Vol.2021-ARC-244 No.23
Vol.2021-SLDM-194 No.23
Vol.2021-EMB-56 No.23
2021/3/25

IoT 仮想環境「箱庭」による自律移動ロボットの制御パラメータの自動探索手法

立川 悠輝[1,a)]　福田 竜也[2]　森 崇[3]　高瀬 英希[1,4]

概要：「箱庭」とは，IoT/自動運転時代の包括的な仮想シミュレーション環境である．箱庭プロトタイプモデルのひとつとして，マイコンシミュレータ Athrill および物理演算エンジン Unity で構成される単体ロボットシミュレータが成果物として公開されている．本研究では，単体ロボットシミュレータを活用した自律移動ロボットの開発において，その制御パラメータを自動探索する手法を提案する．まず，単体ロボットシミュレータの通信方式を拡張し，シミュレータ上のテストを制御できるようにする．そして，自律移動ロボットの制御パラメータを変更してテストを繰り返し実行することで，適切な値の探索を実現する．提案手法の有効性を議論するため，ステアリング量の計算のための PID パラメータに提案手法を適用した．その結果，コースの周回時間を最短にできる制御パラメータを探索できることを確認した．

A searching method for control parameters of the autonomous mobile robot using the IoT virtual environment “Hakoniwa”

1. はじめに

IoT（Internet of Things）システムの発展と普及が，近年ますます進んでいる．品質の高い IoT システムを実現するためには，多種かつ大量の計算機器をネットワークによって密接に絡め合うことが不可欠である．このような大規模な IoT システムは，我々の社会生活の利便性を向上させる一方で，開発時の課題が多く存在する．まず，IoT システムは，先述の通り多くの情報機器が複雑に絡み合うため，問題の発生時には問題発生経路および原因が複雑になり，またデバッグが非常に困難となってくる．さらには，システムの構築段階から想定した場合，モジュールの最適な組み合わせの発見が困難であること，および，開発に伴って頻繁に変動する要件に対してシステムの仕様を実際にその都度変更することの負担も大きくなる．

このような課題に対する解決のアプローチとして，我々は，IoT システム向けの仮想シミュレーション環境である「箱庭」の研究開発を進めている [1], [2]．箱庭では，仮想環境上に様々なソフトウェアおよびサービスを展開し，自由に組み合わせて実証実験を行うことで，複雑なシステムの事象および状態を任意の精度で検証できるようにすることを目指している．箱庭 WG における成果物のひとつとして，ET ソフトウェアデザインロボットコンテスト（ET ロボコン）[3][4] を題材とした単体ロボット向けシミュレータがあり，これをプロトタイプモデルとして公開されている．本シミュレータではマイコンシミュレータおよび物理シミュレータ間で通信を行い，仮想環境上にて 1 つのマイコンで 1 台のロボットを制御することを実現できる．

単体ロボット向けシミュレータの題材である自律移動ロボットにおいては，本番環境に導入されるまでに何度もトライアンドエラーを繰り返すことで開発を進めていくことが多い．その開発の中では，頻繁にシステム要件が変更されることも多く，その都度要求に応えて仕様を変更する必要がある．特に，ロボットの制御における最適なパラメータの値を計算で導き出すことは難しい．すなわち，実際にロボットを制御する各機器に対してそれぞれテストを行うだけでなく，開発対象のロボットを導入される環境に合わせて繰り返しテストを行うことで，効率的により良い制御パラメータを導くことが重要となってくる．

そこで本研究では，自律移動ロボットの開発において，

[1] 京都大学
[2] インテック
[3] 永和システムマネジメント
[4] JST さきがけ
[a)] emb@lab3.kuis.kyoto-u.ac.jp

Vol.2021-ARC-244 No.23
Vol.2021-SLDM-194 No.23
Vol.2021-EMB-56 No.23
2021/3/25

要件を満たす最適な制御パラメータを探索できるようにすることを目指す．繰り返し何度でも容易に試行できる仮想環境の利点を活かし，繰り返しテストを実行し評価することで，頻繁に変更されうるシステム要件に効率的に対応できるようにする．

本研究では，箱庭を活用し，与えられた要件に対応する制御パラメータを自動的に探索する手法を提案する．まず，箱庭における通信方式を拡張し，各シミュレータの動作を管理するモジュールを設計し，テストを繰り返し自動で実行できるようにする．さらに，ロボット制御のパラメータに対しテスト結果を報酬として設定し，これに沿った目的関数に従った最適化を実現する．これによって，テストシナリオにおいて与えられた条件や基準を満たす制御パラメータを探索する．本研究では，最適化の手段としてベイズ最適化を採用する．提案手法によって，開発においてプログラム変更の度にそのプログラムが正しく動作するかどうかを確認することが可能となる．さらに，何度もテストを行って制御パラメータの設定値を評価することで，自律移動ロボットのより良い制御パラメータを開発者へフィードバックすることができるようになる．

本稿の構成は，次のとおりである．まず 2 章では，箱庭および関連研究を紹介する．次に 3 章で，箱庭上でロボットの制御パラメータを探索する提案手法ついて説明する．4 章では，提案手法の適用事例を示し，その有効性を議論する．最後に，5 章で本研究のまとめと今後の方針を示す．

