F-032

階層的強化学習系列による自然知能模倣アーキテクチャの提案と実装
Proposal and Implementation of a Natural Intelligence Imitation Architecture Using Hierarchical Reinforcement Learning Sequences

佐藤 玲於奈† 田胡 和哉†
Reona Sato Kazuya Tago

1. はじめに

近年，特定の意思決定タスクに特化した人工知能を実現する手法として，深層強化学習が急速に発展している．その応用分野は金融工学や自動運転，ロボティクスなど，様々な領域に広がりつつあり，ゲーム AI 以外の領域においても人間と同等以上のパフォーマンスを出すことが期待されている．しかし，現状の発展的な強化学習の研究は，人間の手によって設定される特定のタスクに特化した意思決定の学習方法についての議論である場合が多い．本稿では，実世界の生物が本来持っていると考えられる，より広範囲な領域下での汎用的な意思決定の構造を工学的に構築することに着目し，人工知能技術のさらなる発展のために必要な指針についての検討を行う．

現実で汎用的に活動する動物の多くは，自己生存という大規模かつ抽象的なタスクを持ち，それを継続的に達成し続けるための学習機能を有している．彼らは過去の知見を利用し，様々な状況で即座にもっともらしい行動を取ることができる．ここでは，この知能を自然知能と呼ぶこととする．例えば，サバンナに棲む野生のサーバルは，狩りや休眠，退避などの行動を継続的に選択して行動し続けることで，生き残るというタスクを日々達成し続けている．さらに，退避や狩りなどのあらゆる行動方針は定常的ではなく，状況に応じて適宜調整することによって，生存確率を上げることを試み続けていると考えられる．ここから導出される自然知能としての要件は次の 3 つである．1 つ目は，報酬が疎な問題に対して漸近的に対処する術を持つことである．2 つ目は，過去に得た知見を理解可能な形で分割し，明示的に再利用することによって効率的に問題に対処できることである．3 つ目は，恒久的に動作し続ける環境と学習構造のセットを持つことである．以降では，これらの要件を満たし，ロボットなどの行動主体が実世界でより汎用的に活動するためのシステムを実装することを目指す．

2. 背景

強化学習は，特定の行動を取ることによってある状況を達成した場合に報酬が与えられるような環境において，獲得する報酬の累積値を最大化するように行動戦略を最適化する機械学習手法である．これは，生物の脳の学習メカニズムとの類似性が指摘されており，生物が実際に判断基準を学習することと大きく関係していると考えられている．しかし，このアルゴリズムは，あらかじめ人間の手で綿密に設計された報酬関数を与える必要があるのに加えて，報酬を得るまでに必要な一連の行動の長さが極端に長くなるようなタスクに対しては学習が困難である．

報酬が疎な問題に対する強化学習のアプローチとして，階層的強化学習（HRL）が存在する．これは，サブゴール状態の概念を導入し問題をより小さなサブ問題に分割することで，本来は長い一連の行動を必要とするタスクを短い意思決定タスクとして解決する手法である．近年の主要な HRL のひとつである Hierarchical Actor-Critic（HAC）[1] は，連続的な行動空間，状態空間を持つ複雑なロコモーションタスクにおいて複数の方策階層を並行的に学習させ，グリッドワールドとロボット工学の両方のドメインで学習を大幅に加速できることを示している．この手法では，エージェントが移動するための行動である各関節のトルク出力または下位レベルの方策の出力を要素として複数持つ行動系列が実行された後の状態を，抽象化した行動として扱う．これを入れ子構造的に行うことによって，エージェントは学習すべき時間スケールに適合したサブゴールを階層的に自ら発見しつつ，報酬が疎なタスクでの学習を加速させることに成功している．

HAC 以外にも，複雑なタスクを扱う多くの深層強化学習手法が現在研究されており，人間が設定する特定のタスクに対して十分な効果を発揮するようになりつつある．しかし，あらゆる強化学習アルゴリズムは，タスクの開始条件，終了条件，報酬関数など実行に必要な要素を事前に人の手で設定しておく必要がある．そのため，エージェントを実世界で汎用的に活動させるための構造としては，よりメタ的な動作構造が根本的に不足している．

