 파일의 로그 검토 기록

마찬가지로, Mission Planner상에서도 map view를 보면 UAV 뷰어와 동일함을 교차검증을 통해 알 수 있습니다.

정확한 반환 지점에 대한 GPS 값을 확인하기 위해 해당 BIN파일의 로그를 분석하기 쉽게 변환하고자 하였습니다. Ardupilot/pymavlink의 mavlogdump.py를 통해 변환하고자 하였고, 교차 검증을 위해 mission planner의 bin to log 기능도 활용하였습니다.

```
2024-07-14 13:47:50.65: MSG {TimeUS: 42842023, Message: ArduCopter V4.6.0-dev (39686522)}
2024-07-14 13:47:50.65: VER {TimeUS: 42842023, BT: 3, BST: 65535, Maj: 4, Min: 6, Pat: 0, FWT: 0, GH: 963142946, FWS: ArduCopter V4.6.0-dev (39686522), AP
2024-07-14 13:47:50.65: MSG {TimeUS: 42842023, Message: 96eb04c5579b46648496eb5435a10a9c}
2024-07-14 13:47:50.65: MSG {TimeUS: 42842023, Message: Param space used: 254/4096}
2024-07-14 13:47:50.65: MSG {TimeUS: 42842023, Message: RC Protocol: SITL}
2024-07-14 13:47:50.65: MSG {TimeUS: 42842023, Message: New mission}
2024-07-14 13:47:50.65: CMD {TimeUS: 42842023, CTot: 1, CNum: 0, CId: 16, Prm1: 0.0, Prm2: 0.0, Prm3: 0.0, Prm4: 0.0, Lat: -35.3632621, Lng: 149.1652374,
2024-07-14 13:47:50.65: MSG {TimeUS: 42842023, Message: New rally}
2024-07-14 13:47:50.65: MSG {TimeUS: 42842023, Message: New fence}
2024-07-14 13:47:50.65: MSG {TimeUS: 42842023, Message: Frame: QUAD/PLUS}
2024-07-14 13:47:50.65: MODE {TimeUS: 42842023, Mode: 4, ModeNum: 4, Rsn: 2}
```

[그림 7] mavlogdump.py로 뽑아낸 로그 중 일부

```
MSG, 42842023, ArduCopter V4.6.0-dev (39686522)
VER, 42842023, 3, 65535, 4, 6, 0, 0, 963142946, ArduCopter V4.6.0-dev (39686522), 0, 2, 2
MSG, 42842023, 96eb04c5579b46648496eb5435a10a9c
MSG, 42842023, Param space used: 254/4096
MSG, 42842023, RC Protocol: SITL
MSG, 42842023, New mission
CMD, 42842023, 1, 0, 16, 0, 0, 0, 0, -35.3632621, 149.1652374, 584.08, 0
MSG, 42842023, New rally
MSG, 42842023, New fence
MSG, 42842023, Frame: QUAD/PLUS
MODE, 42842023, Guided, 4, 2
ORGN, 42842023, 0, -35.3632621, 149.1652374, 584.09
ORGN, 42842023, 1, -35.3632621, 149.1652374, 584.08
```

[그림 8] Mission Planner의 bin to log 기능을 통해 뽑아낸 로그 중 일부 – 1

위 두 그림을 통해 로깅을 수행한 드론 장치는 ArduCopter로 보이며, RC Protocol : SITL이라는 부분을 통해 실제 드론 없이 컴퓨터 상에서 비행을 시뮬레이션 한 환경에서 로깅을 진행했음을 파악할 수 있습니다. 또한, Guided모드를 통해 목표 지점으로 지정된 위치 또는 경로로 이동하는 자동 비행 모드로 볼 수 있습니다. 그 부분은 사전에 New Mission이라는 부분을 통해서 명령 기입이 되었음을 추측해볼 수 있습니다.

