**Proceedings Paper:**
Kim, J (2022) State-space segmentation for faster training reinforcement learning. In: IFAC-PapersOnLine. 10th IFAC Symposium on Robust Control Design ROCOND 2022, 30 Aug - 02 Sep 2022, Kyoto, Japan. Elsevier , pp. 235-240.

https://doi.org/10.1016/j.ifacol.2022.09.352

# State-space segmentation for faster training reinforcement learning ⋆

**Jongrae Kim** *

* *Institute of Design, Robotics & Optimisation (iDRO), School of Mechanical Engineering, University of Leeds, Leeds LS1 9JT, the UK, (email: menjkim@leeds.ac.uk).*

**Abstract:** Nonlinear control problems have been the main subjects in control engineering from theoretical and applicational aspects. Reinforcement learning shows promising results for solving highly nonlinear control problems. Among many variants of reinforcement learning, Deep Deterministic Policy Gradient (DDPG) considers continuous control signals, which makes it an ideal candidate for solving nonlinear control problems. The training requires frequently, however, a large number of computations. To improve the convergence of DDPG, we present a state-space segmentation method dividing the state-space to expand the target space defined by the best reward. An inverted pendulum control example demonstrates the performance of the proposed segmentation method.

*Keywords:* reinforcement learning, learning convergence, reward, linear control

## 1. INTRODUCTION

Nonlinear control problems have been the main subjects in control engineering from theoretical and applicational aspects (Slotine et al., 1991; Kokotović and Arcak, 2001; Khalil, 2002). While there are rich systematic control design results, they mostly rely on minimalistic versions of nonlinear system descriptions (Mao and Billings, 1997; Ronch et al., 2012; Ruderman, 2018). In practice, most of the control algorithm design efforts have been devoted to trial-error or optimization-based design parameter tuning (Pozo et al., 2008; Smith, 2009; Garriga and Soroush, 2010).

A new opportunity arises to solve non-minimalistic nonlinear control problems. Computation speed increase by faster CPU (Central Processing Unit), multi-core CPU and massive numbers of parallel processing using GPU (Graphical Processing Unit) has promoted computationally intensive algorithms (Owens et al., 2008). Deep learning is one of the most successful uses of computing power and speed improvements (Goodfellow et al., 2016). Reinforcement learning is a promising control algorithm that exploits increased computation power (Sutton and Barto, 2018). It shows promising results for solving highly nonlinear control problems, especially in the field of robotics (Ghadirzadeh et al., 2021; Ji et al., 2021).

Among many variants of reinforcement learning, DDPG (Deep Deterministic Policy Gradient) operates over continuous control signals rather than a finite number of discrete actions (Lillicrap et al., 2015) and is an ideal candidate for solving nonlinear control problems (Tuyen and Chung, 2017; Xu et al., 2019; Wang et al., 2020). However, a convergence issue in the training of DDPG highlights the necessity of improvements.

⋆ A 2-page early version of the paper in Korean is presented at the ICROS2021 Conference, 23-25 June 2021, Yeosoo, Republic of Korea.

There have been several attempts to improve the training convergence with different motivations. For example, exploiting stable trajectory generations for quadcopter (Hwangbo et al., 2017), embedding linear controllers in reinforcement learning (Fernandez et al., 2020), a linear combination of linear controllers with reinforcement learning (Yoo et al., 2020), task segmentation (Kamio et al., 2004), and incremental state-space segmentation for learning (Takahashi et al., 1996) are some of the previous attempts.

We present a state-space segmentation algorithm to improve the convergence of DDPG in the training phase based on the modification of the reward function definition. The idea originates from the fact that the best reward of DDPG is frequently a pointwise state in the state-space, which makes it extremely challenging for the gradient descent algorithm used in the training phase to arrive at the state corresponding to the best reward.

The rest of the paper is organized as follows: firstly, we present the summary of DDPG; secondly, we present the state-space segmentation algorithm as the main part of the paper; thirdly, a simulation example shows the efficiency of the proposed algorithm; finally, we conclude the paper including future works.

## 2. NONLINEAR CONTROL PROBLEM & DDPG

**Nonlinear Control Problem:** The following differential equation provides one of the general forms of nonlinear control problems (Slotine et al., 1991):

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})\mathbf{u} \tag{1}$$

where $\dot{\mathbf{x}} = d\mathbf{x}/dt$, $t$ is the time, $\mathbf{x}$ is the $n$-dimensional state in $\Re^n$, $n$ is a positive integer, $\Re^n$ is the real number set, $\mathbf{u}$ is the $m$-dimensional control input in $U$, which is a compact set in $\Re^m$, $m$ is a positive integer, and $\mathbf{f}(\cdot)$ and $\mathbf{g}(\cdot)$ are nonlinear functions. The right-hand side of (1)

satisfies the existence and uniqueness conditions for the solution of the nonlinear differential equation, (1).

