Artificial Intelligence and Mobility Lab

CommNet을 활용한 전기차/UAV충전 인공지능 설계

Electric vehicle/UAV charging AI using CommNet

Won Joon Yun
Korea University, School of Electrical Engineering
Artificial Intelligence and Mobility Laboratory

Contents

- Establish a strategy to solve the problem
  - State / action / objective analysis
  - State / Action / Reward design
- Charging Scenario analysis
  - Assumptions for ESS / Drone
  - Electric vehicle charging scenario
  - Drone wireless charging scenario
- State / Action / Reward description and actual parameters
  - Electric vehicle charging scenario
  - Drone wireless charging scenario

- Does the system satisfy Markov property?

memoryless property of a stochastic process

$\rightarrow P\left(s_{t+1} \middle| \tau_{\sim(s_0, s_t)}\right) = P(s_{t+1}|s_t)$

- Is there an objective function in the system? In other words, are you trying to optimize?

$\underset{\boldsymbol{x}}{\operatorname{argmin}}\, P(\boldsymbol{x}) \;\; or \;\; \underset{\boldsymbol{x}}{\operatorname{argmax}}\, Q(\boldsymbol{x})$

In Dynamic Programming…

- What variables are there.

→ Variables(x, y, z, w, θ)

- How variables have a relationship.

→ Equations(eq1,eq2,…)

- Which one to optimize

→ Minimize x, z and Maximize θ

In Reinforcement Learning…

- What variables are there.

→ State(x, y, z, w, θ)

- How variables have a relationship.

→ Environment

- Which one to optimize

→ Action$(a_1, a_2, \ldots)$

In Dynamic Programming…

- Optimizing Method

→ Gradient Descent

- Objective Function

→ J(**x**) = (function of **x**) *s.t.*

Constraint 1,

Constraint 2,

Constraint 3,

Constraint 4,

In Reinforcement Learning…

- Optimizing Method

→ Policy Gradient

- Objective Function

→ $J(\theta) = E_\tau[\sum_{t=0}^{T} \boxed{r(s_t, a_t)}]$ ← For $^\forall$scenario

In Dynamic Programming…

- Optimizing Method

→ Gradient Descent

- Objective Function

→ J(**x**) = (function of **x**) *s.t.*

Constraint 1,

Constraint 2,

Constraint 3,

Constraint 4,

In Reinforcement Learning…

- Optimizing Method

→ Policy Gradient

- Objective Function

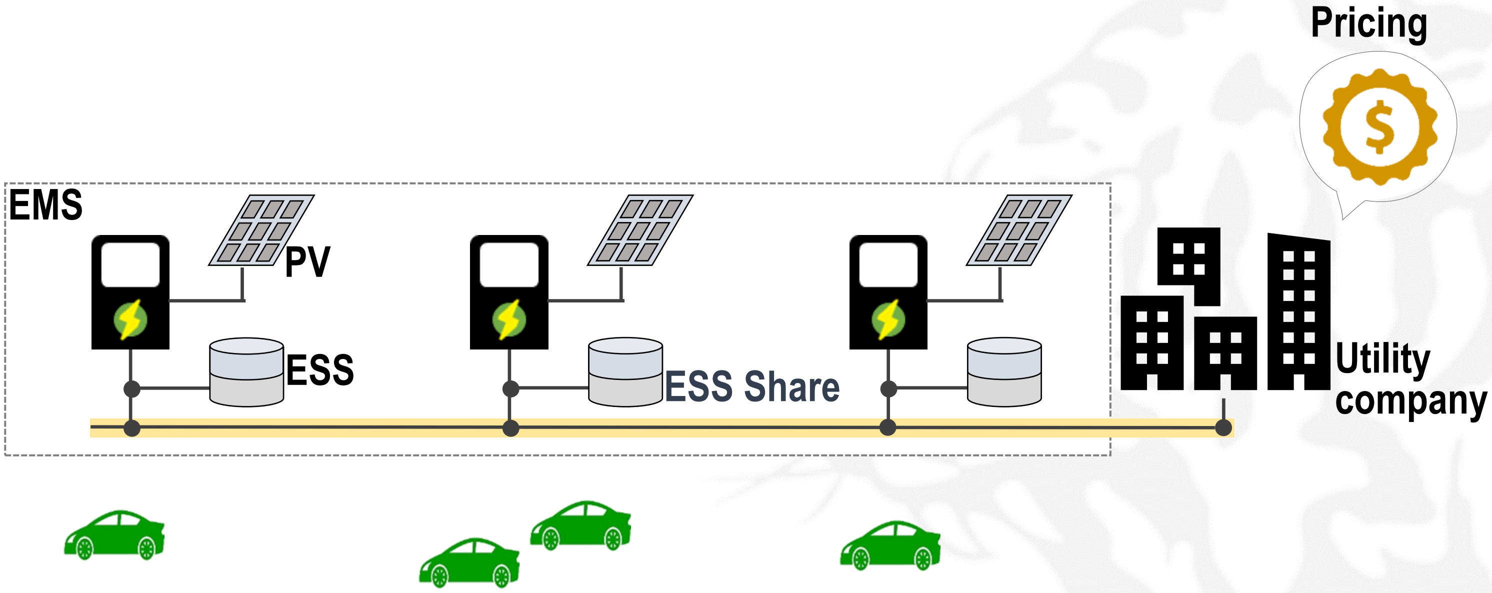
→ $J(\theta) = E_{\tau}[\sum_{t=0}^{T} \boxed{r(s_t, a_t)}]$ ← For $^{\forall}$scenario

