Engineering Application Note 105eFD100 Deep Reinforcement Learning (June 3, 2021)

SAC Soft Actor Critic Algorithm Overview

Harry Li ‡, Ph.D.
Computer Engineering Department, San Jose State University
San Jose, CA 95192, USA

Email†: harry.li@ctione.com

***Abstract*—This note describes the soft actor critic algorithm (SAC). First, we describe the concept of deep reinforcement learning (DRL) with a tuple of state, action, transition functions, and reward function, then we define stochastic policy function. In particular, we choose Gaussion probability distribution function for the policy function to define continued actions. Then we describe expectation of all rewards and define it as an objective function to maximize for long term reward. We formulated the maximization by stochastic gradient ascent algorithm, which forms stochastic gradient policy technique.**

I. Introduction

The soft actor critic algorithm (SAC) is the technique widely adoption for deep reinforcement learning (DRL). The scope of this algritm with relaship to the related algorithms is as follows and as illustrated in the following figure:

1 A Markov Decision Process (MDP) which serves as the foundation for the discussion of all algorithms;
2 Deep-Q (DQ) and double Deep-Q (DDQ) algorithms for the discrete control actions based DRL; and
3 Stochastic gradient policy algorithm and SAC algorithm for continuous control action based DRL.

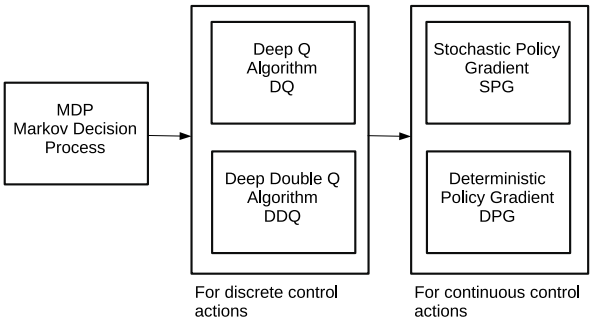

Fig. 1. The scope of SGP algrithm with relaship to the related algorithms.

II. DRL Architecture

In DRL archictecture as illustrated in the figure, we can use one of the Deep Neural Networks (DNN) models, e.g., SPG or DPG in the DRL system design.

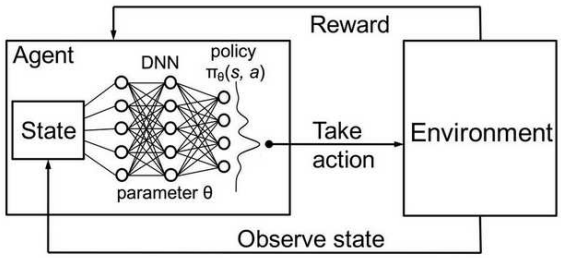

Fig. 2. we can use one of the Deep Neural Networks (DNN) models, e.g., SPG or DPG in the DRL system design.

Note in the DRL architecture, there are 2 feedbacks, one feedback is for the feedback of the agent state, or, system state denoted as S. This feedback is common in control systems. For example in PID (Proportional, Integral, Derivative) controller design, we have feedback like that, or in modern control we have state feedback. However, the second feedback is quite unique, which is reward feedback. The reward feedback forms the core of reinforcement learning where the maximization of long term reward is the objective of deep learning process.

In the context of robotic operations, a robot makes sequential movement, e.g., control actions in a stochastic environment. A stochastic environment makes robot action not necessarily bring to the desired position. The action only makes the robot move to a desired position with certain likelihood, which is described as probability distribution $Prob(a_{t+1}|s_t, a_t)$.

DRL mathematical formulation consists of

1. a set S of states, denoted as $S = \{s_1, s_2, ..., s_N\}$, and

2. a set A of actions, denoted as $A = \{a_1, a_2, ..., a_M\}$ as well as

3. a transition function T and

4. a reward function $R(s_t, a_t)$.

