学術・技術論文

強化学習によるロボットの動作獲得のための基底関数に基づく行動空間生成手法

山口明彦* 高松淳* 小笠原司*

Constructing Action Space from Basis Functions for Motion Acquisition of Robots by Reinforcement Learning

Akihiko Yamaguchi*, Jun Takamatsu* *and* Tsukasa Ogasawara*

Discrete action sets are often used in many reinforcement learning (RL) applications in robot control, since such sets are compatible with many RL methods and sophisticated architectures, such as Q(λ)-learning [1] and the Dyna. However, one of the problems is the absence of general principles on designing a discrete action set for robot control in higher dimensional input space. In this paper, we propose a discrete action set DCOB which is generated from the given basis functions (BFs) for approximating a value function. Though the DCOB is a discrete set, it has an ability to acquire high performance. Moreover, we utilize a method that generates a set of BFs based on the dynamics of the robot to reduce the number of the BFs. This way also makes the DCOB compact. Thus, the DCOB is compact and has an ability to acquire performance. Moreover, we also propose a method WF-DCOB, where the wire-fitting [2] is utilized to learn within a continuous action space which the DCOB discretizes. The purpose of the WF-DCOB is to evaluate the possibility of acquiring higher performance. Our proposition in the WF-DCOB is to constrain and initialize the parameters to relax the instability of the wire-fitting. We apply the proposed methods for a humanoid robot to learn crawling motion. The simulation results demonstrate outstanding advantages of the proposed method both in learning speed and ability to acquire performance, compared to conventional action space.

Key Words: Reinforcement Learning, Discrete Action Set, Continuous Action Space, Motion Learning, Crawling

1. はじめに

ヒューマノイドに代表される高機能ロボットは，歩行や跳躍など，多種の動作を行うことができる．しかしこれらの動作は事前にプログラムされたものであり，高機能ロボットの普及に向けて，エンドユーザ側で新しく動作目的を指示し，獲得させられるような技術が望まれる．このような動作目的から行動を獲得する技術の一つとして強化学習が研究されており，ロボットの行動獲得にも応用されている [3]〜[13]．しかしながら，高次元の制御入力空間 $\tilde{\mathcal{U}}$ を持つ高機能ロボットの扱いは，今なお未解決問題の一つである．

強化学習をロボティクスに応用した研究の多くでは，ロボットの制御入力空間 $\tilde{\mathcal{U}}$ が連続であるため，強化学習の行動空間 $\mathcal{U}$ も連続集合として定義される．もっとも単純なものとして，制御入力空間を直接ロボットの行動空間として扱う方法，すなわち，$\mathcal{U}=\tilde{\mathcal{U}}$ として学習する方法がある [4]〜[7]．制御入力空間 $\tilde{\mathcal{U}}$ の代わりに，強化学習で学習しやすい連続行動空間 $\mathcal{U}$ を構成する方法も提案されている [3] [8] [9]．しかしながら，高次元の連続行動空間における強化学習では，一般的に，(1) 適切にパラメタを初期化しないと，動作のパフォーマンス†が望ましくない局所解に陥ってしまう，(2) 行動価値関数の行動に関する最大化計算が困難なため Q(λ)-learning [1] などの手法の適用が困難，などの課題がある．

一方で連続行動空間を離散化した，離散行動集合もしばしば用いられる [10]〜[13]．この理由としては，(1) Q(λ)-learning [1] などの高速で安定した強化学習手法の適用が容易になる，(2) 価値関数の表現が単純になるため階層構造（例えば文献 [11]）や複数の学習器からなるシステム（例えば文献 [10]）のように，複雑なシステムの構築が簡潔になる，(3) Dyna アーキテクチャと相性がよい（例えば文献 [14]），などが考えられる．離散化の一般的な手法としては，行動空間の各次元を独立に離散化する方法などが存在する．しかしながらこのような方法では，離散行動集合の要素数が制御入力空間の次元数に対して指数関数的に増加してしまい，高次元の制御入力空間への適用が困難とな

原稿受付　2009 年 9 月 3 日
*奈良先端科学技術大学院大学
*Nara Institute of Science and Technology
■ 本論文は学術性で評価されました．

†厳密には学習後にグリーディ方策を実行して得られる収益．以下同．

るという問題がある．さらに，一般的に，離散化によるパフォーマンスの低下が起こり得る．

そこで本論文では，高次元の制御入力空間を強化学習で効率的に扱えるように，要素数が少なく，かつ高いパフォーマンスも維持できる離散行動集合の生成方法を提案する．まず，行動価値関数の近似のために与えられた基底関数から，高いパフォーマンスを獲得できる離散行動集合 **DCOB** の生成方法を提案する．さらに，ロボットやタスクの特性によって要素数を削減した基底関数集合を与えることで，パフォーマンスに加えて学習速度も向上させる．また，離散化前の連続行動空間で学習を行う方法 **WF-DCOB** を構築し，より高いパフォーマンス獲得の可能性について検討する．

DCOB では，基底関数が状態空間に配置されていて，パラメタとして中心を持つことを仮定する．DCOB の一つの行動は，行動開始時の状態から目標とする基底関数の中心に向かう軌道として設計される．ここで，一つの行動が引き起こす状態変化を最近傍の基底関数に遷移する程度に制限することで，基底関数の分解能に合わせた行動集合を生成する．このとき，DCOB の要素数は基底関数の数の数倍程度に抑えられる．基底関数集合の与え方としては，Wolpert ら [15] が提案し，Doya・Samejima ら [16] が強化学習に応用，山口ら [17] がロボットの運動学習に応用した，「ロボットのダイナミクスに基づく基底関数集合の生成（=状態空間の分節化）手法」を用いる．これにより，基底関数の数が状態空間の次元に対して指数関数的に増加する問題を解消でき，DCOB の要素数削減につながる．

さらに，離散行動集合 DCOB が離散化する前の連続行動空間で学習を行う方法 WF-DCOB を構築する．DCOB との比較を公平にするために，両者の学習アルゴリズムとして Peng らの Q(λ)-learning [1] を用いる．このとき，wire-fitting [2] を用いて行動価値関数の行動に関する最大化計算の困難さを解決する．ただし wire-fitting を用いると，パフォーマンスの低い局所解に陥ったり，学習が不安定になることがあるが，パラメタの初期化・制約方法を新たに提案することでこれらの問題を解決する．

提案手法をヒューマノイドロボットの全身運動学習の例として匍匐(ほふく)動作の獲得に適用した．このときロボットのダイナミクスに基づいて生成した基底関数集合のみでなく，強化学習で一般的に用いられる，グリッド上に配置した基底関数集合を用いた場合とも比較し，DCOB の有効性を検討した．この結果，グリッド配置の場合は，既存の離散行動集合で学習する場合や，単純な wire-fitting で連続行動空間を学習する場合と比較して高いパフォーマンスが獲得できることを示し，ダイナミクスに基づく配置の場合は，パフォーマンス，学習速度のいずれにおいても高い性能が得られることを示した．

関数近似器の基底関数の中心に向かうように行動を定義するアイディアは，山口ら [17] に基づく．しかし，同論文ですでに指摘されているように，文献 [17] の手法では動作のパフォーマンスを損なう可能性があった．本論文では，基底関数集合の分解能に合わせた行動集合の生成（「軌道短縮」ステップの追加）によってパフォーマンスを向上させる．さらに，(1) ある条件下で強化学習の収束性を保証，(2) 離散化する前の連続行動空間で学習を行う方法を構築，(3) より高次元のタスクで他手法と比較，を行った．

以下，2 章で強化学習の基礎について概説し，3 章で提案手法の基礎となる連続行動空間の定義を行う．4 章でこの連続行動空間を離散化して離散行動集合を作り，5 章では wire-fitting を用いてこの連続行動空間で直接学習する方法について述べる．6 章でダイナミクスに基づく基底関数の配置手法について述べ，7 章でヒューマノイドの運動学習に適用するシミュレーション実験を行う．8 章で収束性，計算コスト，使用できる基底関数，適用可能なタスク，関連研究について議論し，9 章でまとめる．

2. 強化学習の基礎

強化学習の目的は，入力が**状態** $x_n \in \mathcal{X}$ と**報酬** $R_n \in \mathbb{R}$，出力が**行動** $u_n \in \mathcal{U}$ であるシステム（**エージェント**）が，**割引収益**の期待値 $\mathbb{E}\left[\sum_{k=1}^{\infty} \gamma^{k-1} R_{n+k}\right]$ を最大化するような**方策** $\pi(x_n) : \mathcal{X} \to \mathcal{U}$ を獲得することと定義される．$n \in \mathbb{N} = \{0, 1, \dots\}$ は時間ステップ，$\gamma \in [0, 1)$ は**割引率**を表す．Q-learning や Sarsa などの強化学習手法では，**行動価値関数** $Q(x, u) : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ が状態 x で行動 u を取ったときに将来期待される割引収益を推定するように学習される．このとき，最適な行動は**グリーディ方策** $\pi(x) = \arg\max_u Q(x, u)$ によって得られる．

2.1 行動価値関数の近似器

状態空間 $\mathcal{X}$ や行動空間 $\mathcal{U}$ が連続である場合には，適当な関数近似器が行動価値関数 $Q(x, u)$ を近似するために用いられる．Peng らの Q(λ)-learning [1] における行動価値関数の更新則を一般的な関数近似器に対して書き下すと，

$$e_n = R_n + \gamma V_n(x_{n+1}) - V_n(x_n) \tag{1a}$$

$$e'_n = R_n + \gamma V_n(x_{n+1}) - Q_n(x_n, u_n) \tag{1b}$$

$$\theta_{n+1} = \theta_n + \alpha e_n Tr_n + \alpha e'_n \nabla_\theta Q_n(x_n, u_n) \tag{1c}$$

$$Tr_{n+1} = (\gamma\lambda)(Tr_n + \nabla_\theta Q_n(x_n, u_n)) \tag{1d}$$

となる．ただし $\nabla_\theta Q(x, u)$ は関数近似器のパラメタ $\theta \in \Theta$ に関して $Q(x, u)$ を微分したベクトルを表す．α はステップサイズパラメタ，$\lambda \in [0, 1]$ は減衰定数を表す．Tr_n は eligibility trace（$Tr_0 = \mathbf{0} \in \Theta$），$V_n(x) \triangleq \max_u Q_n(x, u)$ である．

2.2 NGnet: 連続状態・離散行動の線形関数近似器

状態空間 $\mathcal{X}$ のみが連続で，行動空間は離散の場合を考える．以下では離散行動空間を（離散）行動集合と呼び，$\mathcal{A} \ni a$ で表す．本論文では，Tsitsiklis ら [18] によって TD(λ)-learning の収束が保証されている線形の関数近似器を用いて，行動価値関数を以下のように近似する．

$$Q(x, a) = \theta_a^\top \phi(x) \tag{2}$$

$$\phi(x) = (\phi_1(x), \dots, \phi_{|\mathcal{K}|}(x))^\top \tag{3}$$

ただし $\theta_a \in \mathbb{R}^{|\mathcal{K}| \times 1}$ は行動 a に関するパラメタベクトル，$\phi(x)$ は特徴ベクトル（基底関数の出力），$\mathcal{K} = \{\phi_k \mid k = 1, 2, ..\}$ は事前に定義された**基底関数**の集合をそれぞれ表す．「価値関数が線形」とは，$\phi(x)$ のパラメタを学習の対象としないことを意味する．このときパラメタベクトルは $\theta^\top = (\theta_1^\top, \theta_2^\top, \dots, \theta_{|\mathcal{A}|}^\top)$ と表され，行動価値関数の微分は $\nabla_\theta Q_n(x, a) = \big(\delta(1, a)\phi(x)^\top,$

