

2次元ボクセルベースソフトロボットの
効果的な最適化に関する研究

2024年 9月

齊藤 拓己

2次元ボクセルベースソフトロボットの
効果的な最適化に関する研究

齊藤 拓己

理工情報生命学術院
システム情報工学研究群
筑波大学

2024年 9月

概要

ボクセルベースソフトロボット (VSRs) は、その柔軟性と適応性により、従来の剛体ロボットでは実現が難しい多様なタスクに対応できることから、広範な応用が期待されている。VSRs は、複数の小さなボクセルから構成され、それぞれのボクセルが異なる物理特性や動作を持つことが可能である。この特性により、VSRs は複雑な地形や環境においても高い運動能力と適応力を発揮し、人間や動物に近い自然な動きを再現することができる。また、自己修復能力や形状変化を通じて、ミッションの要求に応じて最適な形態をとることができるため、災害救助、医療ロボティクス、宇宙探査などの分野で特に開発が期待されている。

VSRs の設計と制御には、多くの課題が存在する。その主な理由は、ボクセルの複雑な物理特性や相互作用、動作の多様性にある。VSRs は柔軟で適応性が高い一方で、その柔軟性が制御の難しさを増し、正確な動作や目標達成のための調整が困難となる。このような複雑性を克服するために、シミュレーションが有効な手段として用いられている。シミュレーション環境では、さまざまな設計パラメータや制御アルゴリズムを迅速かつ効率的にテストすることができ、実世界での実験では時間やコストがかかるプロセスを大幅に短縮できる。

本研究では、2次元シミュレーションを用いて、効果的な VSRs の最適化方法について検討する。具体的には、設計の最適化方法と、制御の最適化方法、および複雑な環境への最適化、という3つの観点で実験を行なった。(1) 設計の最適化の観点では、設計の特徴量に基づく最適化手法を提案した。この手法による実験によって、設計全体から計算されるような特徴ではなく、設計の局所的で部分的な構造的な特徴が、タスクの遂行性に大きく影響していることが明らかになった。重要な特徴が明らかになることで設計開発のプロセスはより効果的になる。(2) 制御の最適化の観点では、強化学習と2つの進化的手法によって制御の最適化実験をおこなった。この結果、使用する手法によって引き出すことができる設計の動作性が異なることが示された。これは、設計の最適化に影響を与えるものであり、目的とする設計に対して、適切な制御最適化手法を選択することの重要性を強調した。(3) 複雑な環境への最適化の観点では、動的に複雑に変化する環境と相互作用しながら設計を最適化する手法を提案した。この手法によって、1つの固定された環境に対する最適化では得られない、適応的な VSRs の設計とその制御へ同時にアプローチすることが可能となった。複雑で不確実な環境においても頑健な適応力をもつ VSRs の開発は、現実社会への応用を見据える上でも重要な意義を持つ。

これらの知見は、災害救助や医療ロボティクス、宇宙探査などの現実社会での応用を視野に入れた VSRs の開発において、設計と制御の両面からアプローチする必要性を強調している。今後の研究では、実環境での実証実験を通じて、シミュレーションで得られた結果を現実の応用に展開し、さらに効果的な VSRs の開発を目指すことが期待される。これにより、VSRs の実用化に向けた新たな道が開かれ、ロボティクス分野における革新が進むことが期待される。

目次

第1章	序論	1
第2章	背景	8
2.1	ボクセルベースソフトロボット (VSRs)	8
2.1.1	VSRs 開発の経緯	8
2.1.2	シミュレーションプラットフォーム Evolution Gym	10
2.2	VSRs 最適化の基本的な方法論	12
2.2.1	設計の最適化 遺伝的アルゴリズム (GA) Compositional Pattern Producing Network (CPPN)	13
2.2.2	制御の最適化 Proximal Policy Optimization (PPO)	15
2.2.3	設計と制御の共同最適化	16
第3章	関連研究	19
3.1	設計の最適化	19
3.2	制御の最適化	21
3.3	複雑な環境に対するロボット最適化	23
第4章	設計特徴量に基づく設計最適化	26
4.1	VSRs 設計問題の定義	27
4.2	提案手法	29
4.2.1	部品組み立てによる設計セットの作成 設計の生成手順 十分な量の設計の生成 設計特徴量の定義	30
4.2.2	ベイズ最適化による設計最適化 代理関数 獲得関数	35
4.3	実験の詳細	39
4.3.1	提案手法におけるベイズ最適化	40

4.3.2	遺伝的アルゴリズム (GA)	40
4.3.3	CPPN-NEAT	41
4.4	結果	41
4.4.1	最適化性能の比較	41
4.4.2	探索の多様性の比較	44
4.4.3	設計特徴量の分析	45
	Random Forest の特徴重要度	45
	Partial Dependence Plot	47
4.5	まとめ	48
第 5 章	制御の最適化アルゴリズムの比較	49
5.1	VSRs 制御の問題定義	50
5.1.1	使用するベンチマークタスク	51
5.1.2	ロボットの観測値	53
5.2	3つの制御ニューラルネットワーク最適化手法	54
5.2.1	Proximal Policy Optimization (PPO)	55
5.2.2	Neuro Evolution of Augumenting Topologies (NEAT)	55
5.2.3	Hypercube-based NEAT (HyperNEAT)	56
5.3	比較実験の詳細	57
5.4	実験結果	59
5.4.1	HyperNEAT と PPO の比較	61
5.4.2	NEAT と HyperNEAT の比較	63
5.5	まとめ	64
第 6 章	複雑化する環境に適応可能な設計	66
6.1	環境の複雑化とロボットの最適化アルゴリズム	67
6.1.1	メインループ	67
6.1.2	制御の学習	69
6.1.3	ロボットの転送	69
6.1.4	新しいロボットの生成	70
6.1.5	新しい環境の生成	71
6.2	実験と結果	74
6.2.1	実験 A：提案手法による動的環境における最適化実験	75
6.2.2	実験 B：固定環境に対する最適化実験	79
6.3	環境とロボットの相互作用の分析	82
6.4	まとめ	83
第 7 章	結論	85
7.1	まとめ	85

7.2 今後の課題	87
謝辞	89
参考文献	90

目次

2.1	VSRs 研究の先駆け. 左: シミュレーション状での VSRs 設計の探索. 右: 現実への VSR の再現と動作テスト. [24] より引用.	9
2.2	Evolution Gym の概要. [5] より引用.	11
2.3	2次元 VSRs の設計問題. グリッド状の各セルにボクセルを配置することで VSR は定義される.	13
2.4	CPPN を用いた VSR の設計. グリッドの各セルの座標を CPPN に入力することでボクセルの状態を決定する.	14
4.1	Evolution Gym 上での VSRs 設計の定義.	28
4.2	提案手法の概要. ランダムな部品の組み立てによる探索対象のロボット設計のセットの事前準備と, ベイズ最適化という 2 段階で構成される.	29
4.3	10,000,000 の設計セットにおける 3 種類の簡単な統計量.	35
4.4	各最適化手法の最適化過程の比較.	42
4.5	最適化手法ごとの, 適応度の高い 8 つの設計.	43
4.6	最適化手法ごとの, 探索した設計の評価値の分布.	44
4.7	最適化手法ごとの, 評価値の高さと多様性の関係性.	45
4.8	ランダムフォレストによる, 評価値に対する設計特徴の重要度.	46
4.9	2次元の Partial Dependence Plot による, 重要な部品の配置位置の可視化.	47
5.1	実験に用いる, 8 つのタスクとそれぞれ 2 つの設計のペア	58
5.2	PPO-Optimized の設計に対する, 各手法による制御最適化の過程.	59
5.3	Hand-Designed の設計に対する, 各手法による制御最適化の過程.	61
5.4	Thrower-v0 タスクに対して, Hand-Designed 設計の制御を, HyperNEAT と PPO によって最適化したときの動作の様子.	62
5.5	PlatformJumper-v0 タスクに対して, Hand-Designed 設計の制御を, HyperNEAT と PPO によって最適化したときの動作の様子.	62
5.6	Walker-v0 タスクに対して, Hand-Designed 設計の制御を, NEAT と HyperNEAT とによって最適化したときの動作の様子.	63
5.7	Flipper-v0 タスクに対する PPO-Optimized 設計, Climber-v0 タスクに対する Hand-Designed 設計, それぞれで, NEAT と HyperNEAT とによって最適化したときの動作の様子.	64

6.1	CPPN を用いた環境の生成.	73
6.2	集団に加わったロボット設計.	76
6.3	生成された環境の履歴.	77
6.4	環境の変化で得られた特徴的な3つの環境と、それらに全てに適応した1つの ロボット設計.	78
6.5	3つの環境における適応度の変化と転送の例.	79
6.6	固定環境に対する設計最適化で得られた最適なロボット設計と動作.	80
6.7	固定環境に対する PPO で制御最適化で得られた動作.	81
6.8	主要なロボットの人口遷移.	82
6.9	環境の特徴の遷移.	83

表目次

2.1	本研究で利用するベンチマークタスクの一覧.	12
4.1	10,000,000 の設計を生成した際の, 組み立てに使用する部品の数 k ごとの生成試行回数, 重複しなかった回数, 重複した回数, 生成可能な設計数の推定値, 生成に失敗した回数.	34
4.2	設計に対して計算される各種特徴量.	36
4.3	ランダムフォレストのハイパーパラメータ候補	40
5.1	各タスクに設定されているロボットの観測値	54
5.2	超立方体上に配置したニューラルネットワークの入力ノードと出力ノードの座標	57
5.3	PPO-Optimized の設計に対する, 各手法による制御最適化の結果.	60
5.4	Hand-Designed の設計に対する, 各手法による制御最適化の結果.	61
6.1	PPO の主要なハイパーパラメータ.	75
6.2	提案手法ののハイパーパラメータ.	75
6.3	環境の変化で得られた特徴的な 3 つの環境それぞれに適応度の高い 8 つのロボット設計.	77
6.4	固定環境に対する設計最適化によって得られた, 3 つの環境それぞれの適応度の高い 8 つのロボット設計.	79

第1章 序論

近年のコンピューテーション基盤の発達に伴い、仮想的な物理シミュレーション技術を用いたロボット開発が促進されている。従来のロボット開発は、実際のハードウェアの設計、製作、試験に膨大な時間とコストを要し、設計の欠陥や動作の不具合を発見するために多くの試行錯誤が必要だった。ロボットの設計段階で発見される欠陥や不具合は、物理的なプロトタイプを何度も作成し直すことで解決されることが多く、その過程でのリソース消費は非常に高くつくものであった。さらに、実環境での試験は高価であり、故障のリスクも伴うため、試験プロセス自体が制約を受けていた。しかし、シミュレーション技術の発達により、これらの問題は大きく改善された。シミュレーション技術を用いることで、仮想環境内でのモデル試作や動作検証が可能となり、設計段階での欠陥検出や制御アルゴリズムの最適化が容易になった。この技術により、物理的なプロトタイプを作成する前に多様なシナリオをテストすることができるため、時間とコストを大幅に削減し、安全性と効率性が大幅に向上している。シミュレーションを通じて、ロボットが異なる環境や条件に適応する能力を事前に学習させることも可能となり、実環境での実装前に十分なテストを行うことで、開発サイクルの短縮と品質の向上を実現している [12]。シミュレーション技術は現代のロボット開発において重要な役割を担っている。

さらに、シミュレーション技術の進展はより複雑なロボット開発の可能性を広げる。困難な環境を想定したシミュレーションにより、極限環境や災害現場におけるロボットの動作を事前に検証できるようになり、実環境での試験が難しい条件下でのパフォーマンス評価が可能となった。また、新素材を使用したロボットのシミュレーションは、柔軟素材や複合材料の特性を最大限に活かしたデザインの試行錯誤を容易にし、設計の革新を促進している。さらに、生物を模倣したロボット、いわゆるバイオミメティックロボットの開発も進展しており、自然界の動きを再現するための詳細なシミュレーションが求められている。こうした複雑なロボットの開発は、人間の生活や産業における多様な課題解決に新たな道を拓くものであり、その意義は極めて大きい。

Sims の仮想生物の研究 [59] は、生物的なロボットの研究における重要な転機となり、その後多くの注目を集めることとなった。Sims の研究は、進化的アルゴリズムを用いて仮想環境内で生物のように動作するクリーチャーを自動生成し、適応的な行動を進化させるというものであった。このアプローチは、自然界の生物の進化プロセスを模倣することで、創造性豊かで予期しない形態や動作を持つロボットを生み出す可能性を示した。特に、柔軟で適応性のある動作を持つ仮想生物の生成は、従来の設計手法では達成できなかった新しいロボットデザインの可能性を広げた。この研究成果は、生物模倣や進化ロボティクスの分野に大きな

影響を与え、より生物的な動作や適応能力を持つロボットの開発に向けた研究が活発化するきっかけとなった。

近年、生物のような柔軟なロボットを設計することができる開発方法として、ボクセルベースのソフトロボット（VSRs）に注目が集まっている。VSRsは、ボクセルと呼ばれる複数の小さな立方体から構成されており、それぞれのボクセルが独立して動作することが可能である。この構造により、非常に柔軟で適応性の高いロボットの設計が可能となり、従来の硬い素材で構築されたロボットでは実現できない動作や形状変化が可能である。特に、複雑な地形や狭小空間での移動、柔軟な物体の取り扱いが求められる状況において、その利点が顕著である。さらに、ボクセルベースのアプローチは、シミュレーションを通じて最適な形状や動作パターンを迅速に試行錯誤できるため、開発の効率化にも寄与している。VSRsの研究は、医療、救助活動、探索ミッションなど、さまざまな応用分野での革新的な解決策を提供する可能性を秘めており、その進展が期待されている。

VSRsの開発には、設計と制御の両面で多くの困難が伴う。まず、設計の難しさが挙げられる。ボクセルベースのロボットは、その構造が柔軟であり、ボクセルの配置や結合方法によって全体の柔軟性や強度が大きく変化する。そのため、最適な設計を見つけるには、多数のシミュレーションと試行錯誤が必要となり、設計過程が非常に複雑で時間がかかる。また、制御の難しさも大きな課題である。各ボクセルが独立して動作するため、全体の動作を形成するにはそれぞれの動作を精密に制御し、協調させる必要がある。これには高度な制御アルゴリズムが不可欠であり、設計段階で可能な動作を予測することが難しいため、制御アルゴリズムの設計と実装には高度な技術が求められる。

さらに、VSRsの設計と制御は密接に関連していることが大きな特徴である。最適な設計を達成するためには、ボクセルの配置や形状、素材がロボットの動作にどのような影響を与えるのかを詳細に理解する必要がある。これには高度な制御アルゴリズムが不可欠である。同時に、効果的な制御には、ロボットが制御しやすい形状や構造を持つことが求められる。このように、設計と制御は相互に依存しており、一方を最適化するだけでは十分ではない。このため、設計と制御を同時に最適化するアプローチが重要となっているが、これもまた大きな研究課題である。進化的アルゴリズムや機械学習を用いた共進化的手法が注目されており、設計と制御の両面から最適な解を見つけ出すことが期待されている。この相互最適化のプロセスは複雑であり、ロボット工学における新たな挑戦として取り組まれている。

VSRsの開発においては、設計と制御だけでなく環境も重要な要素の一つである。従来のロボット開発では、ロボットは特定の環境やタスクに対して最適化されることが多く、静的で予測可能な環境での動作を前提として設計されてきた。しかし、現実の環境は時間とともに変化し、予測不可能な状況が発生することがある。VSRsはその柔軟性から、このような動的な環境に対して頑健であることが期待されている。そのため、VSRsの最適化においては、ボクセルの配置や形状、素材だけでなく、環境の変化を考慮した設計と制御が必要である。これを達成するためには、環境との相互作用をシミュレーションで再現し、動的な条件下での最適な設計と制御を探求することが求められる。

こうした複雑な課題を孕んだVSRsの研究開発では、簡略化した2Dシミュレーション [17,

5]が主に用いられる。初期の研究では3Dシミュレーションが主流であり、複雑な形状や動作のシミュレーションが行われていた[27]。しかし、VSRsの設計や動作の複雑さから、3Dシミュレーションには膨大な計算リソースと時間が必要であり、設計や制御アルゴリズムの試行錯誤が困難であった。このため、研究者たちはより簡略化した2Dシミュレーションを採用するようになった。2Dシミュレーションでは計算負荷が軽減され、迅速なプロトタイピングや多数の試行錯誤を行うことが可能であるため、研究効率の向上に寄与する。このアプローチは、ロボットの基本的な動作原理を理解し、効果的なデザインを見出すための初期段階として非常に有用であり、その後の3Dシミュレーションや実環境での試験に向けた土台を築く役割を果たしている。

そこで本研究では、2Dシミュレーションを用いて、より効果的なVSRsの設計と制御の最適化アルゴリズムについて検討し、今後のVSRs研究開発の促進に貢献する。VSRsの開発において重要となる、設計、制御、環境の3つの観点から、以下の3つを研究課題として設定した。

- 設計の特徴量を定義し、その特徴量に基づいて設計を最適化するアルゴリズムを構築することで、タスク遂行に寄与している特徴量を明らかにする。
- 異なる特性を持った制御の最適化手法を実験により比較し、設計との共同最適化に対する影響や効率性について検討する。
- 動的に変化する環境に対して設計と制御を共同最適化するアルゴリズムを構築することで、環境の変化に頑健な設計を明らかにする。

1つ目の研究課題では、VSRsにおける設計の特徴量に基づいて設計最適化をすることで、どのような設計の特徴によってタスク遂行しているのかを明らかにする。VSRsにおけるボクセルの配置や形状が動作にどのような影響を与えるのかを説明することは極めて重要である。VSRsはその柔軟性から、無数の構造を設計することが可能であり、その設計の可能性は膨大である。しかし、この広範な設計空間から効率的に最適な設計を探索することは非常に困難である。ボクセルの配置や形状がロボットの動作や性能にどのように影響を与えるかを理解することで、探索プロセスを効率化し、より早く最適な設計に到達することが可能となる。

従来のVSRsの設計最適化には、direct encoding と indirect encoding の2つのアプローチが存在する。direct encoding は、個々のボクセルの配置や形状を直接的に探索するアプローチである。例えば、2DのVSRsで 5×5 の設計の場合には、合計25次元の空間を遺伝的アルゴリズムなどによって探索する。このアプローチは設計を詳細に制御することが可能であるが、最小単位のボクセル単位で設計を探索するため、設計全体の有効性を理解するには細かすぎる。一方、indirect encoding は、設計を間接的に生成するニューラルネットワークを探索するアプローチであり、大規模な設計空間を効率的に探索できる。しかし、この方法ではニューラルネットワークがブラックボックスであるために、設計の有効性を説明することが難しく、生成されたデザインがなぜ効果的であるのかを理解するのが困難である。

そこで本研究では、設計最適化の新しいアプローチを提案する。まず、1つ1つのボクセルではなく、いくつかのボクセルで構成される部品単位の組み合わせで可能な設計で、設計空

間を再定義する。これにより、細かい最小単位ではなく、設計を部品単位で捉えることが可能となり、それぞれの部品の動作における役割を理解することが可能となる。そして、ニューラルネットワークのようなブラックボックスな中間表現ではなく、直接的に解釈可能な設計の特徴量を定義し、この特徴量空間で設計を探索する。これにより、どのような設計がタスクの遂行性が高いかを理解することが可能である。特に、物体操作などのタスクにおいては、設計のある特定の部分的構造が重要な役割を担っている可能性が高い。そのため、設計におけるそれぞれの部品の配置情報を特徴量に着目して分析する。

また、探索可能な設計に制限を加えることによって、より多様な設計を探索することができる可能性がある。探索が多様であることは、最適化を行う面や、設計空間全体の理解する面で重要な要素である。VSRsは、その柔軟性と構造の多様性から、無数の設計が可能であり、設計空間は非常に広範で複雑である。この広大な設計空間を効率的に探索するためには、単一の設計に固執せず、多様な設計を幅広く試行することが不可欠である。さらに、多様な設計を探索することは、設計空間全体の構造や特性を把握し、どのような配置や形状が効果的な動作や性能をもたらすかについて深く理解することにもつながる。探索の多様性の観点で、従来のアプローチと比較し、提案アプローチの有効性についても検証する。

2つ目の研究課題では、制御を最適化する手法について比較実験を行い、VSRsにおけるそれぞれの手法の有効性や、設計との共同最適化に与える影響について検討する。VSRsの動作は非常に複雑であり、同じ設計であっても多様な動作が可能である。制御を最適化するアルゴリズムとして、進化的手法や強化学習が挙げられるが、それぞれ異なる最適化の傾向があるために、手法によって引き出すことができる動作性が異なることや、タスクとの相性があることが想定される。さらに、制御の最適化アルゴリズムは、設計との共同最適化に大きな影響を与える。設計と制御は相互に依存しており、制御の最適化によって引き出される動作性が異なることによって、最適な設計も大きく異なることになる。そのため、制御の最適化アルゴリズムを比較しその特性傾向を明らかにすることは、今後のVSRs開発において大きな価値がある。

VSRsは、その複雑な動作性のため、制御にはニューラルネットワークが用いられる。VSRsは多様な動作パターンにより、従来のロボット制御手法では対応が難しい複雑な制御問題を抱えている。ニューラルネットワークは、このような複雑な制御問題に対処するための強力なツールである。ニューラルネットワークは多くの入力変数を同時に処理し、非線形な関係をモデル化する能力を持つため、ボクセルごとの動作を精密に制御し、全体として協調させることが可能である。ニューラルネットワークを最適化する手法には、進化的手法と強化学習が利用可能である。強化学習は制御の最適化として最もスタンダードな手法であり、頻繁に利用される。強化学習では、1つのニューラルネットワークに対して、そのパラメータを勾配法により更新することで最適化する。勾配法による安定で効率的な最適化が期待できる。本研究では具体的に、VSRsの制御を最適化した先行研究でも用いられている Proximal Policy Optimization (PPO) [58] を実験に使用する。一方、進化的手法では、強化学習とは対照的に、集団的に最適なニューラルネットワークを探索することが特徴である。広く多様な解を探索する傾向があり、最適化に時間を要するが、局所最適に陥りにくい利点がある。実験には、ニューラルネットワークの構

造を遺伝的に進化させる Neuroevolution of Augmenting Topologies(NEAT) [65] と、ニューラルネットワークの接続パターンを遺伝的に進化させる Hypercube-based NEAT(HyperNEAT) [64] を用いる。NEAT は、ゲームにおけるキャラクターの動作制御で有効性が示されている [71]。HyperNEAT は、VSRs のボクセルの連動パターンを効率的に制御できる可能性がある。

本研究では、多数の設計-タスクのペアに対して、3つの制御最適化アルゴリズムを比較実験する。制御の最適化アルゴリズムによって、同じ設計であっても引き出される動作性が異なること、同じタスクであっても優れた設計が異なることを確認する。また、各タスクに対して用意する設計には、制御の最適化に PPO を用いて設計の共同最適化実験を行った先行研究 [5] で得られた設計と、人手で有効性が高くなることを期待してデザインした設計の2パターンを用意した。前者は、PPO が動作性を引き出すことができる設計であり、後者はそういったバイアスのない設計である。この2つの設計で制御の最適化結果を比較することで、使用するアルゴリズムによる設計の共同最適化に与える影響について検討する。

3つ目の研究課題では、環境を動的に変化させる中で、VSRs の設計と制御を共同最適化するアルゴリズムを構築し、環境の変化に頑健な設計を明らかにする。VSRs は従来のロボットとは異なり、実世界の変化する環境でも動物のように柔軟に適応できることが期待されている。従来のロボティクスと異なる利点についてより深く理解することは、VSRs の研究開発の促進において重要な意義を持ち、今後の社会実装にも大きく貢献する。特に、環境の変化を考慮することは、災害現場や海中など人が作業するのが困難である環境においても安定的なロボットの開発を目指す上で必要不可欠である。

しかしながら、変化する環境を考慮した VSRs の最適化研究は、過去に十分に行われていない。これまでの研究は VSRs の動作や設計の複雑性に着目した研究が主流であり、環境には単純なものを想定することがほとんどであった。しかしながら、単純な環境に対する最適化では、過適合した設計が得られる可能性があり、柔軟な適応性を持つ VSRs として誤った解釈を引き起こしてしまうことが危惧される。VSRs は1つの設計であっても多様な動作を生み出すことが期待されていることから、複数のタスクをこなすことを目的とした研究 [32, 69] が行われている。こうしたマルチタスクでの評価によっても過適合が抑制されることが期待されるが、環境の変化について考慮することの意義はこれとは別に存在している。

環境の複雑化とロボットの制御の最適化を相互作用させるアルゴリズムである Paired Open-Ended Trailblazer (POET) アルゴリズム [76, 77] が過去に提案されている。実験は、2D の単純な二足歩行ロボットのシミュレーションプラットフォームである Bipedal Walker [33] を用いて行われた。POET アルゴリズムは進化的手法の一種で、環境とロボットの制御のペアを1つの単位として扱い、このペアの集団を更新していくことでアルゴリズムが進行する。それぞれのペアで、ロボットは環境に適応するように制御を最適化し、一定回数更新するごとに、新たな環境を生成し集団に追加する。古い環境は徐々に置き換わっていくが、ロボットの制御機構は多数の環境間で転送し合うことで、多様な環境に対して頑健な制御へと洗練されていく。複雑で困難な環境であっても、他の環境で最適化された制御を転送することで、適応できることを示した。

そして、POET アルゴリズムを利用し、同時にロボットの設計を最適化する実験が行われて

いる [66]. 当研究でも POET と同様に Bipedal Walker を利用して実験を行なっている. ここでは, ロボットの設計パラメータとして, 足の長さとかさという単純な要素を扱っている. この研究では, ロボットの設計パラメータと制御のパラメータを1つのゲノムとして扱い, 遺伝的アルゴリズムによって設計と制御を同時に進化させている. これは, ロボットの設計パラメータが変化しても, センサーの数や駆動機構などが変化しないため, 共通の制御機構を利用できるという前提で成立している. VSRs においては, 設計が異なった場合には, センサーの数や駆動ボクセルの数も異なるため, 共通の制御機構を利用することはできない.

本研究では, 変化する環境の中で VSRs の設計と制御を共同最適化するために, POET を拡張し新しいアルゴリズムを構築する. VSRs では1つの設計に対してそれぞれ専用の制御機構を用意する必要がある. つまり, POET における環境-ロボットのペアを, 環境-(設計, 制御)のように, 明確に設計と制御を分離して記述する必要がある. 従って, 制御の最適化とは独立して設計を最適化するモジュールを追加することで POET アルゴリズムを拡張する. これに伴い, 他の環境へロボットを転送する処理においては, 2つの場合分けが発生する. まず制御のみを転送する場合である. 制御は同一の設計でのみ有効であるため, 転送先の環境にすでに同一の設計がペアとなっている場合には制御の転送が発生する. そして, 転送先の環境に同一の設計が存在しない場合には, 設計と制御のペアの転送が発生する. このように POET を拡張することで, VSRs において変化する環境に頑健な設計を得ることが期待される.

また, POET アルゴリズムは多数の環境で制御を最適化することで, ロボットの動作性を引き出すことができる点で優れている. これは VSRs においても有効であることが想定され, 1つの静的な環境に対する制御の最適化では引き出すことのできない動作性を引き出すことができ, 設計の有効性を公平に評価することができる. つまり, 設計と制御の共同最適化によって, より適応的な VSR を得ることができる可能性を示している. 多数の環境を用いることの効果を評価するために, 提案アルゴリズムで得られるいくつかの環境に対して, 従来の設計と制御の共同最適化実験を行う. この比較により, 多様な環境で設計を最適化することの利点を明らかにする.

これらの研究課題に取り組むことで, VSRs の設計, 制御, 環境適応の各要素を総合的に最適化するための新しい手法を開発し, ロボット工学における新たな知見を提供することを目指す. また, 本研究の成果は, 将来的な実環境での VSRs の応用可能性を高めるための重要なステップとなり得る. 最終的には, 柔軟かつ適応性の高いロボットの実現に向けた重要な基盤を築くことを期待している.

本論文は合計7章で構成される. 1章は序論である. ここでは本研究の目的とその意義について詳細に述べた. 2章は背景である. VSRs の設計思想や本研究で利用するシミュレーションプラットフォームの特徴, 基本となる VSRs 設計開発の概念や手法について詳細に説明する. 3章は関連研究である. これまでに行われてきた VSRs に関する研究を俯瞰することで, 本研究の意義を強調する. 4章では, VSRs の設計特徴に基づく最適化手法を提案し, 探索性能の観点で従来手法と比較する. さらに, 実験結果を分析することで, タスクの遂行に寄与している設計の特徴を明らかにする. 5章では, VSRs の制御を最適化する手法を実験により比較する. 多数のタスク-設計のペアに対する制御最適化手法の比較することで, 各手法の特

性を理解し、それぞれを VSRs 開発に用いることについて検討する。6 章では、変化する環境に対して VSRs の設計と制御を共同最適化するアルゴリズムを提案し実験する。環境の変化に対して頑健な設計を明らかにするとともに、VSRs の最適化に多様な環境を用いることの影響を検討する。7 章は結論である。本論文の成果を総括し、また、今後の展望と課題について述べる。

第2章 背景

2.1 ボクセルベースソフトロボット (VSRs)

2.1.1 VSRs 開発の経緯

VSRs が研究されるようになった経緯は、ソフトロボット技術の進展とその応用可能性に根ざしている。ソフトロボットとは、従来の剛体ロボットとは異なり、柔軟で適応力のある構造を持つロボットである。これは自然界に存在する生物の特徴を模倣することに基づいており、特に生物の柔軟な動きや複雑な形態を再現することを目指している [57]。例えば、タコやイカのような軟体動物は、その柔軟な体を使って狭い隙間を通り抜けたり、異なる形状に変形して捕食や防御を行ったりする。このような生物の特性を取り入れることで、ロボットに新たな機能を持たせ、従来のロボットでは実現できなかった動作や環境への適応を可能にしようという考え方が生まれた。

次に、ソフトロボットの開発には3Dプリンティング技術が不可欠であることを強調する。この技術は、複雑な形状や多様な材料を使用した部品を迅速かつ正確に製造することを可能にする。3Dプリンティングを利用することで、ソフトロボットのプロトタイプを短時間で作成し、設計の反復改良を行うことができる。特に、ソフトロボットの複雑な内部構造や多機能性を実現するためには、3Dプリンティングの精度と材料の多様性が非常に重要である [20]。これにより、研究者は迅速に試作品を製作し、実際の動作や性能を確認しながら最適な設計を追求することができるようになった。

さらに、3Dプリンティング技術とボクセルの概念が融合することで、VSRsの基盤が形成された。ボクセルとは、3次元空間を小さな立方体（体積素子）に分割する方法であり、これにより複雑な3次元構造を効率的に表現することができる。ボクセルの積層によって、微細な構造や動作を詳細に設計・制御することが可能となり、ソフトロボットの柔軟性と適応性を最大限に引き出すことができる。この技術は、画像処理や3Dモデリングで広く利用されており、特に医学の分野ではCTスキャンやMRIなどの画像解析に使用されている [81]。ボクセルの積層を用いることで、ロボットの設計者は細部にわたる構造を詳細に制御し、特定の機能や動作を実現するための精密な設計が可能となった。

VSRs への着想は、これらの技術的進展を背景に生まれたものである。ボクセル単位での設計は、ロボットの形態や機能を微細に制御することを可能にし、従来のロボット設計手法では困難であった複雑な形状や動作パターンを実現するための強力なツールとなった。まず、VSRs は非常に高い柔軟性と適応性を持つ点が挙げられる。これは、各ボクセルが独立した機能を持ち、それらを組み合わせることで複雑な動作や形状を実現できるためである。例え

ば、タコのように柔軟に体を変形させることができる VSR は、狭い隙間を通り抜けたり、複雑な地形を移動したりすることが可能である。また、モジュール性が高いことも VSRs の大きな特徴である。各ボクセルがモジュールとして機能するため、必要に応じて特定の機能を持つボクセルを追加したり、交換したりすることが容易である。これにより、用途に応じてロボットの機能をカスタマイズすることができる。さらに、可変形性も重要な特徴の一つである。ボクセルの配置や特性を変更することで、ロボット全体の形状や動作を動的に変更することが可能である。例えば、障害物を回避するために一時的に体の形を変えることができるロボットは、多様な環境に柔軟に対応することができる。VSRs は従来のロボット設計手法では実現できなかった柔軟で適応力のあるロボットを可能にし、多様な用途に対応するための強力な設計手法であると言える。

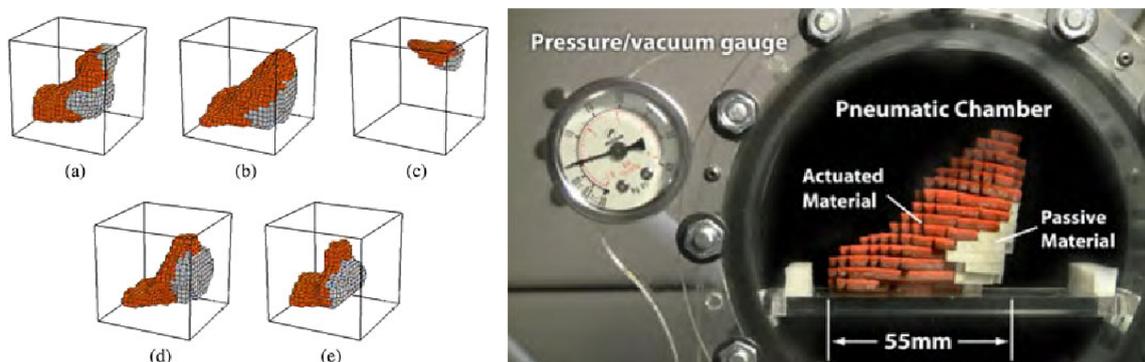


図 2.1: VSRs 研究の先駆け。左：シミュレーション状での VSRs 設計の探索。右：現実への VSR の再現と動作テスト。[24] より引用。

VSRs の最初の研究は Hiller らによって行われた [24]。ここでは、図 2.1 左に見られるように 3D シミュレーション上で VSR を設計し、図 2.1 右のように 3D プリンティングによって実際のロボットを製造することで、その動作をテストした。ボクセルは周期的に伸縮を繰り返すアクチュエータと動作しない 2 種類で構成され、この 2 種類のボクセルの配置を進化的アルゴリズムによって最適化した。最も単純な移動タスクに対してシミュレーション上で最適化を行い、実際に製造し移動性能をテストした。その結果、シミュレーションと実際の物理実験の移動距離が一致し、VSRs のシミュレーションによる最適化の有効性が示された。しかし、ここで取り扱ったアクチュエータは規則的な伸縮を繰り返す単純なもので、複雑な動きを形成するには至らなかった。VSRs のより高い適応性や柔軟性を引き出すには、それぞれのボクセルに独立した制御を行う必要があるが、そうした複雑な物理制御を求めるほど現実への移行が困難となる。実際に複雑な制御を行うことができるようなボクセル素材の開発も進んでおり [56, 28]、今後ますます重要になると考えられる。

2.1.2 シミュレーションプラットフォーム

VSRs の性能向上と設計効率の最適化を目指して、リアルタイムかつ高精度なシミュレーション環境の開発が進んでいる。このシミュレーション環境は、複雑なボクセル構造の動作を正確に再現し、多様な材料特性や動作パターンを詳細に評価することができるものである。これにより、設計プロセスの初期段階から詳細な動作予測と最適化が可能となり、実際のプロトタイプ製作前に問題点や改善点を明確にすることができる。また、このシミュレーション環境は、進化計算や生成的設計のアルゴリズムと連携して、最適なボクセル配置を自動的に探索する機能を持ち、設計の効率と精度を飛躍的に向上させる役割を果たしている。

Hiller らによって、最初の VSRs のシミュレーション環境が開発された [27]。ここでは、ボクセルは 3D 空間内の立方体セルとして扱われ、剛性、密度、ポアソン比、熱膨張係数、摩擦係数など、異なる材料特性を設定することができる。このボクセルを積み重ねることでロボットが構築される。提示されている四足歩行ロボットの例では、各脚のボクセルが決まった周期で膨張および収縮を繰り返すことにより、脚の上下運動が生じ、ロボット全体が前進する歩行動作が実現される。これらの動作パターンは、各ボクセルにあらかじめ設定されたアクチュエーションパターンに従っており、ボクセル間の内部力の計算と位置更新を繰り返すことで、ロボットの動きをリアルタイムでシミュレートする。これにより、複雑な動的挙動を持つロボットの設計と制御が効率的に行われている。しかし、3D の高精度なシミュレーションのために、非常に多くの計算資源が必要となることや、複雑な設定やカスタマイズには専門知識が必要となるなど、簡単に利用することは難しい。

計算負荷を軽減してより効率的に VSRs の開発をするために、2D の VSRs シミュレーション環境が Medvet らによって開発された [17]。2D でロボットを設計することができるため、3D に比べて設計空間が大幅に縮小され、迅速な最適化実験を可能とした。また、センシング機能が組み込まれており、ロボットは環境からのフィードバックを受け取り、それをもとに各ボクセルの動作を決定することで、より高度で柔軟な制御を可能としている。さらに、視覚化機能が充実しており、複数のシミュレーション結果を直感的に比較・分析できることも VSRs の研究開発に大きく貢献している。

Evolution Gym

本研究では VSRs のシミュレーション環境として Evolution Gym を利用する。Evolution Gym は、VSRs の設計と制御の最適化実験を促進させることを目的として Bhatia らによって開発された [5]。これまでのボクセル特性を詳細に設定できるシミュレーションと異なり、4 種類の決められたボクセルを組み合わせることでロボットを設計する。これにより、利用者はより手軽に VSRs の最適化実験を行うことができる。また、これまでは地面の歩行や水中での水泳など、単純なタスクで実験が行われていたが、Evolution Gym では異なる特徴を持った地形での移動や、様々な物体操作を行う、合計 32 のベンチマークタスクが用意されており、より複雑な VSRs の動作テストが可能となった。

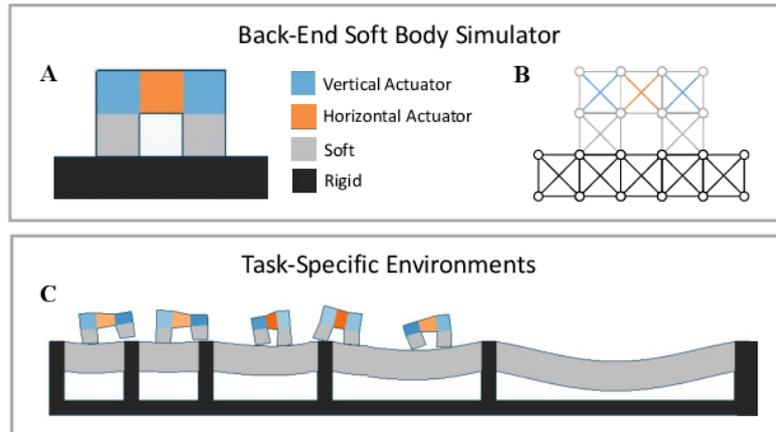


図 2.2: Evolution Gym の概要. [5] より引用.