2. 準備

2.1 箱庭

箱庭は IoT/自動運転システム開発のための仮想環境であり，仮想化したシステムに対して全体の動作の観察および問題発生経路の解明など，様々な検証を行うことをコンセプトとしている．環境上の IoT システムの構成要素をアセットと呼び，これらの要素を自由に組み替えることで，任意の対象，抽象度およびレベルで検証可能とすることがを目指している．箱庭は，IoT システムを活用したシステムサービスを提供する技術者が，開発しているシステムを検証する場として利用できるだけではなく，IoT システムの構成要素の開発者が，成果物を箱庭上で試行する場として活用することもできる．箱庭では，様々なシミュレータを分散制御方式で動かして通信を行うことで，仮想環境を実現している．

2.1.1 単体ロボット向けシミュレータ

図 1 に，ET ロボコンを題材として開発されている箱庭のプロトタイプモデルである単体ロボットシミュレータの概要および構成を示す．このモデルでは Athrill が組込みマイコンを，Unity が実証実験環境を再現する．本モデルによって物理シミュレータとマイコンシミュレータの間での連携方法や，異なるシミュレータ間の時間管理の仕組みを確立している．単体ロボット向けシミュレータではこれら 2 つのシミュレータ間で通信を行うことで，コース上の 1 台のロボットを動作させることができる．

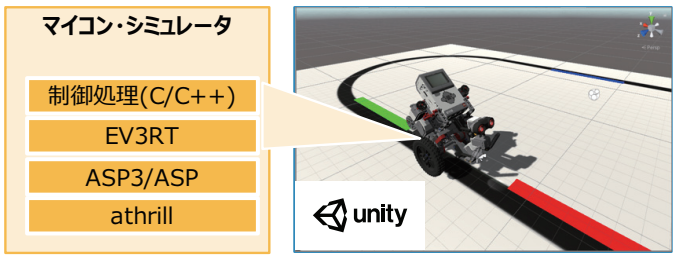

図 1 単体ロボット向けシミュレータ

Athrill は，TOPPERS プロジェクト箱庭 WG によって開発されている，組込みソフトウェア開発向けの命令レベルの CPU エミュレータである．これによって，仮想マイコン上におけるリアルタイム OS を含む組込みプログラムの実行およびデバッグが可能となる．現在は組込み用マイクロコントローラーである V850/RH850 および ARMv7-A の CPU 命令がサポートされている．Unity は IDE を内蔵した 3D ゲームエンジンであり，プラットフォームに依存しないことから汎用性が非常に高い．また，物理エンジンおよびレンダリングエンジン，物体同士の衝突検出機能等が充実しており忠実な 3D 環境を再現することができる．

単体ロボット向けシミュレータでは，Athrill はマイコンシミュレータとして利用している．Athrill は起動された後一定時間ごとに Unity とシミュレーション時間を同期させつつ，Unity からロボットのセンサのデータを受信し，これを元にロボットのモータの出力を計算したのち，Unity へ送信する．

Unity は，物理演算エンジンおよび空間可視化のために利用されている．Unity では短周期でセンサの値を更新する必要があり処理負荷が高くなるため，本プロトタイプモデルでは Unity のシミュレーション時間を基準として時間管理を行う．箱庭のテスト実行後，Unity は初期化を行う．その後は一定時間ごとにシミュレーション時間の送受信を行う．Athrill と接続後，Athrill と Unity のシミュレーション時間の差が許容範囲の場合は Unity のシミュレーション時間を進め，ロボットの更新をしたのち，物理演算処理を行う．ロボットが更新されるとき，まずは Athrill から送信されるアクチュエータのデータを読み込み，それぞれロボットのパーツに適用する．その後 Unity 内の各センサによって読み取ったデータを Athrill へ送信する．すなわち，Unity は Athrill と接続されるまでロボットデータのやり取りを行わない．

なお，ソフトウェアプラットフォームは Mindstorms EV3 用の開発プラットフォームである TOPPERS/EV3RT (Real-Time platform for EV3) [5] を採用している．ロボットについては ET ロボコンでも使用されている走行体ロ

ボット HackEV だけでなく，単純化された SimpleRobot などのロボットを動かすことができる．ロボットの制御アプリケーションは C または C++によって記述できる．

2.1.2 通信方式

箱庭では MMAP および UDP の二種類の方式から 1 つを指定してシミュレータ間の通信を行っている．

MMAP 通信では，ファイルの中身をメモリ上へマッピングし，マッピングされた領域に対して書き込みおよび読み込みを行うことで通信を行う．MMAP 通信を採用した場合，計算機のメモリを共有することで個々のシミュレータを同じ計算機に配置する必要があるが，通信ロストが発生しないためシミュレーションの精度が上がるという利点がある．

UDP 通信では，互いのシミュレータの IP アドレスおよびポートを指定することで，ネットワークを介したコネクションレス型の通信を行う．UDP 通信を採用した場合，コネクションレス型のネットワーク通信であることから信憑性が低く通信ロストが発生する可能性がある反面，シミュレータを別の計算機に配置することでシステム負荷を分散することができる．

本プロトタイプモデルでは，Unity から Athrill へのセンサのデータおよび Athrill から Unity へのアクチュエータのデータを，それぞれ 1024 バイトのサイズ長で送信している．なお，MMAP 通信を行う場合は，unity_mmap.bin および athrill_mmap.bin という名のバイナリファイルがメモリにマッピングされる．