$\delta(2,a)\phi(x)^\top, \ldots, \delta(|\mathcal{A}|,a)\phi(x)^\top\big)^\top$ と表される．$\delta(i,j)$ はクロネッカーのデルタを表す．

本論文では，基底関数として，強化学習の研究でしばしば用いられる正規化ガウシアンネットワーク（NGnet: Normalized Gaussian Network）を仮定する [3] [5]．NGnet の基底関数は

$$\phi_k(x) = \frac{G(x;\mu_k,\Sigma_k)}{\sum_{k'\in\mathcal{K}} G(x;\mu_{k'},\Sigma_{k'})} \tag{4}$$

と表される．ただし $G(x;\mu,\Sigma)$ は平均 μ，共分散 Σ のガウス関数を表す．前述のように，本論文の強化学習ではこれらのパラメタは定数として扱われる．

2.3 Wire-fitting: 連続状態・連続行動の関数近似器

次に，状態空間 $\mathcal{X}$ と行動空間 $\mathcal{U}$ がともに連続の場合を考える．この場合，通常の関数近似器を用いると，グリーディ行動や $\max_u Q_n(x,u)$ の計算が複雑になる（ほとんどの場合，解析的に解けない）．この問題を解決する関数近似器の一つとして wire-fitting が提案されており [2]，以下のように定義される．

$$Q(x,u) = \lim_{\epsilon\to 0+} \frac{\sum_{i\in\mathcal{W}} (d_i+\epsilon)^{-1} q_i(x)}{\sum_{i\in\mathcal{W}} (d_i+\epsilon)^{-1}} \tag{5}$$

$$d_i = \|u-u_i(x)\|^2 + \mathrm{C}\big[\max_{i'\in\mathcal{W}}(q_{i'}(x)) - q_i(x)\big] \tag{6}$$

ここで，ある i における関数の組 $q_i(x):\mathcal{X}\to\mathbb{R}$ と $u_i(x):\mathcal{X}\to\mathcal{U}$ $(i\in\mathcal{W})$ は**制御ワイヤ**（*control wire*）と呼ばれ，wire-fitting は制御ワイヤ集合 $\mathcal{W}$ の補間関数とみなされる．C は補間の滑らかさを調整する定数を表す．定義から明らかなように，$q_i(x)$ は行動価値に関係し，$u_i(x)$ は行動に関係する．これらの関数を表現する関数近似器の種類にかかわらず，wire-fitting は以下の特徴を持つ．

$$\max_u Q(x,u) = \max_{i\in\mathcal{W}}(q_i(x)) \tag{7}$$

$$\arg\max_u Q(x,u) = u_i(x)\Big|_{i=\arg\max_{i'\in\mathcal{W}}(q_{i'}(x))} \tag{8}$$

このように，グリーディ行動の計算や $\max_u Q_n(x,u)$ の計算が，すべての制御ワイヤ $i\in\mathcal{W}$ について $q_i(x)$ を評価するだけで完了する．

本論文では文献 [19] を参考に，$q_i(x)$ を NGnet を基底関数とした線形近似器で表現し，$u_i(x)$ を単に定数ベクトルで表現する．すなわち，$q_i(x)=\theta_i^\top\phi(x)$, $u_i(x)=U_i$ とする．特徴ベクトル $\phi(x)$ は式 (3), (4) と同様に定義される．よって，行動価値関数のパラメタベクトルは，$\theta^\top=(\theta_1^\top, U_1^\top, \theta_2^\top, U_2^\top, \ldots, \theta_{|\mathcal{W}|}^\top, U_{|\mathcal{W}|}^\top)$ と定義できる．このとき行動価値関数の微分 $\nabla_\theta Q(x,u)$ は解析的に計算できる．

3. BFTrans: 基底関数から生成した連続行動空間

本論文で提案する離散行動集合 DCOB は，**Fig. 1** に示すように，基底関数集合を用いて連続行動空間 BFTrans を生成し，これを離散化することによって得られる．さらに BFTrans を直接学習するために，関数近似器として wire-fitting を用い，そのパラメタ制約を基底関数集合から生成する手法 WF-DCOB を提案する．そこで本章では，DCOB と WF-DCOB の基礎となる連続行動空間 BFTrans を定義する．

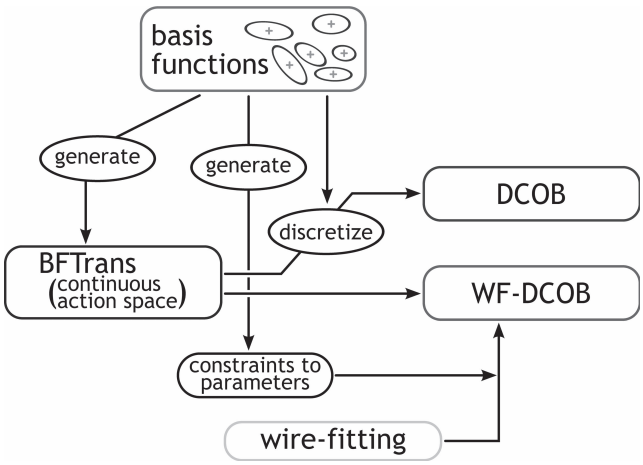


Fig. 1 Relation among the basis functions (BFs), the wire-fitting, and the proposed methods, BFTrans, DCOB, WF-DCOB. The continuous action space BFTrans is generated from the BFs. The discrete action set DCOB is obtained by discretizing the BFTrans based on the BFs. The WF-DCOB directly learns within the BFTrans using the wire-fitting where the constraints of the parameters of the wire-fitting are generated from the BFs

次章で述べるように，DCOB は基底関数の中心からなる集合が状態空間を離散化していることを利用して，行動空間の離散化を行う．これを実現するために，本章で定義する連続行動空間 BFTrans は，状態空間を行動空間の一部として含むように定義される．

なお，「行動空間を定義する」とは，実数ベクトル空間として定義された行動空間の要素 $u\in\mathcal{U}$ から，ロボットの制御入力 $\tilde{u}\in\tilde{\mathcal{U}}$ をある期間出力する「制御器」を構成することに等しい．以下では行動空間の要素から制御入力の系列を生成することを**実行する**という．BFTrans では，参照軌道を計画し，これを追従するように制御入力を出力する．

3.1 仮定

BFTrans を定義するために，以下の仮定をおく．

（**A**） 個々の基底関数 $k\in\mathcal{K}$ は，固定された中心 $\mu_k\in\mathcal{X}$ を状態空間上に持つ．

（**B**）「参照軌道」上の各点が定義される空間 $\mathcal{Q}$（例えば関節角空間）が与えられている．状態 $x\in\mathcal{X}$ から $q\in\mathcal{Q}$ を抽出する関数が $q=C_\mathrm{p}(x):\mathcal{X}\to\mathcal{Q}$ として与えられており，状態 $x\in\mathcal{X}$ から $q\in\mathcal{Q}$ の時間微分（例えば関節角速度）を抽出する関数が $\dot{q}=C_\mathrm{d}(x)$ として与えられている．

（**C**） 参照軌道 $q^\mathrm{d}(t)$ を追従する制御器，すなわち制御入力 $\tilde{u}\in\tilde{\mathcal{U}}$（例えばトルク）を参照軌道と現在の状態から計算する関数が $\tilde{u}(t)=Ctrl(x(t),q^\mathrm{d}(t+\delta\mathrm{t}))$（例えば PD 制御器）として与えられている．$\delta$t は制御周期を表す．

本論文で用いる NGnet の基底関数は明らかに（A）を満たす．（A）を満たせば NGnet 以外の基底関数でも使用可能である（8.4 節で議論）．中心 μ_k の固定を仮定しているのは，収束性

を保証するためである（8.1 節で議論）．

3.2 概要

ここで定義する連続行動空間 BFTrans の要素は，目標状態 $x^{\mathrm{trg}} \in \mathcal{X}$，および到達までの時間幅を決定する正の係数（**到達時間幅係数**）$g \in \mathbb{R}$ から構成される．すなわち $u = (g, x^{\mathrm{trg}}) \in \mathbb{R} \times \mathcal{X} \triangleq \mathcal{U}_{\mathrm{BFTrans}}$ と書ける．このとき，時間ステップ n，開始時刻 t_n で選択した行動 u_n は，以下の手順で実行される（**Fig. 2** 参照）．

Algorithm 1: Executing an action in the BFTrans

Input : 開始状態 $x_n = x(t_n)$,
行動 $u_n = (g, x^{\mathrm{trg}}) \in \mathbb{R} \times \mathcal{X} = \mathcal{U}_{\mathrm{BFTrans}}$

1: g, x_n, x^{trg} から到達時間幅 T_{f} を計算
2: **軌道生成**： x_n から x^{trg} に時間幅 T_{f} で遷移する参照軌道 $q^{\mathrm{d}}(t_n + t_a)$, $t_a \in [0, T_{\mathrm{f}}]$ を生成
3: **軌道短縮**： 参照軌道を $t_a \in [0, T_{\mathrm{n}}(x_n, u_n)] \subseteq [0, T_{\mathrm{f}}]$ に短縮
4: **軌道追従**： 短縮された参照軌道を，与えられた制御器で追従： $\tilde{u}(t_n + t_a) = Ctrl(x(t_n + t_a), q^{\mathrm{d}}(t_n + t_a + \delta \mathrm{t}))$, $t_a \in [0, T_{\mathrm{n}}(x_n, u_n))$
5: 行動 u_n が終了; $n \leftarrow n + 1$

行動に目標状態 $x^{\mathrm{trg}} \in \mathcal{X}$ を含めることによって，基底関数集合を用いた離散化が容易になり，さらに行動集合の多様性を維持できる．ただし，これだけだと，基底関数が状態空間全体に散らばっているため，「軌道生成」で生成される参照軌道は概して状態を大きく変化させ（Fig. 2 “Reference Trajectory”），これを一つの行動として動作（行動の系列）を生成すると，粗い動作が生成されてしまう．そこで，軌道を短縮することによって，行動集合の多様性を維持したまま，動作が細かくなるようにしている（Fig. 2 “Abbreviated Trajectory”）．この短縮は，より短くしたほうが正確に動作を表現できる．しかし，探索に掛かる時間が増加すること，および基底関数の配置の細かさ以上に細かく行動を選択できないことを考慮して，「近接する基底関数に遷移する程度」が強化学習に適していると判断した．なお「軌道短縮」を目標状態の変更ではなく行動時間の変更としているのは，後者の方法だと行動終了時点での加速度成分に多様性を持たせられるため，ダイナミックな行動の獲得に有利だと考えられるからである．

