[IFAC2023]

MATHWORKS MINIDRONE COMPETITION

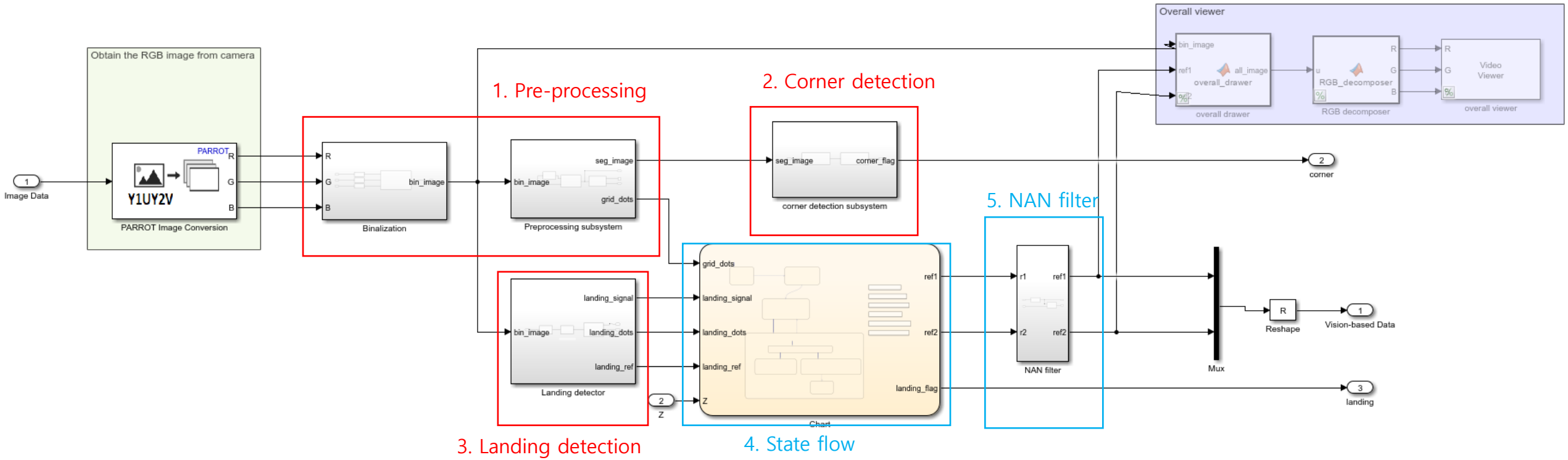
서울시립대학교 제어 및 동역학 연구실

(김인겸 변순석 박원영 김준수)

2023년 05월 03일

목차

Overview


Obtain the RGB image from camera
1
Image Data
PARROT
Y1UY2V
R
G
B
PARROT Image Conversion
Binalization
bin_image
1. Pre-processing
seg_image
grid_dots
Preprocessing subsystem
2. Corner detection
corner_flag
corner detection subsystem
2
corner
Overall viewer
all_image
overall_drawer
overall drawer
RGB_decomposer
RGB decomposer
Video Viewer
overall viewer
5. NAN filter
landing_signal
landing_dots
landing_ref
Landing detector
3. Landing detection
2
Z
ref1
ref2
landing_flag
Chart
4. State flow
r1
r2
NAN filter
Mux
Reshape
1
Vision-based Data
3
landing

1. pre-processing : Binarization

```
function bin_image = binalizer(R, G, B, VisualParam)
R_THRESHOLD = VisualParam.R_THRESHOLD;

bin_image = zeros(120,160);
for i=1:120
        for j=1:160
        if R(i,j)-G(i,j)/2-B(i,j)/2>R_THRESHOLD
                bin_image(i,j)=1;
        end
        end
end
```

: R_THRESHOLD = 100; 일 때 빨간선이 잘 인식되는 것을 확인함.

Input : RGB_image / Output : bin_image

1. pre-processing : Image segment

- Image_to_dots

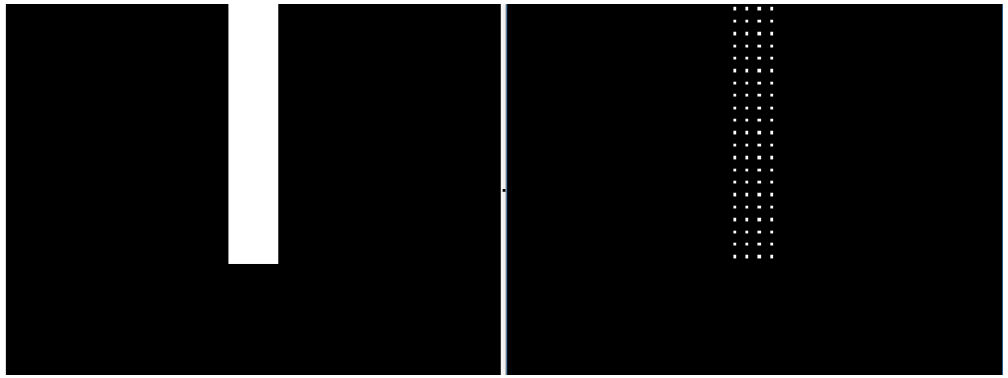
Input : bin_image / Output : dots (x,y)

- Image_segment

Input : bin_image, dots / Output : seg_image

```
function seg_image = image_segment(dots, bin_image, VisualParam)

N = 120*160;
CX = VisualParam.CX;
CY = VisualParam.CY;

min_dist = Inf;
center = [CX, CY];
if ~bin_image(CX, CY)          % 드론과 가장 가까이 있는 pixel 찾기
        for i = 1:N
                if dots(i,1) == 0
                        break;
                end

                dist = norm( dots(i,:) - [CX, CY]);
                if dist < min_dist
                        center = dots(i,:);
                        min_dist = dist;
                end
        end
end

L_image = bwlabel(bin_image);          % image 조각마다 labeling

label = L_image(center(1), center(2));          % 드론이 위치한 image 조각만 남기기
seg_image = L_image==label;
```

1. pre-processing : Image Gridization



Input : seg_image / Output : grid_image, grid_dots

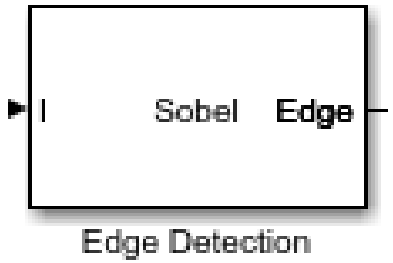
->점의 개수를 줄여 연산량을 줄인다.

```
function [grid_image, grid_dots] = gridization(seg_image)

L = 4; % length of grid side
threshold = 10;
n = (120/L) * (160/L);
grid_image = zeros(120,160);
grid_dots = zeros(n,2);
idx = 0;

for i=0:(120-L)/L
        for j=0:(160-L)/L
                if sum(seg_image(L*i+1:L*i+L, L*j+1:L*j+L),'all') > threshold
                        grid_image(L*i+(L/2),L*j+(L/2))=1;
                        idx = idx+1;
                        grid_dots(idx,1) = L*i+(L/2);
                        grid_dots(idx,2) = L*j+(L/2);
                end
        end
end
```

2. Corner detection : Edge detection

- edge detection (Simulink block) : Sobel method

Input : seg_image / Output : edge_image



2. Corner detection : Straight line detection

- Hough transform : edge_image의 점(x,y)들을 (rho,theta)로 변환하여 표현하고, 변환된 점들을 이용하여 straight_line을 찾는다.