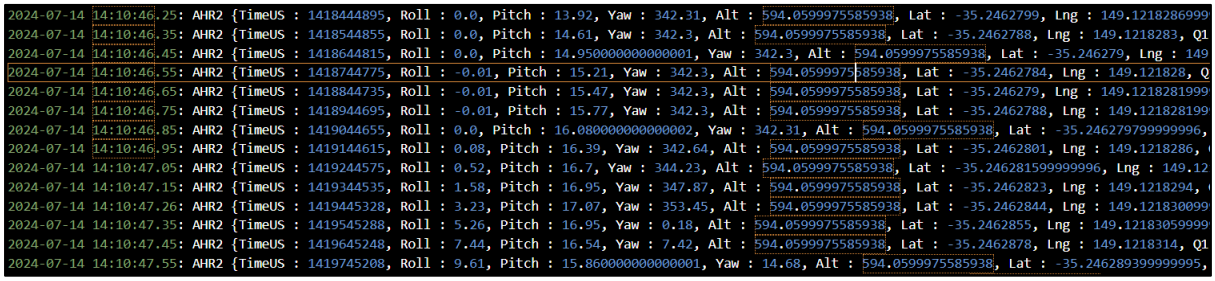
```
PARM, 42842023, SIM_OPOS_LAT, -35.36326, -35.36326
PARM, 42842023, SIM_OPOS_LNG, 149.1652, 149.1652
PARM, 42842023, SIM_OPOS_ALT, 584, 584
PARM, 42842023, SIM_OPOS_HDG, 353, 353
```

[그림 9] Mission Planner의 bin to log 기능을 통해 뽑아낸 로그 중 일부 - 2

그리고 위 그림을 통해 vehicle의 시작 위도, 경도, 고도, 헤딩 방향을 알 수 있었습니다.

해당 기기의 위도 경도 값을 나타내는 유형은 GPS, POS, AHR2, SIM 등으로 다양했는데, 위도나 경도뿐만 아니라 Roll, Pitch, Yaw 데이터를 포함하여 Q1, Q2, Q3, Q4 값까지 시뮬레이션 환경에서의 드론 결과 데이터를 보여주는 SIM 유형의 로그를 중점으로 두고 AHR2와 POS 로그도 살펴보았습니다.

먼저, AHRS(자세방향기준시스템)의 백업 데이터를 나타내는 AHR2 로그를 살펴보았습니다. [그림 5]에서 드론이 반환점을 돌 때 기록된 시간 위주로 살펴보았습니다.



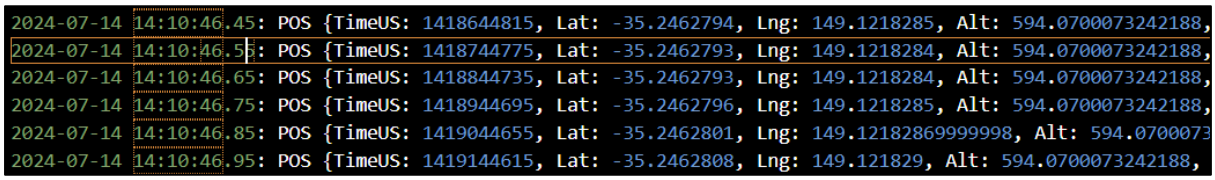
[그림 10] AHR2 로그만 추출한 결과 중 일부

Timestamp	Yaw	Lat(소수점 7자리 반올림)	Lng(소수점 7자리 반올림)
2024-07-14 14:10:46.35	342.3	-35.2462788	149.1218283
2024-07-14 14:10:46.45	342.3	-35.2462790	149.1218282
2024-07-14 14:10:46.55	342.3	**-35.2462784**	**149.1218280**
2024-07-14 14:10:46.65	342.3	-35.2462790	149.1218282
2024-07-14 14:10:46.75	342.3	-35.2462788	149.1218282
2024-07-14 14:10:46.85	342.31	-35.2462798	149.1218285
2024-07-14 14:10:46.95	342.64	-35.2462801	149.1218286
2024-07-14 14:10:47.05	344.23	-35.2462816	149.1218291
2024-07-14 14:10:47.15	347.87	-35.2462823	149.1218294
2024-07-14 14:10:47.26	353.45	-35.2462844	149.1218300
2024-07-14 14:10:47.35	0.18	-35.2462855	149.1218305
2024-07-14 14:10:47.45	7.42	-35.2462878	149.1218314
2024-07-14 14:10:47.55	14.68	-35.2462894	149.1218319

[표 1] AHR2에서의 Yaw, Lat, Lng 데이터

위 그림을 기반으로 정리한 표에서 확인한 결과, 2024-07-14 14:10:46.55 경에 드론이 북쪽에서 서쪽으로 약 18도 기울어진 342.3도 방향으로 비행을 하는 과정에서 위도와 경도 값이 가장 낮게 나온 것을 알 수 있습니다. 시작 지점으로부터 해당 heading 방향으로 갈수록 위도(-를 제외했을 때)와 경도는 낮아지기 때문에 가장 낮은 지점이 반환지점이라고 생각했습니다. 반환지점을 찍고 도는 과정에서 Yaw값이 상승하면서 시계 방향으로 회전을 하여 0부터 값이 증가하는 것도 알 수 있습니다.