Reward Shaping is the most important thing to optimize system

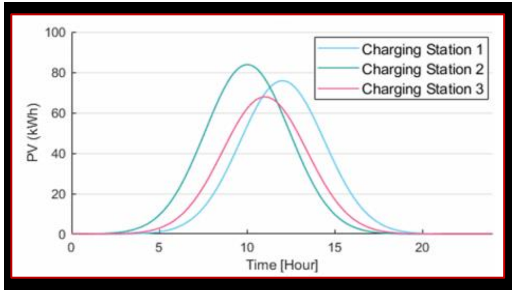
Contents

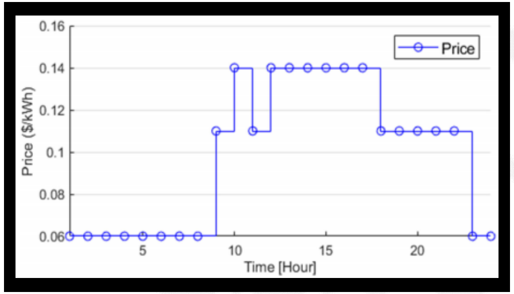
- Establish a strategy to solve the problem
  - State / action / objective analysis
  - State / Action / Reward design
- ESS Scenario analysis
  - Assumptions for ESS
  - Electric vehicle charging scenario
  - Drone wireless charging scenario
- State / Action / Reward description and actual parameters
  - Electric vehicle charging scenario
  - Drone wireless charging scenario