**Deep Deterministic Policy Gradient:** This is the summary of DDPG based on (Lillicrap et al., 2015) and (Keras: Code examples – reinforcement learning). Define the discounted return, $R_k$,

$$R_k = r(\mathbf{x}_k) + \sum_{i=k+1}^{T_f} \gamma^{i-k} r(\mathbf{x}_i) \tag{2}$$

where $k$ is the $k$-th step action instance of the reinforcement learning, $r(\mathbf{x}_k)$ is the instantaneous reward at $k$ as a function of the current state, $\mathbf{x}_k$, $T_f$ is the final time step, and $\gamma$ in $[0, 1]$ is the discount rate of the future reward. The discount return can be written in the Bellman equation form as follows:

$$R_k = r(\mathbf{x}_k) + \gamma R_{k+1}(\mathbf{x}_{k+1})$$

The critic network, $Q^{\boldsymbol{\pi}}(\mathbf{x}_k, \mathbf{u}_k)$, is the discounted return under the current deterministic control policy, $\boldsymbol{\pi}(\mathbf{x})$.

$$Q^{\boldsymbol{\pi}}(\mathbf{x}_k, \mathbf{u}_k) = \mathbb{E}[R_k | \mathbf{x}_k, \mathbf{u}_k]$$

where $\mathbb{E}(\cdot)$ is the expectation, the control input, $\mathbf{u}_k$, is given by

$$\mathbf{u}_k = \boldsymbol{\pi}(\mathbf{x}_k) + \mathbf{n}_k$$

and $\mathbf{n}_k$ is the noise to make the training algorithm to explore the action space during the learning phase of DDPG. $\mathbf{n}_k$ is typically modelled as the Ornstein-Uhlenbeck process. The critic network is written in the Bellman equation expression as follows:

$$Q^{\boldsymbol{\pi}}(\mathbf{x}_k, \mathbf{u}_k) = \mathbb{E}\left\{ r(\mathbf{x}_k) + \gamma Q^{\boldsymbol{\pi}}\left[\mathbf{x}_{k+1}, \boldsymbol{\pi}(\mathbf{x}_{k+1})\right] \right\}$$

where $\mathbf{x}_{k+1}$ is stochastic because of the exploration noise, $\mathbf{n}_k$, in $\mathbf{u}_k$, propagated by (1) from $t_k$ to $t_{k+1}$.

The critic function constructs its approximation using deep neural networks.

$$Q^{\boldsymbol{\pi}}(\mathbf{x}_k, \mathbf{u}_k) \approx Q^{\boldsymbol{\pi}}_{\boldsymbol{\theta}_Q}(\mathbf{x}_k, \mathbf{u}_k) = h^L\left[\mathbf{w}_L^T \mathbf{h}^{L-1}(\ldots) + b_L\right]$$

where $L$ is the number of layer, $h^L(\cdot)$ is the output layer activation function, $\mathbf{h}^{L-1}$ is the $(L-1)$-th hidden layer activation function, $\mathbf{w}_L$ is the output layer weighting vector, $(\cdot)^T$ is the transpose, $b_L$ is the output layer bias,

$$\mathbf{h}^p(\ldots) = \mathbf{h}^p\left[W_p \mathbf{h}^{p-1}(\ldots) + \mathbf{b}_p\right]$$

for $p = L-1, L-2, \ldots, 3, 2$, $W_p$ is the $p$-th layer weighting matrix, $\mathbf{b}_p$ is the $p$-th layer bias vector,

$$\mathbf{h}^1(\ldots) = \mathbf{h}^1\left\{ W_1 \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix} + \mathbf{b}_1 \right\}$$

$W_1$ is the first layer weighting matrix, and $\mathbf{b}_1$ is the first layer bias vector. The decision variable of the critic function approximation, $\boldsymbol{\theta}_Q$, includes $\mathbf{w}_L$, $b_L$, $W_p$, and $\mathbf{b}_p$ for $p = 1, 2, \ldots, L-2, L-1$.

Similarly, a deep neural network provides the approximation of the actor or control policy function.

$$\boldsymbol{\pi}(\mathbf{x}_k) \approx \boldsymbol{\pi}_{\boldsymbol{\theta}_\pi}(\mathbf{x}_k)$$

where $\boldsymbol{\theta}_\pi$ includes all weighting matrices and bias vectors of the neural network approximation of the control policy.

Set the initial networks of the target actor, $Q^{\boldsymbol{\pi}'}_{\boldsymbol{\theta}'_Q}$, and the target policy, $\boldsymbol{\pi}_{\boldsymbol{\theta}'_\pi}$, equal to $Q^{\boldsymbol{\pi}}_{\boldsymbol{\theta}_Q}$ and $\boldsymbol{\pi}_{\boldsymbol{\theta}_\pi}$, respectively.