このような背景から，本研究では，任意の強化学習アルゴリズムを実装した複数の学習器をサブタスクとして並列動作させ，実環境における広範な行動を継続的に学習することが可能なロボット制御アーキテクチャを提案する．

3. 提案するアーキテクチャ

3.1 全体構造

ここでは，自己生存など生物が根本的な行動目標として設定する可能性のある，抽象的かつ報酬が極めて疎なタスク設定のことをメインタスクと呼ぶこととする．また，逃避や狩猟など，メインタスクを達成するための部分的な行動方針をサブタスクと呼ぶこととする．

†東京工科大学大学院バイオ情報メディア研究科
Graduate School of Bionics, Computer and Media Sciences, Tokyo University of Technology

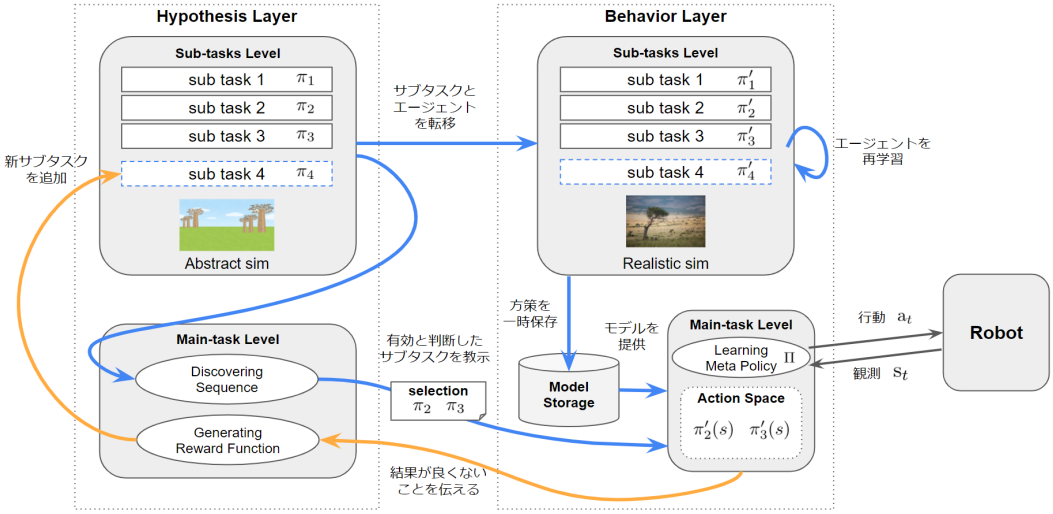

図 1　保有するサブタスクを利用して暫定的な対応をとりつつ，現状の選択肢では満足するパフォーマンスを得られない場合に新たな報酬関数を生成することで対応を学習していく構造

提案するアーキテクチャは，ソフトウェア的に実装される 2 つの層で構成される．(i) 特定のメインタスクを達成するために必要なサブタスクの実行順序をイメージする領域としての役割を持つ仮説層と，(ii) 仮説層のサブタスク方策を実際の環境で使用できるように微調整し，実際の環境と相互作用させる領域としての役割を持つ実行層の 2 つである．このアーキテクチャは，エージェントがメインタスクを認識したとき，自身が習得している複数のサブタスク方策を用いてメインタスクへ直ちに対応しつつ，その対応方法を継続的に学習することを目的とする．実行層と仮説層は，各層で一意に対応するサブタスクの強化学習器と，抽象度の異なるシミュレーション環境をそれぞれ持つ．以下では，アーキテクチャの挙動を表す図 1 を用いて，各層の構成要素とその詳細な役割を説明する．

3.2 仮説層

仮説層は，図 1 で Hypothesis Layer と表記されている領域に該当する．これは，異なる報酬関数を持つサブタスクの方策を事前に訓練すると同時に，現在のメインタスクに対するサブタスクの有効性を検証する層である．また，新しい報酬関数を持つサブタスクを新たに生成する機能を有する．この層では，実際の精密な制御行程などを抽象化した軽量なシミュレーション環境や環境のダイナミクスモデルを利用することによって，素早く方策を構築することを目標とする．実際の生物が脳内で瞬時に行動を計画するのと同じような挙動を期待する．