다음은 캐노니컬 장치에서 인식하는 기체 위치를 나타내는 POS 로그입니다.



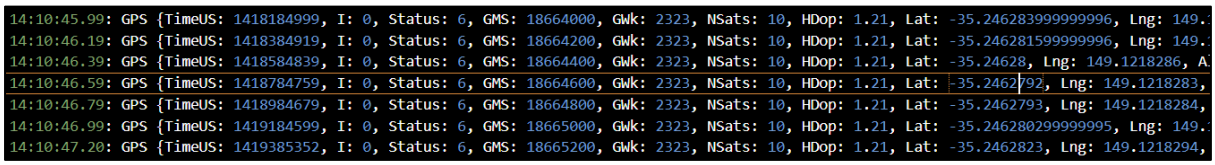
[그림 11] POS 로그만 추출한 결과 중 일부

Timestamp	Lat(소수점 7자리 반올림)	Lng(소수점 7자리 반올림)
2024-07-14 14:10:46.45	-35.2462794	149.1218285
2024-07-14 14:10:46.55	**-35.2462793**	**149.1218284**
2024-07-14 14:10:46.65	-35.2462793	149.1218284

[표 2] POS 로그에서 Lat, Lng 데이터

POS 로그의 경우에도 2024-07-14 14:10:46.55 경에 가장 반환 지점 끝에 가까운 위도와 경도 값을 보였습니다.

다음은 GNSS 시스템에서 수신된 정보를 나타내는 GPS 로그입니다.



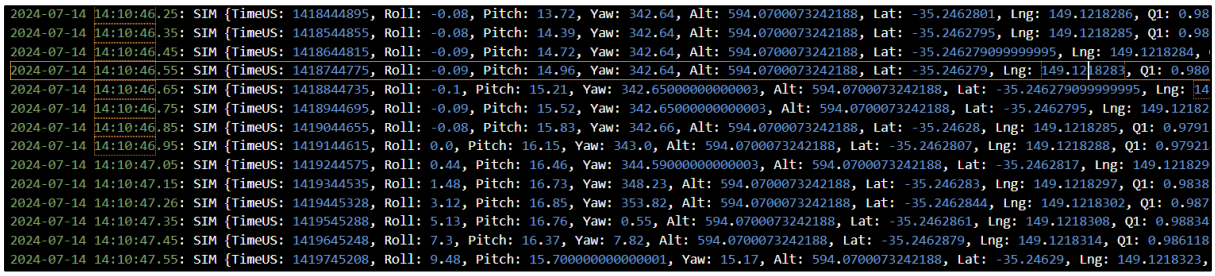
[그림 12] GPS 로그만 추출한 결과 중 일부

Timestamp	Lat(소수점 7자리 반올림)	Lng(소수점 7자리 반올림)
2024-07-14 14:10:46.39	-35.24628	149.1218286
2024-07-14 14:10:46.59	**-35.2462792**	**149.1218283**
2024-07-14 14:10:46.79	-35.2462793	149.1218284

[표 3] GPS 로그에서 Lat, Lng 데이터

GPS 로그는 시간 값은 미세하게 다른 2024-07-14 14:10:46.59 경에 가장 반환 지점 끝에 가까운 위도와 경도 값을 보였습니다.

마지막으로 SIM데이터도 살펴보도록 하겠습니다.