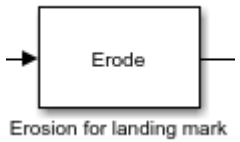
Input : edge_image / Output : corner_flag

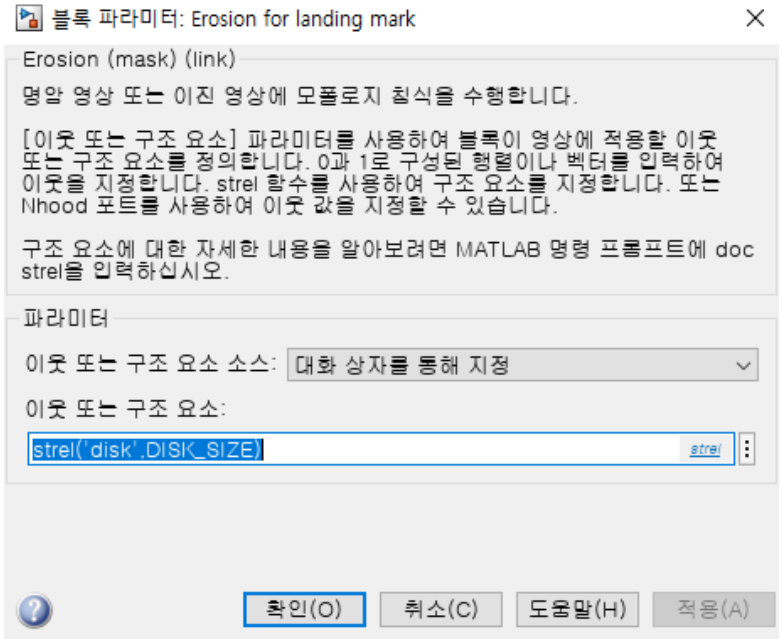
-> straight_line이 3개 이상이면 corner_flag = 1;

```
function corner_flag = hough_trans(edge_image,VisualParam)

[H,T,R] = hough(edge_image);
P = houghpeaks(H,4,'Threshold',VisualParam.HOUGH*max(H(☺),'NHoodSize',[15 15]);
Theta = zeros(4,1);
Rho = zeros(4,1);
n = size(P,1); % number of different lines
for i=1:n
        Rho(i) = R(P(i,1));
        Theta(i) = deg2rad(T(P(i,2)));
end

if n>=3
                corner_flag = 1;
else
                corner_flag = 0;
end
```

3. Landing detection : Detect disk shape

- Erosion (Simulink block) : 드론 높이 1.3m -> DISK_SIZE = 12일 때 landing mark를 잘 인식함.





3. Landing detection : Resize to 120x120 and check landing

- ROI : Rectangular of Image (120x160) -> (120x120)

- landing_check

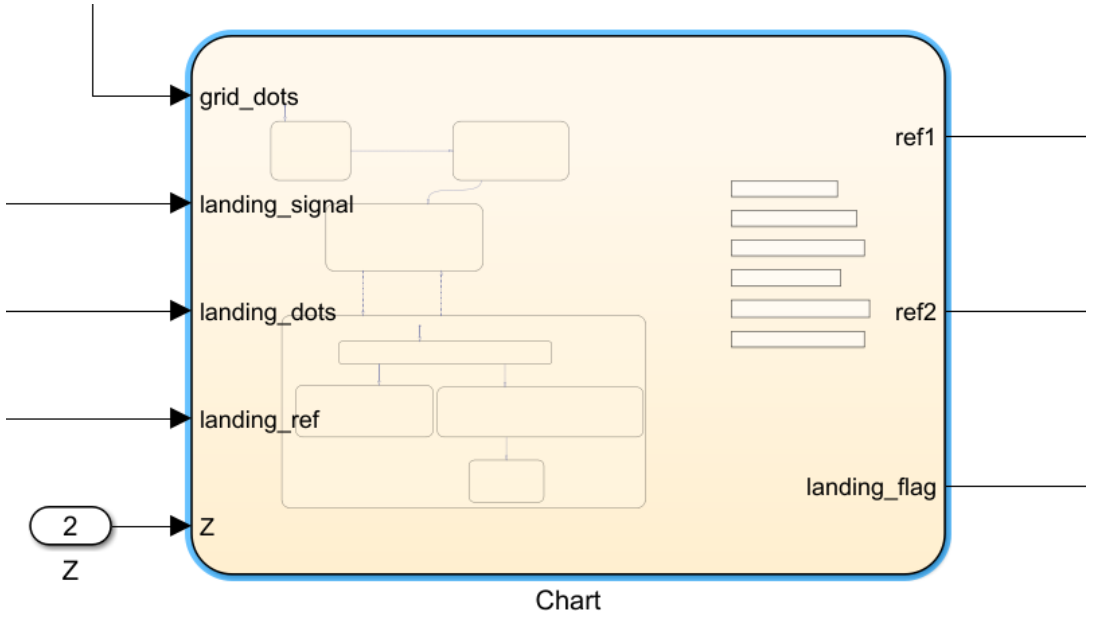
Input : ROI / Output : landing_signal, landing_dots (Erosion으로 찍힌 점들), landing_ref (landing_dots의 평균)

->Erosion으로 찍힌 점이 하나라도 있다면 landing_signal = 1;