2.2 関連研究

ロボット開発における通信方式や仮想シミュレーション環境に関連する研究を紹介する．

文献 [6] では，ロボット工学においてタイミング要件を満たす必要のあるリアルタイムシステムに焦点を当てている．その中でリアルタイム性能を提供することを目的とした Robot Operating System 2（ROS 2）の通信のリアルタイムアプリケーションに対する適合性を評価している．最長遅延時間と期限切れに注目し，負荷の影響を示すことで ROS2 の通信をリアルタイムアプリケーションに最適化するためには，アプリケーションの設定に応じたカーネルスレッドの構成，およびネットワークと CPU の使用量の制限が重要であることが示されている．

文献 [7] では，ros-bridge によって ROS と Unity を接続する一般的な方法論が紹介されている．ROS 側ではメッセージを rosbridge を介して Unity から出版および購読するためのリンクとして機能する unity-node を作成する．一方 Unity 側では同様の動作をする ros-script を作成する．この 2 つのアーキテクチャ間で JSON としてフォーマットされたメッセージを送信することで，ROS と Unity の通信を実現している．

文献 [8] では，Unity をベースとした仮想環境である 3D Simulator for Cooperative ADAS and Automated Vehicles Simulator（3DCoAutoSim）が提案されている．交通流シミュレータ Simulation of Urban MObiliy（SUMO）と Unity が TraCI プロトコルによって通信を行い，交通モデルに基づいた様々な運転環境をエミュレートすることができる．3DCoAutoSim には車車間，路車間，および車歩行者間の通信に基づくさまざまなアプリケーションをテストできる協調 ADAS 機能が含まれており，さまざまな車両，歩行者，インフラストラクチャから収集されたデータから生じる運用上の相互接続性を評価することが可能となる．

文献 [9] では，ツールに依存しない標準規格である Functional Mockup Interface（FMI）および FMI2 が提案されている．Functional Mockup Unit（FMU）と呼ばれるコンポーネントによって，シミュレータ等のツール間で相互接続を行う．相互接続のモデルは Model Exchange および Co-Simulation の 2 種類が存在する．Model Exchange は時間，状態およびステップイベントを持つ方程式で記述された，他のモデリング環境やシミュレーション環境で利用できるモデル方式であり，完全な時間同期が可能である．Co-Simulation は 2 つ以上のモデルとソルバを結合させるモデル方式であり，シミュレータ間の許容遅延時間を設定する．

3. 提案手法

本章では，自律移動ロボットの制御パラメータの自動探索手法を提案する．まず提案手法の方針を示したのち，その詳細について解説する．

3.1 方針

ロボットの制御パラメータ探索を行うためには，実証実験に掛かるコストや頻度を考慮すると，仮想環境上で行うことが望ましい．ここで，本研究では，仮想環境上でロボットの実証実験を自動的に繰り返し行うことで，ロボットの制御パラメータを探索することを目的としている．

まず，検証を行う環境は，アセット管理による汎用性の高さから，様々な実証実験を行うことができ，分散制御方式により拡張性も高い点から，箱庭を使用する．本手法ではまず，箱庭の単体ロボット向けシミュレータをベースとし，テスト制御モジュールを追加することで与えられたシナリオに対して箱庭上でテストを繰り返せるようにする．ここで，テスト内容は，ロボットをコースに沿って周回させ，この周回時間が最も短くなることを目的とする．

ここで本研究では，テスト制御モジュールによって制御パラメータを探索する手法としてベイズ最適化を採用する．ベイズ最適化では，現在の回帰関数の確率モデルに対し推定値および分散を利用し，目的変数が最大となる可能性が高い説明関数を導き，サンプル数を増やしていくことで目

Vol.2021-ARC-244 No.23
Vol.2021-SLDM-194 No.23
Vol.2021-EMB-56 No.23
2021/3/25

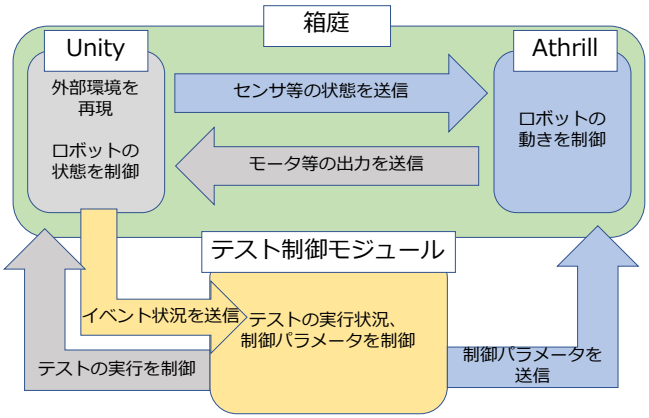

図 2 提案手法におけるテスト自動化のための全体像

的変数の最大値を探索する．パラメータ最適化の方法としては，他の手法を採用することもできるが，本手法では探索すべきパラメータが複数存在し，なおかつパラメータ 1 組ごとに行うテストに時間がかかるため，少ないサンプル数で探索を行うことのできるベイズ最適化を採用する．また，本手法で動かすロボットは 1 台であり，UDP 通信の利点であるシミュレーションを行う計算機を分散することができる点が活かしにくいため，シミュレータ間の通信方式は精度の高い MMAP とする．

