Journal of Institute of Control, Robotics and Systems (2020) 26(11):955-963
https://doi.org/10.5302/J.ICROS.2020.20.0125 ISSN:1976-5622 eISSN:2233-4335

PPO 기반 강화학습을 이용한 드론의 자율비행

Autonomous Flying of Drone Based on PPO Reinforcement Learning Algorithm

박 성 관, 김 동 환*
(Sung Gwan Park[1] and Dong Hwan Kim[2,*])
[1]Dept. of Mechanical Design and Robot Engineering, Seoul National University of Science and Technology
[2]Dept. of Mechanical System Design Engineering, Seoul National University of Science and Technology

Abstract: In this study, the performance of autonomous flight was analyzed by introducing the PPO method as reinforcement learning for autonomous flight of drones. A simulator based on the dynamics of a drone was produced, and the performance of autonomous flight was confirmed when reinforcement learning was applied to a drone using this simulator. After that, the possibility of autonomous flight was confirmed by applying the PPO algorithm to the actual drone. Also, a lightweight embedded PC was attached to the drone to perform independent calculations to simultaneously construct obstacle avoidance and path planning.

Keywords: Autonomous drone, Simulator, PPO(Proximal Policy Optimization), Reinforcement learning

I. 서론

현재 조기경보, 긴급 서비스, 뉴스 레포팅, 배달, 모니터링, 게임용, 스포츠, 농업용 등에 수많은 전용 드론 및 무인 비행체가 현재 사용되고 있고, 더욱 다양한 범위에서 사용될 것으로 보고 있다[1]. 또한 여러 다양한 임무를 저비용과 고효율로 수행하기 위해서는 드론이 자율적으로 비행을 수행해야 할 필요가 있다. 드론과 무인비행체의 자율비행은 임무수행을 위하여 측위(localization), 센싱(sensing), 경로계획(path planning), 그리고 법칙(rule) 기반의 충돌회피와 제어(control)를 실행하게 된다[2]. 그러나 앞의 방식은 사람과 같은 동적 장애물이나 기후 등의 환경의 변화로 인해 생성되는 불확실성에 취약하므로 이에 대비하기 위한 해법이 필요하게 된다. 이러한 불확실성 문제를 해결하기 위해 최근 심층 강화학습이 큰 관심을 받고 있다.

기계 학습의 한 영역인 강화 학습은 특정 환경에서 사용할 수 있도록 정의된 에이전트(agent)가 현재의 상태(state)를 인식 후 선택할 수 있는 행동들 중 보상(reward)을 최대화하는 행동순서 또는 행동(action)을 선택하는 방법이다. 이를 기반으로 심층 강화학습을 통해 실제 하드웨어를 강화학습으로 제어하여 장애물 회피(obstacle avoidance)와 경로계획을 동시에 수행하여 복잡한 문제의 해결을 모색하는 연구들이 최근 활발하게 진행되고 있다.

강화학습을 적용한 비행체의 자율비행에서는 model-free prediction 심층 강화학습을 기반으로 하는 MDP (Markov Decision Process) framework을 사용하여 quadcoptor의 모터 throttle을 제어하여 드론의 전진 및 회전을 성공시킨 예[3], LQR (Linear Quadratic Regulator) 컨트롤과 MDP framework를 이용한 policy 컨트롤을 비교하며, 원하는 높이로 드론을 띄운 사례[4], 모션 탐지기를 이용해 지속적으로 드론의 상태를 관측하며 위치제어를 구현한 예가 있다[5]. 이후 카메라로 바닥 이미지를 이용하는 강화학습으로 높이와 위치를 제어하거나(장애물 회피를 하지는 않는다)[6-8], 비슷한 방식으로 전방 이미지의 변화를 학습하여 그것을 기반으로 드론의 위치를 유지시키거나, 변경할 수 있도록 한 방식도 있고[9] 기존 위치 컨트롤 방식에 독자적인 LSPI (Least Square Policy Iteration) 알고리즘을 추가하여 드론의 위치를 컨트롤하였고, 이를 기반으로 주행 및 지면 착륙을 구현하였다[10].

이후 Microsoft의 Airsim[11]을 기반으로 하는 시뮬레이터에서 Pixhawk[12]의 hardware-in-the-loop를 이용하여 이미지와 여러 센서 데이터들을 학습시켜 장애물 회피와 경로계획을 구현하였고[13], Unity[14] 시뮬레이션에서는 앞의 학습 매커니즘을 자세제어와 탐사 알고리즘으로 나누어 학습하고 이를 2개의 신경망을 동시에 사용하여 장애물 회피와 탐사를 통한 경로계획을 동시에 구현하는 프로그램을 선보였다[15]. 또한 시뮬레이터에서 여러 종류의 강화학습 알고리즘을 사용하여 PID 제어를 대체하는 실험을 진행하고, 최종적으로는 PPO(Proximal Policy Optimization) 알고리즘을 기반으로 각도와 위치 제어를 구현하는 방식을 보였다[16-19].

*Corresponding Author
Manuscript received August 10, 2020; revised September 10, 2020; accepted October 6, 2020
박성관: 서울과학기술대학교 기계설계로봇공학과 석사과정(qkrtjdrhks21@gmail.com, ORCID 0000-0002-3481-9255)
김동환: 서울과학기술대학교 기계시스템디자인공학과 교수(dhkim@seoultech.ac.kr, ORCID 0000-0002-7977-6981)
※ This paper was supported by Korea Institute for Advancement of Technology(KIAT) grant funded by the Korea Government(MOTIE)(P0008473, HRD Program for Industrial Innovation)

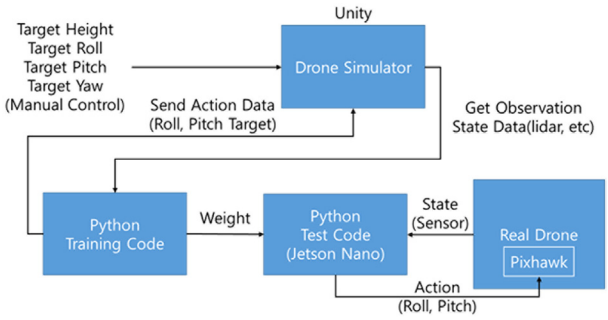


그림 1. 전체 연구의 흐름도.

Fig. 1. Flow chart of whole research.

