# Matlab Programming Term Project

백창인, 허선, 권우석, 김선용, 박준우

목차 : 1. 개요 및 목적

2. M-file 개발 단계

3. Simulink 개발 단계

4. 3D Animation (VR) 개발 단계

5. 참고 문헌, 자

### 1.1 개요 및 목적

이 코드는 (init_fighter_pos)에 위치한 전투기가 속력 (speed_fighter)의 속도로 +x 방향으로 정속도 이동을 하고 출발과 동시에 (init_missile_pos)의 위치에 있는 미사일이 발사되어 미사일을 향해 속력 (speed_missile)로 비행한다. 각 시간마다 미사일의 방향은 그 순간의 전투기의 위치와 미사일의 위치를 연결하는 벡터 방향으로 진행한다. 미사일의 위치가 전투기의 위치좌표에서 5km 안으로 들어오면 미사일이 명중한 것으로 간주하고 프로그램이 종료되는 것을 목적으로 진행한다. 높이와 위치의 단위는 km이고 속도는 km/sec로 받는다.

Init_fighter_pos = (0, 0, 100), speed_fighter = 30km/sec, init_missile_pos = (500, 0, 0) Speed_missile = 60km/sec

## 2. m-file 개발 단계 설명

### 2.1 개발 초기단계

개발 초기단계의 프로그램은 전투기 위치, 미사일 위치 등 사용자 정의 함수를 활용하여 작성했다. 예를 들어서 초기단계의 미사일 위치 함수는

```
% 미사일 위치 함수
function missilePos = getMissilePos(t, hitRange, fighterPos, initMissilePos, speedMissile)
    missilePos = initMissilePos;
    k = 1;
    timeToEnd = numel(t); % 종료시간 초기화
    while (k <= numel(t))
        direction = fighterPos(k, :) - missilePos(k, :);  % 미사일의 방향을 업데이트 (전투기와 미사일 위치 연결 벡터 방향)
        missilePos(k + 1, :) = missilePos(k, :) + (speedMissile * direction) / norm(direction); % 방향벡터를 정규화하여 미사일속도에 적용
        distance = fighterPos(k, :) - missilePos(k, :); % 미사일과 전투기의 거리

        if norm(distance) <= hitRange
            timeToEnd = k;
            break; % 거리가 hit range에 도달했을 때 loop 종료
        end

        k = k + 1;
    end

    position = ones(timeToEnd, 3);

    i = 1;
    while (i <= timeToEnd)
        position(t(i), :) = missilePos(i, :); % tiemToEnd 행까지 position에 값을 할당
        i = i + 1;
    end

    missilePos = position;
end
```

위 사진처럼 작성을 했다.

### 2.2 피드백 이유와 과정

이렇게 작성한 프로그램의 경우 미사일의 방향벡터를 구하는 것과 미사일의 위치, 명중 판정까지 한 함수에서 실행하는 동작들이 많아졌고, 불필요한 코딩이 생겼다. 그래서 피드백을 해보니 각각의 함수에서 간단하고 독립적인 동작을 하도록 하자는 의견이 나왔다. 피드백 후 초기의 미사일 위치 함수를 미사일 방향 설정 함수, 미사일 위치 함수, 명중 판정 함수로 분리하여 코드를 작성했고, 최종적으로 지금의 m-file이 만들어지게 되었다.

### 피드백 과정, 파일 흐름도

05/25 (수)
2차 회의 참고자료

백창인

## main (개선안) → 백창인

main()

입력변수:
- speed_fighter %전투기 속력, 단위: km/s
- speed_missile %미사일 속력, 단위: km/s
- init_fighter_pos %전투기 초기좌표 (x,y,z), 단위 : km
- init_missile_pos %미사일 초기좌표 (x,y,z), 단위 : km
- hit_range %명중판정 범위, 단위: km
- time_limit %요격 제한 시간, 단위:10ms

출력:
- is_hit %요격 성공 여부 (logical)

내부 변수:
-

내용:
- 입력된 변수들을 바탕으로 전투기와 미사일 항적을 계산하는 함수를 호출, 항적을 그래프로 출력하는 함수를 호출한다.

### getTrack() → 백창인

track_cell = getTrack
(time_limit, hit_range, speed_fighter, speed_missile, init_fighter_pos, init_missile_pos)

입력:
- **time_limit**
- **hit_range**
- **speed_fighter**
- **speed_missile**
- **init_fighter_pos**
- **init_missile_pos**

출력:
- **track_cell** %(1×4 cell) (1: 항적종료시간, 2: 전투기항적배열, 3: 미사일항적배열, 4: 명중여부)

내부 변수:
- **tmp_fighter_track** %전투기 항적 저장 임시 배열 [x1, y1, z1; x2,y2,z2; ...]
- **tmp_missile_track** %미사일 항적 저장 임시 배열 [x1, y1, z1; x2,y2,z2; ...]