この層は，サブタスクを扱う領域 Sub-tasks Level と，メインタスクを扱う領域 Main-task Level で分割される．Sub-tasks Level は，サブタスクの数だけ存在する環境を利用し，各サブタスクを並行して事前学習する役割を持つ．Main-task Level は，サブタスクの学習に使用したものと同じ条件のシミュレーション環境を用いて，学習したサブタスク方策を要素とする実行系列の最適化を試みる．また，必要に応じて既存のサブタスクと分布の異なる報酬関数を生成し，新たなサブタスクを作成する．

3.3 実行層

実行層は，図 1 で Behavior Layer と表記されている領域に該当する．これは，仮説層から転移させたサブタスクを用いて，実際の環境と相互作用しながらメインタスクのメタ方策を学習させる層である．ここでは，サブタスクを fine-tuning させつつ，調整途中のサブタスクを利用して漸近的にメインタスクに対処する術をエージェントが持てるようにすることを目指す．例えば，対戦ゲームをプレイする人間が上達を目指す場合，脳内でイメージしたテクニックを，実際の試合ではなく練習モードで身体的に習得しつつ，実際の試合を経て戦術に組み込んでいくという流れを繰り返すのがセオリーである．また，強化学習の文脈では，Sim2Real のギャップを軽減する狙いがある．

この層も仮説層と同様に，Sub-tasks Level と Main-task Level の 2 つの領域を持つ．ここでの Sub-tasks Level は，仮説層から転移させたサブタスクを実環境に適合させるために再学習させ，あるタイミングで学習済み方策を同層の Model Storage に保存する役割を持つ．Main-task Level は，Model Storage に保存されている学習済みのサブタスクの中からいくつか選択してロードし，サブタスクの方策関数の出力をそのまま行動として扱うメタ方策を利用した強化学習を実世界で実行する役割を持つ．

3.4 定義

本稿の説明に登場する記号を定義する．ここでは，任意のサブタスクの識別番号をiとしたとき，仮説層側のサブタスク方策を π_i，実行層側のサブタスク方策を π'_i と表記する．仮説層側のサブタスクの行動空間 A_{sub} は，実行層側のサブタスクの行動空間 A'_{sub} と同じである： $A_{sub} = A'_{sub}$ ．また，それらの状態空間 S_{sub}, S'_{sub} についても同様である: $S_{sub} = S'_{sub}$．実行層の Sub-tasks Level で扱う状態空間 S'_{sub} と Main-task Level で扱う状態空間 S'_{main} は同一のものを設定する: $S'_{sub} = S'_{main}$．各層におけるサブタスクの環境は，状況の複雑度，試行の速度のみ異なるものとする．実行層の Main-task Level は，ある時刻 t において状態 $s'_t \in S'_{main}$ を観測したときのサブタスクi の方策関数 π'_i の出力