4. State flow – input output

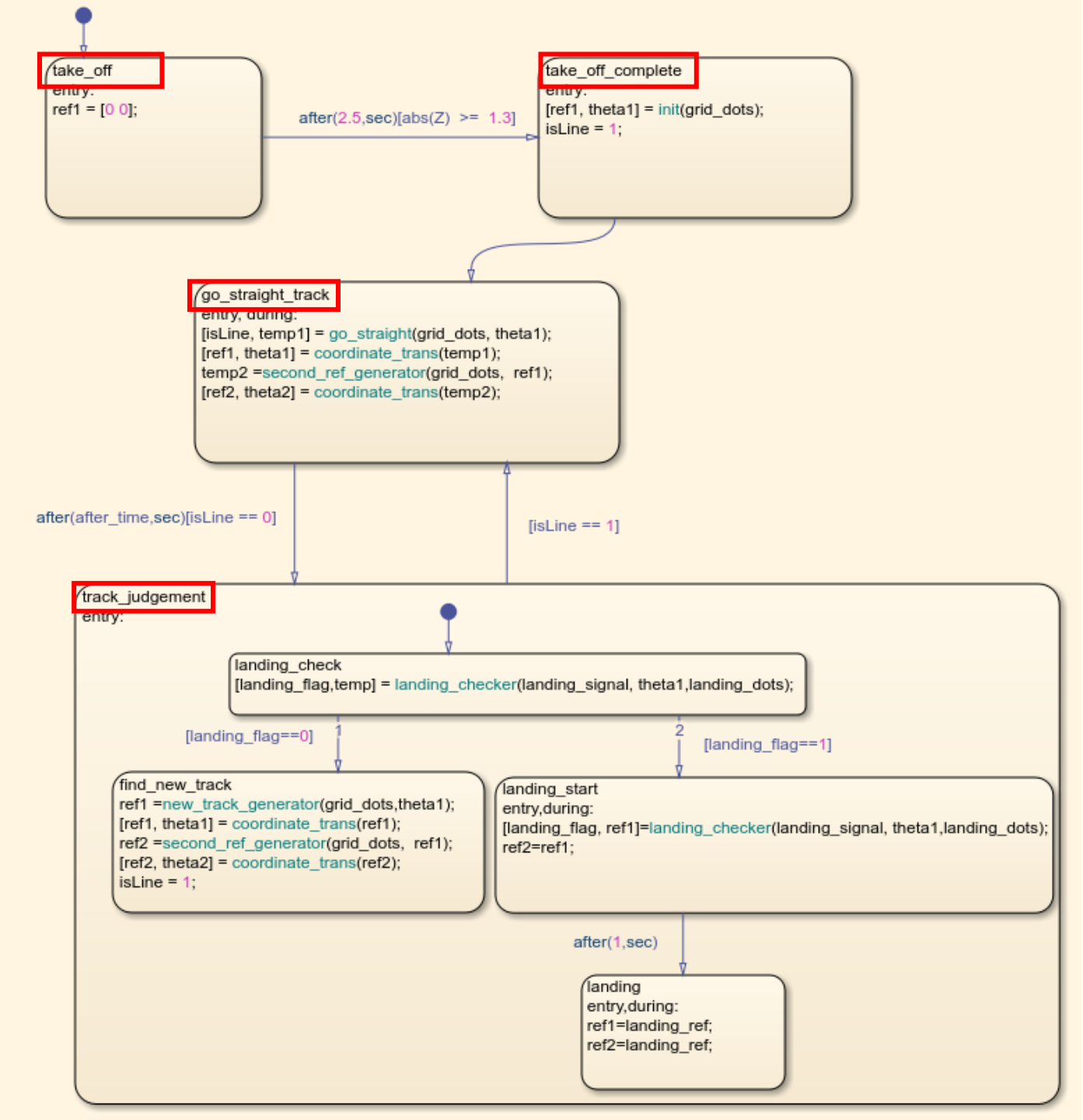
- Input
  1. grid dots : 전처리 과정을 거친 이미지의 좌표들을 n x 2 행렬로 표현한 데이터
  2. landing_signal : 랜딩인지 아닌지를 판단하는 0또는 1의 값
  3. landing_dots : erode된 점들의 좌표를 n x 2 행렬로 표현한 데이터
  4. landing_ref : 랜딩 좌표
  5. Z : 센서 데이터로부터 받아온 현재 드론의 Z값

- Output
  1. ref1 : Chart에서 계산된 드론의 첫번째 Reference 값
  2. ref2 : Chart에서 계산된 드론의 두번째 Reference 값