내용:
- 시간 t의 값을 1에서 time_limit까지 증가하며 다음의 과정을 반복한다.
- 전투기의 위치를 구하여 tmp_fighter_track의 다음 행에 추가한다.
- 미사일의 위치를 결정할 방향 벡터를 구한다. (속력을 곱하여 이동거리까지 포함)
- 미사일의 위치를 구한다.
- 명중여부를 판별, 명중하면 반복을 종료한다.
- 끝나는 시간, 전투기와 미사일의 항적, 명중여부를 셀 "track_cell"에 저장하고 종료한다.

**#주요 변경점**
**변경 전:**
- 화면 출력과 무관한 기능(명중판정, 항적 종료시간 계산 등) 내장

**변경 후:**
- 화면 출력과 무관한 기능 분리
- 내부 알고리즘 간소화
- 변수명 통일성 있게 변경

백창인

### displayPlot() → 허선

displayPlot
(end_time, fighter_track, missile_track, is_hit)

입력:
- **end_time** %항적 종료 시간 (스칼라), 단위 : /10ms
- **fighter_track** %전투기 항적 행렬, (end_timex3) [x1,y1,z1; ...]
- **missile_track** %미사일 항적 행렬, (end_timex3) [x1,y1,z1; ...]
- **is_hit** %미사일 명중 여부

출력:
- **void (출력값 없음)**

내용:
- 항적 종료시간, 전투기와 미사일의 항적을 인자로 전달받는다.
- 1 : 1 : end_time (항적 종료시간)까지의 항적 행렬을 그래프 상에서 애니메이션으로 표현한다.

### getFighterPos() → 박준우

fighter_pos = getFighterPos
(t,speed_fighter,init_fighter_pos)

입력:
- **t** %시간값 (스칼라)
- **speed_fighter** %전투기속력 (스칼라)
- **init_fighter_pos** %전투기 초기좌표 [x,y,z]

출력:
- **fighter_pos** %전투기 좌표 [x,y,z]

내용:
- 시간=t 지점에서 전투기의 좌표값 (1×3)를 반환한다.

**#주요 변경점**
**변경 전:**
- 주어진 데이터로부터 t의 모든 범위에 대한 항적 계산 (비효율적)
- t값이 벡터로 주어짐

**변경 후:**
- 시간=t에서의 단일좌표[x,y,z] 계산
- 미사일 좌표 계산과의 병렬 처리를 위한 변경
- 변수명 통일성 있게 변경
- t값이 스칼라로 주어짐

백창인

### getMissileDirection() → 김선용

unit_direction = getMissileDirection
(prev_fighter_pos, prev_missile_pos)

입력:
- **prev_fighter_pos** %시간= t-1 에서의 전투기 좌표 [x,y,z]
- **prev_missile_pos** %시간= t-1 에서의 미사일 좌표 [x,y,z]

출력:
- **unit_direction** % 미사일 진행방향 단위벡터 [x,y,z]

내용:
- 시간 t-1 에서의 전투기, 미사일 좌표 (prev_fighter_pos, prev_missile_pos), 입력받아 미사일 진행방향의 단위벡터 (unit_direction) (1×3)를 반환한다.
- unit_direction은 방향을 의미하는 단위벡터이다. (getNextMissilePos() 함수에서 미사일의 다음 좌표를 구하기 위함)

**#주요 변경점**
**변경 전:**
- 미사일 좌표 계산 함수의 내장 기능

**변경 후:**
- 기존 함수로부터 기능 분리
- 변수명 통일성 있게 변경

백창인

### getNextMissilePos() → 권우석

missile_pos = getNextMissilePos
(prev_missile_pos, direction, **speed_missile**)

입력:
- **prev_missile_pos** %시간 = t-1에서 미사일 좌표 [x,y,z]
- **direction** % 미사일 진행방향 단위벡터 [x,y,z]
- **speed_missile** %미사일속력 (스칼라)

출력:
- **missile_pos** % 시간 = t 에서 미사일의 좌표값 [x,y,z]

내용:
- 시간 t-1 에서의 미사일 좌표 (prev_missile_pos), 미사일 방향 단위 벡터(direction), 미사일 속력을 입력받고,
- 시간 t에서 미사일의 좌표값 (missile_pos) (1×3)를 반환한다.