**DDPG Training:** The optimization problem for the critic network update is given by

$$\underset{\boldsymbol{\theta}_Q}{\text{Minimize}}\ L = \frac{1}{N}\sum_{i=1}^{N} \left[\Delta Q_i(\boldsymbol{\theta}_Q)\right]^2 \tag{3}$$

where

$$\Delta Q_i(\boldsymbol{\theta}_Q) = \left\{ r(\mathbf{x}_k) + \gamma Q^{\boldsymbol{\pi}'}_{\boldsymbol{\theta}'_Q}\left[\mathbf{x}_{i+1}, \boldsymbol{\pi}_{\boldsymbol{\theta}'_\pi}(\mathbf{x}_i)\right] \right\} - Q^{\boldsymbol{\pi}}_{\boldsymbol{\theta}_Q}(\mathbf{x}_i, \mathbf{u}_i)$$

and $N$ is the number of transitions, $(\mathbf{x}_i, \mathbf{u}_i, r_i, \mathbf{x}_{i+1})$, selected from the stored transition set, where $r_i = r(\mathbf{x}_i)$ is obtained by the simulations of each episode. Update the critic network weighting as follows: $\boldsymbol{\theta}_Q \leftarrow \arg\min L(\boldsymbol{\theta}_Q)$.

As the reinforcement learning control policy is to maximize the reward function, $J = \mathbb{E}(R_1)$, the policy gradient is given by

$$\nabla_{\boldsymbol{\theta}_\pi} J \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_{\boldsymbol{\pi}} Q^{\pi}_{\boldsymbol{\theta}_Q}\left[\mathbf{x}_i, \boldsymbol{\pi}_{\boldsymbol{\theta}_\pi}(\mathbf{x}_i)\right]\ \nabla_{\boldsymbol{\theta}_\pi} \boldsymbol{\pi}_{\boldsymbol{\theta}_\pi}(\mathbf{x}_i) \tag{4}$$

At its simplest, $\boldsymbol{\theta}_\pi$ updates the value using the gradient as in

$$\boldsymbol{\theta}_\pi \leftarrow \boldsymbol{\theta}_\pi + \alpha \nabla_{\boldsymbol{\theta}_\pi} J \tag{5}$$

where $\alpha$ is a step-size parameter or a scaling factor. More advanced optimization algorithms such as the stochastic gradient descent or Adam (Adaptive Momentum Estimation) are used to train reinforcement learning (Baird and Moore, 1999; Kingma and Ba, 2014). These algorithms also use the gradient obtained in (4).

Finally, the target networks of the critic and the control policy function approximations are updated by

$$\boldsymbol{\theta}'_Q \leftarrow \tau \boldsymbol{\theta}_Q + (1-\tau)\boldsymbol{\theta}'_Q \tag{6a}$$

$$\boldsymbol{\theta}'_\pi \leftarrow \tau \boldsymbol{\theta}_\pi + (1-\tau)\boldsymbol{\theta}'_\pi \tag{6b}$$

where typically small values of $\tau$ between 0 and 1 provide stability in the algorithm convergence.

DDPG training repeats the procedures between (3) and (6) until the target networks, $Q^{\boldsymbol{\pi}'}_{\boldsymbol{\theta}'_Q}$ and $\boldsymbol{\pi}_{\boldsymbol{\theta}'_\pi}$, converge or the algorithm reaches to the maximum number of learning steps.

## 3. STATE-SPACE DIVISION

*Assumption 1.* From now on, without loss of generality, $r(\mathbf{x}_k)$ is assumed to be less than or equal to 0 for all $\mathbf{x}_k$ in the state-space, $\Re^n$.

Hence, the best possible value of $r(\mathbf{x}_k)$ is 0.

**Motivations:** Define the desired set, $S_D$, in the state-space

$$S_D = \{\mathbf{x} \in R^n \mid \mathbf{x} = \mathbf{x}_d\}$$

where the instantaneous reward, $r(\mathbf{x})$, is equal to zero for $\mathbf{x} \in S_D$. When the state arrives and stays in the desired state, all subsequent discounted returns, $R_k$, are zero and the policy cost function, $J$, converges to a finite value.

For the regulation control problems, the desired set is defined using a desired constant state vector, $\mathbf{x}_d$, and is an isolated point in the state-space. For the tracking control problem, $\mathbf{x}_d$ is a vector-valued function of time, $\mathbf{x}_d(t)$, which is continuous in time, $t$, and $S_D$ is a one-dimensional manifold in the state-space, $\mathcal{R}^n$. For both the control design problems, the desired sets have measure zero in $\mathcal{R}^n$. Arriving a measure zero set provides challenges