We define a tuple $< S, A, T, R >$ for the above defined sets and functions. This tuple describes the relationship of the agents and its interaction with an environment via state feedback and reward functions.

III. Markov Decision Process

Robot motion by the mathematical nature is a sequential decision process modeled as a Markov Decision Process (MDP) which consists of the tuple $< S, A, T, R >$, and has the following "short-memory" characteristics for its states S:

$$Prob(s_{t+1}|s_t) = Prob(s_{t+1}|s_t, s_{t-1}, ..., s_2, s_1). \quad (1)$$

Now, let's take a look at the reward function $R(s_t, a_t)$ which define an one step reward at current state s_t per the action a_t. Since the environment is stochastic, the agnet has its next state s_t stochastic as well, because the action a_t may not lead the agent to the desired state s_t as it is intended for. Therefore, the reward at this step $r(s_t, a_t)$ is stochastic as well. Define a policy function

$$\pi(a|s) = Prob(a|s, \theta), \quad (2)$$

where θ is a set of agent (DNN) parameters. The policy π is defined as a conditional probability of an action a given a state s with the agent's DNN parameter θ.

Choose Gaussian distribution for the policy function π as illustrated in the figure below.

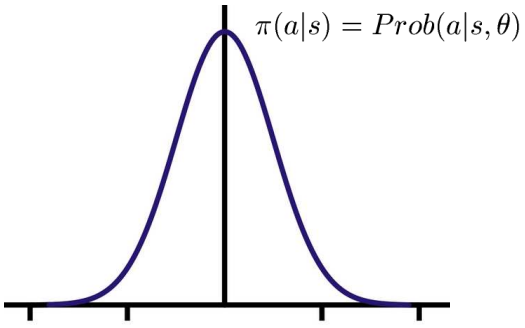

Fig. 3. Choose Gaussian distribution for the policy function π.

So to describe the entire process (e.g., an episode), for all rewards, we will have to take an average of the rewards. This is the statistical expecation of the rewards as follows:

$$E_{s\sim P_\pi, a\sim\pi_\Theta}[r(s, a)] \quad (3)$$

Denote this expectation as an objective function $J(\theta)$, e.g.,

$$J(\theta) = E_{s\sim P_\pi, a\sim\pi_\Theta}[r(s, a)] \quad (4)$$

Computationally, we have

$$J(\theta) = \sum_{s\sim S} Prob(s) \sum_{a\sim A} \pi_\theta(s, a)R(s, a). \quad (5)$$

When in any state $s \in S$, an action $a \in A$ will lead to a new state with a transition probability $P_T(s, a, s\prime)$, and a reward R(s, a) function. In case of discrete actions a_t, we can have a_t as left, or right, up or down in the case of driving a simulated car in a computer game such as the work by google deep-mind team. For the discrete actions, we will use Deep Q (DQ) or Double Deep Q (DDQ) algorithm.

In the case of autonomous vehicle steering or 6 degree-of-freedom (DoF) robot arm movement, we will have to operate with with continued actions a_t. We will use Stochastic Policy Gradient (SPG) technique. Given in this figure is a FD100 robot and it is operated in a stochastic environment and whose actions are continuous.

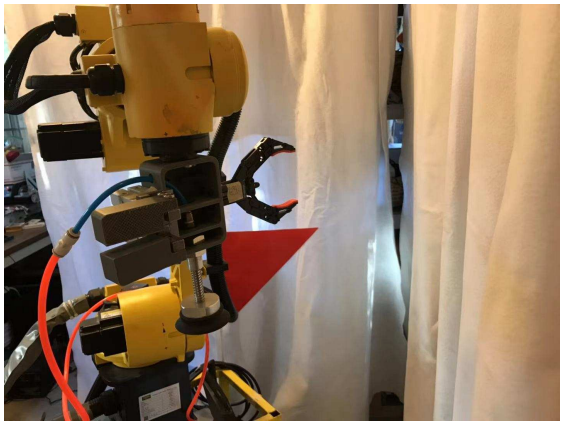

Fig. 4. An end effector (a set of three in this case) from CTI's FD100 robot, whose control action is continuous.

