

**Convergence Capstone Design**

**Final Project Report**

**PPO 기반의 강화학습을 이용한  
드론 배송시스템 구현**

2조 김현우, 김범진, 김창기, 안종원

# 목 차

## 1. 서론

1.1 프로젝트 선정 배경.....	p.03
1.2 프로젝트 목표.....	p.04

## 2. 본론

2.1 프로젝트 환경.....	p.05
2.2 PPO 알고리즘.....	p.06
2.3 컴포넌트 설명.....	p.07
2.4 ML-Agents를 이용한 강화학습 구현.....	p.09
2.5 강화학습 수행 결과.....	p.16
2.6 텐서보드를 이용해 확인한 결과.....	p.18

## 3. 결론

# 1. 서론

## 1. 프로젝트 개요

### 1.1 프로젝트 선정 배경

기술 발전이 빠르게 진행되어 가면서 택배와 같은 물류의 이동이 점점 빨라지고 있다. 심지어 전날 상품을 구매하면 다음날 새벽에 배송되는 등의 서비스를 제공하면서 이런 변화가 눈에 띄게 드러나고 있다. 최근에는 이런 트랜드의 변화와 함께 늘어난 물류량을 효율적으로 처리하기 위하여 많은 기업들이 드론을 이용한 배달을 구상하고 있는 단계이다. 이런 변화는 비단 우리나라 뿐만이 아니라 세계적인 트랜드로 세계적인 온라인 쇼핑몰을 운영하고 있는 기업인 아마존의 경우는 이미 기술적으로 드론배송을 구현하였고, 실제로 상용화하려고 노력중이지만 기술적 문제와 수요 예측이 어려운 문제로 인해 서비스 제공에 난항을 겪고있다. 아래 우측 사진을 보게되면 알 수 있듯 드론 택배의 시장규모가 실제로 점점 증가하고 있으며 다가오는 26년에는 지금의 약 2 배 규모로 늘어날 것으로 예상되고 있다. 또한 우리나라에서도 실제로 산간지역이나 섬 등 물류 운송이 어려운 지역에 대해 우체국에서 드론배송 시범운영을 진행하는 등 더 이상 드론 배송이면 이야기가 아니라는 것을 알 수 있다. 따라서 강화학습을 이용하여 위에서 말한 것 같은 드론을 이용한 배달을 직접 구현해 보면 실제로 해당 기술을 구현하는데 있는 어려움이나, 강화학습을 통해 보완할수 있는점을 직접 탐구해 보기 위하여 위와 같은 주제를 선정하여 프로젝트를 진행해 보았다.



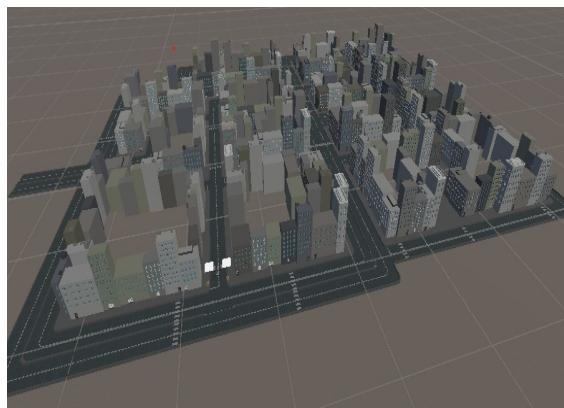
## 1.2 프로젝트 목표

Unity ML Agent 기반의 PPO 강화학습을 이용해 한정된 공간 안에서 드론과 드론이 배달할 물류, 드론이 물류를 배달할 장소를 지정해 놓고 드론이 물류를 탐색하여 해당 물류를 집고, 집은 물류에 해당하는 목적지로 다가가서 목적지에 해당하는 물류를 내려놓고 동일한 동작을 모든 물류에 대해 반복할 수 있도록 강화학습을 진행할 수 있도록 프로젝트를 진행하였다.

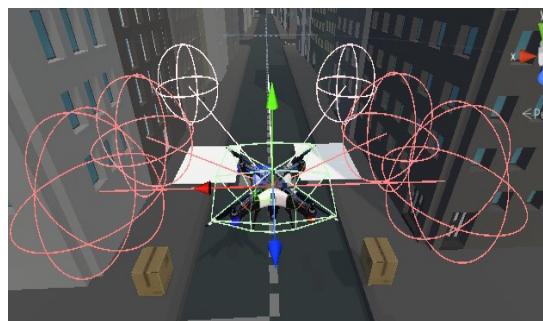


## 2. 본론

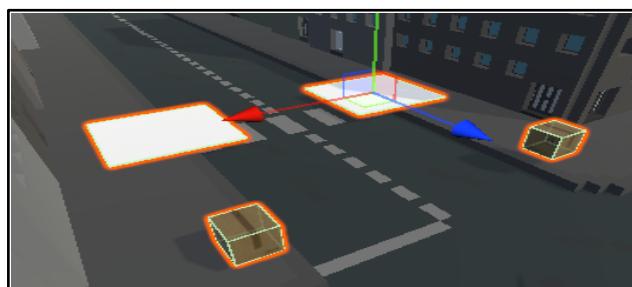
### 2.1 프로젝트 환경



우선 드론이 물건을 배송할 도시환경을 구현하기 위하여 유니티 애셋스토어의 무료 애셋인 city voxel pack이라는 애셋을 사용하여 기본적인 도시환경을 구현했다.



드론의 경우도 유니티 애셋스토어의 professional drone free pack을 사용하여 기본적인 드론의 움직임이나 추력, 방향전환과 같은 다소 구현하기 어려운 요소들을 애셋의 도움을 받아 쉽게 구현 할수 있도록 했다.