Evolution Gym では、図 2.2 A に示されているように、4 種類のボクセルを使用することができる。Rigid は硬い素材で変形しにくく、ロボットの骨格や基盤として使用され、形状を維持しながら力を伝える役割を果たし形状を維持しながら力を伝える役割を果たす。Soft は柔らかい素材で容易に変形する。そして、Vertical Actuator は縦方向に、Horizontal Actuator は横方向に伸縮することができるボクセルであり、この 2 つを適切に制御することでロボットを動かすことができる。これら 4 種類のボクセルを組み合わせることで VSR を設計することができる。また、図 2.2 C で示されているように、タスクごとの動作テストを行う環境も Rigid と Soft をグリッド状に配置することによって定義されている。

本研究の実験で利用する、Evolution Gym で提供されているベンチマークタスクを表 2.1 に示す。タスクは歩行タスクのような単純で簡単なタスクから、物体を持ち上げるような複雑な動作を要する難しいタスクがある。開発者である Bhatia らは、それぞれのタスクのおおよその難易度を、easy, medium, hard の 3 つの段階で表現している。それぞれ異なる難易度や異なる動作を求められるタスクで、VSRs の最適化実験をすることで、より深い洞察を得ることができる。各タスクでは、それぞれ決められたタイムステップだけシミュレーションを繰り返し、最終的な移動距離や状態で、評価値が計算される。本論文の 4, 5 章で扱う実験では、表 2.1 のうち、それぞれ 8 つのタスクを用いる。また、Evolution Gym では提供されているベンチマークタスクだけではなく、利用者が独自のタスクを開発することも容易である。32 のベンチマークタスクでは、それぞれ全て決まった固定の環境が定義されているが、6 章での実験では環境を変化を扱うために、環境情報を入力として与えることができるタスクを新しく定義する。

シミュレーションは連続時間で動作し、各タイムステップごとにロボットはセンサーから情報を受け取り、次のタイムステップの動作を決定する。センサーは、ロボット自身の状態、環境の状態、目的となる物体の状態の 3 種類に関して定義されている。まず、ロボット自身の状態として、VSR を構成するグリッド状に配置されたボクセルの各頂点の位置情報と速度、

表 2.1: 本研究で利用するベンチマークタスクの一覧.

タスク名	難易度	概要
Walker-v0	easy	平坦な地形を横に移動する.
Climber-v0	medium	両端の壁を利用して上に登る.
ObstacleTraverser-v0	medium	凹凸のある地形を横に移動する.
PlatformJumper-v0	hard	高さの異なる足場を横に移動する.
Hurdler-v0	hard	複数の壁のある地形を横に移動する.
Flipper-v0	easy	平坦な地形を反時計回りに回転する.
Thrower-v0	medium	物体を遠くに投げる.
Carrier-v0	easy	平坦な地形上で物体を落とさずに運ぶ.
Carrier-v1	hard	段差のある地形形状で物体を落とさずに運ぶ.
Lifter-v0	hard	穴の中にある物体を持ち上げる.
BeamSlider-v0	hard	狭い空間に入り上部の物体を運ぶ.
AreaMaximizer-v0	easy	ロボットの幾何的面積を最大化する.

ロボット自身の角度の情報を受け取る. 環境の情報では, ロボットの周囲 1 マスごとの地面からの高さ, あるいは天井までの高さの情報を受け取る. そして, 目的となる物体の状態としては, ロボットから物体までの相対的な位置, 物体の速度と角度の情報を受け取る. また, ベンチマークタスクによって使用されるセンサーは異なる. 例えば, 平坦な地形が環境となるタスクや移動を伴わないタスクでは環境の状態に関するセンサーは使用されず, 物体操作をしないタスクでは物体に関するセンサーを使用されない. こうしたセンサーからの情報を処理し, ロボットは自身が持つ全てのアクチュエータボクセルそれぞれの伸縮度合いを決定することで, 連続的に複雑な動作を生み出す.

Evolution Gym では, 2次元で小規模な VSRs の設計空間を取り扱う. しかしながら, 例えばサイズが 5×5 の VSR でも, 各セルは 4 種類のボクセルあるいは空白の状態を取りうる. つまり, $5^5 \times 5^5$ だけの可能な VSR 設計がある. 実際には, アクチュエータボクセルが含まれないものや, ボクセルが完全に接続されていないようなものは探索対象から省かれる. しかし, それでも膨大な数から最適な設計を探索する必要があるため, 依然として困難な課題である. さらに, 制御についても, 複数のアクチュエータボクセルを, どのタイミングでどう動かすかによって, 全体としての動き全く異なってくるため, 問題は複雑である.

2.2 VSRs 最適化の基本的な方法論

次に, VSRs の基本的な最適化について説明する. VSRs の最適化では, 主に, 設計を最適化する手法と, 制御を最適化する手法の 2 つに大別される. しかし, 設計の最適化には, 設計の有効性を評価する必要があるため, 実際に設計ごとに制御シミュレーションを行なって

評価する。そのため、設計の最適化と制御の最適化は密接に関わっており、この2つを組み合わせる、設計と制御の共同最適化が重要となる。それぞれについて説明する。

2.2.1 設計の最適化

遺伝的アルゴリズム (GA)

VSRs の設計最適化で最も一般的で単純な手法が GA である。GA は、進化生物学における自然選択と遺伝のメカニズムを模倣した最適化手法である [22]。具体的には、GA は解候補となる個体の集合を生成し、各個体の適応度を計算することから始まる。次に、適応度の高い個体を選択し、交叉や突然変異といった遺伝操作を適用して新たな個体群を生成する。この過程を繰り返すことで、個体群全体の適応度が向上し、最適解に収束することが期待される。GA は、非線形かつ高次元の複雑な問題に対して有効であり、機械学習のハイパーパラメータ調整やロボット工学など、幅広い分野で利用されている。

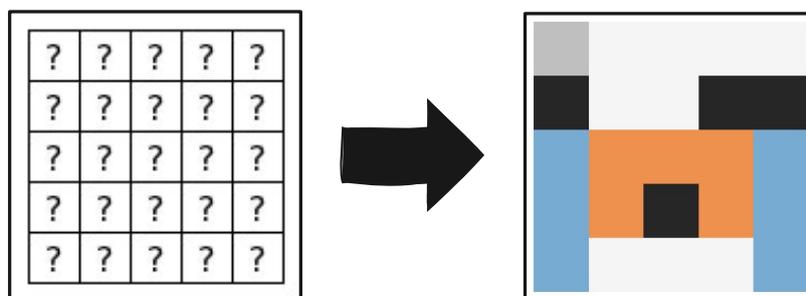


図 2.3: 2次元 VSRs の設計問題。グリッド状の各セルにボクセルを配置することで VSR は定義される。

こうした特性から GA は VSRs の設計最適化問題に有効に働くことが期待される。VSRs の設計問題は、図 2.3 に示すように、グリッド状の各セルに配置するボクセルの組み合わせから、最適な組み合わせを探索する問題である。GA では、このグリッドの各セル1つ1つを遺伝子として扱い、適応度の高い個体の遺伝子をランダムに突然変異させることで新しい解候補を探索する。このアプローチは、グリッド状の設計空間をそのまま直接的に扱うことから、*direct encoding* と呼ばれる。

VSRs の設計最適化に GA を用いることで、細かい局所的な設計の調整が可能となる。しかし、GA は優れた個体の形質を遺伝し、突然変異によって世代を重ねるうちに、次第に適応度の高い解に収束していく性質を持つ。この過程では、選択圧によって有望な解が強化される一方で、初期集団の多様性が失われる傾向があるため、設計空間の一部に集中しやすくなる。その結果、GA は局所的な探索には優れるものの、大域的な探索能力が制限され、広範な設計空間全体を効果的に探索することが難しい。2次元のような小さい設計空間では有効に働くも

の、特に、3次元のような設計空間が広大で複雑な場合、GAは局所最適解に収束するリスクが高まり、真の最適解を見つけ出すことは難しくなる。

Compositional Pattern Producing Network (CPPN)

CPPNは、遺伝的アルゴリズムと結びついて進化的アートや形態生成の領域で注目されている、人工ニューラルネットワークの一種である [63]。CPPNは、複雑なパターンや形態を生成する能力を持ち、その生成過程は人間の芸術的表現と類似する。具体的には、CPPNは座標情報を入力として受け取り、これを基にして出力として色や形状の値を生成する。この座標情報の変換プロセスは、伝統的なニューラルネットワークと同様に重み付けされた結合と活性化関数を用いて行われるが、CPPNでは複数の異なる活性化関数（例：ガウス関数、シグモイド関数、周期関数など）を組み合わせることで、より多様で複雑なパターンを生成することが可能である。さらにCPPNは、ニューラルネットワークを遺伝的アルゴリズムによって進化させる Neuroevolution of Augmenting Topologies (NEAT) アルゴリズム [65] と統合されることが多く、これにより、生成されたパターンの美的評価や適応度に基づいてネットワーク構造が進化し、最適化される。CPPNの特筆すべき点は、相対的に少ないパラメータで高次元のパターンを効率的に生成できることであり、この特性は進化的デザイン、ゲームキャラクターデザイン、ロボティクスにおける形態生成など、多岐にわたる応用において強力なツールとなっている。

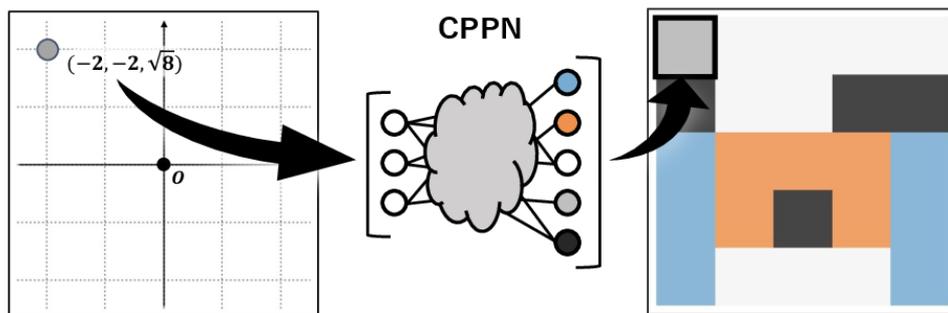


図 2.4: CPPN を用いた VSR の設計。グリッドの各セルの座標を CPPN に入力することでボクセルの状態を決定する。

CPPN を用いた VSRs の設計の生成方法を図 2.4 に示す。グリッド状の設計空間における、各セルの座標情報と中心からの距離を CPPN に入力することで、そのセルのボクセルの種類を決定する。Evolution Gym の場合には、4 種類のボクセルと空白が取りうる状態である。これに対応して、CPPN は 5 つの値を出力し、その中で最も大きい値を持つセルの状態に決定される。つまり、あるセルの状態 $cell_{x,y}$ は以下の数式で定義される。

$$cell_{x,y} = \mathbf{argmax} \text{ CPPN}(x, y, \sqrt{x^2 + y^2}) \quad (2.1)$$

このように CPPN を用いた設計アプローチは、グリッド状の設計空間ではなく、それぞれのセルの状態を間接的に決定するニューラルネットワークを扱うことから、indirect encoding と呼ばれる。

VSRs の設計最適化に CPPN を用いることで、効率的に設計を幅広く探索することができる。CPPN は、ニューラルネットワークによって複雑なパターンや構造を生成することができる。そのため、CPPN は設計空間の大域的な特徴を捉えやすく、特定のパターンや形状を広範に探索できるため、初期段階での設計探索には非常に効果的である。しかし、一方で CPPN では複雑な設計を探索することが難しい。これは、CPPN が生成するパターンが関数の組み合わせによって決定されるため、設計の細部を精密に調整する能力が制限されるためである。複雑な設計や細かい機能的な特徴を必要とする場合、CPPN の生成する設計はしばしば粗い近似に留まることが多い。そのため、特に 2次元の狭い設計空間より、3次元のような広大な設計空間で有効に働くことが期待される。

2.2.2 制御の最適化

旧来の古典的なロボットの制御方法は、主にルールベースやフィードバック制御に依存してきた。これらの方法では、予め定められたルールや方程式に基づいてロボットの動作を制御し、センサーから得られるフィードバックを利用して目標値に近づけるよう調整する。PID 制御（比例積分微分制御）がその代表例であり、比較的単純なタスクや静的な環境での運用において高い効果を発揮していた [23]。しかし、これらの手法は、予測可能で固定された環境やタスクに最適化されているため、動的で複雑な環境に対する柔軟性や適応性に限界がある。

より複雑な環境やタスクに対応するために、近年ではニューラルネットワークを用いた制御方法が注目されている。ニューラルネットワークは、大量のデータから学習し、非線形な関係をモデル化する能力に優れているため、従来のルールベースやフィードバック制御では対応しきれない複雑な問題に対処できる。特に、強化学習を組み合わせたニューラルネットワークは、ロボットが環境との相互作用を通じて最適な行動を自律的に学習することを可能にする [29]。これにより、動的な環境や予測不可能な状況においても高い適応性を発揮し、複雑なタスクを効率的に遂行できるようになった。VSRs の、環境に合わせた複雑な動作の制御のためには、ニューラルネットワークを用いることが有効である。

Proximal Policy Optimization (PPO)

強化学習によってニューラルネットワークによる制御を最適化する優れた手法として、PPO が広く知られている。PPO は深層強化学習の一種であり、エージェントの現在の状態 s_t を観測として行動 a_t を決定するポリシー $\pi_\theta(a_t|s_t)$ を、勾配降下法によって最適化する手法である [58]。PPO はシンプルな学習構造とその安定性から、ゲームプレイや自動運転、ロボット制御など、幅広い連続制御タスクで利用されている [35, 55]。さらに近年では、大規模言語モ

デルによるチャットボットの開発にも利用されている [73] など、複雑な問題にも応用されている。PPO は、具体的に以下の手順を繰り返すことでポリシーの学習が進行する。

1. 現在のポリシー $\pi_\theta(a|s)$ に従って、環境からデータを一定回数サンプリングする。
2. データから目的関数 $L^{CLIP}(\theta)$ を計算する。
3. 目的関数を最小化するように、ポリシーのパラメータ θ を更新する。

まず、現在のポリシーを基準にして環境でランダムな行動をとることでデータをサンプリングする。エージェントはランダムな行動をすることで、新しい行動や新しい状態を探索し、より報酬を環境から探し出す。このランダムなサンプリングを一定回数繰り返すことで、エージェントが現在のポリシーのもとでどのように行動し、どのような報酬を得ているかを反映したデータセットが形成される。このとき、コンピュータシミュレーションなどの場合には、複数の環境から並列でデータをサンプリングすることで、より安定的で効率的な学習を行うことができる。

次に、収集したデータからポリシーを更新するための目的関数を計算する。このとき、ポリシーが更新前後で大きく変動してしまうと、学習が不安定になり、既に学習した有用な行動方針が破壊される可能性がある。そのため、PPO では、過去のポリシー $\pi_{\theta_{old}}(a|s)$ から変動しすぎないように、更新最中の現在のポリシー $\pi_\theta(a|s)$ との比率 $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ を用いて、以下のように目的関数 $L^{CLIP}(\theta)$ が定義する。

$$L^{CLIP}(\theta) = \mathbb{E}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t)] \quad (2.2)$$

ここで、 \hat{A}_t はデータ t における状態とった行動がどれだけ優れていたかを表す、深層強化学習において基本的な学習指標である。そして ϵ は、ポリシーの更新幅を直接的に制御するハイパーパラメータである。この目的関数によって、ポリシーの更新が大きくなりすぎるのを防ぎ、安定的な学習が保証されている。

最後に、この目的関数を最小化するように、勾配降下法によってポリシーのパラメータを更新する。このとき、収集されたデータセットを用いて、バッチ学習によってエポック更新する。これによって、同じデータセットから複数回学習できるため、効率的にデータサンプルを活用することができる。この手順を繰り返していくことによって、ポリシーは最適化され、複雑なエージェントの制御を可能とする。

2.2.3 設計と制御の共同最適化

従来、設計と制御は個別に最適化されることが多く、設計プロセスでは固定された物理的構造を前提とし、その後で適切な制御アルゴリズムを開発するという手順が一般的であった。しかし、この方法では、物理的設計が制御の制約を十分に考慮していないため、最適な性能を引き出すことが難しい場合が多かった。例えば、ロボットの形状が複雑であるほど、制御

アルゴリズムの設計は困難を極め、逆に制御の複雑さが設計の自由度を制限することもある。共同最適化はこの課題に対処するためのアプローチであり、ロボットの設計と制御を同時に、かつ相互に調整しながら最適化する必要がある。これにより、ロボットの全体的な性能を最大化し、動的な環境や複雑なタスクに対する適応性を向上させることが可能となる。

Sims が初めてロボットの設計と制御の共同最適化に関する研究を行ったことは、ロボット工学と進化的アルゴリズムの分野における画期的な出来事として広く認識されている。Sims の研究は、仮想環境内で進化的アルゴリズムを用いて自律的にロボットの形態とその動作制御を同時に最適化する方法を探求したものである [59]。このアプローチでは、進化的プロセスを通じて仮想生物が生成され、これらの生物は生存競争のシミュレーションを経て、より適応的な形態と制御を獲得していく。具体的には、身体構造や関節の配置などの生物形態と、移動パターンや環境応答の制御機構が同時に進化し、最終的に効率的に動作できるロボットが誕生する。この研究の革新性は、従来の設計と制御を別個に最適化する手法とは異なり、両者を統合的に進化させることで、より高度で適応性のあるロボットシステムを実現する点にあった。この研究によって、ロボットの設計と制御が密接に関連していることを示され、一体化した最適化の重要性が明らかとなった。また、Sims の研究は後の研究者たちに大きな影響を与え、進化的アルゴリズムや機械学習を用いたロボット工学の新たなアプローチの基礎を築いた。

しかし、設計と制御を共同で最適化する場合に、設計が早期に局所解に収束してしまう問題が、Cheney らによって報告された [8]。Cheney らはこの問題を、「身体化された認知」の理論 [40] に基づいて説明している。この理論は、知能や行動は身体と環境の相互作用によって形成されるという考え方である。ロボットにおいて、その形態は生物の制御機構と外部環境の間の情報伝達の役割を果たし、その形態が変わると、この情報伝達のチャンネルが混乱するため、制御機構が新しい形態に適応するのが難しくなる。このため、設計の変化が制御系に予測不可能で大きな影響を与え、最適化プロセスが複雑化し、結果として設計が早期に収束してしまう。さらに、設計の早期収束は、制御の進化を妨げ、最終的なフィットネス向上の可能性を制限してしまう。これに対し、制御の変化は設計の変化ほど外部環境との相互作用に劇的な影響を与えないため、制御は比較的長期間にわたり進化を続けることができる。したがって、設計の収束が制御の収束よりも早く、かつ問題の根本的な原因として身体化された認知の理論を提唱している。

この設計の早期収束問題は、ある設計に対して最適化された制御機構を、別の設計の制御に継承するために発生する。このアプローチは、異なる設計ごとに最初から制御を最適化することなく、学習された知能を他の設計に流用することができるため、計算処理を減らし効率的に共同最適化できる。しかし、その一方で、設計最適化のプロセスと制御最適化のプロセスが複雑に相互作用することになり、困難な問題を生み出している。

計算基盤の発達と、シミュレーションの単純化により、設計ごとに制御を最初から最適化するアプローチの実験ができるようになってきている。制御の最適化は、本来それ自体に多くの計算が必要となる問題であるため、それを何度も繰り返す必要があるこのアプローチは、膨大な計算が必要となる。これに対し、Evolution Gym のように 2次元に空間を単純化することに

よって、シミュレーション負荷が軽くなった。実際に、Bhatia らがおこなった Evolution Gym での VSRs 最適化実験は、設計ごとに制御を最初から最適化した [5]。特に、Evolution Gym での VSRs は、設計ごとに、入力となるセンサーの数や出力となるアクチュエータボクセルの数が異なるために、定義されるニューラルネットワークの構造が異なる。そのため、特殊なアプローチを取らない限り、設計ごとに制御を最適化する方法が、単純で自然である。また、設計の最適化と制御の最適化の相互作用の影響が小さくなるため、単純に設計と制御をそれぞれ独立で検討することができる点でも優れている。

第3章 関連研究

3.1 設計の最適化

仮想生物の形態の最適化において、遺伝的アルゴリズムが広く使用されてきた。この背景には、生物の進化過程を模倣することで最適な形態を探索する手法が有効であるという考えがある。GAは、自然選択の原理に基づき、個体群の中で最も適応度の高い形態を選択し、交叉と突然変異を通じて次世代に受け継ぐことで進化をシミュレートする手法である。Simsの研究は、この手法を仮想生物の形態と動作の最適化に応用した先駆的なものである[59]。Simsは、進化的アルゴリズムを用いて仮想環境内で仮想生物を生成し、その形態と行動パターンを同時に最適化することで、環境に最も適した形態と運動を持つ生物を進化させることに成功した。彼の研究では、遺伝的アルゴリズムが形態の多様性を維持しながら最適化を行う能力を示し、複雑な形態の生成や、環境への適応能力の高い形態を効率的に見つけ出すことができることを実証した。このアプローチは、後のロボット工学や人工生命研究における基礎となり、仮想生物の進化を通じて新たなデザインや制御方法の発見に貢献している。

初期の3次元VSRs(ボクセルベースのソフトロボット)の設計最適化研究では、Compositional Pattern Producing Networks(CPPN)[63]を遺伝的に進化させる手法が採用されてきた。それまでの多くのロボティクス研究では、過度に規則的なエンコーディングを使用しており、これが複雑で多様な形態を生成する能力を制約していた。Cheneyらは、CPPNを用いてボクセルの配置や形状を生成し、これを遺伝的アルゴリズムによって進化させることで最適なVSRs設計を見つけ出すアプローチを採用した[11]。CPPNは規則的でありながら多様なパターンを生成できるため、進化的に生成されたロボットがより効果的かつ自然で複雑な形態に進化させることができた。その後もCheneyらは、狭い隙間を通り抜けるタスクにCPPNを適用し、優れた性能を発揮することを確認した[9]。また、Corucciらは、水中と陸上での移動タスクに対してCPPNによってVSRsの設計を最適化した[14]。その結果、陸上では硬いボクセルが多くエネルギー効率の高い設計が得られ、水中では柔らかいボクセルでエネルギー効率の悪い設計が得られた。

3次元の広い設計空間では、CPPNが効率的に探索できていたが、2次元の狭い設計空間では遺伝的アルゴリズムの方が優れる傾向があることが示されている。Bhatiaらは、様々なタスクに対して、遺伝的アルゴリズムとCPPNとベイズ最適化の3つによるVSRsの設計最適化実験をおこなった[5]。5×5の設計空間では、CPPNは複雑な形状を探索することができず、ほとんどのタスクで、細かく設計を探索できる遺伝的アルゴリズムの方が安定的で優れる結果であった。Pigozziらがおこなった、遺伝的アルゴリズムとCPPNで設計最適化実験で

も、同様の結果が得られた [53]. ここではさらに、制御の学習傾向についても比較しており、遺伝的アルゴリズムは学習能力の向上、特に学習速度の向上に寄与し、CPPN は必ずしも学習能力の向上につながるわけではないと結論づけられた. このように、2次元の VSRs で優れる遺伝的アルゴリズムについて、Katayama らは、生成された設計候補を段階的に評価し、初期段階で有望でない個体を早期に除外することで、全体の計算コストを削減し、効率的な設計の最適化を実現した.

また、VSRs の設計空間は膨大で、動作の可能性も想定することが困難であることから、多様な設計を探索して評価することが、最適な設計を得るために重要である. そこで Mouret らによって提案されたのが Map-Elites である [48]. 従来の評価値のみに基づいた集団の維持ではなく、Map-Elites はユーザーが定義する特徴次元に基づいて、各特徴空間の領域における最高性能解をマップで維持する. このアプローチにより、異なる特性を持つ解の全体像を視覚化し、特定の局所最適解に陥ることなく広範な探索を可能にした. 従来の最適化アルゴリズムが見逃しがちな解を見つける能力が高く、多目的最適化や高次元の探索空間において特に有効であり、VSRs においても多様な設計の探索ができることを示した. Nordmoen らは、従来の遺伝的アルゴリズムと、集団の多様性を維持する特徴を持った NSGA-II と、Map-Elites を比較した [50]. この結果 MAP-Elites は、多様で高性能な設計のレパートリーを生成することに特に優れていることが明らかになった. また、遺伝的アルゴリズムにおいて、新しい設計を保護することで多様性を維持する方法も研究されている. Cheney らは、新しい設計に対して選択圧を一時的に減少させ、制御が新しい形態に再適応する時間を与える手法を提案した [10]. これにより、設計の変異が初期段階で適応度を低下させても、長期的にはより高い適応度を達成することを示した. Medvet らは、設計を種として保護することで、従来の手法よりも多様性を高めることに成功した [42].

近年では、従来の進化的手法による設計の探索ではなく、強化学習などを用いた連続的な探索が注目を集めている. 通常の強化学習タスクは、エージェントの物理的構造は固定されていたが、その構造はタスクに最適ではないことが多い. そこで、Ha は、ロボットの制御パラメータと同時に設計パラメータも強化学習によって最適化した [21]. この方法は、難易度の高いタスクにおいて、ロボットは元の設計よりもはるかに早くタスクを解決することを示した. VSRs の設計を深層強化学習によって最適化するために、Neural Cellular Automata (NCA) [47] の有効性が検討されている. NCA では、各セルがその近傍のセルの状態に基づいて自身の次の状態を決定する Cellular Automaton の、その更新ルールをニューラルネットワークによって学習したものである. この方法により、手作業で設計されたルールに頼ることなく、複雑なパターンの成長や再生が可能になる. Horibe らは、NCA を用いることで、損傷した VSRs から元の設計の一部を再構築し、運動能力を部分的に回復できる可能性を示した [25]. そして、Wang らは NCA を用いた深層強化学習によって、効率的に設計を最適化することができることを示した [79]. しかし、規則的な設計を生成する傾向があり、複雑なタスクには不向きである可能性も示された. その後、彼らは Transformer [75] ニューラルネットワーク構造を応用して、設計と制御の関係性を捉えるように強化学習する手法を提案し、従来の手法よりも優れた性能となったことを実験により確かめた [80].