IV. Stochastic Policy

The objective for DRL is to maximize long term rewards, e.g., to maximize $J(\theta)$. In order to achieve this maximization, we are going to explain the stochastic policy reward.

From equation 2, we substitute into equation 5, so we have

$$J(\theta) = E_{s\sim P_\pi, a\sim\pi_\Theta}[r(s, a)] \quad (6)$$

Computationally, we have

$$J(\theta) = \sum_{s\sim S} Prob(s) \sum_{a\sim A} Prob(a|s, \theta)R(s, a). \quad (7)$$

So, we have defined policy by using Gaussian probability distribution in equation 2, and consequently refers to as a stochastic policy.

V. Stochastic Policy Gradient

Now let's take a gradient of $J(\theta)$,

$$\bigtriangledown J(\theta) = \sum_{s\sim S} Prob(s) \sum_{a\sim A} \bigtriangledown Prob(a|s, \theta)R(s, a). \quad (8)$$

Note:

$$dx = xd(\log x), \quad (9)$$

so is

$$\bigtriangledown Prob(a|s, \theta) = Prob(a|s, \theta) \bigtriangledown log(Prob(a|s, \theta)). \quad (10)$$

Therefore, let's rewrite equation 8, we have

$$\bigtriangledown J(\theta) = \sum_{s\sim S} Prob(s) \sum_{a\sim A} Prob(a|s, \theta) \bigtriangledown log(Prob(a|s, \theta))R(s, a). \quad (11)$$

VI. Maximization of Reward by Gradient Ascent Technique

???The stochastic policy $\pi : S \rightarrow D$ maps from a space state to a probability over the set of actions, and $\pi(a|s)$ represents the probability of choosing action a at state s.

The goal is to find the optimal policy π^* to produce the highest rewards [Rein, 2020]:

$$\arg\max_{\pi \in \Omega_\pi} \{E[\sum_{k=0}^{H-1} \gamma^k R(s_k, a_k)]\} \tag{12}$$

For the stochastic nature of these rewards, we use statistical expectation as

$$E[\sum_{k=0}^{H-1} \gamma^k R(s_k, a_k)] = \int_{\inf} \gamma^k R(s_k, a_k) P(s_k, a_k) ds \tag{13}$$

Hence, we have the average discounted rewards under policy π.

VII. States and Actions

Definition 1. *Trajectory τ. A trajectory τ of a robot motion is defined as a sequence of state-action pairs in time sequence $t_1, t_2, ..., t_N$, denoted as $\tau = (s_1, a_1, s_2, a_2, ..., s_N, a_N)$.*

Consider any trajectory τ of a robot motion, e.g., $\tau = (s_1, a_1, s_2, a_2, ..., s_N, a_N)$, the trajectory of the end effector in 3D space. See an end effector (a set of three in this case) from CTI's FD100 robot below,

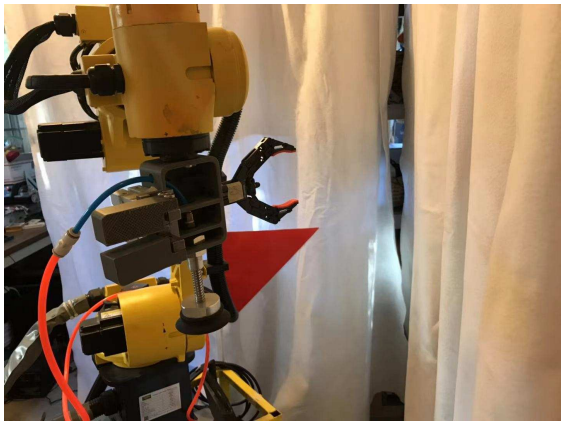

Fig. 5. An end effector (a set of three in this case) from CTI's FD100 robot below, whose movement forms a trajectory in 3D space".