도시환경에 오브젝트를 직접 구현하여 box 형식의 물류와 흰색 사각형 모양의 목적지를 구현하여 해당 목적지에 물류가 도달하게 되면 그를 확인 할 수 있도록 home과 cube라는 이름으로 저장해 두어 이후 강화학습을 진행하며 충돌을 감지하거나 목적지에 도달했는지를 확인하기 위하여 해당 오브젝트를 구성했다.

## 2.2 PPO 알고리즘

학습데이터를 재사용하는 모델로, Episode 단위로 반영하는 것이 아닌 Step 단위로 학습데이터를 만들어 내어 학습하는 방식이다. PPO는 MM(Minorize-Maximization) 알고리즘을 기반으로 모델링 수식이 구성된다. MM 알고리즘에서는 반복할 때마다 대리 함수  $M$ 을 찾는 것을 목표로 하며, 그 때마다 Optimal point  $M$ 을 찾는다. 그리고 도출한 point  $M$ 을 현재 훈련에 사용할 기반정책으로 사용한다. 여기서 정책  $M$ 을 기반으로 lower bound를 재계산하며 이 과정을 반복해 나가는 것이 MM 알고리즘이다. 그러나 정책은 학습의 결과에 영향을 미치는 중요한 요소이므로 이 알고리즘을 반복적으로 수행할 때마다 정책을 지속적으로 향상시킨다. PPO의 objective function은 이전 학습시 저장해 둔 기준 보상값과 clip된 보상값들 중 작은 값을 취하는 형태로 되어 있다. 이를 통해서 예외적 결과인 변화가 큰 값의 정책 업데이트는 줄여 안정화된 학습이 되도록 유도한다.

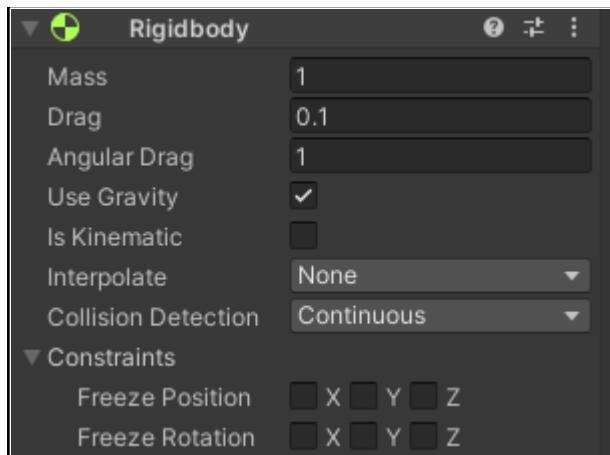
Trainer\_type ppo로 설정하고, 연속적인 행동 알고리즘이므로 batch size를 1024로 크게 설정했다. 이외의 learning rate, beta, epsilon, lambd 파라미터 값의 경우 학습을 여러번 수행해 보며 안정적으로 학습을 수행하는 동시에 적절한 학습속도를 갖도록 설정하였다.

```
1  behaviors:
2    CarBehaviour:
3      trainer_type: ppo
4      hyperparameters:
5        batch_size: 1024
6        buffer_size: 5120
7        learning_rate: 0.00035
8        beta: 0.0025
9        epsilon: 0.3
10       lambd: 0.95
11       num_epoch: 5
12       learning_rate_schedule: linear
13
14     network_settings:
15       normalize: true
16       hidden_units: 264
17       num_layers: 3
18
19     reward_signals:
20       extrinsic:
21         gamma: 0.95
22         strength: 0.99
23
24       keep_checkpoints: 15
25       checkpoint_interval: 100000
26       time_horizon: 264
27       max_steps: 100000000
28       summary_freq: 100000
29       threaded: true
30
31
```

## 2-3 컴포넌트 설명

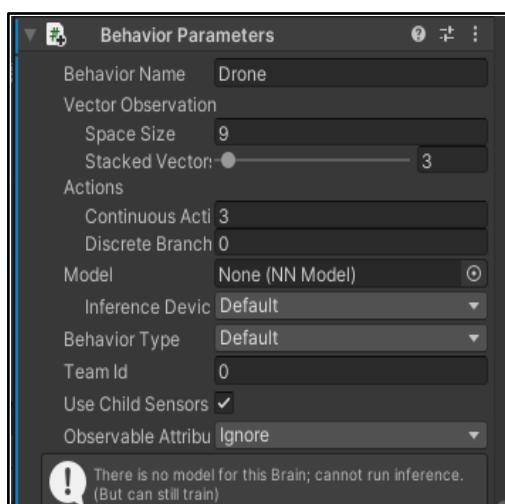
### 1) Rigidbody

Rigidbody는 Gameobject가 물리적으로 동작할수 있도록 한다. 다시말해 어떤 물체가 물체 받을 때 더 사실적인 움직임을 보일수 있도록 설정해주는 옵션으로 드론의 움직임 및 물류와의 상호작용 등을 위하여 Rigidbody를 통해 질량을 조절하고 목적지의 position을 고정시켜 움직이지 못하고 고정된 위치에 있게 만드는 등 해당 object에 대한 다양한 설정을 진행하여 충돌을 감지하는 등의 기능을 구현하였다.



## 2) Behavior Parameters

Behavior Parameters 컴포넌트를 통해 에이전트의 ML-Agents를 통한 학습에 관련된 다양한 Parameter를 설정한다.



Space Size(벡터 관측의 크기) = 9

Stacked Vector(관측의 누적 횟수) = 3

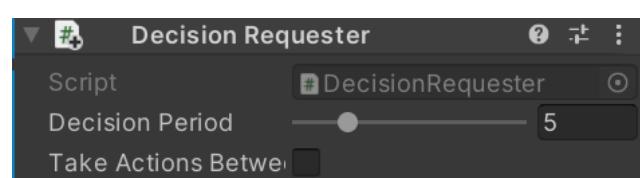
Continuous Actions = 3

Discrete Branches = 0

연속적 행동은 3개의 축으로 행해지며 이산적 행동은 없다.