그러나 앞의 연구에서는 시뮬레이터에 한정해 실험을 진행하거나 실제 드론을 사용할 때 PC에 데이터를 송수신하며 학습 및 실험을 진행했다. 이 중 연구가 진행된 강화학습을 이용한 실제 드론 비행 연구의 경우, 구동 시 드론이 컴퓨터와의 통신에 종속되어 근거리에서만 동작한다는 단점이 생긴다. 이에 본 연구에서는 그림 1과 같은 흐름으로 경량 embedded PC를 부착한 드론이 자체적으로 연산하여 장애물 회피와 경로계획을 동시에 구성할 수 있게 하였고, 외부 통신 없이 드론의 내장 알고리즘만으로 모든 활동을 가능하게 하였다. 또한 경량화를 위해 본 논문에서는 카메라 데이터 대신 상대적으로 데이터 연산이 적은 lidar와 UWB 센서를 사용한다. 또한 embedded PC의 연산 부담을 낮추기 위해 일반적인 자세 제어 등의 제어는 기존 상용 제어기 Pixhawk를 사용하였다.

II. 심층 강화학습 기법 적용

1. 가치기반 심층 강화학습

강화학습의 목적은 환경과 상호작용을 이루면서 높은 보상을 획득하는 것이다. 여기서는 보상을 정의하는 두가지 함수가 있다. 이 중 하나는 앞으로 받을 것이라 예상되는 보상의 기댓값을 정의하는 가치함수(value function)이다. 이 가치 함수 $v(s)$는 agent가 t시점에 상태 데이터 s, 즉 관찰된 데이터를 얻어 그 때 받았던 일련의 보상의 합을 구하는 개념으로 구성되어 있다. 여기에 중요성이 덜해지는 과거의 데이터에 감가율을 곱해 식 (1)과 (2)와 같이 나타낸다.

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1}$$

$$v(s) = E[G_t | S_t = s] \tag{2}$$

그러나 가치 함수는 그 상태로 지향하는 행동이 고려되지 않아 상태 가치함수(state-value function)라 표기되고, 행동 a를 같이 고려하여 구한 기댓값을 Q-function (Quality function)이라 하며 식 (3)과 같이 표기된다.

$$Q(s,a) = E[G_t | S_t = s, A_t = a] \tag{3}$$

Q-function은 식 (4)와 같이 상태 s와 행동 a로 사용하는 것으로 높은 보상을 기대하는 함수이다. 그리고 Q-function는 그 다음 상태 s'와 행동 a'를 받으며 Bellman 최적화 방정식을 통해 지속적으로 갱신되어 높은 보상을 얻게 된다.

$$Q(s,a) = r + \gamma \max_{a'} Q_{\pi}(s',a') \tag{4}$$

위 가치 함수를 아래 식과 같이 신경망 파라미터 θ를 기반으로 하는 neural network 신경망으로 대체하는 알고리즘을 DQN(Deep-Q-Network)[16]이라 하며 식 (5)와 같이 목표 Q-function과의 cost function을 계산하고 이를 기초로 off-policy방식 등을 복합하여 신경망을 갱신한다.

$$Cost = [Q(s,a;\theta) - r(s,a) + \gamma \max_a Q(s',a;\theta)]^2 \tag{5}$$

2. 정책기반 심층 강화학습

정책은 $\pi_\theta(a|s)$로 표현하며, 상태를 받아 정책 신경망 파라미터 θ를 기반으로 연산을 하여 행동을 내보내는 함수이다. 즉 정책 기반 강화학습은 상태에 따라 행동을 선택한다. 따라서 발생하는 가치함수의 변화를 토대로 파라미터 θ를 갱신하게 되고, 갱신 방향은 더 높은 가치함수를 얻는 gradient ascent 방향으로 향하게 된다. 이 과정을 PG(Policy Gradient)라 하고 식 (6)과 같이 표현한다.

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta) \tag{6}$$

$\nabla_\theta J(\theta)$는 목표함수로, PG에서는 식 (7)과 같이 정의한다[17].

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q_\pi(s,a)] \tag{7}$$

여기에서 Bias를 줄이기 위해 Baseline $V(s)$를 뺀 advantage function, 즉 $A(s,a) = Q(s,a) - V(s)$으로 바꾸어 식 (8)로 대체되고, 이를 Monte-carlo PG라 부른다[18].

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) A] \tag{8}$$

앞의 PG는 SPG(Stochastic Policy Gradient)라고도 하는데, 이 값을 continuous한 행동 값으로 연산하도록 구성한 알고리즘을 DPG[19]라 하고, 여기에 앞의 DQN과 actor-critic을 결합시키면 model-free, off-policy, actor-critic 알고리즘인 DDPG[20]가 되고 여기서 distributional 개념이 추가로 반영되면 D4PG[21]가 된다. Actor-critic은 알고리즘 내의 정책과 가치함수를 각각 actor와 critic이라 할 때 상호 갱신에 도움을 주게 된다. 정책을 갱신할수록 가치함수의 값도 높아지게 되고 높아진 가치함수의 값을 기반으로 정책 또한 더 좋은 방향으로 주고받으며 갱신하기 때문이다.

3. TPRO (Trust Region Policy Optimization)와 PPO (Proximal Policy Optimization)[22,23]

TRPO와 PPO는 trust region, 즉 범위 제한을 만들어 PG에서 발생하는 정책의 과한 갱신으로부터 일어나는 문제를 방지하여 안정적인 신경망의 갱신을 가능하게 하는 알고리즘이다. TRPO와 PPO에서는 기존 정책기반 강화학습과 같이 정책 신경망 $\pi_\theta(a|s)$가 환경과 상호작용하여 메모리에 데이터를 모은다. 해당 알고리즘에서는 timestep마다 보상값을 모은 후, 자신이 정한 A(advantage)를 계산하고 importance sampling을 곱한 값을 사용하여 신경망 갱신을 위한

loss를 식 (9)와 같이 표현한다.

$$L(\theta) = E_{s \sim p^{\pi_{\theta_{old}}}, a \sim \pi_{\theta_{old}}}[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} \hat{A}_{\theta_{old}}(s,a)] \quad (9)$$

$\hat{A}$은 예측된 보상에 관한 항이고, 해당 논문에서는 앞에서 정의한 $Q-V$값을 A로서 사용하였다. 이후 위에서 연산한 Loss의 범위를 제한하는 surrogate function을 PPO의 전신이 되는 TRPO에서는 KL-divergency를 사용하여 식(10)과 같이 표현한다.

$$E_{s \sim p^{\pi_{\theta_{old}}}}[D_{KL}(\pi_{\theta_{old}}(\cdot|s) \| \pi_\theta(\cdot|s)] \leq \delta \quad (10)$$

본 논문에서는 위 surrogate function을 간단히 하고 기존 TRPO와 유사한 성능을 발휘하는 PPO 알고리즘을 사용하였다. TPRO에서 표현한 위 수식과 달리 PPO에서는 제한 범위를 식 (11)의 $r_t(\theta)$를 상수 ϵ(대체로 0.2)로 제한하였고, 그에 따른 L^{clip}를 식(12)와 같이 표현한다[25].

$$r_t(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}, r_t(\theta_{old}) = 1 \quad (11)$$

$$L^{clip}(\theta) = \hat{E}_t[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t] \quad (12)$$

이후 일반적인 신경망을 갱신하듯 메모리에 쌓인 데이터를 batch만큼 잘라 지정된 데이터를 loss를 이용하여 순서대로 K회 갱신하게 되고, 위로부터 지금까지의 과정을 Algorithm 1과 같이 지속적으로 반복한다.