BFTrans のもう一つの利点として，状態空間の制約を考慮した行動が生成できることがあげられる．例えば関節角に制約があるロボットの場合，目標状態を関節可動領域の内部に設定することで，物理的に取ることができない行動が避けられる．基底関数集合を用いて離散化する場合も，通常は関節可動領域の内部または境界に基底関数を配置するため，この特徴を利用できる．

3.3 参照軌道の生成

参照軌道 $q^{\mathrm{d}}(t_n + t_a)$, $t_a \in [0, T_{\mathrm{f}}]$ は開始状態 $x_n = x(t_n)$ から x^{trg} に時間幅 T_{f} で到達するように計画される．この軌道の表現には，単純な三次関数を用いる．

$$q^{\mathrm{d}}(t_n + t_a) = c_0 + c_1 t_a + c_2 t_a^2 + c_3 t_a^3 \qquad (9)$$

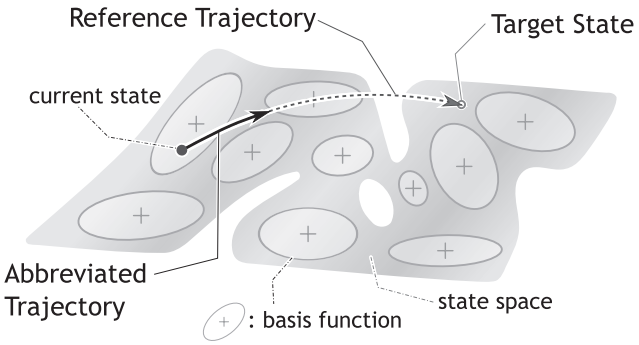


Fig. 2 Illustration of how an action in the BFTrans is *executed*. First, a reference trajectory is generated, then it is abbreviated. The reference trajectory may change the state greatly, so the obtained motion is coarse. To make the motion fine, the trajectory is abbreviated

この係数は以下の境界条件から計算される．

$$\begin{aligned} q^{\mathrm{d}}(t_n) &= C_{\mathrm{p}}(x_n), \quad q^{\mathrm{d}}(t_n + T_{\mathrm{f}}) = C_{\mathrm{p}}(x^{\mathrm{trg}}), \\ \dot{q}^{\mathrm{d}}(t_n + T_{\mathrm{f}}) &= C_{\mathrm{d}}(x^{\mathrm{trg}}), \quad \ddot{q}^{\mathrm{d}}(t_n + T_{\mathrm{f}}) = \mathbf{0} \end{aligned} \qquad (10)$$

ただし $\mathbf{0}$ はゼロベクトルを表す．境界条件の妥当性は，8 章で議論する．

「動作の素早さ」は到達時間幅 T_{f} のみでなく，x_n と x^{trg} の距離にも依存する．そこで，$\mathcal{Q}$ 空間での最大値ノルムで到達時間幅 T_{f} を正規化した到達時間幅係数 $g \in \mathbb{R}$ を行動として持たせることで，g にほぼ比例して動作の素早さが決まるようにしている．これにより，g も適切に離散化しやすくなる．

$$T_{\mathrm{f}} = g \delta C_{\mathrm{p}}(x_n, x^{\mathrm{trg}}) \qquad (11)$$

$$\delta C_{\mathrm{p}}(x_n, x^{\mathrm{trg}}) \triangleq \max_j \left| C_{\mathrm{p}}(x^{\mathrm{trg}})_{[j]} - C_{\mathrm{p}}(x_n)_{[j]} \right| \qquad (12)$$

$\square_{[i]}$ はベクトル $\square$ の i 番目の要素を表す．

3.4 参照軌道の短縮

次に，参照軌道を $t_a \in [0, T_{\mathrm{n}}(x_n, u_n)]$ に短縮する．ここで $T_{\mathrm{n}}(x_n, u_n) \leqslant T_{\mathrm{f}}$ は，行動開始時の状態において，隣接する基底関数に遷移する程度の時間幅として計算される．$T_{\mathrm{n}}(x_n, u_n)$ を計算するために，まず開始状態 x_n において，隣接する基底関数間の距離 $D_{\mathrm{n}}(x_n)$ を推定する．そして，$D_{\mathrm{n}}(x_n)$ と，現在の状態と目標状態の距離の比率から $T_{\mathrm{n}}(x_n, u_n)$ を決定する．

隣接する基底関数間の距離 $D_{\mathrm{n}}(x_n)$ は，x_n がある基底関数 $k \in \mathcal{K}$ の中心 μ_k の場合は k の最近傍の基底関数 $k_{\mathrm{n}}(k)$ までの距離 $d_{\mathrm{n}}(k)$ で定義する．それ以外の x_n については基底関数の出力 $\phi(x_n)$ を用いて補間する．$d_{\mathrm{n}}(k)$ は，以下で定義される．

$$k_{\mathrm{n}}(k) = \underset{k' \in \mathcal{K}, k' \neq k}{\arg\min} \|\mu_{k'} - \mu_k\| \qquad (13)$$

$$d_{\mathrm{n}}(k) = \max\left(\|\mu_{k_{\mathrm{n}}(k)} - \mu_k\|, \mathrm{d}_{\min k}\right) \qquad (14)$$

ただし $\mathrm{d}_{\min k}$ は基底関数 k に関連付けられた正の定数で，$\|\mu_{k_{\mathrm{n}}(k)} - \mu_k\|$ が非常に小さい場合に $d_{\mathrm{n}}(k)$ を調整する働きを持つ．NGnet の場合，共分散行列 Σ_k の最大固有値 λ_k を用いて $\mathrm{d}_{\min k} = \sqrt{\lambda_k}$ と定義した．

次に $d_{\mathrm{n}}(k)$ を $x_n = \mu_k$ である場合の $D_{\mathrm{n}}(x_n)$ とみなし，基底関数の出力 $\phi(x_n)$（式（3））を用いて補間したものを $D_{\mathrm{n}}(x_n)$ の定義とする．

$$d_{\mathrm{n}} \triangleq (d_{\mathrm{n}}(1),\, d_{\mathrm{n}}(2),\, \ldots,\, d_{\mathrm{n}}(|\mathcal{K}|))^{\top} \tag{15}$$

$$D_{\mathrm{n}}(x_n) = d_{\mathrm{n}}^{\top}\phi(x_n) \tag{16}$$

最後に，T_{n} を次式で計算する．

$$T_{\mathrm{n}}(x_n, u_n) = \frac{\min(D_{\mathrm{n}}(x_n),\, \|x^{\mathrm{trg}} - x_n\|)}{\|x^{\mathrm{trg}} - x_n\|} T_{\mathrm{f}} \tag{17}$$

3.5 短縮された参照軌道の追従

短縮された参照軌道 $q^{\mathrm{d}}(t = t_n + t_a),\ t_a \in [0, T_{\mathrm{n}}(x_n, u_n))$ は，与えられた制御器 $\tilde{u}(t) = Ctrl(x(t), q^{\mathrm{d}}(t + \delta \mathrm{t}))$，$t \in [t_n, t_n + T_{\mathrm{n}}(x_n, u_n))$ で追従される．この制御器が単純な PD 制御器の場合，

$$\tilde{u}(t) = \mathrm{K_p}\{q^{\mathrm{d}}(t + \delta \mathrm{t}) - C_{\mathrm{p}}(x(t))\} - \mathrm{K_d} C_{\mathrm{d}}(x(t)) \tag{18}$$

と書き下せる．ただし $\mathrm{K_p}$ と $\mathrm{K_d}$ は PD 制御器のゲインパラメタを表す．

4. DCOB: BFTrans を離散化した行動集合

本章では，連続行動空間 BFTrans（$\mathcal{U}_{\mathrm{BFTrans}} = \mathbb{R} \times \mathcal{X}$）を離散化することによって，離散行動集合 DCOB（$\mathcal{A}_{\mathrm{DCOB}}$）を定義する．具体的には，行動 $u = (g, x^{\mathrm{trg}}) \in \mathcal{U}_{\mathrm{BFTrans}}$ について，基底関数集合 $\mathcal{K}$ を用いて目標状態 x^{trg} を $\{\mu_k \mid k \in \mathcal{K}\}$ で離散化し，到達時間幅係数 g を少数の実数集合 $\mathcal{I}$ で離散化する．この離散行動集合 $\mathcal{A}_{\mathrm{DCOB}} \triangleq \mathcal{I} \times \mathcal{K}$ を DCOB と呼ぶ．DCOB は “Directed to the Center Of a Basis function” を表し，基底関数の中心に向かうように動作が生成されることを意味する．時間ステップ n，開始時刻 t_n で選択した DCOB の一つの行動 $a_n \in \mathcal{A}_{\mathrm{DCOB}}$ は，以下のように実行される．

Algorithm 2: Executing an action in the DCOB

Input : 開始状態 $x_n = x(t_n)$,
行動 $a_n = (g, k) \in \mathcal{I} \times \mathcal{K} = \mathcal{A}_{\mathrm{DCOB}}$

1: $u_n \leftarrow (g, \mu_k)$
2: u_n を BFTrans の要素として実行（Algorithm 1）

行動集合 $\mathcal{A}_{\mathrm{DCOB}}$ の要素数は $|\mathcal{I}||\mathcal{K}|$ と表される．$|\mathcal{I}|$ は小さな整数であるため，DCOB の行動数は基底関数の数倍程度となる．よって，基底関数集合の要素数が削減されている場合，DCOB の要素数も低く抑えられる．

5. WF-DCOB: 基底関数によるパラメタの制約と初期化

2.3 節で述べた wire-fitting を用いれば，連続行動空間 BFTrans を直接学習することができる．しかしながら，wire-fitting を用いた学習は概して不安定になる．そこで，まず wire-fitting の問題点を述べ，次にこれを解消するために wire-fitting のパラメタの制約を基底関数から生成する方法，およびパラメタを初期化する方法を提案する．

5.1 Wire-fitting を用いた学習の問題点

Wire-fitting では，制御ワイヤ $i \in \mathcal{W}$ を構成する $q_i(x)$ や $u_i(x)$ のパラメタは学習中に変化する．大雑把にいうと $q_i(x)$ は状態 x で行動 $u_i(x)$ を選択したときに期待できる行動価値に関係した値を学習しているため，$u_i(x)$ の変化は，$q_i(x)$ の状態空間全体における値を変化させ得る．このためパラメタの収束時間に悪影響を与える．

さらに，u_i のパラメタを適切に初期化することは難しい．一般的には乱数を使って行われるが，後述の実験結果からも分かるように，パフォーマンスの低い局所解に収束してしまう．

5.2 Wire-fitting のパラメタ制約

基本的なアイディアは，BFTrans が作る連続行動空間 $\mathcal{U}_{\mathrm{BFTrans}}$ を「小さな領域」に分割し，wire-fitting の制御ワイヤと「小さな領域」を一対一に対応させ，制御ワイヤの行動に関係する関数近似器 $u_i(x)$ の出力がその「小さな領域」から出ないように制約する，というものである．これによって $u_i(x)$ が大幅に変化することがなくなるため，前節で述べたような不安定さは解消される．