**#주요 변경점**
**변경 전:**
- 미사일 명중 시까지 미사일의 항적 행렬을 구하는 함수
- 명중 여부 판정 기능 내장
- 방향벡터 계산 기능 내장

**변경 후:**
- 미사일의 t-1에서 좌표와, 단위 벡터를 가지고 시간=t에서의 단일 좌표 [x,y,z]를 구하는 함수로 변경
- 명중 여부 판정 기능 분리 (isHit() 함수)
- 단위벡터 계산 기능 분리 (getMissileDirection() 함수)
- 변수명 통일성 있게 변경
- 함수 인자 형식 변경
- 함수명 변경

백창인

### isHit() → 백창인

is_hit = isHit
(hit_range, pos_a, pos_b)

입력:
- **hit_range**
- **pos_a**
- **pos_b**

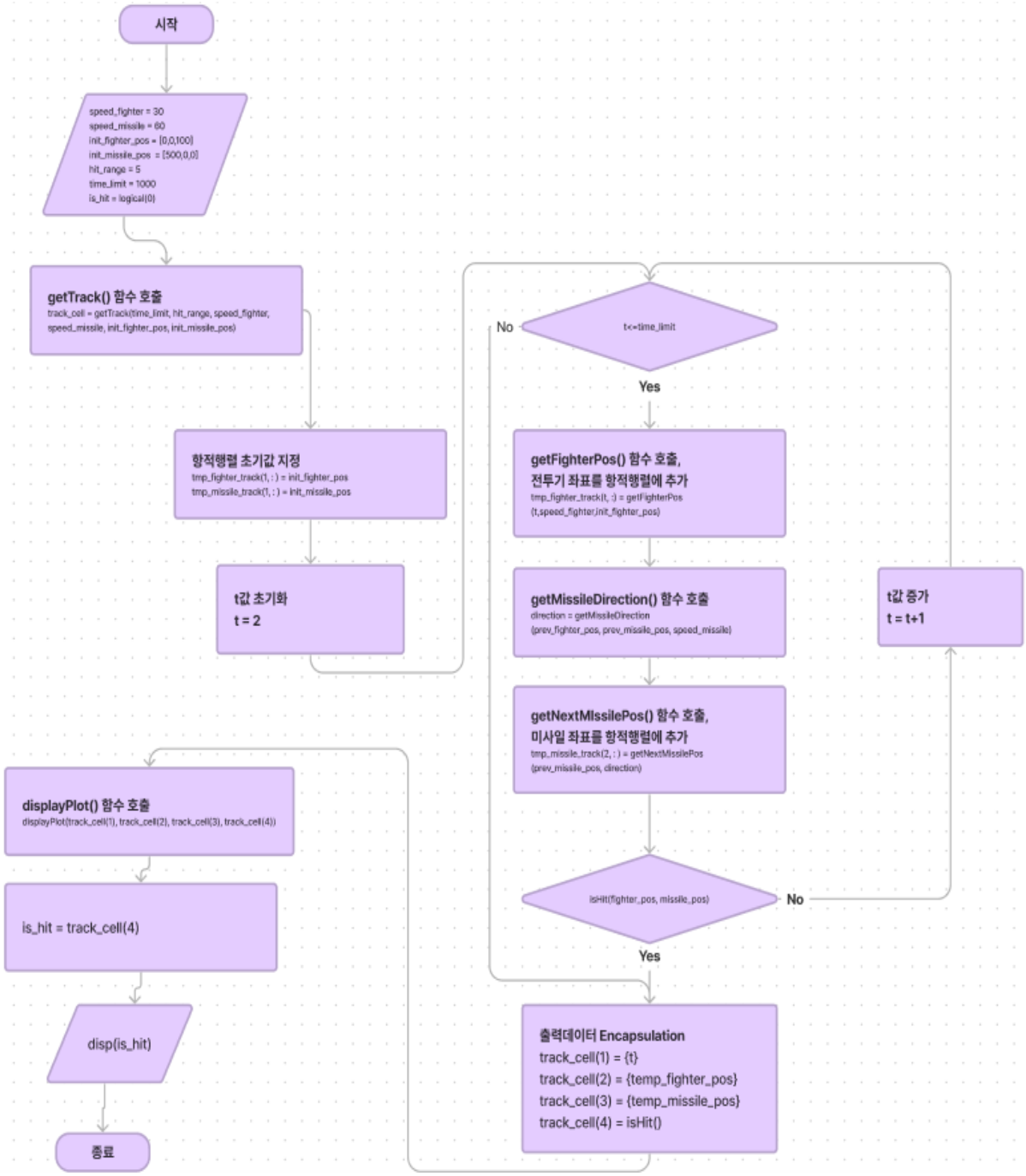
출력:
- **is_hit** %(logical 값)

내용:
- 전투기와 미사일의 위치를 (fighter_pos, missile_pos) 입력받는다.
- 명중여부를 (is_hit) 반환한다.