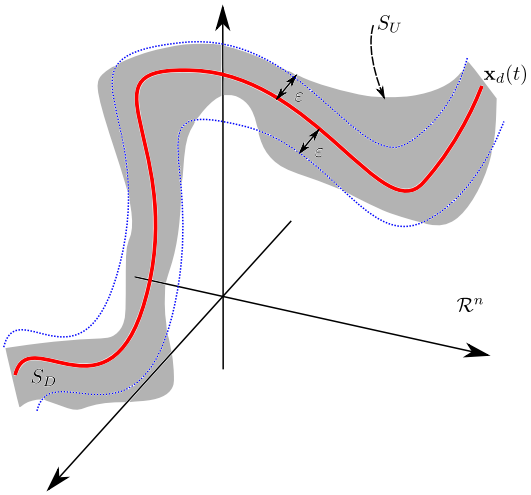


Fig. 1. State-space segmentation: $S_D$ is the desired target set, and $S_U$ is the region providing $\mathbf{u}(\mathbf{x}) \in U$ for $\mathbf{x} \in S_U$.

for the policy gradient algorithm, (5), to find the optimal parameters, $\boldsymbol{\theta}_\pi$, through the stochastic gradient algorithm or any other stochastic gradient based algorithm such as Adam (Kingma and Ba, 2014).

Measure-zero target state-space requires long $T_f$, and premature terminations of simulation episodes lead to poorly designed controllers or no convergence. Simulating dynamic systems is the most expensive part of the training. The longer $T_f$, the longer computing time is required to complete each episode. These motivate the necessity to define nonmeasure-zero target sets.

**Nonmeasure-zero set:** Define a nonmeasure-zero set as follows:

$$S_D = \{\mathbf{x} \in \mathcal{R}^n \mid \|\mathbf{x} - \mathbf{x}_d\| < \varepsilon\} \tag{7}$$

where $\|\cdot\|$ is one of the vector norms, and $\varepsilon$ is a positive real number. Figure 1 shows $S_D$ in $\mathcal{R}^n$ space. $S_D$ is the space surrounded by the dashed lines, whose distances from the desired state, $\mathbf{x}_d(t)$, are $\varepsilon$.

**State-space segmentation:** We propose to use the conventional control in the neighbourhood of desired state, $\mathbf{x}_d$. In the first step of the DDPG design iteration, the target set is defined by

$$S_T = S_D \cap S_U$$

where $S_U$ is the region satisfying the control constraint, i.e.,

$$S_U = \{\mathbf{x} \in \Re^n \mid \mathbf{u}_T(\mathbf{x}, \mathbf{x}_d) \in U\}$$

$\mathbf{u}_T(\mathbf{x}, \mathbf{x}_d)$ to be designed controls the states in $S_T$. In the first step, the initial conditions for DDPG training are sampled from the set $S_1$, which is a subset of $S_T^C$. And, $S_1$ is defined such that some of the boundaries of $S_1$ and $S_T$ are shared.

In the second step, the target set is $S_T \cup S_1$, and the initial conditions are sampled from the set $S_2$, which is a subset of $S_T^C - S_1$. And, $S_2$ is defined such that some of the boundaries of $S_2$ and $S_1$ are shared.

Similarly, in the third step, the target set is $S_T \cup S_1 \cup S_2$, and the initial conditions are sampled from the set $S_3$, which is a subset of $S_T^C - S_1 - S_2$. And, $S_3$ is defined such that some of the boundaries of $S_3$ and $S_2$ are shared.

The same step is repeated $p$-times such that

$$S_T^C = S_1 \cup S_2 \ldots \cup S_p$$

**Nonlinear control design:** For the states inside the nonmeasure-zero desired set for a chosen $\varepsilon$, we can use many existing control design methods, e.g., feedback linearization control, sliding mode control, backstepping control, linear quadratic regulator.

Various methods in nonlinear control theory can prove the stability of the closed-loop system with the feedback control. For example, Polycarpou and Ioannou (1993) presents a control design for a class of uncertain nonlinear systems to guarantee global uniform ultimate boundedness. We represent the control algorithm as $\mathbf{u} = \mathbf{u}_T(\mathbf{x}, \mathbf{x}_d)$ for $\mathbf{x} \in S_T$.

It is frequently, however, difficult to provide global stability proof with control constraints. Robust stability to modelling uncertainties and unmodelled dynamics becomes more challenging. In practice, we would have only the local stability proof and verification. One of the constraints frequently ignored in nonlinear control design is the control input constraint, i.e., the norm of $\mathbf{u}$ is bounded. As indicated in Figure 1, the shaded region is the feasible control input set, $S_U$, which does not necessarily coincide with $S_D$. The closed-loop system would become unstable if simply the boundary values of $U$ is chosen when the calculated control input is outside of the feasible control input region (Johnson et al., 2000; Alasty and Salarieh, 2007).

**Reward for training DDPG in $S_1$:** While $\mathbf{u}_T(\mathbf{x}, \mathbf{x}_d)$ controls the system at $\mathbf{x} \in S_T$, a DDPG to be designed controls the system at $\mathbf{x} \in S_1$, i.e., $\mathbf{u} = \mathbf{u}_1(\mathbf{x}, \mathbf{x}_d)$ for $\mathbf{x} \in S_1$. The main question is how to define the reward in the target set so that the DDPG design of $S_1$ leads all initial states of $S_1$ to converge to $S_T$. The importance of reward functions in algorithm convergence has been well known since the early days of reinforcement learning research (Mataric, 1994).