Algorithm 1 PPO, actor-critic style[25]

for iteration=1,2,...**do**
  for actor=1,2,...,N **do**
    Run policy $\pi_{\theta_{old}}$ in environment for T timesteps
    Compute advantage estimates
    $\hat{A}_1,...,\hat{A}_T(\hat{A}_t = Normalize(Q_t(s,a)) - V(s_t))$
  end for
  Optimize surrogate $L^{clip}(\theta)$,with K epochs and minibatch size M < NT
end for

위 알고리즘을 바탕으로 training 코드를 실행하기 위해 PPO의 제한 범위 ϵ과 같은 상수들을 지정해 줄 필요가 있다. 본 논문에서는 다음 표 1과 같이 상수를 지정했다. 표 1의 batch size와 buffer size는 좀 더 깊은 학습이 가능하도록 사용하는 컴퓨터의 최대 memory 용량까지 끌어올린 값이고, ϵ은 PPO의 surrogate function 값이며, learning rate는 1회의 갱신 정도를 뜻하며, γ는 앞의 가치함수에서 언급한 이전 보상의 감소율을 뜻한다.

이후에는 위 상수들을 이용한 알고리즘과 loss를 이용해 신경망을 갱신하고 데이터를 모으는 과정을 반복하여 보상을 최적화하게 된다. training 코드에서는 위와 같은 actor-critic 기반 PPO 알고리즘을 사용하게 되는데, 이 알고리즘

표 1. PPO 알고리즘에서 내부 상수.

Table 1. Internal constants in PPO algorithm.

지정 상수	
Batch size	4096
Buffer size	40960
ϵ	0.2
Learning rate	3*10-4
γ	0.99

에서는 학습할 때 actor와 critic 신경망을 2개 갱신하는 과정을 거친다. 일반적으로 신경망의 연산은 높은 연산 능력을 필요로 하나 해당 신경망을 경량화하여 갱신하기 위해 State로부터 지정한 개수만큼의 상태 데이터를 받아들이고, linear regression과 tanh activation function을 순서대로 번갈아 3회 거쳐 64개와 32개만큼의 node를 통하는 가벼운 연산을 거치도록 설정하였다. 또한 일반적인 신경망에서 사용하는 fully connected layer가 없으므로 학습과 테스트 과정의 데이터 연산이 2중으로 경감된다.

4. SAC(Soft Actor Critic) 과의 비교[24]

SAC는 off-policy 알고리즘이며 이름과 같이 actor-critic구조이고 알고리즘의 exploration을 보완하기 위해 보상에서 정책에 대한 entropy measurement $H(\pi_\theta(\cdot|s_t))$를 추가한 알고리즘이다. SAC 알고리즘은 2020년 초에 소개된 최신 알고리즘인 만큼 PPO와 비교하여 신경망의 갱신 대비 학습 효율이 대체로 높아 데이터 효율이 좋다는 장점이 있으나, 신경망 갱신의 횟수가 늘고 연산 자체의 복잡성으로 인해 연산 부하가 좀 더 늘었다는 단점이 있다. 이는 PPO와 SAC 2가지 알고리즘 모두 off-policy, 즉 환경과의 상호작용 데이터를 메모리에 모은 후 Update를 진행할 수 있기 때문에 그림 2와 같이 agent 다수를 동시에 사용하는 경우 시뮬레이션의 효율을 높일 수 있다는 점이다. 한편 다중 agent를 사용하여 학습을 진행할 때 PPO에 비해 SAC는 많은 연산 처리로 인한 속도 저하 문제로 PPO보다 학습 속도가 늦어지는 경향이 있으므로 본 연구에서는 주요 학습 알고리즘으로 PPO를 사용하게 되었다.

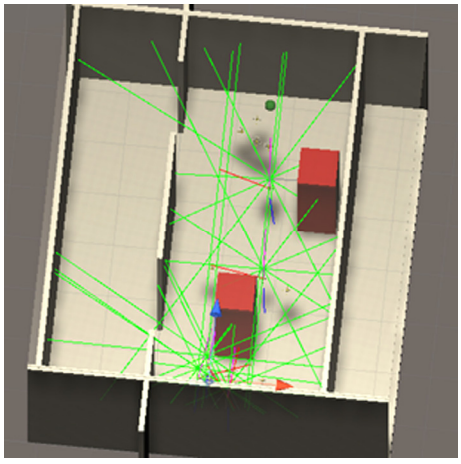


그림 2. ML-agent로 여러 객체를 사용한 학습.

Fig. 2. Multi-agent learning by ML-agent.

5. 시뮬레이션을 통한 학습 과정

Unity는 자체 소프트웨어를 강화학습의 환경으로 삼고 그 외 학습을 Python에서 신경망의 학습을 진행할 수 있도록 하는 서비스 ML-Agent(Machine Learning-Agent)를 제공한다. 이를 통해 Python으로 구성한 코드의 환경 테스트를 Unity 시뮬레이터에서 진행할 수 있게 된다. 2장에서 제작한 드론 시뮬레이터는 컨트롤러로 동체를 조종하지만, 강화학습 코드는 관측한 상태 데이터를 연산하여 행동, 즉 컨트롤 데이터를 시뮬레이터에 사용한다. 즉 그림 3과 같이 기존 사람이 넣던 입력을 인공지능이 대신해 넣게 된다.

그림 4는 관측 도식이다. 실제 관측에는 lidar 거리데이터, Pixhawk의 IMU (Inertial Measurement Unit) 등 센서 데이터, UWB (Ultra Wide Band) 센서의 위치 데이터를 조합해 사용할 것이기 때문에 시뮬레이터에도 최대한 유사한 데이터를 구현하였다. 측정 데이터는 위에서부터 순서대로 빛을 조사하는 Unity의 Raycast 메서드를 통해 빛의 길이로서 구현한 2d lidar의 거리 데이터, 드론의 위치, 이동 방향(roll, pitch, yaw 기반), 마지막으로 목표 지점까지의 방향과 거리 데이터이다. 이후 위 데이터를 algorithm 2와 같이 가공하여 신경망에 직접 투입한다. 그 이유는 신경망에 데이터를 삽입할 때 -1에서 1 사이의 값을 넣어야 학습에 bias를 없앨 수 있기 때문이다.