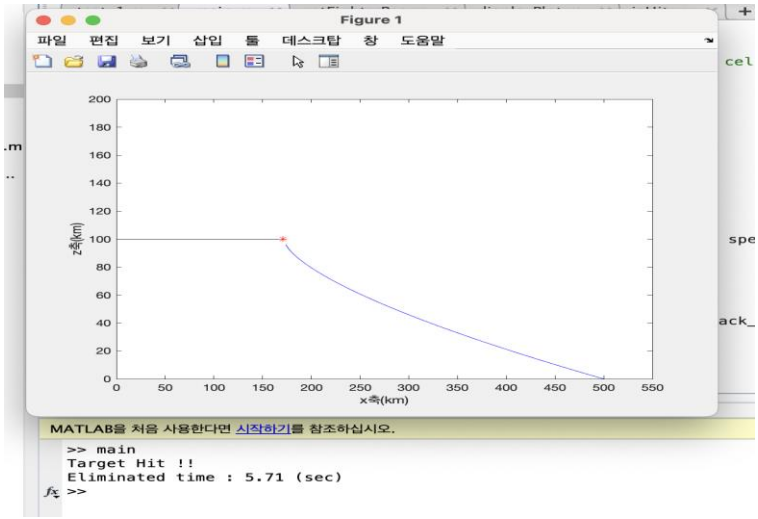
**#주요 변경점**
**변경 전:**
- 미사일 좌표 계산 함수의 내장 기능

**변경 후:**
- 기존 함수로부터 기능 분리
- 벡터 입력(항적)에도 대응 가능
- 변수명 통일성 있게 변경

백창인


시작
speed_fighter = 30
speed_missile = 60
init_fighter_pos = [0,0,100]
init_missile_pos = [500,0,0]
hit_range = 5
time_limit = 1000
is_hit = logical(0)
getTrack() 함수 호출
track_cell = getTrack(time_limit, hit_range, speed_fighter, speed_missile, init_fighter_pos, init_missile_pos)
항적행렬 초기값 지정
tmp_fighter_track(1, : ) = init_fighter_pos
tmp_missile_track(1, : ) = init_missile_pos
t값 초기화
t = 2
No
t<=time_limit
Yes
getFighterPos() 함수 호출,
전투기 좌표를 항적행렬에 추가
tmp_fighter_track(t, :) = getFighterPos
(t,speed_fighter,init_fighter_pos)
getMissileDirection() 함수 호출
direction = getMissileDirection
(prev_fighter_pos, prev_missile_pos, speed_missile)
getNextMIssilePos() 함수 호출,
미사일 좌표를 항적행렬에 추가
tmp_missile_track(2, : ) = getNextMissilePos
(prev_missile_pos, direction)
isHit(fighter_pos, missile_pos)
No
Yes
출력데이터 Encapsulation
track_cell(1) = {t}
track_cell(2) = {temp_fighter_pos}
track_cell(3) = {temp_missile_pos}
track_cell(4) = isHit()
t값 증가
t = t+1
displayPlot() 함수 호출
displayPlot(track_cell(1), track_cell(2), track_cell(3), track_cell(4))
is_hit = track_cell(4)
disp(is_hit)
종료

### 2.3 최종 m-file 작동




검은색 실선이 전투기, 푸른색 실선이 미사일을 나타내며, 미사일의 방향벡터가 계속 바뀌며 나아가 최종적으로 전투기를 타격하는 것을 확인이 가능하다.