The bigger view of the FD100 robot with an end effector (a set of three in this case) in the view.

Fig. 6. The bigger view of the FD100 robot with an end effector (a set of three in this case) in the view.

VIII. Reward

Definition 2. A reward $r(s_i, a_i)$ is defined as function with a numerical value assigned to each state-action pair (s_i, a_i), formulate as $r : S \times A \rightarrow R$, where $S \times A = (s_1 a_1, s_2 a_1, ..., s_1 a_N, s_2 a_N, ..., s_N a_1, ..., s_N a_N)$, and $R(s_i, a_i) \in (-\infty, \infty)$.

We can build tables for S, A and R repectively for FD100 6 DoF robot as follows

TABLE I
State Table for FD100 Robot

Category	Description	Note
$s_{j1-angle}$	joint 1 angle α $[0, 350]$ degree	Continuous
$s_{j1-speed}$	joint 1 α-speed $[0, 80\%]$	Continuous
$s_{j1-accel}$	joint 1 α-acceleration $[0, 60\%]$	Continuous
...	...	...
$s_{j6-angle}$	joint 6 angle α $[0, 350]$ degree	Continuous
$s_{j6-speed}$	joint 6 α-speed $[0, 80\%]$	Continuous
$s_{j6-accel}$	joint 6 α-acceleration $[0, 60\%]$	Continuous

TABLE II
Action Table For FD100 Robot

Category	Description	Note
MoveJ	Joint movement action	explicit control α_i
MoveL	Linear movement action	...
MoveA	Arc movement action	...

The design of reward function $R(s_t, a_t)$ is defined based on the general guidelines from Unity AI Robot github site [Unity AI, 2020].

The general guideline to define reward function per Unity AI implementation of Google team's work is based on the movement of the end effector motion trajectory. It can either moves toward to the target or move away from the target. We describe in the following details:

1 Define the starting end effector's position as $distance(0)$, or in short notation $d(0)$, [1] as illustrated in figure 7.

2 Define the target position as "NearestComponent" as in the code implementation. The NearestComponent is a constant throughout each training episod.

3 Define distance $d(t)$ as the distance from the position of end effector to the NearestComponent, e.g., the target position, see *line 109* of RobotControllerAgent.cs code (the base line unmodified program from Unity AI distribution [Unity AI 2020]).

$$d(t) = \sqrt{(x_e(t) - x_N)^2 + (y_e(t) - y_N)^2 + (z_e(t) - z_N)^2} \tag{14}$$

where $(x_e(t), y_e(t), z_e(t))$ is the position of the end effector at time t, and (x_N, y_N, z_N) is the position of the "NearestComponent" appeared in the code, e.g., target. The distance $d(t)$ will be changing for each time index t during the training. Note we use time index t here in the mathematical formulation, and index i or k in the program.

4 Define $\Delta d(t)$ from the base line code, see equation (5) from my design note Part II, pp. 31 and pp. 36, which are derived based on *line 1???* of RobotControllerAgent.cs.

$$\Delta d(t) = d(t) - PrevBest(t). \tag{15}$$

5 Define PrevBest(t) based on the following notion which is from my design note Part II, pp. 43, which is derived from the based code RobotControllerAgent.cs.

$$\begin{cases} \text{if } d(t) < PrevBest(t), & PrevBest(t) = d(t), \\ \text{if } d(t) > PrevBest(t), & PrevBest(t) = PrevBest(t-1). \end{cases} \tag{16}$$

So, we have

$$PB(t) = \min(d(t), PrevBest(t-1)). \tag{17}$$

where $PB(t)$ is short hand notation for $PrevBest(t)$.

[1] In the mathematical formulation we use short hand notation such as $d(0)$, while in coding part, we will use verbatim format to write it exactly as it appears in a program.