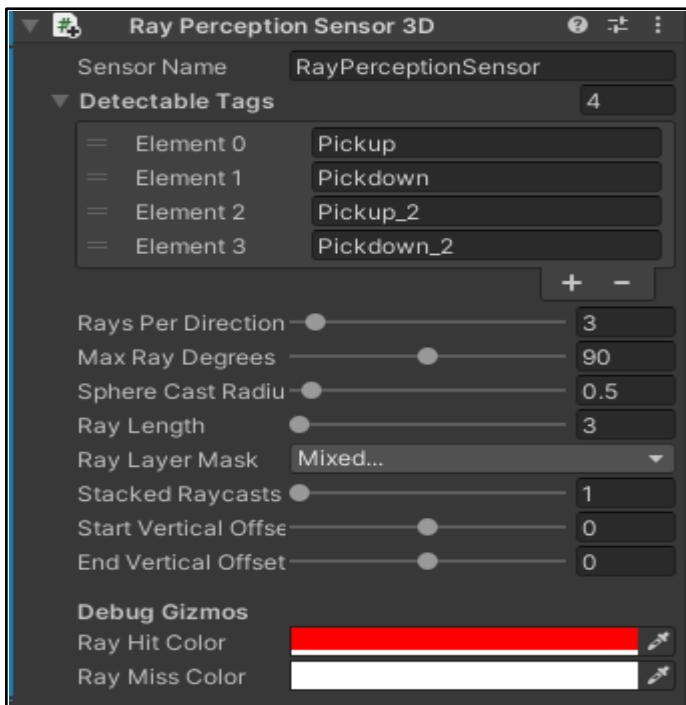
## 3) Decision Requester

Decision Requester 컴포넌트를 통해 일정시간마다 Decision request를 보내서 에이전트가 어떤 Action을 취할지 정책에 결정을 요청한다.



Decision Period: 에이전트가 새로운 행동을 결정하는 스텝의 간격, 5스텝마다 한 번씩 행동을 결정하도록 한다.

#### 4) Ray Perception Sensor 3D



Ray Perception Sensor 3D를 이용해 드론이 움직이면서 충돌할 수 있는 건물이나 바닥의 거리를 측정하고 물류를 확인하고, 해당 물류를 내려놓을 목적지에 대한 정보를 센싱하기 위하여 사용.

좁은 공간이기 때문에 거리를 3으로 설정하고 평면적으로만 탐지하는 것이 아니라 아래 쪽이나 윗쪽도 동시에 탐지하기 위하여 원 형태로 된 범위를 키워 더 정확한 감지가 되도록 하였다.

#### 2-4 ML-Agents를 이용한 강화학습 구현

##### 1. Agent, State, Action, Reward 정의

###### 1) Agent: 드론



## 2) State

- ① Agent의 위치 좌표 (x, y, z)



Agent의 초기 위치는 다음과 같고 모든 배송에 성공할 때까지 지속적으로 위치가 변화한다.

```
private Transform agentTrans;
```

```
agentTrans = gameObject.transform;
```

```
Vector3.Magnitude(agentTrans.position);
```

`transform.position` 함수를 사용해 Agent의 현재 좌표를 State로 사용한다.

- ② 랜덤으로 생성되는 두개의 목적지의 위치 좌표 (x, y, z)

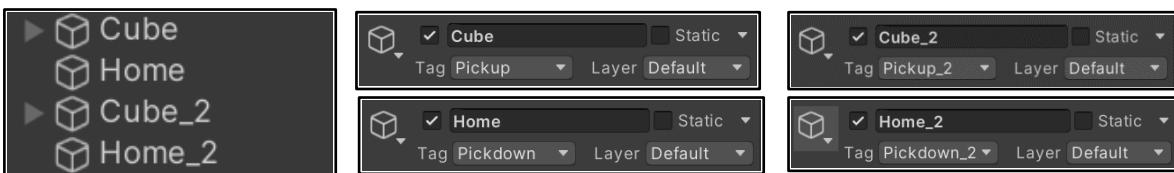


```
public void AreaSetting()
```

```
HomeTrans.position = HomeInitPos + new Vector3(Random.Range(-0f, 0f), Random.Range(-0f, 0f), Random.Range(-40f, 0f));
Home_2Trans.position = Home_2InitPos + new Vector3(Random.Range(-0f, 0f), Random.Range(-0f, 0f), Random.Range(-40f, 0f));
```

DroneSetting 스크립트에서 환경이 새로 시작될 때 환경을 초기화하는 함수 AreaSetting을 선언 한다. 그리고 두개의 목적지의 초기 위치 HomeInitPos, Home\_2InitPos에 -40에서 0사이의 랜덤한 z축 좌표를 더해 환경이 초기화 될 때마다 목적지가 직선상의 보도 위에서 랜덤으로 배치되도록 한다.

### ③ 각 오브젝트와 드론 사이의 거리



환경에 사용된 오브젝트는 총 4개로 배송할 물체 2개, 배송 목적지 2개로 이루어지고 C# 스크립트에서 각각 Pickup, Pickup\_2, Pickdown, Pickdown\_2의 태그가 부여된다.

```
//픽업(큐브1)과 드론사이 거리
float distance_pickup_agent = Vector3.Magnitude(PickupTrans.position - agentTrans.position);
```

```
//픽다운(홈1)과 드론사이 거리
float distance_pickdown_agent = Vector3.Magnitude(PickdownTrans.position - agentTrans.position);
```

```
//픽업2(큐브2)과 드론사이 거리
float distance_pickup_2_agent = Vector3.Magnitude(Pickup_2Trans.position - agentTrans.position);
```

```
//픽다운2(홈2)과 드론사이 거리
float distance_pickdown_2_agent = Vector3.Magnitude(Pickdown_2Trans.position - agentTrans.position);
```