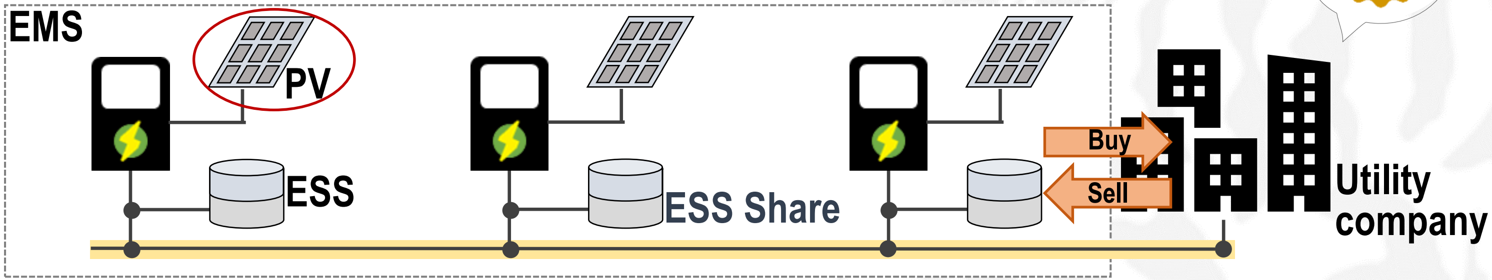
- System overview

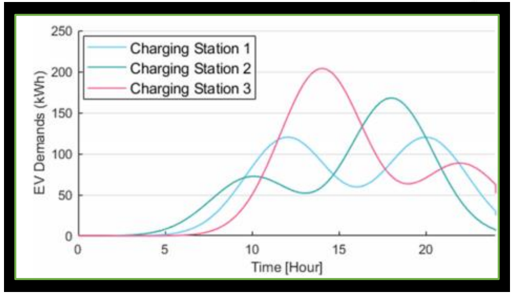


Scenario Description (1) Electric Vehicle Charging Scenario









Design state, action, objectives

- Let us assume agent is EVC.
- State : $\{o_t^n,\ price_t,\ price_t^{avg},\ v_t^n,\ d_t^n\}$
- Action : {Buy, Sell}
- Objective : Maximize Benefit
- Objective Function : $J(\theta)\ =\ E_\tau[\sum_{t=0}^{T} r(s_t, a_t)]$
- Optimal Benefit should be represented to $J(\theta)\ =\ E_\tau[\sum_{t=0}^{T} r(s_t, a_t)]$.

(o) amount of energy charged in the ESS

$(price)$ price

$(price\hat{}avg)$ price average

(v) PV generation

(d) sum of energy demands

$r(s_t, a_t)$ Shaping

- r_b: benefit reward → [+] Reward $[0, \infty)$
- r_p: pay reward → [−] Reward $(-\infty, 0]$
- r_o: overcharged energy reward → [−] Reward $(-\infty, 0]$
- r_s: shared energy reward → [+] Reward $[0, \infty)$

$$\boxed{r(s_t, a_t) = r_b + r_p + r_o + r_s}$$

$r(s_t, a_t)$ Shaping

- r_b: benefit reward → [+] Reward $[0, \infty)$
- r_p: pay reward → [−] Reward $(-\infty, 0]$
- r_o: overcharged energy reward → [−] Reward $(-\infty, 0]$
- r_s: shared energy reward → [+] Reward $[0, \infty)$

$$\boxed{r(s_t, a_t) = r_b + r_p + r_o + r_s}$$

⇩

But, What if the magnitude of r_o is much bigger than other reward?

$r(s_t, a_t)$ Shaping

- r_b: benefit reward → [+] Reward $[0, \infty)$
- r_p: pay reward → [−] Reward $(-\infty, 0]$
- r_o: overcharged energy reward → [−] Reward $(-\infty, 0]$
- r_s: shared energy reward → [+] Reward $[0, \infty)$

$$r(s_t, a_t) = r_b + r_p + r_o + r_s$$

But, What if the magnitude of r_o is much bigger than other reward?

r_b, r_o, r_s become meaningless.

$r(s_t, a_t)$ Shaping

- r_b: benefit reward ➔ [+] Reward $[0, \infty)$
- r_p: pay reward ➔ [−] Reward $(-\infty, 0]$
- r_o: overcharged energy reward ➔ [−] Reward $(-\infty, 0]$
- r_s: shared energy reward ➔ [+] Reward $[0, \infty)$

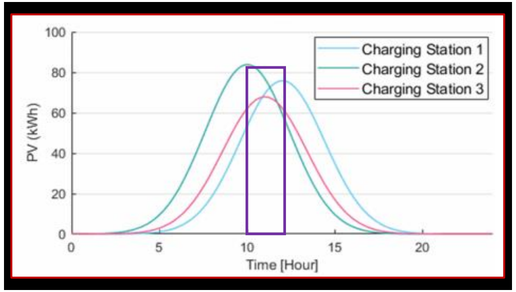
$$\boxed{r(s_t, a_t) = \boldsymbol{W_b} r_b + \boldsymbol{W_p} r_p + \boldsymbol{W_o} r_o + \boldsymbol{W_s} r_s}$$