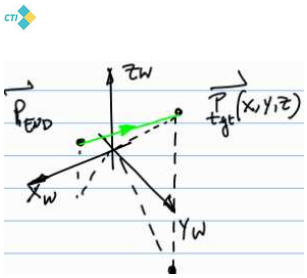

Fig. 7. The illustration of the robot end effector position, $d(t)$ which is denoted here as a point vector with subscript *end* to indicate its end effector, P_{end}, and its motion trajectory to the target as marked green line, e.g., to the "NearestComponent" in the base line code.

Now, let's take a look at the base line algorithm for the reward function.

Description of the base line algorithm

1 For the extrem case I: when the arm hits the ground, a Hefty Penalty (-1) is given, e.g.,

$$R(s_t, a_t) = -1 \tag{18}$$

where s_t = (ground state), then the training episode is terminated. Note each pisode is defined as one full cycle of training.

2 For the extrem case II: when the arm reaches the target, a Hefty Reward (1) is given,

$$R(s_t, a_t) = 1 \tag{19}$$

where s_t = (target state), then the training episode is terminated. The target state can be detected when the end effector $P_{end}(x, y, z)$ reaches the object(target) $P_{tgt}(x, y, z)$, e.g.,

$$||P_{end}(x, y, z) - P_{tgt}(x, y, z)|| \leq \epsilon \tag{20}$$

3 For the general cases, the base line reward function is defined as

$$R(s_t, a_t) = \begin{cases} -\Delta d(t), & \text{if } \Delta d(t) < 0, \\ Bd - Pb - \Delta d(t), & \Delta d(t) \geq 0. \end{cases} \tag{21}$$

where

$$Bd = BeginDistance(0), \tag{22}$$

which is a distance set at time $t = 0$ at the beginning of each Episod and it will be constance during the entire episod, and

$$Pb = PrevBest(t). \tag{23}$$

and

$$\Delta d(t) = \begin{cases} -\Delta d(t), & \text{if } \Delta d(t) < 0, \\ Bd - Pb - \Delta d(t), & \Delta d(t) \geq 0. \end{cases} \tag{24}$$

???

Suppose it takes N steps to move P_{end} to P_{tgt}, e.g., (s_t, a_t) for $t = 1, 2, ..., N$, if $P_{end} = P_{tgt}$, then end training episode, e.g.,

$$||P_{end} - P_{tgt}|| \leq \epsilon \tag{25}$$

Now, let's take a look at the motion trajectory of the robot end effector,

For $t = 1$, $P_{end}(x(1), y(1), z(1))$;

For $t = 2$, $P_{end}(x(2), y(2), z(2))$;

and for $t = i$, $P_{end}(x(i), y(i), z(i))$;

so, for $t = N$, $P_{end}(x(N), y(N), z(N))$.

Now, the distance for each step from $t-1$ to t, we have

$$||P_{end}(x(t), y(t), z(t)) - P_{end}(x(t-1), y(t-1), z(t-1))||, \tag{26}$$

with simplified notation, we often just write the above equation as $||P_{end}(t) - P_{end}(t-1)||$.

Example 1. *(Distance Calculation)*

Given $P_{end}(x(1), y(1), z(1)) = (1, 1, 2)$ *and* $P_{end}(x(2), y(2), z(2)) = (0.5, -1, 2)$,

Find distance $||P_{end}(2) - P_{end}(1)||$.

Sol. $||P_{end}(2) - P_{end}(1)||$= $||P_{end}(2) - P_{end}(1)|| = ||(1, 1, 2) - (0.5, -1, 2)|| = ||(0.5, -2, 0)|| = ||(0.5, -2, 0)|| = \sqrt{(0.5)^2 + (-2)^2} = \sqrt{4.25}$

Now we review the definition of the reward function based on the Unity AI code example as follows. For discussion and mathematical formulation purpose, we have defined defined FIFO, e.g., queue, with time index $t$to keep track 5 parameters