For ideally designed DDPG controllers in $S_1$ forcing the states converging to $S_T$, given that each episode runs long enough, i.e., $T_f \gg t^*$, the following inequality is satisfied:

$$\|\mathbf{x}(t) - \mathbf{x}_d(t)\| < \varepsilon \text{ for } t > t^*$$

as $\mathbf{x}(t)$ is in $S_T$, where $t^*$ is the minimum time satisfying the inequality. If the states converge to the target set, the instantaneous reward, $r_k$, in (2) is set to zero as follows:

$$r(\mathbf{x}_k) = 0 \text{ for } t_k > t^*$$

where $\mathbf{x}_k = \mathbf{x}(t_k)$, and $\mathbf{x}_k \in S_T$.

With the nonmeasure-zero target set, $S_T$, the convergence of the stochastic gradient based optimizer, (5), for example, would be improved in $S_1$. The reward, $r(\mathbf{x}_k)$ for $\mathbf{x}_k \in S_1$, uses the following quadratic function, which is common for many control designs including LQR (Linear Quadratic Regulator) controller:

$$r(\mathbf{x}_k) = -\mathbf{e}^T Q \mathbf{e} - \mathbf{u}^T R \mathbf{u} \tag{8}$$

where $\mathbf{e} = \mathbf{x} - \mathbf{x}_d$, and $Q$ and $R$ are the weighting matrices for the error and the control energy.

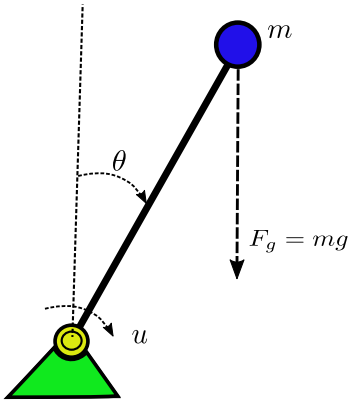


Fig. 2. Inverted pendulum control problem

**Reward for training DDPG in $S_k$ for $k \geq 2$:** Similar to the previous design step, the purpose of DDPG controller at $S_k$ is driving the states in $S_k$ to $S_{k-1}$ for $k \geq 2$. Hence, the reward for $S_T \cup S_1 \cup \ldots S_{k-1}$ is equal to zero, and the reward in $S_k$ is equal to the quadratic form given in (8).

**Initial condition, $\mathbf{x}(0)$, for DDPG training:** Ideally, the DDPG controller designed in $S_i$ would drive the state of $S_i$ to converge to the states of $S_{i-1}$ for $i \geq 1$, where $S_0 = S_T$. In addition, if the DDPG would provide the bounded stability, the state would not come back to $S_i$ after it goes into $S_{i-1}$. Hence, for the training episodes to obtain $\mathbf{u} = \mathbf{u}_i(\mathbf{x}, \mathbf{x}_d)$ for $i \geq 1$, the initial conditions, $\mathbf{x}(0)$, chosen in the set $S_i$ would be sufficient to provide diverse scenarios.

In general, the bounded stability guarantees require certain conditions for the systems, and dynamic simulators for the DDPG training would include more detailed models such as actuator dynamics, sensor models, higher-frequencies modes, etc, which are ignored in the control design. The states in $S_{i-1}$ might escape the set and come back to $S_i$, and the controller must learn these cases to make sure the state forces back to $S_{i-1}$. If there are oscillatory trajectories between two sets, these must be also learned during the training phase. Hence, the initial conditions for the controller in $S_i$ for $i \geq 1$ are sampled from $I_i$, which is the subset of $S_{i-1} \cup S_i$, and $I_i$ is defined by

$$I_i = \{\mathbf{x}(0) \mid \mathbf{x}(0) \in S_i \text{ or } \|\mathbf{x}_i^* - \mathbf{x}(0)\| \leq \delta_i \text{ for } \mathbf{x}(0) \in S_{i-1}\}$$

where $\delta_i$ is positive constant to be chosen, and $\mathbf{x}_i^*$ is the closest point in $S_i$ from $\mathbf{x}(0)$. The initial condition belongs to $S_1$, or the distance to $S_1$ is less than $\delta_i$ if it belongs to $S_{i-1}$.

We design sequentially DDPG controller for $S_i$ from $i = 1$ through the DDPG training procedure. Algorithm 1 provides the summary of the design procedures. The number of state-space segmentation might be undetermined initially, but it increases sequentially, starting at $i = 1$ until it covers the entire state space.