BFTrans の行動は $u = (g, x^{\mathrm{trg}})$ で表されるため，目標状態 $x^{\mathrm{trg}} \in \mathcal{X}$ と，到達時間幅係数 $g \in \mathbb{R}$ に分けて考える．これらは独立に分割される．

状態空間 $\mathcal{X} \ni x^{\mathrm{trg}}$ の分割は，DCOB と同様に基底関数集合 $\mathcal{K}$ が状態空間を分割していることを利用し，各基底関数の中心から最近傍の基底関数までを一つの「小さな領域」とする．すなわち，ある基底関数 k に対応する領域は

$$\|x - \mu_k\| \leqslant d_{\mathrm{n}}(k) \tag{19}$$

を満たすすべての $x \in \mathcal{X}$ として定義される．$d_{\mathrm{n}}(k)$ は最近傍の基底関数までの距離を表し，式（14）で定義される．

到達時間幅係数 g は一次元の正の実数なので，区間 $(\mathrm{g^s}, \mathrm{g^e})$ をいくつか定義し，「小さな領域」とする．$\mathrm{g^s}, \mathrm{g^e}$ は区間の始まりと終りを表す事前に与えられた定数で，$0 < \mathrm{g^s} \leqslant \mathrm{g^e}$ を満たす．この区間の集合を

$$\mathcal{I}_{\mathcal{R}} \triangleq \{(\mathrm{g}_i^{\mathrm{s}}, \mathrm{g}_i^{\mathrm{e}}) \mid 0 < \mathrm{g}_i^{\mathrm{s}} \leqslant \mathrm{g}_i^{\mathrm{e}}, i = 1, 2, \ldots\} \tag{20}$$

とおく．$\mathrm{g}_i^{\mathrm{s}} = \mathrm{g}_{i-1}^{\mathrm{e}}$ となるように定めておけば，到達時間幅係数は最小値 $\mathrm{g}_1^{\mathrm{s}}$ から最大値 $\mathrm{g}_{|\mathcal{I}_{\mathcal{R}}|}^{\mathrm{e}}$ までをすき間なく埋めることができる．

以上の方法で行動空間を「小さな領域」に分けると，行動空間 $\mathcal{U}_{\mathrm{BFTrans}} = \mathbb{R} \times \mathcal{X}$ は $|\mathcal{I}_{\mathcal{R}}||\mathcal{K}|$ 個の領域に分割され，このそれぞれに制御ワイヤを対応させる（$|\mathcal{W}| = |\mathcal{I}_{\mathcal{R}}||\mathcal{K}|$）．このときある制御ワイヤ $i \in \mathcal{W}$ には，到達時間幅係数の区間 $(\mathrm{g}_i^{\mathrm{s}}, \mathrm{g}_i^{\mathrm{e}})$ と基底関数 $k_i \in \mathcal{K}$ が対応づけられており，$u_i(x) = U_i \triangleq (g_i, x_i^{\mathrm{trg}})$ のようにパラメタを書くと，これらを対応する「小さな領域」の中に制約する方法は以下のように書き下せる．

$$\begin{aligned}
&\texttt{if } g_i < \mathrm{g}_i^{\mathrm{s}} \texttt{ then } g_i \leftarrow \mathrm{g}_i^{\mathrm{s}} \\
&\texttt{if } g_i > \mathrm{g}_i^{\mathrm{e}} \texttt{ then } g_i \leftarrow \mathrm{g}_i^{\mathrm{e}} \\
&\texttt{if } \|x_i^{\mathrm{trg}} - \mu_{k_i}\| > d_{\mathrm{n}}(k_i) \texttt{ then} \\
&\quad x_i^{\mathrm{trg}} \leftarrow \mu_{k_i} + d_{\mathrm{n}}(k_i)\frac{(x_i^{\mathrm{trg}} - \mu_{k_i})}{\|x_i^{\mathrm{trg}} - \mu_{k_i}\|}
\end{aligned} \tag{21}$$

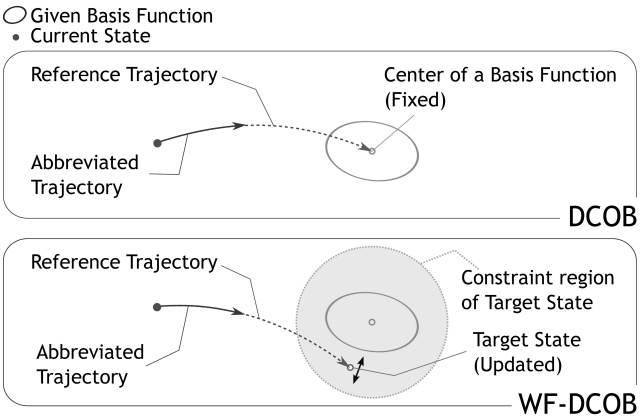


Fig. 3 Illustration of the comparison of the DCOB (top) and the WF-DCOB (bottom). In both methods, the trajectory is calculated in the same manner as the BFTrans. The difference is that in the DCOB, the target state is the fixed center of a selected basis function, while in the WF-DCOB, the target state can change but is constrained around a corresponding basis function

この制約を，Q(λ)-learning（式（1））などによってパラメタを更新したあと，すべての制御ワイヤ $i \in \mathcal{W}$ について適用する．

この手法は，DCOB を wire-fitting を用いて連続行動が学習できるように拡張したものとも見られるので，以下では WF-DCOB と呼ぶ．**Fig. 3** に両者の比較を示す．このように，WF-DCOB では DCOB に比べて広い範囲で行動空間を表現できるため，理論上，より高いパフォーマンスを得る能力を持つ．

5.3 Wire-fitting のパラメタ初期化

Wire-fitting のパラメタ $U_i = (g_i, x_i^{\mathrm{trg}})$ の初期化は，分割された「小さな領域」の中心を表すように行う．このようにすればパラメタの初期値が制約を必ず満たし，また，許容される更新の大きさが最大となるからである．すなわち，

$$g_i \leftarrow \frac{\mathrm{g}_i^{\mathrm{s}} + \mathrm{g}_i^{\mathrm{e}}}{2} \tag{22a}$$

$$x_i^{\mathrm{trg}} \leftarrow \mu_{k_i} \tag{22b}$$

によってパラメタ U_i を初期化する．

6. ロボットのダイナミクスに基づく基底関数の配置

状態空間が高次元の場合，グリッド上に基底関数を配置するような方法だと，膨大な数の基底関数が必要となる．そこで，ロボットのダイナミクスを推定できるように基底関数も含めて関数近似器を最適化し，このとき得られた基底関数を強化学習の行動価値関数の近似にも使う手法 [17] を採用する．具体的には，まずロボットをランダムに動作させてサンプル x（状態），u（制御入力），$\dot{x}$（x で u を入力したときの状態変化）を生成する．これらのデータを用いて，関数近似器

$$\dot{x} = \sum_{k \in \mathcal{K}} \left(A_k \begin{pmatrix} x \\ u \end{pmatrix} + b_k \right) \phi_k(x; \mu_k, \Sigma_k) \tag{23}$$

によってダイナミクスを推定できるように，パラメタ A_k, b_k, μ_k, Σ_k を EM アルゴリズム [20] で最適化する．ただし $\phi_k(x)$ は式（4）で定義された NGnet の基底関数である．このとき得られた基底関数 $\{\mu_k, \Sigma_k \mid k \in \mathcal{K}\}$ を強化学習の行動価値関数の近似や行動集合の生成に用いる．

この方法によって，ロボットのダイナミクスの非線形性が強い状態領域にはより多くの基底関数が配置されるようになる．一方，本研究では近傍の状態への遷移に単純な制御器（PD 制御器など）を利用しているが，ダイナミクスの非線形性が強い状態ではこの制御器だけでは不十分である．このような状態では緻密な方策が要求され，方策を表現する行動価値関数に高い分解能が要求される．よって，ロボットのダイナミクスの近似によって基底関数を配置すると，それらが行動価値関数の近似に必要な基底関数集合に近くなると考えられ，基底関数の数を適切に削減できると期待できる．当然ながら，方策の細かさは報酬関数にも依存するため，最適な基底関数の配置が得られるわけではないが，基底関数の数を減らす上で実用的な手段になり得ると考えられる．類似の方法は，上述の手法がもとにした [15] [16] 以外にも，文献 [21] などで見られる．

7. 実験：ヒューマノイドの全身運動学習

本章では，高次元の状態･行動空間を持つヒューマノイドロボットの運動学習タスクで，提案手法 DCOB および WF-DCOB を，既存の離散行動集合（グリッド行動集合），目標関節角を行動として直接学習する wire-fitting（以下「単純 wire-fitting」），および山口らの手法（以下「OLD」）[17] と比較する．タスクは匍匐(ほふく)動作の獲得とする．以下の実験は動力学シミュレータ ODE[†] 上で行われる．

強化学習手法としては，Peng らの Q(λ)-learning（式（1））を用いる．行動価値関数のパラメタは，タスクに関する事前知識を与えないように，ゼロまたは乱数で初期化する（7.3 節）．割引率は $\gamma = 0.9$, $\lambda = 0.9$ とし，ステップサイズパラメタは $\alpha = \alpha_0 \exp(-\delta_\alpha N_{\mathrm{eps}})$ で減衰させる．探索手法としては，DCOB，グリッド行動集合についてはボルツマン選択，WF-DCOB，単純 wire-fitting については付録 A.2 のボルツマン選択を拡張した方法を用いる．いずれについても，温度パラメタを $\tau = \tau_0 \exp(-\delta_\tau N_{\mathrm{eps}})$ で減衰させる．N_{eps} はエピソード数を表す．以下の実験では $\alpha_0 = 0.3$, $\delta_\alpha = 0.002$, $\tau_0 = 5$, $\delta_\tau = 0.004$ とし，すべての実験条件において共通とした．なお，Q(λ)-learning（式（1））を実装する際，エリジビリティトレースを安定化するために replacing trace [22] を適用する．ただし，これが有効なのは行動価値関数が線形かつ局所モデルの場合に限られる [18] ため，DCOB，およびグリッド行動集合にのみ用いた．

7.1 ロボットシステム

以下の実験では，自由度を 5 に制約したヒューマノイドロボット（**Fig. 4**）を用いる．このときロボットの状態空間および制御入力空間を，以下のようにそれぞれ 21 次元，五次元のベクトルとして定義する．

$$x = (c_{0z}, q_w, q_x, q_y, q_z, q_1, q_2, q_3, q_4, q_5, \dot{c}_{0x}, \dot{c}_{0y}, \dot{c}_{0z}, \omega_x, \omega_y, \omega_z, \dot{q}_1, \dot{q}_2, \dot{q}_3, \dot{q}_4, \dot{q}_5)^\top \tag{24}$$

$$\tilde{u} = (\tilde{u}_1, \tilde{u}_2, \tilde{u}_3, \tilde{u}_4, \tilde{u}_5)^\top \tag{25}$$