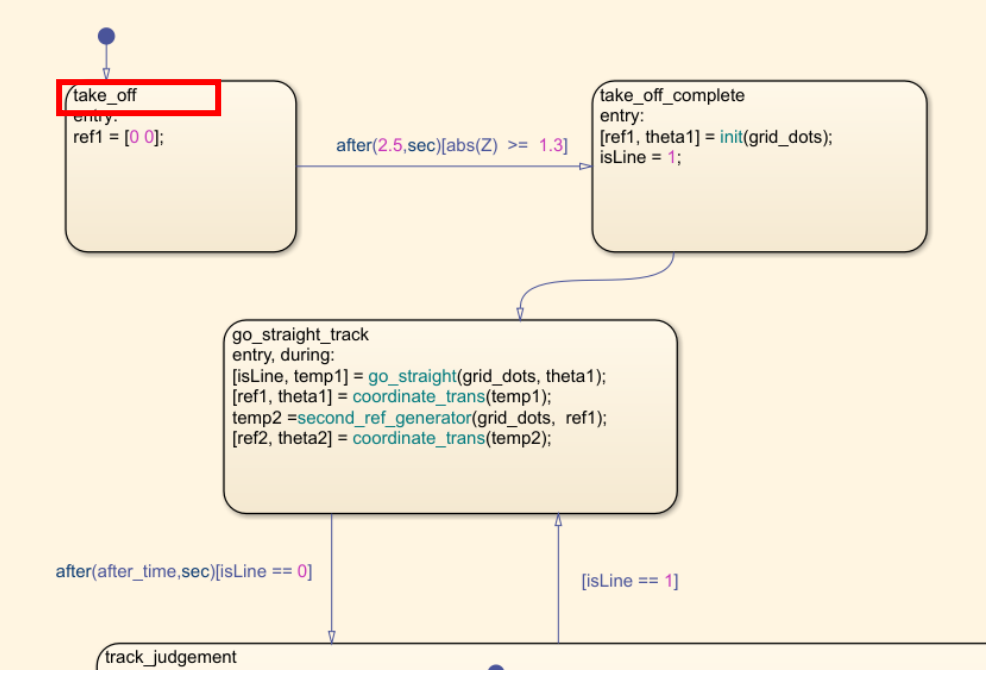


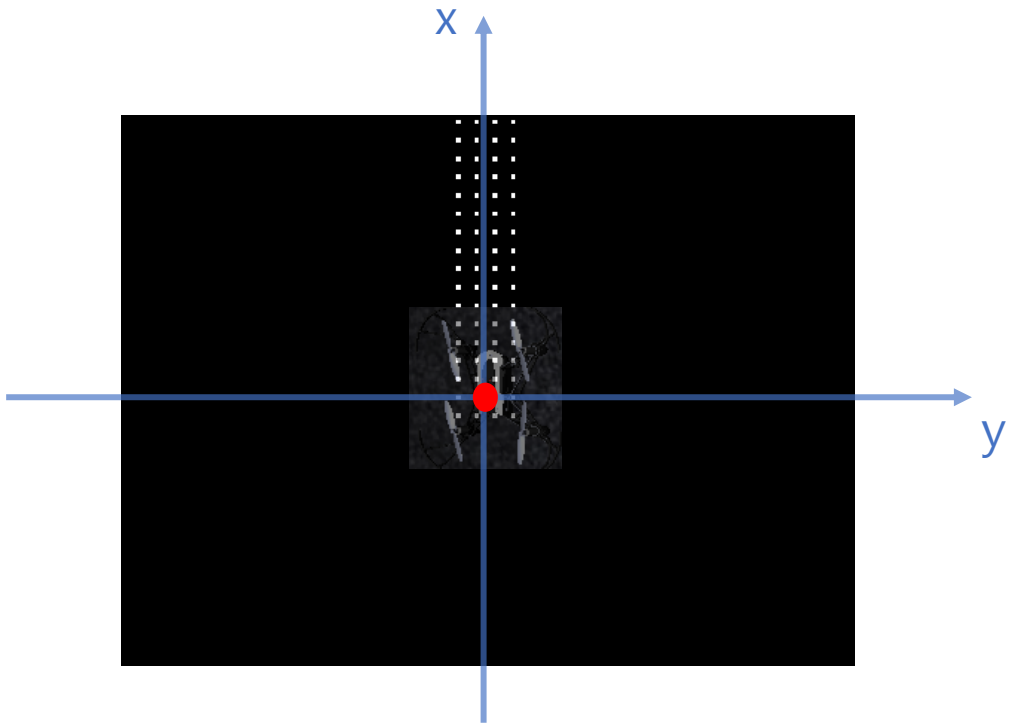
4. State flow

- Take_off : x,y좌표를 고정함으로써 드론이 안정적으로 이륙할 때까지 기다린다.
- Take_off_complete : 드론이 이륙을 완료하면 init함수를 통해 드론의 Reference를 생성한다.
- Go_straight_track : 드론의 진행 방향과 라인이 이루는 각이 0도에 가깝다면 이전 step에서 계산된 theta를 활용하여 reference를 생성한다.
- Track_judgement : 드론의 진행 방향과 라인이 이루는 각이 90도 이상일 때 lading인지 확인한 뒤 새로운 reference를 생성한다.



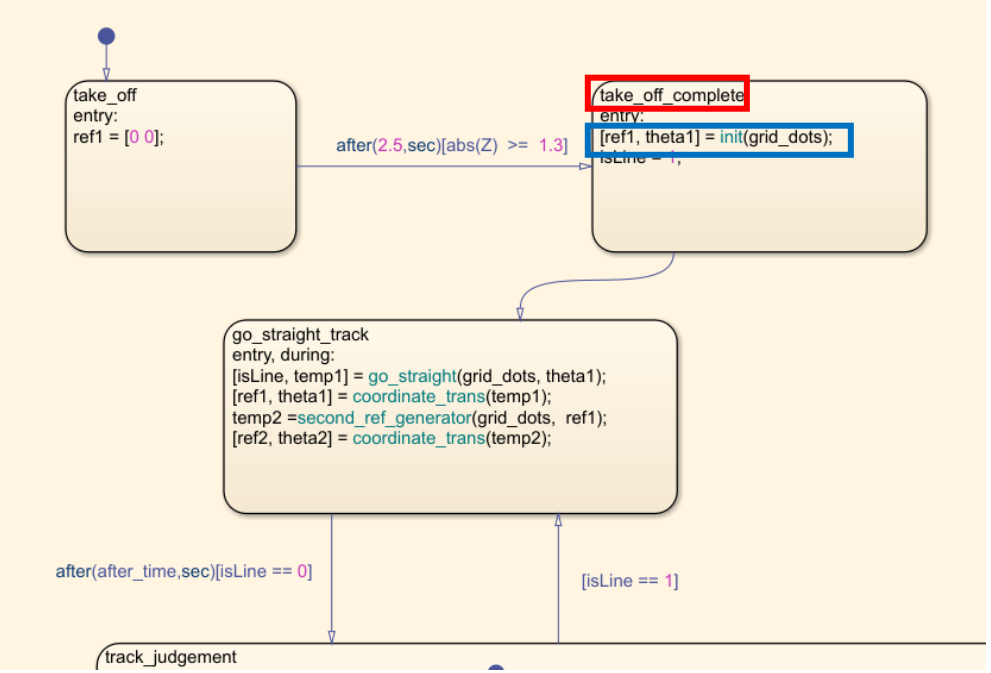
4. State flow – take_off

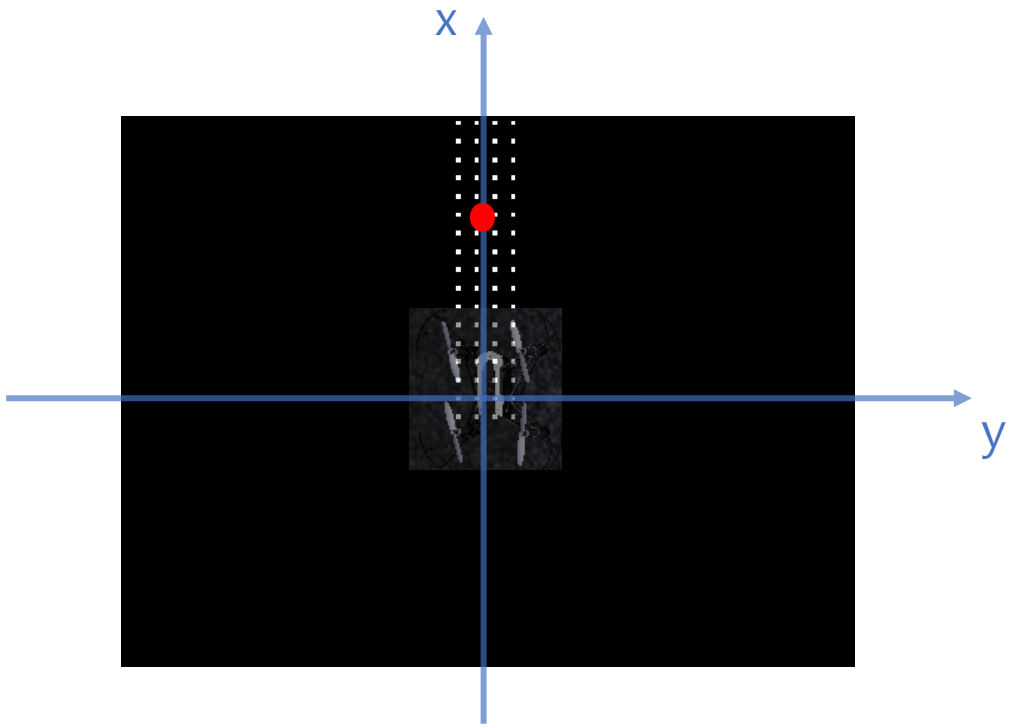




- Z값이 1.3이상이 될 때까지 reference를 원점으로 고정한다
- 초반 불안정성을 해결하기 위해 2.5초 동안은 reference를 원점으로 고정시켰다.