그림 5는 행동 도식이다. 행동은 그림의 드론 모형과 같이 target roll와 target pitch를 약 11.5도의 각도 제한을 둔 상태로 일정 속도 이하로 속도를 변화시켜 드론이 x와 y축으로 이동할 수 있게 하였다. 주위의 거리를 알 수 있는 lidar 센서는 yawing action이 들어가면 혼란을 일으킬 수

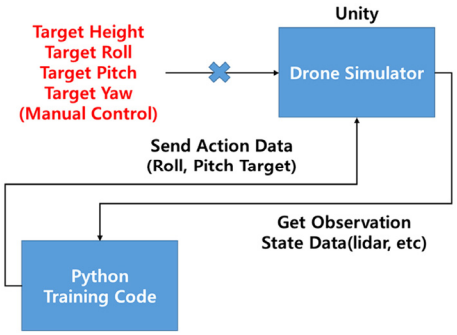


그림 3. 수동 입력을 제외한 시뮬레이터의 흐름도.

Fig. 3. Flowchart of simulator without manual control input.

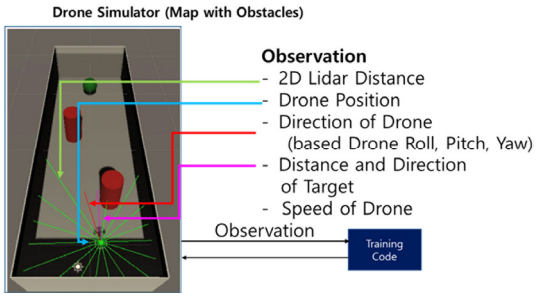


그림 4. 시뮬레이터의 관측(상태 수집).

Fig. 4. Observation of simulator (states earning).

Algorithm 2 Manufactured observation data

2d lidar range data :
If (distance<Lidar Maximum range)
range data=1-distance/lidar maximum range
else range data=-1
Direction : roll/180, -1*pitch/180 (degree/180)
Position : x/limit of x, y/limit of y
Speed : speed/max speed
Distance 2*distance/(1+|distance|) -1
Angle between target and direction

있기 때문에 드론에서 통상적으로 roll, pitch, yaw 및 throttle 제어를 할 수 있음에도 yawing을 제외하였다. 또한 throttle은 학습의 확인으로서 현재 환경에서는 불필요하기 때문에 제외하였다.

마지막으로 그림 6의 보상(reward)은 algorithm 3와 같이 정의하였다. 1번은 target과 direction의 일치성과 속도의 곱, 2번은 목표와 가까울수록 증가하는 보상, 3번은 목표에 도달했을 때 받는 보상이다. 처벌(penalty) 또한 1번은 물체와 근접했을 때, 2번은 빠른 목표 도달을 위한 비행 시간동안 지속적인 처벌, 3번은 벽과 부딪혔을 때, 4번은 제한 시간 동안 목표에 도달하지 못했을 때의 처벌이다.

Algorithm 3 Manufactured data of reward and penalty

Reward
1. Similarity between direction and target * speed
-> (1-)*speed/10
2. Distance between target and drone
x = 1/(goal position - drone position) -> 10*x/1+|x|
3. Target reach -> 8

Penalty
1. Proximity penalty
-> If (lidar range < proximity range)
-10*(1-distance/lidar maximum range)
2. Flight time penalty -> -0.6
3. Collision -> -8
4. Time Out -> -5

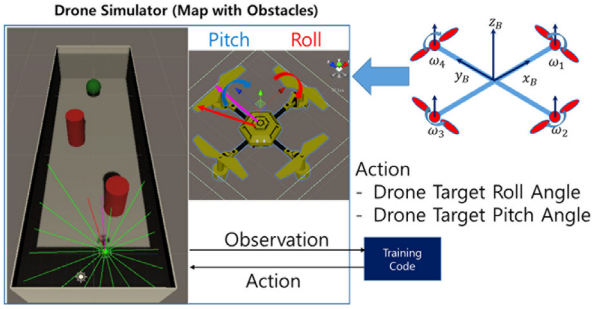


그림 5. 시뮬레이터의 행동.

Fig. 5. Action of simulator.

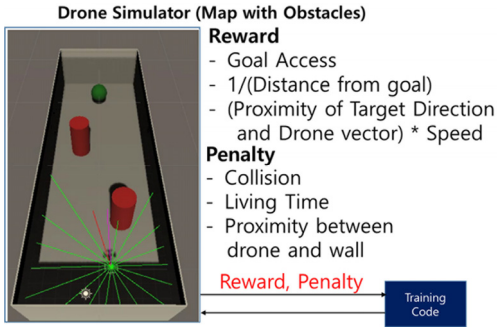


그림 6. 시뮬레이터의 보상과 처벌.

Fig. 6. Reward and penalty of simulator.

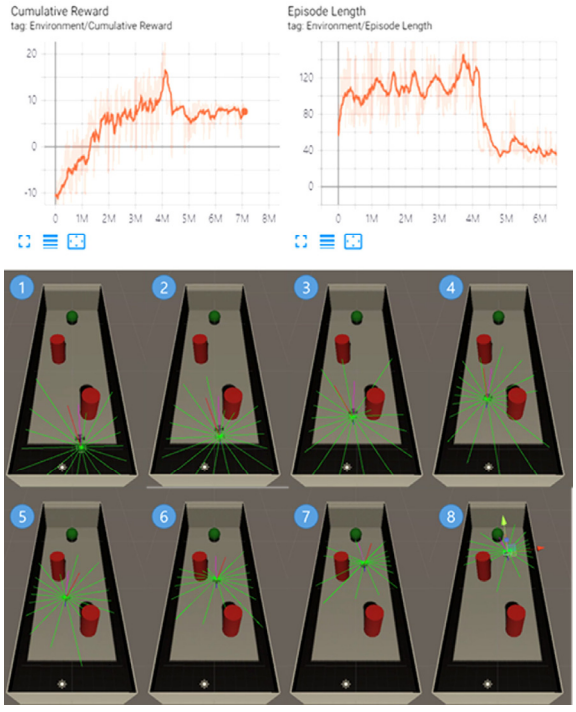


그림 7. 2d 드론 시뮬레이터의 누적 보상, 실행 길이 및 진행과정.

Fig. 7. Cumulative reward, episode length of 2d drone simulator and its proceedings.

그림 7은 앞의 수식을 토대로 학습한 결과이다. 학습이 완료되어 쌓이는 보상, 즉 가치함수는 일정 수치가 유지되고, 비행시간에 처벌을 주었기 때문에 agent가 살아있는 시간이 점점 줄어들게 된다. 그림 7의 아래는 시뮬레이터를 재생한 후 순서대로 1초마다 찍은 사진으로, 드론이 장애물인 붉은색 기둥과 주위 벽은 피하고 목표인 초록색 구로 전진하는 시뮬레이션 결과다.

III. 3d 시뮬레이터

앞에서 언급한 시뮬레이터는 행동 부분에서 target height (z_d)를 제외하여 2d의 움직임을 보였으나, 2d 시뮬레이터에서 현재 데이터만으로 학습이 가능함이 확인되었기 때문에 추가로 그림 8과 같이 행동에 z_d를 포함하여 3d로 동작할 수 있는 학습을 진행하였다.