1 $beginDistance[i]$

2 $prevBest[i]$

3 $distance[i]$

4 $Delta_distance[i]$

5 $reward[i]$

??? 3. When the arm reaches closer to the target, a marginal reward is assigned based on the ???

difference in distance to the targe. That is to define the previous arm's (end effector) position as $P_{end}(x(t-1), y(t-1), z(t-1))$, if the current position is $P_{end}(x(t), y(t), z(t))$, then the distances to the target position can be defined as

$$d(P_{end}(t-1), P_{tgt}) = ||P_{end}(t-1) - P_{tgt}||_2 \tag{27}$$

and

$$d(P_{end}(t), P_{tgt}) = ||P_{end}(t) - P_{tgt}||_2 \tag{28}$$

if we have

$$d(P_{end}(t), P_{tgt}) \leq d(P_{end}(t-1), P_{tgt}) \tag{29}$$

then, the reward is defined as the distance gained to the target

$$R(s_t, a_t) = \frac{d(P_{end}(t-1), P_{tgt}) - d(P_{end}(t), P_{tgt})}{d(P_{end}(t), P_{tgt})} \tag{30}$$

Note,

$$0 \leq R(s_t, a_t) \leq 1. \tag{31}$$

4. When the arm moves far from the target, based on the similar notion for in 3, we define a reward as a marginal penalty as how far is it moved away from the target as:

if

$$d(P_{end}(t), P_{tgt}) \geq d(P_{end}(t-1), P_{tgt}) \tag{32}$$

$$R(s_t, a_t) = \frac{d(P_{end}(t-1), P_{tgt}) - d(P_{end}(t), P_{tgt})}{d(P_{end}(t), P_{tgt})} \tag{33}$$

Note,

$$-1 \leq R(s_t, a_t) \leq 0. \tag{34}$$

TABLE III
TABLE III. REWARD TABLE

Category	Description	Note
$d(P_{end}(t), P_{tgt}) \leq d(P_{end}(t-1), P_{tgt})$	$0 \leq R(s_t, a_t) \leq 1$	...
$d(P_{end}(t), P_{tgt}) \geq d(P_{end}(t-1), P_{tgt})$	$-1 \leq R(s_t, a_t) \leq 0$	...

Now, let's take a look at the computation of a reward function.

Example 2. *(Reward function calculation). Suppose* $d(P_{end}(t-1), P_{tgt}) = \sqrt{6.25}$ *and* $d(P_{end}(t), P_{tgt}) = \sqrt{4.25}$ *Find a reward function* $R(s_t, a_t)$.

Sol. $\widetilde{R}(s_t, a_t) = d(P_{end}(t-1), P_{tgt}) - d(P_{end}(t1), P_{tgt}) = \sqrt{6.25} - \sqrt{4.25}$.

Note, check the code from the github source [???] after down loading, we have the following folder structure:

```
3-23-fd100/~/source/RobotArmMLAgentUnity$
.
 Assets
 Library
 LICENSE
 Logs
 Packages
 ProjectSettings
 README.md
 results
 Temp
 trainer_config.yaml

7 directories, 3 files
```

The code snippet from the Assests/scripts/RobotControllerAgent.cs :

```
92    public override void
          OnActionReceived(float[] vectorAction)
93    {
...
111   float diff = beginDistance - distance;
112
113   if (distance > prevBest)
114      {
115      // Penalty if the arm moves away from target
116        AddReward(prevBest - distance);
117      }
118      else
119      {
120      // Reward if the arm moves closer to target
121        AddReward(diff);
122        prevBest = distance;
123      }
124      AddReward(stepPenalty);
125       }
126    }
```

IX. Policy On Robot Operations (Under Construction)

Definition 3. *Policy π. A policy π in Robotics is a set of guidelines for a robot controller to follow to deliver its control action a_i upon its current state s_i, which is denoted as $\pi(a_i|\ s_i)$.*

A policy π leads to the robot control to deliver its control action as a mapping function $s_i \rightarrow a_i$.