## 4. EXAMPLE

We demonstrate the proposed state-space segmentation approach using the inverted pendulum control problem. The inverted-pendulum stabilisation problem is shown in Figure 2. From the free-body diagram in Figure 2, the equation of motion is given by

$$\ddot{\theta} = \frac{g}{\ell} \sin\theta + \frac{1}{m\ell^2} u \tag{9}$$

where $g$ is the gravitational acceleration equal to 9.81 m/s$^2$, $\ell$ is the length of the pendulum set to 1 m, $m$ is the mass attached at the tip of the pendulum equal to 2 kg, and $u$ is the control torque, whose range is between $-2$ Nm and $+2$ Nm. The desired angle and angular velocities are equal to zero. The number of state-space segmentation, $p$, is set to 3.

The desired set is defined as $S_D = B(5, 5)$, where

$$B(\alpha, \beta) = \left\{(\theta, \dot{\theta}) \mid |\theta| < \alpha[^\circ], |\dot{\theta}| < \beta[^\circ/\text{s}]\right\}$$

In the target set, $S_T$, we design a feedback linearization based controller as follows:

$$u_T(\theta, \dot{\theta}) = (m\ell^2)\left(-\frac{g}{\ell}\sin\theta - k_p\theta - k_d\dot{\theta}\right)$$

and the error dynamics is given by $\ddot{\theta} + k_d\dot{\theta} + k_p\theta = 0$, where $k_p = 22$ and $k_d = 4$ provide the closed-loop being Hurwitz stable as the real parts of the poles are at $-2.0$. The set, $S_U$, is given by

$$S_U = \left\{(\theta, \dot{\theta}) \mid |u_T(\theta, \dot{\theta})| \leq 2\right\}$$

To train the DDPG controller in $S_1$, we define $S_1$ as

$$S_1 = B(10, 10) - S_T$$

where it is assumed that $B(10, 10) \supset S_T$. The initial conditions are uniformly sampled in the following set: $I_1 = B(10, 10)$, which covers $S_1$ and $S_T$ by definition.

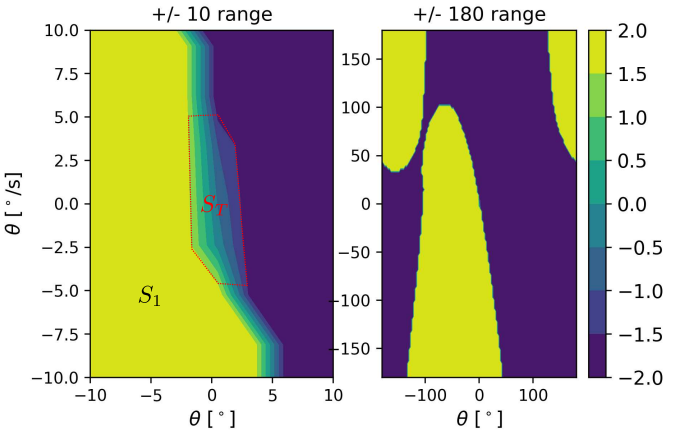


Fig. 3. DDPG with feedback linearization control

**Algorithm 1** Training with State-Space Segmentation

1: Define $S_0$, set $i = 1$
2: **while** $i = 1$ or $S_T^C \neq \cup_{k=1}^{i} S_i$ **do**
3: Define $S_i$ and $I_i$
4: Set the reward as follows:

$$r(\mathbf{x}_k) = 0, \text{ for } \mathbf{x}_k \in S_0, S_1, \ldots, \text{ or } S_{i-1}$$
$$r(\mathbf{x}_k) = -\mathbf{e}^T Q \mathbf{e} - \mathbf{u}^T R \mathbf{u}, \text{ for } \mathbf{x}_k \in S_i$$

5: Train DDPG to obtain $\mathbf{u}_i(\mathbf{x}, \mathbf{x}_d)$ for $S_i$, where $\mathbf{u}_k(\mathbf{x}, \mathbf{x}_d)$ control the states in $S_k$ for $k = 0, 1, \ldots, i-1$
6: **end while**