[†]Open Dynamics Engine: http://www.ode.org/

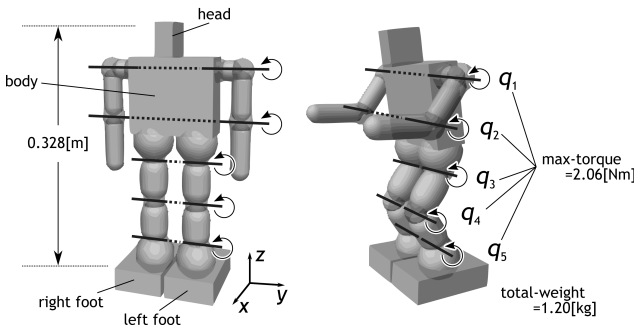


Fig. 4 Humanoid robot whose DoF is constrained to five

ここで (c_{0x}, c_{0y}, c_{0z}) はボディリンクの重心位置，(q_w, q_x, q_y, q_z) はボディリンクの姿勢をクォータニオン表現したもの，$(\omega_x, \omega_y, \omega_z)$ はボディリンクの回転速度，q_j $(j = 1, .., 5)$ は関節角，$\tilde{u}_j$ $(j = 1, .., 5)$ は関節トルクをそれぞれ表す．式（24）の状態 x で c_{0x} や c_{0y} が使われていないのは，以下のタスク（匍匐）ではこれらの観測値がなくともマルコフ性を満たすため，行動獲得可能と考えられるからである．関節トルクは $|\tilde{u}_j| \leqslant \tilde{u}_{\max} = 2.06[\mathrm{Nm}]$ $(j = 1, .., 5)$ に制限されている．動力学シミュレータ（ODE）の時間ステップは $\delta \mathrm{t} = 0.2\,[\mathrm{ms}]$ とした．

7.2 基底関数の配置

基底関数の配置が提案手法に及ぼす影響を明確にするため，以下の実験では，次の 2 種類の基底関数集合を用いる．

グリッド配置： 状態（式（24））空間を，関節角（$q_1, \ldots, q_5$）のみ 3 分割したグリッド（1×1×1×1×1×3×3×3×3×3×1×1×1×1×1×1×1×1×1×1）で区切り，計 243 個の基底関数を配置する．

ダイナミクスに基づく配置： 6 章で述べた方法で基底関数を配置する．この配置は，強化学習のプロセスの前に行われる．以下の実験では，200 個の基底関数から開始してロボットのダイナミクスを学習させ（基底関数の追加・削除処理を含む EM アルゴリズム [20] を用いた），得られた 202 個の基底関数を用いる．200 という値は，$\{50, 100, 200\}$ で予備実験を行った結果から採用した．

これらの基底関数集合は，以下のすべての実験条件において，共通に用いられる．つまり，基底関数集合は DCOB や WF-DCOB の生成のみでなく，グリッド行動集合などを用いて学習する際の行動価値関数を近似するためにも用いられる．

7.3 行動空間・関数近似器の設定

DCOB： 2.2 節の線形関数近似器を，初期値 $\theta_a = \mathbf{0}, a \in \mathcal{A}_{\mathrm{DCOB}}$ で用いる．DCOB を適用するために，C_{p}, C_{d}, *Ctrl*, および $\mathcal{I}$ を以下のように定める．

$$C_{\mathrm{p}}(x) = (q_1, q_2, q_3, q_4, q_5)^\top \tag{26}$$

$$C_{\mathrm{d}}(x) = (\dot{q_1}, \dot{q_2}, \dot{q_3}, \dot{q_4}, \dot{q_5})^\top \tag{27}$$

$$\begin{aligned} &Ctrl(x(t), q^{\mathrm{d}}(t + \delta \mathrm{t})) \\ &\quad = \mathrm{K_p}\{q^{\mathrm{d}}(t + \delta \mathrm{t}) - C_{\mathrm{p}}(x(t))\} - \mathrm{K_d} C_{\mathrm{d}}(x(t)) \end{aligned} \tag{28}$$

$$\mathcal{I} = \{0.075,\ 0.1,\ 0.2\} \tag{29}$$

ただし $\mathrm{K_p} = 5.0\,[\mathrm{Nm/rad}]$, $\mathrm{K_d} = 1.6\,[\mathrm{Nms/rad}]$ とした．このとき行動集合の要素数は，グリッド配置の基底関数集合を用いた場合 $|\mathcal{A}_{\mathrm{DCOB}}| = |\mathcal{I}||\mathcal{K}| = 729$，ダイナミクスに基づく配置の場合 $|\mathcal{A}_{\mathrm{DCOB}}| = 606$ となる．なお，グリッド配置の場合は基底関数集合が $\mathcal{Q}$ 空間のみに分布するため，$\mathcal{Q}$ に変換した DCOB を用いる（付録 B 参照）．

WF-DCOB： 制御ワイヤ $i \in \mathcal{W}$ のパラメタは，U_i を式（22）で初期化し，$\theta_i = \mathbf{0}$ とする．C_{p}, C_{d}，および *Ctrl* については，上述の DCOB で用いたものと同様に定める．さらに，$\mathcal{I}_{\mathcal{R}}$ は以下のように定める．

$$\mathcal{I}_{\mathcal{R}} = \{(0.05, 0.1), (0.1, 0.2), (0.2, 0.3)\} \tag{30}$$

このとき制御ワイヤの数は，グリッド配置の基底関数集合を用いた場合 $|\mathcal{W}| = |\mathcal{I}_{\mathcal{R}}||\mathcal{K}| = 729$，ダイナミクスに基づく配置の場合 $|\mathcal{W}| = 606$ で DCOB の要素数と等しい．なお，5 章では DCOB との対応を明確にするため $u = (g, x^{\mathrm{trg}}) \in \mathbb{R} \times \mathcal{X}$ を学習するように定式化したが，以下では行動を $u = (g, q^{\mathrm{trg}}) \in \mathbb{R} \times \mathcal{Q}$ として学習する．これは $\mathcal{X}$ 空間より $\mathcal{Q}$ 空間の方が次元数が小さいため，学習に有利だからである（付録 B 参照）．

グリッド行動集合（GRID3, GRID5）： ここでは「グリッド行動集合」$\mathcal{A}_{\mathrm{G}}$ を，目標関節角をグリッドで区切って離散化した行動集合として定義する．

$$\begin{aligned} \mathcal{A}_{\mathrm{G}} = \big\{ &\Delta q \mid \Delta q = (\delta q_1, \delta q_2, \delta q_3, \delta q_4, \delta q_5)^\top, \\ &\delta q_{1,2,3,4,5} \in \{0, \pm\Delta\varphi, \ldots, \pm\tfrac{\mathrm{N_{grid}}-1}{2}\Delta\varphi\} \big\} \end{aligned} \tag{31}$$

ただし $\Delta\varphi = \pi/12$ とした．$\mathrm{N_{grid}}$ はグリッドの分割数を表す．すなわち $\mathcal{A}_{\mathrm{G}}$ の要素数は $|\mathcal{A}_{\mathrm{G}}| = (\mathrm{N_{grid}})^5$ であり，$\mathrm{N_{grid}} \geqslant 7$ 程度でも学習が困難となる．そこで以下の実験では $\mathrm{N_{grid}} = 3, 5$ を用いた（それぞれ GRID3，GRID5 と表記）．ある行動 $\Delta q \in \mathcal{A}_{\mathrm{G}}$ は以下のように実行される．

$$q^{\mathrm{d}}(t) = C_{\mathrm{p}}(x(t)) + \Delta q \tag{32}$$

$$\tilde{u}(t) = \mathrm{K_p}\{q^{\mathrm{d}}(t) - C_{\mathrm{p}}(x(t))\} - \mathrm{K_d} C_{\mathrm{d}}(x(t)) \tag{33}$$

ただし $t = t_n + t_a$, $t_a \in [0, \mathrm{T_G})$ と定義される．$\mathrm{T_G} = 0.1\,[\mathrm{s}]$ は行動が実行される時間を表す定数である．

行動価値関数には，2.2 節の線形関数近似器を，初期値 $\theta_a = \mathbf{0}, a \in \mathcal{A}_{\mathrm{G}}$ で用いる．

単純 wire-fitting（WF50, WF100）： 以下のように，選択された行動 $u(x_n)$ から制御入力 $\tilde{u}$ を計算する．

$$q^{\mathrm{d}}(t) \triangleq C_{\mathrm{p}}(x(t)) + u(x_n) \tag{34}$$

$$\tilde{u}(t) = \mathrm{K_p}\{q^{\mathrm{d}}(t) - C_{\mathrm{p}}(x(t))\} - \mathrm{K_d} C_{\mathrm{d}}(x(t)) \tag{35}$$

ただし $t = t_n + t_a$, $t_a \in [0, \mathrm{T_{WF}})$ と定義される．$\mathrm{T_{WF}} = 0.1\,[\mathrm{s}]$ は行動が実行される時間幅を表す定数である．

Wire-fitting で表現された行動価値関数のパラメタは，U_i を一様乱数で初期化し，すべての $i \in \mathcal{W}$ について $\theta_i = \mathbf{0}$ で初期化する．実験では制御ワイヤの数 $|\mathcal{W}|$ として $\{50, 100\}$ の 2 種類を用いる（それぞれ WF50，WF100 と表記）．

Fig. 5 Sequence of the acquired crawling motion with the DCOB

山口ら [17] の手法（OLD）： 文献 [17] で用いられた手法は，DCOB の実行において「軌道短縮」ステップを削除し，「参照軌道」として五次方程式を用いた場合である．以下の実験では，これらの違い以外は DCOB と同じ設定とする．

7.4 タスク設定

匍匐タスクの目的は x 軸の正の方向にできるだけ移動するというものであり，以下のように報酬関数を設計する．

$$r(t) = r_{\rm mv}(t) - r_{\rm sc}(t) - r_{\rm fd}(t) \tag{36}$$

$$r_{\rm mv}(t) = 50\dot{c}_{0x}(t) \tag{37}$$

$$r_{\rm sc}(t) = 0.1\|u(t)\| \tag{38}$$

$$r_{\rm fd}(t) = \begin{cases} 4\delta(0) & \text{(falling down)} \\ 0 & \text{(otherwise)} \end{cases} \tag{39}$$

ただし $r_{\rm mv}(t)$ は移動に対して与えられる報酬，$r_{\rm sc}(t)$ はステップコスト，$r_{\rm fd}(t)$ は転倒に対して与えられる罰（報酬の逆）を表す．転倒 falling down は，各時間ステップ n で最初にボディ（body）リンクまたはヘッド（head）リンクが接地した瞬間に真となる．$\delta(t)$ はディラックのデルタ関数を表す．これらの報酬は**連続時間** t について定義されており，これは異なる行動集合を公平に比較するためである．このときある行動に対して与えられる報酬は，式（41）によって計算される．

各エピソードは，起立状態（Fig. 4 左）から始まり，$\int_0^t r(t')\mathrm{d}t' \leqslant -40$ もしくは $t > 20$ [s] で終了する．