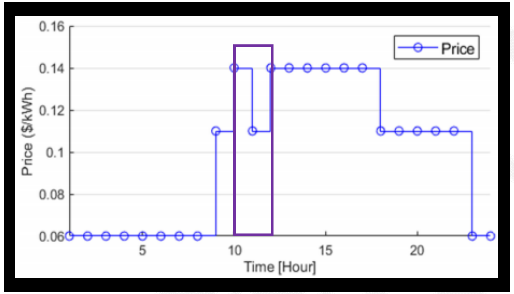
⇩

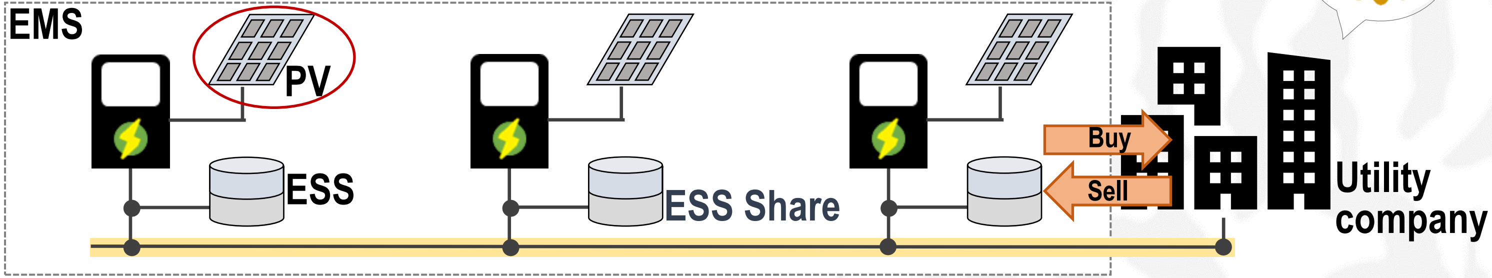
How to find optimal parameters?

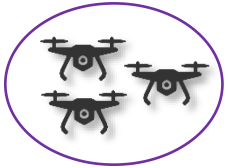
Normalize reward

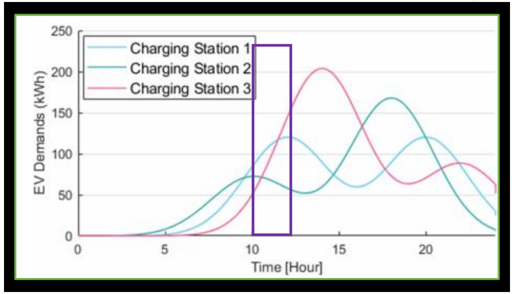
Scenario Description (2) Drone Charging Scenario