ほとんどの VSRs 設計の最適化手法は、設計空間が広大であるため、直接的な設計空間での最適化は計算資源を大量に消費し、サンプル効率が低い。そこで、Dong らは、単純で粗い VSRs 設計から最適化を始め、それをもとに漸進的に細かい部分的な設計を考慮していく手法を提案した [16]。VSR の設計を双極空間に埋め込むアプローチにより、設計を特徴に基づいて階層構造で表現することを可能にし、効果的に粗い設計から細かい設計への最適化を果たしている。多数のタスクにおいて、従来の手法よりも、サンプル効率と最終性能の両方で優れていることを実証した。しかし、小さく細かい設計のみが解決できるようなタスクでは、粗い設計から細かくするこのアプローチは有効でないことも示唆された。

これまでの VSRs 設計最適化手法は、人が解釈困難な空間上で最適化されている。設計の複雑な特徴を機械学習によって捉えることで効果的で優れた最適化を果たしているが、その複雑さゆえに、どういった設計の特徴がタスク遂行に寄与しているのかといった具体的な知見を提供し得ない。そのため、人間が設定した設計の特徴量をもとにして最適化を行うことが、VSRs の設計理解につながり、今後の VSRs 研究で重要となる可能性がある。

既知の連続特徴量空間での最適化では、ベイズ最適化 [46] が優れている。ベイズ最適化は、高次元かつ評価コストが高い目的関数を効率的に最適化するための強力な手法であり、その有効性は、探索と活用のバランスを適切に取ることによって、限られた評価回数で最適解に近づける点にある。これにより、実験や評価コストが高い分野、例えば機械学習モデルのハイパーパラメータ調整 [60, 84] や材料科学の分野 [15, 38] で有効であることが示されている。ベイズ最適化によって VSRs の設計最適化をおこなった以前の研究 [5] では、VSRs の離散的でカテゴリカルな設計空間を直接扱った。しかし、ベイズ最適化は連続的なデータ間の相関関係を前提としており、離散的なデータではその前提が崩れるため、有効性が発揮できない。そのため、ここでは遺伝的アルゴリズムや CPPN に比べて、大きく性能が劣った結果となった。

3.2 制御の最適化

VSRs の動作制御に関する初期の研究では、各ボクセルを決められた周期で伸縮させる方法が採用されていた。この手法の利点は、その実装の容易さにある。周期的な伸縮を制御するためのアルゴリズムは比較的単純であり、各ボクセルが独立して動作するため、システム全体の複雑さを抑えることができる。また、伸縮の周期や位相を調整することで、多様な運動パターンを生成することが可能であり、単純な前進運動から複雑な回転運動まで、さまざまな動作を実現することができる。Cheney らは、物性の異なる 2 種類の受動的なボクセルと、それぞれ逆位相の周期的に伸縮する 2 種類のアクチュエータボクセルの合計 4 種類のみを使用して VSR を構築した [11]。ここでは CPPN を使用して、4 種類のボクセルから最適な設計を探索した。さらにその後、彼らはより複雑な VSR の動作を生み出すために、各ボクセルの伸縮周期をパラメータとして CPPN で最適化するアプローチをとった [8, 10]。これにより、より複雑な動作を生み出すことに成功した。この他にも、周期パラメータの最適化に進化戦略手法を採用した研究もある [69]。

しかしながら、固定の周期でボクセルを伸縮させる方法ではいくつかの制約も存在する。ま

ず、各ボクセルの動作が単純な周期的伸縮に限定されるため、制御の自由度が低く、複雑な環境やタスクに対する適応性が限られている。また、全体としてのロボットの動作が各ボクセルの同期性に依存するため、制御の精度や安定性に課題が生じることがある。これを受けて、Talaminiらは、より高度な制御を実現するために、センサーを導入し、センサー情報をニューラルネットワークによって処理するアプローチを提案した [68]。この方法では、各ボクセルにセンサーを組み込み、環境情報やボクセル自体の状態をリアルタイムで収集する。収集されたデータはニューラルネットワークに入力され、ネットワークはその情報を基に最適な伸縮パターンを生成してボクセルに信号を送る。これにより、VSRsは単純な周期的動作から脱却し、環境に応じた柔軟な応答や複雑なタスクの遂行が可能となった。このアプローチは、センサーの多様な情報を活用することで、ロボットが周囲の状況に適応し、より自然で高度な動作を実現するための重要な技術革新である。

ニューラルネットワークを用いたVSRsの制御方法の一つとして、中央集約的に全体のボクセルの動作を1つのニューラルネットワークで制御する方法がある。このアプローチでは、全てのボクセルから得られるセンサー情報を統合し、1つのニューラルネットワークがこれらのデータを処理してロボット全体の動作を一括して制御する。中央集約型の利点は、ロボット全体の動作を統一かつ一貫性を持って調整できる点にある。ニューラルネットワークは全体の状況を把握し、各ボクセルの伸縮パターンを協調的に制御するため、複雑なタスクや高度な動作を実現することができる。この方法により、ロボットは環境の変化や予期せぬ障害に対しても柔軟に対応する能力を持ち、より高度な適応行動を示すことが可能となる。ニューラルネットワークのパラメータの最適化には様々な方法が採用されている。Bhatiaらは、深層強化学習であるPPOによって、環境と相互作用しながら最適化した [5]。また、進化的手法も用いられており [43, 18]、Tankaraらはニューラルネットワークの接続パターンをCPPNを用いて進化的に最適化するHyperNEAT [64]を用いた [70]。

中央集約的なニューラルネットワークを用いたボクセルベースソフトロボット (VSRs) の制御方法は、全体の統一的な動作を実現する一方で、故障や損傷に対して脆弱であるという課題がある。この課題を克服するために、Medvetらは、分散的な制御アプローチを提案し実験した [41]。この方法では、各ボクセルが同じニューラルネットワークを持ち、自律的にセンサー情報を処理して自身の伸縮を制御する。この分散型アプローチの利点は、各ボクセルが個別に動作するため、1つのボクセルの故障が全体の動作に影響を及ぼしにくい点にある。Kvalsundらは、中央集約的な制御方法と比較実験を行い、モジュールを追加した異なる設計に対して分散型の制御の方が優れた適応性を示すことを明らかにした [36]。VSRsのモジュール性を活かして再構成が容易にするため、柔軟で適応力のあるシステムを構築するための有力な方法として近年注目されており、様々な改善がされている。Pigozziらは、各ボクセルがセンサー入力をself-attention [75]によって高度に処理することによって、ボクセル間の通信を省き、より高いモジュール性を実現した [54]。Nadizarらは、NCAを用いて各ボクセルが設計上での役割を自己認識し、その役割にあったニューラルネットワークを用いるという2段階の方式を提案した [49]。これによりさらに、設計の誤差や損傷への頑健性を高めた。

制御におけるアプローチの違いは、設計との共同最適化においても大きな影響を与える。制

御アプローチによって、設計が持つ潜在的な動作性を引き出すことができるかどうかは異なるため、それは最適な設計の探索方策に直接的に関係する。特に、中央集約的な制御アプローチでは、設計が変更されセンサーやアクチュエータの数が異なった場合には、ニューラルネットワークのノード数も必然的に異なるために、新しく再構築する必要がある。一方で、分散的なアプローチでは、ボクセルごとにニューラルネットワークを適用するため、これが解決される。Mertan らは、この2つの制御アプローチを用いて、設計を共同最適化する実験を行い比較した [44]。同じ設計に対して、それぞれ最初から制御を最適化した場合には、動作性能は同等であった。しかし、設計との共同最適化では、分散的アプローチの方が収束が早く、より動作性能も高い結果が得られた。

制御のアプローチに関して、比較や提案を含め、様々な研究がこれまでされてきた。しかし、VSRsにおける制御機構、つまりはニューラルネットワークの最適化方法に関しての比較は十分に行われていない。これまでの研究では、強化学習や進化的手法、どちらか一方がそれぞれで採用されてきた。制御を適切に最適化できるかどうかは、VSRsの適応性、タスクの遂行性に直結している。強化学習では、1つのニューラルネットワークに対して、環境と相互作用しながら勾配降下法によってパラメータを最適化する。一方で、進化的手法は、集団的に広い範囲を探索することで最適化する。最適化手法は、それぞれに異なる特徴を持つため、最適化するタスクやロボットの設計によっても性能が異なることが想定される。

VSRsの制御問題は、リアルタイムに環境を認識し動作を決定することから、連続制御タスクと呼ばれる。他の連続制御タスクでは、制御の最適化手法について、強化学習や進化的手法の比較研究が行われている。Zhang らは、難易度や特性が異なる4つの連続制御タスクに対して、NEATなどの進化的手法と深層強化学習手法を比較した [85]。結果として、タスクによって最適なアルゴリズムが異なることが示された。進化的手法は探索に優れ、初期化やランダム性に対して安定していた。また、強化学習は動的なタスクに強く、データ効率と実行速度で進化的手法を上回る場合が多かった。さらに、強化学習は大規模なネットワークに対してもスケールしやすいが、進化的手法は小さなネットワークの方が良いパフォーマンスを示した。Kovalsky らは、3Dの運転制御シミュレーションタスクで、制御ニューラルネットワークのパラメータの最適化に、進化的手法と強化学習手法のPPOを使用して比較した [34]。進化的手法は訓練時間が非常に短く初期の収束が早かった。強化学習は、より長い訓練時間を必要とするが、最終的な性能は進化的手法を上回る結果であった。Thurler らは、2DのFlappy Birdを制御するタスクに対して、進化的手法であるNEATと深層強化学習手法を比較実験した [72]。ここでは、強化学習手法は安定した学習性能を示したが、進化的手法の方が最終的な性能は高い結果であった。様々なタスクで比較実験されてきたが、タスクによって性能が異なることが示唆される。

3.3 複雑な環境に対するロボット最適化

VSRsは、その柔軟性と適応性を生かして多様なタスクに対応することが期待されているが、その性能評価には単純な環境でのシミュレーションだけでは不十分である。単純な環境

では、ロボットの適応能力や問題解決力が限定的にしか試されず、実際の複雑な環境で直面する多様な課題に対する有効性が評価されにくい。例えば、異なる地形や障害物の存在、予期せぬ外的要因への対応など、現実の環境は多種多様であり、それに対応できるロボットの設計と制御が求められる。したがって、複雑な環境でのシミュレーションを通じて、VSRsが直面する様々なシナリオに対する適応力や耐久性、汎用性を総合的に評価し、最適化することが必要である。このような包括的な評価が行われることで、VSRsはより現実世界での応用可能性が高まり、実際の使用において信頼性の高い性能を発揮することが期待される。

従来の研究では、固定された環境で単純な移動タスクをシミュレーションすることが主流であった。このアプローチは、ロボットの基本的な動作を評価することには適しているが、現実世界で直面する複雑な環境に対する適応能力を十分に評価することができない。この限界を克服するため、近年では砂漠、泥、氷、水場など、様々な環境をシミュレートできるプラットフォームが開発されている [78]。これにより、ロボットは多様な自然環境や厳しい条件下での動作性能を評価され、より実践的な適応力が求められるようになる。例えば、砂漠のような不安定な地形での歩行、泥や氷上での滑りやすい条件への対応、水場での浮力や抵抗を考慮した移動など、これらの複雑なシナリオは、ロボットの設計と制御に新たなチャレンジをもたらす。

現実の環境においては、ロボットが多様な環境に適応し、その中で安定した性能を発揮するための頑健性が極めて重要である。VSRsは特に、柔軟な動作性から様々な環境に適応できる可能性がある。Medvetらは、ボクセルの摩擦の低さが、未知の地面条件における移動タスクに対しての頑健な適応に重要であることを明らかにした [43]。Kimuraらは、直接的に複数の環境に対してVSRsの最適化する実験をおこなった [32]。Talaminiらは、臨界性が高くなるようにVSRsの設計を最適化し、それが多数の環境に適応できることを示した [69]。また、VSRsに限らない広いロボティクスでは、シミュレーションにとどまらず、直接的に現実の多様な環境に適応するロボットについての開発研究も行われている。例えば、多様な種類の地面に対しての頑健な移動性をもつロボット [51] や、水中と陸上のどちらでも移動できる亀ロボット [4] が開発されている。

一方で、環境の複雑性がロボットの形態の最適化に与える影響について研究されている。Auerbachらは、平坦な環境と、段差のある環境や氷の環境のような複雑な環境に対して、ロボットの形態を進化的に最適化する実験をおこなった [2, 3]。その結果、平坦な地形では比較的単純なロボット形態が進化するが、高さや間隔が異なる氷のブロックの上での移動を必要とする複雑な環境では、より複雑な形態が進化する結果であった。つまり、複雑な環境はより複雑な形態を進化させる圧力を提供し、ロボットがその環境で効果的に動くために必要な物理的形態を進化させた。Mirasらは、平坦な環境と傾斜のある環境に対してロボットを最適化し、傾斜のある環境では、より複雑な形態で安定的に動作するような結果を得た [45]。また、Spanellisらは、障害物のある環境の方が、平坦な環境に比べて、ロボットの形態と制御の複雑性を増加させることを示した [62]。

環境の複雑化と協調することで、ロボットの動作制御の複雑性を効果的に高める手法が開発されている。Brantらが提案したMinimal Criteria Coevolution(MCC) [6] は、ロボットの集団と

環境の集団が、それぞれ特定の基本的なタスクや条件を達成することが求められ、この基準をクリアすることで進化する。ロボットが基準を満たすごとに環境の難易度が増していくため、ロボットは常に新たなチャレンジに直面する。WangらはPaired Open-Ended Trailblazer(POET)を提案し、よりロボットの適応性を高めた[76, 77]。POETでは複数のロボットと環境のペアが生成され、それぞれが独自の進化を遂げる。環境はロボットの能力に応じて複雑化し、ロボットはその環境に適応するために進化する。これにより、ロボットは多様な環境での適応力を高めることができる。また、異なるペア間での制御の移転が行われることで、ロボットは他のペアから学び、新たな複雑な環境にも適応しやすくなる。また、Parkerらは、集団的ではなく1つのロボットに集中することで、より計算効率の高い制御の最適化を実現した[52]。このような方法により、ロボットは複雑な環境への適応性を高めるとともに、単一の固定された環境に特化するのではなく、幅広い環境条件に対応するための柔軟性と適応力を獲得することができる。

ロボットの制御だけでなく、環境の複雑化と同時にロボットの形態の最適化を行う研究も近年行われている。Stensbyらは、POETアルゴリズムを拡張し、ロボットの形態のパラメータと制御のパラメータを遺伝子として持つ1つのゲノムによって表現し、遺伝的アルゴリズムによってロボットを最適化した[66]。ここでは、Bipedal Walker[33]と呼ばれる二足歩行ロボットのシミュレーションを利用しており、それぞれの足の幅と長さをパラメータとして扱った。結果として、困難なタスクでも適応性の高いロボットを得ることができている。また、Aoらは、MuJoCo[74]と呼ばれる物理シミュレータ上での歩行タスクで、強化学習によってロボットの学習進捗に合わせて環境の複雑化をおこなった[1]。ここでは、ロボットの形態を最適化するポリシー、制御を最適化するポリシー、環境を複雑化するポリシーの3つを相互作用させるアプローチを提案された。この実験では、ロボットが多様な環境で効果的に動作できる、かつ、環境の変化に迅速に適応することができる形態への最適化が見られた。

VSRsは、より複雑な設計と制御問題を抱えているため、環境の複雑化はロボットの最適化に有効に働く可能性がある。Songらは、Evolution Gymで提供されている多数のベンチマークタスク[5]を利用して、タスク間での設計の有効性についての知見を利用することで、VSRsの設計を効果的に最適化する方法を提案した[61]。簡単なタスクでは、迅速に最適な設計を探索できたが、反面、困難なタスクでは、その1つのタスクに特化した最適化と比較して劣る結果であった。マルチタスクという観点でVSRsの設計最適化がなされているが、1つのタスクでも複雑な環境に対して頑健性高く適応するVSRsの設計という観点での研究は見られない。

第4章 設計特徴量に基づく設計最適化

ソフトロボティクスは、硬質材料に依存する従来のロボティクスの枠を超え、柔軟性、適応性、そして生物の模倣を重視する研究分野として、近年大きな注目を集めている。この分野の進展は、自然界に見られる生物の複雑で柔軟な動きを技術的に再現しようとする試みに根ざしており、特に人間の健康や安全に密接に関わる領域での応用が期待されている。例えば、ソフトロボットは、その独特の柔軟な構造を活かして狭い空間での作業や、人体への直接的な介入など、硬質ロボットでは難しいタスクを実行可能にする。これらの特性は、災害救助、医療、介護など、多岐にわたる分野で革新的なソリューションを提供する可能性を秘めている。

この背景の下、ボクセルベースソフトロボット (VSRs) の研究が注目を集めている。ボクセルとは、三次元空間におけるピクセルのようなもので、VSRs を構成する柔軟な素材の最小単位として機能する。ボクセルを用いることで、ロボットの形状や動きを微細に制御し、従来では実現困難だった高度な機能や動作を実現することが可能になる。このアプローチは、ロボットの設計における柔軟性とカスタマイズ性を飛躍的に向上させる一方で、膨大な設計空間から最適な設計を見つけ出すという新たな課題をもたらしている。

これまで様々な手法で VSRs の設計の最適化にアプローチされてきた。設計は、グリッド状にボクセルを配置することで定義される。遺伝的アルゴリズムは、この離散的な空間を自然に扱うことができるために、最もシンプルで一般的な方法である [5, 53]。また、ロボットのデザインを幾何学的なパターンとして捉え、そのパターンを探索するために CPPN も頻繁に使用される。パターンとして捉えることは効率的なデザインの探索を促し、特に 3D などの広大な設計空間で利用されている [11, 48]。さらに、それまでの進化的な手法に対する近年の傾向として、ニューラルネットワークを用いた勾配的な方法によって設計を最適化する方法も注目を集めている [79, 80, 61]。これによって複雑なデザインの特徴をブラックボックスに捉えることができ、新たなデザインの可能性を探ることが可能である。

しかし、これまでの研究は探索性能に焦点が当てられ、得られた設計を理解することは軽視されがちである。VSRs の設計では、機能性や効率性を最大化するだけでなく、実社会の応用のために設計の理解や解釈性が重要である。従来の手法において、理解を複雑にする原因は、VSRs の設計の自由度の高さにある。従来の VSRs の設計アプローチは、グリッドに個々のボクセルを 1つ1つ丁寧に配置し、ロボットの設計を探索する方法であった。このアプローチは、ロボットの細部にわたる精密な制御を可能にするものの、大域的なデザイン空間の探索には向かない。この結果、設計の可能性は大いに広がるものの、探索して評価できる範囲は極めて部分的となり、VSRs の全体的な設計理解には長い時間がかかることになる。

そこで、通常の剛体ロボットの設計に見られるような、特定の部品を組み立てて全体の構造を構築するアプローチに着想を得た。部品はいくつかのボクセルで構成される塊であり、このアプローチでは、探索空間を部品を組み立てて得られるデザインに絞ることで、より効率的に広い範囲を探索することが期待される。また、細かい最小単位ではなく、設計を部品単位で捉えることが可能となり、それぞれの部品の動作における役割を理解することが可能となる。さらに、ニューラルネットワークのようなブラックボックスな中間表現ではなく、直接的に解釈可能な設計の特徴量を定義し、この特徴量空間で設計を探索する。これにより、どのような設計がタスクの遂行性が高いかを理解することが可能である。

そして、連続空間上での設計最適化にはベイズ最適化 [46] を用いる。ベイズ最適化は、サンプリング効率の高さと、探索と活用のトレードオフのバランス調整という特徴を持ち、計算コストが高い評価関数やノイズを伴う観測結果のような不確実性が存在する問題において有効に働く。実際に、機械学習のハイパーパラメータのチューニング [60, 84] や材料科学分野の新素材探索でも頻繁に利用されている [15, 38]。VSRs の設計問題においても、ベイズ最適化は有効に働くと期待される。しかしながら、VSRs の設計空間はカテゴリカルな表現で直接的にロボットのボクセルのグリッド配置に対応しており、各セル同士の依存関係が大変複雑である。こうした設計空間では、従来のベイズ最適化アプローチを直接適用することには限界があり、効率的な探索性能を発揮するためにはカスタマイズや拡張が必要である。実際に過去の研究 [5] では、進化的手法に比べて劣った結果が出ているが、これはベイズ最適化の潜在的な強みを最大限に活かしきれていないことに起因する可能性がある。この研究では、高度に依存しあった設計空間ではなく、より設計を一般化した連続的な特徴空間を用いる。この空間上でベイズ最適化を実行することは、より多様な特徴をもつ優れた設計の探索につながる可能性があり、さらに、直接的に解釈可能な設計特徴での一貫した探索がなされることで、得られた結果を容易に解釈することが期待される。

本章での研究の最大の貢献は、部品の組み立てに基づいて設計空間を簡素化することで、より直接的な設計の理解と解釈を提供する新しいフレームワークを開発した点にある。提案アプローチは、VSRs の設計過程において直感的な理解と多様な探索を促進することで、ソフトロボティクス分野における今後の研究への道を拓く。さらに、得られた定量的な知見は、VSRs の理解を深め、将来の設計に対する具体的なガイドラインを提供することが期待される。これにより、設計者は、機能性、効率性、および実用性を兼ね備えたソフトロボットをより容易に開発できるようになる。

4.1 VSRs 設計問題の定義

VSRs は、ボクセルと呼ばれる三次元の体積要素を組み合わせることで設計される。このボクセルは、立方体の形状を持ち、個々のボクセルがロボットの構成要素となる。VSR の設計では、これらのボクセルを互いに接続し、特定の配置やパターンに従って組み立てることで、ロボット全体の形状や構造を形成する。各ボクセルは、異なる物理的特性を持つことが可能であり、たとえば弾性や剛性、密度などのパラメータを調整することで、ロボット全体の動

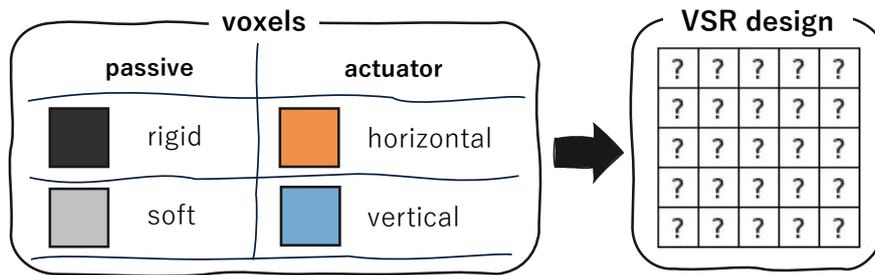


図 4.1: Evolution Gym 上での VSRs 設計の定義.

作特性や環境への適応性を細かく制御できる。ボクセル同士の接続方法や配置は、ロボットの設計において非常に重要であり、これがロボットの柔軟性や適応能力を決定する要素となる。ボクセルをどのように組み合わせるかによって、ロボットが特定のタスクをどれだけ効率的に遂行できるかが大きく左右される。

Evolution Gym では、4 種類のそれぞれ異なる特性を持つボクセルを用いて VSRs を設計する。以前の VSRs のシミュレーションプラットフォームでは、ボクセルの特性パラメータを高度にカスタマイズすることが可能であったが、Evolution Gym ではより簡略化し、あらかじめ決められた特性のボクセルが 4 つ定義されている。図 4.1 左に示すように、それぞれ 2 種類の受動ボクセルと駆動ボクセルがある。受動ボクセルは周囲のボクセルや環境の影響を受けて変形するボクセルであり、高い弾性定数でロボットの骨格のような働きをする rigid ボクセルと、低い弾性定数で柔軟に形状変化する soft ボクセルがある。一方で駆動ボクセルは、信号を与えることで外部から伸縮度合いを制御することができるボクセルであり、横方向に伸縮する horizontal アクチュエータボクセルと、縦方向に伸縮する vertical アクチュエータボクセルがある。駆動ボクセルを適切に外部制御することでロボットの動作を生み出し、多様なタスクへの適応を促すことができる。

4 種類のボクセルを、図 4.1 右のようなグリッド状の空間に配置し組み合わせることで VSRs の設計を定義する。したがって、VSRs 設計の最適化問題は、ボクセルの組合せ最適化問題として位置付けられる。この研究では先行研究 [5, 53, 16] と同様に、 5×5 のグリッド空間を扱うが、探索空間を縮小するために soft ボクセルを除いた 3 つのボクセルのみを使用する。グリッド空間が 5×5 、それぞれのセルが取りうる状態は 3 種類のボクセルか空白であるため、取りうる組み合わせの数は $(3 + 1)^{5 \times 5}$ となる。このうち、ボクセル同士が完全に隣接しておらず複数の塊に分かれてしまうような組み合わせや、駆動ボクセルが含まれず動作することができない設計となる組み合わせは、制約として除外されるが、それでも VSRs の設計空間は膨大である。また、Evolution Gym では、ボクセルの配置のみならず、隣接するボクセル間の接続の有無もそれぞれ設定することができるが、設計空間がさらに複雑になるため、先行研究と同様に、隣接するすべてのボクセルは接続されているものとして扱う。

4.2 提案手法

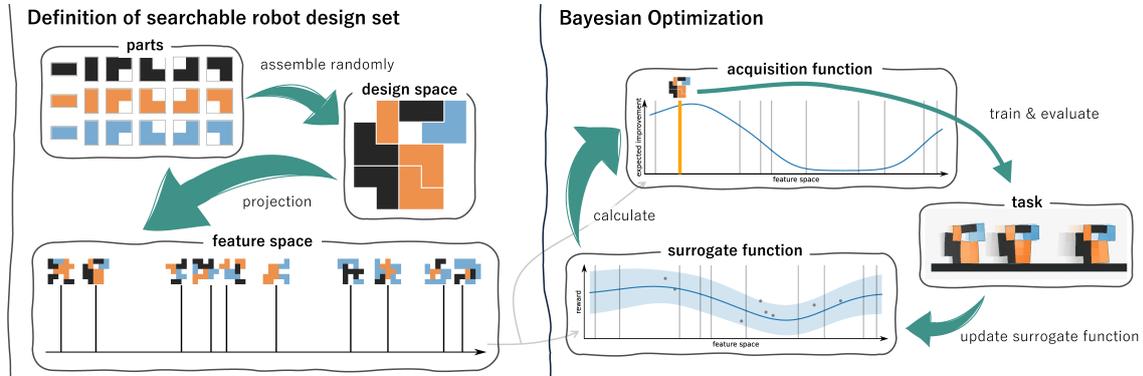


図 4.2: 提案手法の概要. ランダムな部品の組み立てによる探索対象のロボット設計のセットの事前準備と、ベイズ最適化という 2 段階で構成される.

提案手法は、図 4.2 に示すように、次の 2 つの段階で構成される。まず、図の左に示すように、事前準備として探索可能なロボット設計のセットを準備する。ランダムな部品の組み立てによって、重複のない十分な数のロボット設計を生成する。そして、それら全ての設計に対して特徴量を計算することで、設計空間から特徴空間に射影する。特徴空間は表 4.2 で計算されるような特徴のセットで構成される。例えば、図左下の feature space は、vertical アクチュエータボクセルの割合の特徴量の例であるが、実際には 1 次元ではなく、複数の特徴量で定義される多次元上に設計が配置される。次に、図の右に示すように、その特徴空間上でベイズ最適化を実行し、事前準備したロボット設計のセットから優れたものを探索する。ここでは、部品の組み立て方を最適化するのではなく、組み立て済みの設計の中から、解釈性と一般性が高い特徴空間に基づいて最適な設計を探索する。

パーツを組み立てることで定義するロボット設計の探索空間は、十分に大量で重複のないロボット設計を用意している。理想的には、十分な量ではなく、部品の組み立てることで成立する全ての設計、つまり完全な探索空間を用意することが望ましい。完全な探索をするためには、(1) 全ての可能なデザインを網羅的に用意する、(2) 特徴空間上の任意の点から、それに対応するロボット設計（部品の組み立て手順）に逆射影する、という 2 つの方針があり得る。前者は、可能なデザインが膨大すぎるために網羅することは不可能である。後者は、特徴空間とロボット設計が 1 対 1 で対応していないため不可能である。そのためここでは、十分な量のロボットデザインをあらかじめ用意することで、特徴空間を近似的に補完した。

従来手法は、グリッド空間にボクセルを 1 つ 1 つ配置するアプローチであった。遺伝的アルゴリズムでは、直接的に各セルの状態を遺伝子として扱い、CPPN-NEAT などのニューラルネットワークを用いた手法では、ニューラルネットワークによって間接的にボクセルの配置パターンを探索している。提案手法は、これらと異なり、部品単位で設計を探索する。1 つ 1 つのボクセルが持つ設計上での役割は、周りのボクセルの状態に細かく依存するため、複雑

で解釈が難しいが、部品単位で捉えることによってその役割を単純化できる。部品を用いることで、探索の方向性に一貫性を与えることができ、効率的に設計を探索できる可能性がある。また、部品単位で設計を捉えることで、動作特性があまり変化しない細かい設計の調整による探索を抑え、多様な設計を探索できる可能性がある。

そして、グリッド状の設計空間ではなく、特徴空間上で設計を探索する提案手法では、探索の方向性を特徴量に基づいて分析できることが、従来アプローチにはない利点である。従来のアプローチは遺伝的アルゴリズムや、ニューラルネットワークによるブラックボックスな探索であったため、どのような特徴をもつ設計が良いのかといったことを分析・解釈することが困難であった。提案手法では、あらかじめ事前定義した特徴量をもとにして探索するため、直接的に特徴量について解釈することが可能である。しかし探索性能や解釈性は、事前定義する特徴量に依存するため、適切な特徴量について検討を重ねる必要がある。

以下で、提案手法における、探索対象のロボット設計セットの定義と、ベイズ最適化について順に説明する。

4.2.1 部品組み立てによる設計セットの作成

部品を組み立てることで、最適化する際に探索対象となるロボット設計のセットを作成する。この研究では、VSRの設計サイズの制約を 5×5 として、合計で10,000,000の設計を生成する。まず、部品の集合から1つのロボット設計を生成する手順について説明する。その後、十分な量の設計を生成する方法について説明する。最後に、設計に対して計算できる特徴量について説明する。なお、この研究での提案手法による全ての実験では、ここで作成した同一の設計セットを用いる。

設計の生成手順

ロボットを組み立てるために使用する候補の部品を、図4.2左の「parts」枠に定義する。この研究では、生成されるロボット設計の偏りを回避するために、2~3個のボクセルで構成される基本的な形状の部品のみを用いる。タスクに対して、特定の部品が有効であることが事前にわかっているような場合には、特定の部品を利用することも可能である。また、本来使用できるボクセルの種類はRigid, Softと2つのアクチュエータの合計4つであるが、組み合わせの数を抑えるためにSoftボクセルの部品は使用しなかった。

具体的に以下の手順で、部品をランダムに組み立てることで1つの設計を生成する。

1. 組み立てに使用する部品の数を3~8からランダムに選択。
2. 部品集合の中からランダムに使用する部品を選択。
3. ロボットの設計を初期化。
4. 使用する部品を順番に取り出す。

5. 設計と隣接する部品の配置可能な位置をすべて抽出.
6. 部品を配置する位置をランダムに選択し, 設計に統合.
7. 使用する部品がなくなるまで3~6を繰り返す.

まず, 手順1では設計を構成するための部品の数をランダムに選択する. このとき, 5×5 の設計サイズが制約であるため, 使用する数の候補は3~8個とした. また, 使用する部品の数と, 組み立てることができるロボット設計の数の間には, 指数的な関係があることが想定される. 10,000,000個の設計を生成する際に, その指数的な偏りを考慮するために, 以下のように, 組み立てに使用する部品の数 k は, 一様な確率ではなく, 重み付きの確率 w でランダムに選択されるようにする. この重み w は, 10,000,000個の設計を繰り返し生成するとき, 生成状況に合わせて決定される変数となる. これについては, 続く「十分な量の設計の生成」項で詳述する.

$$k \sim \text{choice}([3, 4, 5, 6, 7, 8], w) \quad (4.1)$$

手順2では, 定義した18個の部品集合から, 使用する k 個の部品をランダムな順番で選択する. このとき, 同じ部品は1度までしか使用できないものとした. そして, 手順3では, 組み立てるための設計 R を初期化する. 設計はグリッド状のデータであるが, 初期化の段階では空のデータであり, これに順番に部品を配置していくことで設計が完成する.

手順4~6で, 設計 R に順番に部品を配置する. これは繰り返し処理となっており, 手順2で得られた, 使用する部品がなくなるまで継続する. まず, 手順4では順番に1つずつ配置する部品 p_x を取り出す. そして手順5で, 設計 R にあるいずれかのボクセルに対して, 部品 p_x が隣接し, かつ, 重ならないような位置をすべて抽出する. このとき, 部品 p_x を配置することで, 設計のサイズが 5×5 を超えるものは除外する. その結果, 配置可能な位置が1つもない場合には, 生成が失敗であるとして終了する. 最後に, 手順6で部品 p_x を配置可能な位置からランダムに選択し, 実際に配置し設計 R を更新する. このとき, 設計 R が空である場合には, $R \leftarrow p_x$ として, 部品をそのまま設計に代入し次の部品に進む.

また, VSRはアクチュエータボクセルを駆動することによって動作を形成する. そのため, アクチュエータボクセルが含まれない設計は不適合である. また, 設計に対して構成しているアクチュエータボクセルの割合が少ないものも, 動作性が低いものとして除外する. 手順2で, 使用する部品が決定された際に, 設計におけるアクチュエータボクセルの割合も決定される. そのため, ここで割合が30%未満となる場合には, 手順2をやり直し, 十分な数のアクチュエータボクセルが含まれる設計のみを生成する.