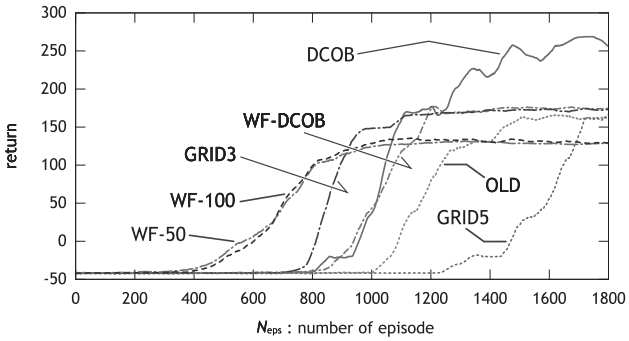
7.5 結果と考察

Fig. 5 は DCOB によって得られた匍匐の様子を表す．**Fig. 6** は匍匐タスクにおける結果の学習曲線を表す．それぞれエピソードに対する収益がプロットしてあり，各曲線は 15 回試行を行った平均を表す．Fig. 6 (a) は基底関数をグリッド配置した NGnet を用いた場合，Fig. 6 (b) は基底関数をロボットのダイナミクスに基づいて配置した場合の結果をそれぞれ表す．これらの NGnet は，すべての手法における行動価値関数の近似，および DCOB・WF-DCOB の生成に共通に用いられている．いずれの曲線についても，学習に失敗した結果（前転してしまう，初期位置から全く移動しない）は除いてある†．強化学習の目的は収益最大化だから，学習曲線は左上を通るほうが性能がよい．学習曲線の立ち上がりの速さを「学習速度」と呼び，学習曲線の収束後の収益を「パフォーマンス」と呼ぶ．

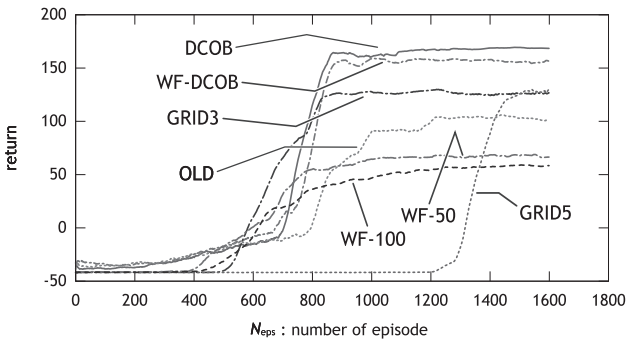
Fig. 6 (a) では，GRID3，WF50，WF100 の学習が DCOB よりも速い．これは，学習すべきパラメタの数が少ないためである．例えば GRID3 と DCOB を比較した場合，いずれも離散行動集合なのでパラメタ数は行動集合の数に比例し，それぞれ $3^5 = 243$，729 である．一方，パフォーマンスについては DCOB が高い値を示している．Fig. 6 (b) についても DCOB が最も高いパフォーマンスを示している．よって，DCOB によって基底関数集合から強化学習に適した（高いパフォーマンスを得られる）行動集合を生成できていると考えられる．さらに，Fig. 6 (b) では DCOB の学習速度が GRID3 などと大差なくなっており，ダイナミクスに基づいて基底関数を配置する方法と DCOB を組み合わせた場合，学習速度・パフォーマンスともに高い値を得られるという本論文の主張を支持する結果となっている．

OLD と DCOB を比較した場合，DCOB のパフォーマンスが向上していることが分かる．これは 3.2 節で述べたように，「軌道短縮」を行ったことによってより細かくロボットの行動を探索できたためだと考えられる．

WF-DCOB のパフォーマンスは，Fig. 6 (b) では DCOB とほぼ同じだが，Fig. 6 (a) では低い結果となっている．これらの違いは，基底関数をグリッド配置した場合のほうがパラメタ数や制御ワイヤの数が多くなり，wire-fitting が持つ学習の不安定さが影響したことによって生じたと考えられる．加えて，前述のように DCOB には replacing trace を適用しているが，これは行動価値関数が非線形の WF-DCOB には適用できない．



(a) BFs are allocated on a grid of the state space



(b) BFs are allocated based on the dynamics of the robot

Fig. 6 Resulting learning curves of the crawling task. Each curve shows the mean of the return over 15 runs per episode. To see the tendency of each curve, a low-pass filter with a time constant of 50 episode is applied. The difference of the two graphs is the way to allocate the basis functions (BFs) of the NGnet

†Fig. 6 (a) では WF-DCOB が 1 回，Fig. 6 (b) では GRID3 が 2 回，WF50 が 3 回，WF100 が 3 回，それぞれ失敗した．

このことも学習を不安定にする一因となっていると考えられる．一方で，WF-DCOB は同じ wire-fitting を用いている WF50 や WF100 よりも高いパフォーマンスが獲得できており，5 章で提案した手法が適切に機能していることが分かる．

8. 議　　論

8.1 BFTrans を用いた場合の強化学習の収束性

連続行動空間 BFTrans を用いた場合の強化学習の収束性を議論する．準備として，**もとのタスク** $x(t)$, $\tilde{u}(t)$, $r(t)$ と BFTrans を行動に用いた場合のタスク（**変換されたタスク**と呼ぶ）x_n, u_n, R_n の関係を以下のように定義する．

$$t_0 = 0, \quad t_n = \sum_{n'=0}^{n-1} T_{\mathrm{n}n'}, \quad x_n = x(t_n) \tag{40}$$

$$R_n = \int_{t_n}^{t_n + T_{\mathrm{n}n}} r(t)\mathrm{d}t \tag{41}$$

ただし $T_{\mathrm{n}n}$ は時間ステップ n において行動 u_n に対して計算された，BFTrans の実行時間 T_{n} を表す．$r(t)$ は状態 $x(t)$ で制御入力 $\tilde{u}(t)$ を加えた場合に発生する報酬を表す．

多くの強化学習の収束証明 [18] [23] と同様に，もとのタスク $x(t)$, $\tilde{u}(t)$, $r(t)$ はマルコフ性を満たすとする．このときもし $\tilde{u}(t)$, $t \in [t_n, t_n + T_{\mathrm{n}n'}]$ が u_n と $x(t)$, $t \in [t_n, t_n + T_{\mathrm{n}n'}]$ によって決定されるとするなら，明らかに変換されたタスク x_n, u_n, R_n はマルコフ性を満たす．参照軌道 $q^{\mathrm{d}}(t)$ と到達時間幅 $T_{\mathrm{n}n'}$ は状態 $x(t_n)$ と行動 u_n のみから計算され，与えられた制御器 $\tilde{u} = \mathit{Ctrl}(x, q^{\mathrm{d}})$ は状態 x と参照軌道のみから計算される．よって，BFTrans によって変換されたタスクもマルコフ性の条件を満たす．このように BFTrans によるタスクの変換は，強化学習手法の収束の保証に影響を及ぼさないと言える．なお，BFTrans を離散化した DCOB についても，同様の収束性の議論が当てはまることに留意されたい†．

8.2 参照軌道の境界条件の妥当性

多くの強化学習の応用では関数近似器に汎化性能が要求されるが††，この汎化性能はマルコフ性に悪影響を与え得る（例えば文献 [24]）．BFTrans においては，参照軌道のパラメタを決定する境界条件（式（10））がこの問題に関係する．ダイナミックな動作を学習する場合，速度成分は大きく変化する．このような場合に，参照軌道が開始状態の速度項 $C_{\mathrm{d}}(x_n)$ に依存するように境界条件を定めると，BFTrans による状態変化が，開始状態の速度成分に大きく依存するようになる．すなわち，推定すべき行動価値関数の真値が，速度成分に対して大きく変化するようになり，要素数を削減した基底関数集合（汎化性能が高い）を用いた場合の価値関数の推定精度が低下する．この問題を避けるため，BFTrans では境界条件に開始状態の速度項 $C_{\mathrm{d}}(x_n)$ を含めないようにしている．実際，ほとんどの予備実験の結果では，式（10）で定めた境界条件は，ほかの境界条件（例えば $\dot{q}^{\mathrm{d}}(t_n + T_{\mathrm{f}}) = \mathbf{0}$ の代わりに $\dot{q}^{\mathrm{d}}(t_n) = C_{\mathrm{d}}(x_n)$ を用いた場合）に比べて収束性および得られた収益を上回った．

8.3 BFTrans の計算コスト

BFTrans の実行において，軌道生成ステップおよび軌道追従ステップではさほど計算コストが要求されない†††．他方，軌道短縮ステップでは d_{n} を計算するために $\mathcal{O}(|\mathcal{K}|^2)$ の計算コストが要求される．しかしながら，d_{n} は基底関数の中心や分散が変化しなければ一定であり，本論文では基底関数は学習中に変化しないと仮定しているから，学習前に計算しておけばよい．このとき，軌道短縮ステップで要求される計算コストは $\mathcal{O}(|\mathcal{K}|)$ に軽減される．毎回の行動選択において，行動価値関数の評価に $\mathcal{O}(|\mathcal{K}|)$ の計算コストが掛かっていることを考えると，BFTrans は計算コストのオーダを増加させない．

8.4 使用できる基底関数

本論文では，基底関数として正規化ガウシアンネットワーク（NGnet）を想定している．しかしながら提案手法（BFTrans, DCOB, WF-DCOB）は，3.1 節で述べた，個々の基底関数が状態空間に中心を持つという条件を満たしさえすれば，ほかの基底関数を用いることも可能である‡．例えば RBF はすべて該当する．あるいは Takahashi ら [25] が用いている，状態空間に中心（representative state vector と呼ばれる）を持つ基底関数を使用することもできる．いずれの場合でも，与えられた基底関数の集合が状態空間に同じように配置されているならば，BFTrans で表現できる動作のパフォーマンスは主に基底関数の中心状態に依存するため，基底関数の種類によって表現できるパフォーマンスに大きな差は現れないと考えられる．本論文の主目的は，ほかの行動空間を用いた場合よりも強化学習の性能を向上させることであり，ある基底関数について性能の向上を実験的に示すことができれば，ほかの基底関数を用いた場合でも，少なくともパフォーマンスについては同様の結果が期待できる．このため，本論文では強化学習で用いられることが多い NGnet のみを使用した．

8.5 適用可能なタスク

提案手法を適用するためには 3.1 節で述べた仮定，すなわち，空間 $\mathcal{Q}$ 上で計算された参照軌道を追従する制御器 *Ctrl*，および状態空間 $\mathcal{X}$ を $\mathcal{Q}$ に写像する関数 C_{p}, C_{d} が必要である．このため $\mathcal{X}$ と $\mathcal{Q}$ の間のキネマティクスが自明でないタスクについては適用が難しい．例えばマニピュレータの制御タスクにおいて，$\mathcal{Q}$ を関節角空間，$\mathcal{X}$ を関節角および角速度空間とするなら，C_{p}, C_{d}, *Ctrl* は容易に決定できる．多脚ロボットの全身運動学習も同様である．一方，$\mathcal{X}$ としてエンドエフェクタの位置・姿勢を用いるタスクの場合，C_{p}, C_{d}, *Ctrl* の構成にはより複雑なキネマティクスが必要となり，要求される事前知識が増加する．ビジュアルサーボなどのタスクでも同様だと考えられる．あるいは，DCOB を全方位移動ロボットのナビゲーションタスクへ適用した場合には良好な結果が得られているが（文献 [26] 参照），非ホロノミックな拘束がある移動ロボットの場合は異なったアプローチが必要である．