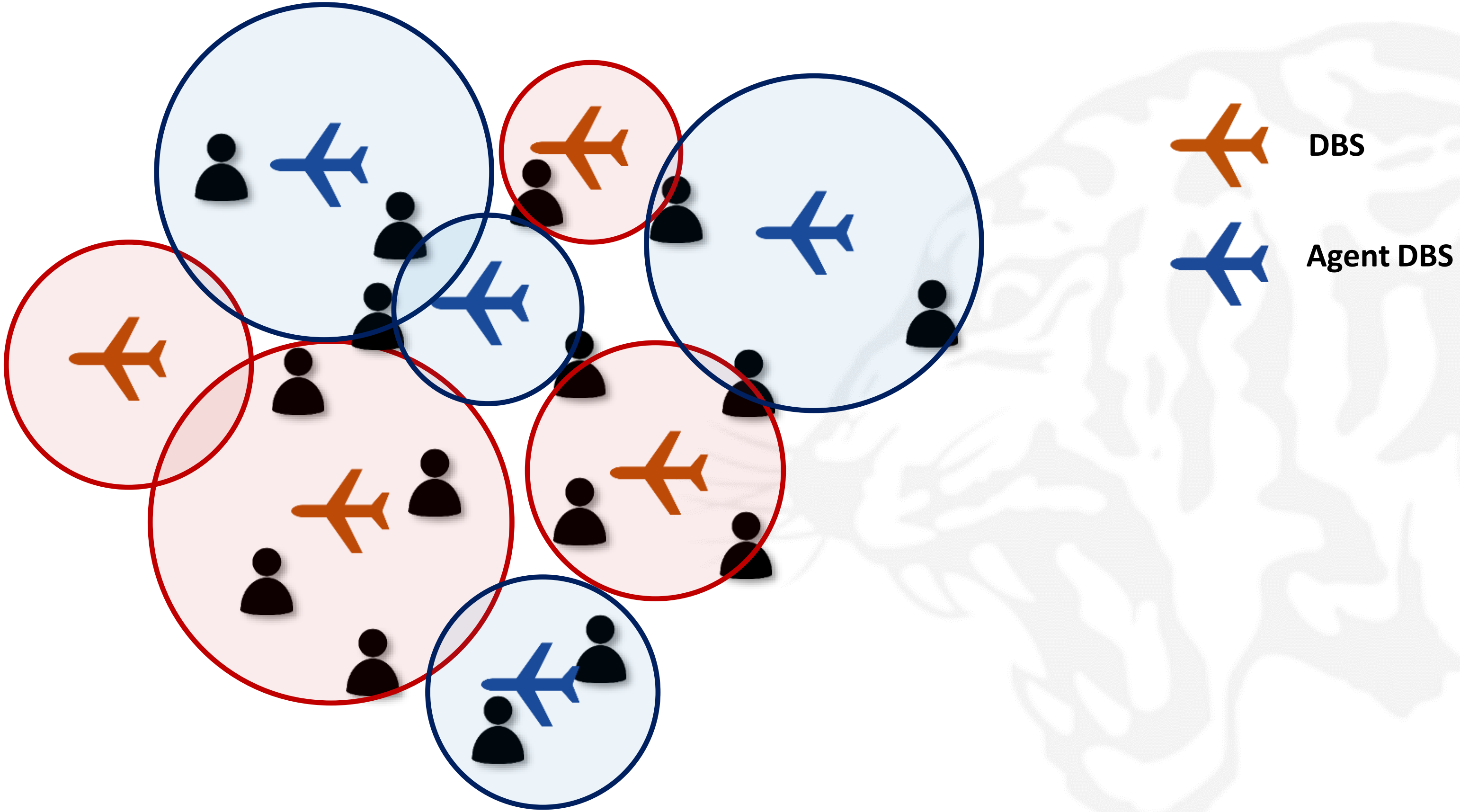








Scenario Description (3) Drone Base Station Scenario



RL Configuration [State]

1. **Relative Location of other MBSs:** (x_other – x_self, y_other – y_self)

2. **Coverage Information of other MBSs**

3. **Relative Location of other users:** (x_users – x_self, y_users – y_self)

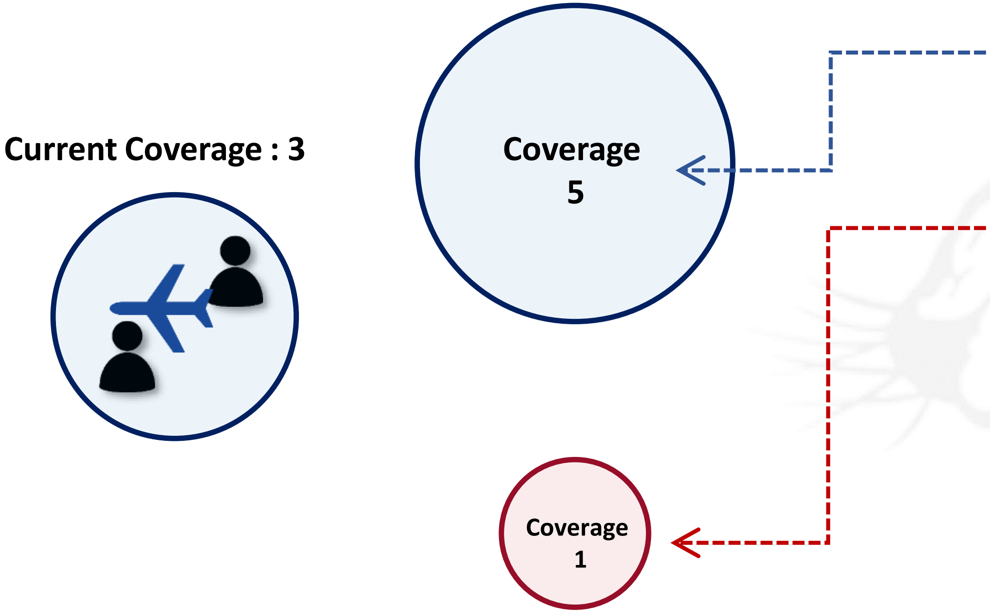
4. **Current Location:** (x, y)