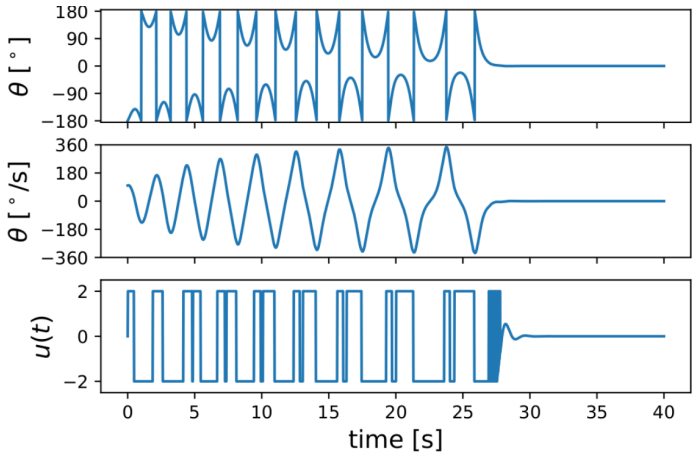


Fig. 4. Inverted pendulum control time history

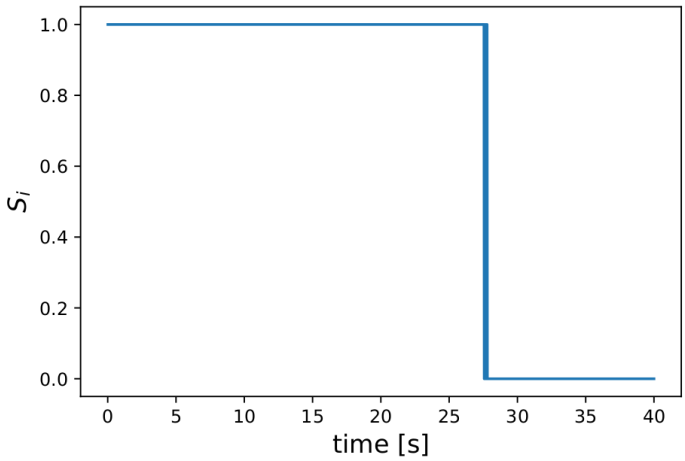


Fig. 5. State-space segmentation switching time history

The DDPG algorithm implemented in (Keras: Code examples – reinforcement learning) is used to train the DDPG design in the above, where the observation state to the controller, $u_1$, is $(\cos\theta, \sin\theta, \dot{\theta})$, i.e., the normalized rectangular coordinates of the mass and the angular velocity. The pendulum model in the code is modified to match with the differential equation, (9). A total of 50,000 episodes are used in the design for the DDPG in $S_1$.

Figure 3 shows the control output with respect to the states: angle and angular velocity. The subfigure on the left hand side shows the control output in $S_T$ and the part of $S_1$ where the initial conditions are for the training episodes. The state-space region surrounded by the dashed line corresponds to $S_T$. The trained DDPG smoothly connects to $S_T$, where the nonlinear controller is active. As the control input constraints are severely restrictive, the region is smaller than $S_D$. Hence, a feedback linearization based controller alone without considering the constraint would not be able to stabilize the pendulum outside of $S_T$.

Although the initial conditions are between $\pm 10°$ and $\pm 10°/\text{s}$ for training the DDPG in $S_1$, the exploration nature of reinforcement learning covers the whole region of the state-space as shown in the designed controller state mapping to actions shown on the right hand side of the figure. Unlike linear controllers would produce -2 Nm or +2 Nm for $\text{sgn}(\theta, \dot{\theta}) = (+1, +1)$ or $(-1, -1)$, where sgn() is the sign function, it learns the nonlinear controller behaviour exploiting the initial angular velocity to reach the desired position with the same initial rotational direction.

Figure 4 shows example trajectories and control input history for $\theta(0) = -180°$ and $\dot{\theta}(0) = 100\,°/\text{s}$. The settling time is around 29 seconds. It is typical to use approximate models to train reinforcement learning control to reduce the simulation time, and an Euler method based model of the differential equation, (9), with the step size equal to 0.2 s is used in the training. The Runge-Kutta method with higher numerical precision, on the other hand, provides the time history simulations for the designed controller.

Figure 5 shows the corresponding switching history. As the nonlinear controller in $S_T$ does not provide the region of attraction the same size as the set itself, $S_T$, the trajectory could escape from $S_T$, and the DDPG acts on forcing the state back into $S_T$.

## 5. CONCLUSIONS & FUTURE WORKS

We present a state-space segmentation method to improve the convergence of reinforcement learning training. The idea is motivated by enlarging the convergent area during the training phase of reinforcement learning. We demonstrate the performance using an inverted pendulum control problem. DDPG is combined with a nonlinear controller, and the combined controller stabilizes the full range of state-space. Current future works include providing the convergence guarantee of the states in $S_i$ to $S_{i-1}$ by introducing additional conditions in the training phase.

## ACKNOWLEDGEMENTS

This research was supported by Unmanned Vehicles Core Technology Research and Development Program through the National Research Foundation of Korea (NRF) and Unmanned Vehicle Advanced Research Center(UVARC) funded by the Ministry of Science and ICT, Republic of Korea.

## REFERENCES

Alasty, A. and Salarieh, H. (2007). Nonlinear feedback control of chaotic pendulum in presence of saturation effect. *Chaos, Solitons & Fractals*, 31(2), 292–304. doi: https://doi.org/10.1016/j.chaos.2005.10.004.

Baird, L. and Moore, A.W. (1999). Gradient descent for general reinforcement learning. *Advances in neural information processing systems*, 968–974.

Fernandez, G.I., Togashi, C., Hong, D.W., and Yang, L.F. (2020). Deep reinforcement learning with linear quadratic regulator regions. *arXiv preprint arXiv:2002.09820*.

Garriga, J.L. and Soroush, M. (2010). Model predictive control tuning methods: A review. *Industrial & Engineering Chemistry Research*, 49(8), 3505–3515.