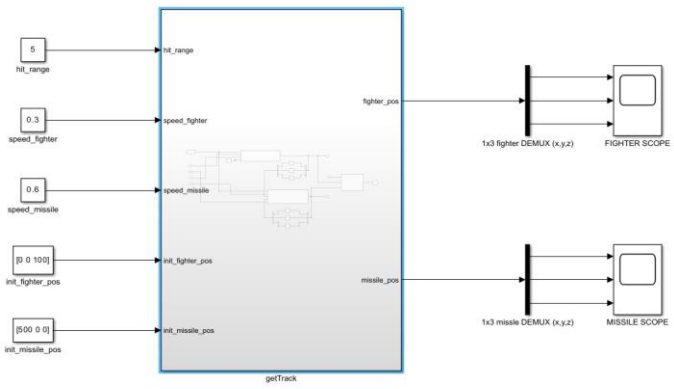
## 3. Simulink 개발 단계 설명

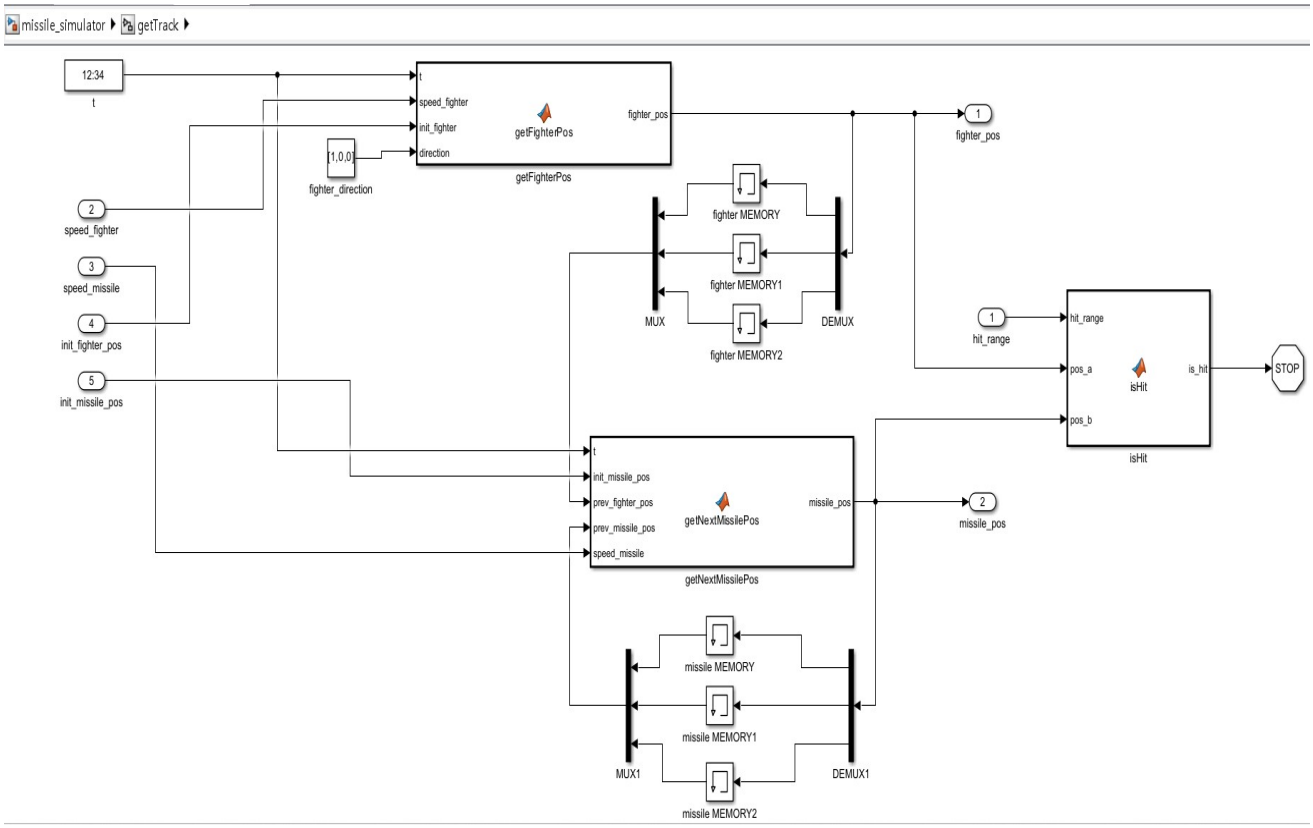
### 3.1 m-file과 simulink 간의 연동 과정

개발 초기에는 m-file 그대로 simulink의 matlab function block을 활용하여 작동시켰다. 하지만 각 함수끼리의 연동 과정이나 중간의 코드에서 문제를 일으키는 부분이 있어 simulink 상에서 다른 블록들을 활용하여 코드를 수정했다.

