Carom
설계 문서

22.07.14 - 22.08.29

소속 | 광운대학교
작성 | 김민지, 이종윤, 이탁균

(주) 에스프레소 미디어

목차

I.Detect 모듈

1. 소개

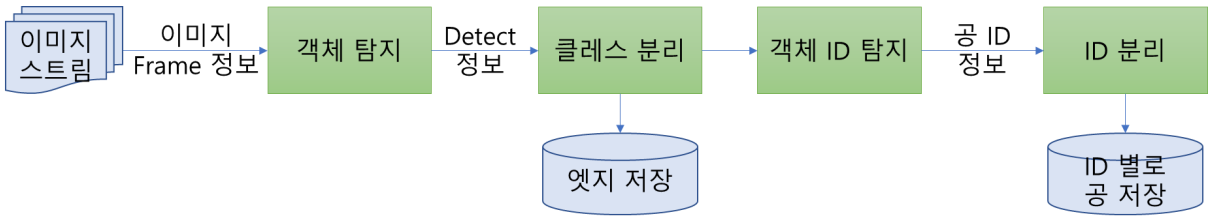
당구대(나사 안쪽 영역 네 개의 모서리)와 공들(당구대 안쪽 세 개의 공)을 찾아 내기 위한 모듈. 영상의 매 프레임 마다 공들의 위치들의 찾고, 당구대 모서리의 위치를 찾아 낸다.

2. 설계

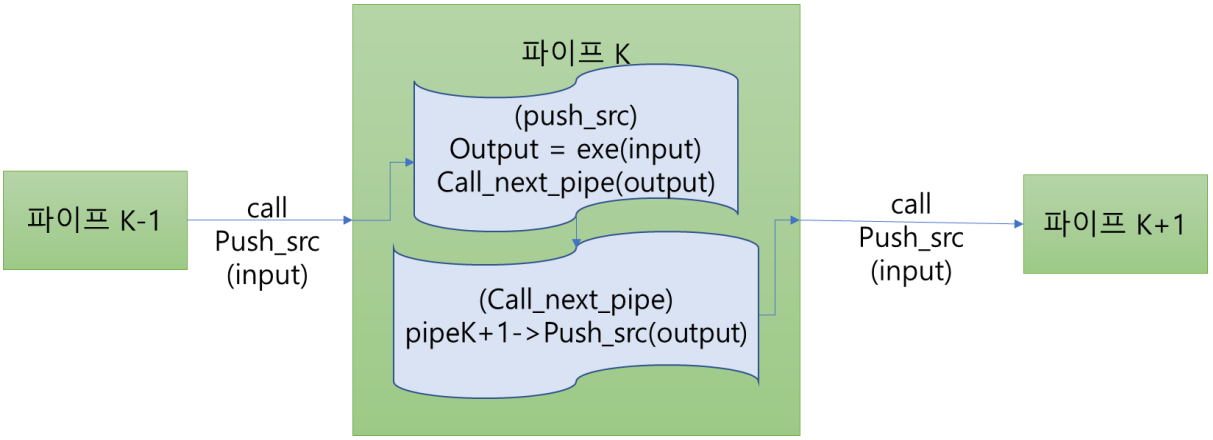
구조 설계

파이프라인 패턴, 옵저버 패턴을 모티브로 구조를 설계했다. 각 기능을 파이프로 제작하여 특정 기능의 변경이나 재사용이 필요할 때 용이하도록 했고 파이프의 입출력을 자동화하기 위해 옵저버 패턴을 추가로 사용했다.

파이프라인



옵저버를 통한 파이프 입출력 자동화



파이프 연결

파이프와 파이프를 연결하기 위해서는 connect_pipe라는 명령어를 통해서 연결해야 한다. 현재 파이프의 다음 파이프가 무엇인지를 설정하고 싶다면,

onePipe.connect_pipe(twoPipe)

와 같은 식으로 파이프를 연결해야 한다. split 계열의 파이프는 뒷 단에 여러개의 파이프가 연결될 수 있는데, 하나씩 파이프를 연결하면, 자동적으로 다른 인덱스가 주어져서 이어진다.