Ghadirzadeh, A., Chen, X., Yin, W., Yi, Z., Björkman, M., and Kragic, D. (2021). Human-centered collaborative robots with deep reinforcement learning. *IEEE Robotics and Automation Letters*, 6(2), 566–571. doi: 10.1109/LRA.2020.3047730.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Hwangbo, J., Sa, I., Siegwart, R., and Hutter, M. (2017). Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4), 2096–2103. doi:10.1109/LRA.2017.2720851.

Ji, G., Yan, J., Du, J., Yan, W., Chen, J., Lu, Y., Rojas, J., and Cheng, S.S. (2021). Towards safe control of continuum manipulator using shielded multiagent reinforcement learning. *IEEE Robotics and Automation Letters*, 6(4), 7461–7468. doi:10.1109/LRA.2021.3097660.

Johnson, E., Calise, A., El-Shirbiny, H., and Eysdyk, R. (2000). Feedback linearization with neural network augmentation applied to x-33 attitude control. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*. doi:10.2514/6.2000-4157.

Kamio, T., Soga, S., Fujisaka, H., and Mitsubori, K. (2004). An adaptive state space segmentation for reinforcement learning using fuzzy-art neural network. In *The 2004 47th Midwest Symposium on Circuits and Systems, 2004. MWSCAS '04.*, volume 3, iii–117. doi:10.1109/MWSCAS.2004.1354305.

Keras: Code examples – reinforcement learning (2020). Deep deterministic policy gradient (DDPG). URL `https://keras.io/examples/rl/ddpg_pendulum/`.

Khalil, H.K. (2002). *Nonlinear systems; 3rd ed.* Prentice-Hall, Upper Saddle River, NJ.

Kingma, D.P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kokotović, P. and Arcak, M. (2001). Constructive nonlinear control: a historical perspective. *Automatica*, 37(5), 637–662. doi:https://doi.org/10.1016/S0005-1098(01)00002-4.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Mao, K.Z. and Billings, S.A. (1997). Algorithms for minimal model structure detection in nonlinear dynamic system identification. *International Journal of Control*, 68(2), 311–330. doi:10.1080/002071797223631.

Mataric, M.J. (1994). Reward functions for accelerated learning. In *Machine learning proceedings 1994*, 181–189. Elsevier.

Owens, J.D., Houston, M., Luebke, D., Green, S., Stone, J.E., and Phillips, J.C. (2008). GPU computing. *Proceedings of the IEEE*, 96(5), 879–899. doi:10.1109/JPROC.2008.917757.

Polycarpou, M.M. and Ioannou, P.A. (1993). A robust adaptive nonlinear control design. In *1993 American control conference*, 1365–1369. IEEE.

Pozo, F., Ikhouane, F., and Rodellar, J. (2008). Numerical issues in backstepping control: Sensitivity and parameter tuning. *Journal of the Franklin Institute*, 345(8), 891–905. doi:https://doi.org/10.1016/j.jfranklin.2008.05.005.

Ronch, A.D., Badcock, K., Wang, Y., Wynn, A., and Palacios, R. (2012). *Nonlinear Model Reduction for Flexible Aircraft Control Design*. doi:10.2514/6.2012-4404.

Ruderman, M. (2018). Minimal-model for robust control design of large-scale hydraulic machines. In *2018 IEEE 15th International Workshop on Advanced Motion Control (AMC)*, 397–401. doi:10.1109/AMC.2019.8371125.

Slotine, J.J.E., Li, W., et al. (1991). *Applied nonlinear control*, volume 199. Prentice hall Englewood Cliffs, NJ.

Smith, C.L. (2009). *Practical process control: tuning and troubleshooting*. John Wiley & Sons.

Sutton, R.S. and Barto, A.G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

Takahashi, Y., Asada, M., and Hosoda, K. (1996). Reasonable performance in less learning time by real robot based on incremental state space segmentation. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS'96*, volume 3, 1518–1524. IEEE.

Tuyen, L.P. and Chung, T. (2017). Controlling bicycle using deep deterministic policy gradient algorithm. In *2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, 413–417. doi:10.1109/URAI.2017.7992765.

Wang, M., Ruan, X., and Zhu, X. (2020). Heuristic gait learning of quadruped robot based on deep deterministic policy gradient algorithm. In *2020 Chinese Automation Congress (CAC)*, 1046–1049. doi:10.1109/CAC51589.2020.9326973.

Xu, J., Hou, Z., Wang, W., Xu, B., Zhang, K., and Chen, K. (2019). Feedback deep deterministic policy gradient with fuzzy reward for robotic multiple peg-in-hole assembly tasks. *IEEE Transactions on Industrial Informatics*, 15(3), 1658–1667. doi:10.1109/TII.2018.2868859.

Yoo, J., Jang, D., Kim, H.J., and Johansson, K.H. (2020). Hybrid reinforcement learning control for a micro quadrotor flight. *IEEE Control Systems Letters*, 5(2), 505–510.