3.2 全体像

箱庭による自動テストを実現するための全体像およびシミュレータ間のデータの流れを，**図 2** に示す．本手法は単体ロボット向けシミュレータをベースとしており，箱庭では Unity と Athrill が通信を行い，Unity が Athrill へセンサの情報を，Athrill が Unity へモータの情報を送信することで Unity 上でロボットを操作する．

提案手法では，ここでさらにテスト制御モジュールを追加する．本モジュールでは，Unity がテスト中のイベント状況を管理してテスト制御モジュールに送信し，テスト制御モジュールが Unity にテスト実行に関するフラグを送信して Unity のテストを制御することで，テストの自動化を実現する．さらには，テスト制御モジュールがテストごとに Athrill へ制御パラメータを送信し，Athrill がそのパラメータを読み込んでロボット制御を実行することでロボットの制御パラメータを探索する．

3.3 テスト自動化のためのデータ通信の仕様

表 1 および**表 2** に，本手法における箱庭のテスト自動化のために `unity_mmap.bin` と `athrill_mmap.bin` に追加した通信データ仕様を，それぞれ示す．単体ロボット向けシミュレータでは Unity から Athrill へ送信するデータは `unity_mmap.bin`，Athrill から Unity へのデータ送信には `athrill_mmap.bin` を使用して通信を行う．

表 1 テスト自動化のための `unity_mmap.bin` の追加の通信データ

名称	オフセット	サイズ	値の単位表現
イベント状態	528	4	整数値
ゴールタイム [usec]	532	4	上位 32bit
	536	4	下位 32bit
制御パラメータ	540	484	—

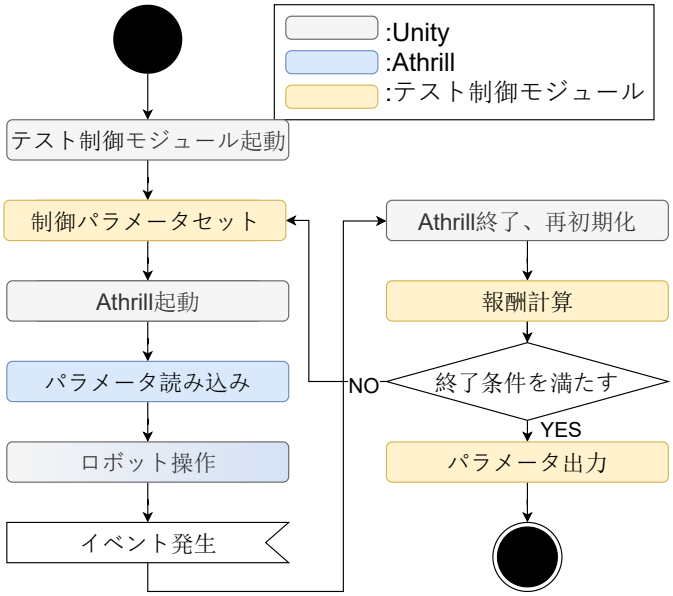

図 3 全体のフロー図

本手法のテスト制御に利用するデータについては Unity から送信するデータを `unity_mmap.bin` へ，Unity が受信するデータを `athrill_mmap.bin` へ格納する．すなわち，Unity でのイベント状態およびロボットのゴールタイムといったイベント状況のデータは `unity_mmap.bin` へ，Unity のテストをスタートさせるフラグおよび終了させるフラグといったテストの実行を制御するデータは `athrill_mmap.bin` へ格納する．これは，単体ロボット向けシミュレータではデータを書き出すファイルと読み込むファイルを分離させているためである．なお，テスト制御モジュールが Athrill へ送信する制御パラメータは，`unity_mmap.bin` へ格納する．これらのイベント状況や Unity の実行を制御するフラグを通信することで，テストを制御して自動化することを実現する．