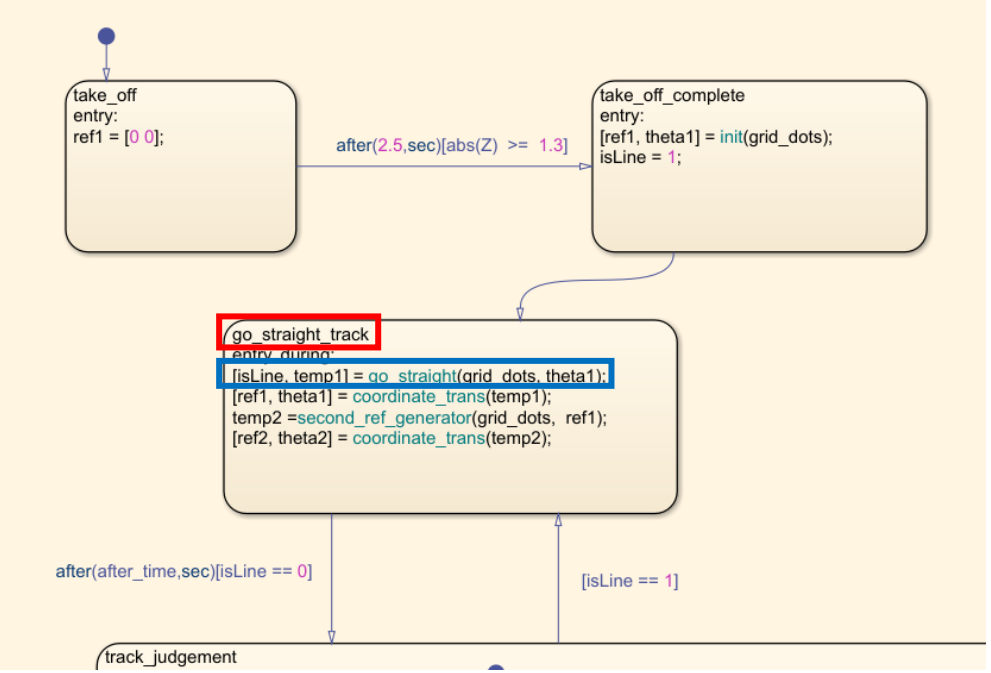
4. State flow – take_off_complete



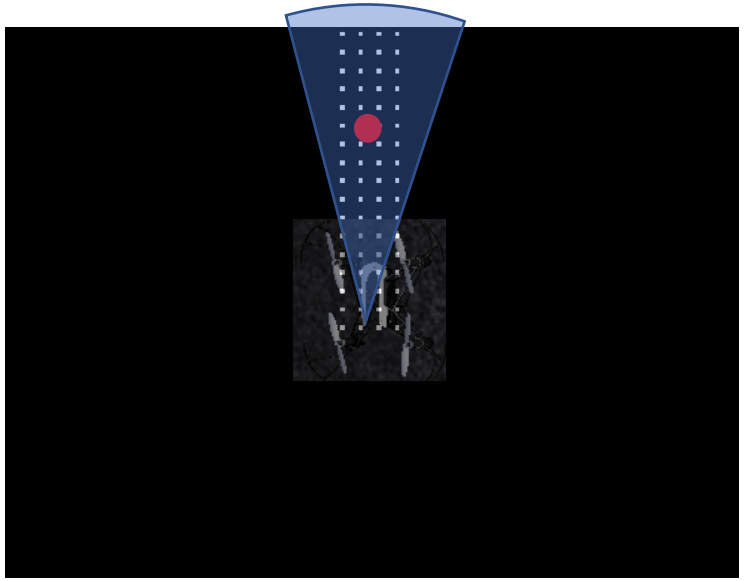


- Init : 각 점들의 평균을 취하여 첫번째 reference를 생성한다.
- theta1은 reference와 드론의 x축과 이루는 각을 의미한다.(지금 상황에서 theta1 값은 0이다)

4. State flow – go_straight_track

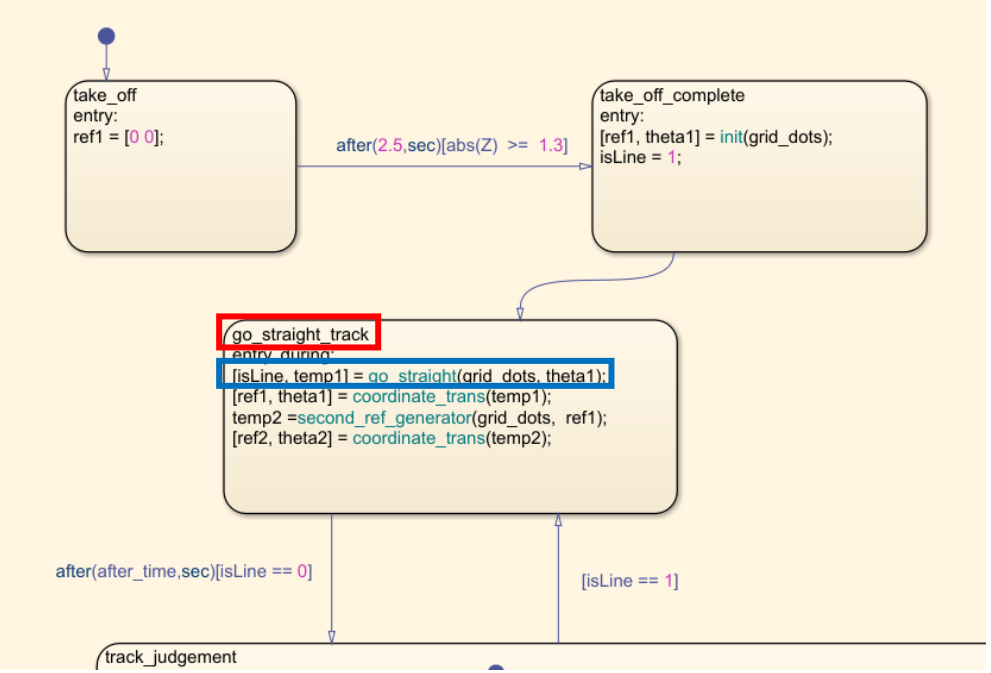



```
%% go_straight (기존 sector_form)
after_time = 0.5;
VisualParam.dots_threshold = 8; % 기존 num_threshold
VisualParam.angle_of_view1= 30; % 기존 degree_range1
```

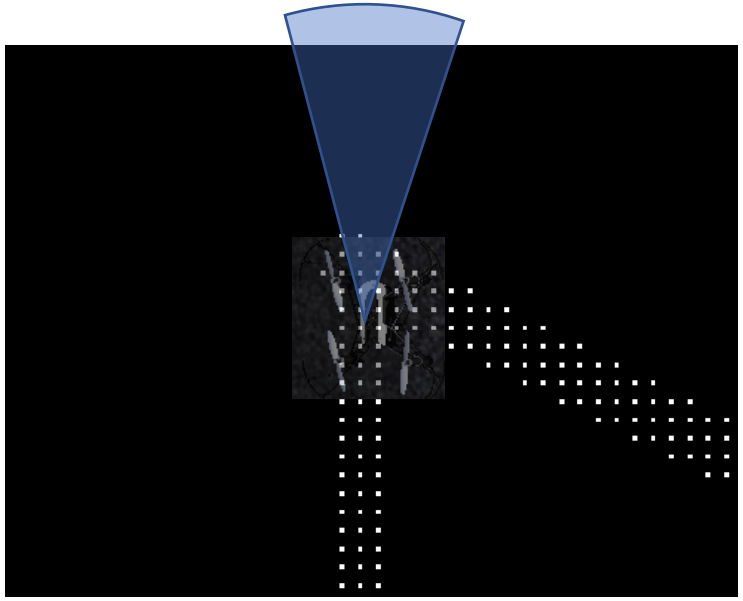


- go_straight : 이전에 생성한 theta1값을 중심으로 angle_of_view1에 해당하는 각도만큼 부채꼴 영역을 생성한다. 부채꼴 영역 내부의 점들의 평균으로 새로운 referenc를 생성한다.