5. **Residual Energy of MBS**

6. **Expected Energy Consumption in this time slot**

4	5	6	1	2	3

Coverage Range Control [Discrete Action]



1. Coverage Range Increase by 1
2. Coverage Range Increase by 2
3. Coverage Range Increase by 3
4. Coverage Range **Not Change**
5. Coverage Range Decrease by 1
6. Coverage Range Decrease by 2
7. Coverage Range Decrease by 3

RL Configuration [Reward]

[Individual Reward]

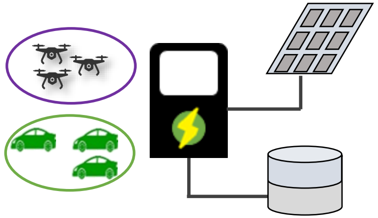
1. Number of users that MBS assists
2. Energy Consumption **(For Coverage Optimization)**
3. Distance between other agents and agent **(For spreading agents)**

[Global Reward]

4. Total Number of users that managed agents **(For spreading agents)**

$$\boldsymbol{Reward = w_1 * 1 - w_2 * 2 - w_3 * 3 + w_4 * 4}$$

Variable to state Process.



Agents	State Variables				
ct_1	o_t^1	$price_t$	$price_t^{avg}$	v_t^1	d_t^1
ct_2	o_t^2	$price_t$	$price_t^{avg}$	v_t^2	d_t^2
ct_3	o_t^3	$price_t$	$price_t^{avg}$	v_t^3	d_t^3

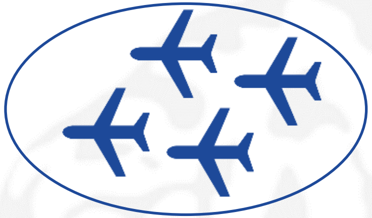
(o) amount of energy charged in the ESS

($price$) price

($price^{avg}$) price average

(v) PV generation

(d) sum of energy demands



Agents	State Variables					
n_1	a_t^1	b_t^1	c_t^1	d_t^1	e_t^1	f_t^1
n_2	a_t^2	b_t^2	c_t^2	d_t^2	e_t^2	f_t^2
n_3	a_t^3	b_t^3	c_t^3	d_t^3	e_t^3	f_t^3
n_4	a_t^4	b_t^4	c_t^4	d_t^4	e_t^4	f_t^4

(a) Relative Location of other MBSs

(b) Coverage Information of other MBSs

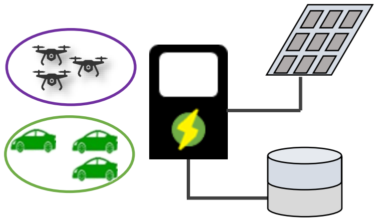
(c) Relative Location of other users

(d) Current Location

(e) Residual Energy of MBS

(f) Expected Energy Consumption in this time slot

Actual Implementation of EV Charging Scenario



Agents	State Variables				
ct_1	o_t^1	$price_t$	$price_t^{avg}$	v_t^1	d_t^1
ct_2	o_t^2	$price_t$	$price_t^{avg}$	v_t^2	d_t^2
ct_3	o_t^3	$price_t$	$price_t^{avg}$	v_t^3	d_t^3