[그림 12] log에서 SIM 데이터만 추출한 결과 중 일부

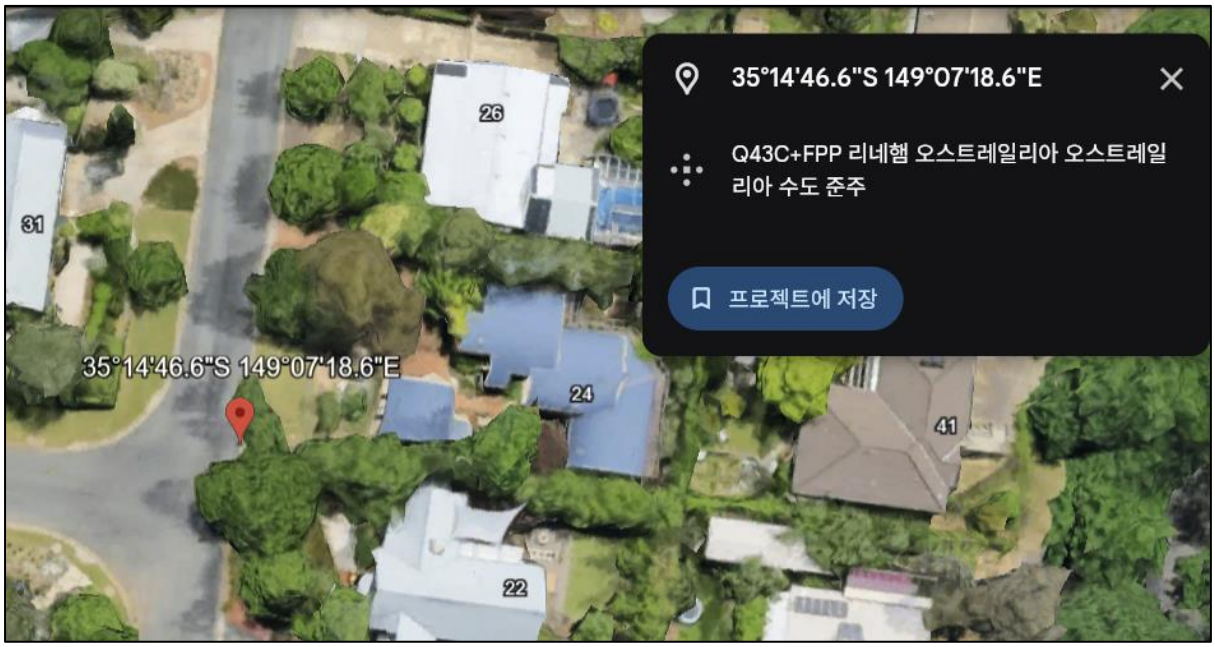
Timestamp	Yaw	Lat(소수점 7자리 반올림)	Lng(소수점 7자리 반올림)
2024-07-14 14:10:46.35	342.64	-35.2462795	149.1218285
2024-07-14 14:10:46.45	342.64	-35.2462791	149.1218284
2024-07-14 14:10:46.55	342.64	**-35.2462790**	**149.1218283**
2024-07-14 14:10:46.65	342.65	-35.2462791	149.1218283
2024-07-14 14:10:46.75	342.65	-35.2462795	149.1218285
2024-07-14 14:10:46.85	342.66	-35.2462800	149.1218285
2024-07-14 14:10:46.95	343.0	-35.2462807	149.1218288
2024-07-14 14:10:47.05	344.59	-35.2462817	149.1218291
2024-07-14 14:10:47.15	348.23	-35.2462830	149.1218297
2024-07-14 14:10:47.26	352.82	-35.2462844	149.1218302
2024-07-14 14:10:47.35	0.55	-35.2462861	149.1218308
2024-07-14 14:10:47.45	7.82	-35.2462879	149.1218314
2024-07-14 14:10:47.55	15.17	-35.2462900	149.1218323

[표 4] SIM에서의 Yaw, Lat, Lng

오히려 SIM데이터에서도 같은 시각인 2024-07-14 14:10:46.55경에 AHR2에서보다 위도와 경도 값이 높은 것을 알 수 있습니다. 그리고 Yaw가 감소했다가 다시 증가하면서 반환 지점을 도는 것을 알 수 있습니다.

이를 통해 여러 유형의 로그 데이터를 살펴보았을 때 드론이 반환지점을 도는 GPS 값은 같은 시간에서 위도와 경도 차이는 미미하게 다르기 때문에 공통적인 GPS location 값을 소수점 다섯 째자리까지 버림으로 잘라서 도출한다면 다음과 같습니다. 각 로그타입에 따른 드론 반환지점에 대한 GPS값은 표에 기록하였습니다.

Lat : -35.24627, Lng : 149.12182



[그림 13] 구글 어스로 확인한 반환지점의 GPS 위치

2. Submit the code used for decryption. (150 points)

위 1번에서 뷰어를 통해 타깃 파일의 전반적인 비행 경로를 파악할 수 있었으며, 자세한 비행 경로를 파악하기 위해서 mavlogdump.py나 Mission Planner에서 제공하는 BIN 데이터에서 log 데이터로 변환해 주는 기능도 사용했었습니다.