### 3.2 simulink에 최적화시킨 코드

그렇게 m-file을 simulink에 최적화 시킨 코드는



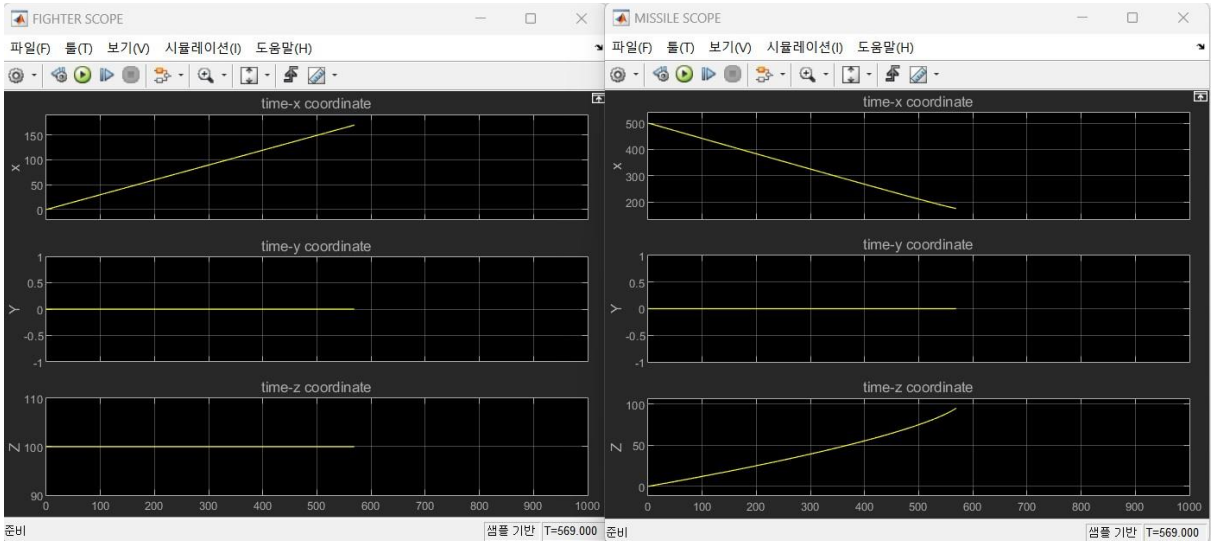



위 사진과 같이 subsystem으로 구현을 했고, 메모리는 미사일의 진행방향을 결정하는 요소 (t-1에서의 전투기와 미사일의 좌표)를 서브시스템에 전달하기 위해서 사용했다. 또한 원래의 코드에서는 정해진 행렬을 받는게 아닌 축방향 단위행렬을 곱해주는 형태의 코드를 활용을 했었다. 하지만 오류가 생겨 direction이라는 행렬을 따로 input으로 활용을 하여 오류를 제거했다.

### 3.3 각 모듈별 기능과 작동 원리

파일을 실행시키게 되면 먼저 getTrack 함수를 호출한다. 그러면 설정해 놓은 초기값을 기준으로 getFighterPos 함수에서 전투기의 시간 t에서의 위치를 구한다. 메모리를 활용해 시간 t-1에서의 전투기의 좌표를 getMissilePos 함수에 전달하고, 미사일의 방향벡터를 구하여 그 방향으로 미사일을 단위시간 * 미사일의 속도만큼 이동시킨다. 그 결과값이 Is hit 함수로 이동하여 충돌판정이 일어나면 stop, 아니면 동일한 동작을 반복한다. 이러한 결과값들이 scope에 보여지게 되는데, 관찰하기 쉽게 만들기 위해 demux를 활용하여 각각의 좌표값을 분리시켜 scope에 나타냈다.