(o) amount of energy charged in the ESS

($price$) price

($price^{avg}$) price average

(v) PV generation

(d) sum of energy demands

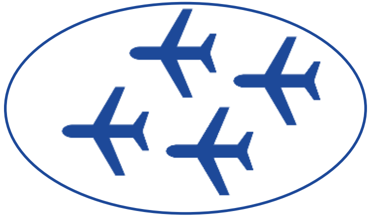
Action : {Buy(5), Sell(5)}

```
class Agent:
    def __init__(self, id, env):
        """
        설정하고자 하는 변수 선언
        self.var1 = 0
        ....
        """

    def get_status(self, id, env):
        """
        상태변수 return
        return np.array([var1, var2, ... varN])
        """

    def transition(self,action):
        """
        action에 의해 상태 변화가 일어날 때, 쓰는 function 혹은 equation을 입력
        if action == 0:
            .....
        elif action == 1:
            .....
        """
    def reset(self):
        """
        __init__ 에서 썼었던 방식 그대로 복사 붙여넣기.
        self.var1 = 0
        ....
        """
```

Actual Implementation of drone base station



Agents	State Variables					
n_1	a_t^1	b_t^1	c_t^1	d_t^1	e_t^1	f_t^1
n_2	a_t^2	b_t^2	c_t^2	d_t^2	e_t^2	f_t^2
n_3	a_t^3	b_t^3	c_t^3	d_t^3	e_t^3	f_t^3
n_4	a_t^4	b_t^4	c_t^4	d_t^4	e_t^4	f_t^4

(a) Relative Location of other MBSs

(b) Coverage Information of other MBSs

(c) Relative Location of other users

(d) Current Location

(e) Residual Energy of MBS

(f) Expected Energy Consumption in this time slot

```
class Agent:
    def __init__(self, id, env):
        """
        설정하고자 하는 변수 선언
        self.var1 = 0
        ....
        """

    def get_status(self, id, env):
        """
        상태변수 return
        return np.array([var1, var2, ... varN])
        """

    def transition(self,action):
        """
        action에 의해 상태 변화가 일어날 때, 쓰는 function 혹은 equation을 입력
        if action == 0:
            .....
        elif action == 1:
            .....
        """
    def reset(self):
        """
        __init__ 에서 썼었던 방식 그대로 복사 붙여넣기.
        self.var1 = 0
        ....
        """
```

Environment Design


```python
class Environment:
    def __init__(self):
        """
        시스템 파라미터 선언 및 초기 에이전트 선언
        self.initAgent()
        self.get_state()
        self.total_reward = 0
        self.numAgent = K
        self.n_actions = 4
        self.agents = [] # agent를 담는 공간
        """
    def initAgent(self):
        """
        for i in range(self.numAgent):
            self.agents.append(Agent(id=i, env=self))
        """

    def get_obs(self):
        obs = []
        for i in range(self.numAgent):
            obs.append(self.agents[i].get_status())) # state variable N개가 K개만큼 적층됨.
        obs = np.array(obs)
        self.state = obs
        return obs

    def get_rewards(self):
        """
        Reward Shaping
        self.total_reward = 0
        for i in range(self.numAgent):
            r1 = self.agents[i].var1 * self.W1
            r2 = self.agents[i].var2 * self.W2
            r3 = self.agents[i].var3 * self.W3
            r4 = self.agents[i].var4 * self.W4
            self.total_reward += r1+r2+r3+r4
        """
        return self.total_reward
```