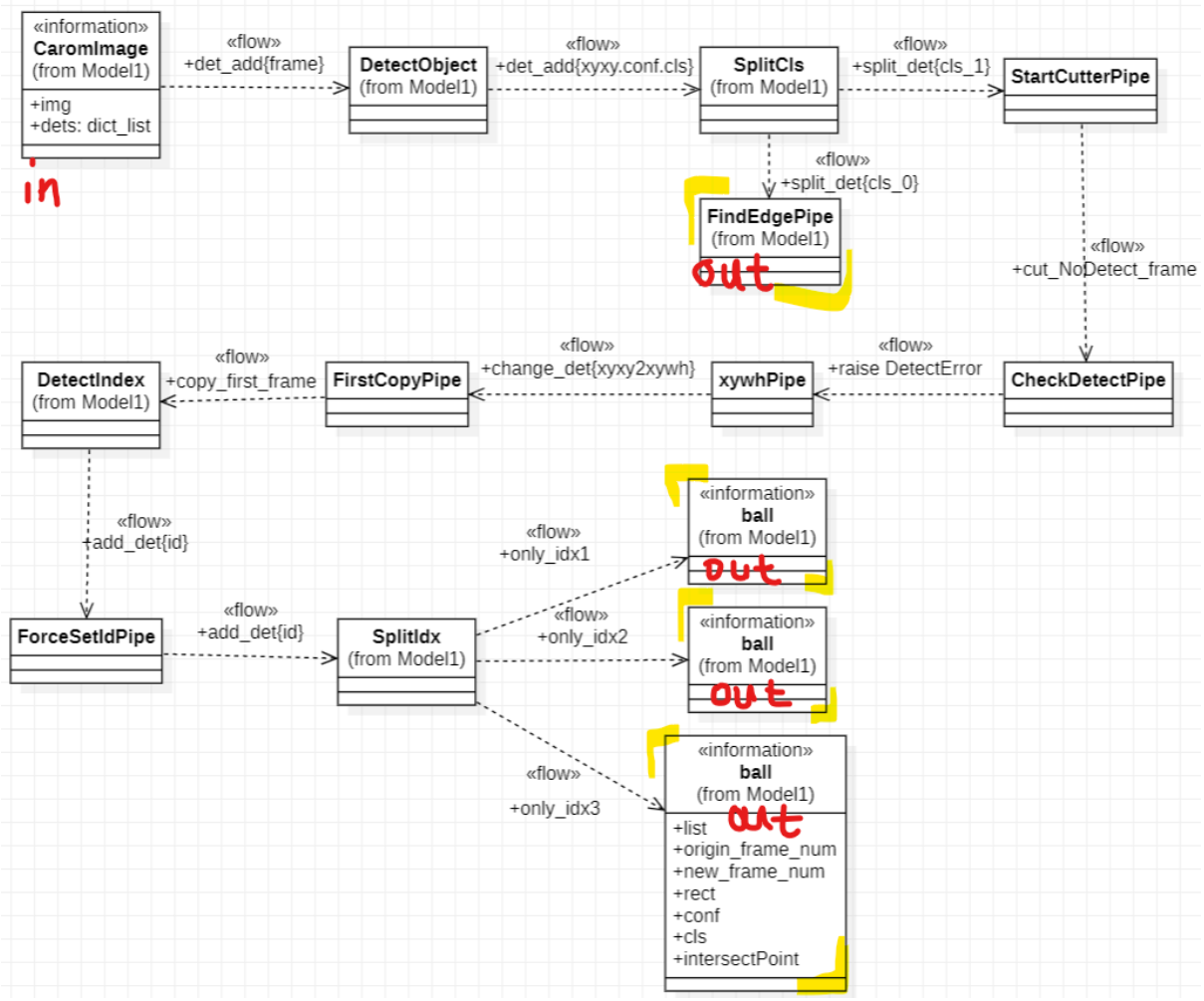
```
for i in range(split_idx_pipe.idx2label.__len__()):
    bag = ResourceBag()
    split_idx_pipe.connect_pipe(bag)
```

파이프 팩토리

파이프의 연결상태를 관리하는 생성자 함수이다. 밑에서 Flow 차트로 설명되는 파이프 설계도에서 나타내는 순서대로 파이프를 연결해 놓았다. 파이프를 생성하고, 해당 파이프를 순서대로 연결 한 뒤에 시작 파이프와 말단 bag을 반환한다.

시작 파이프(DetectObject)에 push_src를 진행하면, 말단 bag에 순차적으로 값이 쌓일 것이다.