각 오브젝트의 위치 좌표에서 Agent의 위치좌표를 빼서 거리를 계산하고 리워드 계산에 사용한

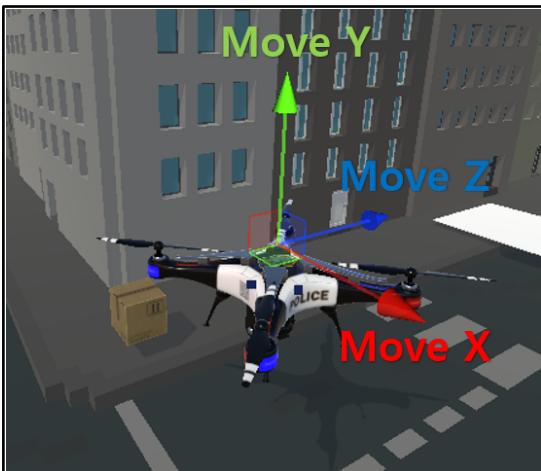
다.

#### ④ 5개의 bool 타입 플래그 변수

```
8 references
public bool flag;
8 references
public bool flag_2;
4 references
public bool pickdownflag;
4 references
public bool pickdownflag_2;
8 references
public bool pickdown_done_flag;
```

에이전트의 현재 배송 상태를 나타내는 bool 타입 변수 5개를 선언한다. 에피소드가 시작되면 모든 플래그가 false로 시작하며 물체의 픽업여부와 픽다운 여부에 따라 true와 false로 바뀌며 5개의 플래그의 상태를 통해 에이전트가 받게 되는 리워드가 바뀌며 그에 따라 Action 또한 바뀌게 된다.

### 3) Action



```
var actions = actionBuffers.ContinuousActions;

float moveX = Mathf.Clamp(actions[0], -1, 1f);
float moveY = Mathf.Clamp(actions[1], -1, 1f);
float moveZ = Mathf.Clamp(actions[2], -1, 1f);
```

Agent의 Action은 x, y, z축 방향으로의 연속적인 이동이며 매 스텝 파이썬 코드로부터 행동 결과를 전달받았을 때 Agent의 행동을 처리하고, 그에 대한 보상을 제공하고 에피소드의 종료 여부를 결정하는 OnActionReceived 함수에서 정의된다. 파이썬 코드에서 전달된 Action은 ActionBuffers

구조로 유니티 코드로 전달되도록 하며 드론 환경에서 Action은 연속적인 값이기 때문에 actionBuffers.ContinuousActions를 이용해 actions에 행동 정보를 저장한다. x, y, z축 총 3개의 실수 값을 받기 때문에 0, 1, 2 총 3개의 인덱스로 각 실수 값을 받아온다. 이때 Mathf.Clamp 함수를 통해 입력 값을 지정한 최솟값, 최댓값의 범위로 잘라서 사용한다.

```
dcoScript.DriveInput(moveX);
dcoScript.StrafeInput(moveY);
dcoScript.LiftInput(moveZ);
```

이어서 dcoScript의 DriveInput, StrafeInput, LiftInput의 입력으로 전달해 드론의 x, y, z축 방향으로의 Action을 제어한다.

#### 4) Reward

Reward는 크게 OnActionReceived 함수와 OnCollisionEnter 함수 내에서 정의된다. 매 스텝 코드로부터 행동 결과를 전달받았을 때 그에 대한 보상을 제공하기 위해서는 OnActionReceived 함수에서, Agent가 오브젝트와 충돌 시의 보상과 그에 따른 에피소드의 종료 여부를 결정하기 위해서는 OnCollisionEnter 함수에서 Reward가 정의된다. 먼저 OnActionReceived함수 내의 Reward는 다음과 같다.

① 매 스텝 드론이 정지하지 않도록 하기 위한 Reward

```
public override void OnActionReceived(ActionBuffers actionBuffers)
{
    AddReward(-0.01f);
```

매 스텝마다 -0.01의 작은 음의 Reward를 부여해 드론이 정지하지 않도록 한다.

② 드론의 고도가 너무 높아졌을 때의 Reward

```
var altitude = agentTrans.position.y;
if(altitude >= 25){
    Debug.Log("Please Throttle down");
    SetReward(-3f);
    EndEpisode();
```

드론의 y좌표를 변수 altitude로 받고 그 값이 25 이상일 경우 -3의 Reward를 set하고 에피소드를 종료해 학습이 진행될수록 드론이 적절한 고도를 유지하도록 한다.

### ③ 드론이 물체를 픽업한 상태가 아니며, 픽다운한 적도 없는 상태(초기상태)

```
float distance_pickup_agent = Vector3.Magnitude(PickupTrans.position - agentTrans.position);
if(flag != true && flag_2 != true && pickdown_done_flag != true)
{
    if(distance_pickup_agent > 30f)
    {
        SetReward(-1f);
        Debug.Log("e1");
        EndEpisode();
    }
    else
    {
        float reward1 = predistance_pickup_agent - distance_pickup_agent;
        AddReward(reward1 * 5f);
        predistance_pickup_agent = distance_pickup_agent;
    }
}
```

flag와 flag\_2가 false, 즉 드론이 두 개의 물체 중 하나도 픽업한 상태가 아니며 pickdown\_done\_flag 또한 false, 즉 물체를 배송해서 픽다운한 적도 없는 상태일 때의 reward는 위와 같이 부여한다. 먼저 if문을 통해 물체1과의 거리가 30보다 멀어지면 -1의 reward를 부여하고 에피소드를 종료하여 드론이 물체와 너무 멀어지지 않도록 한다. 그렇지 않을 경우 물체와의 이전 거리에서 현재 거리를 뺀 변수 reward1을 선언하고 거리 변화량에 5를 곱한 값의 reward를 부여한다. Reward 부여 후 물체와의 이전거리에 해당하는 변수 predistance\_pickup\_agent에는 물체와의 현재거리 distance\_pickup\_agent를 대입해 이전거리와 현재거리가 갱신되도록 한다.