十分な量の設計の生成

次に, 最適化の際に探索対象とする10,000,000個の設計を生成する. 膨大な量ではあるが, 本来の 5×5 の設計空間で可能な設計の数と比べると遥かに少ない. そのため, 10,000,000の設計サンプルは, できるだけ多様な設計で構成されることが望ましい. 例えば, 設計セッ

トが5×5サイズの設計のみで構成されている場合、小さいサイズの設計を探索することができなくなり、タスクによっては最適化が失敗してしまうことが容易に想定できる。設計のサイズは、使用する部品の数に大きく依存するものであるため、10,000,000個の設計を生成するには、使用する部品の数に注意を払う。

k 個の部品をランダムに組み立てることによって、可能なユニーク設計の数を N_k としたとき、 N_k は、部品の組み合わせ方の数であるので、 k に対して指数的な関係性があると考えられる。そのため、 k を一様な等確率で選択した場合、可能な設計の数 N_k に対して相対的に、小さい k で生成した設計の比重が大きくなる。その場合、いわずもがな、大きい k で生成可能な設計を十分に探索することができなくなってしまう。そのため、 N_k に合わせて、使用する部品の数 k の選択される頻度を定めることが求められる。しかしながら、 N_k は組み合わせの数として単純に計算することはできず、部品の形状や設計サイズの制約によって複雑に決定されることから、実際に計算することは困難である。

そこで、10,000,000の設計を生成する中で、 N_k を最尤推定することで、適応的に k の選択確率 w を調整する。まず、前提として、使用する部品の数 k に対して、組み立て可能なサイズ N_k の設計空間は偏りがない一様な分布であると仮定する。つまり、 k が決定された場合、ランダムな組み立てによって生成される設計 R は、 N_k 個から等確率で選ばれるという仮定である。このように仮定することによって、設計の生成を繰り返した際に、 N_k を推定可能となる。

N_k 個からランダムに1つ選択するという試行を M_k 回繰り返し、重複しないユニークなものが U_k 個得られた場合、 M_k と U_k を用いて、以下の3つの要素を用いて、 N_k の尤度 $P(N_k|M_k, U_k)$ を計算できる。

- N_k 個から U_k 個を選ぶ時の組み合わせの数。

$$\binom{N_k}{U_k} = \frac{N_k!}{U_k!(N_k - U_k)!} \quad (4.2)$$

- U_k 個から M_k 回選ぶ際に、 U_k 個の要素がそれぞれ少なくとも1回選ばれるような組み合わせの数。

$$\binom{M_k - 1}{U_k - 1} = \frac{(M_k - 1)!}{(U_k - 1)!(M_k - U_k)!} \quad (4.3)$$

- N_k 個から M_k 回選ぶ際の組み合わせの数。

$$N_k^{M_k} \quad (4.4)$$

すなわち、 N_k の尤度は以下のようなになる。

$$P(N_k|M_k, U_k) = \binom{N_k}{U_k} \binom{M_k - 1}{U_k - 1} \frac{1}{N_k^{M_k}} \quad (4.5)$$

このとき、 $P(N_k|M_k, U_k)$ は、 $U_k = M_k$ のとき、つまり重複した選択がない場合には発散し、 $U_k < M_k - 1$ のとき、つまり1つでも重複した選択がある場合には単峰性となる。従って、 $U_k < M_k - 1$ の場合には N_k を最尤推定することが可能である。しかしながら、式 (4.5) は値が膨大になり計算することが難しい。

N_k は自然数であり、 $P(N_k|M_k, U_k)$ は単峰性であることから、 N と $N + 1$ の尤度を比較することで尤度が最大となる N を見つけることができる。具体的には以下のように、対数尤度を用いて計算を単純化することで $L(N|M_k, U_k)$ を定義する。

$$L(N|M_k, U_k) = \log\left(\frac{P(N+1|M_k, U_k)}{P(N|M_k, U_k)}\right) \quad (4.6)$$

$$= \log(P(N+1|M_k, U_k)) - \log(P(N|M_k, U_k)) \quad (4.7)$$

$$= \log\left(\frac{N+1}{N-U_k-1}\right) + \log\left(\frac{N}{N+1}\right) \times M_k \quad (4.8)$$

この関数 $L(N|M_k, U_k)$ が0以下となる最も小さい $N (\geq M_k)$ が、 N_k の最尤推定値 N'_k である。

各 k における、可能な設計数の推定値 N'_k を用いて、 k が選ばれる確率 w_k を定義する。 N_k は k に対して指数的であると想定されるが、その比率に従って 10,000,000 を分配した場合、小さい k に対応する設計は極少数となる。そこで、対数比率を用いることで分配のバランスを図る。具体的には、以下のように、すでに生成済みのユニークな設計数 U_k に対して、生成可能な設計数がどれだけ残っているかで w_k を計算する。

$$w_k = \frac{\log(N'_k) - \log(U_k)}{\sum_k (\log(N'_k) - \log(U_k))} \quad (4.9)$$

分母は $\sum_k w_k = 1$ とするための正規化である。この w_k を式 (4.1) に利用することで、使用する部品の数 k をバランス良くサンプリングする。

ロボットの設計セットは、以下の手順を繰り返すことで作成される。

1. M_k と U_k を用いて w_k を計算する。
2. k を確率 w_k に基づいてランダムに選択する。
3. k 個の部品をランダムに組み立てることで設計を生成する。
4. 生成された設計が既存のものとは重複していなければ、 $U_k = U_k + 1$ 。
5. $M_k = M_k + 1$

上記手順3において、設計サイズが 5×5 を超えた場合は、生成失敗となるが、この場合には、 U_k 及び M_k は更新せずに手順2に戻る。また、繰り返し初期では $U_k = M_k$ となる。この場合、 N_k を推定することができなくなるため、 $U_k = \min(U_k, M_k - 1)$ として、 N'_k を算出することで w_k を得る。

合計 10,000,000 の重複のないロボット設計を生成した後の、各 k に対する最終的な回数を表 4.1 に示す。 M_k は、使用する k 個の部品を使用して設計をランダムに生成した回数であり、

表 4.1: 10,000,000 の設計を生成した際の、組み立てに使用する部品の数 k ごとの生成試行回数、重複しなかった回数、重複した回数、生成可能な設計数の推定値、生成に失敗した回数.

	k					
	3	4	5	6	7	8
M_k	200,480	1,173,772	2,405,455	3,026,998	2,444,669	1,060,360
U_k	85,179	1,025,981	2,368,922	3,017,584	2,442,395	1,059,939
D_k	115,301	147,791	36,533	9,414	2,274	421
N'_k	97,751	4,261,038	78,388,053	485,651,818	1,313,290,705	1,336,534,455
E_k	0	0	7,563	279,856	1,540,533	3,479,261

U_k はそのうちの重複のない設計の数、 D_k は重複した回数、 N'_k は最終的な N_k の推定値、 E_k は生成途中で設計が $t \times 5$ の制約を超えてしまい失敗となった回数である。 N'_k は k に対して指数的な増加が見られるが、実際に使用する設計数である U_k のバランスはより滑らかとなっている。また、 N'_k が大きいほど、 k が選択される確率は高くなるが、 M_k は $k = 6$ が最大となっている。これは、 k が大きいほど、生成の失敗が発生しやすくなるためである。

また、10,000,000 の簡単な統計量を図 4.3 に示す。(A) は、ロボットを組み立てるのに使用した部品の数 k のヒストグラムである。縦軸は対数スケールとなっており、 k に対する設計数 U_k が滑らかに偏りであることが確認できる。(B) は、ロボットの幾何的なサイズ (横幅、縦幅) のヒストグラムである。設計可能な空間が広大である 5×5 のサイズに、最も多くの設計が分布している。(C) は部品ごとの使用された回数のヒストグラムである。rigid ボクセルの部品は、actuator ボクセルの割合が 30% 以上必要という制約により、採用頻度が低下している。また、 5×5 の制約によって、ボクセル数が 2 と 3 の部品で頻度が異なっている。

設計特徴量の定義

生成したロボット設計は、ボクセルが配置されたグリッド空間上で定義されている。これを、より解釈しやすい特徴量の空間に変換する。最適な設計の探索は、この特徴量に基づいて行われるため、タスクの遂行性に関係していると考えられるものを定義することが望ましい。この研究では表 4.2 に示すように、デザイン全体から算出される global 特徴と、各部品ごとに位置情報で算出される local 特徴を定義した。

global 特徴では、ロボット設計のサイズやアクチュエータボクセルの密度などを計算し、合計 10 次元である。設計のサイズは、狭い空間での移動など大きさが重要な意味を持つタスクの遂行に関係している可能性が高い。また、horizontal(vertical) ratio は、アクチュエータボクセルの割合を表しており、ジャンプや移動など、基本動作の強度に直接的に影響していると考えられる。local 特徴では、部品ごとに、使用されているかどうかと、ロボットの設計中の相対的な位置 (x, y) を計算し、合計 $3 \times 18 = 54$ 次元である。global 特徴とは異なり、設計

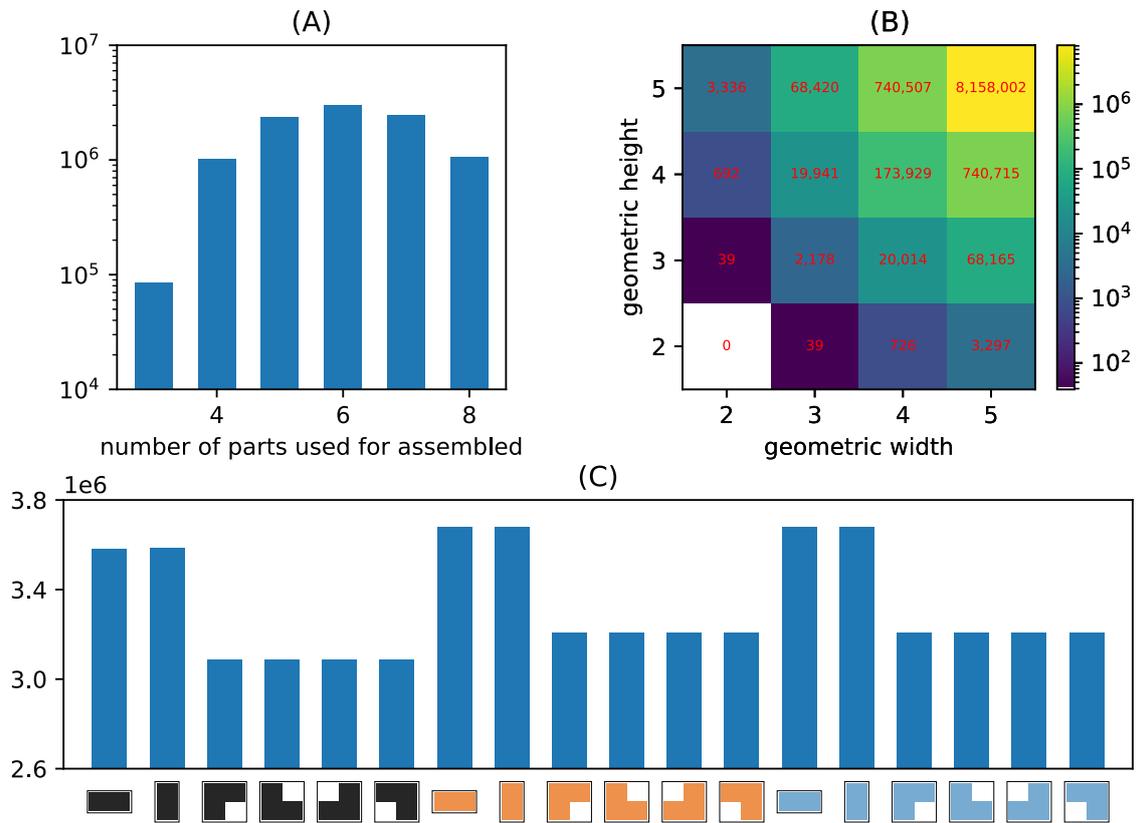


図 4.3: 10,000,000 の設計セットにおける 3 種類の簡単な統計量。

の局所的な特徴を捉えることができる。特に、細かい物体操作を要求するタスクに影響していることが想定される。

4.2.2 バイズ最適化による設計最適化

特徴量空間に配置されたロボットデザインの集合から、特徴量に基づいてバイズ最適化を遂行し、最適なものを探索する。バイズ最適化は、特徴量と目的の評価値の間の不確実性を伴う対応関係を表現する代理関数と、それをもとに高い評価値を得られそうな解を選択する取得関数で構成される [46]。取得関数によって選択した解を実際に評価し、それを受けて代理関数を更新する。これを繰り返すことで最適な解を探索する。2つの関数は、問題設定に応じて適切に選択する必要がある。以下で、この研究での代理関数と取得関数について説明する。

表 4.2: 設計に対して計算される各種特徴量.

feature		formula
global	geometric shape x y	width, height of rectangular frame
	weight	number of voxels
	surface area	outer circumference length
	specific surface area	$\frac{\text{surface area}}{\text{weight}}$
	voxel density	$\frac{\text{number of voxels}}{\text{geometric surface area}}$
	horizontal ratio	$\frac{\text{number of HT actuators}}{\text{number of voxels}}$
	vertical ratio	$\frac{\text{number of VT actuators}}{\text{number of voxels}}$
	center of gravity offset x y	gravity center of robot – geometric center
local*	part used	0 or 1
	relative position x y	gravity center of part – gravity center of robot

代理関数

代理関数は、ベイズ最適化において評価コストの高い目的関数を近似するための重要な要素である。代理関数を使用する主な理由は、目的関数の全域的な振る舞いを効率的に予測し、探索を効率化することである。代理関数には通常、ガウス過程回帰が用いられる。ガウス過程回帰は、観測データから目的関数の分布を学習し、新しい点における関数値の予測分布を提供する [82]。この予測分布は、予測値だけでなく、その不確実性も提供する点が重要である。具体的には、予測値の平均と分散を計算することができ、これにより未評価点での不確実性を明示することが可能となる。不確実性の表現は、探索と活用のバランスを取るために不可欠である。探索は未知の領域を調査する行為であり、不確実性の高い領域において有用である。一方、活用は既知の情報をもとに最適解を絞り込む行為であり、予測値の高い領域に焦点を当てる。このように、代理関数によって不確実性を明示することは、獲得関数が最適な評価点を選定するための基礎となり、結果的に評価回数を抑えつつ高精度な最適解を導くために重要な役割を果たす。

ガウス過程回帰は、代理関数としてベイズ最適化では最も一般的な選択肢であるが、その適用にはいくつかの課題が存在する。まず、ガウス過程は計算コストが高く、特に観測データの数が増えると、逆行列計算により計算時間が急増するため、大規模データセットに対してはスケーラビリティが問題となる。次に、ガウス過程は通常、対象とする関数が連続的かつ滑らかであることを前提としているが、実際の問題では非連続やノイズの多い関数が存在し得る。このような場合、ガウス過程は適切な予測を行うことが難しく、代理関数としての

性能が低下する可能性がある。また、ガウス過程のハイパーパラメータの選定やカーネル関数の選択も難易度が高く、適切なモデルを構築するためには専門知識が必要である。特にこの研究の場合、10,000,000のデータに対して、毎回カーネル関数を計算する必要があるため、非常に相性が悪い。

そこで、この研究ではガウス過程回帰ではなく、ランダムフォレストを代理関数に用いる。ランダムフォレストは、多数の決定木を用いたアンサンブル学習手法であり、各決定木の予測結果を平均することで最終的な予測値を得る [7]。これにより、高い予測精度と堅牢性を確保することができる。特に、ランダムフォレストは非線形関係や高次元データに対しても適応能力が高く、ガウス過程が苦手とする非連続性やノイズの多い関数に対しても有効である。さらに、計算コストの面でもランダムフォレストは有利であり、大規模データセットに対してもスケーラブルであるため、ガウス過程の逆行列計算に伴う計算負荷を回避できることから、この研究でも有効である。さらに、ランダムフォレストは、ハイパーパラメータの調整が比較的容易であり、カーネル関数の選択といった複雑な設定を必要としないため、事前知識の乏しい問題設定に有効である。

ランダムフォレストは、個々の決定木を独立して構築し、それらをアンサンブルとして統合することで構築される。各決定木は、元の訓練データセットからブートストラップサンプリングと呼ばれる手法を用いて、ランダムに抽出されたサブセットを用いて学習される。これは、データの多様性を確保し、過学習を防ぐためである。次に、各ノードの分割においても、全特徴量の中からランダムに選ばれた部分集合を用いる。これにより、各決定木が異なる特徴量に基づいて学習を行うことになり、全体の多様性がさらに増す。各決定木は、特徴量空間のランダムなサブセットを用いて、データを最適に分割するルールを学習し、最終的に葉ノードに達するまでこのプロセスを繰り返す。つまり各葉ノードは、到達したサンプルの集合を表現する平均と分散の値を持っている。新しいデータに対するランダムフォレストの予測は、各決定木においてデータが到達した葉ノードの平均と分散をアンサンブルすることで算出される。

新しいデータに対するランダムフォレストの予測分布の平均 μ と分散 σ^2 は、各決定木 t においての到達ノードにおける平均 μ_t と分散 σ_t^2 で表される分布の混合分布として、以下のよう

$$\mu = \frac{1}{T} \sum_{t=1}^T \mu_t \quad (4.10)$$

$$\sigma^2 = \left(\frac{1}{T} \sum_{t=1}^T \sigma_t^2 + \mu_t^2 \right) - \mu^2 \quad (4.11)$$

これにより、不確実性の推定を行うことが可能となり、探索と活用のバランスを取るための獲得関数を効果的に機能させることができる。

獲得関数

獲得関数は、ベイズ最適化における次の評価点を決定するための基準を提供する重要な要素であり、探索と活用のバランスを効果的に取る役割を果たす。探索とは、新しい領域を調査して未知の最適解を見つけることを意味し、代理関数の不確実性が高い領域に焦点を当てる。一方、活用は既知の有望な領域に対する評価を行い、既存の情報を活かして最適解を絞り込むことを指す。

また、この研究では効率的に多数のロボット設計を評価するために、バッチベイズ最適化を利用する。バッチベイズ最適化は、ベイズ最適化の評価効率をさらに向上させる手法であり、一度に複数の評価点を同時に選択して並行して評価することを可能にする。このアプローチは、特にロボットの設計のような評価コストが高い場合に有効である。バッチベイズ最適化では、通常のベイズ最適化と同様に代理関数と獲得関数を使用するが、複数の評価点を一度に選定するため、一般的に獲得関数の設計には工夫が求められる。ガウス過程を代理関数に使用した単純で有効な方法としては、ガウス過程モデルからバッチ数だけランダムに関数をサンプリングする方法がとられている [30, 13]。

この研究における、代理関数にランダムフォレストを用いたバッチベイズ最適化では、各決定木の混合比率をバッチ数だけランダムにサンプリングすることで、複数の予測分布を得る。式 (4.10) と式 (4.11) では、一様な混合比率を用いて計算していた。このときの混合比率 r を、以下のように一様なディリクレ分布からサンプリングする。

$$\mathbf{r} \sim \text{Dirichlet}(\mathbf{1}) \quad (4.12)$$

このランダムな混合比率 \mathbf{r} を用いると、式 (4.10) と式 (4.11) は以下のように書き換えられる。

$$\mu = \sum_{t=1}^T r_t \mu_t \quad (4.13)$$

$$\sigma^2 = \sum_{t=1}^T r_t (\sigma_t^2 + \mu_t^2) - \mu^2 \quad (4.14)$$

式 (4.12) での混合比率のサンプリングをバッチ数 B だけ繰り返すことで得た、ランダムフォレストによる予測分布 $p(y|x) = \text{Normal}(\mu_b, \sigma_b^2)$ を、それぞれ代理関数とみなして、獲得関数によりバッチ数分だけの次の評価点を得る。

獲得関数には、期待改善量 (EI) を用いる。EI は、現在の最良の結果 y_{best} を基準に、期待される改善量が最大の点を次の評価点として選ぶ手法である。具体的には、以下のように計算される [83]。

$$EI(x) = \int \max(y_{best} - y, 0) p(y|x) dy \quad (4.15)$$

評価点候補の推定値が既存の最良値を超える確率とその超過分の積の積分で計算される。これにより、EI は探索と活用のバランスを自動的に調整することができる。代理関数が予測する不確実性の高い領域 (未知の領域) では、改善の確率が高くなるため、探索を促進する。一

方、既知の有望な領域（既に高い評価を持つ領域）では、改善の量が大きくなるため、活用が進む。この特性により、EI は新しい最適解の発見と既知の最適解の精緻化の両方を効率的に行うことができる。さらに、EI は計算コストが比較的安く、ベイズ最適化全体の計算効率を損なわない点も利点である。10,000,000 の設計セットにおけるベイズ最適化では、未評価の設計全てに対して式 (4.15) を計算し、最大の値をとる設計を次の評価点として選択する。

4.3 実験の詳細

提案手法と従来手法におけるロボット設計の探索性能を比較する実験を行う。Evolution Gym [5] で提供されている、それぞれ性質の異なる以下の 8 つのベンチマークタスクを利用する。

- Climber-v0：左右の壁を利用して垂直に登るタスク。最終的な y 軸の到達点が評価値となる。
- PlatformJumper-v0：異なる高さの足場を飛び越えるタスク。最終的な x 軸の到達点が評価値となる。
- Hurdler-v0：様々な高さのハードルを飛び越えて前に進むタスク。最終的な x 軸の到達点が評価値となる。
- Flipper-v0：反時計回りに回転するタスク。回転した回数が評価値となる。
- Thrower-v0：物体を投げる遠くにタスク。最終的な、初期値から動かずに物体を飛ばした x 軸の到達点が評価値となる。
- Carrier-v1：段差のある足場での運搬タスク。物体を落とさずに、最終的に物体を運んだ x 軸の到達点が評価値となる。
- Lifter-v0：穴に落ちている物体を持ち上げるタスク。最終的な物体の y 軸の到達点が評価値となる。
- BeamSlider-v0：狭い穴に入って上部にある物体を動かすタスク。最終的な物体の x 軸の到達点が評価値となる。

これらのタスクに対して、まず探索性能の比較を行う。比較のための従来手法としては、VSRs の設計最適化においてスタンダードな手法である遺伝的アルゴリズム (GA) と CPPN-NEAT を扱う。また、提案手法におけるベイズ最適化を含めて、全ての手法はランダム性を有しているため実行ごとに結果が異なる。そのため、全ての実験は 3 回ずつ行い評価する。また、提案手法と従来手法を公平に比較するために、以下の条件を従来手法にも適用する。

- Soft ボクセルの探索候補からの除外。

- 5×5 の設計サイズ制約.
- アクチュエータボクセルの構成比が 30% 未満となる設計の除外.
- 合計 1,000 の設計を評価した段階で実行を終了.

また、本研究での主題は、ロボットの設計の探索である。そのため、ロボットの制御を最適化する方法は、全ての設計最適化手法で一致させる。ロボットの制御の学習方法は先行研究 [5] と同様である。ニューラルネットワークを用いてアクチュエータを駆動させ、強化学習手法である PPO によってそのパラメータを学習する。ロボット構造ごとに独立で一定回数学習し、タスクの評価値をフィードバックすることで、優れたロボットの設計を探索する。

以下で、設計最適化手法の具体的な準備と設定を説明する。

4.3.1 提案手法におけるベイズ最適化

前節で作成した、64 次元の特徴空間上にある 10,000,000 のロボット設計の中から、ベイズ最適化によって最適な設計を探索する。バッチサイズを 10 として、合計で 100 回処理を繰り返すことで、合計 1,000 の VSRs 設計を評価する。毎回の繰り返しの度に、既存の観測データと新規の観測データを結合し、代理関数に用いるランダムフォレストモデルを再構築する。代理関数のフィッティング性能は、ベイズ最適化における効果的な探索を行うために重要である。そのため毎回のモデル再構築の際には、ランダムフォレストのハイパーパラメータを 5-fold cross validation により決定する。評価指標には RMSE を用い、表 4.3 に示した候補から最良のものを選択する。

表 4.3: ランダムフォレストのハイパーパラメータ候補

parameter name	parameter space
max_depth	[2, 3, ..., 10]
max_features	[log, sqrt]
min_samples_split	[2, 3, ..., 10]
n_estimators	$2^{[5,6,\dots,9]}$

ベイズ最適化は、既存の観測を活用することで効果的に探索を進めるが、初期状態においては観測データが存在しない。そのため、実行の初期では、完全にランダムに 10 のロボット設計を選択する。

4.3.2 遺伝的アルゴリズム (GA)

遺伝的アルゴリズムは、生物の進化システムに触発されたアルゴリズムで、さまざまな問題に適用することができる手法である。その適用性の高さから、ボクセルベースソフトロボッ

トの最適化にもよく利用されている。5×5の各セルの状態を遺伝子として、世代ごとに個体を突然変異させることで、優れた構造を探索する。各遺伝子は、Rigid ボクセル、Horizontal アクチュエータボクセル、Vertical アクチュエータボクセル、ボクセルなしの4種類のカテゴリカルな変数をとる。人口サイズを30として、世代ごとに親をランダムに10選択し突然変異させることで新しい設計を生成する。新しく生成された設計を評価し既存人口に加え、人口数30となるように適応度の低い設計を取り除く。突然変異では、親となる個体の遺伝子をそれぞれ10%の確率でランダムに変更する。実行初期の集団となる30個体は完全にランダムに作成し、合計1,000個体を評価するまで繰り返す。

4.3.3 CPPN-NEAT

Compositional Perttern Producing Networks(CPPN) [63] は空間的なパターンを生成するために特別に設計されたニューラルネットワークであり、NEAT アルゴリズム [65] によってこれらのネットワークを遺伝的に進化させる。直接的に各セルの状態を指定する遺伝的アルゴリズムと異なり、座標情報から状態を決定するニューラルネットワークを進化させることで、間接的に構造を最適化することができる。VSRsの最適化においても、スタンダードな手法であり、効率的に広大なデザイン空間を探索できる可能性を持つ。本研究では、人口サイズを20として、合計1,000個体評価するまで世代を繰り返す。細かいパラメータについては先行研究 [5] と同様である。

4.4 結果

まず、提案手法 (BO) と従来手法 (GA, CPPN-NEAT) の最適化性能を比較する。次に手法ごとに、探索したロボットデザインの多様性を評価する。最後に、提案手法において代理モデルに用いたランダムフォレストを分析することで、有効な部品や構造的特徴を定量的に評価する。

4.4.1 最適化性能の比較

図4.4に、各タスクごとの、手法による探索の最大の評価値の変化を示す。横軸は評価した累積のロボット設計の数で、アルゴリズムの進行度合いを表す。縦軸はそれまでのロボット設計の最大報酬である。曲線は3回の実行における平均値で、バンドは標準偏差を表す。結果を見ると、タスクによって探索性能が異なることがわかる。例えば Climber-v0 と PlatformJumper-v0 では、CPPN-NEAT と他の2つの間には大きな差がある。他には、Flipper-v0 や Thrower-v0 では、BO, GA が CPPN-NEAT より大きく優れた結果を出した。探索性能という観点では、BO と GA が得意なタスクの傾向は似ていると読み取れるが、CPPN-NEAT は全く異なる傾向を持つことがわかる。BO と GA は最終的な到達性能は近いしいものであるが、GA の方が優れたタスクが多い。

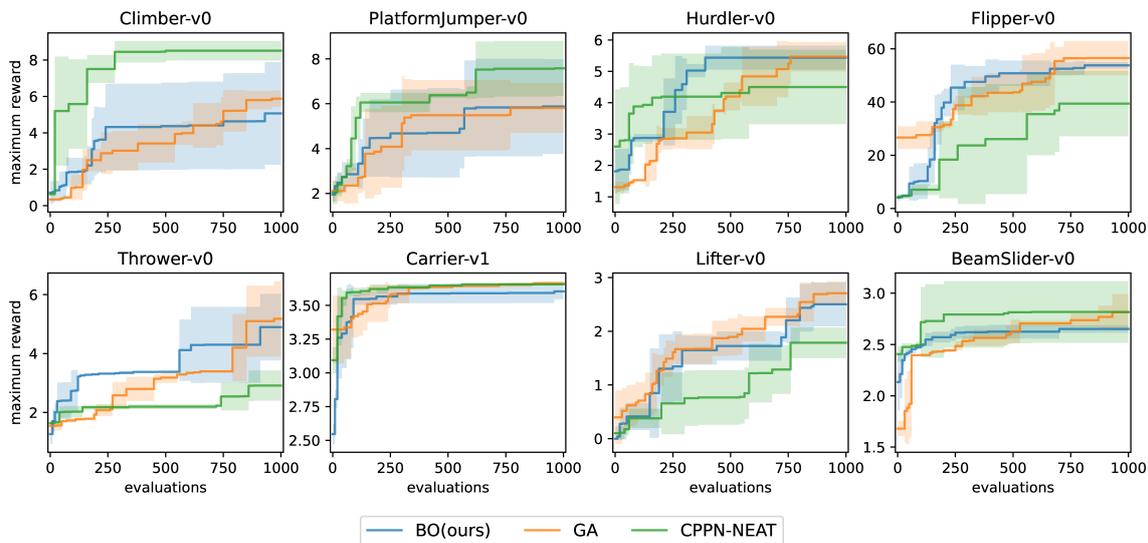


図 4.4: 各最適化手法の最適化過程の比較.

優れた設計を早く見つけられるか、という探索の効率性という点でも、同様にタスクによって異なる傾向が見られる。Hurdler-v0では、BOの曲線の傾きが急であり、早く優れた設計を得られている。一方、Carrier-v1では、CPPN-NEATが最も早く優れた設計に到達している。最終的な到達性能の傾向が似ているBOとGAについて、効率性においては、BOが優れているものとGAが優れているタスクに異なる傾向がある。

また、3回の試行のうち最大の性能を得た試行を取り出し、図4.5にその試行におけるrewardの高い8のロボット設計を並べた。まず全てのタスクで俯瞰して全体を見ると、CPPN-NEATの結果に大きな特徴がある。例えば、Climber-v0やPlatformJumper-v0では、CPPN-NEATで得られる優れた設計は、 5×5 全てのボクセルが詰まっているような、至極単純な形状であるものが多い。他のタスクでも、他の2つの手法に比べて、同じ種類のボクセルがかたまっているような単純な設計が多い。これに対して、BOとGAは複数種類のボクセルが複雑に配置された設計を探索している。GAに関して、それぞれのタスクにおける一回の探索結果である8つの設計を見ると、似ているボクセル構成が多い。GAは、優れた設計をもとに、突然変異によって一部のボクセルを変更することによって、次の設計を探索する。そのため、このような結果となっている。対して、BOでは、より多様な設計を得ている傾向がある。総じて、手法ごとに探索の方策が全く異なることが読み取れる。

提案手法によるBOは、探索性能ではGAに対して今一步劣る結果であった。これは、部品によって組み立て可能な設計からの探索であるため、細かい設計の調整をすることができないためである。この傾向は特に、BeamSlider-v0の結果に顕著に表れている。BeamSlider-v0では狭い通路の中を進む必要があり、細長い形状でないとタスクの遂行が難しい。そのため、提案手法のBOでは、性能の改善が停滞する結果となった。

また、GA、CPPN-NEATとBOを比較した先行研究[5]では、BOの性能は乏しかった。そ



図 4.5: 最適化手法ごとの、適応度の高い 8 つの設計。

の原因は、そこで述べられているように、カテゴリカルな設計空間上での BO が問題であったと思われる。今回の提案手法による BO では、設計空間ではなく、連続的な特徴空間上で探索をおこなったことが有効に働き、他手法に競合する結果を得ることができた。

4.4.2 探索の多様性の比較

次に、設計性能の高さ設計多様性の関係を考察する。図 4.6 に、探索で得られた 1,000 のロボットデザインを性能が高い順に並べ、左から累積したときの、評価値の平均の変化を示す。曲線は 3 回の実行における平均で、バンドは標準偏差を表す。横軸について、左端は評価値の高い 10 の設計、右端は探索した 1,000 全ての設計のリストとして、その中での平均評価値を表している。

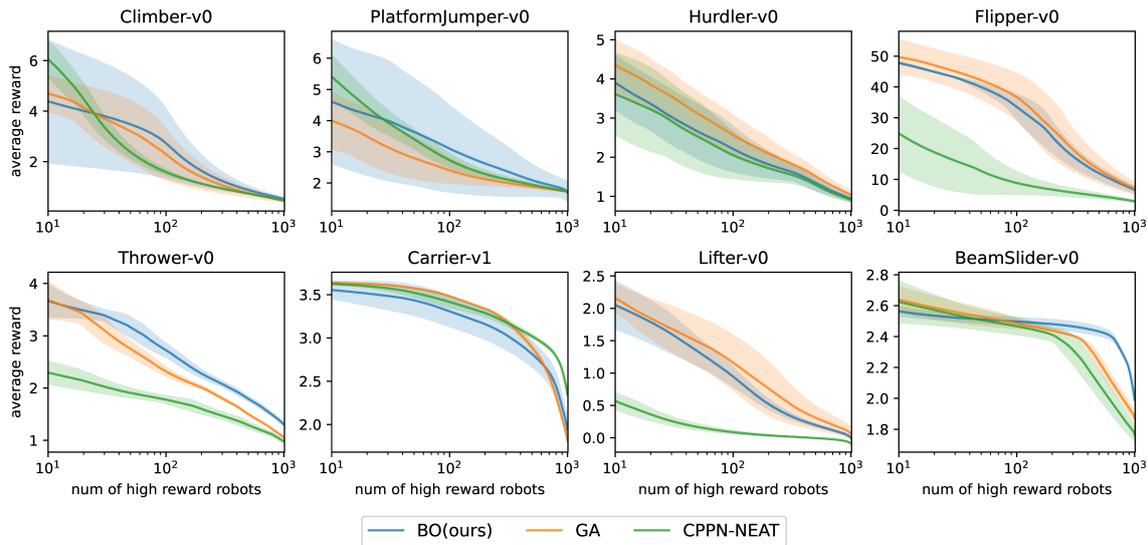


図 4.6: 最適化手法ごとの、探索した設計の評価値の分布。

図 4.6 からは、性能の高い性能をどれだけ探索できているかを読み取ることができる。横方向の傾斜が緩やかであるほど評価値の差が小さく、急であれば差が大きいことを表している。例えば、Climber-v0 の CPPN-NEAT では、左端の平均 reward は最も高い。しかし、傾斜が急であるため、高い評価値を得た設計は少ないことがわかる。また、Thrower-v0 では、BO の方が GA よりも傾斜が緩やかであり、多数の優れたデザインを探索できている。特に BeamSlider-v0 の結果は顕著であり、右端を見ると、BO が右上に張っている。つまり、より高い性能の設計を探索することはできていないが、ある一定の性能の設計を他の手法より多数探索していることを表している。

また、図 4.7 に多様性の変化を示す。横軸は図 4.6 と同様であり、左端は評価値の高い 10 の設計、右端は探索した 1,000 全ての設計のリストとして、その中での多様性を表している。多様性は、対象となるロボット設計リストにおける全ペアの編集距離の平均で定義する。編集距離は、ペアに対して、一方の設計を何回編集したらもう一方の設計と同じになるかで計算される。Song らの研究では、 5×5 のグリッド状のデータを行列として直接的に差を計算していた [61] が、本研究ではサイズを自由に拡張することができる仮定での編集回数として、厳密に計算する。例えば、ある設計を 5×5 のデータ上で、横方向にスライドしたような設計

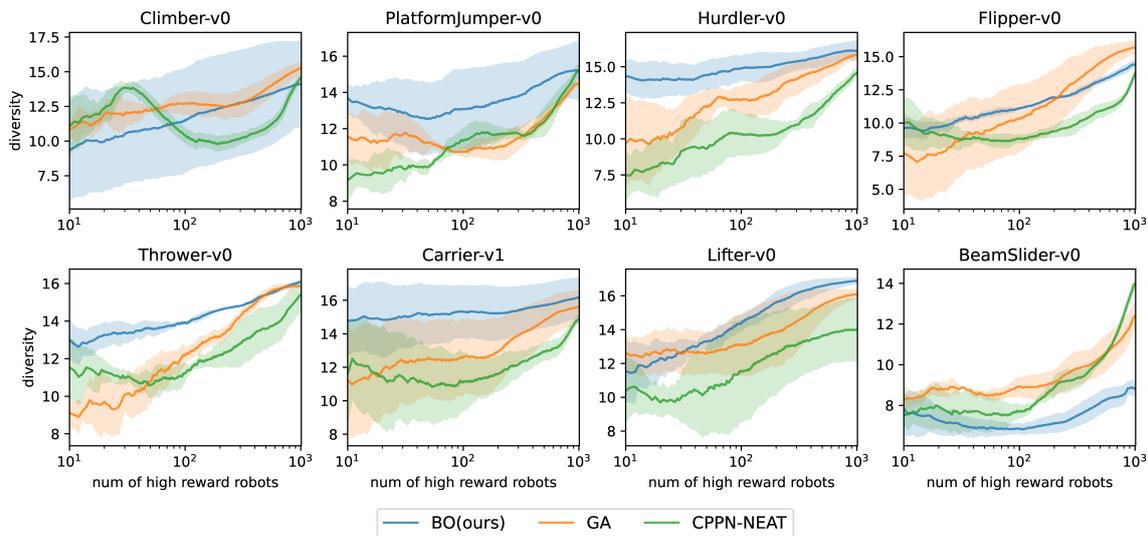


図 4.7: 最適化手法ごとの、評価値の高さと多様性の関係性.