사실 flight path 자체가 특정한 키를 통해 복호화 과정 없이 위 두 로그 변환 도구를 통해 지도에 시각화 해보았을 때 경로가 잘 나왔으며, 교차검증 자체도 별다른 문제는 없었습니다. 따라서, 바이너리 데이터 자체가 수치 데이터들은 바로 확인할 수 없이 매핑 과정을 거쳐야 하기 때문에 변환 과정에서의 decrypt라 생각하고 코드들을 제출합니다. 또한, 정확히 flight path를 파악하기 위해 decryption에 사용했던 오픈소스 코드를 제출하는 부분인 것인지, decryption에 사용하기 위해 개발한 코드를 의미하는 부분인지 판단하기 어려워 두 부분 모두 제출 드립니다.

먼저 오픈소스로 사용한 **Mission Planner의 BinaryLog.cs, DFLogBuffer.cs와 pymavlink mavlogdump.py, mavutil.py**는 bin to log를 통해 바이너리 데이터를 decrypt 해독하여 log 내 flight view를 확인하기 위해 사용한 코드이며, 연관된 유틸리티 코드가 많고 코드 길이가 길어 첨부파일로 제출합니다.

또한, 그런 오픈소스 코드들을 참고하여 BIN데이터를 decrypt하여 비행 경로를 간편하게 파악하는데 저희 팀에서 사용한 코드를 다음 페이지에 첨부하며, 첨부파일로도 제출합니다.

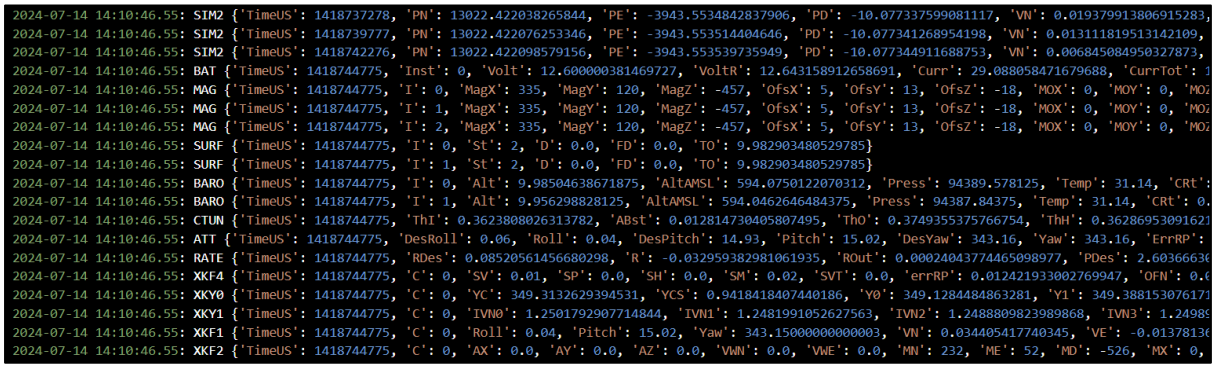
decrypt_bin.py

```
#!/usr/bin/env python

from pymavlink import mavutil
import sys
from datetime import datetime, timedelta, timezone
import os

def format_timestamp_kst(timestamp_us):
    timestamp_sec = timestamp_us / 1_000_000
    #utc_dt = datetime.utcfromtimestamp(timestamp_sec)
    utc_dt = datetime.fromtimestamp(timestamp_sec, tz=timezone.utc)
    kst_dt = utc_dt + timedelta(hours=9)
    return kst_dt.strftime('%Y-%m-%d %H:%M:%S.') + f"{int(kst_dt.microsecond / 10_000):02d}"

def main(input_log_file, output_file):
    if not os.path.isfile(input_log_file):
        print(f"Error: Input file '{input_log_file}' does not exist.")
        sys.exit(1)

    mlog = mavutil.mavlink_connection(
        input_log_file,
        dialect='ardupilotmega',
        notimestamps=False,
        robust_parsing=True
    )

    print(f"Reading log file: {input_log_file}")
    print(f"Writing output to: {output_file}")

    try:
        with open(output_file, 'w') as outfile:
            while True:
                msg = mlog.recv_match(blocking=True)
                if msg is None:
                    break

                timestamp_sec = msg._timestamp
                timestamp_us = int(timestamp_sec * 1_000_000)
```

```
                ts_str = format_timestamp_kst(timestamp_us)
                msg_dict = msg.to_dict()
                msg_dict.pop('mavpackettype', None)

                outfile.write(f"{ts_str}: {msg.get_type()} {msg_dict}₩n")
    except KeyboardInterrupt:
        print("Interrupted by user.")
    except Exception as ex:
        print(f"An error occurred: {ex}")
    finally:
        mlog.close()
        print("Processing complete.")

if __name__ == "__main__":
    if len(sys.argv) != 3:
        print("Usage: python read_log_to_file.py <input_log_file> <output_file>")
        sys.exit(1)

    input_log_filename = sys.argv[1]
    output_filename = sys.argv[2]
    main(input_log_filename, output_filename)
```