### 3.4 최종 simulink 작동 그래프



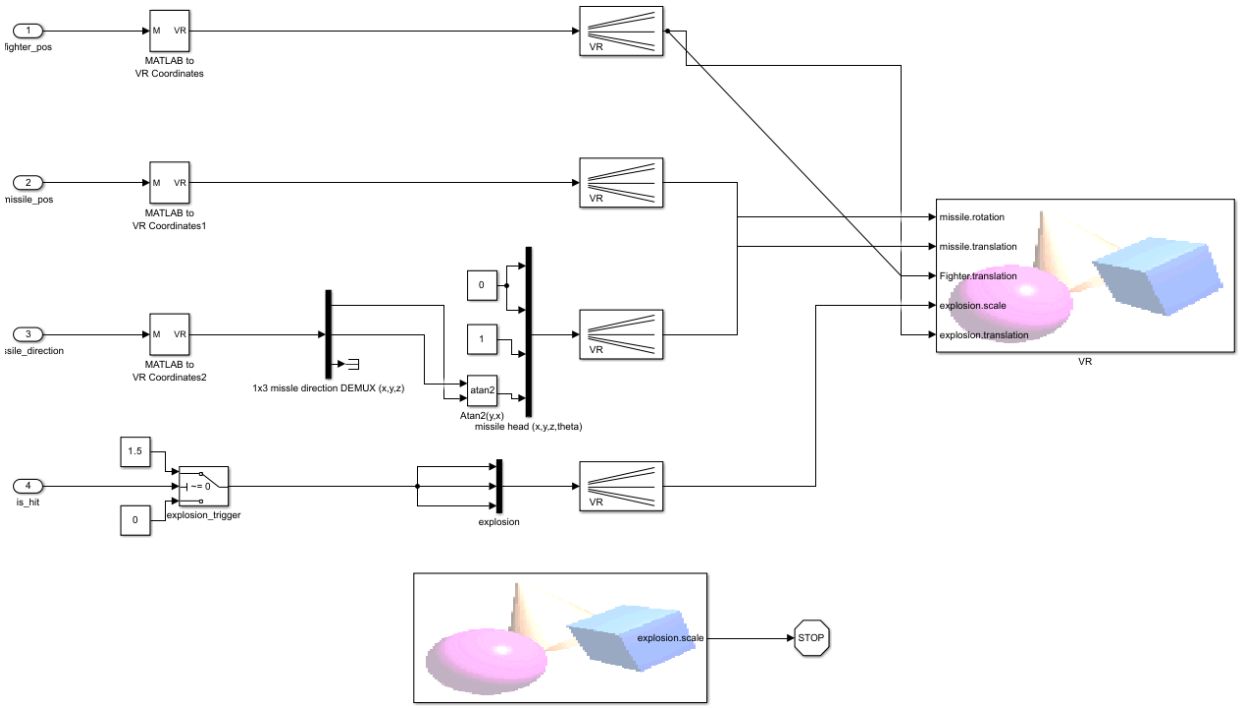

위 그래프로 알 수 있듯이 전투기는 x축 방향으로 일정하게 속도가 증가하며 진행하는 것을 볼 수 있고, 미사일은 -x축 방향으로 일정한 속력으로 진행하고, z축 방향으로 방향벡터가 계속 바뀌며 진행하는 것을 확인할 수 있다.

## 4. 3D animation (VR) 개발 단계 설명

### 4.1 simulink에서 3d animation으로의 연결 과정

m-file, simulink에서는 우리가 평소에 사용하던 x, y, z축을 순서대로 사용한다면, VR 좌표계에서는 z축과 y축이 바뀐 좌표계를 사용하고 있었기 때문에 좌표값을 바꿔줄 필요가 있었다. 그래서 미사일 위치, 미사일 방향벡터, 전투기 위치를 MATLAB to VR coordinates 함수를 활용하여 좌표계를 바꿔 주었다. Missile_direction의 경우는 라이브러리에서 불러온 미사일의 머리 방향의 기본값을 전투기의 방향으로 바꿔주기 위해서 atan2 함수를 사용했다.