Flow 설계



파이프 설계

DetectObjectPipe

원리	1. yolov5를 통해 3개의 공과 4개의 모서리를 detect 한다. 이때 max_det 옵션을 7개로 고정하여 최대 7개의 객체만 탐지될 수 있도록 한다. 2. 탐지된 객체를 PipeResource 의 dets 변수에 저장한다. 3. SplitCls 를 이용해 Edge와 Ball 로 분리하고 다음 파이프로 전달한다.
동작	yolo 알고리즘을 활용하여 동작한다

precondition
1. 영상은 항상 탑 뷰로 들어온다. 2. 정상적인 경우 매 프레임 별 3개의 공 det과 4개의 모서리 det이 입력된다. 3. 때때로 모서리의 일부가 가려져서 입력되기도 한다. 4. 때때로 모서리의 전체가 가려져서 입력되기도 한다. 5. 공의 움직임을 가리는 경우는 없다. (오류다)
postcondition
1. 정상적인 경우 3개의 공과 4개의 모서리를 탐지한다. 2. 4개의 모서리가 입력됐지만 2개나 3개만 탐지하는 경우가 존재한다. 3. 2개/3개의 모서리가 입력되고 2개/3개의 모서리를 탐지하는 경우가 존재한다. 4. 3개의 공이 입력됐지만 2개만 탐지하는 경우가 존재한다.

FindEdgePipe

기능	프레임 별로 조금씩 detect 된 Edge가 다른 부분을 보정하고 detect되지 못한 Edge를 추론한다.
원리	1. 한 영상의 모든 프레임에서 찾은 Edge 리스트를 입력 받는다. 2. Edge를 영상의 중심을 기준으로 좌상단, 우상단, 우하단, 좌하단으로 구분한다. 3. 모든 프레임에서 검출된 Edge를 확인하여 한 영상에서의 Edge를 하나로 결정한다. - Edge가 4개 검출 된 경우 각 모서리 마다 가장 많이 검출된 값을 영상에서의 Edge로 함 - Edge가 3개 검출 된 경우 perspective transform을 통해 나머지 한 Edge를 추론하여 Edge를 결정한다. - Edge가 2개 검출된 경우 직사각형의 모습으로 가정하고 가로:세로=2:1 의 비율에 맞게 나머지 두 Edge를 추론하여 Edge를 결정한다. 4. 최종 결정된 Edge의 네 점을 출력한다.

DetectIndexPipe

원리	이전 프레임에 존재했던 객체(박스)와 현재 프레임의 객체를 비교해서, 객체가 이동 했을 법한 위치의 유사도(모션코스트)와 객체의 시각적 특징의 유사도(특성코스트)를 측정하여 객체를 트레킹하게 된다. 즉 같은 객체로 인식한다.
동작	deep sort 알고리즘을 활용하여 동작한다. deep sort 를 통하여 나온 id에 대해서 dets에 id를 갱신한다.

Parameter			
이름	현재 값	설명	기타
weights	Osnet_x0_25_market1501.pt	DeepSort의 특징 유사도를 구하는 osnet의 weight 파일	이 이름이 아니면, 오류가 난다. 한번 오류를 내보면, 쓸수 있는 신경망이 정해져 있다는 것을 알 수 있다.
ECC	True	카메라 움직임에 대한 보정 연산	입력영상이 Top view로 고정되어 있기 때문에 False이여도 별 차이가 없다. 연산속도를 위해 False로 변경해도 무방할 것 같다.
MC_LAMBDA	0.8 [0,1]	현cost(1-MC_LAMBDA)와 모션cost(MC_LAMBDA)을 확인하는 비중, 표현cost는 osnet을 통해 나온 벡터값의 유사도이고, 모션cost는 칼만필터를 비롯한 움직임을 통한 유사도이다.	현제는 모션cost에 따른 유사도가 훨씬 더 크다. 공의 충돌할 때 공의 아이디가 바뀌는 경우가 많다면, MC_LAMBDA의 값을 조금 하향 조정하도 좋을 것 같다.
EMA_ALPHA	0.9 [0,1]	모양을 업데이트 하는 인자. 지수함수 평균방식으로 보정한다.	실제로 보정이 되고 있는 것인지 잘 모르겠다.
MAX_DIST	0.4 [0,1]	매칭 경계값. 표현, 모션 cost를 통해 나온 confidence값을 가지고 나누는 기준 값	객체가 많지 않기 때문에 좀더 잘 같은 객체로 인식했으면 좋겠다면, 낮추어 주는 것도 좋을 것 같다. 어차피 객체는 추가 되지 않기 때문이다.
MAX_IOU_DISTANCE	1.5	프레임간 객체가 떨어져 있을 수 있는 최대거리. 객체가 나중에 등장했을 때 어느정도 거리에 떨어져 있어도 같은 객체로 인식해 주는지를 판단하는 경계값	공의 속도가 빨라서 탐지하지 못하는 경우가 보인다면 좀더 값을 키워주는 것도 괜찮은 것 같다.