3.4 制御パラメータ探索のための動作フロー

本節では，制御パラメータ探索の自動探索手法について，まず全体の動作フローを示す．その後，各モジュールの動作フローの詳細を説明する．

図 3 に制御パラメータ探索が完了するまでの全体のフローを示す．この図では Unity，Athrill，およびテスト制

表 2 テスト自動化のための `athrill_mmap.bin` の追加の通信データ

名称	オフセット	サイズ	値の単位表現
スタートフラグ	512	4	整数値
終了フラグ	516	4	整数値

Vol.2021-ARC-244 No.23
Vol.2021-SLDM-194 No.23
Vol.2021-EMB-56 No.23
2021/3/25

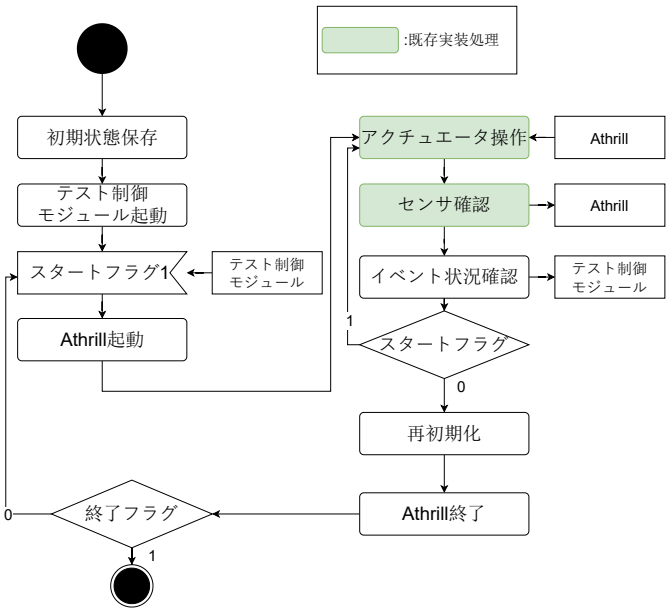

図 4 Unity のフロー図

御モジュールが制御パラメータ探索を行う過程でそれぞれどのような役割を担っているかを簡単に示している．また，各処理は実行するモジュールごとに色分けをしている．

本手法では Unity が MMAP 通信用のバイナリファイルをメモリにマッピングする．このため，Unity を他のモジュールよりも先に起動する必要があり，なおかつ他のモジュールが実行されている状態で Unity を再起動することができない．そのため，まずはじめに Unity を実行した後 Unity からテスト制御モジュールを起動する．その後，テスト制御モジュールが制御パラメータをバイナリファイルにセットしたことを確認し，Unity から Athrill を起動する．Athrill は起動後テスト制御モジュールによってセットされた制御パラメータを読み込み，Unity でイベントが発生するまで Unity と通信を行いながらロボットを操作する．イベント発生後は Unity は Athrill を終了させ，ロボットの位置やテスト環境の状態を再初期化する．その後発生したイベントに対してテスト制御モジュールが報酬計算を行い，終了条件を満たしていない場合はその報酬を元に次の制御パラメータをセットしたのち，もう一度テストを実行する．なお，Athrill の再初期化はテストごとに起動することで実現している．

3.4.1 Unity

図 4 に本手法における Unity の動作フローを示す．

Unity は起動された後，通常の初期化に加え再初期化のためにロボットと運転環境，すなわち `Robot` という名前のオブジェクト，`env` という名前のオブジェクト，および，それら全ての子オブジェクトの位置および角度を保存する．そして，実行ボタンが押されるとテスト制御モジュールが実行される．ここで，テスト制御モジュールが書き込むスタートフラグが 1 になると Athrill を起動するが，単体ロボット向けシミュレータでは Athrill と接続するまで受信用ファイルである `athrill_mmap.bin` のデータが自動で読み込まれることはない．よって，本手法では従来のファイルの読み込みに加え，Athrill の通信に関わらず一定時間ごとにスタートフラグを読み込ませる．このようにしてテストが開始される．

テストが開始された後は，単体ロボット向けシミュレータと同様に一定時間ごとに Athrill が書き込むアクチュエータのデータを読み込んでセットし，センサの情報を書き込むことでロボットを操作する．本手法ではこれに加え，イベント情報を表す列挙型の整数値を書き込む．イベント情報の対応を**表 3** に示す．

イベント情報はテスト開始時は未発生とし，Unity で管理されているシミュレーション時間が事前に設定した値を超過した場合は制限時間超過とみなす．また，コース外に壁をセットし，衝突したオブジェクトの親オブジェクトの名前が `Robot` である場合ロボットが壁に衝突したとみなすスクリプトを壁オブジェクトにセットすることで衝突検出を行っている．ここで，ロボットの超音波センサへの反応を防ぐため衝突対象が実体を持たない場合は除外している．そして，空のチェックポイントをコース上に設置し，壁と同様にロボットの通過を検知させ，定められた順番でロボットが全てのチェックポイントを通過した時，ロボットは周回に成功したと判定する．この時，周回時間を記録する．

このようにセンサ情報およびイベント情報を送信し，アクチュエータ情報を受信しながらロボットを動かす中で，イベント発生を検出したテスト制御モジュールが書き込むスタートフラグが 0 になったとき，テストをリセットする．すなわち，Unity 起動時に保存した初期状態へ戻し，イベント状況等の情報を初期化したのち，Athrill を終了させる．この後，テスト制御モジュールが書き込む終了フラグが 1 の場合は Unity を終了し，0 の場合はスタートフラグが 1 になるまで待機する．

3.4.2 Athrill

Athrill は Unity により起動された後，テスト制御モジュールによって書き込まれた制御パラメータを読み込む．`unity_mmap.bin` のマッピングの開始アドレスは `0x090F0000` であるため，（`0x090F0000`+オフセット）のメモリを直接読み込むことで制御パラメータを読み込む．その後は一定時間ごとに Unity が書き込んだセンサ情報を元

表 3 イベント情報の対応

イベント	列挙子	整数値
イベント未発生	EVENT_NONE	0
ロボットがコース周回成功	GOAL	1
制限時間超過	TIME_OVER	2
ロボットが壁に衝突	HIT_WALL	3