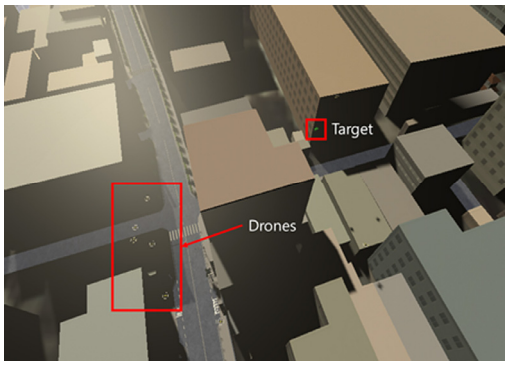


그림 8. 강화학습을 이용한 도심 내 드론 비행.

Fig. 8. Drones flying at city using reinforcement learning.

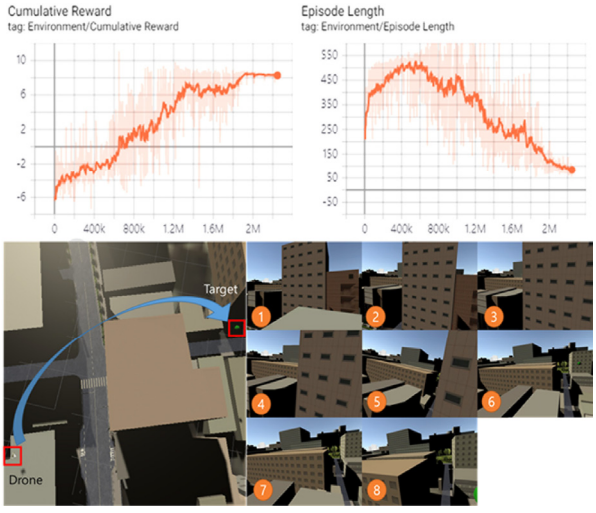


그림 9. 3d 드론 시뮬레이터의 누적 보상, 실행 길이 및 진행과정.

Fig. 9. Cumulative reward, episode length of 3d drone simulator and its proceeding.

throttle action을 포함하여 2D 시뮬레이터에서 고도가 추가되었기 때문에 평면으로만 위치하던 2D 라이다 센서에 추가로 lidar lite 고도 거리 센서를 가정한 고도 데이터를 추가하여 observation data가 4개로 증가했다. 3D 드론 시뮬레이터 또한 cumulative reward가 점차 증가하며 episode length가 감소하는 것을 확인할 수 있고, 그림 9의 상단과 비슷한 양상을 보이는 것을 알 수 있다. 그림 9의 하단은 시뮬레이터를 재생한 후 순서대로 3초마다 찍은 사진으로, 드론이 장애물인 건물을 피하고 목표인 초록색 구로 전진하는 시뮬레이션 결과를 보여주고 있다.

시뮬레이션에서 학습을 진행할 때, 다수의 드론이 무작위 위치에서 생성되고, 장애물 회피와 간단한 경로계획을 구현한 것을 볼 수 있다.

IV. 강화학습 모델의 실제 적용

실험에 사용할 드론은 위 시뮬레이터에서 관측에 사용했던 데이터들을 실제로 모을 수 있도록 그림 10과 같이 2d lidar, distance sensor, Pixhawk 내부 센서 데이터 및 UWB의 위치 데이터를 사용한다. 앞에서 드론 시뮬레이터를 거쳐

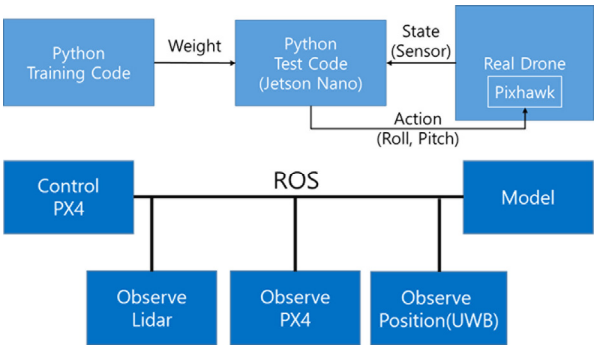


그림 10. 실제 드론의 데이터 흐름과 사용 센서.
Fig. 10. Data flow of actual drone and its sensors.

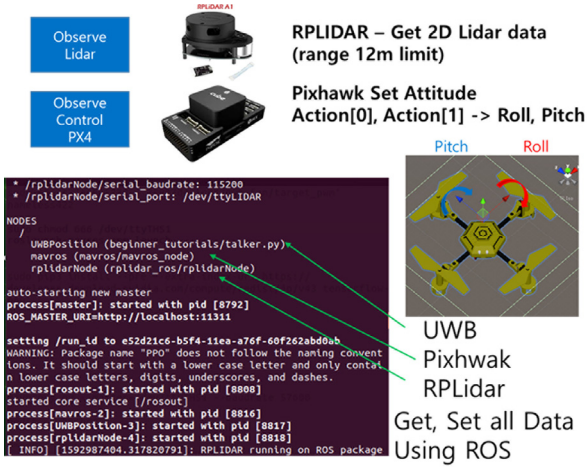


그림 11. 사용 센서 및 ROS 기반 프로그램 환경[25][26].
Fig. 11. Sensors and program environment based ROS[25][26].

파이선 프로그램에서 model을 생성하면, 해당 모델을 기반으로 Pixhawk상에 드론 조종 명령을 보내게 된다.

실제로 사용하는 센서들 중 그림 11과 같이 2d lidar센서는 Slamtech사의 Rplidar A1이다. Rplidar는 5.5 hz 주기로 회전하며 1도마다 적외선 거리 데이터를 수신하여 embedded PC에 송신한다. 다른 센서 및 제어기 Pixhawk는 상용 드론 제어기로, 내부에 IMU가 들어있어 자세, 속도, 가속도, 온도, 고도의 측정이 가능하며, 기체를 특정 방향, 위치로 회전시키거나 움직이기 위해 필요한 모터 Throttle에 대한 PWM명령 등을 각도 입력 등의 커맨드로 대체할 수 있게 해주는 middleware 역할을 한다. 예를 들어, 적절한 각도나 위치 명령어로 Pixhawk가 공중에서 호버링을 하거나, 위치를 변경할 수 있다. 현재 모든 센서의 데이터는 후술할 Jetson nano embedded PC에서 ROS를 사용해 받아들이고 목표로 하는 Roll 및 Pitch 명령을 Pixhawk가 송신하여 드론를 제어한다.