MAX_AGE	300	연속적으로 해당 객체를 찾지 못했을 때 Max_Age만큼 지날 시 해당 객체를 더 이상 같은 객체로 인식하지 않음	객체가 증감하지 않기 때문에 객체가 사라지지 않도록 크게 잡았다.
N_INIT	50	N_INIT만큼 등장 해야 객체로 인식하기 시작한다.	값이 작으면, id가 튀었을 때 새로운 객체가 만들어 지기도 한다. 이를 방지하기 위해서 크게 잡아 주었다. 하지만 시작 부분을 못잡는 경우가 생기는데, 그를 방지하기 위해서 처음 탐지를 성공한 프레임을 N_INIT번 복제해 주었다.
NN_ BUDGET	40	객체의 최대 크기	현제 테스트케이스에서는 끽해봐야 40정도 밖에 나오지 않는다. 굳이 건들지 않아도 될 것 같다.

참고 : 자세한 내용은 StrongSort에 대한 논문이나 오픈소스를 참고하길 바랍니다.

precondition
1. 공의 움직임을 전부 인식하기 위해서는 공이 움직이기 전 충분한 프레임이 필요 하다. : Sort 알고리즘은 객체에 ID 가 부여하기 위해 일정 횟수 만큼 객체를 인식해야함 2. 정상적인 경우 3개의 공 det가 들어온다. 3. 때때로 2개의 공 det가 들어온다. 4. 단 1개 이하의 공 det가 들어오는 경우는 없다. (오류다)
postcondition
1. 정상적인 경우 세개의 공 객체에 대해서 각각의 ID로 잘 구분한다. 2. id를 2개이상 못 찾은 경우 오류로 생각한다. 3. 공이 충돌시 id를 바꾸어서 인식하는 경우가 존재한다. : 여기서는 해결하기 힘들다. 4. 입력에서 공 세개를 찾았지만 id는 2개만 설정한 경우가 존재한다. : 같은 id에 대해서 연속해서 인식하지 못할 경우 오류로 인식한다. 5. 입력에서 공 두개를 찾았고, id는 2개를 설정한 경우가 존재한다. : 같은 id에 대해서 연속해서 인식하지 못할 경우 오류로 인식한다.

xyxyPipe

기능	xyxy(top,left / bottom,right)기반의 box 좌표를 xywh(center-x, center-y, width, height)로 변경해 준다.

이유	DetectOjbectPipe의 xywh의 값이 xyxy이지만, DetectObject의 box 형식은 xywh이기 때문이다.

FirstCopyPipe

기능	해당 파이프에 도달한 가장 첫 프레임을 일정 반복횟수 만큼 복제하여 다음 파이프에 넘겨준다.
이유	DetectIndexPipe에서 공 객체에 id를 부여하기 위해서 해당 객체가 일정 횟수만큼 등장 해야 하는데, 첫 프레임 부터 공에 id를 부여하기 위해서 첫 프레임을 복제했습니다.

StartCutterPipe

기능	처음부터 공 3개를 다 찾은 프레임 전까지 전부 무시해 버린다.
이유	FirstCopyPipe에 도달하는 첫번째 프레임의 정보가 복제될 때, 공을 전부 인식하지 못한 프레임이 복제되면, 공의 위치가 정상적으로 인식하지 못한다.
첨언	사실 첫 프레임에 공 3개를 찾지 못하는 경우는 영상오류이다. 해당 파이프를 넣어서 많은 오류 가능성을 내포하게 되었다. 차라리 첫 프레임에서 공 3개를 찾지 못한 경우 영상오류로 처리하는 파이프가 좋을 것이라고 생각한다.