### ④ 드론이 1번 물체를 픽업한 상태

```
float distance_pickdown_agent = Vector3.Magnitude(PickdownTrans.position - agentTrans.position);
if(flag == true)
{
    if(distance_pickdown_agent > 100f)
    {
        SetReward(-1f);
        Debug.Log("e2");
        EndEpisode();
    }
    else
    {
        float reward2 = predistance_pickdown_agent - distance_pickdown_agent;
        AddReward(reward2 * 5f);
        predistance_pickdown_agent = distance_pickdown_agent;
    }
}
```

Flag가 true일 때 즉 드론이 첫번째 물체를 픽업한 상태일 때의 reward는 위와 같이 부여한다. 물체1을 배송할 목적지1과의 거리가 100보다 멀어지면 -1의 reward를 부여하고 에피소드를 종료하여 드론이 목적지와 너무 멀어지지 않도록 한다. 그렇지 않을 경우 목적지와의 이전 거리에서 현

재 거리를 뺀 변수 reward2를 선언하고 거리 변화량에 5를 곱한 값의 reward를 부여한다.

#### ⑤ 드론이 1번 물체를 목적지1에 픽다운 완료한 상태

```
float distance_pickup_2_agent = Vector3.Magnitude(Pickup_2Trans.position - agentTrans.position);
if(pickdownflag == true && flag_2 != true && pickdownflag_2 != true)
{
    if(distance_pickup_2_agent > 100f)
    {
        SetReward(-1f);
        Debug.Log("e3");
        EndEpisode();
    }
    else
    {
        float reward3 = predistance_pickup_2_agent - distance_pickup_2_agent;
        AddReward(reward3 * 5f);
        predistance_pickup_2_agent = distance_pickup_2_agent;
    }
}
```

물체1을 픽다운 하면 pickdownflag가 true가 되므로 다음 물체를 픽업하기 위해 reward는 위와 같이 부여한다. 물체2와의 거리가 100보다 멀어지면 -1의 reward를 부여하고 에피소드를 종료하여 드론이 물체와 너무 멀어지지 않도록 한다. 그렇지 않을 경우 물체2와의 이전 거리에서 현재 거리를 뺀 변수 reward3을 선언하고 거리 변화량에 5를 곱한 값의 reward를 부여한다.

#### ⑥ 드론이 2번 물체를 픽업한 상태

```
float distance_pickdown_2_agent = Vector3.Magnitude(Pickdown_2Trans.position - agentTrans.position);
if(flag_2 == true)
{
    if(distance_pickdown_2_agent > 100f)
    {
        SetReward(-1f);
        Debug.Log("e4");
        EndEpisode();
    }
    else
    {
        float reward4 = predistance_pickdown_2_agent - distance_pickdown_2_agent;
        AddReward(reward4 * 5f);
        predistance_pickdown_2_agent = distance_pickdown_2_agent;
    }
}
```

물체2를 픽업하면 flag\_2가 true가 되므로 드론이 두번째 물체를 픽업한 상태일 때의 reward는 위와 같이 부여한다. 물체2를 배송할 목적지2와의 거리가 100보다 멀어지면 -1의 reward를 부여하고 에피소드를 종료하여 드론이 목적지와 너무 멀어지지 않도록 한다. 그렇지 않을 경우 목적지와의 이전 거리에서 현재 거리를 뺀 변수 reward4를 선언하고 거리 변화량에 5를 곱한 값의 reward를 부여한다.

다음으로 OnCollisionEnter 함수 내의 Reward는 다음과 같이 부여된다.

① 빌딩과 충돌시

```
public void OnCollisionEnter(Collision coll)
{
    if(coll.collider.CompareTag("Building"))
    {
        Debug.Log("Collision with Building");
        SetReward(-10f);
        EndEpisode();
    }
}
```

물체와 목적지의 위치가 빌딩과 가까운 도보에 설정되어 있어 강화학습 수행 시 빌딩과 충돌하는 경우가 많이 발생하여 상대적으로 큰 -10의 reward를 부여한다.

② 도로와 충돌시

```
if(coll.collider.CompareTag("Road"))
{
    Debug.Log("Collision with Road");
    SetReward(-30f);
    EndEpisode();
}
```

물체와 목적지가 바닥에 붙어 있어 학습이 시작되고 드론이 물체를 픽업하는 과정에서 Road에 충돌하는 경우가 가장 잦게 발생했으므로 -30의 가장 큰 리워드를 부여해 비슷한 Action이 반복되지 않도록 한다.

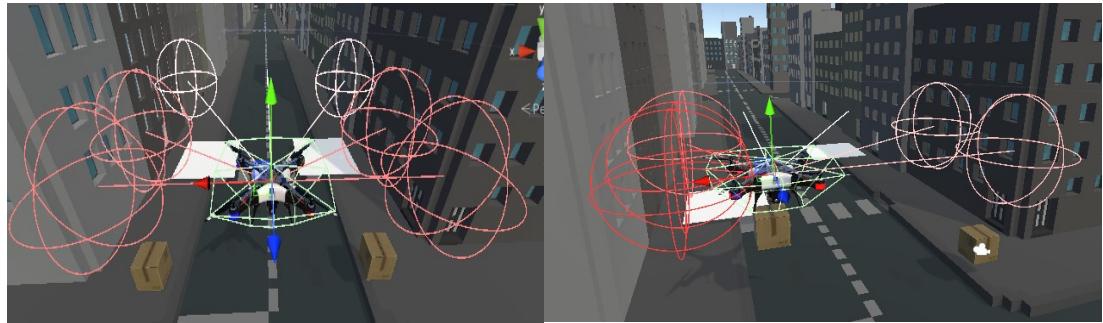
③ 도보와 충돌시

```
if(coll.collider.CompareTag("Pavement"))
{
    Debug.Log("Collision with Pavement");
    SetReward(-1.0f);
    EndEpisode();
}
```

위 두가지 경우에 비해 도보와 충돌하는 경우는 덜 발생했으므로 -1의 적은 reward를 부여하였다.