は、スライド前と後で同じ設計となる。厳密な編集距離を計算することで、正確に異なる設計であるかを測ることができる。ボクセルのないセルへのボクセルの追加を含めて、1つのボクセルの変更には、全て一回の編集として編集距離を計算する。

図 4.7 から、多様で優れた設計を探索できているかを読み取ることができる。探索した 1,000 全ての設計における多様性（右端）は、手法によって大きな差はないタスクが多い。高い評価値の設計に絞り込むと、手法ごとに異なる傾向が現れる。基本的には、左に寄るほど多様性は低くなる傾向である。しかし、GA は評価値が高い設計をもとにして、局所的な領域を探索する傾向があるために、BO に比べて傾斜が急で多様性が低い傾向がある。ほとんどのタスクで BO の曲線は、他の手法より高い位置にあり、多様な設計を探索できていると言える。しかし、BeamSlider-v0 においては、図 4.5 で見られるように狭所に適したデザインが求められる。つまり、局所的な探索が不得意な BO は多様性が低い結果であった。

4.4.3 設計特徴量の分析

Random Forest の特徴重要度

提案手法での BO で代理関数として使用したランダムフォレストを分析する。図 4.8 に 3 回の実行分の、1,000 のロボット設計における g と評価値で構築したランダムフォレストにおける特徴重要度を箱髭図で示す特徴重要度は、特徴ごとの決定木における分岐への貢献度を表している [39]。特徴量は合計 64 次元であるが、簡単な解釈のために、 (x, y) や、同じパーツについての重要度は足し合わせて 1 つに集約した。つまり、global 特徴が 8 次元、local 特徴は部品の数である 18 だけある。図 4.8 では、local 特徴である部品については、部品のボクセ

ル種類に対応した色付けをした。

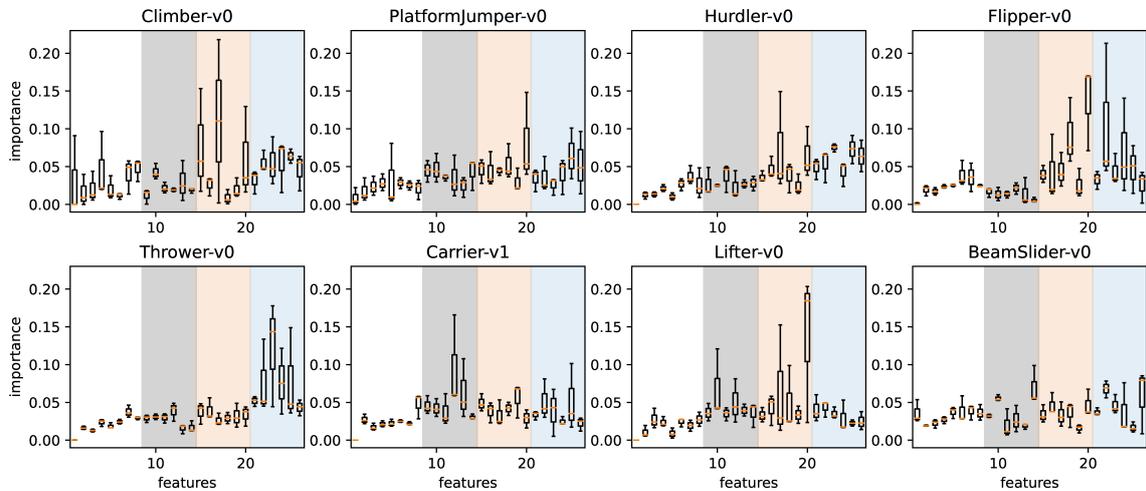


図 4.8: ランダムフォレストによる、評価値に対する設計特徴の重要度。

ほとんどのタスクで、アクチュエータボクセルの部品が高い重要度である。当然、アクチュエータボクセルがなければロボットの動作を生み出すことができないためである。しかしながら、左から 6,7 番目の特徴は、設計における 2 つのアクチュエータボクセルの構成比率をそれぞれ表しているが、特段高い重要度ではない。つまり、アクチュエータボクセルの密度が重要なのではなく、特定の形状のアクチュエータ部品を、ロボット設計のどこに配置するか、という情報がタスク遂行に大きな影響を持っているということである。また、Carrier-v1 と BeamSlider-v0 では、Rigid ボクセルの部品の重要度が高く、動作性よりも安定性が求められるということが言える。

global 特徴と local 特徴では、local 特徴の方が総じて高い重要度となっている。つまり、全体的な形状の特徴などはタスクの遂行性に大きく影響しておらず、部品で表される局所的な形状が重要であることを示している。Climber-v0 と BeamSlider-v0 においては、1 目特徴である設計のサイズ特徴が比較的高い重要となっている。実際に、Climber-v0 では左右にある壁を利用するために設計の横幅が重要であり、また、BeamSlider-v0 では、狭い通路を進むために小さいサイズである必要がある。そのため、この特徴を利用した探索が行われた。しかしながら、Climber-v0 における、ある実行では、ジャンプによる y 軸の高さで評価値を得るような、局所的な領域の設計に探索が偏ってしまった。他の実行では、うまく壁を利用して登る設計を得ることができ、評価値の高い設計を探索することができている。こうした探索の不安定性が、図 4.6 に見られるような分散の高さに繋がっている。また、BeamSlider-v0 においても、図 4.5 の GA や CPPN-NEAT の結果にみられるような設計のように、設計サイズが小さいのではなく、細長いような形状が、より高い評価値に必要であることがわかる。BO では、設計サイズの特徴によって、探索が困難となっているため、これを特徴量から除外することで探索性能を改善できる可能性がある。

Partial Dependence Plot

次に、重要度が高いパーツが、具体的にロボットデザイン上のどこに配置されているとよいかを分析する。実行ごとに結果が異なるため、3回の実行のうち、最大の性能の設計を得た実行における結果を用いる。図 4.9 は、最も重要度が高いパーツの位置情報についての Partial Dependence Plot(PDP)である。PDP は、ある入力特徴について、入力する値を変化させたときの予測値の変化の傾向を調べる手法である [19]。データセットの各レコードごとに、対象の特徴を変化させた全パターンでの予測値を集約している。ここでは、対象の部品の、ロボットの重心からの相対的な位置 (x, y) について調べている。ロボットの重心 $(0, 0)$ を赤い丸で示しており、これを基準として色の明るい領域に部品を配置するように設計を組み立てると、ランダムフォレストの予測値が高くなることを表している。

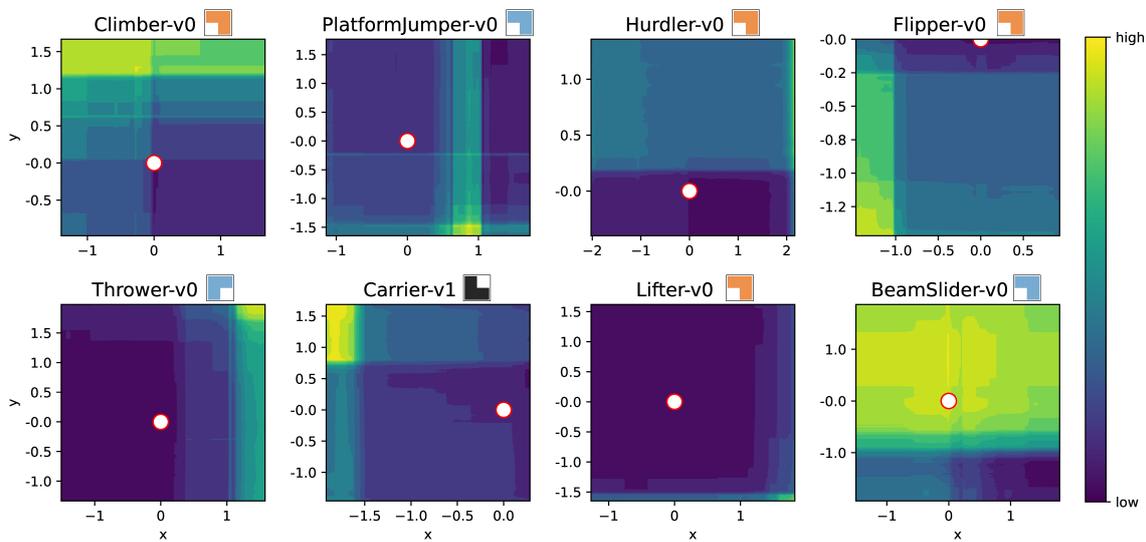


図 4.9: 2次元の Partial Dependence Plot による、重要な部品の配置位置の可視化。

図 4.9 に示した、部品の位置情報による PDP によって、タスクと部品の関係を解釈することが可能である。例えば、Flipper-v0 では、設計の左下に該当のパーツがあることによって、重心を左に傾けやすくなり、回転タスクをこなすことができる。また、Thrower-v0 では、右上の物体との接触点に部品を配置することで、バネのように右側に放物線上で物体を投げることができる。Lifter-v0 では、右下に Horizontal アクチュエータの部品を配置することで、下にある物体を挟むようにして持ち上げるために重要な役割を果たしている。Carrier-v1 では、頭上の物体を支えながら右に移動することに貢献している。また、図 4.9 の結果は図 4.5 で示したロボット設計に対応しており、部品の実際の位置関係を確かめることができる。

4.5 まとめ

本章では、部品を組み合わせることで生成することができるロボット設計に制約して最適な設計を探索するアプローチを提案した。部品を組み合わせることで、細やかな最適化を行うことなく、広く多様にデザインを探索することができる。また、グリッド上のカテゴリカルな設計空間ではなく、設計全体の特徴や部品の位置情報などの連続した特徴空間に基づいた探索を行った。その結果を分岐することで、従来手法では得られなかった、より具体的で定量的な解釈を提供した。

1つの最適解を探索する性能では、提案手法は従来と比較して特別優れてはいなかった。これは、設計の局所的な調整ができないためであるが、反面、多様な次善の解の探索に繋がった。提案手法の探索で得られる多様な解をもとにして、局所的な探索に優れるGAを適用することで、さらに良い設計を効果的に発見できる可能性がある。また、今回定義した設計の特徴量は十分な検討をしていない。実際に、設計サイズの特徴量が局所的な領域への収束を引き起こしており、性能改善の停滞に繋がった。使用する特徴量は探索性能と密接な関係があるため、深く検討することで、探索性能が大きく改善されることが期待される。

また、この実験では一般的な探索性能を評価するために、基本的な形状の部品を用いた。しかし、タスクごとに特定の部品が有効であるというような事前知識がある場合には、これを直接的に活かすことが可能である。実際にその有効な可能性のある部品を候補部品に取り込み、探索する設計セットの作成と特徴量定義することで、実験により、定量的にその部品の有効性を評価することも可能である。

今回の実験では、簡易的にVSRsを実験できるEvolution Gymを利用した。そのため、サイズの強い制約が課された。こうした小さいサイズの空間では、局所的な調整が高い重要性を持つ。3Dやより大きなサイズでは、探索空間がより広大であるため、提案アプローチがより有効に働く可能性がある。コンピュータシミュレーションだけでなく、現実での応用を見据えるために、より現実に近い空間での実験が不可欠である。

第5章 制御の最適化アルゴリズムの比較

ボクセルベースソフトロボット (VSRs) は、ロボットを構成するボクセルを伸縮させることによって、全体の動作を形成する。多数のボクセルの動きをそれぞれ協調させる制御をすることで、複雑な環境への適応やタスクの遂行が可能となる。しかしながら、細かい1つ1つのボクセルをどのように協調させることで適切な動作を生み出すかは非常に複雑な問題であるため、困難な課題となっている。環境や状況に合わせて適切に VSRs を動作することは難しく、あらかじめ動作の制御をプログラムによって記述することは不可能である。そのため、ロボットのセンサーによる環境や状況の観測情報を、ニューラルネットワークによって処理し、各ボクセルの動きをリアルタイムに制御するための研究が盛んに行われている。

適切に VSRs の動作を制御するようにニューラルネットワークのパラメータを最適化する手法として、従来の研究では、進化的手法や強化学習によるアプローチが採られてきた。進化的手法は、ニューラルネットワークのパラメータを1つの個体の遺伝子表現として、進化的に最適なパラメータを探る。各個体は、一定回数の制御シミュレーションによって評価され、評価値が高い個体をもとにして、より評価値の高いパラメータに最適化される。強化学習では、1つのニューラルネットワークに対して最適化が進む。シミュレーションにおける各時間ステップごとの評価値をフィードバックすることで、評価値が高くなるような動作をするように、ニューラルネットワークのパラメータを勾配降下法によって最適化される。

従来の VSRs の研究では、制御の最適化には進化的手法あるいは強化学習の、どちらか一方のみが利用されてきた。しかしながら、VSRs の制御問題は複雑であり、VSRs が潜在的に持つ柔軟で多様な動作性を十分に引き出すことは困難である。つまり、最適化手法によっては、引き出すことができる動作が異なることが想定される。また、タスクによっても、評価値が高くなるような動作を発見するための難易度や特性が異なるため、最適化手法によって結果が異なる。

VSRs ではないが、他の連続的なシミュレーションにおけるロボットの制御最適化問題においては、進化的手法と強化学習が比較検討されている。難易度や特性の異なる4つの連続制御タスクに対する、NEAT と深層強化学習手法の比較では、強化学習が学習の効率性が高く、進化的手法は探索に優れており安定して制御を獲得している結果であった [85]。反対に、3次元シミュレーションの自動車運転制御タスクでは、進化的手法の方が収束までの時間が短く、強化学習の方が良い最適化性能を示した [34]。エージェントを上下に動かすだけの、より単純な2次元のゲームタスクでは、進化的手法である NEAT アルゴリズムの方が、強化学習よりも高い性能が得られている [72]。つまり、タスクによって進化的手法と強化学習の性能が異なるということであり、VSRs の制御問題でのこれらの比較は必要である。

また、VSRsの制御問題は設計問題にも大きく関係がある。制御の最適化手法によって、得られる結果が異なるとすると、ある設計の評価値も異なることになる。そのため、優れた設計をもとに新たな設計を探索していく設計最適化でも、探索する設計の方向性が変化する。したがって、制御の最適化は、設計の最適化に密接に関わっており、制御の最適化について検討することは、今後のVSRs研究において重要な意義を持つ。

そこで本章では、VSRsの制御最適化問題における、進化的手法と強化学習を比較する。実験には、Evolution Gymで提供されている、8つのそれぞれ異なるタスクを利用する。これによって、タスクごとの制御最適化手法の傾向の差を検討する。さらに、それぞれのタスクに対して、複数のVSR設計を用意し、設計によっても制御最適化の傾向が異なることを確認する。これによって、設計の最適化に与える影響についても考察する。

5.1 VSRs制御の問題定義

VSRsの柔軟で適応的な動作を実現するためには、制御の最適化が極めて重要である。VSRsは、その構造が柔軟で変形可能であるため、さまざまなタスクや環境に対応するためには、複雑で精緻な制御戦略が必要となる。複雑な地形での歩行や物体操作、障害物の回避などの困難なタスクにおいては、VSRが適切な動作を学習し、状況に応じて柔軟に対応できる能力が求められる。そのため、制御の最適化手法は、VSRが環境やタスクの変化に迅速に適応できるように設計される必要がある。さらに、これらの制御の最適化手法は、シミュレーションを通じて多様なタスクで評価される必要があり、異なるタスク間での汎用性とロバスト性を確保するために、幅広い環境でのシミュレーションが不可欠である。

Evolution Gymでは、VSRsの複雑な動作性を評価するために、単純な移動タスクから複雑な物体操作タスクまで、多様なベンチマークタスクが提供されている。各タスクの環境は、VSRsが移動する地形や操作する物体などで構成されており、これらはすべてボクセルによって表現されている。地形は100マス程度の横幅のグリッドで定義され、rigidボクセルで構成されている。タスクによっては物体も環境上に配置されており、同様にボクセルで構成されている。多様な環境条件とタスクでVSRsの動作をシミュレーションすることが可能である。また、あるタスクの達成度は、一定回数シミュレーションステップを繰り返すことによる、最終的な結果によって評価される。VSRsは、このシミュレーションの毎ステップで、センサーによる環境情報を受け取り、その都度都度で適切な次の動作を決定する。これによって、環境や自分自身の状態に合わせて適応的に複雑な動作を生み出すことが可能となる。

Evolution Gymでは、水平アクチュエータボクセルと垂直アクチュエータボクセルの2種類が提供されており、VSRを構成しているこの2種類のボクセルを協調して伸縮させることで、VSRの動作を生み出す。VSRsには、「歩く」や「掴む」などの具体的な動作が定義されてはおらず、VSRsを構成する各アクチュエータボクセルをそれぞれ適切に動かすことで、時間的に複雑な動作を形成する。それぞれのアクチュエータの伸縮が、どのような全体の動作につながるかを予測することが難しいため、VSRs制御は複雑系の一種である。また、2つのアクチュエータボクセルは、 $[0.6, 1.6]$ の比率範囲で伸縮が定義されており、この範囲内の連続値

信号をそれぞれのアクチュエータに送ることで、実際に伸縮させる。つまり、VSRs 制御は連続値制御問題として位置づけられる。

こうした VSRs の適応的で複雑な制御には、ニューラルネットワークが用いられる。ニューラルネットワークの入力は、各シミュレーションステップで取得されるセンサー情報、出力は VSR を構成している各アクチュエータボクセルそれぞれの伸縮度合いとなる。 i 個目のアクチュエータボクセルの伸縮信号 y_i は $[0.6, 1.6]$ の範囲であるため、 i 番目の出力ノードの値 x_i は、シグモイド関数を利用した以下の式で変換される。

$$y_i = \frac{1}{1 + e^{-x_i}} + 0.6 \quad (5.1)$$

以下で、この研究で用いる 8 つのベンチマークタスクと、設定されているセンサーについて詳細に述べる。

5.1.1 使用するベンチマークタスク

この研究では、Evolution Gym で提供されている、それぞれ性質の異なる 8 つのベンチマークタスクを利用する。タスクは、それぞれ固定のステップ数が設定されており、そのステップ数だけシミュレーションを進行した最終時点での結果で報酬が計算される。以下でそれぞれタスクの内容や報酬計算について説明する。

- Walker-v0：平坦な地形の上を歩くタスク。水平に 100 マスの rigid ボクセルが並んだ環境で、1 つのボクセルの幅を 0.1 とし、ロボットが正の x 方向に移動した分だけ報酬が与えられる。右端に到達すると追加で 1 の報酬が与えられるため、報酬のおよその理想値は左端から右端までの距離に足し合わせた 11 である。ただし移動距離は、ロボットの重心の初期位置で計算されるため、実際にはこれより小さい値となる。最大 500 ステップのシミュレーションで評価され、右端に到達した時点で終了する。
- Carrier-v0：物体を遠くに運ぶタスク。水平に 100 マスの rigid ボクセルが並んだ環境で、1 つのボクセルの幅を 0.1 とし、ロボットと物体が正の x 方向に移動した分だけ報酬が与えられる。右端に到達すると追加で 1 の報酬が与えられるため、報酬のおよその理想値は左端から右端までの距離に足し合わせた 11 である。ただし移動距離は、ロボットの重心の初期位置で計算されるため、実際にはこれより小さい値となる。物体を地面に落とすとペナルティが与えられる。最大 500 ステップのシミュレーションで評価され、右端に到達した時点で終了する。
- Thrower-v0：壁を超すように物体を遠くに投げるタスク。横に 70 マスの環境で、1 つのボクセルの幅を 0.1 とし、物体が正の x 方向に移動した分だけ報酬が与えられる。報酬のおよその理想値は左端から右端までの距離の 7 である。物体を投げるときにロボットが初期位置から x 軸上で変化した距離の $\frac{1}{4}$ だけペナルティが与えられる。300 ステップのシミュレーションで評価される。

- **Climber-v0**: 上に伸びる平らな管を垂直に高く登るタスク。縦に 90 マスの環境で、1 つのボクセルの高さを 0.1 とし、ロボットが正の y 方向に移動した分だけ報酬が与えられる、上端に到達すると追加で 1 の報酬が与えられるため、報酬のおよその理想値は下端から上端までの距離に足し合わせた 10 である。ただし移動距離は、ロボットの重心の初期位置で計算されるため、実際には 10 より小さい値となる.. 400 ステップのシミュレーションで評価され、上端に到達した時点で終了する。
- **ObstacleTraverser-v0**: 凹凸のある地形を横切って進むタスク。横に 80 マスの環境で、1 つのボクセルの幅を 0.1 とし、ロボットが正の x 方向に移動した分だけ報酬が与えられる。右端に到達すると追加で 2 の報酬が与えられるため、報酬のおよその理想値は左端から右端までの距離に足し合わせた 10 である。ただし移動距離は、ロボットの重心の初期位置で計算されるため、実際にはこれより小さい値となる。ロボットが左右に 90 度以上回転すると 3 のペナルティが与えられる。1,000 ステップのシミュレーションで評価され、右端に到達した時点で終了する。
- **PlatformJumper-v0**: 間隔をあけて配置された高さの異なる複数の平坦な足場を横切って進むタスク。横に 90 マスの環境で、1 つのボクセルの幅を 0.1 とし、ロボットが正の x 方向に移動した分だけ報酬が与えられる。報酬のおよその理想値は左端から右端までの距離の 9 である。ただし移動距離は、ロボットの重心の初期位置で計算されるため、実際にはこれより小さい値となる。ロボットが左右に 90 度以上回転したり、足場から落ちたりすると、3 のペナルティが与えられる。1,000 ステップのシミュレーションで評価される。
- **AreaMaximizer-v0**: 可能な限り地面との接地面積を最大化するタスク。水平な環境上で、1 つのボクセル幅を 1 とし、ロボットの初期状態から増加した接地面積だけ報酬が与えられる。ロボットの設計によって、初期の接地面積や、形状を変化させたときの可能な理想的接地面積が異なるため、報酬の理想値は異なる。600 ステップのシミュレーションで評価される。
- **Flipper-v0**: 平坦な地形で反時計回りに回転するタスク。水平に 70 マスの rigid ボクセルが並んだ環境で、右端から左端に反時計回りに回転した分だけ報酬が与えられる。報酬はラジアン単位で計算される。回転回数に上限はないため、報酬の上限値はない。600 ステップのシミュレーションで評価される、左端に到達した時点で終了する。

移動や物体操作を行うタスクでは、移動距離で報酬が計算されるため、環境のサイズがおおよその報酬上限となる。しかし一部のタスクの報酬は、外的な要因による制限はなくロボットの設計が持つ潜在的な特性に依存するため、明示的な報酬の上限は存在しない。また、タスクごとに理想とする動作も与えられておらず、よりよい評価値を得るように動作をするための制御ニューラルネットワークを得ることが求められる。

5.1.2 ロボットの観測値

Evolution Gym の各タスクごとに、ロボットが観測する情報が定義されている。タスクをシミュレーションする中の各タイムステップで、ロボットはセンサーから環境情報を受け取る。センサーは大きく3種類あり、ロボットの状態、環境の地形、操作する対象物の状態についての観測が得られる。以下でそれぞれ詳細に説明する。

- ロボットの状態
 - 位置：ロボットの構造や形状の変化を把握するための情報。ロボットが含んでいるグリッド状のすべての頂点に対して2次元の座標が取得される。すべての頂点の平均をとることでロボットの重心の座標を算出し各頂点の、重心からの相対的な座標を計算し、これが観測値となる。この観測情報は、ロボットが持つ頂点の数によって次元数が異なり、頂点数×2の次元で表される。例えば5×5のグリッドすべてにボクセルが配置されているようなロボット設計では、頂点数は36となり、ロボットの位置情報に関する観測は72次元となる。1つのボクセルの幅を0.1とした、位置情報の値尺度である。
 - 速度：ロボットがどのように動いているかを把握するための情報。ロボットのすべての頂点を持つ2次元上の速度が取得される。すべての頂点の平均を取ることによって、ロボット全体の速度として2次元で観測値が構成される。1つのボクセルの幅を0.1とした、速度の値尺度である。
 - 角度：ロボット全体の姿勢を把握するための情報。取得されたロボットのすべての頂点の座標から、ロボットの傾きが計算される。値はラジアン単位であり、 $(-\pi, \pi]$ の値域である。
- 地形の状態
 - 高さ：地面までの高さを把握するための情報。ロボットの重心とその水平上の前後5マス分の、合計11点を基準として、y軸下方の地面までの距離を観測する。観測は11次元で構成され、1つのボクセル幅を1.0とした値尺度である。ただし、地面までの距離が5.0を超える場合や、地面が存在しない場合には、5.0が与えられる。
- 対象物の状態
 - 位置：操作対象となる物体の位置を把握するための情報。対象物のグリッド上のすべての頂点に対して2次元の座標が取得される。これから物体の重心を計算し、ロボットの重心からの相対的な座標とすることで、ロボットから対象物までの距離を観測する。観測は2次元で構成され、1つのボクセル幅を0.1とした値尺度である。
 - 速度：操作対象となる物体がどのように動いているかを把握するための情報。対象物のすべての頂点を持つ2次元の速度が取得される。すべての頂点の平均を取

ることで、物体全体としての速度が2次元で観測される、1つのボクセル幅を0.1とした値尺度である。

表 5.1: 各タスクに設定されているロボットの観測値

タスク	ロボット			地形	対象物	
	位置	速度	角度	高さ	位置	速度
Walker-v0	○	○	-	-	-	-
Carrier-v0	○	○	-	-	○	○
Thrower-v0	○	○	-	-	○	○
Climber-v0	○	○	-	-	-	-
ObstacleTraverser-v0	○	○	○	○	-	-
PlatformJumper-v0	○	○	○	○	-	-
AreaMaximizer-v0	○	-	-	-	-	-
Flipper-v0	○	-	○	-	-	-

表 5.1 に、各タスクに設定されているロボットのセンサーを示す。○が設定されている観測値を表している。ロボットの姿勢や部位の位置関係を認識するための位置観測はすべてのタスクで設定されている。足場が複雑な移動タスクでは、ロボットの角度や地形の観測が設定されている。また、物体を操作するタスクでは対象物に関する観測が設定されている。また、この研究で扱わないが、天井までの高さや対象物の角度などの観測値が設定されているタスクもある。

シミュレーションの毎時点で、設定されている観測情報を処理し、その時点の状態ごとに適切な動作を出力するような制御ニューラルネットワークを得ることが目的となる。ただしタスクとロボットの設計によって、ロボットを制御するニューラルネットワークの入出力のノード数が異なる。設計が異なると、ロボットの位置に関する観測値の次元数が異なる。また、ニューラルネットワークの出力は、ロボットに含まれるアクチュエータボクセルそれぞれの伸縮度合いであるため、出力ノード数も異なる。

5.2 3つの制御ニューラルネットワーク最適化手法

VSRs の制御にニューラルネットワークを用いることで、環境や姿勢に応じた適切な動作を実現することが可能となる。ニューラルネットワークは、複雑な非線形関数を近似する能力があり、ロボットの複雑な動作や多様な環境に対応するための最適な制御戦略を学習することが可能である。また、環境からの入力に基づいて適切な動作をリアルタイムで決定するため、予期しない状況や不確定な環境にも柔軟に対応できる。しかし、タスクや制御する VSR の設計に対して、最適な制御ニューラルネットワークを得ることは難しい問題である。

これまでの VSRs 研究では、制御ニューラルネットワークを最適化するために、強化学習手法あるいは進化的手法のどちらか一方のみが使用されていた。どの手法がタスクや設計の動作性を引き出すために適しているのかは十分な検討がされてきていない。また、VSRs の研究で主に扱われる問題は、設計の最適化である。設計最適化では、ある設計の有効性を評価するために、その制御をシミュレートして評価する必要がある。このとき、制御ニューラルネットワークの最適化が必要であり、その結果が設計最適化の方向性に大きな影響を与えることが想定される。そのためこの研究では、強化学習手法と進化的手法の、タスクや VSR の設計による最適化性能の傾向や特徴、さらには設計最適化に与える影響についても検討する。これにより、今後の VSRs 開発で適切な手法を選択することに貢献する。

この研究では、VSRs の制御ニューラルネットワークの最適化手法として、強化学習手法である Proximal Policy Optimization (PPO)、進化的手法である Neuroevolution of Augmented topologies (NEAT) と Hypercube-based NEAT (HyperNEAT) を用いる。以下でそれぞれの特徴と、この研究でのパラメータ設定について詳細に説明する。

5.2.1 Proximal Policy Optimization (PPO)

PPO [58] は、ロボットの制御最適化を行う強化学習手法として広く使われている手法である [5, 79]。PPO は、ロボットが環境から観測を受け取りどのような行動をとるかを定める policy ネットワークと、その行動がどの程度良かったかを評価する critic ネットワークが連携して学習を行う。PPO は複数の環境を一定回数ランダムな行動でサンプリングし、それらの行動と報酬のデータを用いて policy ネットワークと critic ネットワークを同期して更新する。複数の環境を並列して探索することで効率的な学習を実現している。また、policy が一度に大きく更新されないように更新幅をクリップすることで、安定した学習を行うことができる、シンプルで優れた手法となっている。

PPO は VSRs の制御ニューラルネットワークのパラメータ最適化に頻繁に利用されている手法である。本実験でも Evolution Gym が提案された論文の実験で用いられている PPO のパラメータ設定を採用した [5]。コントローラとなるニューラルネットワークは、それぞれ 64 のノードを持った 2 層の隠れ層から構成されている。入出力のノード数は前節で述べた通り、タスクや設計によって異なる。学習率は 2.5×10^{-4} 、クリップパラメータは 0.1、critic ネットワークの損失係数は 0.5、エントロピー係数は 0.01 である。そして、並列で進行させる環境の数を 4 として、環境を 128 ステップサンプリングするごとに勾配降下法によって 4 エポックの学習を行う。これを 1 サイクルとし、合計で 5,000 回の学習を行い、50 回ごとにタスクを一回分を評価する。

5.2.2 Neuro Evolution of Augmenting Topologies (NEAT)

NEAT はニューラルネットワークのノードとエッジを遺伝子で表し、遺伝的アルゴリズムによってニューラルネットワークを最適化する手法である [65]。エッジの重みだけでなくそ

の構造を進化させることから、柔軟に複雑なタスクに利用することが可能である。入力ノード、出力ノード、バイアスノード、隠れノードのみの非常にシンプルなトポロジーを初期構成とし、突然変異によりノードやエッジの遺伝子を追加することで徐々にニューラルネットワークを複雑化するため、幅広い解空間を探索可能である。進化アルゴリズムであるため学習には時間がかかるが、複雑な非凸問題に対して有効に働くことから、ロボット制御タスクにしばしば用いられる。

NEAT では、一世代の集団数を 200 として、合計 500 世代進化を繰り返す。初期集団の各個体は 1 つの隠れノードと、入力ノードと隠れノードと出力ノード間に 50% の確率でランダムに張られたエッジによって初期化される。個体の突然変異は、50% の確率で新たにエッジが張られ、10% の確率で既存のエッジが取り除かれる。同様に 10% の確率で新たにノードが追加され、10% の確率で既存のノードが取り除かれる。ノードの活性化関数にはシグモイド関数を用いる。

5.2.3 Hypercube-based NEAT (HyperNEAT)

HyperNEAT [64] は、Compositional Pattern Producing networks (CPPN) [63] を進化させてニューラルネットワークの接続パターンを最適化する手法である。CPPN は座標情報を入力としてその座標の状態を決定するネットワークであり、HyperNEAT においては、超立方体上に配置されたニューラルネットワークのノードの状態やノード間のエッジの状態を決定するために利用される。NEAT では直接的にニューラルネットワークの構造を最適化するのに対して、HyperNEAT では CPPN を進化させることで間接的にニューラルネットワークの構造を最適化する。そのため、大規模なニューラルネットワークにおいても効率的に最適化することが可能である。

CPPN は座標情報から状態を決定することから、対称性や繰り返し、反復などの規則性を持つパターンを表現することが可能である。そのためロボット制御のような、位置情報に基づいた観測があるタスクに HyperNEAT を用いることで、位置関係を活かした効率的なニューラルネットワークの最適化が期待できる。位置情報を活かすためにニューラルネットワークのノードの座標を適切に設定することが重要となる。

そこで本実験では、観測の種類やロボットの構造、セルの種類などに従って、入力ノードと出力ノードの座標を表 5.1 のように設定した。ニューラルネットワークの入力ノードのは、タスクに設定されている観測に対応しており、それぞれの観測に合わせて入力ノードの座標を決定する。例えば、ロボットの速度は x, y の 2 次元で観測される。そのため、それぞれの速度の観測情報も観測 x と観測 y で区別して配置している。ロボット位置の観測値は、ロボット設計のグリッドの頂点ごとに観測される。接続されていないセル同士では、同じ座標の頂点であっても別々の観測となるため、その頂点が四隅のどのセルによって共有されているかを、0 または 1 の値をとるセル「L」、「T」、「R」で区別している。ロボット位置やアクチュエータのノードにおける位置 x, y では、グリッドの頂点やセルごとにノードがあるため、その物理空間での位置情報 x, y で区別している。

表 5.2: 超立方体上に配置したニューラルネットワークの入力ノードと出力ノードの座標

ノード	ロボット	地面	対象物	速度	観測		相対的	位置		角度	セル				アクチュエータ	
					x	y		x	y		」	レ	ヲ	「	水平	垂直
ロボット速度 x	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
ロボット速度 y	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
ロボット位置 x	1	0	0	1	1	0	1	*	*	0	*	*	*	*	0	0
ロボット位置 y	1	0	0	1	0	1	1	*	*	0	*	*	*	*	0	0
ロボット方向	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
地面	0	1	0	0	0	1	0	*	-1	0	0	0	0	0	0	0
対象物速度 x	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0
対象物速度 y	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0
対象物位置 x	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0
対象物位置 y	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0
水平アクチュエータ	1	0	0	0	0	0	0	*	*	0	0	0	0	0	1	0
垂直アクチュエータ	1	0	0	0	0	0	0	*	*	0	0	0	0	0	0	1

つまり、ニューラルネットワークの各ノードは、表 5.1 における列の数である 16 次元の超立方体空間に配置されている。CPPN による 2 つのノード間の接続を決定では、接続元となるノードの 16 次元の数値と、接続先となるノードの 16 次元の数値、合わせて 32 次元の値を CPPN に入力することで、エッジの重みを出力する。また、表中の * は、対応するノードが複数あることを表している。例えば、「ロボット位置 x」は、設計グリッドの頂点の数だけある。この頂点はそれぞれ (x, y) 座標が異なっているため、その座標の数値が、観測値の「位置 x, y 」における * 記号に代入される。また、セル「」,「レ」,「ヲ」,「「」は、頂点ごとに異なるために、該当する頂点のみ 1 となる。

HyperNEAT は、NEAT と同様に、一世代の集団数を 200 として、500 世代進化を繰り返す。初期集団の各個体 (CPPN) は 1 つの隠れノードと、入力ノードと隠れノードと出力ノード間に 50% の確率でランダムに張られたエッジによって初期化される。CPPN の突然変異では、20% の確率で新たにエッジが張られ、10% の確率で既存のエッジが取り除かれる。同様に 20% の確率で新たにノードが追加され、10% の確率で既存のノードが取り除かれる。CPPN の各ノードの活性化関数には、対称性や周期性を表現するサイン関数を用いる。また、コントローラとなるニューラルネットワークの基盤におけるノードの活性化関数には、NEAT と同様に、シグモイド関数を用いる。

5.3 比較実験の詳細

強化学習と進化的手法による VSRs の制御最適化を比較する実験を行う。強化学習として Proximal Policy Optimization (PPO) , 進化的手法として, Neuroevolution of Augmented

topologies (NEAT) と Hypercube-based NEAT (HyperNEAT) を用いる。実験には、前述の 8 つのベンチマークタスクを利用し、各タスクで制御を最適化するロボットの設計を 2 種類を用意した。8 つのタスクそれぞれに用いるロボットの設計を図 5.1 に示す。まず 1 つ目は、PPO による制御最適化を用いた設計最適化で得られた設計 (PPO-Optimized) である。これらの設計は、先行研究によって得られており、設計の最適化には遺伝的アルゴリズムが用いられている [5]。2 つ目は、タスクごとに手動でデザインした設計 (Hand-Designed) である。例えば、Walker-v0 や PlatformJumper-v0 に設定した Hand-Designed の設計は、猫のような二足歩行の動物を模しており、前後の足の Vertical アクチュエータボクセルによって、地面を蹴り上げることで歩行やジャンプが可能になると考えられる。Flipper-v0 では、左に傾いた風車のような設計で、各羽は、接する地面に対して押し出すようにアクチュエータボクセルを配置しており、左側への回転を効率的に行うことが期待される。