4. State flow – go_straight_track

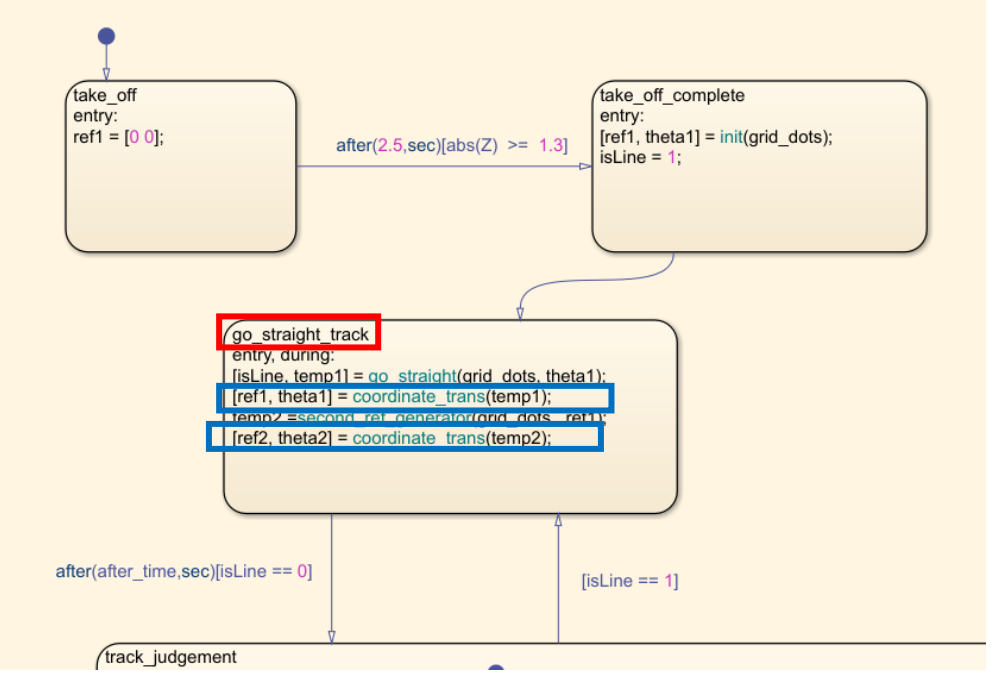



```
%% go_straight (기존 sector_form)
after_time = 0.5;
VisualParam.dots_threshold = 8; % 기존 num_threshold
VisualParam.angle_of_view1= 30; % 기존 degree_range1
```

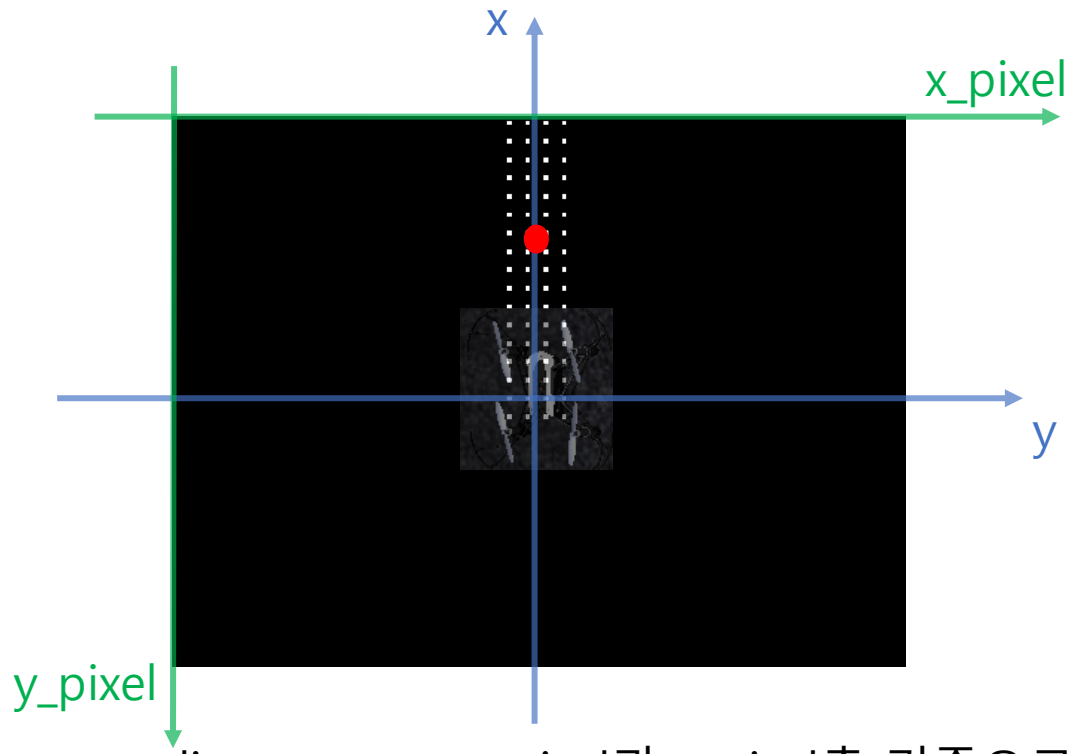


- go_straight : 위 그림과 같이 부채꼴 내부의 점들의 개수가 dots_threshold의 개수보다 작으면 isLine변수값이 0이되고 track_judgement로 상태가 바뀐다
- after_time: state가 바뀌는 직후에 불안정한 상태를 막기 위해 0.5초 동안은 go_straight_track을 수행하도록 하였다.