이렇게 연결시킨 최종 코드는

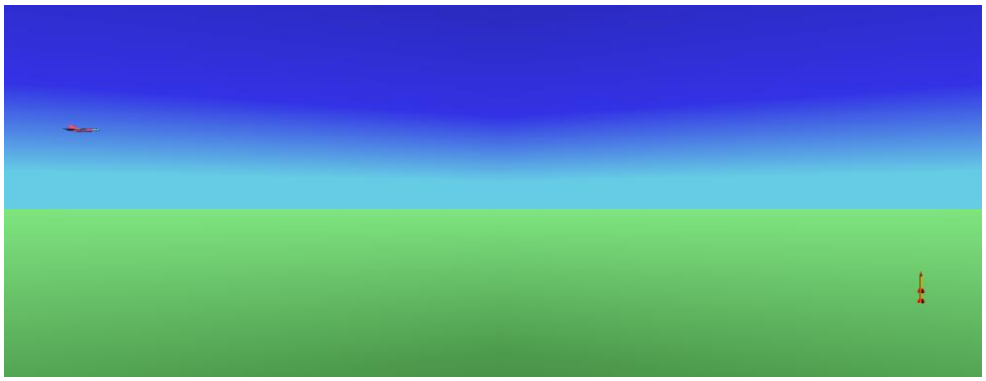


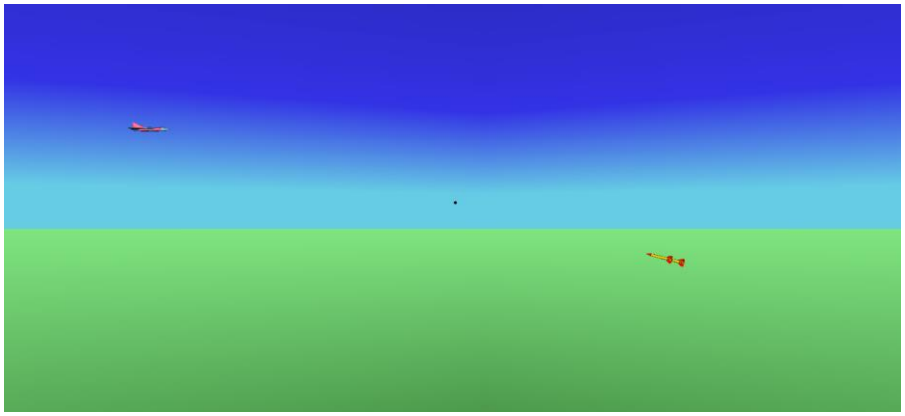

보여지는 위 사진은 visualization 모듈인데, 앞 장의 simulink의 getTrack 모듈에서 들어온 정보들을 바탕으로 좌표계 변환을 하고, VR sink에 신호와 연동시켜주는 block을 사용하여 VR에 나타낼 수 있도록 하였다. 또한 기체의 폭발 조건은 스위치를 활용하여 is_hit 함수가 1을 출력하게 되면 폭발 모델링의 크기가 [0, 0, 0] 에서 [1.5, 1.5, 1.5] 로 변화해 폭발 효과가 나타나도록 설계하였다.

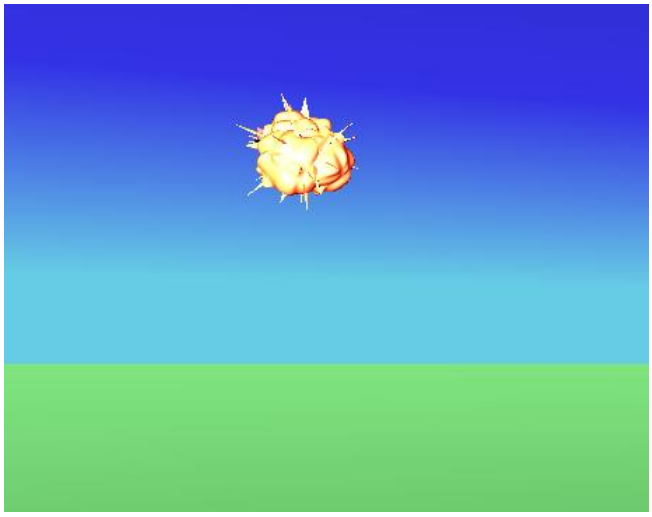
### 4.2 VR sink, VR editor

VR로 프로그램을 보이기 위해서는 VR world를 제작해야 했다. 그래서 VR editor에서 전투기와 미사일, background는 라이브러리에 있는 모형들을 불러와 나타내었다. 또한 전투기와 미사일이 날아가는 모습을 보다 정확하게 관찰하기 위해 viewpoint의 좌표값을 조정하였다. 각 모형들의 이름을 설정하고 block parameter를 통해 VR editor와 simulink의 값들을 연결하여 최종적으로 VR sink에서 전투기를 격추하는 미사일을 관찰할 수 있게 하였다.

### 4.3 VR 최종 결과







이렇게 미사일이 전투기에 일정 범위안에 접근하여 타격 판정 시 기체가 폭발하는 모습을 관찰할 수 있다.

### 5.1 참고 자료 및 문헌

Simulink 3D VR : https://youtu.be/FUitbiQ2XG4

Simulink user-defined-functions :

https://kr.mathworks.com/help/simulink/user-defined-functions.html

axis-angle representation : https://en.wikipedia.org/wiki/Axis%E2%80%93angle_representation

sf rotation :

https://kr.mathworks.com/matlabcentral/answers/319954-how-does-sfrotation-works-in-simulink-3d-world-editor

Matlab to VR coordinates : https://kr.mathworks.com/help/sl3d/matlabtovrcoordinates.html

3D world editor : https://kr.mathworks.com/help/sl3d/build-virtual-reality-worlds.html

connect a virtual world : https://kr.mathworks.com/help/sl3d/example-of-building-a-virtual-world-and-connecting-it-to-a-simulink-model.html