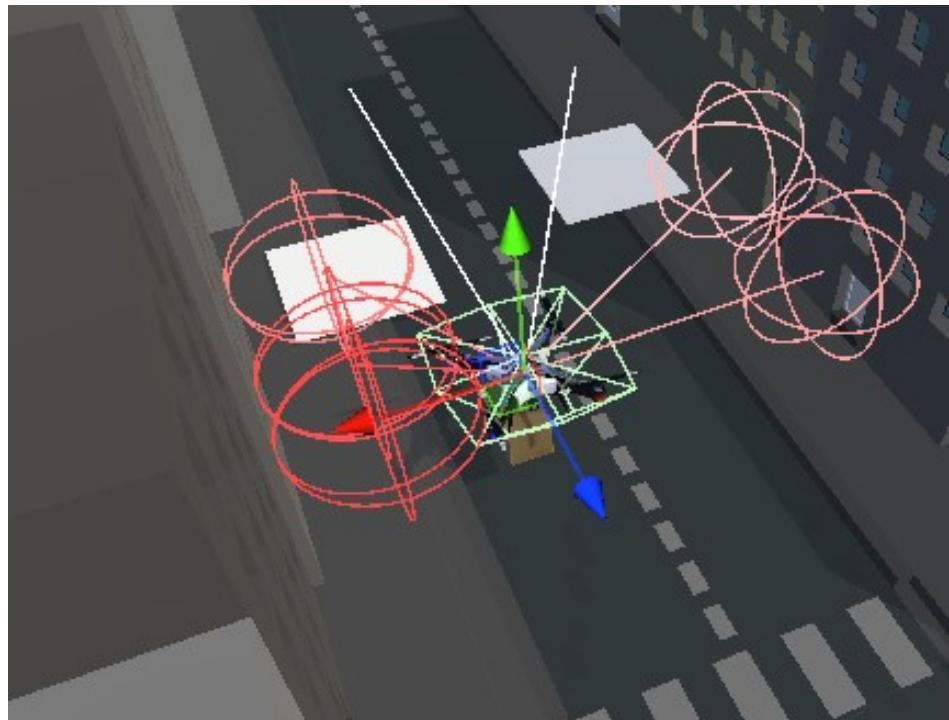
## 2-5 강화학습 수행 결과

앞서 말한것과 같이 우선 city voxel pack과 professional drone free pack 에셋을 이용하여 기본적인 드론의 배송시스템을 구현할 도시환경과 드론을 구현한 뒤 첫번째 목표는 우선 드론이 배송할 물체를 찾아갈 수 있도록 하였다.

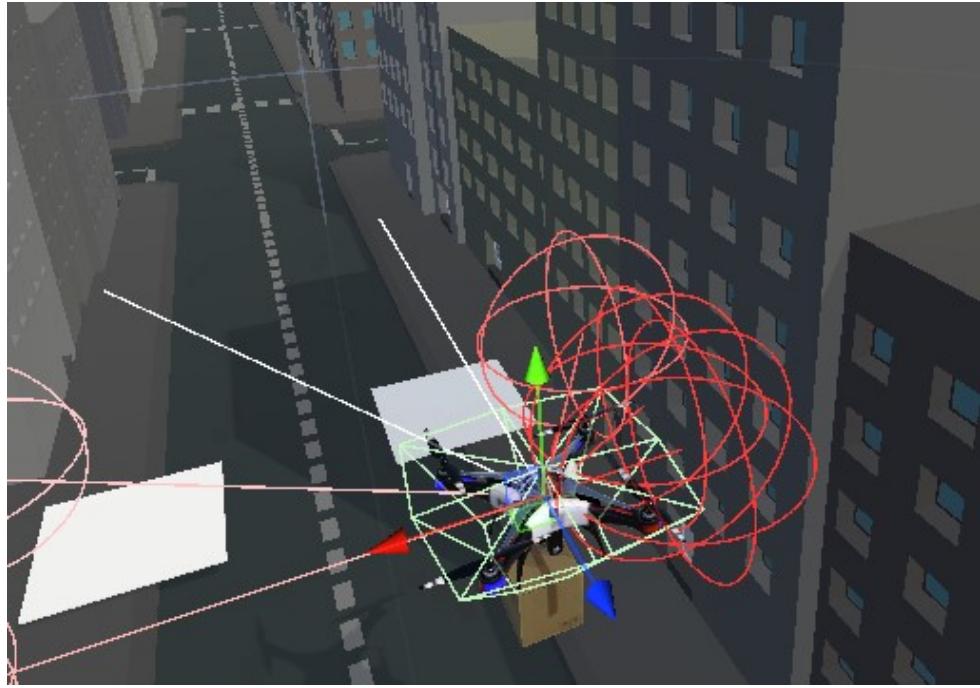


설명한 것처럼 Raycast를 이용하여 해당 물류에 대한 정보를 얻고, 그를 이용해 해당 물체에 접근하여 충돌할 경우 드론이 물류와 연결되어 해당 물류를 들수 있는 동작을 확인할 수 있었다.

이후 드론이 목적지까지 해당하는 물류를 배송할 수 있도록 각 동작에 따른 flag를 이용해 행동을 지정하였고, 아래와 같이 목적지를 향해 움직이는 것을 확인이 가능하였다.



아래와 같이 물류를 집는 행동을 마친 이후에는 해당 물류에 대한 목적지를 찾아서 이동하는 모습을 확인이 가능하였고, 마찬가지로 Raycast의 tag를 등록하여 건물이나 도로 등 장애물로 지정된 오브젝트 들에 대한 회피를 진행하는 것을 확인이 가능했다.