8.6 関連研究

8.6.1 Options

Sutton らは options と呼ばれる，強化学習においてプリミ

†WF-DCOB の場合はそもそも非線形の関数近似器を利用しているため，マルコフ性を満たしても強化学習が収束性しない場合がある [18]．

††NGnet の場合，ある程度基底関数の分散を大きくしておき，近接する状態における行動価値の推定に汎化させる．

†††厳密には $\mathcal{O}(\dim(\mathcal{Q}))$．通常 $\dim(\mathcal{Q}) < |\mathcal{K}|$．

‡ただし，式（14）の $\mathrm{d}_{\min k}$ を変更する必要がある．

ティブ行動（あるいはマクロ行動）を一般化した理論を提案した[27]．本論文で提案する DCOB はロボットの制御に特化した options の一種であるとみなすこともできる．options やサブゴールを自動で発見する手法の研究も行われているが[28]〜[30]，最適な options を発見することはいまだ困難な課題である．DCOB は動作のパフォーマンスをさほど損なうことなく，離散行動集合を定義できているため，実用的な解決策の一つだと考えられる．

8.6.2 Parti-game Algorithm

Moore と Atkeson は，連続な状態行動空間における強化学習手法として，parti-game algorithm と呼ばれる手法を提案した[31]．この手法では，近傍状態への制御器を仮定し，状態空間を区分けした離散状態空間上で，隣接する区分への遷移を行動として最適な方策を探索する．この手法の特徴は，状態空間の区分けが学習中に最適化されることである．この点，与えられた基底関数集合（状態空間の区分けに相当）を変更しない DCOB よりも優れている．しかし，タスクがゴール状態によって表され，かつその状態が明示的に与えられる問題に限定されており，本論文で適用した匍匐タスクなどには適用が難しい．DCOB は通常の行動集合として扱うことができ，報酬関数を制限せずに強化学習手法を適用できる点で優れている．

8.6.3 連続行動空間

制御入力空間 $\tilde{\mathcal{U}}$ の代わりに学習しやすい行動空間 $\mathcal{U}$ を定義することは，強化学習のロボティクスにおける応用において，しばしば行われてきた．典型的な方法は，PD 制御器を用意し，目標位置や目標角を強化学習で獲得するというものである[3]．Central pattern generator（CPG）を制御器として用い，そのパラメタを強化学習で学習すると，歩行などのリズム運動を学習する場合に効果的である[8]．本論文で提案する行動集合 BF-Trans は，CPG では学習が困難なエピソード的タスクにも適用可能である（誌面の都合上割愛したが，跳躍に対する DCOB の適用結果[26] を参照されたい）．

Ijspeert・Nakanishi らは，高次元状態空間の運動学習のために非線形の運動プリミティブを提案した[32]．さらに，Peters らは運動プリミティブのパラメタを強化学習によって最適化する方法を開発した[9]．この手法は，学習しやすい連続行動空間を構成し，連続行動空間に対する強化学習を適用している点で，本論文で提案する WF-DCOB と類似している．しかしながら，この手法は模倣学習の枠組で学習パラメタを初期化することを前提としている．このため，タスクの事前知識を与えられない場合，7 章の単純 wire-fitting のように，高次元の問題設定ではパフォーマンスの低い局所解に陥る可能性が高いものと考えられる．本論文で提案する WF-DCOB は事前知識なしでもパラメタの初期化を適切に行うことができる点で優位である．Miyamoto らは経由点表現を用いた強化学習手法を提案した[33]．この手法も行動空間を作る一つの方法とみなせるが，運動プリミティブの場合と同様に，高次元の問題設定ではパラメタの初期化が困難であると考えられる．

9. 結　　論

本論文では，高次元の制御入力空間を強化学習で効率的に扱えるようにするため，離散行動集合 DCOB を提案した．DCOB は，行動価値関数の近似のために与えられた基底関数集合から生成され，高いパフォーマンスが獲得できるように工夫されている．さらに，本論文で用いた「ロボットのダイナミクスに基づいて基底関数を配置する手法」などによって，基底関数集合の要素数が削減されている場合，DCOB の要素数も低く抑えられる．これにより，既存の離散行動集合（グリッド行動集合など）が持つ，要素数が制御入力空間の次元に対して指数関数的に増加するという問題を緩和できた．さらに，より高いパフォーマンスを獲得するため，DCOB が離散化する前の連続行動空間（BFTrans）を wire-fitting [2] で直接学習させた．ただし wire-fitting は潜在的に不安定さを持つため，これを解決するためにパラメタの制約・初期化方法を提案し，WF-DCOB と名付けた．

シミュレーション実験では，強化学習で一般的に用いられる，グリッド上に配置した基底関数を用いた場合と，ダイナミクスに基づいて基底関数を配置した場合のそれぞれについて他手法と比較した．結果，グリッド配置では，グリッド行動集合のような既存の離散行動集合で学習する場合や，単純な wire-fitting で連続行動空間を学習する場合と比較して，DCOB は高いパフォーマンスが得られることを示した．さらに，ダイナミクスに基づく配置では，パフォーマンス，学習速度いずれにおいても高い性能が得られることを示した．一方，WF-DCOB は wire-fitting が持つ学習の不安定さなどが原因で，DCOB よりも高いパフォーマンスを得られなかったものの，既存の wire-fitting と比べて，かなり高いパフォーマンスを獲得できることを示した．

提案手法を適用するためにはいくつかの仮定が必要であり（3.1 節），タスクによっては適用が困難となる（8.5 節）．また，「基底関数集合が事前に定義されていて，その中心が固定されている」という仮定は，収束を保証し安定した学習を行うためであるが，強化学習で基底関数の中心や分散などのパラメタを更新することで高いパフォーマンスを実現している研究もある（例えば文献[3] [5]）．よって，これらの仮定がなくても学習を行えるように手法を改良することは重要である．

謝　辞　本研究の一部は科研費（22·9030）の助成を受けた．

参 考 文 献

[1] J. Peng and R.J. Williams: “Incremental multi-step Q-learning,” International Conference on Machine Learning, pp.226–232, 1994.

[2] L.C. Baird and A.H. Klopf: Reinforcement learning with high-dimensional, continuous actions, Technical Report WL-TR-93-1147, Wright Laboratory, Wright-Patterson Air Force Base, 1993.

[3] J. Morimoto and K. Doya: “Reinforcement learning of dynamic motor sequence: Learning to stand up,” IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS’98), pp.1721–1726, 1998.

[4] H. Kimura, T. Yamashita and S. Kobayashi: “Reinforcement learning of walking behavior for a four-legged robot,” Proceedings of the 40th IEEE Conference on Decision and Control, 2001.

[5] T. Kondo and K. Ito: “A reinforcement learning with evolutionary state recruitment strategy for autonomous mobile robots control,” Robotics and Autonomous Systems, vol.46,

no.2, pp.111–124, 2004.
[6] C. Gaskett, L. Fletcher and A. Zelinsky: "Reinforcement learning for a vision based mobile robot," IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'00), 2000.
[7] J. Zhang and B. Rössler: "Self-valuing learning and generalization with application in visually guided grasping of complex objects," Robotics and Autonomous Systems, vol.47, no.2–3, pp.117–127, 2004.
[8] Y. Nakamura, T. Mori, M. Sato and S. Ishii: "Reinforcement learning for a biped robot based on a CPG-actor-critic method," Neural Networks, vol.20, no.6, pp.723–735, 2007.
[9] J. Peters, S. Vijayakumar and S. Schaal: "Reinforcement learning for humanoid robotics," Humanoids2003, IEEE-RAS International Conference on Humanoid Robots, 2003.
[10] E. Uchibe and K. Doya: "Competitive-cooperative-concurrent reinforcement learning with importance sampling," Proc. of International Conference on Simulation of Adaptive Behavior: From Animals and Animats, pp.287–296, 2004.
[11] Y. Takahashi and M. Asada: "Multi-layered learning systems for vision-based behavior acquisition of a real mobile robot," Proceedings of SICE Annual Conference 2003, pp.2937–2942, 2003.
[12] C.K. Tham and R.W. Prager: "A modular q-learning architecture for manipulator task decomposition," Eleventh International Conference on Machine Learning, pp.309–317, 1994.
[13] F. Kirchner: "Q-learning of complex behaviours on a six-legged walking machine," Robotics and Autonomous Systems, vol.25, no.3–4, pp.253–262, 1998.
[14] R.S. Sutton, C. Szepesvári, A. Geramifard and M. Bowling: "Dyna-style planning with linear function approximation and prioritized sweeping," Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence, pp.528–536, 2008.
[15] D.M. Wolpert and M. Kawato: "Multiple paired forward and inverse models for motor control," Neural Networks, vol.11, no.7–8, pp.1317–1329, 1998.
[16] K. Doya, K. Samejima, K. Katagiri and M. Kawato: "Multiple model-based reinforcement learning," Neural Computation, vol.14, no.6, pp.1347–1369, 2002.
[17] 山口，杉本，川人："回避行動の再利用メカニズムを備えた強化学習手法と多関節ロボットの全身運動学習への応用"，日本ロボット学会誌，vol.27, no.2, pp.209–220, 2009.
[18] J.N. Tsitsiklis and B.V. Roy: "An analysis of temporal-difference learning with function approximation," IEEE Transactions on Automatic Control, vol.42, no.5, pp.674–690, 1997.
[19] 内部，銅谷："複数報酬のもとでの階層強化学習"，日本ロボット学会誌，vol.22, no.1, pp.120–129, 2004.
[20] M. Sato and S. Ishii: "On-line EM algorithm for the normalized gaussian network," Neural Computation, vol.12, no.2, pp.407–432, 2000.
[21] 高橋，浅田："実ロボットによる行動学習のための状態空間の漸次的構成"，日本ロボット学会誌，vol.17, no.1, pp.118–124, 1999.
[22] S.P. Singh and R.S. Sutton: "Reinforcement learning with replacing eligibility traces," Machine Learning, vol.22, no.1–3, pp.123–158, 1996.
[23] J.N. Tsitsiklis and B.V. Roy: "Feature-based methods for large scale dynamic programming," Machine Learning, vol.22, pp.59–94, 1996.
[24] H. Kimura, K. Miyazaki and S. Kobayashi: "Reinforcement learning in pomdps with function approximation," ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning, pp.152–160, 1997.
[25] Y. Takahashi, M. Takeda and M. Asada: "Continuous valued q-learning for vision-guided behavior acquisition," IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'99), 1999.
[26] A. Yamaguchi, J. Takamatsu and T. Ogasawara: "Constructing action set from basis functions for reinforcement learning of robot control," IEEE Internactional Conference in Robotics and Automation (ICRA'09), pp.2525–2532, 2009.
[27] R.S. Sutton, D. Precup and S. Singh: "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," Artificial Intelligence, vol.112, pp.181–211, 1999.
[28] A. Mcgovern and A.G. Barto: "Automatic discovery of subgoals in reinforcement learning using diverse density," Proceedings of the eighteenth international conference on machine learning, pp.361–368, 2001.
[29] I. Menache, S. Mannor and N. Shimkin: "Q-cut - dynamic discovery of sub-goals in reinforcement learning," ECML '02: Proceedings of the 13th European Conference on Machine Learning, pp.295–306, 2002.
[30] M. Stolle: Automated discovery of options in reinforcement learning, Master's thesis, McGill University, 2004.
[31] A.W. Moore and C.G. Atkeson: "The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces," Mach. Learn., vol.21, no.3, pp.199–233, 1995.
[32] A. Ijspeert, J. Nakanishi and S. Schaal: "Learning attractor landscapes for learning motor primitives," Advances in Neural Information Processing Systems 15 (NIPS2002), pp.1547–1554, 2002.
[33] H. Miyamoto, J. Morimoto, K. Doya and M. Kawato: "Reinforcement learning with via-point representation," Neural Networks, vol.17, no.3, pp.299–305, 2004.
[34] R. Sutton and A. Barto: Reinforcement Learning: An Introduction. MIT Press, 1998.