[표 5] binary 데이터 복호화 코드

python3 decrypt_bin.py 00000005.BIN [output log 이름]를 통해 실행하고, 나온 데이터를 일부 살펴보면 다음과 같습니다.



[그림 14] output.log 일부

추가로, 보고 싶은 타입의 로그 파일만 추출하기 위한 extract.py도 제출합니다.

extract.py

```
import argparse

def process_log(input_file, output_file, param_type):
    """
    주어진 입력 로그 파일을 읽고, 특정 조건에 맞는 라인만 출력 파일에 저장합니다.

    Args:
        input_file (str): 입력 로그 파일의 경로.
        output_file (str): 출력 로그 파일의 경로.
        mode (str): 필터링 모드
    """
    try:
        with open(input_file, 'r') as infile, open(output_file, 'w') as outfile:
            for line in infile:
                if param_type in line and '-35' in line and '149' in line:
                    line = line.replace(" :", ":")
                    outfile.write(line.replace("'", ""))

                elif 'MODE' in line and 'ModeNum' in line and 'FMT' not in line:
                    line = line.replace(" :", ":")
                    outfile.write(line.replace("'", ""))
        print(f"{output_file}에 저장이 완료되었습니다.")
    except FileNotFoundError:
        print(f"오류: 파일 '{input_file}'을(를) 찾을 수 없습니다.")
    except Exception as e:
        print(f"알 수 없는 오류가 발생했습니다: {e}")

def main():
    """
    커맨드라인 인자를 파싱하고 로그 파일을 처리합니다.
    """
     parser = argparse.ArgumentParser(description='드론 로그 파일을 처리하는 스크립트입니다.')
    parser.add_argument('input_file', help='입력 로그 파일의 경로')
    parser.add_argument('output_file', help='출력 로그 파일의 경로')
    parser.add_argument('param_type', help='log parameter type')

    args = parser.parse_args()
```

```
    process_log(args.input_file, args.output_file, args.param_type)

if __name__ == '__main__':
    main()
```

[표 6] 특정 유형의 타입 로그 추출 코드

위 코드를 통해 AHR2, GPS, POS, 그리고 SIM데이터를 뽑아서 간편하게 확인할 수 있습니다.

예시) python3 extract.py [위 decrypt_bin에서 뽑은 로그] [output log] [추출하고자 하는 type]

visualization.py

```
import re
import folium
import argparse
import sys

def main():
    parser = argparse.ArgumentParser(description='로그 파일을 읽어 Folium 지도로 변환합니다.')
    parser.add_argument('input_file', help='입력 로그 파일 경로')
    parser.add_argument('-o', '--output_file', default='map.html', help='출력 HTML 파일 이름 (기본: map.html)')
    args = parser.parse_args()

    # 정규 표현식을 사용하여 MODE 및 Lat, Lng 값 추출
    # MODE 패턴: "Mode: 숫자" 또는 "Mode : 숫자"
    mode_pattern = re.compile(r'Mode₩s*:₩s*(₩d+)')

    # Lat, Lng 패턴: "Lat: 숫자" 또는 "Lat : 숫자", "Lng: 숫자" 또는 "Lng : 숫자"
    coord_pattern = re.compile(r'Lat₩s*:₩s*([-+]?₩d+₩.₩d+)₩s*,₩s*Lng₩s*:₩s*([-+]?₩d+₩.₩d+)')

    # 좌표와 모드를 저장할 리스트
    positions = []

    # 현재 모드 초기화
    current_mode = None
```