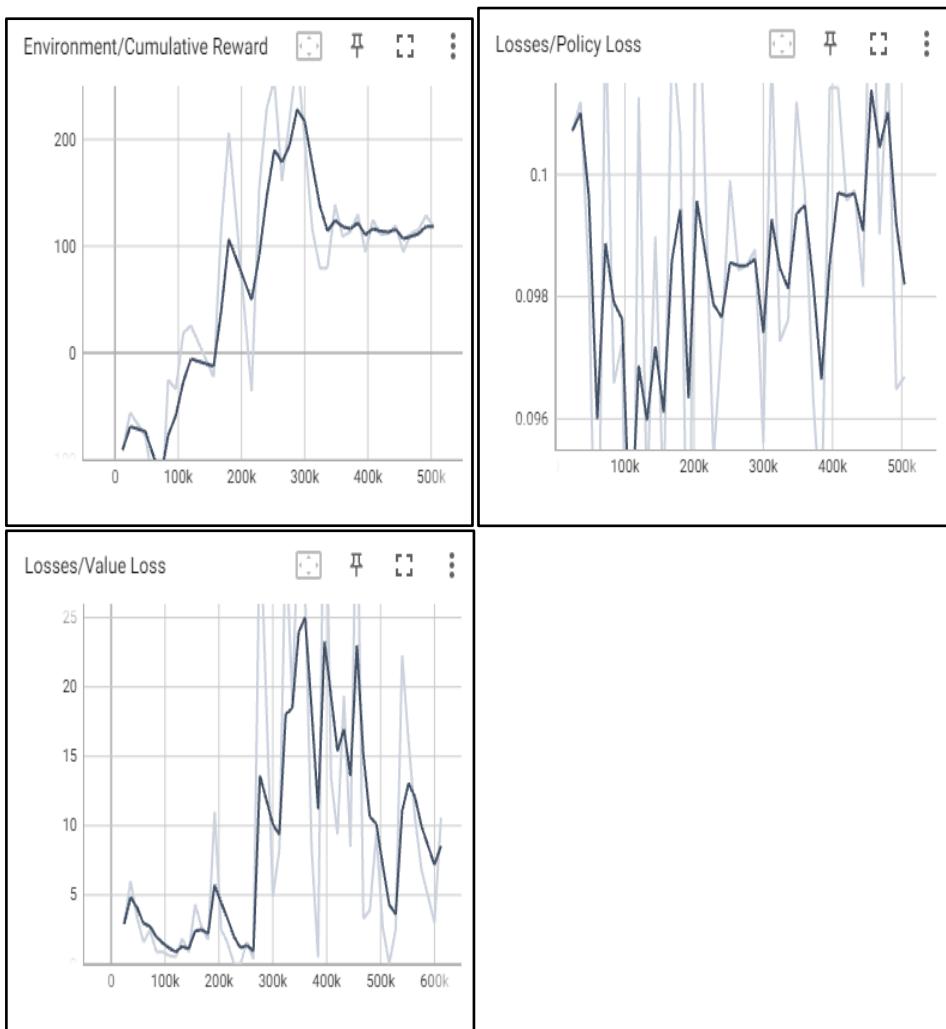


이에 따라 처음 목적으로 삼았던 드론을 이용한 물류의 배달에 대한 구현은 성공하였지만 처음 물류를 배달한 이후에 옆에 보이는 추가적인 물류와 목적지에 대한 배달 또한 구현하여 마치 실제로 드론이 물류 운송을 진행할 때와 같은 상태를 구현하려고 하는 작업에는 실패하였다. 아래의 사진은 실제 학습을 진행하면서 얻은 reward에 대한 값을 cmd창에서 받아온 값으로

```
[INFO] Drone. Step: 156000. Time Elapsed: 1122.282 s. Mean Reward: -22.639. Std of Reward: 43.332. Training.
[INFO] Drone. Step: 168000. Time Elapsed: 1215.137 s. Mean Reward: 117.701. Std of Reward: 89.983. Training.
[INFO] Drone. Step: 180000. Time Elapsed: 1305.106 s. Mean Reward: 206.278. Std of Reward: 48.263. Training.
[INFO] Drone. Step: 192000. Time Elapsed: 1389.244 s. No episode was completed since last summary. Training.
[INFO] Drone. Step: 204000. Time Elapsed: 1482.400 s. No episode was completed since last summary. Training.
[INFO] Drone. Step: 216000. Time Elapsed: 1555.904 s. Mean Reward: -35.994. Std of Reward: 94.741. Training.
[INFO] Drone. Step: 228000. Time Elapsed: 1641.173 s. Mean Reward: 154.032. Std of Reward: 13.487. Training.
[INFO] Drone. Step: 240000. Time Elapsed: 1732.815 s. Mean Reward: 229.769. Std of Reward: 0.000. Training.
[INFO] Drone. Step: 252000. Time Elapsed: 1814.954 s. Mean Reward: 255.371. Std of Reward: 7.015. Training.
[INFO] Drone. Step: 264000. Time Elapsed: 1902.652 s. Mean Reward: 161.107. Std of Reward: 30.070. Training.
[INFO] Drone. Step: 276000. Time Elapsed: 1984.176 s. Mean Reward: 218.040. Std of Reward: 72.680. Training.
[INFO] Drone. Step: 288000. Time Elapsed: 2066.224 s. Mean Reward: 279.270. Std of Reward: 26.950. Training.
[INFO] Drone. Step: 300000. Time Elapsed: 2149.068 s. Mean Reward: 199.629. Std of Reward: 54.628. Training.
[INFO] Drone. Step: 312000. Time Elapsed: 2236.067 s. Mean Reward: 115.810. Std of Reward: 184.474. Training.
[INFO] Drone. Step: 324000. Time Elapsed: 2318.072 s. Mean Reward: 79.411. Std of Reward: 123.692. Training.
[INFO] Drone. Step: 336000. Time Elapsed: 2403.500 s. Mean Reward: 79.399. Std of Reward: 117.714. Training.
[INFO] Drone. Step: 348000. Time Elapsed: 2491.644 s. Mean Reward: 139.022. Std of Reward: 38.903. Training.
[INFO] Drone. Step: 360000. Time Elapsed: 2581.972 s. Mean Reward: 109.211. Std of Reward: 65.952. Training.
[INFO] Drone. Step: 372000. Time Elapsed: 2667.045 s. Mean Reward: 113.298. Std of Reward: 23.764. Training.
[INFO] Drone. Step: 384000. Time Elapsed: 2756.448 s. Mean Reward: 129.730. Std of Reward: 18.158. Training.
[INFO] Drone. Step: 396000. Time Elapsed: 2848.536 s. Mean Reward: 94.336. Std of Reward: 68.949. Training.
[INFO] Drone. Step: 408000. Time Elapsed: 2935.729 s. Mean Reward: 124.681. Std of Reward: 18.992. Training.
[INFO] Drone. Step: 420000. Time Elapsed: 3031.909 s. Mean Reward: 110.669. Std of Reward: 33.697. Training.
[INFO] Drone. Step: 432000. Time Elapsed: 3121.436 s. Mean Reward: 111.996. Std of Reward: 18.869. Training.
[INFO] Drone. Step: 444000. Time Elapsed: 3216.030 s. Mean Reward: 119.080. Std of Reward: 19.233. Training.
[INFO] Drone. Step: 456000. Time Elapsed: 3301.518 s. Mean Reward: 94.214. Std of Reward: 38.061. Training.
[INFO] Drone. Step: 468000. Time Elapsed: 3391.884 s. Mean Reward: 112.164. Std of Reward: 6.456. Training.
[INFO] Drone. Step: 480000. Time Elapsed: 3484.737 s. Mean Reward: 115.408. Std of Reward: 28.791. Training.
[INFO] Drone. Step: 492000. Time Elapsed: 3568.306 s. Mean Reward: 128.789. Std of Reward: 24.561. Training.
```