ForceSetIdPipe

기능	공 3개를 찾았고 id는 2개만 부여 되어 있을 때, id가 부여되지 않는 객체가 남은 id를 부여한다.
이유	이유 Sort의 파라미터를 잘못 설정해서 성능이 안나왔다. 적당한 성능으로 끌어올리기 위해서 해당 파이프를 붙였다.
첨언	Sort을 성능이 개선된 이후 득보다 실이 많게 되었다. 가끔 완전 기존 경로와 다르게 튀어서 잡은 경우가 있는데, 꽤 많은 경우가 이 파이프의 문제이다. 당구대 밖 손을 잡거나 했을 때 거르지 못하게 만드는 원인이 된다. 해당 파이프로 제거하는 것이 좋을 것 같다.

CheckDetectPipe

기능	DetectObjectPipe에서 오탐지한 공의 개수가 10개 이상이면 해당 영상을 오류 영상으로 처리한다.
기준	오탐지의 기준은 해당 프레임에 발견된 공의 개수이다. 3개가 정상인데 그것보다 많거나 적으면 모두 오탐지 이다.
이유	오탐지한 공의 개수가 너무 많아지면 해당 영상의 분석 결과물에 대한 신뢰성이 떨어지기 때문이다.

3. 예외 처리

DetectObjectPipe

예외	처리
탑뷰 당구대 이미지가 아닌 것이 들어오거나 공을 가리거나 당구대 밖에서 공으로 탐지된 경우 -> ball det가 3개이 아닌 값을 반환한다.	1. 시작 프레임이 오류인 경우(공을 3개 찾지 못한 경우) 비디오를 오류로 처리한다.(미완) 2. 매 프레임 마다 공 오탐지 개수 (abs(N-3) 값)를 구해서 총 10개가 넘으면 미분석 처리한다.**(완료)** **- 참고 : CheckDetectPipe에서 연산**
엣지의 개수가 4개가 아닌 경우	1. 2,3개인 경우 다른 엣지의 분포에 따라 가상의 엣지를 만들어 4개의 엣지를 반환한다.**(완료)** 2. 1개인 경우 비디오 오류로 처리한다.**(완료)** **참고 : FindEdgePipe에서 연산**

DetectIndexPipe

예외	처리
입력 det가 3개보다 많은 경우	1. 처음부터 계속 4개 이상인 경우를 제외하면, Sort가 ID를 잘 줄 것이라고 가정한다. 2. 꽤 오랜 프레임 동안 4개 이상 검출되는 경우는 CheckDetectPipe에서 걸러줄 것이라고 가정한다.
입력 det가 3개보다 적은 경우	1. 첫 프레임이 공 3개가 없는 경우는 없다고 가정합니다. 2. 연속해서(3~4개?) 같은 ID의 공을 찾지 못하는 경우는 미분석 처리합니다.(미완) 3. 이전 프레임에서 공을 놓치면서, 다음 공을 놓치게 되는 경우 StrongSort 알고리즘 문제이기 때문에 미분석으로 빼야한다.(결과는 연속해서 같은 ID를 인식못하는 형태로 나올 것임 으로) - 버그 발생 시 (MAX_IOU_DISTANCE)를 좀더 큰 값으로 설정할 것을 권장함
출력 det가 3개보다 많은 경우	1. 이런 경우는 대체로 존재하지 않으며, 이 조건이 만족한 경우 CheckDetectPipe에서 오류 처리 된다.
출력 det가 3개보다 적은 경우	1. 입력 det가 3개보다 적을 때는 위에서 정의한 것과 동일합니다. 2. 입력 det가 3개인데 출력 det가 2개인 일때는 공이 가려졌거나 공이 아닌 것이 공으로 인식되어 들어온 경우 가 존재합니다. (미완) 3. 입력 det가 3개인데 출력 det가 2개인 일때는 공이 충돌로 인해서 공이 바뀐 첫 프레임에 공을 ID만 인식할 수 있습니다. (미완) 4. 2,3번에 문제에 대해서는 파이프 내부에서 처리할 수 있는 것이 없습니다. 파이프 밖에서 당구대의 크기를 알면, ID를 부여하지 못한 당구공이 정상인지