UWB 센서는 지정한 센서간의 거리를 파악할 수 있는 센서다. 드론에 부착된 센서와 특정 위치에 설치된 3개 이상의 센서가 있으면 그림 12의 삼각측량을 사용해 드론의 위치를 측정할 수 있고, Noise를 줄이기 위해 gradient descent 알고리즘으로 1차 최적화를 거치고 디지털 필터를 이용하여 더욱 정확한 위치를 갱신한다[25]. Pixhawk 또한 위치를

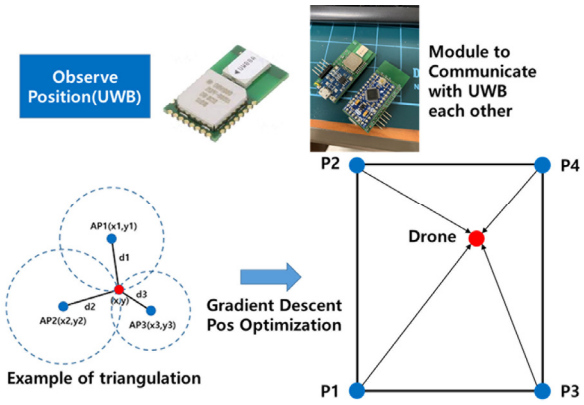


그림 12. UWB 센서와 삼각측량을 활용한 위치측정[27].
Fig. 12. Triangulation positioning using UWB sensor[27].

GPS나 다른 장비로부터 수신할 수 있지만, 본 논문의 드론은 실내에서 비행하기 때문에 UWB를 이용하여 드론의 위치를 알 필요가 있다.

본 실험에서는 센서들의 데이터를 받아 신경망에서 연산하기 위한 embedded PC가 필요하게 되고, GPU를 사용할 수 있는 Jetson nano를 추가 사용한다. Jetson nano는 최근 나온 라즈베리 파이 4와 흔히 비교되는데, 성능과 가격(약 $100)은 비슷하지만 장점은 내부에 CUDA를 지원하는 GPU가 존재한다는 것이다. 또한 성능은 기존의 Jetson TX 모델과 비교해 느리지만 크기가 작고 가볍기 때문에 배터리를 사용하는 드론에 장착하여 용이하게 비행할 수 있다는 장점이 있다.

드론에 앞의 구성들을 그림 13과 같이 구성한 후 시뮬레이션과 실제 드론 사이의 데이터 유사성에 관한 실험을 진행하기 위해 그림 14와 같이 환경을 조성하였다. 실제 드론과 시뮬레이터 드론과의 roll, pitch와 yaw 방향을 맞추어준다. 시뮬레이터에서는 그림 14와 같이 기존 관측 데이터에서 이동 속도와 위치를 고정한 채 진행했고, 실제 드론 또한 시뮬레이터의 조건과 맞추어 실험을 진행하였다. 이 상태에서 시뮬레이터의 드론은 주위에 아무 장애물이 없기 때문에 장애물을 피하는 동작은 없어지고 구체로 직진하는 움직임만을 보이게 된다. 예를 들어 그림 14와 같이 target 지점이 (0,4)로 설정되어 방향이 드론 기준 후방 좌측일 때 drone의 동체는 roll이 양수, pitch가 음수인 방향으로 움직인다. 위 조건에서 드론의 위치는 시뮬레이터와 실제 UWB 데이터 모두 (1.7, 5.7) 좌표에 고정되어 있다.

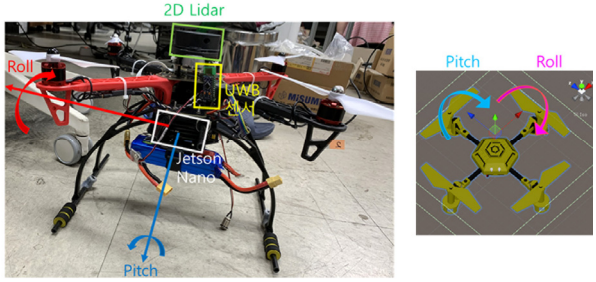


그림 13. 실제 드론과 시뮬레이터 드론 맞춤.
Fig. 13. Alignment of real and simulation drone.

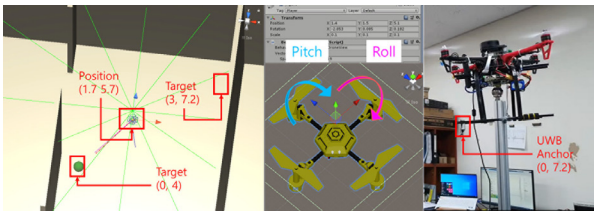


그림 14. 목표 지점 (0,4)와 (3,7)에서의 실제 드론과 시뮬레이터의 환경 설정.

Fig. 14. Environment setup between real drone and sumulator for targets of (0,4) and (3,7).

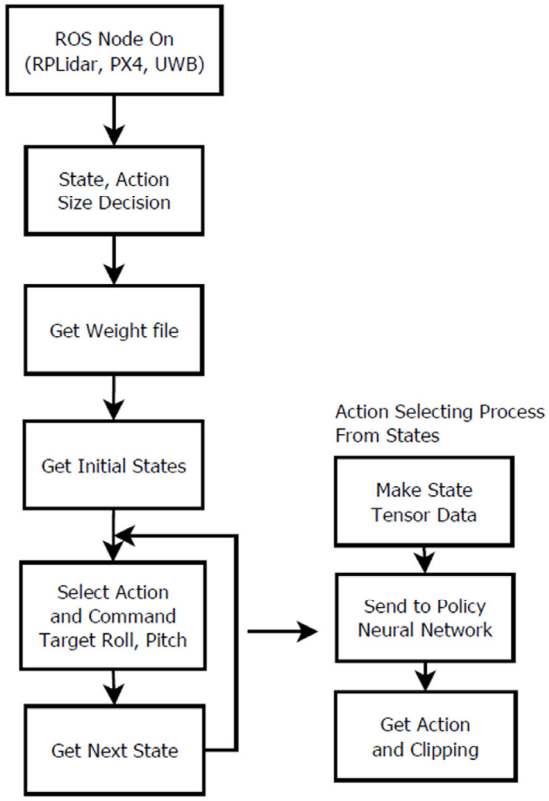


그림 15. 드론 구동 코드의 흐름도.

Fig. 15. Flow chart of drone control.

그림 15는 Unity에서 학습할 때 사용하는 코드에서 학습에 관한 부분을 삭제하고, state 명령을 ROS를 통해 받아들인 센서의 데이터로, 행동 명령을 ROS의 Target roll 각도와 target pitch 명령으로 전환한 Python 테스트 코드의 flow chart다. 프로그램 기동 시 ROS를 가동하여 센서 데이터를 수신할 준비를 마치고, 방어, 행동의 크기를 정의한 후 가중치 파일을 받아들여 학습된 데이터를 기반으로 행동 데이터를 연산할 준비를 마친다. loop문으로 돌입하기 전에 초기 상태를 외부 센서로부터 받아들이고 이후엔 loop문에서 행동을 배출한다. 이 때 사고 방지 및 제어의 용이성을 위해 시뮬레이션과 드론 모두 행동은 2.865도로 크기를 제한한다. 이후 다음 state를 받아들이는 과정을 20Hz로 반복한다. 이 과정에서 신경망을 통해 연산되는 과정을 flow chart의 우측에 표기했다.