4. State flow – go_straight_track

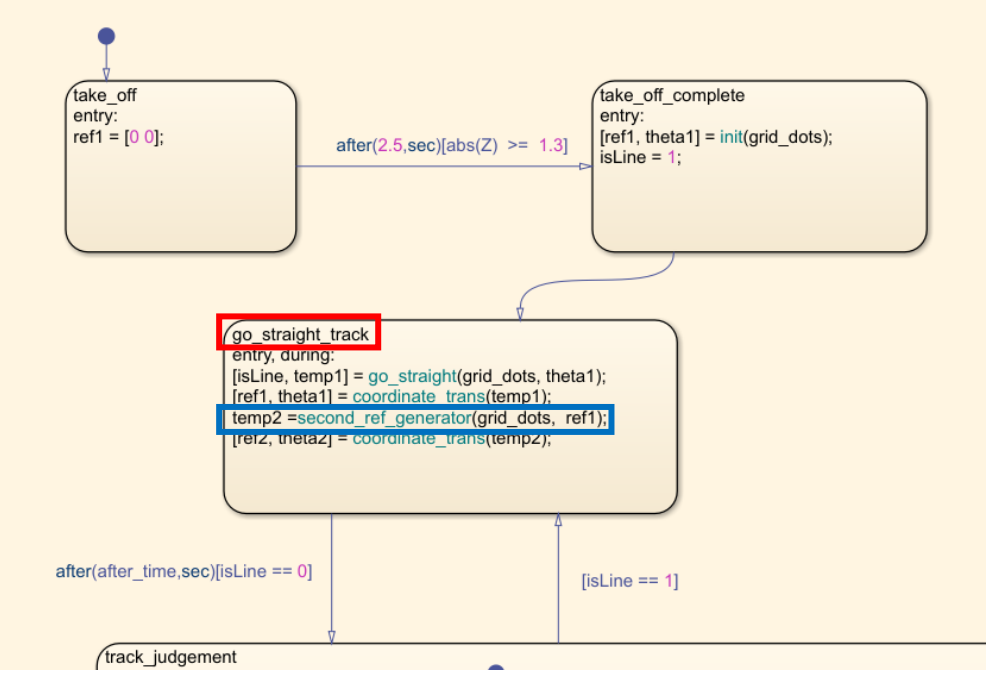



```
%% go_straight (기존 sector_form)
after_time = 0.5;
VisualParam.dots_threshold = 8; % 기존 num_threshold
VisualParam.angle_of_view1= 30; % 기존 degree_range1
```

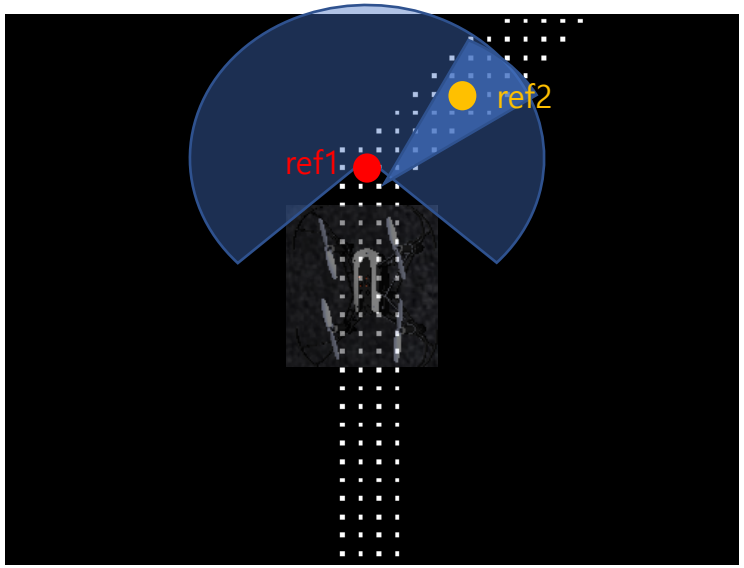


- coordinate_trans : x_pixel과 y_pixel축 기준으로 생성된 reference의 값을 드론의 좌표축인 x와 y축으로 바꿔주는 역할을 한다.

4. State flow – go_straight_track

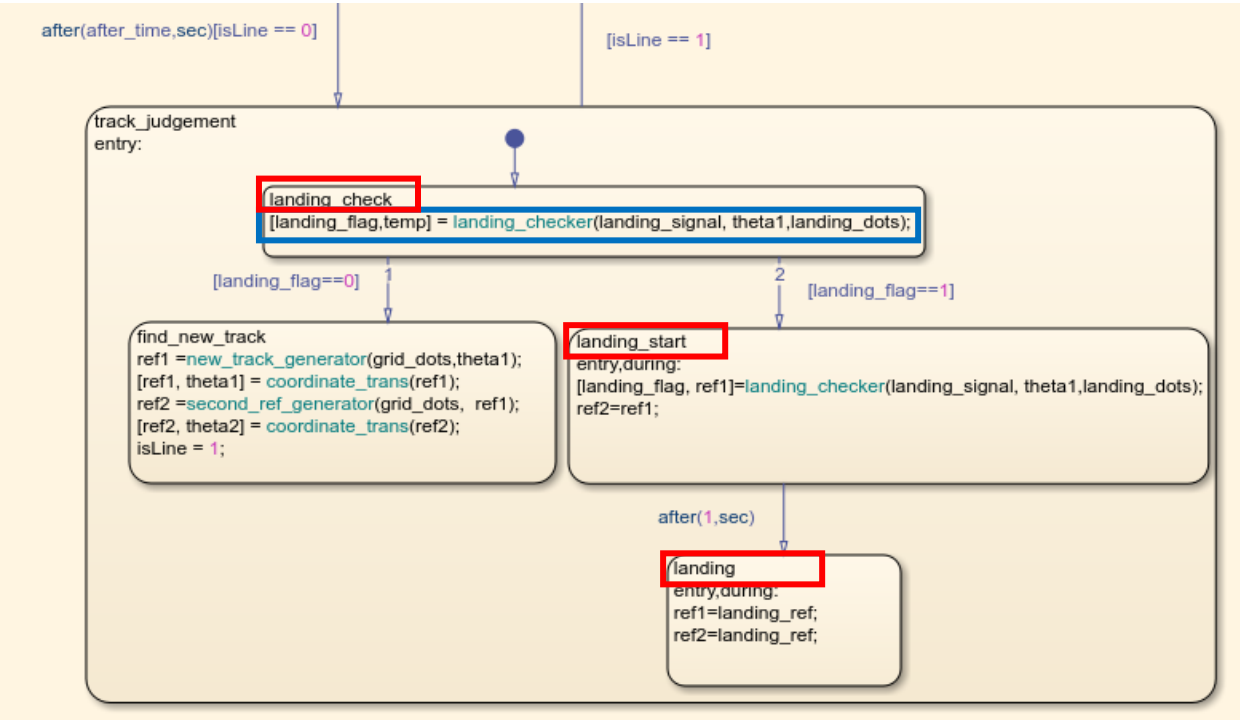



```
%% go_straight (기존 sector_form)
after_time = 0.5;
VisualParam.dots_threshold = 8; % 기존 num_threshold
VisualParam.angle_of_view1= 30; % 기존 degree_range1
```



- second_ref_generator : 첫번째 reference에서 드론 방향의 각 erased_angle2만큼을 제외한 영역을 검사한다.
- angle_of_view2각도를 angle_of_rotation만큼 돌려가면서 부채꼴 영역 내부의 점이 가장많은 경우를 찾는다.
- 점이 가장 많은 부채꼴 영역의 평균값으로 두번째 reference값을 구한다.
- 이때 radius2이내의 점들만 검사한다.