$\pi_i'(s_t')$ を行動として利用するようなメタ方策 Π' を持つ．Π' は，行動としてサブタスクの識別番号を出力する決定論的な方策である．

3.5 全体の挙動

図 1 のうち青い矢印で示される流れは，行動主体であるロボットが何らかのメインタスクを認識したときに，持ちうるサブタスクを活用して行動を起こすまでの処理を示したものである．ここで前提として，システムは初期状態の時点で仮説層にいくつかサブタスクを持っているものとする．認識されたメインタスクの情報が各層の Main-task Level に適用されると，仮説層は次の 2 つのプロセスを同時に開始する．1 つ目は，事前学習済みのサブタスクを実行層側へ転移させるプロセスである．このとき転送される情報には，各エージェントがサブタスクの学習を再開するために必要な報酬関数，価値関数，方策関数の情報に加えて，仮説層側の環境で達成することができた累積報酬値などのスコア情報も含まれる．サブタスクの情報を受け取った実行層は，Sub-tasks Level にサブタスクの個数分だけ学習環境を展開し，仮説層側でのスコアを目標基準として並行的にサブタスクの fine-tuning を開始すると同時に，定期的に学習済みモデルを Model Storage へ保存していく．2 つ目は，仮説層の Main-task Level でメインタスクに対する現在のサブタスクの有効性を検証するプロセスである．ここでは，現在のサブタスクのどの組み合わせがメインタスクに対して有効なのかをシミュレーションで検証し，最も評価の高い組み合わせ C^* を選出する．以上の 2 つのプロセスが完了すると，実行層の Model Storage には全てのサブタスクの方策関数を再現するためのパラメータが登録され，仮説層の Main-task Level は有効なサブタスクの組み合わせを提示できる状態になる．それらの準備が整った段階で，実行層の Main-task Level は利用すべきサブタスクの情報を仮説層に問い合わせ，その内容に従い Model Storage から方策をロードする．そして，用意されたサブタスク方策の組み合わせに応じてメインタスクの行動空間 $A'_{main} = C^*$ を設定し，メタ方策 Π' の強化学習を開始する．これによって，エージェントは本能的に持っている一定レベルの知見を利用して実環境下で動作し始める．

実行層の Sub-tasks Level は，以上の動作が開始した後もサブタスクの微調整を行い，定期的に Model Storage の方策を更新し続ける．また，それに伴い実行層の Main-task Level でのメタ方策が行動として利用する各方策のパラメータも同時に更新する．これによって，実際のエージェントは，イメージとして持っていた戦術である各サブタスクが実環境に合わせて調整されるのを待つことなく，即座に暫定的な行動を取ることが可能になる．

図 1 の黄色い矢印で示される流れは，メタ方策 Π' が収束し始め現状持つサブタスクのセットでは満足するパフォーマンスを達成できないと実行層の Main-task Level が判断したときに，動作する処理を示したものである．実行層の Main-task Level は，メタ方策 Π' の学習の収束を検出した時点での平均的なパフォーマンス指標がある閾値を下回っていた場合に，仮説層の Main-task Level へ新たなサブタスクの生成を依頼する．依頼を受けた仮説層の Main-task Level は，これまでのログデータをもとに，分布の異なる新たな報酬関数の生成を試みる．それを新たなサブタスクとして仮説層の Sub-tasks Level に加え，全体の動作を再開させる．これらの流れを繰り返すことよって，エージェントは能動的に新たな行動方針の選択肢を取り入れながら，メインタスクに漸近的に対処していくことが可能になる．

4. 検討

4.1 実装上の課題

移動ロボットなどを用いてアーキテクチャを実装するとき，次のような点に注意する必要がある．1 つ目は，サブタスクとして扱う問題の規模である．本稿では，サブタスクの例として，生物の自己生存というメインタスクに対する部分的な行動方針として狩猟や休眠などといったものを挙げた．しかし，これらのサブタスクは，連続値の速度指令を扱うような従来の深層強化学習法の観点からは，未だに報酬が疎な問題として扱われる．例えば，ロボットの各関節のトルクを出力するような細かい行動を扱うタスクを本アーキテクチャのサブタスクとして採用した場合，サブタスクの学習の難易度が上がり，事前学習の低速化やメタ方策 Π' の学習の高難度化につながる可能性がある．そのため，ここでのサブタスクの設計は，1 ステップの行動の時間スケールを増大させ，単体でもある程度高度な挙動を行えるような粒度にするのが望ましい．2 つ目は，実行層での Sim2Real ギャップの問題である．本アーキテクチャでは，抽象的な環境を持つ仮説層からより実環境に近いシミュレーション環境を利用する実行層へサブタスクを転移させる際に，fine-tuning を行うことによってギャップの解消を試みる．ただし，実行層のシミュレーション環境で調整したサブタスク方策は，依然として実環境との間にギャップが存在する可能性が高い．そのため，実行層の Sub-tasks Level のシミュレーション環境に Domain Randomization [2] を適用するか，実際の制御のノイズを無視できるレベルの抽象的な行動を扱うサブタスクを設計する必要がある．3 つ目は，実行層の Main-task Level のメタ方策の学習途中に，利用するサブタスク方策の遷移関数が変化し続ける点である．強化学習では，遷移関数の分布が安定している必要があるため，サブタスクの調整過程で当初の分布と大きく差が生じる場合，メタ方策の収束効率が低下する可能性がある．そのため，サブタスクの熟練度に応じて学習率の逐次的に変更するなどの対処によって，調整不足のサブタスク方策がメタ方策の学習に与える影響をコントロールする必要がある．