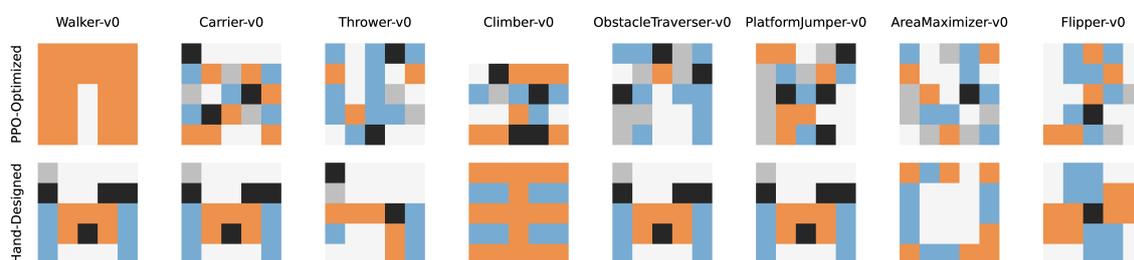


図 5.1: 実験に用いる、8 つのタスクとそれぞれ 2 つの設計のペア

PPO-Optimized の設計については、Evolution Gym において、PPO による制御最適化を通じて、最適化された設計である。そのため、PPO を用いた制御の最適化で高い報酬を得られることがすでに保証されている。このロボット設計に対して、NEAT や HyperNEAT で制御を最適化した結果を比較することで、異なるアルゴリズムがロボットの動作性能にどのような影響を及ぼすかを検証する。一方で、Hand-Designed の設計については、PPO-Optimized とは対照的に、PPO の制御最適化による影響を受けず、人手で高い動作性を持つことを期待して設定している。PPO-Optimized の設計が Evolution Gym 環境で、PPO による制御最適化によって得られた最適解を示しているのに対し、Hand-Designed 設計は部分的または別の種類の最適解の可能性を探るものである。

PPO-Optimized と Hand-Designed の異なる 2 つの設計それぞれに対して、異なる制御最適化手法の性能を比較し、ロボット設計と手法の特性の関係を明らかにする。また、タスクの種類や難易度による、各手法の傾向の差異や、2 種類の設計間で比較することによる設計最適化に与える影響について考察する。3 つの手法は全てランダム性があるため、全ての実験は 6 回ずつ行い評価する。

5.4 実験結果

まず、PPO-Optimized の設計を使用した、各実験の最大報酬の遷移を図 5.2 に示す。横軸は累積したシミュレーションのステップ回数を示しており、縦軸は、その時点での最大報酬を表す。曲線は 6 回の実行における平均であり、バンドは標準偏差をカバーしている。NEAT と HyperNEAT では、1 つの個体を評価するために、各タスクに設定されているステップ数のシミュレーションを行う。このシミュレーションのステップ数を累積していったときの最大報酬を示している。一方 PPO では、1 回の学習サイクルごとに 4 つの並列プロセスで、128 ステップ数ずつ環境が探索されている。その累積ステップ数と、50 サイクルごとに、タスクの遂行性能を評価するために使用されたタスク 1 回分のステップの累積に対して、報酬がプロットされている。NEAT と HyperNEAT、および PPO におけるシミュレーションステップ数のアルゴリズム上の意味合いは異なるが、演算回数を目安として過去の比較研究でも利用されている [72]。

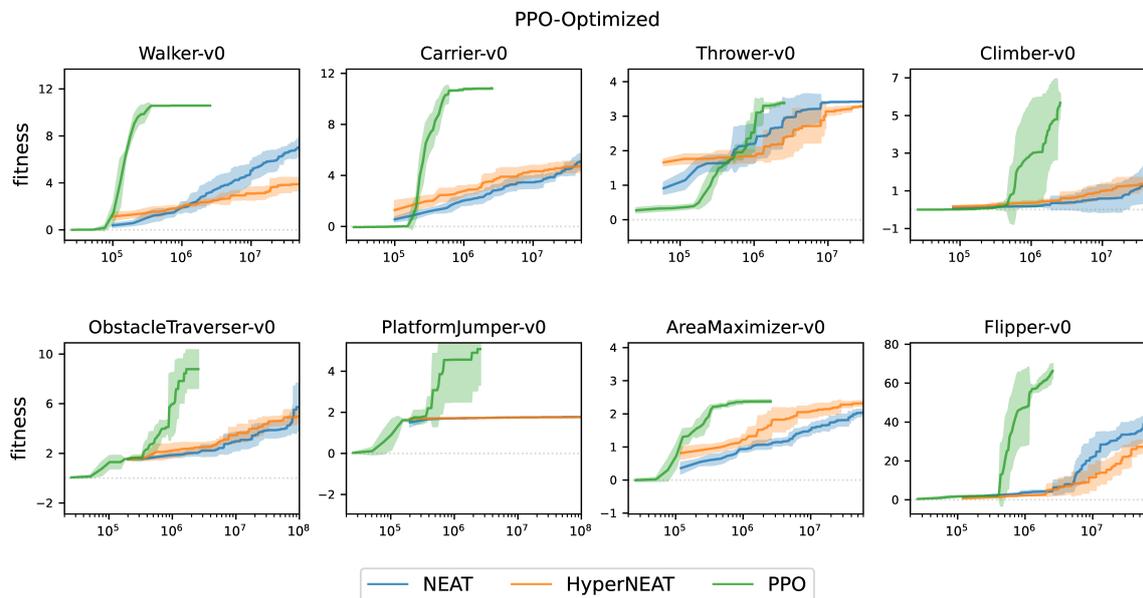


図 5.2: PPO-Optimized の設計に対する、各手法による制御最適化の過程。

図 5.2 から、PPO と進化的手法の 2 つの間には、学習効率の観点で大きな差がある。PPO は、進化的手法の 100 分の 1 程度のシミュレーションステップ数で、より高い最適化の成果をあげている。進化的手法は、1 つ 1 つの個体の評価に、タスクに設定されている固定のシミュレーションステップ数が必要となる。そのため、強化学習に比べ、効率が大幅に悪い結果となった。

また、それぞれの制御最適化手法の、PPO-Optimized の設計を使用した、各タスクにおける報酬の結果を表 5.3 に示す。6 回実行した結果の報酬の平均と標準偏差を示している。NEAT と、

HyperNEAT は全個体の中での最大報酬，PPO は合計 100 回評価された中での最大の報酬を用いて，統計している．この表では，各手法の最適化性能を比較することができる．最適化性能においても，PPO がほとんどのタスクにおいて，他の進化的手法に比べ大きく優れている結果である．Thrower-v0 に関しては，NEAT が最も高い報酬を得ているが，PPO と HyperNEAT と比べても大きな差はなく，PPO が優れていることには変わらない．

tasks	PPO-Optimized		
	NEAT	HyperNEAT	PPO
Walker-v0	7.010 ± 0.731	3.889 ± 0.548	10.582 ± 0.001
Carrier-v0	5.065 ± 0.621	4.721 ± 0.442	10.816 ± 0.076
Thrower-v0	3.421 ± 0.008	3.284 ± 0.024	3.384 ± 0.030
Climber-v0	1.326 ± 0.974	1.371 ± 0.319	4.540 ± 0.200
ObstacleTraverser-v0	5.716 ± 1.918	4.971 ± 0.484	8.783 ± 1.534
PlatformJumper-v0	1.775 ± 0.005	1.762 ± 0.005	5.079 ± 1.753
AreaMaximizer-v0	2.037 ± 0.096	2.318 ± 0.074	2.378 ± 0.042
Flipper-v0	38.818 ± 3.378	27.242 ± 3.329	66.279 ± 3.740

表 5.3: PPO-Optimized の設計に対する，各手法による制御最適化の結果．

次に，図 5.2 と表 5.4 と同様に Hand-Designed の設計に対する結果を，図 5.3 と表 5.4 に示す．学習効率に関しは，言わずもがな，PPO が最も優れている．表 5.4 の最大報酬でも，PPO-Optimized の場合と同様に，ほとんどのタスクで，PPO の最適化性能が優れていることが確認できる．この結果は，PPO が多くの設計に対しても制御の最適化がうまく進む汎用的な手法であることを示唆している．しかし，Thrower-v0 と PlatformJumper-v0 では，HyperNEAT は PPO に対して十分な差をともなって優れた最適化結果であった．

また，PPO-Optimized と Hand-Designed に対する結果を比較する．特に PPO-Optimized の設計は，PPO による制御最適化を用いた設計最適化によって得られたものであり，Hand-Designed の設計よりも優れた動作性を引き出すことができると想定された．その想定通り，実際にほとんどのタスクで，Hand-Designed の設計よりも，PPO-Optimized の設計に対する，PPO による制御最適化の結果が優れていた．もちろん，PPO-Optimized の設計の方が潜在的な能力で優れているということは考えられる．しかしながら，進化的アルゴリズムに関して 2 つの設計に対する結果を比較すると，HyperNEAT では，Walker-v0 と Climber-v0 を除く 6 つのタスクで，Hand-Designed の方が優れた結果であった．NEAT についても，タスクごとに異なる結果であった．つまり，設計の潜在的な能力の間に明確な差があるわけではなく，動作性を引き出すために用いる制御最適化手法によって差が生まれている．制御最適化手法によって，性能を引き出すことができる設計が異なるということは，設計最適化をしたときに得られる最適な設計も大きく異なるということを示唆している．

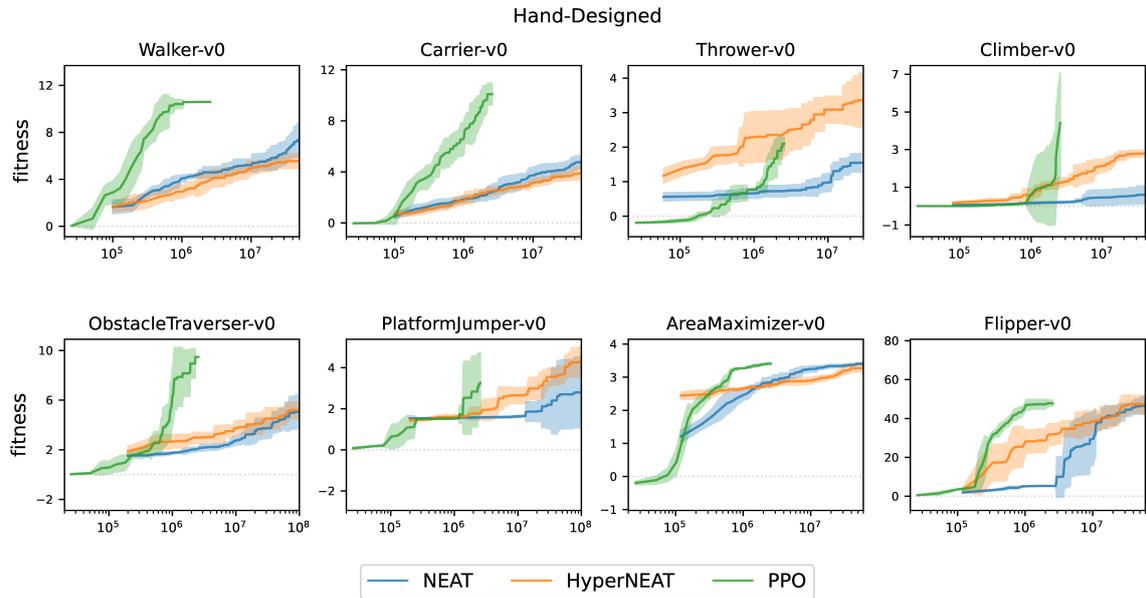


図 5.3: Hand-Designed の設計に対する、各手法による制御最適化の過程.

tasks	Hand-Designed		
	NEAT	HyperNEAT	PPO
Walker-v0	7.357 ± 1.415	5.556 ± 0.616	10.593 ± 0.004
Carrier-v0	4.759 ± 0.497	3.889 ± 0.502	10.090 ± 0.849
Thrower-v0	1.547 ± 0.263	3.361 ± 0.782	2.094 ± 0.191
Climber-v0	0.602 ± 0.468	2.789 ± 0.174	4.411 ± 2.665
ObstacleTraverser-v0	5.054 ± 1.359	5.195 ± 0.638	9.477 ± 0.007
PlatformJumper-v0	2.794 ± 1.713	4.270 ± 0.733	3.258 ± 1.488
AreaMaximizer-v0	3.405 ± 0.036	3.273 ± 0.118	3.410 ± 0.019
Flipper-v0	46.557 ± 4.369	47.400 ± 4.534	47.835 ± 1.522

表 5.4: Hand-Designed の設計に対する、各手法による制御最適化の結果.

5.4.1 HyperNEAT と PPO の比較

次に、HyperNEAT と PPO の間にある最適化傾向について詳細に考察する。HyperNEAT が PPO よりも優れていた、Hand-Designed の設計に対する Thrower-v0 と PlatformJumper-v0 について、HyperNEAT と PPO によって最適化された実際の制御シミュレーションの様子を、図 5.4 と図 5.5 にそれぞれ示す。

まず、図 5.4 に示した Thrower-v0 の結果を見ると、HyperNEAT では箱を遠くまで投げることが成功し、高い報酬を獲得できている。一方で、PPO では箱を遠くまで投げられるような



図 5.4: Throwing-v0 タスクに対して、Hand-Designed 設計の制御を、HyperNEAT と PPO によって最適化したときの動作の様子。

行動を学習させることができず、低い報酬となった。Thrower-v0 では箱に瞬間的に大きなインパクトを加える必要がある。PPO-Optimized の設計では箱の位置を調整することなく、初期位置がインパクトを与えるのに適切であった。そのため、PPO に限らずどの最適化手法でも制御の最適化が比較的容易な設計となっていた。しかし、Hand-Designed の設計では、まず箱を最適なインパクトの位置に移動させる必要がある。PPO による制御最適化の場合、箱を最適な位置に移動させることに報酬が与えられず、適切でないタイミングでインパクトを与えてしまうため苦戦したと考えられる。一方、HyperNEAT の場合は、箱を移動させるような報酬に関係ない動作も探索することが可能である。結果として、全てのアクチュエータボクセルを連動させ、バネのように大きな力で箱の飛距離を伸ばした。



図 5.5: PlatformJumper-v0 タスクに対して、Hand-Designed 設計の制御を、HyperNEAT と PPO によって最適化したときの動作の様子。

次に、図 5.5 に示した PlatformJumper-v0 の結果を見ると、HyperNEAT では段差を超えて遠くまでロボットが移動できており、高い報酬を獲得できている。一方で、PPO では一段目

の段差を超えるような動作を学習することができず、低い報酬となった。HyperNEATの結果で示されたように、Hand-Designedの構造は段差を飛び越える能力を持っているにもかかわらず、PPOではその動作を最適化によって得ることができなかった。その要因として、ジャンプという行動そのものには報酬が与えられないことが考えられる。PlatformJumper-v0の地形は、1つ目の段差に突き当たるまでは直線が続いている。正のx方向に報酬が与えられるため、段差に当たるまではWalker-v0と同じ動きが最適となる。しかしながら、段差を越えるためのジャンプ行動は、走ることを学習した状態からは派生しにくいと考えられる。一方、PPO-Optimizedの設計では、走るという動作ではなくジャンプの動作が生まれるような設計であり、段差を越えるための大きなジャンプにも派生しやすくなったと考えられる。

5.4.2 NEATとHyperNEATの比較

次に、NEATとHyperNEATの最適化結果について考察することで、それぞれの最適化傾向を理解する。NEATとHyperNEATは、異なる特徴を持った制御に収束する。NEATは観測ノードとアクチュエータノードで、エッジまたは隠れノードを、1ずつしか増やすことができないが、それぞれのパラメータを独立して調整することが可能である。そのため、少数の特定のアクチュエータを適切に動かすことが必要なタスクにおいては、高い報酬を得ることができる。一方、HyperNEATは、基盤となるニューラルネットワークの全てのエッジの状態を一度に決定する。エッジの重みはノードの座標に基づいて決定されるため、近いアクチュエータボクセル同士は連動した動きを見せやすく、まとまった動きから進化が進むことが特徴的である。これらNEATとHyperNEATの最適化傾向の差が顕著に見られた例を図5.6と図5.7に示す。



図 5.6: Walker-v0 タスクに対して、Hand-Designed 設計の制御を、NEAT と HyperNEAT とによって最適化したときの動作の様子。

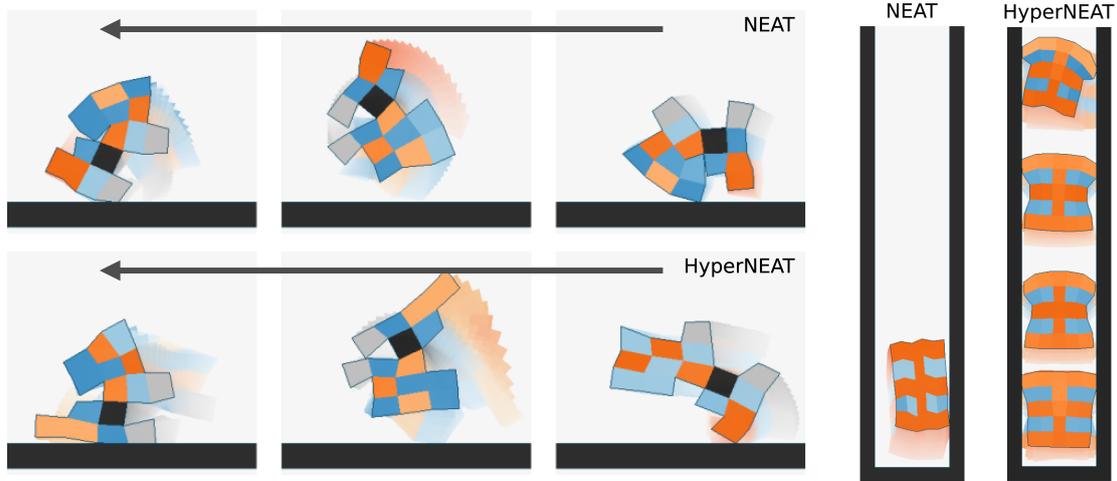


図 5.7: Flipper-v0 タスクに対する PPO-Optimized 設計, Climber-v0 タスクに対する Hand-Designed 設計, それぞれで, NEAT と HyperNEAT とによって最適化したときの動作の様子.

図 5.6 に示すとおり, PPO-Optimized の設計を用いた Walker-v0 タスクでは, NEAT では隣接するアクチュエータボクセル同士で連動性は認められず全く独立した動きをしているが, HyperNEAT ではボクセルが連動しながら, まとまりとなって活動している様子を観察することができる. また, 図 5.7 に示すとおり, PPO-Optimized の設計を用いた Flipper-v0 タスクでは, アクチュエータボクセルの一部を動かすように学習した方が早い回転につながり高い報酬を得ている. その結果, 全てのセルが連動して動くように学習が進む HyperNEAT に比べて, NEAT の方が高い報酬を得る結果であった. Climber-v0 タスクにおける Hand-Designed の設計は左右対称である. そのため全てのアクチュエータセルを均等に使うことでより高く壁を登ることができる. その結果, HyperNEAT は NEAT よりも有効に働き, より高い報酬につながったと考えられる.

NEAT と HyperNEAT による制御の最適化結果から, それぞれの手法の特徴を活かせるロボットの設計が明らかとなった. HyperNEAT による制御の最適化が活かされるのは, 全てのアクチュエータボクセルが連動しながらまとまりとなって動くことでタスクが遂行されるようなロボット設計である. 一方, NEAT は一部のアクチュエータボクセルを動かすことでタスクを遂行することができるようなロボット設計が適している. これらの制御最適化手法を用いて, 設計を最適化した場合には, アクチュエータボクセルの配置傾向に差が生じると考えられる.

5.5 まとめ

本章の実験では, VSRs の設計やタスクと, 制御の最適化手法の間にある関係性について探求した. 具体的には, 8 つのベンチマークタスクに対して, PPO による制御最適化によ

て最適化された設計（PPO-Optimized）と、人手で優れたタスク遂行を期待して作成した設計（Hand-Designed）を用意して、PPO、NEAT、HyperNEAT の3つの手法により制御の最適化実験をした。

まず全体的な結果として、PPO が最適化効率と最適化性能の面で優れていた。最適化効率として、制御の最適化に必要なシミュレーションステップの回数を用いた。この指標では、進化的手法である2手法よりも PPO が大幅に少なく済んだ。また、最適化性能についても、ほとんどのタスクで PPO の最適化で得られた制御による評価値が優れていた。PPO による最適化が有効に働くことが保障されている PPO-Optimized の設計のみならず、Hand-Designed の設計においても有効であった。

また、PPO-Optimized と Hand-Designed の設計についての比較した。PPO は、事前に想定した通り、PPO-Optimized の設計の方が、制御の最適化が有効に働いた。しかし、NEAT や HyperNEAT では、その傾向は確認されなかった。これは、最適化手法によって、性能を引き出すことができる相性の良い設計が異なるということを示唆しており、制御の最適化に使用する手法によって、設計の最適化で得られる結果は大きく異なるということにつながる。

また、NEAT と HyperNEAT による制御最適化の傾向について考察した。NEAT は一部のアクチュエータボクセルを動かすことによって、ロボットをタスクに適応させる。また、HyperNEAT は、設計における全てのアクチュエータボクセルを高い連動性で動かすことで、ロボットをタスクに適応させる。この2つの手法における最適化傾向には、大きな違いがあり、これらを制御最適化に用いた設計の最適化の結果にも、この傾向が反映されることが想定される。特に、NEAT の最適化傾向は、少ないアクチュエータボクセルによる省エネルギーな設計につながる可能性がある。また、HyperNEAT では、同種のアクチュエータボクセルがまとまっているような単純な設計につながる可能性がある。

今後の重要な研究課題として、まず、HyperNEAT や NEAT を制御最適化に用いたロボット設計の最適化実験が挙げられる。この実験で得られた、設計最適化に与える影響に関する示唆を、実際に確認するためにも、重要な意義を持つ。また、手法ごとにより適したロボット設計が明らかになることは、制御最適化手法やロボットの設計に役立つ知見を提供することが期待できる。

第6章 複雑化する環境に適応可能な設計

VSRsは、動物のような柔軟な動作性による適応性の高さから、人間の活動困難な環境での作業を期待されている。例えば、宇宙空間や海中、災害現場などが挙げられる。こういった環境は、もちろん人による整備はされておらず、そして予測困難な変化が起こるような不確実性を伴う。不確実な外的要因による影響に対しても、頑健に作業をこなすようなロボットの設計と制御が求められている。しかしながら、これまでのVSRsの研究では、単純な1つの固定された環境のみを設定している。最も単純な歩行タスクにおいても、段差や傾斜、障害物、落とし穴など、さらにはそれらが組み合わさった複雑な環境が現実では想定される。固定された1つの環境に対してのみ最適化されたロボットでは、こうした多様な環境には対応できない可能性がある。

環境を動的に変化させていくと同時に、ロボットが適応するように動作制御を複雑に更新していく手法が開発されている。迷路タスクを用いた研究では、地形が異なる迷路の集団と、迷路を解くエージェントの集団の2つを相互作用させることで、複雑な迷路とそれを解くことが可能なエージェントを進化的に得ることに成功している [6]。これをさらに複雑に拡張し、Bipedal Walker と呼ばれる、二足歩行ロボットの制御タスク [33] で行なった実験がある。Paired Open-Ended Trailblazer(POET) と呼ばれるこの手法では、複数のロボット-環境のペアを用意し、ペアとなる環境に適応するようにエージェントの制御を最適化する中で、新しい環境の生成や、他の環境にエージェントを転送することで、複雑な制御を得ることができ、多様な環境への適応性を高めることに成功した [76, 77]。最適化する環境を適切に変化させていくことで、1つの環境に過適合するのではなく、多様な環境に頑健に適応可能な制御を得ることを可能としている。

さらに、POET アルゴリズムを拡張して、同時にロボットの形態も最適化する研究が行われている [66]。この研究では、POET による実験と同様に、Bipedal Walker のタスクが実験されている。ロボットの形態は、それぞれの足の長さや幅をパラメータとして最適化する。この形態パラメータと、ロボットの動作を制御するニューラルネットワークのパラメータを1つに結合し、これをPOET アルゴリズムでのエージェントとして、環境とペアにして最適化している。結果として、困難なタスクでも適応性の高いロボットを得ることができていることや、単純な環境でも形態が収束せずに多様なロボットを評価することができた。ここで用いたロボットの形態は単純なものであり、VSRsのような複雑な設計問題では、どのような振る舞いが現れるか未知である。

そこで、本章の研究では、POET アルゴリズムを拡張し、環境の変化に応じてVSRsの設計と制御を最適化するアルゴリズムを提案する。VSRsの動作を制御するニューラルネットワー

クの構造は、その設計のセンサーの数やアクチュエータボクセルの数によって異なる。そのため、設計ごとにそれぞれ独立した制御ニューラルネットワークを用意する必要がある。前述の先行研究では、ロボットの形態が変化しても、センサーやアクチュエータの設定は変化しないため、同じ制御ニューラルネットワークを用いることができた。そのため、VSRsでは、より複雑にPOETアルゴリズムを拡張する必要がある。具体的には、制御の最適化とは独立して、設計の最適化を行うように拡張する。この拡張により、多様で複雑な環境でも頑健に適応可能なVSRsの設計を得ることが期待される。

6.1 環境の複雑化とロボットの最適化アルゴリズム

6.1.1 メインループ

この研究では、環境とロボットのコントローラの相互作用に加えて、ロボットの設計を進化させるアルゴリズムを提案する。本提案手法の基盤となるPOETアルゴリズムは、環境とロボットのコントローラがペアとなり、両方を相互作用させていく手法である。初めは単純な環境から、次第に複雑な環境を生成していく。コントローラの学習が一定回数行われるたびに、そのコントローラは異なる環境に転送される。この転送のプロセスによってコントローラは様々な環境に対して学習が行われるため、ロボットの動作性が強化される。実際に、1つの複雑な環境に対して適応する動作を学習することは難しいが、他の環境で学習したコントローラを転送することで適応できることが示されている。

POETアルゴリズムでは、ロボットの形態は所与のものであり、コントローラのみを学習させている。本研究は、POETアルゴリズムの概念を拡張し、ロボットの物理的な形態も相互作用の過程に組み込む。これにより、VSRsの設計や動作性も、環境の複雑化に応じて最適化されることが期待される。

提案手法の主なプロセスは、コントローラの学習と、環境間でのロボットとそのコントローラの転送、新しいロボットの生成、新しい環境の生成の4つの関数を繰り返し実行することである。その主要なループをAlgorithm 1に示す。POETアルゴリズムでは環境 $E(\cdot)$ に対して、その環境を学習したロボットのコントローラのパラメータ θ が1つのペアであった。これに対し提案手法ではロボットの設計 R を変数化し、ロボット設計-コントローラのセット (R, θ) が環境のペアとなる。ロボット設計の多様性を維持するために、1つの環境に対して複数のロボット設計-コントローラのセットをペアにする。

アルゴリズムは、次に示される環境とロボット設計-コントローラのペアリスト ERC_{list} に対して、操作を繰り返すことで進行する。主要なループでは、1ループごとに各環境にペアとなっている各ロボットのコントローラの学習を進める。これを $T^{transfer}$ 回繰り返すごとに転送、 T^{repro^R} 回ごとに新しいロボット設計の生成、 T^{repro^E} 回ごとに新しい環境の生成を

Algorithm 1 Main Loop

```
1: Input: initial environment  $E^{init}(\cdot)$ , list of initial robot(R)-controller( $\theta$ ) set  $RC\_list^{init}$ 
2: Parameters: iteration  $T$ , transfer interval  $T^{transfer}$ , reproduce robot interval  $T^{repro^R}$ , reproduce environment interval  $T^{repro^E}$ 
3: Initialize: Set  $ERC\_list$  empty
4: Add  $(E^{init}(\cdot), RC\_list^{init})$  to  $ERC\_list$ 
5: for  $t = 0$  to  $T-1$  do
6:    $N \leftarrow len(ERC\_list)$ 
7:   for  $n = 1$  to  $N$  do
8:      $E^n(\cdot), RC\_list^n \leftarrow ERC\_list[n]$ 
9:      $M \leftarrow len(RC\_list^n)$ 
10:    for  $m = 1$  to  $M$  do
11:       $R^m, \theta_t^{n,m} \leftarrow RC\_list^n[m]$ 
12:       $\theta_{t+1}^{n,m} \leftarrow \text{LEARN\_STEP}(E^n(\cdot), R^m, \theta_t^{n,m})$   $\triangleright$  optimize controller for each robots-environment pair
13:    end for
14:  end for
15:  if  $t \bmod T^{transfer} = 0$  then
16:     $ERC\_list \leftarrow \text{TRANSFER}(ERC\_list, ERC\_list)$   $\triangleright$  transfer robot-controller pairs to other environment
17:  end if
18:  if  $t \bmod T^{repro^R} = 0$  then
19:     $ERC\_list \leftarrow \text{REPRODUCE\_ROBOTS}(ERC\_list)$   $\triangleright$  reproduce new robots by mutation
20:  end if
21:  if  $t \bmod T^{repro^E} = 0$  then
22:     $ERC\_list \leftarrow \text{REPRODUCE\_ENVS}(ERC\_list)$   $\triangleright$  reproduce new environments by mutation
23:  end if
24: end for
```

行う。

$$ERC_list = [(E^n(\cdot), RC_list^n)]_{n=1}^{S^E} \quad (6.1)$$

$$RC_list^n = [(R^m, \theta^{n,m})]_{m=1}^{S^R} \quad (6.2)$$

ここで、 RC_list^n は環境 $E^n(\cdot)$ にペアとなっているロボット設計-コントローラのセットを表しており、 $\theta^{n,m}$ は環境 $E^n(\cdot)$ におけるロボット R^m のコントローラのパラメータを表す。優れたロボットは複数の環境とペアになることができるが、コントローラは環境ごとに独立

で学習させるため、ペアとなっている環境の数だけコントローラが用意される。また S^E, S^R はハイパーパラメータであり、それぞれ同時に保持できる環境の最大数、1つの環境にペアとなることのできるロボットの最大数を表す。初期の ERC_list は、初期の環境 $E^{init}(\cdot)$ と、初期のロボット設計-コントローラのリスト RC_list^{init} のペアから始まる。このとき初期のコントローラの学習が頓挫しないように、 $E^{init}(\cdot)$ は単純で適応が容易な環境を用いる。アルゴリズムの主要な関数である学習 (LEARN_STEP)、転送 (TRANSFER)、ロボット設計の生成 (REPRODUCE_ROBOTS)、環境の生成 (REPRODUCE_ENVS)、それぞれについて順に説明する。

6.1.2 制御の学習

アルゴリズムのループごとに、全てのロボット R^m がペアとなっている環境 $E^n(\cdot)$ ごとに、その環境に適応できるようにコントローラを学習し、パラメータ $\theta^{n,m}$ を更新する。このとき、ロボットのコントローラのパラメータは、ペアとなっている環境ごとに異なり、それぞれ独立で学習が進行する。コントローラの学習手法には進化アルゴリズムや強化学習など、反復的な学習を行う手法であれば様々な手法を用いることが可能である。

6.1.3 ロボットの転送

それぞれの環境で独立で学習したロボット設計-コントローラのセットを異なる環境に転送する。環境とロボットを相互作用させるために最も重要な機能である。POET アルゴリズムではロボットの設計は所与であったため、コントローラのための転送であった。提案手法ではロボット設計-コントローラのセットを転送することで、環境にペアリングするロボットの置換が発生し、優秀なロボット設計と優秀なコントローラを残していくことが可能である。

転送処理の流れを Algorithm 2 に示す。 ERC_list^{from} に含まれるすべてのロボット設計-コントローラのセットを ERC_list^{to} に含まれるすべての環境に転送する。転送するロボット設計-コントローラ $(R^k, \theta^{from,k})$ を転送先の環境 $E^{to}(\cdot)$ に適応できるように一定回数学習を行い、コントローラを微調整する ($\theta^{to,k}$)。これが、転送先の環境に既にペアとなっているのロボット設計-コントローラよりも適応度が優れていれば転送が成功となり、置き換えが発生する。このとき、コントローラのみを置き換える場合と、ロボット設計-コントローラのセットを置き換える場合の2通りがある。

まず、転送するロボット R^k が転送先の環境 $E^{to}(\cdot)$ にペアとして存在しない場合には、 $E^{to}(\cdot)$ にペアリングされた RC_list^{to} にロボット設計-コントローラ $(R^k, \theta^{from,k})$ を追加する。次に、 R^k が既に $E^{to}(\cdot)$ のペアとして存在しており、そのコントローラ $\theta^{to,k}$ よりも微調整後の転送するコントローラ $\theta^{from,k}$ が $E^{to}(\cdot)$ において優れている場合には、コントローラの置き換えが発生する。最後に、 RC_list^{to} のサイズが S^R を超える場合には、転送先の環境 $E^{to}(\cdot)$ における適応度が低いロボット設計-コントローラを削除する。どちらの場合にも、転送先の環境

Algorithm 2 TRANSFER

```
1: Input: two Environment(E)-(Robot(R)-Controller( $\theta$ ) sets) pairs list  
    $ERC\_list^{to}, ERC\_list^{from}$   
2: Parameters: limit size of robot-controller pairs list that can be paired to one environment  $S^R$   
3: for each ( $E^{to}(\cdot), RC\_list^{to}$ ) in  $ERC\_list^{to}$  do  
4:   for each ( $E^{from}(\cdot), RC\_list^{from}$ ) in  $ERC\_list^{from}$  do  
5:     for each ( $R^k, \theta^{from,k}$ ) in  $RC\_list^{from}$  do  
6:        $\theta^{from,k} \leftarrow \text{LEARN\_STEP}(E^{to}(\cdot), R^k, \theta^{from,k})$  ▷ fine-tune controller  
7:       if  $R^k$  not in  $RC\_list^{to}$  then  
8:         Add ( $R^k, \theta^{from,k}$ ) to  $RC\_list^{to}$   
9:       else if  $E^{to}(R^k, \theta^{from,k}) > E^{to}(R^k, \theta^{to,k})$  then  
10:         $\theta^{to,k} \leftarrow \theta^{from,k}$  ▷ replace controller  
11:      end if  
12:    end for  
13:  end for  
14:  if  $\text{len}(RC\_list^{to}) > S^R$  then  
15:     $RC\_list^{to} \leftarrow \text{SORTED}(RC\_list^{to}, E^{to}(\cdot))$  ▷ sort robots by fitness in the  
    environment that transfer to  
16:     $\text{num}^{rm} \leftarrow \text{len}(RC\_list^{to}) - S^R$   
17:    Remove  $\text{num}^{rm}$  robots with lower fitness from  $RC\_list^{to}$  ▷ select adaptive robots  
18:  end if  
19: end for
```

において適応度が高いもののみを残すことによって、同時に優秀なロボット設計の探索とコントローラの洗練を行うことが可能となる。

Algorithm 1: 16におけるTRANSFER関数の呼び出しでは、 $ERC_list^{to} = ERC_list^{from} = ERC_list$ として、その時の ERC_list に含まれる全ての環境間で、ペアとなっているロボット-コントローラを他の全ての環境へ転送する。