	아닌 지를 판별할 수 있을 것 같습니다. 그 외에 부분에 대해서는 알고리즘을 손대야 하는 문제이기 때문에 이전에 설정했던 예외 처리에서 더 나아가서 분류하는 것은 당장은 아이디어가 없습니다.
출력 det에 id가 갱신되지 않은 경우	1. 입력 det의 개수보다 id를 설정한 det의 개수가 적을 때 id가 설정되지 않은 상태로 입력이 들어옵니다. 2. 해당 파이프 뒷단에서 id로 dets를 접근하면, 오류가 날 수 있으며, id로 접근이 실패했을 때의 에러처리 코드를 작성해야한다.(미완)
입력 det는 3개 출력 det는 2개인 경우	1. 입력이 세개이고, 출력이 두개인 경우 남은 ID를 놓친 객체라 판별해서 강제로 삽입합니다. (ForceSetIdPipe) - 하지만 성능이 별로 좋지 못함, 가끔 손을 잡거나 하는 것도 앞에서 걸러내면 다시 잡기 때문에 오히려 오류를 만들어 냅니다. 파이프에서 제거하는 것이 좋을 것 같습니다.
출력 det의 id가 3이상인 경우	1. 정상적인 경우 id는 [0,2]의 범위를 가집니다. id가 3이상인 경우는 이상이 있는 경우에 속합니다. 2. 다른 객체가 추가로 잡혀서 3이상이 나온 경우, 이런 경우는 CheckDetectPipe에 의해 나올 수 없다고 가정합니다. 3. 기존 공이 새로운 공으로 다시 인식된 경우, 이 경우 한동안 인식되지 않다가 id가 바뀐 뒤 다시 인식되야 합니다. 그럴 경우 한동안 det가 두개이기 때문에 "입력 det가 3개보다 적은 경우의 1번"에 이유와 같은 항목으로 미분석 처리됩니다.(미완)

II. Tracking 모듈

1. 소개

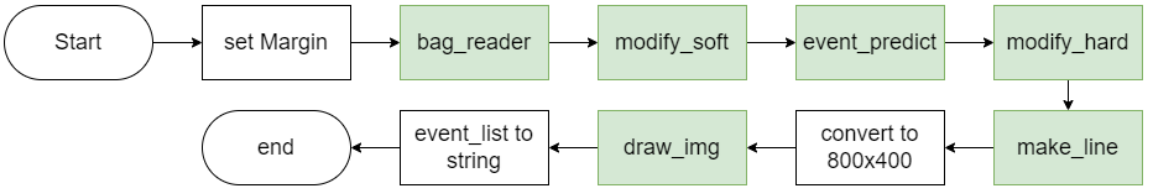
당구공의 충돌 지점을 알아내기 위한 모듈.
각 당구공의 위치와 쿠션까지 거리를 비교하여 충돌이 발생한 지점을 추론한다.

목표

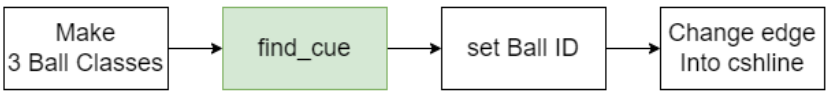
1. 각 공을 큐볼, 제1목적구, 제2목적구로 구분한다.
2. 공과 공이 충돌한 지점을 찾는다.
3. 공과 쿠션이 충돌한 지점을 찾는다.
4. 찾아낸 충돌 지점들을 실제 충돌 지점에 가깝도록 보정한다
5. 당구대 내부 영역을 분리하여 800x400 크기로 변환하고 각 공의 이동 경로를 영상에 그린다.

2. 설계

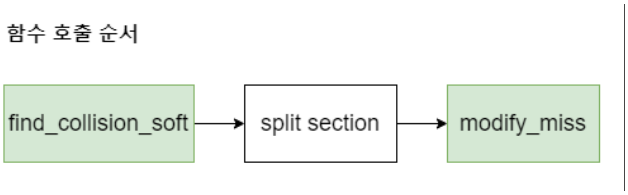
Flow 설계


Start
set Margin
bag_reader
modify_soft
event_predict
modify_hard
make_line
convert to 800x400
draw_img
event_list to string
end