Vol.2021-ARC-244 No.23
Vol.2021-SLDM-194 No.23
Vol.2021-EMB-56 No.23
2021/3/25

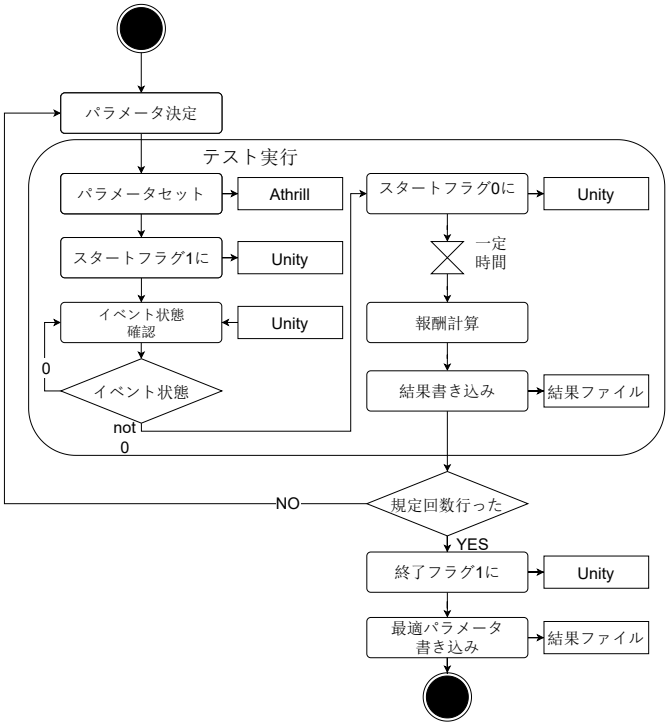

図 5 テスト制御モジュールののフロー図

にロボットのアクチュエータの出力を計算し，書き込むことでロボットを操作する．このループは Unity から Athrill を終了されるまで実行される．

3.4.3 テスト制御モジュール

図 5 に本手法におけるテスト制御モジュールの動作フローを示す．ここで，3 つの制御パラメータを探索するとする．

まず，制御パラメータを引数として報酬を出力とする関数を定義する．ここで，Unity と Athrill は MMAP ファイルに対してベースオフセットを 32 としているため，テスト制御パラメータから MMAP ファイルにアクセスする際は，先述のパラメータのオフセットに対して 32 を加えたオフセットにアクセスする．

関数が呼び出されるとまず，各パラメータの値を書き込み，スタートフラグを 1 にして Unity のテストを開始させる．一定時間ごとに `unity_mmap.bin` ファイルを開き，イベント状況を読み込む．イベントが発生した場合は結果を保存してスタートフラグを 0 にして Unity をリセットさせた後，Unity が読み込むため一定時間待機する．その後はイベントの結果から報酬を計算して出力する．

パラメータを最適化するときはまず，パラメータ探索に関する情報を定義しておく．ここでは，パラメータ探索手法は `GPyOpt` パッケージによるベイズ最適化を採用して説明していく．まずは各パラメータの定義を設定する．すなわち，変数名，連続であるかどうか，および定義域等を設定する．その後最適化する関数，パラメータ定義，およびベイズ最適化のパラメータ選択における獲得関数を設定することでテスト実行関数の出力を最大化するベイズ最適化オブジェクトを作成する．その後ベイズ最適化オブジェクトをテスト回数を設定して実行する．このようにしてベイズ最適化によって選択された制御パラメータでのテストが一定回数行われ，制御パラメータの探索が行われる．最適化が完了した後は，終了フラグを 1 にして Unity を終了させ，結果を出力して終了する．

4. 適用事例

提案する箱庭のテスト制御モジュールは，Python で実装した．本章では，提案手法を実際のロボット開発事例に適用した事例を示し，その有効性を議論する．

4.1 適用対象

まずは実行するテストの内容について説明する．このテストでは**図 6** に示すオーバルコースに対し，`SimpleRobot` がコースに沿って周回する時間が最も早くなる PID パラメータをベイズ最適化を用いて探索する．PID 制御はフィードバック制御方式の一つであり，現在の測定値と目標値に対して，差，差の積分および微分にそれぞれ定数をかけた値の和を操作量とする．PID パラメータおよびモータの基本出力の通信仕様を**表 4** に示す．オーバルコースの周回において，4 つの透明で正方形のチェックポイントを順番に通過する必要があるものとする．

成功判定に関してはロボットが壁に衝突，もしくは 60 秒の制限時間を超過した場合は失敗，失敗せずにコースの周回を完了した場合は成功とみなす．ここで，テストを行う計算機の CPU 性能に対しテストによるシステムへの負荷が高いことで同じ制御パラメータでのテストに対し結果に差が出ることがあるため，1 つのパラメータに対して複数回テストを実行し，先に 3 回成功した場合は成功し，先に 3 回失敗した場合は失敗とする．そして，成功した場合は周回時間の平均を，失敗した場合は制限時間と同じ 60 秒を結果とし，この時間を最小化させる PID パラメータを探索する．

PID パラメータはそれぞれ [0,5] の範囲で探索し，モータの基本出力は 15 とする．また，ベイズ最適化では，既存のサンプルにおける目的変数の最大値および結果の差の期待値が最も高くなる説明関数を選択する戦略である Expected Improvement（EI）を採用する．初めにランダムに決定した制御パラメータに対し 5 回テストを行い結果を評価し，それを初期データとしてベイズ最適化によるパラメータ探索を 100 回繰り返す．

本テストのロボットに使用するアプリケーションは単体ロボット向けシミュレータのサンプルアプリケーションである `line_trace` アプリケーション [10] をベースにしている．**図 7** に，本テストで使用する．ロボット制御プログラムのコードの一部を示す．なお，white は 100，black は 50