V. 실험 결과

시뮬레이션 데이터와 실제 드론 구동 데이터를 동일한 target (0,4)와 시간의 길이를 기준으로 비교하는 실험을 진행했다. 그림 16과 그림 17의 그래프는 드론의 Python 테스트 코드와 시뮬레이터를 79 step씩 실험한 결과이다. 두 그림의 파랑색 실선 action[0]은 Roll 각도를 의미하고, 주황색 실선 action[1]은 pitch 각도를 의미한다. 목표가 (0,4)일 때 roll이 양수이고 pitch가 음수인 것을 확인할 수 있고, 실제 드론이 그림 18과 같이 지속적으로 한 방향을 향해 비행하는 것을 확인할 수 있었다. 이와 반대 방향으로 target의 위치가 (3,7)일 때 드론은 전방 우측으로 향해야 한다. 즉 시뮬레이터에서 평균적으로 roll이 음수, pitch가 양수를 출력해야 하고, 이를 그림 17의 시뮬레이션 데이터에서 값을 확인할 수 있었으며, 실제 드론의 데이터에서도 부합하는 것을 확인할 수 있었다.

그림 16과 그림 17에서 표시된 데이터의 평균과 표준편차는 표 2와 같다. 여기서 평균은 드론이 주로 향한 방향을 가리키고, 시뮬레이션과 드론이 각 조건마다 향하는 방향을 벡터로 만든 후 같은 조건 하에서 시뮬레이터와 드론 사이의 벡터 각도를 오차로 정의하였다.

VI. 결론

기존 인공지능을 사용하여 드론을 제어하는 연구들은 학습 연산을 PC에서 상태와 행동 데이터를 송수신하며 학습이 진행되었고, 학습된 가중치를 사용하여 테스트 동작을 진행할 때 또한 드론의 상태와 PC에서 연산한 행동 데이터

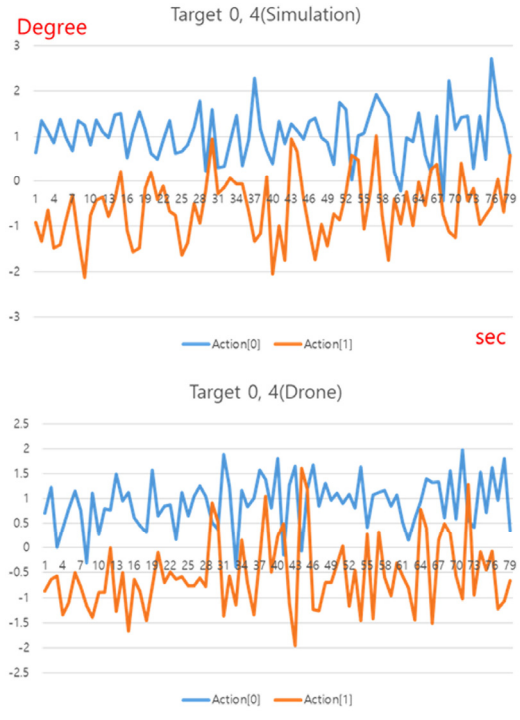


그림 16. 목표 지점 (0,4)에서의 실제 드론과 시뮬레이터의 시간 반응.

Fig. 16. Time response of simulation and experiments for target (0,4).

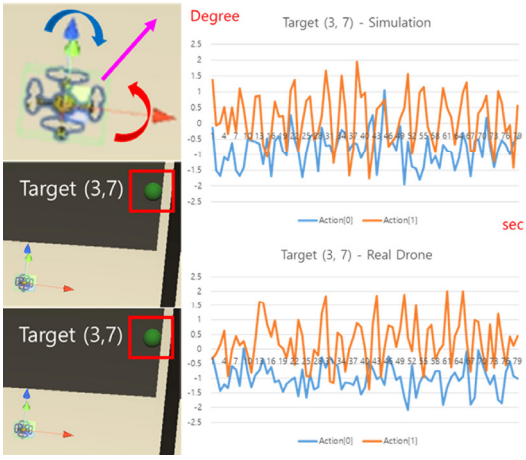


그림 17. 목표 지점 (3,7)에서의 실제 드론과 시뮬레이터의 시간 반응.

Fig. 17. Time response of simulation and experiments for target (3,7).

그림 18. 목표 지점 (3,7)에서의 실제 드론의 자세 반응.

Fig. 18. Drone posture response for target (3,7).

표 2. 목표 별 시뮬레이션과 드론의 각도 평균, 표준편차 (Unit: m, degree).

Table 2. Angle averages and standard deviations of simulations and experiments at each target (Unit: m, degree).

	Simulations (°)	Experiments (°)	Errors (°)
Target Pos (0,4) Avg. Std.	(1.048, -0.587) (0.560, 0.711)	(0.925,-0.536) (0.513, 0.738)	0.830
Target Pos (0,4) Avg. Std.	(-0.827, 0.116) (0.531, 0.881)	(-0.987, 0.268) (0.458,0.802)	4.107

를 지속적으로 송수신했다. 이러한 방식은 시뮬레이션과 디바이스의 테스트로서는 적절한 방법일 수 있으나, stand-alone으로 디바이스를 동작시킬 수 없다는 단점이 있었으나, 본 연구에서는 경량 embedded PC를 드론에 부착하여 독립적인 연산을 진행하여 장애물 회피와 경로계획을 동시에 구성하게 하였고, 이를 위해 드론의 시뮬레이션과 실제 구동을 수행한 결과 다음과 같은 결론들을 얻었다.