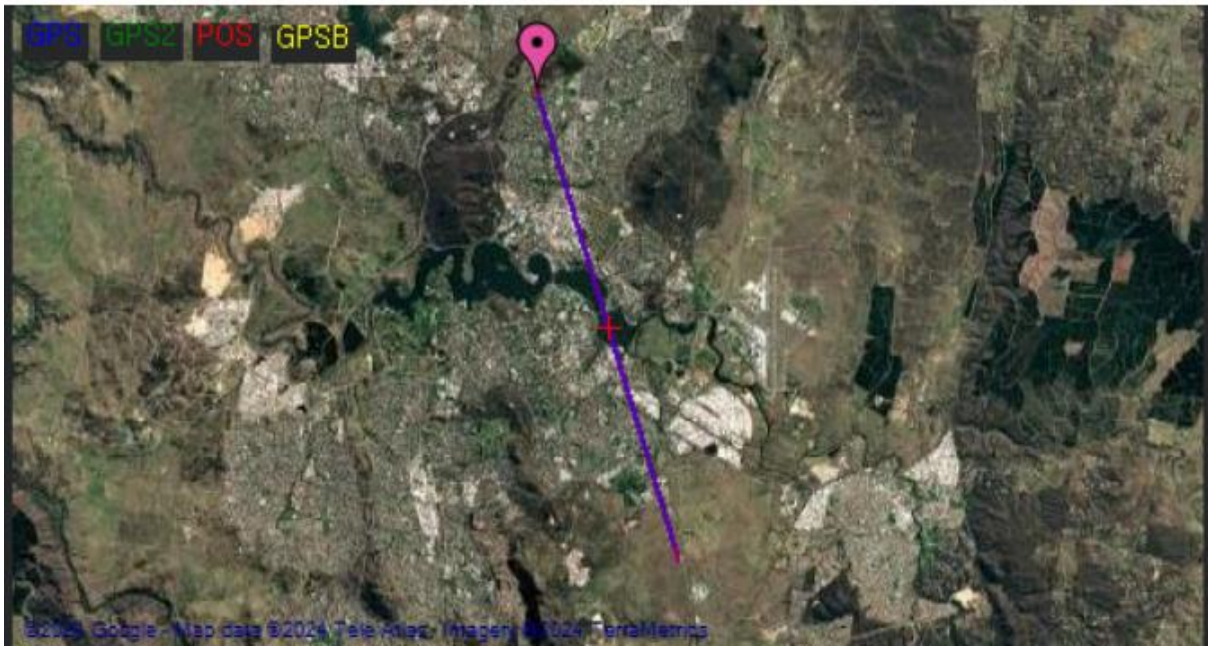
```
# 로그 파일 읽기 및 파싱
try:
    with open(args.input_file, 'r') as file:
        for line in file:
            # MODE 라인 처리
            mode_match = mode_pattern.search(line)
            if mode_match:
                current_mode = mode_match.group(1)

            # Lat, Lng 라인 처리
            coord_match = coord_pattern.search(line)
            if coord_match:
                lat = float(coord_match.group(1))
                lng = float(coord_match.group(2))
                positions.append((lat, lng, current_mode))
except FileNotFoundError:
    print(f"파일을 찾을 수 없습니다: {args.input_file}")
    sys.exit(1)
except Exception as e:
    print(f"파일을 읽는 중 오류가 발생했습니다: {e}")
    sys.exit(1)

# 좌표가 없는 경우 종료
if not positions:
    print("좌표를 찾을 수 없습니다.")
    sys.exit(1)

# 지도의 중심을 첫 번째 좌표로 설정
first_coord = (positions[0][0], positions[0][1])
m = folium.Map(location=first_coord, zoom_start=15)

# 모드에 따른 색상 매핑
color_mapping = {
    '4': 'red',     # GUIDED 모드
    '9': 'blue'     # LAND 모드
}

# 이전 좌표와 모드 초기화
```

```
    prev_coord = None
    prev_mode = None

    # 경로를 폴리라인으로 추가
    for pos in positions:
        current_coord = (pos[0], pos[1])
        current_mode = pos[2]

        if prev_coord is not None and prev_mode in color_mapping:
            # 모드에 따른 색상 결정
            color = color_mapping.get(prev_mode, 'green')    # 기본 색상은 green

            # 폴리라인 추가
            folium.PolyLine([prev_coord, current_coord],
                            color=color,
                            weight=2.5,
                            opacity=1).add_to(m)

        # 이전 좌표와 모드 업데이트
        prev_coord = current_coord
        prev_mode = current_mode

    # 전체 좌표 리스트 생성 (시작점과 끝점 마커용)
    all_coords = [(pos[0], pos[1]) for pos in positions]

    # 지도에 마커 추가 (시작점과 끝점)
    #folium.Marker(all_coords[0], popup='Start', icon=folium.Icon(color='green')).add_to(m)
    #folium.Marker(all_coords[-1], popup='End', icon=folium.Icon(color='red')).add_to(m)

    # 레전드 추가
    legend_html = '''
         <div style="position: fixed;
                     bottom: 50px; left: 50px; width: 150px; height: 90px;
                     border:2px solid grey; z-index:9999; font-size:14px;
                     background-color:white;
                     ">
          <b>Legend</b><br>
              <i  class="fa  fa-map-marker  fa-2x"  style="color:red"></i> GUIDED
Mode<br>
```