付録 A. 実装上の工夫

以下では wire-fitting および WF-DCOB を実装する際の工夫点について述べる．

付録 A.1 勾配のノルムの制約

Wire-fitting は非線形の関数近似器であるため，Q(λ)-learning が発散することがある．これを防ぐために学習のステップサイズパラメタ α を小さく取っておくのも一つの方法だが，強化学習が完了するまでに掛かる時間が長くなってしまう．そこで，本論文では行動価値関数の勾配のノルム $\|\nabla_\theta Q\|$ に制約を設ける．すなわち，以下のように，ある小さな定数 $\mathrm{C_{maxgrad}}$ よりもノルムが小さくなるようにする（実験では $\mathrm{C_{maxgrad}} = 1$ を選んだ）．

$$\texttt{if } \|\nabla_\theta Q_n(x_n, u_n)\| > \mathrm{C_{maxgrad}} \texttt{ then}$$
$$\nabla_\theta Q_n(x_n, u_n) \leftarrow \mathrm{C_{maxgrad}} \frac{\nabla_\theta Q_n(x_n, u_n)}{\|\nabla_\theta Q_n(x_n, u_n)\|} \tag{A.1}$$

付録 A.2 行動選択手法

どのように行動選択するかは強化学習の未解決問題の一つであり，いわゆる「**探索**と**知識利用**の間のトレードオフ」を扱う必要がある [34]．離散行動集合で学習する場合は，ϵ-グリーディやボルツマン選択が一般的である [34]．連続行動空間で学習する場合は，ガウス分布で方策を表現する方法がよく用いられるが（例えば文献 [4]），ガウス分布の中心周辺しか探索できないため，高次元の行動空間では探索と知識利用の制御が極端になりがちである．本論文で用いる wire-fitting の場合制御ワイヤが状態行動空間に広がっているため，制御ワイヤをボルツマン選択で選択し，行動とする．すなわち $u_i(x)$ $(i \in \mathcal{W})$ を確率

$$\pi(i|x) = \frac{\exp\left(\frac{1}{\tau}q_i(x)\right)}{\sum_{i' \in \mathcal{W}} \exp\left(\frac{1}{\tau}q_{i'}(x)\right)} \qquad \text{(A.2)}$$

で選択し，選ばれた $u_i(x)$ を行動として出力する．τ は温度パラメタである．これにより行動空間を広く探索でき，かつ τ をエピソードに対して減衰するように調節すれば，学習が進むとグリーディ方策に収束させられる．本論文では $\tau = \tau_0 \exp(-\delta_\tau N_{\text{eps}})$ によって τ を減衰させている（7 章参照）．

付録 B. $\mathcal{Q}$ 空間上の点を目標状態とする BFTrans, DCOB，WF-DCOB

連続行動空間 BFTrans の行動は $u = (g, x^{\text{trg}}) \in \mathbb{R} \times \mathcal{X} = \mathcal{U}_{\text{BFTrans}}$ と表されるが，「軌道生成」ステップをはじめとするほとんどの計算で x^{trg} は $C_{\text{p}}(x^{\text{trg}})$ によって $\mathcal{Q}$ 空間に変換される．このため BFTrans の行動を $u = (g, q^{\text{trg}}) \in \mathbb{R} \times \mathcal{Q} \triangleq \mathcal{U}^{\mathcal{Q}}_{\text{BFTrans}}$ のように指定しても，ほぼ同様の結果が得られる．以下では $u = (g, q^{\text{trg}})$ で行動を指定する場合の BFTrans，DCOB，および WF-DCOB の変更点を示す．

付録 B.1 BFTrans

到達時間幅の計算： $u = (g, q^{\text{trg}})$ に対して，到達時間幅 T_{f} は，以下で計算される．

$$T_{\text{f}} = g\delta C_{\text{p}}(x_n, q^{\text{trg}}) \qquad \text{(B.3)}$$

$$\delta C_{\text{p}}(x_n, q^{\text{trg}}) \triangleq \max_j \left| q^{\text{trg}}_{[j]} - C_{\text{p}}(x_n)_{[j]} \right| \qquad \text{(B.4)}$$

参照軌道の生成： 境界条件の式（10）において，$C_{\text{p}}(x^{\text{trg}})$ を q^{trg} で置き換え，$C_{\text{d}}(x^{\text{trg}})$ を $\mathbf{0}$ で置き換える．

参照軌道の短縮： $T_{\text{n}}(x_n, u_n)$ の計算（式（17））の x^{trg} を q^{trg} で置き換え，x_n を $C_{\text{p}}(x_n)$ で置き換える．これと整合性を取るため，状態空間 $\mathcal{X}$ における距離として計算されている $D_{\text{n}}(x_n)$ を，$\mathcal{Q}$ 空間における距離 $D^{\mathcal{Q}}_{\text{n}}(x_n)$ として計算しなおす．基本的には，単純に μ_k を $C_{\text{p}}(\mu_k)$ に置き換えるだけですむ．

$$k^{\mathcal{Q}}_{\text{n}}(k) = \underset{k' \in \mathcal{K}, k' \neq k}{\arg\min} \|C_{\text{p}}(\mu_{k'}) - C_{\text{p}}(\mu_k)\| \qquad \text{(B.5)}$$

$$d^{\mathcal{Q}}_{\text{n}}(k) = \max\left(\|C_{\text{p}}(\mu_{k^{\mathcal{Q}}_{\text{n}}(k)}) - C_{\text{p}}(\mu_k)\|,\, \mathrm{d}^{\mathcal{Q}}_{\min k}\right) \qquad \text{(B.6)}$$

$$D^{\mathcal{Q}}_{\text{n}}(x_n) = d^{\mathcal{Q}\top}_{\text{n}} \phi(x_n) \qquad \text{(B.7)}$$

ここで $\mathrm{d}^{\mathcal{Q}}_{\min k}$ は基底関数 k に関連付けられた正の定数で，$\mathrm{d}_{\min k}$ と同じ役割を持つ．NGnet の場合，$\mathcal{Q}$ 空間に変換した共分散行列の最大固有値 $\lambda^{\mathcal{Q}}_k$ を用いて，$\mathrm{d}^{\mathcal{Q}}_{\min k} = \sqrt{\lambda^{\mathcal{Q}}_k}$ と定義した．なお，共分散行列の変換は，簡単のため $C_{\text{p}}(x)$ を線形 $C_{\text{p}}(x) = \hat{\text{C}}_{\text{p}} x$ とすると（$\hat{\text{C}}_{\text{p}}$ は定数行列），$\Sigma^{\mathcal{Q}}_k = \hat{\text{C}}_{\text{p}} \Sigma_k \hat{\text{C}}^{\top}_{\text{p}}$ によって実現できる．7 章の実験では $C_{\text{p}}(x)$ は線形であり，$\hat{\text{C}}_{\text{p}}$ は式（26）から容易に求まる．

付録 B.2 DCOB

行動集合 $\mathcal{A}_{\text{DCOB}} = \mathcal{I} \times \mathcal{K}$ の定義は変わらない．行動 $a = (g, k) \in \mathcal{A}_{\text{DCOB}}$ に対して，$u = (g, C_{\text{p}}(\mu_k))$ で前節の BFTrans を実行する．

付録 B.3 WF-DCOB

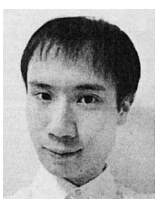
行動が $u = (g, q^{\text{trg}})$ なので，これに合わせて制御ワイヤ $i \in \mathcal{W}$ のパラメタを $u_i(x) = U_i \triangleq (g_i, q^{\text{trg}}_i)$ のように定義する．このパラメタの制約は

$$\texttt{if } \|q^{\text{trg}}_i - C_{\text{p}}(\mu_{k_i})\| > d^{\mathcal{Q}}_{\text{n}}(k_i) \texttt{ then}$$
$$q^{\text{trg}}_i \leftarrow C_{\text{p}}(\mu_{k_i}) + d^{\mathcal{Q}}_{\text{n}}(k_i) \frac{(q^{\text{trg}}_i - C_{\text{p}}(\mu_{k_i}))}{\|q^{\text{trg}}_i - C_{\text{p}}(\mu_{k_i})\|} \qquad \text{(B.8)}$$

によって行われ，初期化は，

$$q^{\text{trg}}_i \leftarrow C_{\text{p}}(\mu_{k_i}) \qquad \text{(B.9)}$$

で行われる．g_i の制約および初期化は 5 章と同様である．



山口明彦（Akihiko Yamaguchi）

2006 年京都大学工学部電気電子工学科卒業．2008 年奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．同年同博士後期課程入学．2010 年日本学術振興会特別研究員．知能ロボットの運動学習，行動獲得の研究に従事．電子情報通信学会会員．（日本ロボット学会学生会員）

小笠原司（Tsukasa Ogasawara）

1983 年東京大学大学院工学研究科情報工学専門課程博士課程修了．同年通商産業省工業技術院電子技術総合研究所入所．1993〜1994 年ドイツ，カールスルーエ大学客員研究員．1998 年奈良先端科学技術大学院大学情報科学研究科教授となり現在に至る．知能ロボットの研究に従事．工学博士．計測自動制御学会，情報処理学会などの会員．（日本ロボット学会正会員）

高松 淳（Jun Takamatsu）

1999 年東京大学理学部情報科学科卒業．2004 年東京大学大学院情報理工学系研究科 コンピュータ科学専攻博士課程修了．博士（情報理工学）．東京大学生産技術研究所特任助教，同特任講師を経て，2008 年から奈良先端科学技術大学院大学情報科学研究科准教授．知能ロボットの動作獲得，三次元形状モデリング・形状解析，物理ベースビジョンに関する研究に従事．情報処理学会，電子通信学会，IEEE の会員．（日本ロボット学会正会員）