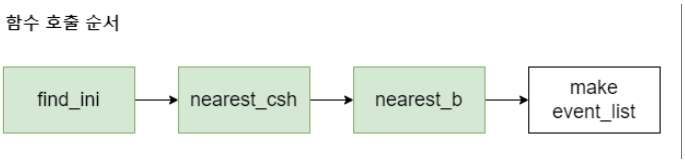
Bag Reader

흐름	함수 호출 순서 
기능	detect 모듈에서 전달된 정보들을 Predictor 모듈에서 사용할 클래스의 형태로 저장하고 각각의 공을 큐볼, 제1목적구, 제2목적구로 구분
원리	1. 각 공이 처음 움직인 프레임을 비교하여 빠른 순서대로 큐볼, 제1목적구, 제2목적구로 구분 2. detector에서 전달 받은 edges에서 정보를 읽어와 당구대 모서리의 좌표에서 각 쿠션에 해당하는 두 모서리의 좌표 쌍으로 저장 3. Ball클래스에서 각각의 공을 id에 맞게 분류한 후 find_cue()를 통해 큐볼과 목적구로 구분 4. 우상단, 좌상단, 좌하단, 우하단의 모서리 좌표를 [(우상단, 좌상단), (좌상단, 좌하단), (좌하단, 우하단), (우하단, 우상단)]으로 이루어진 cshline 리스트로 변환
내부 함수	**find_cue** 세 공을 입력으로 받아 큐볼, 제1목적구, 제2목적구 순서로 반환 **is_cue** 입력받은 공이 다른 공보다 먼저 움직였는지 여부를 반환 동시에 움직인 경우 두 공의 중심을 이은 방향으로 튕겨 나간 공이 늦게 움직인 것으로 여겨 목적구로 판단되도록 함

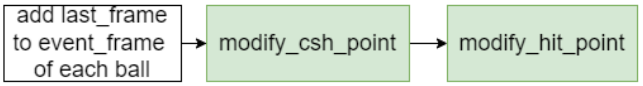
Modify Soft

흐름	
기능	detect 모듈에서 전달 받은 공의 정보들 중 좌표가 누락된 프레임에 대해서 좌표를 추론하여 생성
원리	1. detector에서 전달받은 좌표들 중 공을 검출하지 못한 프레임에서의 좌표를 추정하여 생성 2. find_collision_soft()를 통해 충돌이 발생할 가능성이 있는 모든 지점을 구함 3. find_collision_soft()에서 구한 지점들을 기준으로 구간을 나누고 각 구간에서 프레임 번호를 비교해 공이 검출되지 않은 프레임이 있는지 검사 4. modify_miss()를 통해 검출하지 못한 프레임에서의 좌표들을 생성 5. 각각의 구간들을 합쳐 누락이 보정된 전체 구간을 반환
내부 함수	**find_collision_soft** 충돌이 발생할 가능성이 있는 모든 지점을 반환 이동방향이 15’ 이상 틀어진 지점들 **modify_miss** 충돌 예상 지점을 기준으로 구간을 나누고 각 구간에서 누락된 좌표를 생성하여 보정

Event Predict

흐름	
기능	누락을 보정한 좌표들 중에서 충돌이 발생한 지점을 추론 마지막 충돌 프레임 이후로 각 유형의 가장 먼저 발생하는 충돌 정보들 중에서 가장 빠른 충돌 정보들을 저장하고 각 공의 마지막 충돌 정보들을 갱신하는 과정을 더 이상 충돌이 일어나지 않을 때까지 반복
원리	1. modify_soft()를 통해 누락된 프레임에 대해 보정을 거친 공의 좌표들에서 충돌이 있었던 프레임의 충돌 정보를 모아 반환 2. 충돌은 큐볼이 움직인 이후부터 발생하므로 last_event_idx를 큐볼이 움직인 시점으로 초기화 3. nearest_csh()와 nearest_b()를 통해 last_event_idx 이후 각 유형의 충돌에 대해 발생할 수 있는 가장 빠른 충돌 정보들을 모아 possible_event_list에 저장 4. possible_event_list를 충돌이 발생한 프레임 기준 오름차순으로 정렬 5. last_event_idx 이후로 가장 먼저 충돌이 발생하는 프레임으로 last_event_idx를 갱신하고 충돌 정보를 event_list에 저장 6. 해당 프레임에 여러 번의 충돌이 있었다면 그 충돌 정보들을 모두 저장 7. event_list에 저장한 충돌 정보에 따라 각 공 객체들의 last_event_obj와 last_csh_dist를 갱신 8. 다시 nearest_csh()와 nearest_b()로 예상 충돌 정보들을 얻고 가장 빠른 충돌 정보들만 저장하는 과정을 반복
내부 함수	**nearest_csh** 마지막 충돌 프레이 이후로 입력받은 공이 가장 먼저 쿠션과 충돌하는 프레임의 충돌 정보를 반환 **nearest_b** 마지막 충돌 프레임 이후로 입력받은 두 공이 가장 먼저 충돌하는 프레임의 충돌 정보를 반환