5. 実装

実際のロボットシステムと小型の移動ロボットを用いて実行層を実装することで，アーキテクチャの具体的な実装要件を確認する．なお，ここでは仮説層によってメインタスクに対する有効なサブタスクの組み合わせと事前学習済みの方策関数が既に与えられていると仮定して，再現的に実装を行う．

5.1 実装上の課題への対応

はじめに，前述した実装上の課題について検討する．まず，1 つ目で説明したタスクの設定基準の問題と 2 つ目で説明した実行層内における環境間のギャップ問題の両方に対しては，自律移動システムの存在を前提とするタスクを定義することによって解決する．ここで利用する自律移動

システムは，ロボット開発用のオープンソースソフトウェアプラットフォームである Robot Operating System (ROS) が提供する Navigation Stack である．これは，自己位置推定と地図生成を同時に行う SLAM 技術を利用して経路計画を行いながら，ロボットを目標地点に到達するまで自律的に制御を行う機能を持つ．つまり，これを利用する強化学習タスクを作成することで，本来の強化学習タスクが考慮する必要のあった細かな制御命令の系列は，この自律移動システムの目標姿勢の指定という抽象化された要素で置き換えることが可能となる．これを踏まえて，今回定義するタスクは，自律移動の目標姿勢を指定するという連続的な行動空間を持つものとする．また，3 つ目で指摘した遷移関数の分布の変化という問題に対しては，仮説層側サブタスクと実行層側サブタスクの達成スコアの比率を基準にメタ方策 Π' の学習率を変化させるアルゴリズムを実装することで対処する．

5.2 適用するタスク

今回は，メインタスクとして，未知の環境下から目標物を効率的に発見するための移動方策を発見するという問題を定義する．また，それを部分的に達成するための基本行動として，地図探索，目標物への接近という 2 種類のサブタスクを定義する．

5.2.1 各サブタスク・メインタスクの共通設定

これらのタスクは，ROS の Navigation Stack など，自律移動の機構が背後に存在することを前提としたタスクである．この環境では，エージェントは行動 $a \in A'$ として 2 次元的な自律移動システムに相対目標姿勢 (x_a, y_a, ψ_a) を指定することを行う．このうち，2 次元の相対座標情報である x_a と y_a の値は，1 ステップに指定できる最大距離 L を上限とする範囲内に収まる値に限定する．ψ_a は目標姿勢のヨーの回転成分であり，$-\pi \leq \psi_a \leq \pi$ の範囲をもつ．また，エージェントが環境から観測する状態 $s \in S'$は，タスクの開始地点を原点とする座標系での現在位置情報 (x_s, y_s, ψ_s) に加えて，SLAM アルゴリズムによって逐次作成される Occupancy Grid Map のピクセルデータ情報の 2 つである．

このタスク設定は，エージェントの行動の度に自律移動システムによる移動制御の完了を待機するという特徴を持つ．ここでは，その待機時間を一定に保つために 1 回の目標姿勢指定で自律移動制御できる時間の上限を設定することで，一般的な連続値制御と似た性質を保ちつつ，1 回の行動の時間スケールを任意の長さで抽象化できる特殊なロボット制御方法として扱えるようにしている．

5.2.2 タスクの条件を満たす環境の自動生成

定義したメインタスク，各サブタスクには，未知の環境と目標物が用意された専用のシミュレーション環境が必要である．これを用意するため，ROS を通して利用可能な物理シミュレーションである Gazebo を利用し，エージェントの学習環境を自動生成して利用できる仕組みを実装する．