1) 이미지 데이터 없이 거리, 각도, 위치, 속도 데이터만으로 강화학습이 가능하고, 이렇게 학습된 데이터로 장애물을 피해 target까지 도달할 수 있음을 확인했다. 그러나 시뮬레이션을 진행할 때 적은 batch size와 buffer size를 사용할 때는 학습이 제대로 진행되지 않는 문제가 발생하여 자율 비행을 구현하기 위해 속도저하 문제로 인하여 SAC 알고리즘을 배제하고 PPO 알고리즘의 학습을 위한 표 1과 같은 상수가 기록되는 Json 파일을 사용하여 수치를 변경할 때마다의 학습 rate를 측정했고, 최종적으로는 4096의 batch size와 40960의 buffer size를 사용해 학습을 진행시켰다. 학습할 때는 앞의 batch와 buffer를 이용하지만, 가중치를 update할 때 외에는 memory stack의 개수를 조절하는 이 수치를 사용하지 않으므로 실제 드론을 비행할 때는 영향은 없다.
2) 강화학습 알고리즘 구성은 2d를 기준으로 구성했으나 관측하는 vector를 2d에서 3d로 전환하고 target throttle을 행동에 추가하는 것으로 3d 지도에서의 학습을 진행하였고, 2d에서 3d로 전환했을 때 state의 개수는 5개 이상 증가하지 않아 학습 진행 시 계산부담은 미미했다. 2d였던 수치가 3d로만 전환한 것이기 때문에 알고리즘 상에서 문제없이 학습이 진행될 것이라 예상하였고, 충분히 동작함을 확인하였다. 현재 테스트를 진행한 드론에서는 2d 자율비행을 위해 12 m 거리의 측정이 가능한 저비용 Lidar를 장착하였으나, 추가로 3d 자율 비행 실험을 진행할 때 위치 데이터 수신용으로 UWB 대신 GPS를 장착하고 30 m 이상 측정이 가능한 lidar 센서를 부착하면 시뮬레이션에서도 동작했듯이 충분히 자율비행을 실행할 수 있을 것이라 기대한다.
3) 주행할 때 사용하는 테스트 코드는 상태를 받아들여 연산하는 데이터의 개수가 3d 기준으로도 60개 이하이고 (2d lidar 거리 측정이 반 이상을 차지함) 데이터 측정 연산도 fully connected layer가 없기 때문에 연산 부하가 낮다. 이는 실제 Jetson Nano에서 데이터를 수신하고 roll 및 pitch 입력신호를 출력할 때 20 hz 이상의 제어신호 출력을 확인할 수 있었다.

REFERENCES

[1] T. Frey, “192 Future Uses for Flying Drones,” Futurist Speaker Business Trends, 2013. [Online]. Available: https://futuristspeaker.com/business-trends/192-future-uses-for-flying-drones/

[2] D. J. Lee, “Recognition of environment and autonomous flight based on deep reinforcement learning,” *Journal of the KSME (in Korean)*, vol. 59, no. 5, pp. 43-48, May. 2019.

[3] M. C. Koval, C. R. Mansley, and M. L. Littman. “Autonomous quadrotor control with reinforcement learning,” 2010.

[4] S. Omidshafiei, “Reinforcement learning-based quadcopter control,” 2013.

[5] Jemin Hwangbo, Inkyu Sa, R. Siegwart, and M. Hutter, “Control of a Quadrotor With Reinforcement Learning,” *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096-2103, 2017.

[6] C. H. Liu, Z. Chen, J. Tang, J. Xu, and C. Piao, “Energy-efficient UAV control for effective and fair com-

munication coverage: A deep reinforcement learning approach," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 9, pp. 2059-2070, 2018.

[7] H. Bayerlein, P. De Kerret, and D. Gesbert, "Trajectory optimization for autonomous flying base station via reinforcement learning," *2018 IEEE 19th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, IEEE, pp. 1-5, Jun. 2018.

[8] H. X. Pham, H. M. La, D. Feil-Seifer, and A. Nefian, "Cooperative and distributed reinforcement learning of drones for field coverage," 2018.

[9] T. Sugimoto and M. Gouko, "Acquisition of hovering by actual UAV using reinforcement learning," *2016 3rd International Conference on Information Science and Control Engineering (ICISCE)*, IEEE, pp. 148-152, Jul. 2016.

[10] M. B. Vankadari, K. Das, C. Shinde, and S. Kumar, "A reinforcement learning approach for autonomous control and landing of a quadrotor," *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 676-683, IEEE, Jun. 2018.

[11] Aisim. Airsim Tech Support Page, 2020. [Online] Available: https://microsoft.github.io/AirSim/

[12] Cube Flight Controller, Pixhawk Tech Support Page, 2020. [Online] Available: https://docs.px4.io/v1.9.0/en/flight_controller/pixhawk-2.html

[13] S. Y. Shin, Y. W. Kang, and Y. G. Kim, "Obstacle Avoidance Drone by Deep Reinforcement Learning and Its Racing with Human Pilot," *Applied Sciences*, vol. 9, no. 24, 5571, 2019.

[14] Unity. Unity Home Page, 2020. [Online] Available: https://unity.com/kr

[15] YouTube, Top searches on YouTube: Explorer Drone | Unity ML-Agents, 2020. [Online] Available: https://www.youtube.com/watch?v=3-4_-FBDr8Q

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. "Playing atari with deep reinforcement learning," 2013.

[17] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, pp. 1057-1063, 2000.

[18] REINFORCE, Williams, R. J., "Simple statistical gradient-following algorithms for connectionist reinforcement learning," Machine learning, vol. 8, no. 3-4, pp. 229-256, 1992.

[19] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," Jun. 2014.

[20] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, and D. Wierstra, "Continuous control with deep reinforcement learning," arXiv:1509.02971, 2015.

[21] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. Tb, and T. Lillicrap, "Distributed distributional deterministic policy gradients," arXiv:1804.08617, 2018.

[22] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization. In International conference on machine learning," pp. 1889-1897, Jun. 2015.

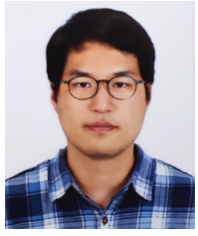
[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, Proximal policy optimization algorithms, arXiv:1707.06347, 2017.

[24] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," arXiv:1801.01290, 2018.

[25] C.-Y. Jung, T. Kim, and D. H. Shim, "Development of ROS-based mobile robot system and experiments on indoor autonomous driving," *Journal of Institute of Control, Robotics and Systems (in Korean)*, vol. 25, no. 5, pp. 438-444, Mar. 2019.

[26] J.-H. Lee, K.-J. Ahn, T.-G. Lee, K.-I. Min, O.-S. Kwon, S.-W. Baek, M.-C. Park, D.-H. Cho, J.-R. Hwang, J.-K. Kang, S.-H. Lee, D.-Y. Seo, J.-H. Kim, and Y. Kang, "Development of control system for international autonomous driving competition using robot operating system," *Journal of Institute of Control, Robotics and Systems (in Korean),* vol. 25, no. 5, pp. 363-373, Mar. 2019.

[27] S. G. Park, G. T Kim, J. H. Kim, and D. H. Kim, "Development of waypoint driving robot system using UWB communication," *Spring Conference of the Korean Society of Mechanical Engineers(in Korean),* pp. 1753-1754, 2018.



박 성 관

2018년 서울과학기술대학교 기계시스템디자인공학과 졸업. 2019년~현재 서울과학기술대학교 석사과정. 관심분야는 Mechatronics, Control, AI.

김 동 환

1986년 서울대학교 기계설계학과 졸업. 1988년 동 대학원 석사. 1995년 Georgia Tech 박사. 1998년~현재 서울과학기술대학교 기계시스템디자인공학과 교수. 관심분야는 Mechatronics, Robotics, 자동화, 인공지능.