Mean Reward 값은 시간이 증가할수록 증가하는 추세를 보였지만 안정적이지는 않은 모습을 보였다. 강화학습을 통해 드론이 첫번째 물류를 찾고 해당 물류를 목적지에 도달시키는 목적은 달성하였지만 두번째 물류를 찾고 배달하는 단계에서 문제가 발생하는 것으로 보아 해당 step을 더 단순화 하여 적합한 행동을 찾을 수 있도록 하는 작업이 추가로 필요할것으로 생각된다.

## 2-6 텐서 보드를 이용해 확인한 결과



Tensor Board 상으로 확인 결과 Cumulative Reward 값은 대체로 일정 크기만큼 증가하는 경향을 보였으나 Policy Loss나 Value Loss 값은 불규칙적으로 튀는 것을 보드 상에서 나타나는 것을 확인하였다. 이는 강화학습이 주어진 조건대로 끝까지 진행되지 못하고 중도에 중단되는 Episode 들이 학습 과정에서 지속적으로 발생하면서 불완전하게 진행되면서 생긴 현상으로 보인다.

이 부분 역시 두번째 물류를 찾고 배달하는 단계에서의 적합한 행동을 찾는 작업이 반영된다면 Reward 값이나 Loss 값의 변동이 Board 상에서 이전보다 안정적인 형태로 나타나게 될 것으로 추정된다.

### 3. 결론

본 프로젝트 팀은 PPO 기반의 강화학습을 이용한 드론 배송시스템을 구현하는 것을 목표로 하였고 이를 위해 Asset store와 직접 구현 등의 수단으로부터 환경을 구축하였다.

Unity 상에서 학습을 진행한 결과, Drone이 배송 물류를 인식해 그것을 픽업하여 목표 지점에 핀다운하는 것을 확인하였고 그에 맞게 Mean Reward 값도 주어진 Reward 식에 맞게 일정한 값으로 수렴하는 것을 알 수 있었다. 하지만 학습 도중 장애물 등에 너무 빨리 충돌하는 등의 요소로 중단되는 Episode들로 인해 Reward 값이 더 크게 증가하지 못하고 새로운 Episode로 시작되는 상황이 빈번하게 발생하여 학습 자체의 안정도는 다소 떨어짐을 확인할 수 있었다. 이러한 불안정한 학습의 결과는 텐서 보드 상에서도 loss 값들이 수렴하지 않고 다소 흔다는 점으로도 나타났으며 안정적으로 Drone 배달의 강화 학습을 진행하기 위해서는 Drone의 배송과정에서 소요되는 step을 더 적게 하는 것의 구현이 필요하다고 본 프로젝트 팀은 생각하였다.

실습 결과, 물류 배달이라는 목표에 부합해 정상적으로 동작을 하였으나, Policy Loss나 Value Loss 값은 불규칙적으로 진동하는 것을 보드 상에서 나타나는 것을 확인하였다. 물론 일정하게 수렴하는 그래프가 이상적이겠지만 PPO와 같은 actor-critic RL에서는 Policy net이 방문한 state-action을 value net이 평가한다. 하지만, 이 value net도 하나의 신경망이기 때문에 이 value net의 출력값을 토대로 loss를 계산하는 policy net의 loss가 진동할 수 있지 않을까라는 생각이 듈다.

이번 학기 프로젝트를 진행하면서 물체가 생성되고 고정된 위치에 물류를 배달하는 간단한 작업을 하는것조차 수행하는데 어려움이 많다라는 것을 느낄수 있었고, 머지않은 미래에 드론 배달 시스템이 상용화되어 더 나은 서비스 제공을 받기를 기대해본다.