A policy π can be either stochastic which is characterized by a conditional probability as

$$\pi(a_i|s_i) : s_i \rightarrow Pr(a_i|s_i), \tag{35}$$

or deterministic

$$\pi(a_i|s_i) : s_i \rightarrow a_i = \mu(s_i). \tag{36}$$

X. Accelerated Reward Functions

The reward function from Unity implementation based on the published work on Soft Actor Critic (SAC) by google deep mind team [???] adopt the distance based reward function, which is a linear reward function with the bigger reward for larger distance of the robot end effector traveled towards the target. We define this reward function as Type I reward, which we belive does not serve the purpose of good reward policy. The following code of type I reward is given in the following list, and it is illustraded in Figure below as well. Note based on this reward function, Type I, the bigger the differnce between the end effector and the target, the bigger the reward, we belive this is either a bug or less thoughtful design.

Property 1. *Policy π. A policy π in Robotics is a set of guidelines for a robot controller to follow to deliver its control action a_i upon its current state s_i, which is denoted as $\pi(a_i|\ s_i)$.*

A policy π leads to the robot control to deliver its control action as a mapping function $s_i \rightarrow a_i$.

XI. Adaptive Reward Functions

XII. Accelerated-and-Accelerated Reward Policy (AARP)

XIII. Policy As DNN (Under Construction)

We now introduce a notation to policy π as π_θ where a set of variables which affects π. In deep reinforcement learning (DRL), a policy π_θ is formulated as a deep neural network (DNN), where θ is the general parameter storing all the networks weights and biases $W = (w_{i,j})$.

Definition 4. *Policy π_θ. A policy π_θ is a deep neural network (DNN), where θ is the collection of all parameters of the neural networks (NN) weights and biases, simply denoted as $W = (w_{i,j})$.*

A typical realization of π_θ is a gaussian Multi-Layer Perceptron (MLP) net, which samples the action to be taken from a gaussian distribution of actions over states as follows

$$\pi_\theta(a_i|s_i) = \frac{1}{\sqrt{(2\pi)^{n_a} det \sum_\theta(s)}} E[-\frac{(||a - \mu_\theta(s)||)^2}{2\sum_\theta(s)}] \tag{37}$$

Example 3.??? *(Distance Calculation.)*

Supppose we have $s_1 = -1, s_2 = 2$ and $a_1 = 10$ and $a_2 = 20$,

Find

(1) $\sum_\theta(s)$ and $det \sum_\theta(s)$,

(2) $\mu_\theta(s)$,

(3) $\pi_\theta(a_i|s_i) =?$

Sol:

(1) $\sum_\theta(s) = \frac{1}{2}(s_1 + s_2)$,

XIV. Coding Implementation

A. Re-write The Base Line Code

We define the following parameters explicitly.

1 *EndEff[i]* which defines the robot end effector's position in $x_w - y_w - z_w$ space. Clearly define *EndEff[i]* in $x_w - y_w - z_w$ space, e.g., $P_{end} = EndEff[i] = (x_{ef}(i), y_{ef}(i), z_{ef}(i))$,

2 Seperate relative movement from absolute movement. See figure below for illustration. The relative movement is the movement from *EndEff[i-1]* to *EndEff[i]*, while the absolute movement is the movement defined with reference in $x_w - y_w - z_w$ space, simply stated, it is *EndEff[i]*.

3 Define *gap[i]* as follows $P_{tgt} - P_{end}$.

4 Define distance *distance[i]* as follow $EndEff[i] - EndEff[i-1]$, e.g., which is the same as relative movement.

Include the base code here,

and include the modified base line code here.

XV. Performance Evaluations

Compute the averaged reward from the base-line algorithm and the proposed K_2 algorithm. Each of these 2 accumulated rewards plotted with step as an independent variable. We illustrated one of the accumulated reward function in the following figure. Note we define the steps which give the negative reward as region Ω_1 while the steps give the positive reward as region Ω_2, the cross-over point is the point to separate these two regions.