Vol.2021-ARC-244 No.23
Vol.2021-SLDM-194 No.23
Vol.2021-EMB-56 No.23
2021/3/25

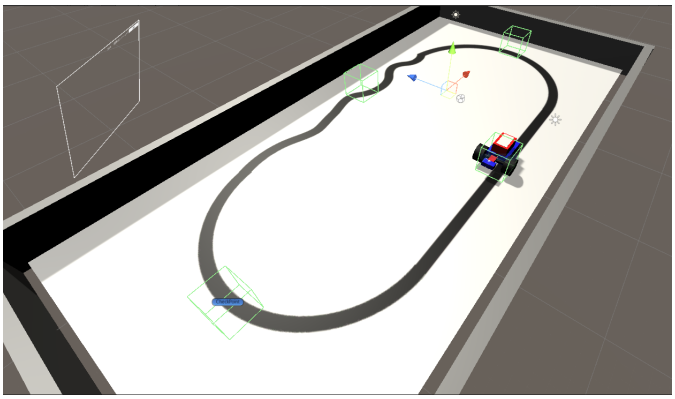

図 6 Unity のテストコース

```
1  static float lasterror = 0, integral = 0;
2  static float mid = (white - black)/2 + black;
3
4  while(1){
5    float error = mid - sensor_color(PORT_1);
6    integral = error + integral * 0.01;
7    float steer = (*Kp) * error + (*Ki) * integral
         + (*Kd) * (error - lasterror);
8    motor_steer(left_motor, right_motor,
         MOTOR_POWER, steer);
9    lasterror = error;
10   tslp_tsk(100000); /* 100msec */
11 }
```

図 7 ライントレースアプリケーションの制御コードの一部

と定義されている．また，Kp，Ki，Kd および MOTOR_POWER は Athrill 起動時に読み込ませる制御パラメータである．このアプリケーションでは 2 行目にて white と black の中間の明るさを目標値とし，ロボットのカラーセンサが検知した反射光の強さを目標値に近づけるようにステアリングを行う．そこで，5-7 行目にて計算される目標値とカラーセンサの値との差をもとに PID パラメータによってステアリング量を計算し，8 行目でモータの基本出力とステアリング量から左右のモータの出力を書き込む．この処理を繰り返すことでロボットを制御している．

なお，本テストでは Unity のシミュレーション精度を 0.001，ロボット制御プログラムの実行周期を 100ms，Unity と Athrill のシミュレーション時間の差の許容範囲を 20ms とする．

表 4 本適用事例における unity_mmap.bin の通信仕様

名称	オフセット	サイズ	値の単位表現
P パラメータ	540	4	浮動小数点数
I パラメータ	544	4	浮動小数点数
D パラメータ	548	4	浮動小数点数
モータ出力	552	4	浮動小数点数

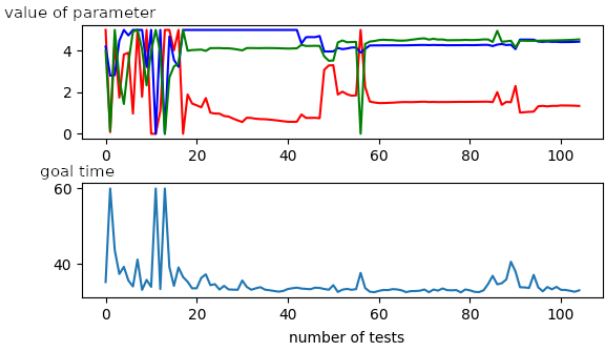

図 8 制御パラメータの探索値および周回時間の遷移

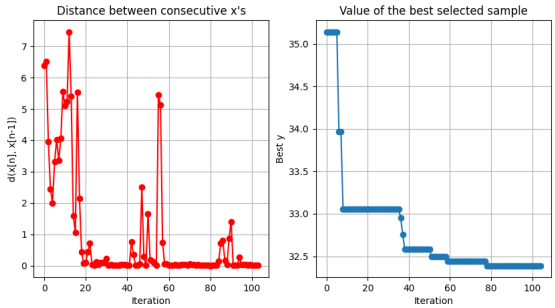

図 9 全パラメータの変化量の遷移および最適解の遷移

4.2 適用結果および考察

箱庭においてテストを行った結果，試行中で最も周回時間を短くできるロボットのそれぞれの制御パラメータは，[Kp, Ki, Kd] = [1.539, 4.267, 4.511] である時であった．その時の周回時間は 32.39 秒となった．

まず，**図 8** に各テストごとの制御パラメータの探索値の遷移および周回時間の遷移を示す．制御パラメータの遷移図において，赤は Kp，青は Ki，緑は Kd をそれぞれ表している．ここから，20 回目までは様々な値でパラメータ探索を行っているが，その後は探索値がある程度安定していることがわかる．また，周回時間の遷移を見ると，最小時間の探索が進むにつれて周回の成功率も上がり，探索パラメータの値が安定している 20 回目以降は周回失敗が起こっていないことが確認できる．次に，**図 9** に，テストごとの全パラメータの変化量の遷移，および最適解の遷移を示している．全パラメータの変化量の遷移を調べると，各パラメータの定義域が [0,5] であることから全パラメータの変化量の値域がおよそ [0, 8.5] であるのに対して，最終的に 0 に近づいていることから，制御パラメータの探索値が収束していることがわかる．これらの点から，パラメータ探索が最適値付近にたどり着き，探索が成功していると言える．また，テストごとの最適解の遷移を調べると約 40 回目以降から最適解にあまり変化がない．ここから，少ない試行回数で探索が成功していることがわかり，ベイズ最適化の利点が発揮されているといえる．