ここで必要となる要件は，生成される環境の状況が実環境下での観測と近い複雑な状態を再現することである．これは，仮説層で事前学習したサブタスクを実環境用に fine-tuning するために必要な要件であり，実行層の Sub-tasks Level で調整したサブタスクが実環境上での自律移動制御を駆使する際に想定に近い挙動を取らせるためにも重要な点である．これを踏まえ，部屋の壁の形，障害物となる 3D モデルの個数，位置情報をある条件のもとランダムに生成して Gazebo 上に配置し，かつそれらの障害物に触れることのない領域からランダムに座標をサンプリングするなどの機能を持つフレームワーク roomor を作成した．これは，生成した部屋のレイアウト情報をインスタンスとして個別に持ち，それを利用して 3D モデルを再配置することや，本来 SLAM で観測的に作成する Occupancy Grid Map を作り出すことが可能である．これを利用して部屋の構造をランダム化することにより，エージェントが観測する地図のパターンを複雑化させ，実行層内の Sub-tasks Level と Main-task Level の環境下での自律移動システムの挙動のギャップを減らすことを目指す．

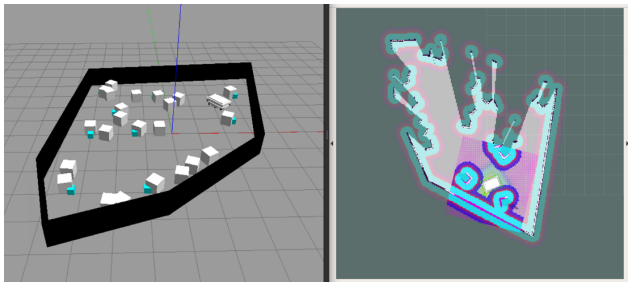

図 2　自動生成されたタスク環境

今回のタスクに合わせた環境として実際に生成された部屋の例を図 2 に示す．図 2 は，面積 70 m^2 前後の多角形の壁を形成し，障害物として正方形の白い 3D モデル 20 個をランダムに内部へ配置したあと，ある距離で任意個にクラスタリングされた障害物群の付近に 1 つずつ小さなターゲットを配置するという条件のもと自動的に生成された環境である．図の左側は実際のシミュレーション環境を表示する Gazebo の画面，右側はシミュレータ内のロボットの LiDAR センサによって取得される周囲の点群データをもとに SLAM が作成した Occupancy Grid Map を表示する RViz の画面である．

5.2.3 タスクに必要な自律移動システムの初期化

今回のタスク設定では，ROS の自律移動システムを利用した強化学習を行う．そのため，環境のリセットと同時に自律移動に利用する内部状態を初期化する必要がある．ここでは，オープンソース SLAM である hector slam [3] を利用して周囲の環境をマッピングしながら，ROS の Navigation Stack の主要なコンポーネントである move_base を利用し，目標地点までの経路計画と制御を自律的に行わせる．これらの内部状態を任意のタイミングで初期化できるようにすることで，環境が突然切り替わることがあっても問題なく自律移動制御を再開可能にすることを目指す．今回は，それぞれ対応する ROS ノードに用意された ROS Service の API を介して，必要な内部状態を初期化することで対応する．hector slam は，ROS ノード hector_mapping が公開する Service の syscommand をコールし，SLAM アルゴリズムの内部状態の初期化を行う．また，move_base に対しては，clear_costmaps をコールすることで，経路計画に扱うコストマップを初期化する．これらの初期化によって，環境内の 3D モデルの再配置に対処し，常に定常な自律移動を開始できるようにする．