6.1.4 新しいロボットの生成

遺伝的アルゴリズムを用い、既存のロボット設計を突然変異させることで新たなロボット設計を生成する。優れたロボットの設計を探索するために新たにPOETアルゴリズムに追加した機能である。処理の流れをAlgorithm 3に示す。まずペアとなっている環境に対して ϵ^{repro^R} 以上の適応度を持っているロボットを親の候補として取得する。そして親の候補の中からロボット設計をランダムに1つ選び、それを突然変異させることで新たなロボット設計を生成する。これを N^{repro^R} 回繰り返す。 ϵ^{repro^R} と N^{repro^R} はハイパーパラメータであり、事前に与える。親とするロボット設計に適応度の閾値 ϵ^{repro^R} を設けることによって、適応性の高いロ

Algorithm 3 REPRODUCE_ROBOTS

```
1: Input: Environment( $E$ )-(Robot( $R$ )-Controller( $\theta$ ) sets) pairs list  $ERC\_list$ 
2: Parameters: Number of robot to reproduce  $N^{repro^R}$ , reproduction threshold  $\epsilon^{repro^R}$ 
3: Initialize: Set  $R\_parent\_list, R\_child\_list$  empty
4: for each ( $E(\cdot), RC\_list$ ) in  $ERC\_list$  do
5:   for each ( $R, \theta$ ) in  $RC\_list$  do
6:     if  $E(R, \theta) \geq \epsilon^{repro^R}$  then
7:       Add  $R$  to  $R\_parent\_list$ 
8:     end if
9:   end for
10: end for
11: for  $n = 0$  to  $N^{repro^R} - 1$  do
12:    $R\_parent \leftarrow \text{CHOICE}(R\_parent\_list)$  ▷ randomly pick one robot as parent
13:    $R\_child \leftarrow \text{MUTATE\_ROBOTS}(R\_parent)$  ▷ reproduce new robot by mutation
14:   Add  $R\_child$  to  $R\_child\_list$ 
15: end for
16:  $new\_RC\_list \leftarrow \text{BOOTSTRAP}(E^{init}(\cdot), R\_child\_list)$  ▷ initialize and pretrain controllers
17:  $ERC\_list \leftarrow \text{TRANSFER}(ERC\_list, new\_RC\_list)$  ▷ transfer to current
18: Return:  $ERC\_list$ 
```

ロボットが形質を遺伝させることができ、優れたロボット設計に進化させることが可能となる。

新しく生成したロボットのコントローラは初期状態であるため、環境とペアになり学習が進行している既存のロボットと比較することができない。そのため、新しく生成されたロボットに初期の環境 $E^{init}(\cdot)$ で事前学習させることで基本的な動作を獲得させる。その後、新しいロボット設計-コントローラのペアを ERC_list 内の環境に転送する。転送の際には、既存のペアとなっているロボットと比較して適応度が高ければその環境のペアとなることができる。そのため、新しく生成したロボット設計全てが集団に加えられるわけではない。

ロボット設計の生成には、一般的に CPPN [63] が利用されてきた [11, 5]。本実験でも同様に CPPN を用いてロボット設計の生成を行う。CPPN はロボットのグリッド状のマス目の座標 (x, y) と中心からの距離を入力として受けとり、5 種類のボクセルそれぞれの度合いを出力する。この度合いが最大のボクセル種類を、各マス目のボクセルのとして決定する。新しいロボットの生成には、CPPN を突然変異させることで同様に得ることが可能である。また、本研究の実験で生成するロボットの設計のサイズは 5×5 で固定とした。

6.1.5 新しい環境の生成

遺伝的アルゴリズムを用い、既存の環境を突然変異させることで新しい環境を生成する。生成方法は POET アルゴリズムと同様であり、難易度と新規性の 2 つの指標に基づいて環境を

Algorithm 4 REPRODUCE_ENVS

```
1: Input: Environment( $E$ )-(Robot( $R$ )-Controller( $\theta$ )) sets) pairs list  $ERC\_list$ 
2: Parameters: Number of candidates environment to reproduce  $N^{repro^E}$ , Number of environments to admit  $N^{admit}$ , reproduction threshold  $\epsilon^{repro^E}$ , limit size of environment list  $S^E$ 
3: Initialize: Set  $E\_parent\_list, E\_child\_list$  empty
4: for each ( $E(\cdot), RC\_list$ ) in  $ERC\_list$  do
5:   if  $\exists(R, \theta) : E(R, \theta) > \epsilon^{repro^E}, (R, \theta) \in RC\_list$  then
6:     Add  $E(\cdot)$  to  $E\_parent\_list$ 
7:   end if
8: end for
9: for  $n = 0$  to  $N^{repro^E} - 1$  do
10:   $E\_parent(\cdot) \leftarrow \text{CHOICE}(E\_parent\_list)$   $\triangleright$  randomly pick one environment as parent
11:   $E\_child(\cdot) \leftarrow \text{MUTATE\_ENV}(E\_parent(\cdot))$   $\triangleright$  reproduce new environment by mutation
12:  if  $mc\_lower < \max [E\_child(R, \theta)]^{\forall(R, \theta)} < mc\_upper$  then  $\triangleright$  check difficulty of the new environment
13:    Add  $E\_child$  to  $E\_child\_list$ 
14:  end if
15: end for
16:  $E\_child\_list \leftarrow \text{SORTED\_BY\_NOVELTY}(E\_child\_list)$   $\triangleright$  sort new environments by novelty
17:  $num^{rm} \leftarrow N^{repro^E} - N^{admit}$ 
18: Remove  $num^{rm}$  environments with lower novelty from  $E\_child\_list$ 
19:  $new\_ERC\_list \leftarrow \text{TRANSFER}(E\_child\_list, ERC\_list)$   $\triangleright$  transfer current robots and controllers to them
20: Add  $new\_ERC\_list$  to  $ERC\_list$ 
21: if  $\text{len}(ERC\_list) > S^E$  then
22:   $num^{rm} \leftarrow \text{len}(ERC\_list) - S^E$ 
23:  Remove  $num^{rm}$  environments with older from  $ERC\_list$ 
24: end if
25: Return:  $ERC\_list$ 
```

生成する。これによってロボットは円滑な学習と新しい動作を獲得することができるようになる効果がある。

処理の流れを Algorithm 4 に示す。まず適応度が ϵ^{repro^E} より大きいロボットをペアに持つ環境を親の候補として選択する。そして親の候補から環境をランダムに 1 つ選び、それを突然変異させることで新たな環境の候補を生成する。このとき、その環境に対する既存のロボット設計-コントローラの適応度を評価し、そのうちの最大の適応度によって環境の難易度を判

定する．最大の適応度が mc_lower 以下であれば難しすぎる， mc_upper 以上であれば簡単すぎるものとして，新しい環境の候補から除外する．これを繰り返して N^{repro^E} 個の新しい環境を生成し，それぞれ新規性を計算する．その後，新規性の高い N^{admit} 個の環境に対して， ERC_list に含まれる既存のロボット設計-コントローラのセットを転送することでペアを作成し， ERC_list に加える．このときに ERC_list のサイズが S^E を超える場合には古いものから削除する．

新規性の計算は，Enhanced-POET [77] と同様の方法を採用した．既存のすべてのロボット設計-コントローラによる適応度を評価しそれをベクトルとし，既存の各環境の同ベクトルのうちの第一近傍の距離を新規性とする．これにより，ロボットに対する本質的な環境の差異を評価することができ，効率的な環境の複雑化を行うことができる．

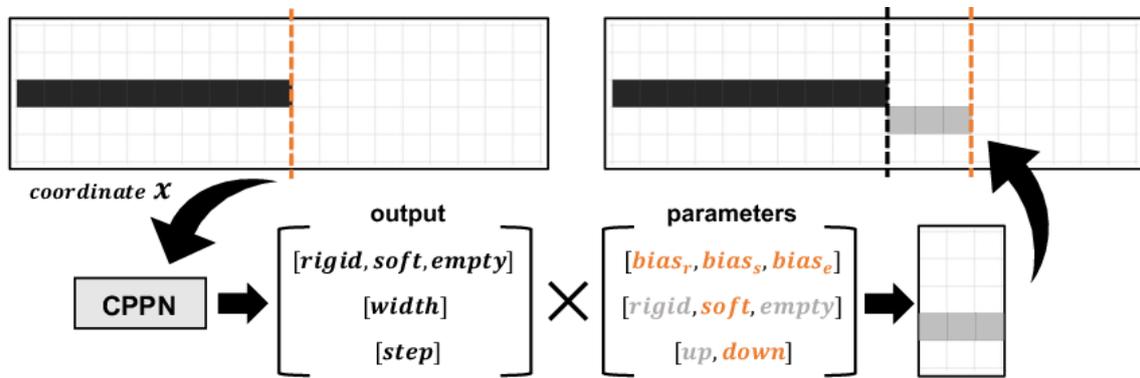


図 6.1: CPPN を用いた環境の生成.

POET アルゴリズムの実験 [77] と同様に CPPN を用いて環境を生成する．当研究では実験環境に Bipedal Walker [33] を利用していたが，本研究では Evolution Gym を利用する．従って，ここでは Evolution Gym における CPPN を用いた環境生成を定義する．

CPPN を用いた環境の生成方法を図 6.1 に示す．環境は複数の平坦な足場によって構成され， i 番目の足場は，設置する x 座標 x_i と足場のボクセル種類 $type_i$ ，足場の長さ $width_i$ ，前の足場からの段差 $step_i$ によって定義される．CPPN に x_i を入力することで 3 つの足場属性を決定し， $x_{i+1} = x_i + width_i$ として， x 座標が 100 になるまで繰り返す．ここで提案手法ではコントローラを徐々に洗練していくために，単純で簡単な環境から徐々に難しく複雑化していくことが求められる．しかし，生成する難易度を調整するために CPPN の突然変異を制御することは困難である．そこで本研究では CPPN とは別に，CPPN の出力を補正するパラメータを設けることで難易度の制御を行った．

図 6.1 に示すように，CPPN は足場のボクセル種類について $\mathbf{o}^{type} = [o_r^{type}, o_s^{type}, o_e^{type}]$ と，長さについて o^{width} ，段差について o^{step} の合計 5 つのパラメータを出力する．次に制御パラメータは，ボクセル種類について $\mathbf{p}^{type} = [p_r^{type}, p_s^{type}, p_e^{type}]$ ，長さについて $\mathbf{p}^{width} = [p_r^{width}, p_s^{width}, p_e^{width}]$ ，段差について $\mathbf{p}^{step} = [p_{up}^{step}, p_{down}^{step}]$ の合計 8 つ値で構成される．

これらの値を使用して， i 番目の足場のボクセル種類 $type_i$ ，長さ $width_i$ ，段差 $step_i$ は以

下のように計算する.

$$type_i = \mathop{\text{argmax}}(\mathbf{o}^{type} \odot \mathbf{p}^{type}) \quad (6.3)$$

$$width_i = o^{width} \cdot p_{type_i}^{width} \quad (6.4)$$

$$step_i = \begin{cases} o^{step} \cdot p_{up}^{step} & \text{if } o^{step} > 0 \\ o^{step} \cdot p_{down}^{step} & \text{otherwise} \end{cases} \quad (6.5)$$

ここで、 \odot は要素積を表し、式 6.3 では値が最大となるボクセル種類を取得する. 式 6.4 では、式 6.3 で得たボクセル種類に基づいた $p_{type_i}^{width}$ を用いて計算する. 式 6.5 では、CPPN の出力 o^{step} の正負で段差の上下を決定し、それぞれの場合に対応する p^{step} を用いて計算する.

CPPN の突然変異に合わせて制御パラメータをランダムに変化させることで、徐々に複雑な環境が生成されるようにする. 初期状態では単純な環境を得るために、 p^{type} には $[1, 0, 0]$, p^{width} には $[10, 3, 1]$, p^{step} には $[0, 0]$ を初期値に用いることで、初期の環境は必ず全て rigid ボクセルの平坦な足場が生成される. これらの値に、平均が正のガウス分布に基づく乱数を加えることで、制御パラメータを変異させる. ただし、 p_r^{width} についてのみ、rigid ボクセルの足場が短くなるほど難易度が高くなることが想定されるため、変異には平均が負のガウス分布を用いる.

6.2 実験と結果

本研究でロボットに学習させるのは踏破タスクである. Evolution Gym のベンチマークタスクに踏破タスクは用意されているが環境が固定であったため、環境を入力として与えるように新しくタスクを作成した. 3種類のボクセルを組み合わせた環境に対して、ロボットは環境を正の x 軸方向に進むように学習される. シミュレーションは 512 ステップの進行で終了し、ロボットの適応度は $\frac{\text{進んだ距離}}{10}$ で計算される. 生成する環境の横幅は 100 で固定とし、適応度の最大値は 10 となる.

この研究では、提案手法による実験 A と提案手法の有効性を確認する比較実験 B を行う. 実験 A では、多様な環境とロボットを相互作用させることでロボットの動作性を引き出し、同時に多様な環境にも適応可能なロボット設計を探索する. 実験 B では、実験 A で得られたいくつかの環境それぞれに対して、CPPN-NEAT を用いた従来のロボット設計最適化手法と同様に、固定された 1 つの環境に対するロボット設計の最適化をする. 提案手法によって得られたロボット設計とそのコントローラと、従来手法によって得たロボット設計を比較することで、提案手法における環境との相互作用の有効性を検証する.

本実験ではロボットのコントローラの学習には強化学習手法の PPO(Proximal Policy Optimization) [58] を用いる. 安定して精度良く学習を行うことができるため、ロボットの制御の学習によく利用されている. ニューラルネットワークによってロボットを制御し、そのパラメータ θ を PPO によって学習する. PPO の主要なハイパーパラメータは先行研究 [5] を参考

表 6.1: PPO の主要なハイパーパラメータ.

parameter	value
n_envs	4
learning_rate	$2.5 \cdot 10^{-4}$
n_steps	128
batch_size	128
n_epochs	4
clip_range	0.1
clip_range_vf	0.1
ent_coef	0.01
vf_coef	0.5

に表 6.1 に示す通り設定し, 実験 A, B で共通である. PPO による学習を繰り返す回数については, 各実験で扱いが異なるため, それぞれ後述する.

6.2.1 実験 A : 提案手法による動的環境における最適化実験

表 6.2: 提案手法ののハイパーパラメータ.

parameter	value	description
T	6,000	繰り返す回数
$T^{transfer}$	10	転送の頻度
T^{repro^R}	20	新しいロボットの生成の頻度
T^{repro^E}	60	新しい環境の生成の頻度
S^R	6	1つの環境にペアにできるロボットの上限
S^E	10	保持する環境の上限
N^{repro^R}	10	一度に生成する新しいロボットの候補数
N^{repro^E}	16	一度に生成する新しい環境の候補数
N^{admit}	1	一度の生成で集団に取り入れる環境数
ϵ^{repro^R}	5.0	親となるロボットの適応度の閾値
ϵ^{repro^E}	8.0	親となる環境の適応度の閾値

提案アルゴリズムによる実験を行う. 各種ハイパーパラメータを表 6.2 に示す. アルゴリズムは合計で 6,000 回繰り返され, 60 回ごとに新しい環境を生成し集団に加える. 8.0 以上の適応度 (80%以上の踏破率) のロボットがペアとなっていれば十分に適応されていると判断し, それらを親として突然変異させることで 16 の新しい環境の候補を生成する. この中から

難易度が適切で新規性が高い1つを集団に加える。このとき、環境の数が10を超える場合には古いものと入れ替える。

初期のロボット設計はランダムに生成され、新しく生成され複雑化した環境に十分に適応できるように、より高い頻度である20回ごとに新しいロボット設計を生成する。ペアとなっている環境に対して5.0以上の適応度を持つ、つまり適応性がある程度高いロボットを親として突然変異させることで10の新しいロボットの候補を生成する。生成されたロボットは事前学習を経た後に既存の環境に転送されることで集団に加わるかが判定される。

また、新しく集団に加わったロボットのコントローラの学習も十分にできるように、さらに高い頻度である10回ごとに環境間でロボット設計-コントローラの転送を行う。転送によって新しくロボットがペアとなる際、その環境にペアとなっているロボットの数が6を超える場合には適応度の低いロボットと入れ替える。集団に存続できるロボットの合計数は $S^E \times S^R = 60$ であり、同じロボット設計は同じ環境にペアとなることはできないため、1つのロボット設計の最大人口は10となる。

ロボットのコントローラの学習は、表6.1で示したパラメータに基づいて行われる。Algorithm 1: 12のLEARN_STEPではペアとなる環境に対して4回、Algorithm 2: 6のLEARN_STEPでは転送先となる環境に対して32回、Algorithm 3: 16のBOOTSTRAPでは初期環境に対して基礎的な動作を学習するために128回、PPOによる学習をそれぞれ繰り返す。

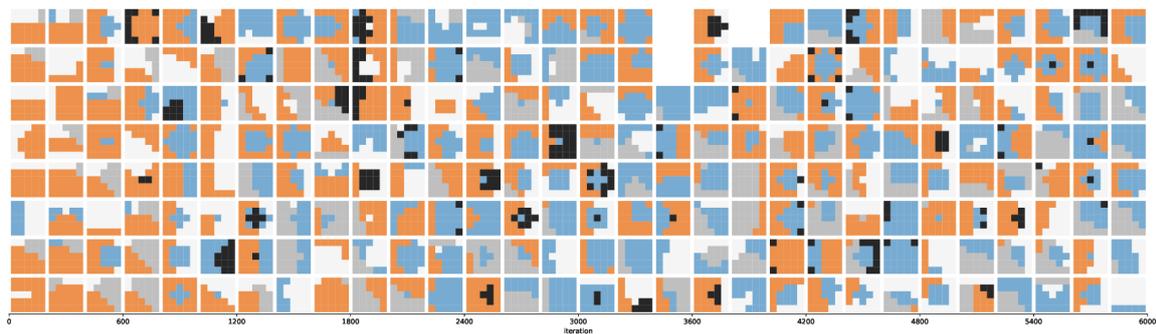


図 6.2: 集団に加わったロボット設計.

実験の結果、図6.2に示すような多様なロボット設計と図6.3に示すような複雑で多様な環境が生成された。ロボット設計はアルゴリズムの進行とともに合計で3,000種類が生成された。しかし集団に加わった、つまり環境とペアになることができたロボット設計は413種類であった。図6.2では集団に加わったロボットから、200 iterationごとにその期間中に生成されたロボットをランダムに24個ずつ抽出している。

図6.3では、集団に加わった100の環境全てを示しており、それぞれの環境の左端のx座標が生成されたiterationに対応している。このうち青く塗りつぶされた環境は8.0以上の適応度を得たロボットが存在しない環境である。アルゴリズムの進行と共に環境は順調に複雑化していき、それに伴いロボットもほとんどの環境に適応することができている。青く塗りつぶされた環境は、穴が大きかったり、段差が細かく続いている環境である。こういった環境は物



図 6.3: 生成された環境の履歴.

理的に踏破することが難しいため、どのロボットによっても適応できなかったと考えられる。

環境の生成は 60 iteration ごとに実行されるが、その時点で適応度が 8.0 以上のロボットが存在しなければ、親となる環境がなく生成は行われない。図 6.3 において、120 iteration 時点を見ると、環境が生成されていないことが確認できる。60 iteration の時点では新しく環境が生成できているが、その後に適応度が 8.0 未満に下がったということを表している。これは PPO は適応度が高くなるように学習するが、その途中では適応度が一貫して改善するとは限らないためである。特に学習回数が少ない段階では、こうした不安定性が現れやすい。

表 6.3: 環境の変化で得られた特徴的な 3 つの環境それぞれに適応度の高い 8 つのロボット設計。

		実験 A							
	soft	9.89	9.89	9.88	9.87	9.76	7.08	6.46	5.64
	empty	9.68	9.68	9.66	9.66	9.10	8.30	5.78	4.79
	step	8.30	7.71	6.65	6.47	6.36	6.20	5.84	2.90
									
									
									

図 6.3 で赤枠で示した 3 つの環境は、以下のそれぞれ異なる観点から選んだ特徴的な環境である。

- soft : soft ボクセルが最も多い。足場が重みで沈むため安定した制御が求められる。

- empty : empty ボクセルが最も多い。穴を認識し次の足場までの距離を飛び越える必要がある。
- step : 段差が最も多い。ジャンプで段差を登る必要がある。一定の高さをジャンプするためには精密な制御が求められるため難易度が高い。

これらの環境それぞれに対して適応度の高いロボット設計 8 つを表 6.3 に示す。赤枠で示したロボット設計は 3 つの環境全てで 8.0 以上の適応度を得た 1 つのロボット設計である。このロボット設計の 3 つそれぞれの環境における動作を図 6.4 に示す。



図 6.4: 環境の変化で得られた特徴的な 3 つの環境と、それらに全てに適応した 1 つのロボット設計。

また、3 つの環境に対するこのロボットの、学習と転送による適応度の変化の過程を図 6.5 に示す。赤い丸は他の環境からコントローラの転送が行われたタイミングを表す。また最大の適応度を得たタイミングを終点、そのとき発生していた転送における転送元の環境と動作を始点とした矢印を描画している。ロボットのコントローラであるニューラルネットのパラメータは微小な変化でもロボットの動作に大きな影響を与えるため、適応度は一貫して上昇せずに上下を繰り返す。下がったタイミングに他の環境で学習されたコントローラが転送されることによって、異なる動作をするパラメータから学習を再開できる。soft 環境では高い適応度で安定的であるのに対し、step 環境では生成時の転送によって最大の適応度を獲得しその後ほとんど高い適応度を獲得できていないことから、特に難易度の高い環境であると言える。

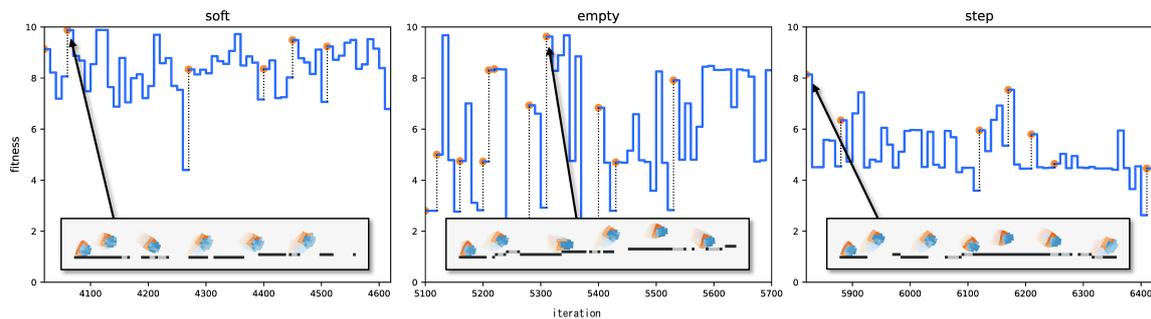


図 6.5: 3つの環境における適応度の変化と転送の例.

6.2.2 実験 B：固定環境に対する最適化実験

実験 A では多様で複雑な環境に適応することができる優秀なロボット設計を得ることができた。次に、提案手法との比較のため、他の環境間で転送を行わず、固定した1つの複雑な環境に対して適応可能なロボット設計を得られるかを実験する。実験 A で得た、図 6.4 に示した3つの環境を用い、それぞれに対して独立にロボット設計の最適化実験を行う。最適化には従来手法である CPPN-NEAT を用い、人口数 20 で 50 世代進化させ合計 1,000 種類のロボット設計を評価する。各ロボットのコントローラは表 6.1 のパラメータに基づいて、PPO で 1536 回学習を行う。学習を 16 回繰り返すごとに適応度を評価し、そのうちの最大値を最終的なロボットの適応度とした。

表 6.4: 固定環境に対する設計最適化によって得られた、3つの環境それぞれの適応度の高い8つのロボット設計。

		実験 B							
	soft	9.84	9.80	9.20	9.20	9.17	9.07	9.05	8.75
	empty	9.37	9.16	8.65	8.55	8.47	8.38	8.36	8.34
	step	3.42	2.78	2.71	2.71	2.70	2.68	2.65	2.57

実験の結果得られた適応度の高い8つのロボット設計を表 6.4 に、そのうち最も適応度の高かったロボットの動作を図 6.6 に示す。各環境ごとに独立した実験を行なっているため、得

られるロボット設計は各環境ごとに異なっている。表 6.3 と表 6.4 において、実験 A と B の各環境における適応度の最大値を比較すると、soft, empty の環境においては実験 A の結果と遜色ない結果であった。しかし step の環境においては、他の環境から転送しないロボット設計の進化では適応することはできなかった。soft と empty の 2 つの環境では、実験 A よりも実験 B の方が適応度の高いロボット設計を多く発見できていることが確認できる。これについて提案手法では、1 つの環境にペアとあることができる環境の数 S^R を大きくすることで、より多数の適応的なロボット設計を発見できる可能性がある。

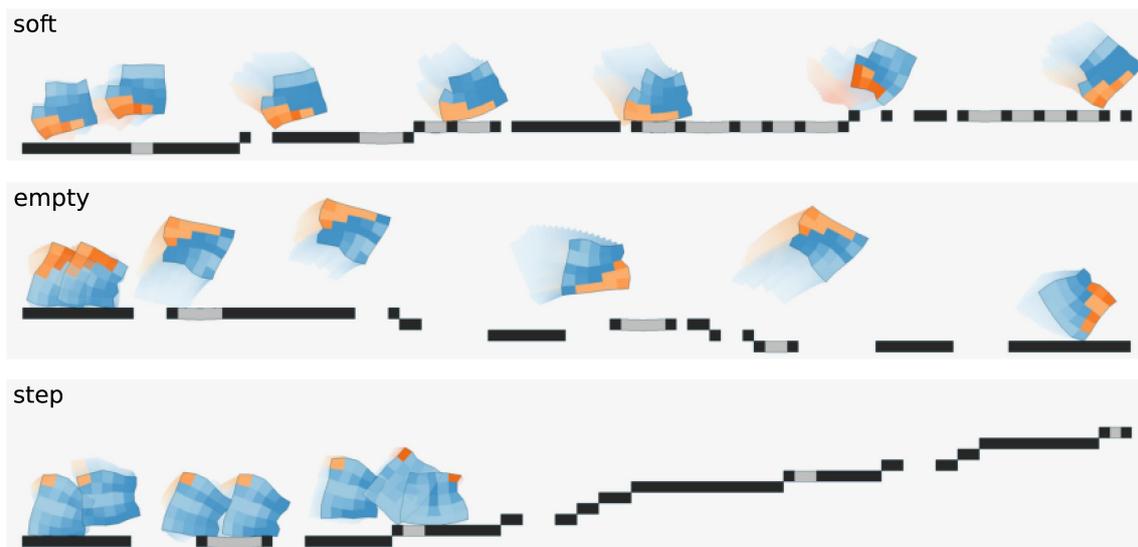


図 6.6: 固定環境に対する設計最適化で得られた最適なロボット設計と動作。

また、学習回数の観点から実験 A と実験 B を比較する。まず実験 A の提案手法において、1 つの環境に対して学習が行われるのは以下の 3 通りの場合である。

- (i) 該当の環境とペアとなっているロボットの学習をする場合。1 ループごとにペアとなっている 6 のロボットにおいて、それぞれ 4 回ずつ学習が行われる。
- (ii) 他の環境にペアとなっているロボットを転送する場合。ループ 10 回ごとに転送処理が発生し、集団内の他の 9 つの環境にそれぞれペアとなっている 6 のロボット全てに 32 回ずつ学習が行われる。ここで、該当の環境が生成された際の最初の転送に限っては、10 の環境からの転送が行われる。
- (iii) 新たに生成されたロボットを転送する場合。新しいロボットの生成はループ 20 回ごとに発生し、生成された 10 のロボットそれぞれが転送により 32 回の学習が行われる。

以上の学習は、環境が生成され集団に加わり、新たに 10 の環境が生成されて集団から外れるまでの $10 \times 60 = 600$ ループの間行われ続ける。それぞれの場合で学習が行われる回数は、(i)

$600 \times 6 \times 4 = 14,400$, (ii) $((\frac{600}{10} - 1) \times 10 + 1 \times 9) \times 6 \times 32 = 103,872$, (iii) $\frac{600}{20} \times 10 \times 32 = 9,600$
と計算できる。つまり、実験 A では 1 つの環境に対する PPO でのロボットの総合学習回数は、これらを足し合わせて 127,872 回である。実験 B では、1,000 のロボットそれぞれに 1,536 回の学習が行われるため、総合学習回数は単純に 1,536,000 回と計算できる。学習回数を比較すると、1 つの環境に対するロボット設計の探索においては従来手法に比べ、提案手法が大幅に効率的であるといえる。

実験 A で 3 つの環境全てに適応できることが示された、表 6.3 で赤枠で示したロボット設計は実験 B では得られなかった。このロボット設計であれば転送を行わずに各環境それぞれに適応できる可能性がある。そこでこのロボット設計を用いて、3 つの環境それぞれに対して実験 B での各ロボットと同様に PPO によってコントローラを学習させる実験を行った。PPO は確率的な学習を行うため、実験を 10 回試行してそのうちの最大の適応度を比較に用いる。結果の動作を図 6.7 に示す。soft では 8.65, empty では 8.59, step では 3.58 であった。soft, empty では十分に適応できているものの、step の環境には適応できないことが確認された。このロボット設計は、実験 A の結果から 3 つの環境全てに適応できる可能性を持っていることがわかっているが、1 つの環境のみに対するコントローラの学習ではその可能性を引き出すことができなかった。

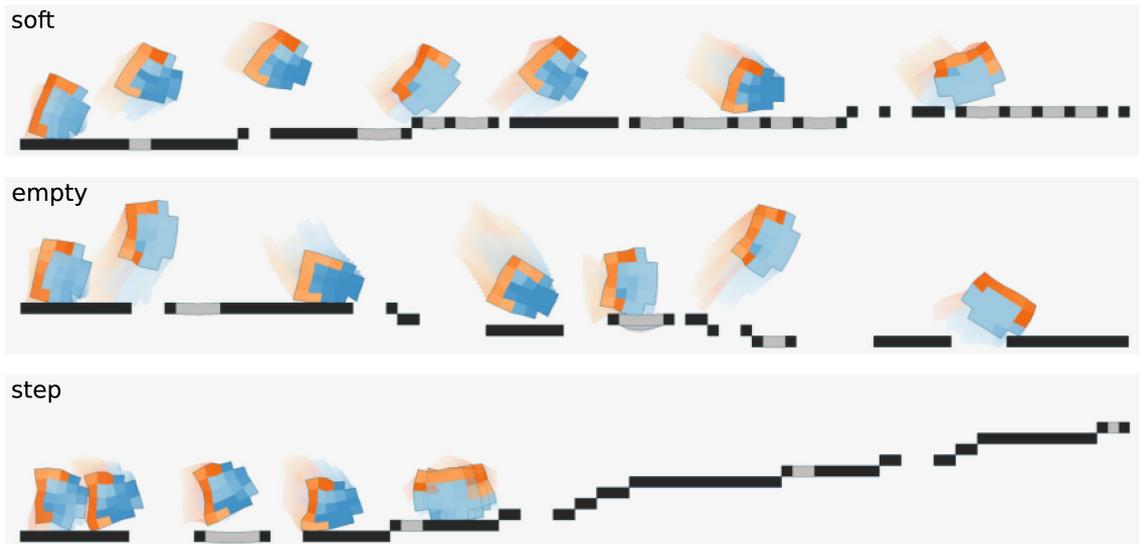


図 6.7: 固定環境に対する PPO で制御最適化で得られた動作。

比較実験の結果、固定した 1 つの環境のみに対するコントローラの学習およびロボット設計の進化では、難易度の高い環境に対しては十分に適応することができなかった。提案手法では、異なる環境間で学習したコントローラを転送することでロボットの設計が持つ動作の可能性を引き出すことが可能となり、難易度の高い環境でも適応可能なロボット設計を探索することができると示された。

6.3 環境とロボットの相互作用の分析

提案手法における環境の複雑化が、ロボット設計の進化にどのような影響を与えているのかを分析する。同時に4つ以上の環境にペアとなったことがある主要なロボットの人口の遷移を図6.8に示す。斜線部分はそれ以外のロボットの合計人口を表す。縦軸はロボット設計の累積総人口を表し、これは集団内の環境の数に比例するため、iteration 0の6から環境の数が10で最大となったiteration 600で上限の60に達する。塗りつぶしの色はロボットの絶滅したiterationを表しており、緑が濃いほど早いiterationでの絶滅、黄色は最後まで残り続けたロボットを表している。初めは橙色のボクセル（水平方向に伸縮させることができるアクチュエータボクセル）の密度が高いロボットが集団の中で大きい割合を占めていたが、1,000 iterationを超えたあたりから人口が減り始め、2,300 iterationを超えてからは、青色のボクセル（垂直方向に伸縮させることができるアクチュエータボクセル）の密度が高いロボットが台頭している。

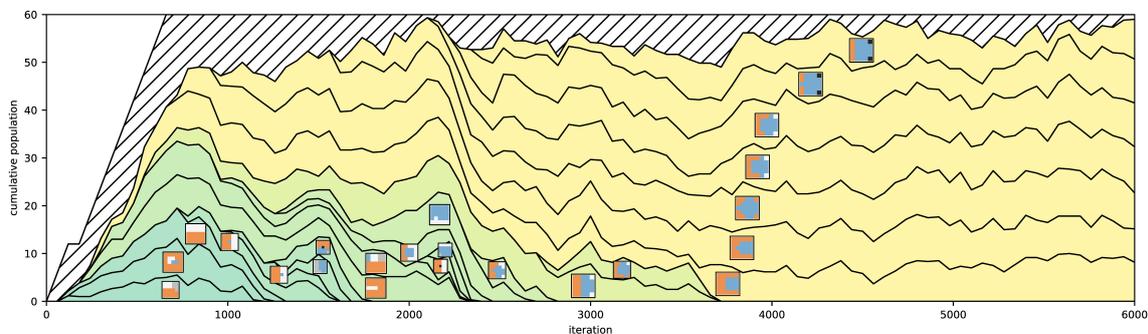


図 6.8: 主要なロボットの人口遷移。

図 6.9 に各 iteration ごとの、環境の soft ボクセルの数 (soft), empty ボクセルの数 (empty), 段差の数 (step) それぞれの集団内の平均を示す。0 iteration においては、全て rigid ボクセルで構成された平坦な環境であるために3つ全ての値が0となっている。これらの値の遷移を見ると、empty ボクセルの数と段差の数は順調に増え続けていることが確認できる。2,300 iteration 付近で、empty ボクセルの数が大幅に増えており、ロボットの人口に大きな変化が見られた時期と一致している。ロボットは初期の平坦な環境であれば、Horizontal アクチュエータボクセルを駆動することで早く移動することができたが、段差や穴がある場合には、水平方向への駆動だけでは適応できなくなる。そのため empty ボクセルの増加に合わせて、Vertical アクチュエータを持つロボット設計に進化が進んだのだと考えられる。

また、図 6.8 において、iteration が増えるにつれてロボット設計は、似た設計が集団のほとんどを占めるようになり多様性が失われている。こうした収束は、他の有望な設計への探索の可能性を閉ざしてしまうため、避けることが望ましい。収束してしまった原因として2つ考えられる。1つ目は環境が複雑化する速度である。図 6.9 から iteration 後半でも、段差の数が増え続けていることがわかるが、ロボット集団への選択圧にはなり得ていない。複雑化を

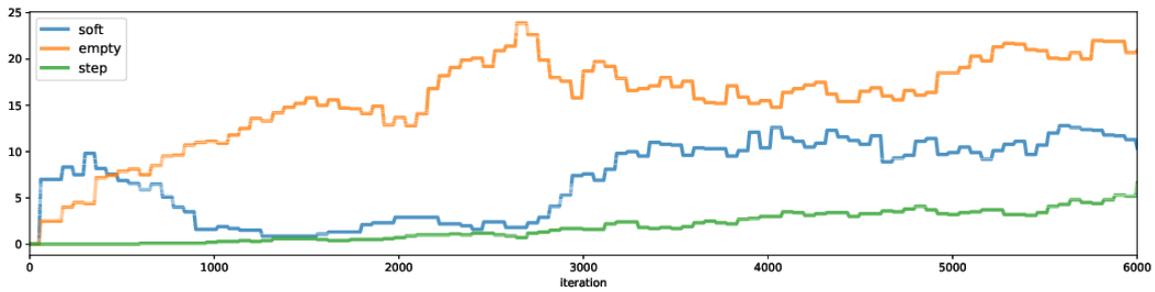


図 6.9: 環境の特徴の遷移.