これらの結果から，本手法による箱庭を使った自動テス

トおよび制御パラメータの探索が実現されていることが示された．今回の適用事例ではオーバルコースに対してロボットの周回時間のみを評価してステアリングの PID パラメータの値を探索したが，テスト内容を変更することで様々な事象を検証することが可能である．例えば，周回するコースを変更したり，ロボット制御アプリケーションのライントレースに対し，ロボットのカラーセンサが読み取った反射光に対する目標値，すなわちトレースコースの閾値や，モータ出力を変更，もしくは探索を行うことも可能である．また，評価方法およびパラメータ探索手法を変え，目的関数の予測を重視して検証を行うことで，与えられたパラメータの定義域におけるテストの成功率や，成功の可否の閾値を求めることも可能である．

5. おわりに

本研究では，箱庭を活用した自律移動ロボットの開発において，低コストで検証が行える仮想環境の利点を活かし，その制御パラメータを自動で探索する手法を提案した．まず，箱庭における通信方式を拡張し，各シミュレータの動作を制御するモジュールを実装することで，箱庭上でテストを繰り返し実行できるようにした．そして，実行するテスト要件に沿った目的関数を設定し，ベイズ最適化によって報酬を最大化することで制御パラメータの探索ができるようにした．これによって，通常は実証実験が必要となる最適な制御パラメータの探索を低いコストで探索できるようになる．さらに，ロボット開発における要件の変更に応じた定期的なフィードバックを得ることが可能となる．

今後の方針として，まず，本研究で実施した適用事例より複雑なテスト要件に応えた評価実験が挙げられる．様々なイベントを検出対象とし，周回時間以外のテストの評価基準に対応することが必要である．また，複数の検証を同時に行える仮想環境の利点を活かすには，テストの実行および評価の機能とパラメータ探索のための機能でモジュールを分割し，それらを分散処理することも考えられる．箱庭上のテストを複数の計算機で並列に実行できれば，パラメータの探索速度を向上させることが期待できる．さらには，本研究の提案手法を CI/CD パイプラインに組み込み，箱庭によってロボット開発の継続的インテグレーションを推進させることも検討している．

謝辞 本研究の一部は，JST さきがけ JPMJPR18M8 の支援を受けたものである．本研究の実施にあたって議論や支援をいただいた TOPPERS プロジェクトならびに箱庭 WG の各位に深く感謝する．

参考文献

[1] 高瀬英希，細合晋太郎，高田光隆，庭野正義，辻悠斗，森崇：IoT 時代の仮想シミュレーション環境「箱庭」の実現に向けた検討および初期実装，情報処理学会研究報告，Vol. 2020-EMB-55, No. 3, pp. 1–7 (2020).

[2] TOPPERS プロジェクト箱庭 WG：箱庭（オンライン），`https://toppers.github.io/hakoniwa/`（参照 2021-01-24）.

[3] ET ロボコン実行委員会：ET ロボコン — 組込みソフトウェア技術教育をテーマとした「ET ロボコン」，ET ロボコン実行委員会（オンライン），`https://www.etrobo.jp/`（参照 2021-01-24）.

[4] 吉留忠史：ET ロボコンを通じた組込みソフトウェア技術教育，日本ロボット学会誌，Vol. 38, No. 9, pp. 805–808 (2020).

[5] 李奕驍，松原豊，高田広章：EV3RT: LEGO Mindstorms EV3 用リアルタイム・ソフトウェア・プラットフォーム，日本ソフトウェア科学会コンピュータ ソフトウェア，Vol. 34, No. 4, pp. 91–115 (2017).

[6] Gutiérrez, C. S. V., Juan, L. U. S., Ugarte, I. Z. and Vilches, V. M.: Towards a distributed and real-time framework for robots: Evaluation of ROS 2.0 communications for real-time robotic applications, *arXiv preprint arXiv:1809.02595* (2018).

[7] Hussein, A., García, F. and Olaverri-Monreal, C.: ROS and Unity Based Framework for Intelligent Vehicles Control and Simulation, *2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, pp. 1–6 (2018).

[8] Hussein, A., Díaz-Álvarez, A., Armingol, J. M. and Olaverri-Monreal, C.: 3DCoAutoSim: Simulator for Cooperative ADAS and Automated Vehicles, *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 3014–3019 (2018).

[9] Blockwitz, T., Otter, M., Akesson, J., Arnold, M., Clauss, C., Elmqvist, H., Friedrich, M., Junghanns, A., Mauss, J., Neumerkel, D., Olsson, H. and Viel, A.: Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models, *Proceedings of the 9th International Medelica Conference*, pp. 173–184 (2012).

[10] TOPPERS プロジェクト箱庭 WG： Sample collection of scenario for Hakoniwa（オンライン），`https://github.com/toppers/hakoniwa-scenario-samples/tree/master/single-robot/line_trace`（参照 2021-01-24）.