Modify Hard

흐름	함수 호출 순서 
기능	Event Predict에서 구한 충돌 지점들을 실제 충돌 지점에 가깝도록 위치를 보정
원리	1. 각 공의 모든 쿠션과의 충돌 지점을 공이 쿠션을 향해 이동하는 방향의 직선과 쿠션에서 튕겨 나오는 방향의 직선을 구해 두 직선의 교점으로 보정 2. 쿠션 충돌 전후로 다른 객체와의 충돌로 인해 쿠션을 향해 이동한 방향과 튕겨 나온 방향 중 하나를 알 수 없는 경우 방향을 알 수 있는 직선이 쿠션으로부터 ball_radius만큼 떨어진 지점으로 보정 3. 큐볼과 목적구가 처음 충돌하는 지점을 큐볼이 목적구로 향하는 방향의 직선과 목적구가 튕겨 나가는 방향의 직선을 구해 두 직선의 교점으로 보정 4. 큐볼이 목적구로 향하는 방향을 알 수 없는 경우 큐볼이 목적구와 충돌 후 튕겨 나온 방향의 직선과 목적구가 튕겨 나온 방향의 직선을 구해 두 직선의 교점으로 보정
내부 함수	**modify_csh_point** 각 공의 모든 쿠션 충돌 지점에 대해서 실제 충돌 지점에 가깝도록 보정 공이 쿠션을 향해 들어온 방향으로 그은 직선과 쿠션에 튕겨 나간 방향으로 그은 직선의 교점으로 보정 **modify_hit_point** 큐볼이 목적구와 최초로 충돌하는 지점들에 대해서 실제 충돌 지점에 가깝도록 보정 큐볼이 목적구를 향해 들어온 방향의 직선과 목적구가 튕겨 나간 방향의 직선의 교점으로 보정

3. 개선

nearest_b(cue, tar, last_event_idx, ball_margin)

- 빠른 속도의 공에 대해서는 가장 가까웠던 지점도 margin 범위 밖에 찍히는 경우가 많고 느린 속도의 공에 대해서는 충돌하지 않은 경우도 충돌로 판단하는 문제가 존재
- 두 공이 스쳐지나가는 경우 두 공의 충돌 전후로 이동방향을 비교하여 각도가 15’ 이상 변화한 공이 하나라도 있을 경우 충돌로 판단하였으나 속도가 느리거나 검출된 좌표가 흔들린 경우 잘못 판단하는 경우가 많음

modify_hit_point(cue, tar, event_list, cshline)

- 보정된 좌표가 기존의 좌표보다 목적구에 더 가까운지 비교하도록 개선해야 함
- 쿠션 충돌 지점을 갱신하는 과정에서 쿠션 충돌 지점이 보정으로 생성된 지점으로 갱신되어 목적구 충돌 지점 전후 프레임에 없어 더 가까운 지점으로 갱신되지 못하는 문제가 있음
- 충돌 보정 지점을 갱신하는 과정에서 event_frame이 오름차순으로 정렬되지 않는 문제가 있음, 같은 원인으로 영상에 경로가 그려지는 도중에 끊김

modify_csh_point(ball, event_list, hit_frame, cshline, ball_radius)

- 보정된 좌표가 기존이 좌표보다 쿠션에 가까운지 비교하도록 개선해야 함
- 쿠션 충돌 전후로 다른 객체와 충돌이 있었던 경우는 쿠션에 대해서 들어오거나 나가는 방향을 알 수 없어 보정이 불가능

modify_hard(cue, tar1, tar2, event_list, cshline, ball_radius)

- 현재 제 2목적구가 큐볼이 아니라 제 1목적구와 먼저 충돌한 경우에 대한 보정이 이루어지도록 개선해야 함

is_cue(b1, b2)

- detector에서 공의 좌표가 흔들려 정지상태의 공이 움직인 것으로 판단된 경우 큐볼을 잘못 구분하는 문제가 발생