進行させる速度を調整することや、壁や天井など新しい方向性への複雑化を検討する必要がある。2つ目は、新しいロボット設計と既存のロボット設計の間にあるコントローラの学習差である。新しく生成されたロボット設計は、初期の平らな環境で事前学習を行なってから集団の環境に転送され、多様な環境で学習されている既存のロボットと比較される。ここには明確な学習差があり、iteration 後半になるほどこの差は広がっていき、新しいロボット設計が集団に加わることが難しくなっている可能性がある。新しいロボット設計を集団に取り入れることで集団の収束を避けるような仕組みを検討する必要がある。

6.4 まとめ

本章の研究では、環境を複雑化させることで多様な環境とロボット設計の進化を相互作用させるアルゴリズムを提案し、実験を行なった。実験の結果、多様な環境に適応することが可能なロボット設計を探索できることが確認された。また、異なる環境間でのロボットを転送し合うことが、ロボット設計の持つ可能性を引き出し多様で複雑な環境への適応性を高めていることを比較実験により示した。

本研究では、ロボットの制御に一般的に採られるニューラルネットワークを用いてロボット全体を制御する方法を採用したが、工夫することで様々なロボットで活用できる汎用的なコントローラを獲得できる可能性がある。全体を一度に制御する方法は、シンプルで学習の速度や安定性が高いが、1つのロボット設計に対して1つのコントローラとなり、設計が異なるロボットには転用することができない。そこでもう1つの方法として、ロボットのモジュールごとにニューラルネットワークを割り当てる分散的な方法がある [41, 54]。分散的な制御方法であれば、異なる設計のロボット間でもコントローラを転用することが可能となるため、より複雑な相互作用を見られる。その結果、1つのタスクに汎用的なロボット設計だけでなく、多様なロボットで通用する汎用的なコントローラを得られる可能性がある。今後の研究ではアルゴリズムにこれらの改善を加えていく予定である。

さらに、LLM(Large Language Model)を用いることで、より効率的に複雑な環境を生成できる可能性がある [67]。この実験で用いた CPPN による足場単位での環境の生成方法では、生成できる環境に大きな制約があった。LLM を用いて言語空間を介することによってより多様

で複雑な環境を生成できることが期待される。また同様にロボットの生成にも LLM を用いる [37] ことで、より効率的で複雑なロボット設計を探索できる可能性がある。

また、実験の結果、ロボットの設計が収束してしまい多様性が失われてしまうという課題も明らかとなった。Quality Diversity アルゴリズム [48] に見られるように、特徴が似ているグループ内でロボットの競争を行う方法を用いることでこの課題を解決することが期待される。提案手法では単純に適応度の高いロボットを集団に残したことで、ロボットの集団は似た設計に収束してしまった。ロボットを転送する際に置き換えの対象とするロボットを明示的にわけることで、特徴の異なるロボットを保護し多様性を維持することが可能となる。

最後に、提案手法は、環境の複雑化とロボットの進化と学習を相互作用させるものであり、これを基盤にした実験とその詳細な分析を通じて、生物学や人工生命の分野への新しい知見を提供する可能性があると考えている。生物学や人工生命の分野では、生物がどのようにして多様な環境に適応して進化してきたのか、またその適応のメカニズムは何かという疑問が持たれてきた。提案手法を応用し、生物の適応や進化のシミュレーションを行うことで、生物の進化や適応のメカニズムに関する新しい知見を提供できるのではないかと考えている。

第7章 結論

7.1 まとめ

コンピュータ基盤の発達は、近年著しい進展を遂げており、その結果として高度な物理シミュレーションが可能となっている。この進展により、従来の手法では困難であった複雑な環境やタスクに柔軟に対応するロボットの開発が現実味を帯びてきた。特に、ソフトロボティクス分野では、シミュレーション技術を活用することで、より精密かつ多様な設計が可能となり、複雑な動作を持つロボットの開発が進められている。この中で、ボクセルベースソフトロボット (VSRs) は特に注目されている研究領域である。ボクセルベースのアプローチは、三次元空間における各ボクセルの特性を詳細にシミュレートし、相互作用を最適化することで、柔軟な形態変化や動作を実現する。この技術は、ロボットの適応能力を飛躍的に向上させ、従来のロボットでは対応できなかった複雑な環境やタスクに対するソリューションを提供する。これにより、災害救助、医療、探査といった多岐にわたる分野での応用が期待され、VSRsの研究は、その可能性を最大限に引き出すための重要な取り組みとして注目されている。

VSRsの開発においては、最適な設計と制御を達成することが最も重要な課題である。VSRsは、複雑で動的な現実世界の環境において、柔軟性と適応性を持つことが求められている。しかし、これらのロボットが様々な環境やタスクに対して効果的に機能するためには、単にロボットの形状や素材を工夫するだけでは不十分である。最適な設計とは、ロボットの物理的特性と構造が、その目的に合致し、効率かつ効果的に機能するようにすることである。一方、最適な制御とは、ロボットが直面する多様な状況に応じて、適切な動作を遂行できるようにするアルゴリズムやシステムを指す。これらの二つの要素は相互に関連しており、一方が不十分であれば他方のパフォーマンスにも悪影響を及ぼす。さらに現実世界では、環境の変動や予測不可能な要素が存在するため、ロボットの設計と制御がこれに対して頑健であることが必要不可欠である。具体的には、ロボットが障害物を避けたり、急な地形変化に対応したり、予期せぬ負荷に耐えることができるような設計と制御が求められる。これを実現するためには、高度なシミュレーション技術を駆使し、設計と制御の両面で最適化を行うことが必要であり、これが現在のVSRs研究における主要な課題となっている。

そこで本研究では、2Dシミュレーションを用いて、より効果的なVSRsの開発を目指した研究を行った。具体的には、VSRsの開発において重要となる、設計、制御、環境の3つの観点から、以下の3つを研究課題として設定した。

- 設計の特徴量を定義し、その特徴量に基づいて設計を最適化するアルゴリズムを構築することで、タスク遂行に寄与している特徴量を明らかにする。

- 異なる特性を持った制御の最適化手法を実験により比較し、設計との共同最適化に対する影響や効率性について検討する。
- 動的に変化する環境に対して設計と制御を共同最適化するアルゴリズムを構築することで、環境の変化に頑健な設計を明らかにする。

まず、効果的な VSRs の設計開発を行うために、タスク遂行性に寄与している特徴量を明らかにするフレームワークを提案した。従来の設計最適化手法は、膨大なグリッド状の設計空間から最適な設計を探索する。しかし、これらの方法では、設計空間全体を理解することは難しい。そこで本研究では、1つ1つのボクセルではなく、いくつかのボクセルで構成される部品単位の組み合わせで可能な設計で、設計空間を再定義する。そして、直接的に解釈可能な設計の特徴量を定義し、この特徴量空間で設計を探索した。これにより、どのような設計がタスクの遂行性が高いかを理解することが可能となった。結果として、部品の設計上での位置情報がタスク遂行に寄与していることが明らかになった。この知見は、そのまま設計開発の具体的な指針となり効率化を促す。また、このアプローチは、特定の部品についての検証を提供するため、さらなる VSRs の設計理解に貢献できる。

次に、VSRs の動作性能を効果的に引き出し、タスク遂行性を高めることができる制御の最適化アルゴリズムについて検討した。これまでの VSRs の最適化研究では、ニューラルネットワークを用いた制御方法が主流であったが、そのパラメータの最適化手法は研究によって異なっていた。本研究では、主要な方法論である深層強化学習と2種類の進化的アルゴリズムによって、複数のタスクと VSR 設計に対して、制御最適化実験を行い比較した。その結果、強化学習アルゴリズムが最適化性能と計算効率において性能に優れていることが示されたが、一部のタスクや設計においては十分にその動作性を引き出すことができず、進化的アルゴリズムが有効であった。最適化手法が有効に働くタスクや設計に傾向を明らかにしたことは、今後の VSRs の設計開発において、目的とするタスクや設計の制約によって、適切な制御最適化手法の検討のために重要である。

最後に、複雑で多様な環境でも頑健な性能を持つ設計に最適化するアルゴリズムを提案した。従来の VSRs 研究では、シミュレーション上での1つの固定された環境設定が用いられてきた。しかしながら、単純な環境に対する最適化では、過適合した設計が得られる可能性がある。そこで本研究では、環境の複雑化とロボットの制御の最適化を相互作用させることで、多様な環境に対して頑健な制御を得る POET アルゴリズムをもとに、新しく設計も同時に最適化するアルゴリズムを提案した。実験の結果、複雑な環境でも頑健に適応することができる設計を得ることができると示した。また、従来の1つの環境に対する最適化との比較実験により、多様な環境での学習成果を利用することで、複雑な環境にも有効な設計及び制御を得ることができると示した。この手法で得られる設計は多様で複雑な環境によって最適化されていることから、現実の不確実な環境において有効である可能性が高い。

本研究では、VSRs の2次元シミュレーションを用いて、設計、制御、環境の3つの観点から実験をおこなった。今回の研究で得られた、設計の特徴や制御の最適化アルゴリズムに関する知見は、VSRs の設計開発の際に大きな指針となることが期待される。これらの知見を利

用することで、タスクごとに適切な設計の制約や制御の最適化を選択することができ、今までよりも効果的な VSRs の最適化が可能となる。また、多様な環境での学習がより頑健で適応的な設計への最適化を促すこと示したことは、環境を考慮することで今後のより複雑な VSRs を効果的に開発することにつながる。

また、本研究で提案した VSRs の最適化手法を拡張することで、さらなる知見を得られる可能性がある。例えば、設計特徴に基づく設計の最適化では、タスクや事前知識に合わせて適切に特徴量を設定することで、より効果的な最適化につながる可能性がある。さらに、その結果を分析することで、タスクや設計に関してより深い洞察が提供されることが期待される。さらに、環境の複雑化においても、設計の最適化や制御の最適化の手法は容易に交換可能であり、今後の VSRs の進展で得られる知見を取り込むことで、また新たな示唆を得られる。

7.2 今後の課題

本研究では、VSRs の設計と制御に関わる最適化アルゴリズムについて検討した。しかし、その際のパラメータや設定は、先行研究での設定や標準的なものを使用した。パラメータによっても、最適化の性能や方向性は大きく異なることが当然想定される。特に、従来研究でも、手法ごとの比較が多く、詳細なパラメータについての検討は十分になされていないことが多い。そのため、パラメータの違いによる影響の違いを詳細に分析することは、VSRs の理解にもつながり、より効果的な最適化手法の開発につながることを期待される。

また本研究では、VSRs の 2 次元シミュレーションを用いた。VSRs の最適化研究において、2 次元シミュレーションは初期段階の設計と制御の有効性を検証するための重要なツールである。2 次元シミュレーションは計算資源の消費が少なく、迅速に多くの試行錯誤を行うことができるため、基礎的な理解を深める上で非常に有用である。この段階では、ロボットの基本的な動作パターンや簡単なタスクへの対応能力を評価し、最適な設計パラメータや制御アルゴリズムを探索することができる。しかし、2 次元の世界では現実の複雑な環境を完全に再現することはできず、得られた結果を現実世界にそのまま適用するには限界がある。

そのため、次のステップとして、3 次元シミュレーションが必要である。3 次元シミュレーションは、現実世界の物理現象をより忠実に再現することができ、VSRs が実際に遭遇する多くの課題に対応するための試験環境を提供する。具体的には、3 次元シミュレーションでは、重力や摩擦、衝突などの要素を含めた複雑な動作解析が可能である。これにより、ロボットの設計と制御が現実世界の条件下でどのように機能するかをより正確に評価できる。また、3 次元シミュレーションを通じて、ロボットの動作の安定性やエネルギー効率、耐久性などの重要な性能指標を詳細に分析することができる。これらの分析結果を基に、さらに設計と制御の最適化を進めることができる。

最終的には、現実世界での実験が不可欠である。シミュレーションは多くの現象を予測することができるが、実際の環境では予測不能な要素が多々存在する。例えば、材料の微細な不均一性や環境条件の変動、人為的な干渉など、シミュレーションでは再現しきれない問題が発生する可能性がある。現実世界での実験を通じて、これらの要素がロボットの性能に与

える影響を評価し、設計と制御の改善を図ることが求められる。また、現実の環境で実験を行うことで、ロボットが実際にどのように動作し、タスクを遂行するかを確認することができる。これにより、最終的な実用化に向けた信頼性の高いデータを収集し、実際の応用に適したロボットの開発を加速することができる。したがって、2次元シミュレーション、3次元シミュレーション、そして現実世界での実験という段階的なアプローチが、VSRsの現実世界への転用を成功させるために重要なプロセスである。

謝辞

この度の学位論文作成にあたり、多くの方々のご指導、ご支援を賜りましたことに深く感謝申し上げます。まず、指導教官である岡瑞起准教授には、研究の方向性から論文執筆に至るまで多大なるご指導を賜りました。心より感謝申し上げます。教授の卓越した指導と激励により、本研究を遂行することができました。本論文の副査を心良く引き受けていただき、的確なご指摘を頂いた、筑波大学・加藤和彦教授，同大学・亀山啓輔教授，同大学・天笠俊之教授，同大学・Aranha Claus 准教授，名古屋大学鈴木麗璽准教授に深く感謝いたします。皆様のご助言がなければ、本研究の完成は成し得なかったでしょう。また、実験や議論に際して多大なるご協力をいただいた研究室の皆様にも感謝いたします。そして、本論文の執筆において、OpenAI の ChatGPT には文章の校正や構成に関するアドバイスをいただきました。ChatGPT の支援により、論文の品質向上に大いに役立ちましたことをここに感謝いたします。最後に、私の学業を支えてくれた家族に感謝いたします。家族の理解と支えがあったからこそ、ここまで頑張ることができました。心より感謝の意を表します。本論文を通じてお世話になったすべての方々に、改めて深く感謝申し上げます。

参考文献

- [1] Shuang Ao, Tianyi Zhou, Guodong Long, Xuan Song, and Jing Jiang. “Curriculum Reinforcement Learning via Morphology-Environment Co-Evolution”. *CoRR* (2023).
- [2] Joshua E Auerbach and Joshua C Bongard. “On the relationship between environmental and morphological complexity in evolved robots”. *Proceedings of the 14th annual conference on Genetic and evolutionary computation*. 2012, pp. 521–528.
- [3] Joshua Evan Auerbach and Josh C. Bongard. “Environmental Influence on the Evolution of Morphological Complexity in Machines”. *PLoS Computational Biology* 10 (2014).
- [4] Robert Baines, Sree Kalyan Patiballa, Joran Booth, Luis Ramirez, Thomas Sipple, Andonny Garcia, Frank Fish, and Rebecca Kramer-Bottiglio. “Multi-environment robotic transitions through adaptive morphogenesis”. *Nature* 610.7931 (2022), pp. 283–289.
- [5] Jagdeep Bhatia, Holly Jackson, Yunsheng Tian, Jie Xu, and Wojciech Matusik. “Evolution gym: A large-scale benchmark for evolving soft robots”. *Advances in Neural Information Processing Systems* 34 (2021), pp. 2201–2214.
- [6] Jonathan C Brant and Kenneth O Stanley. “Minimal criterion coevolution: a new approach to open-ended search”. *Proceedings of the Genetic and Evolutionary Computation Conference*. 2017, pp. 67–74.
- [7] Leo Breiman. “Random forests”. *Machine learning* 45 (2001), pp. 5–32.
- [8] Nicholas Cheney, Josh Bongard, Vytas SunSpiral, and Hod Lipson. “On the difficulty of co-optimizing morphology and control in evolved virtual creatures”. *Artificial life conference proceedings*. 2016, pp. 226–233.
- [9] Nick Cheney, Josh Bongard, and Hod Lipson. “Evolving Soft Robots in Tight Spaces”. *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. 2015, pp. 935–942.
- [10] Nick Cheney, Josh Bongard, Vytas SunSpiral, and Hod Lipson. “Scalable co-optimization of morphology and control in embodied machines”. *Journal of The Royal Society Interface* 15.143 (2018), p. 20170937.
- [11] Nick Cheney, Robert MacCurdy, Jeff Clune, and Hod Lipson. “Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding”. *ACM SIGEVOLUTION* 7.1 (2014), pp. 11–23.

- [12] HeeSun Choi, Cindy Crump, Christian Duriez, Asher Elmquist, Gregory Hager, David Han, Frank Hearl, Jessica Hodgins, Abhinandan Jain, Frederick Leve, Chen Li, Franziska Meier, Dan Negrut, Ludovic Righetti, Alberto Rodriguez, Jie Tan, and Jeff Trinkle. “On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward”. *Proceedings of the National Academy of Sciences* 118.1 (2021), e1907856118.
- [13] Sayak Ray Chowdhury and Aditya Gopalan. “On Batch Bayesian Optimization”. *CoRR* (2019).
- [14] Francesco Corucci, Nick Cheney, Francesco Giorgio-Serchi, Josh Bongard, and Cecilia Laschi. “Evolving soft locomotion in aquatic and terrestrial environments: effects of material properties and environmental transitions”. *Soft robotics* 5.4 (2018), pp. 475–495.
- [15] Siva Krishna Dasari, Abbas Cheddad, and Petter Andersson. “Random forest surrogate models to support design space exploration in aerospace use-case”. *Artificial Intelligence Applications and Innovations*. Springer. 2019, pp. 532–544.
- [16] Heng Dong, Junyu Zhang, and Chongjie Zhang. “Leveraging Hyperbolic Embeddings for Coarse-to-Fine Robot Design”. *CoRR* (2023).
- [17] Medvet Eric, Bartoli Alberto, De Lorenzo Andrea, and Seriani Stefano. “2D-VSR-Sim: A simulation tool for the optimization of 2-D voxel-based soft robots”. *SoftwareX* 12 (2020), p. 100573.
- [18] Andrea Ferigo, Giovanni Iacca, Eric Medvet, and Federico Pigozzi. “Evolving Hebbian learning rules in voxel-based soft robots”. *IEEE Transactions on Cognitive and Developmental Systems* 15.3 (2022), pp. 1536–1546.
- [19] Jerome H Friedman. “Greedy function approximation: a gradient boosting machine”. *Annals of statistics* (2001), pp. 1189–1232.
- [20] Jahan Zeb Gul, Memoon Sajid, Muhammad Muqeet Rehman, Ghayas Uddin Siddiqui, Imran Shah, Kyung-Hwan Kim, Jae-Wook Lee, and Kyung Hyun Choi. “3D printing for soft robotics - a review”. *Science and Technology of Advanced Materials* 19.1 (2018), pp. 243–262.
- [21] David Ha. “Reinforcement Learning for Improving Agent Design”. *Artificial Life* 25.4 (2019), pp. 352–365.
- [22] Dorothea Heiss-Czedik. “An Introduction to Genetic Algorithms.” *Artificial Life* 3 (1997), pp. 63–65.
- [23] Ang Kiam Heong, Chong Gregory, and Li Yun. “PID control system analysis, design, and technology”. *IEEE transactions on control systems technology* 13.4 (2005), pp. 559–576.
- [24] Jonathan Hiller and Hod Lipson. “Automatic design and manufacture of soft robots”. *IEEE Transactions on Robotics* 28.2 (2011), pp. 457–466.

- [25] Kazuya Horibe, Kathryn Walker, and Sebastian Risi. “Regenerating soft robots through neural cellular automata”. *Genetic Programming*. 2021, pp. 36–50.
- [26] Frank Hutter, Lin Xu, Holger H Hoos, and Kevin Leyton-Brown. “Algorithm runtime prediction: Methods & evaluation”. *Artificial Intelligence* 206 (2014), pp. 79–111.
- [27] Hiller Jonathan and Lipson Hod. “Dynamic simulation of soft multimaterial 3d-printed objects”. *Soft robotics* 1.1 (2014), pp. 88–101.
- [28] Legrand Julie, Terryn Seppe, Roels Ellen, and Vanderborght Bram. “Reconfigurable, multi-material, voxel-based soft robots”. *IEEE Robotics and Automation Letters* 8.3 (2023), pp. 1255–1262.
- [29] Arulkumaran Kai, Deisenroth Marc Peter, Brundage Miles, and Bharath Anil Anthony. “Deep reinforcement learning: A brief survey”. *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38.
- [30] Kirthevasan Kandasamy, Akshay Krishnamurthy, Jeff Schneider, and Barnabás Póczos. “Parallelised Bayesian optimisation via Thompson sampling”. *International conference on artificial intelligence and statistics*. 2018, pp. 133–142.
- [31] Jungtaek Kim and Seungjin Choi. “On uncertainty estimation by tree-based surrogate models in sequential model-based optimization”. *International Conference on Artificial Intelligence and Statistics*. 2022, pp. 4359–4375.
- [32] Tomoya Kimura, Ziqiao Jin, Ryuma Niiyama, and Yasuo Kuniyoshi. “Evolving soft robots to execute multiple tasks with combined-cppn-neat”. *2016 IEEE/SICE International Symposium on System Integration (SII)*. 2016, pp. 409–414.
- [33] Oleg Klimov. *Bipedalwalkerhardcore-v2*. https://gymnasium.farama.org/environments/box2d/bipedal_walker/.
- [34] Kristián Kovalský and George Palamas. “Neuroevolution vs reinforcement learning for training non player characters in games: The case of a self driving car”. *International Conference on Intelligent Technologies for Interactive Entertainment*. 2020, pp. 191–206.
- [35] Shao Kun, Tang Zhentao, Zhu Yuanheng, Li Nannan, and Zhao Dongbin. “A survey of deep reinforcement learning in video games”. *CoRR* (2019).
- [36] Mia-Katrin Kvalsund, Kyrre Glette, and Frank Veenstra. “Centralized and decentralized control in modular robots and their effect on morphology”. *Artificial Life Conference Proceedings* 34. Vol. 2022. 1. 2022, p. 49.
- [37] Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O Stanley. “Evolution through large models”. *Handbook of Evolutionary Machine Learning*. 2023, pp. 331–366.

- [38] Qiaohao Liang, Aldair Gongora, Zekun Ren, Armi Tiihonen, Zhe Liu, Shijing Sun, James De-neault, Daniil Bash, Flore Mekki-Berrada, Saif Khan, Kedar Hippalgaonkar, Benji Maruyama, Keith Brown, John III, and Tonio Buonassisi. “Benchmarking the performance of Bayesian optimization across multiple experimental materials science domains”. 7 (2021), p. 188.
- [39] Gilles Louppe, Louis Wehenkel, Antonio Sutera, and Pierre Geurts. “Understanding variable importances in forests of randomized trees”. *Advances in neural information processing systems* 26 (2013).
- [40] Wilson Margaret. “Six views of embodied cognition”. *Psychonomic bulletin & review* 9 (2002), pp. 625–636.
- [41] Eric Medvet, Alberto Bartoli, Andrea De Lorenzo, and Giulio Fidel. “Evolution of distributed neural controllers for voxel-based soft robots”. *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pp. 112–120.
- [42] Eric Medvet, Alberto Bartoli, Federico Pigozzi, and Marco Rochelli. “Biodiversity in evolved voxel-based soft robots”. *Proceedings of the Genetic and Evolutionary Computation Conference*. 2021, pp. 129–137.
- [43] Eric Medvet, Giorgia Nadizar, and Federico Pigozzi. “On the impact of body material properties on neuroevolution for embodied agents: the case of voxel-based soft robots”. *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 2022, pp. 2122–2130.
- [44] Alican Mertan and Nick Cheney. “Modular Controllers Facilitate the Co-Optimization of Morphology and Control in Soft Robots”. *Proceedings of the Genetic and Evolutionary Computation Conference*. 2023, pp. 174–183.
- [45] Karine Miras, Eliseo Ferrante, and A. E. Eiben. “Environmental influences on evolvable robots”. *PLoS ONE* 15.5 (2020), pp. 1–23.
- [46] Jonas Mockus. “On Bayesian Methods for Seeking the Extremum.” *Optimization Techniques*. Vol. 27. 1974, pp. 400–404.
- [47] Alexander Mordvintsev, Ettore Randazzo, Eyvind Niklasson, and Michael Levin. “Growing neural cellular automata”. *Distill* 5.2 (2020), e23.
- [48] Jean-Baptiste Mouret and Jeff Clune. “Illuminating search spaces by mapping elites”. *CoRR* (2015).
- [49] Giorgia Nadizar, Eric Medvet, Kathryn Walker, and Sebastian Risi. “A fully-distributed shape-aware neural controller for modular robots”. *Proceedings of the Genetic and Evolutionary Computation Conference*. 2023, pp. 184–192.
- [50] Jørgen Nordmoen, Frank Veenstra, Kai Olav Ellefsen, and Kyrre Glette. “Quality and diversity in evolutionary modular robotics”. *2020 IEEE symposium series on computational intelligence (SSCI)*. 2020, pp. 2109–2116.

- [51] Tønnes F Nygaard, Charles P Martin, David Howard, Jim Torresen, and Kyrre Glette. “Environmental adaptation of robot morphology and control through real-world evolution”. *Evolutionary Computation* 29.4 (2021), pp. 441–461.
- [52] Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. “Evolving curricula with regret-based environment design”. *International Conference on Machine Learning*. 2022, pp. 17473–17498.
- [53] Federico Pigozzi, Federico Julian Camerota Verdù, and Eric Medvet. “How the morphology encoding influences the learning ability in body-brain co-optimization”. *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1045–1054.
- [54] Federico Pigozzi, Yujin Tang, Eric Medvet, and David Ha. “Evolving modular soft robots without explicit inter-module communication using local self-attention”. *Proceedings of the Genetic and Evolutionary Computation Conference*. 2022, pp. 148–157.
- [55] Kiran B Ravi, Sobh Ibrahim, Talpaert Victor, Mannion Patrick, Al Sallab Ahmad A, Yogamani Senthil, and Patrick Pérez. “Deep reinforcement learning for autonomous driving: A survey”. *IEEE Transactions on Intelligent Transportation Systems* 23.6 (2021), pp. 4909–4926.
- [56] Kriegman Sam, Nasab Amir Mohammadi, Shah Dylan, Hannah Steele, Branin Gabrielle, Levin Michael, Bongard Josh, and Kramer-Bottiglio Rebecca. “Scalable sim-to-real transfer of soft robot designs”. *2020 3rd IEEE international conference on soft robotics (RoboSoft)*. 2020, pp. 359–366.
- [57] Kim Sangbae, Laschi Cecilia, and Trimmer Barry. “Soft robotics: a bioinspired evolution in robotics”. *Trends in biotechnology* 31.5 (2013), pp. 287–294.
- [58] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. “Proximal policy optimization algorithms”. *CoRR* (2017).
- [59] Karl Sims. “Evolving Virtual Creatures”. *Proc. of the 21st Annual Conference on Computer Graphics and Interactive Techniques*. 1994, pp. 15–22.
- [60] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. “Practical Bayesian Optimization of Machine Learning Algorithms”. *Advances in Neural Information Processing Systems*. Vol. 25. 2012.
- [61] Junru Song, Yang Yang, Wei Peng, Weien Zhou, Feifei Wang, and Wen Yao. “MorphVAE: Advancing Morphological Design of Voxel-Based Soft Robots with Variational Autoencoders”. *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 9. 2024, pp. 10368–10376.
- [62] Christina Spanellis, Brooke Stewart, and Geoff Nitschke. “The Environment and Body-Brain Complexity”. *Proceedings of the Genetic and Evolutionary Computation Conference*. 2021, pp. 138–145.

- [63] Kenneth O Stanley. “Compositional pattern producing networks: A novel abstraction of development”. *Genetic programming and evolvable machines* 8 (2007), pp. 131–162.
- [64] Kenneth O Stanley, David B D’Ambrosio, and Jason Gauci. “A hypercube-based encoding for evolving large-scale neural networks”. *Artificial life* 15.2 (2009), pp. 185–212.
- [65] Kenneth O Stanley and Risto Miikkulainen. “Evolving neural networks through augmenting topologies”. *Evolutionary computation* 10.2 (2002), pp. 99–127.
- [66] Emma Hjellbrekke Stensby, Kai Olav Ellefsen, and Kyrre Glette. “Co-optimising Robot Morphology and Controller in a Simulated Open-Ended Environment”. *Applications of Evolutionary Computation*. 2021, pp. 34–49.
- [67] Shyam Sudhakaran, Miguel González-Duque, Matthias Freiberger, Claire Glanois, Elias Najarro, and Sebastian Risi. “Mariogpt: Open-ended text2level generation through large language models”. *Advances in Neural Information Processing Systems* 36 (2024).
- [68] Jacopo Talamini, Eric Medvet, Alberto Bartoli, and Andrea De Lorenzo. “Evolutionary synthesis of sensing controllers for voxel-based soft robots”. *Artificial life conference proceedings*. 2019, pp. 574–581.
- [69] Jacopo Talamini, Eric Medvet, and Stefano Nichele. “Criticality-driven evolution of adaptable morphologies of voxel-based soft-robots”. *Frontiers in Robotics and AI* 8 (2021), p. 673156.
- [70] Fabio Tanaka and Claus Aranha. “Co-evolving morphology and control of soft robots using a single genome”. *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2022, pp. 1235–1242.
- [71] Leonardo Thurler, José Montes, Rodrigo Veloso, Aline Paes, and Esteban Clua. “AI Game Agents Based on Evolutionary Search and (Deep) Reinforcement Learning: A Practical Analysis with Flappy Bird”. *Proc. of Entertainment Computing – ICEC 2021*. 2021, pp. 196–208.
- [72] Leonardo Thurler, José Montes, Rodrigo Veloso, Aline Paes, and Esteban Clua. “AI Game Agents Based on Evolutionary Search and (Deep) Reinforcement Learning: A Practical Analysis with Flappy Bird”. *Entertainment Computing – ICEC 2021*. 2021, pp. 196–208.
- [73] Wu Tianyu, He Shizhu, Liu Jingping, Sun Siqi, Liu Kang, Han Qing-Long, and Tang Yang. “A brief overview of ChatGPT: The history, status quo and potential future development”. *IEEE/CAA Journal of Automatica Sinica* 10.5 (2023), pp. 1122–1136.
- [74] Emanuel Todorov, Tom Erez, and Yuval Tassa. “Mujoco: A physics engine for model-based control”. *2012 IEEE/RSJ international conference on intelligent robots and systems*. 2012, pp. 5026–5033.
- [75] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. “Attention is all you need”. *Advances in neural information processing systems* 30 (2017).

- [76] Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O Stanley. “Poet: open-ended coevolution of environments and their optimized solutions”. *Proceedings of the Genetic and Evolutionary Computation Conference*. 2019, pp. 142–151.
- [77] Rui Wang, Joel Lehman, Aditya Rawal, Jiale Zhi, Yulun Li, Jeffrey Clune, and Kenneth Stanley. “Enhanced poet: Open-ended reinforcement learning through unbounded invention of learning challenges and their solutions”. *International Conference on Machine Learning*. 2020, pp. 9940–9951.
- [78] Tsun-Hsuan Wang, Pingchuan Ma, Andrew Everett Spielberg, Zhou Xian, Hao Zhang, Joshua B Tenenbaum, Daniela Rus, and Chuang Gan. “Softzoo: A soft robot co-design benchmark for locomotion in diverse environments”. *CoRR* (2023).
- [79] Yuxing Wang, Shuang Wu, Haobo Fu, Qiang Fu, Tiantian Zhang, Yongzhe Chang, and Xueqian Wang. “Curriculum-based co-design of morphology and control of voxel-based soft robots”. *The Eleventh International Conference on Learning Representations*. 2023.
- [80] Yuxing Wang, Shuang Wu, Tiantian Zhang, Yongzhe Chang, Haobo Fu, Qiang Fu, and Xueqian Wang. “PreCo: Enhancing Generalization in Co-Design of Modular Soft Robots via Brain-Body Pre-Training”. *Conference on Robot Learning*. 2023, pp. 478–498.
- [81] K.E. Watkins, T. Paus, J.P. Lerch, A. Zijdenbos, D.L. Collins, P. Neelin, J. Taylor, K.J. Worsley, and A.C. Evans. “Structural Asymmetries in the Human Brain: a Voxel-based Statistical Analysis of 142 MRI Scans”. *Cerebral Cortex* 11.9 (2001), pp. 868–877.
- [82] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*. Vol. 2. 3. 2006.
- [83] James Wilson, Frank Hutter, and Marc Deisenroth. “Maximizing acquisition functions for Bayesian optimization”. *Advances in Neural Information Processing Systems*. Vol. 31. 2018.
- [84] Jia Wu, Xiu-Yun Chen, Hao Zhang, Li-Dong Xiong, Hang Lei, and Si-Hao Deng. “Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimization”. *Journal of Electronic Science and Technology* 17.1 (2019), pp. 26–40.
- [85] Shangdong Zhang and Osmar R Zaiane. “Comparing deep reinforcement learning and evolutionary methods in continuous control”. *CoRR* (2017).