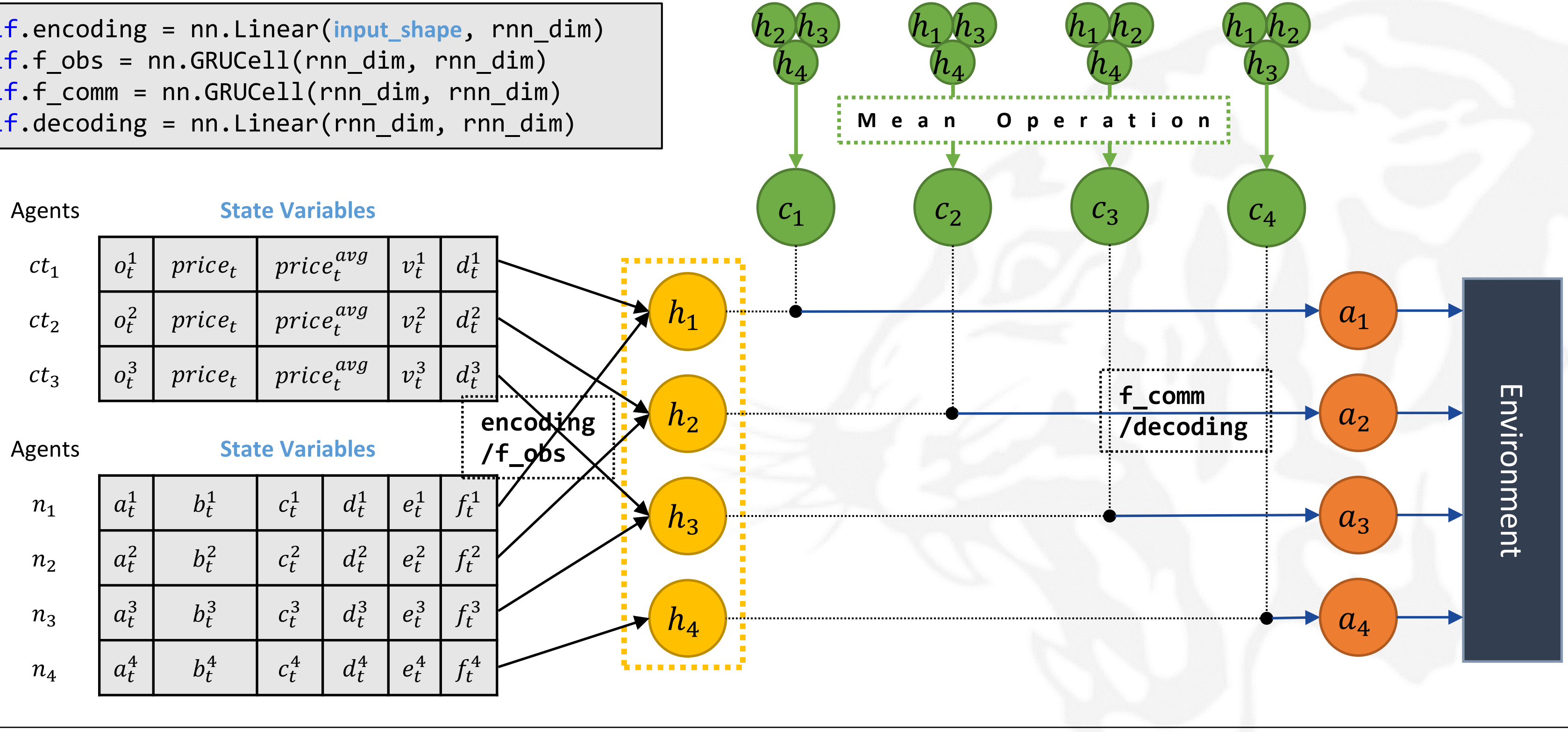
```python
def step(self, actions):
    self.inputs = self.next_inputs
    rewards = 0
    # Action
    for i in range(len(self.uavs)):
        self.agents[i].transition(actions[i])
    # Reward
    for idx in range(len(self.uavs)):
        rewards += self.agents[idx].reward
    # Next State
    for idx in range(len(self.uavs)):
        self.agents[idx].obs_status()

    self.next_inputs = self.get_obs()
    self.episode_step += 1
    self.total_reward = self.get_rewards
    return rewards,

def reset(self):
    """
    초기 상태로 되돌아감
    self.initAgent()
    """
```

Implementation Reinforcement Learning using CommNet

```
self.encoding = nn.Linear(input_shape, rnn_dim)
self.f_obs = nn.GRUCell(rnn_dim, rnn_dim)
self.f_comm = nn.GRUCell(rnn_dim, rnn_dim)
self.decoding = nn.Linear(rnn_dim, rnn_dim)
```



Iterate until scenario terminated.

Thank you for your attention!

- More questions?
  - ywjoon95@korea.ac.kr