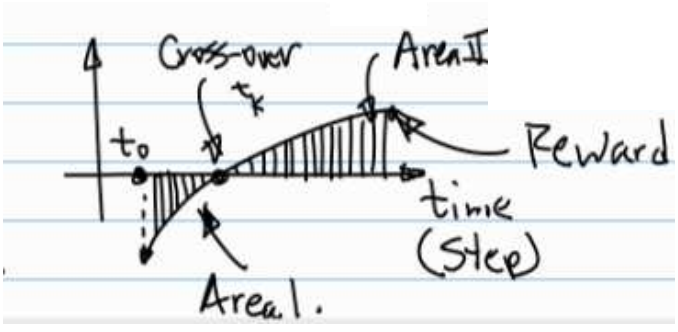

Fig. 8. Illustration of accumulated reward and the region Ω_1 and region Ω_2, the cross-over point is the point to separate these two regions.

1 For a better performance, we are expecting to have smaller accumulated negative reward, e.g., the smaller shaded region in region Ω_1 in figure **??**, and
2 the bigger accumulated positive reward, e.g., the bigger area in the region Ω_2.

A. Index for Accumulated Reward

Use the averaged outpout from both base-line algorithm and the proposed K_2 algorithm to calculate the accumulated reward as follows:

$$\int_{\Omega_N} r_1(t; s_t, a_t)\, dt + \int_{\Omega_P} r_1(t; s_t, a_t)\, dt \tag{38}$$

Computationally, we have

$$\int_{\Omega_N} r_1(t; s_t, a_t)\, dt = \sum_{t=1}^{t_k} r_1(t; s_t, a_t) \tag{39}$$

and

$$\int_{\Omega_P} r_1(t; s_t, a_t)\, dt = \sum_{t=t_{k+1}}^{t_N} r_1(t; s_t, a_t). \tag{40}$$

Define index for accumulated negative reward as

$$I_N = \sum_{t=1}^{t_k} r_1(t; s_t, a_t) \tag{41}$$

and the index for accumulated positive reward as

$$I_P = \sum_{t=t_{k+1}}^{t_N} r_1(t; s_t, a_t). \tag{42}$$

We introduce the second subscript for both equations 41 and 42 respectively. So we have $I_{N,K2}$ as the index covering the negative accumulated reward for the proposed K2 algorithm and $I_{N,B}$ fas the index of the accumulated reward for the base-line algorithm. Therefore,

$$\eta_N = \frac{I_{N,K2}}{I_{N,B}} \tag{43}$$

and

$$\eta_P = \frac{I_{P,K2}}{I_{P,B}} \tag{44}$$

B. Performance Comparison

To compare the performance of our proposed K_2 technique with the base-line algorithm by google deepmind team (with Unity implementation), we have the following guideline.

If $\eta_N < 1$, then the accumulated negative reward of the proposed K_2 algorithm performs better in generating lesser negative penalty. And if If $\eta_P > 1$ then the accumulated positive reward of the proposed K_2 algorithm performs better in generating more positive reward.

ACKNOWLEDGMENT

I would like to express my thanks to CTI One engineering member and CTI One engineering intern team, as well as to SJSU graduate research assistants for letting me introducing DRL in Robotics to our next project and for the codings in Deep Learning, Robotis and embedded systems.

REFERENCES

[1] [Franceschetti, 2020], Andrea Franceschetti, Elisa Tosello, Nicola Castaman, and Stefano Ghidoni, "Robotic Arm Control and Task Training through Deep Reinforcement Learning", https://arxiv.org/pdf/2005.02632.pdf, May 2020.
[2] [Reinforcement, 2020], Reinforcement learning, https://en.wikipedia.org/wiki/Reinforcement_learning, 2020.
[3] [Unity AI, 2020], https://github.com/Unity-Technologies/ml-agents 2020.