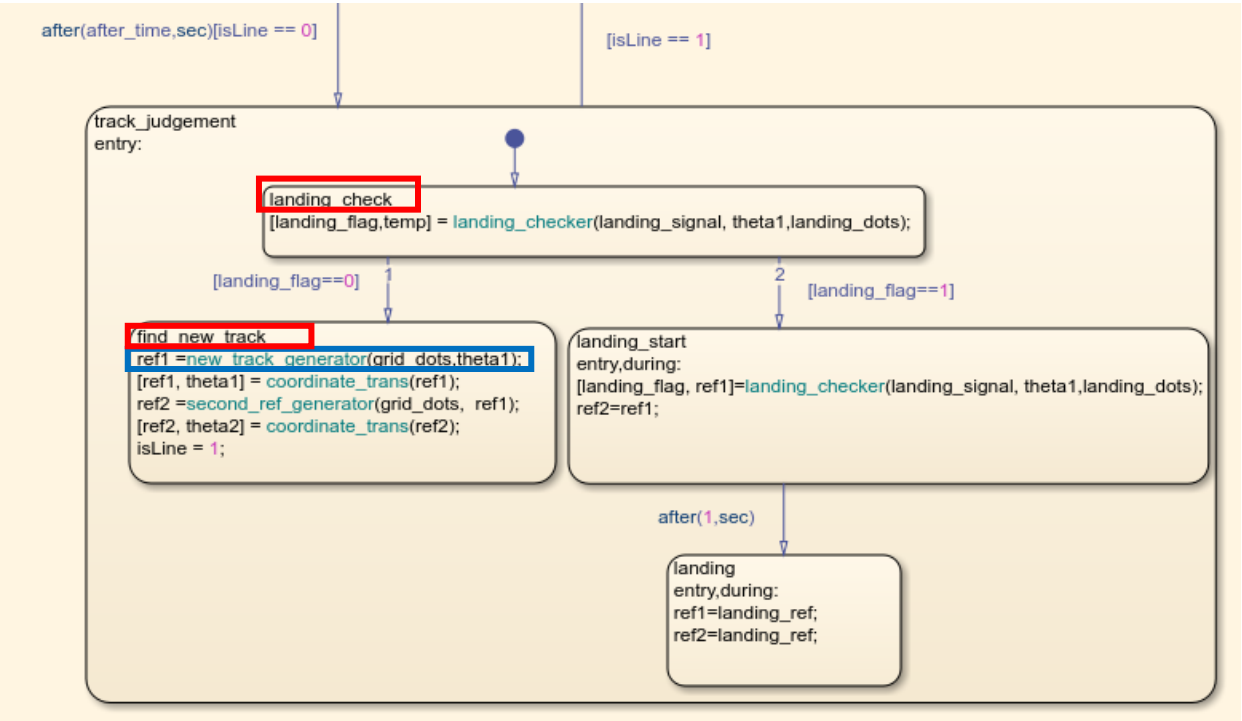
4. State flow – track_judgement

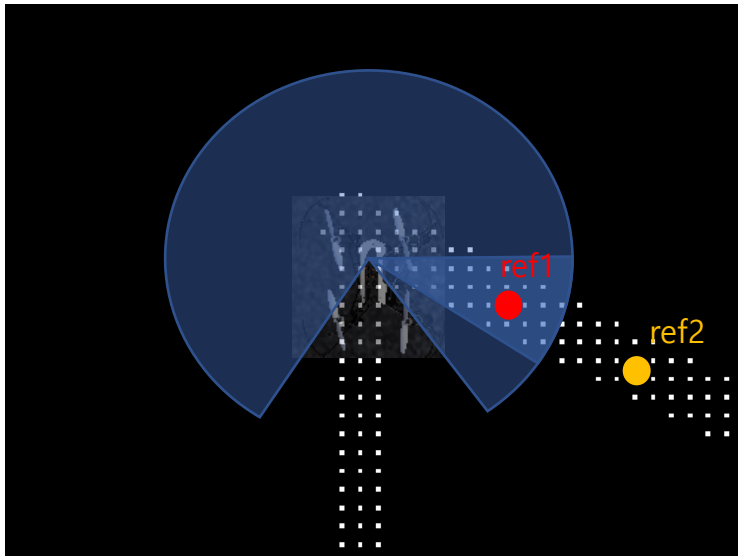


```
%% Landing checker
VisualParam.landing_threshold = 10;
```

- landing_checker : 드론의 진행방향으로 90도 만큼의 부채꼴 영역에서 erode된 점들의 개수가 landing_threshhold보다 크면 landing_start상태로 이동한다.
- landing 시에 드론의 흔들림으로 인해 reference가 사라지거나 erode가 제대로 되지 않는 문제점이 있는데, 이를 해결하기 위해 1초 동안은 지속적으로 landing을 위한 reference를 생성한다.
- 1초가 지나면 Landing detector subsystem에서 계산한lading_ref를 출력한다.

4. State flow – track_judgement

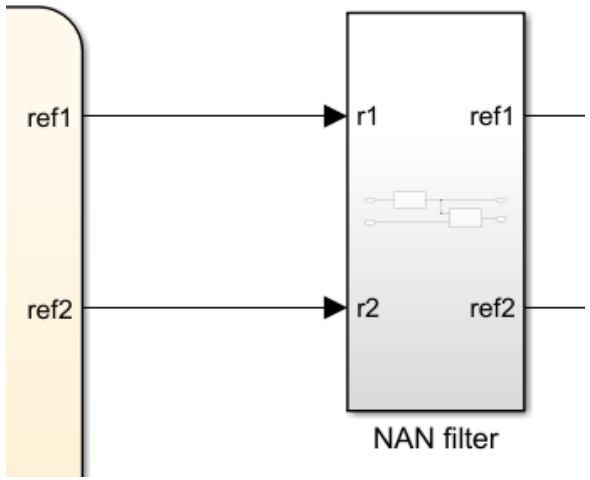





```
%% new_track_generator
VisualParam.angle_of_view3 = 30; % 기존 degree_range2
VisualParam.erased_angle3 = 60; % 기존 prev_degree_range
VisualParam.radius3 = 50; % 기존 radius
```

- Landing이 아니라면 find_new_track상태로 천이한다.
- new_track_generator : 드론 중심에서 이전 방향 erased_angle3각도를 제외하고 angle_of_view3만큼의 각도를 angle_of_rotation만큼 돌려가면서 부채꼴 영역 내부의 점이 가장많은 경우를 찾는다. 이때 점이 가장 많은 부채꼴 영역의 평균값으로 두번째 reference값을 구한다.
- 이후 second_ref_generator 함수를 통해 두 번째 reference를 생성한다.

5. NAN filter (Not A Number filter)



- Landing referenc를 계산하는 과정에서 드론의 흔들림으로 인해 이미지 상에 어떠한 binarization된 점들이 나타나지 않아 ref1, ref2에 NAN값을 갖게 되는 문제점을 해결하기 위해 고안되었다.
- Reference가 NAN이면 MPC를 계산하기 위해 quadprog함수를 실행하는 과정에서 문제가 발생한다.
- ref1, ref2이 NAN라면 (0.0)으로 좌표를 변경해줌으로써 오류를 해결한다.

END

Thank you