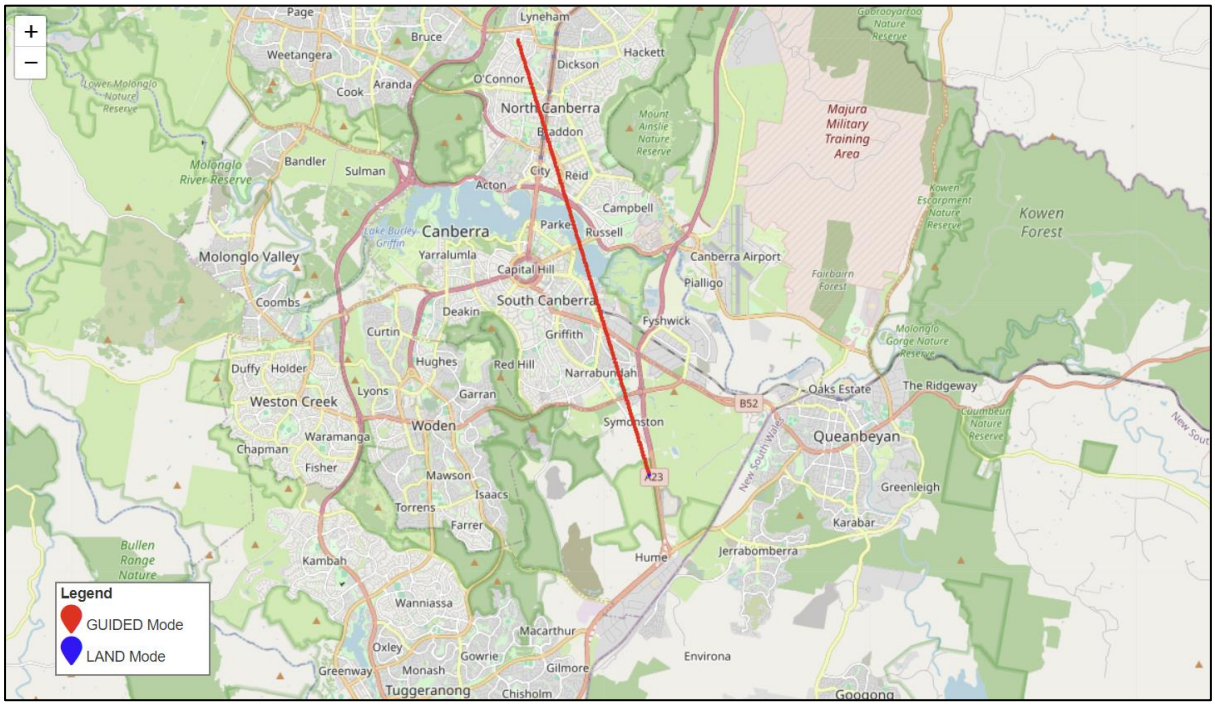
```
             <i class="fa fa-map-marker fa-2x" style="color:blue"></i> LAND Mode
            </div>
            '''

    m.get_root().html.add_child(folium.Element(legend_html))

    # 지도를 HTML 파일로 저장
    try:
        m.save(args.output_file)
        print(f"지도 생성 완료! '{args.output_file}' 파일을 열어 확인하세요.")
    except Exception as e:
        print(f"지도를 저장하는 중 오류가 발생했습니다: {e}")
        sys.exit(1)

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

[표 7] flight path 시각화

python3 visualization.py [extract를 통해 추출한 로그] 명령어를 통해 시각화도 수행하였습니다.



[그림 15] Mission Planner flight path 시각화



[그림 16] 자체 제작 Visualization 시각화