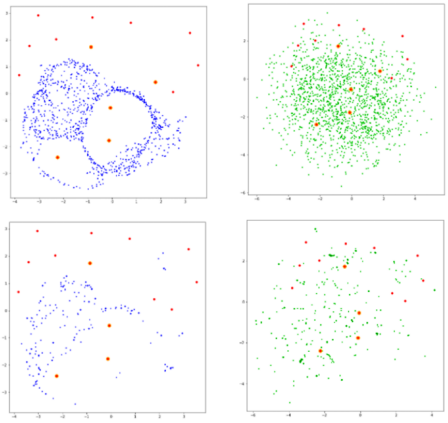

図 3　ランダム方策（上段）と学習済み方策（下段）における実際の位置（左）目標地点（右）の分布の差

5.3 環境の Gym 化

今回のタスクを扱う学習環境は，ROS 環境上ではなく，強化学習用のプラットフォームとして一般的に用いられる OpenAI Gym を介して利用できるようにすることを目指す．ここでは，実環境とシミュレーション環境の両方でシームレスに深層強化学習アルゴリズムを検討するためのフレームワークである robo-gym [4] を用いる．これは，実際のロボットを用いて深層強化学習するときの構造を，(i) ROS や Gazebo を扱う Robot Server と，(ii) OpenAI Gym のインタフェースを介して一般的な形式の環境として提供するための Environment の 2 層に分割して実装できるようにしたフレームワークである．これは，層間の通信に gRPC を用いて，robot server state と呼ばれるメッセージを相互に送り合うことで強化学習フローを実現する．このメッセージの内容は次の 2 つの役割をもつ．1 つ目の役割は，観測情報として Robot Server からロボットが観測した情報を Environment へ送信することである．2 つ目の役割は，Environment で環境のリセット条件を指定し，Robot Server へ送信することである．

今回は，メッセージの内容として，Occupancy Grid Map の地図データやロボットの座標情報など状態を表す情報と，環境初期化時に指定する各条件のパラメータを持つように設定した．これにより，環境のリセット時に生成される部屋レイアウトの条件を Environment 側から操作することと，今回のタスクで必要な観測情報の送信を同一のメッセージ構造で行うようにしている．Robot Server は，ここに含まれる情報をもとに roomor を利用したシミュレーションのレイアウト生成を行い，Gazebo 内で 自律移動システムによる制御の行うために必要な 5.2.3 の処理を実行するように実装した．また，Environment は，メッセージのうち観測情報である地図データと位置情報を 5.2.1 で示す形式に変換し，一般的な Gym 環境として前述の行動を利用できるように実装した．

5.4 実装したタスク環境の検証

実装したタスク環境を利用して，今回定義したメインタスクとサブタスクが本稿のアーキテクチャで扱うものとして適切であるかを検証する．ここでは，複雑なメインタスクを，単一の深層強化学習アルゴリズムで学習させようとした際にかかるコストについて着目する．

5.2 で定義した今回のメインタスクは，未知の環境下でターゲットを多く発見する自律移動指定の系列を学習するというタスクである．今回は，報酬設計として，新たにターゲットを発見（十分に接近）したときに+50，移動できた距離が一定以上のときに+0.05，無条件で-0.05 という設定を行い，1 エピソードの制限時間を 50 ステップとして方策の学習を行わせた．このときの学習アルゴリズムは，off-policy 手法の Soft Actor-Critic (SAC) [5] を利用し，地図画像と位置情報を状態として扱うように拡張したエージェントを 5 日間動作させ 50000 ステップだけ学習を行わせている．

学習した方策とランダム方策を同一の部屋で用いた場合の，目標設定した地点と実際に移動した地点の分布の差を図 3 に示す．実験時は，初期状態としてロボットの位置を部屋内のランダムな位置から選択してエピソードを開始させた．図中の赤い点はターゲットの座標を表し，それ以外の細かい点はステップごとのロボットの座標，または目標位置の分布を表す．この図では，ランダム方策と比べると 50000 ステップで学習した方策は目標地点として無意味な場所を指定することが少なくなっていることが見て取れる．ただし，この実験では，1 回の行動の時間スケールが大きく，120 時間経過してわずか 50000 ステップしか試行できていない．このように，複雑なタスクをオンラインで学習することは非効率であり，今回のような抽象度の高いタスクをロボットが能動的に学習できるようにするためには，我々が提案するアーキテクチャのようなメタ構造が必要である．

5.5 実行層の構築

本稿で提案するアーキテクチャは，複数の物理シミュレーションを並行的に起動しつつ，かつ複数の強化学習を同時に動作させる構造をもつ．そのため，必要となる計算リソースは非常に膨大なものとなる可能性が高い．そこで，Sub-tasks Level，Main-task Level でそれぞれ動作させるシミュレーション環境や学習アルゴリズムをそれぞれ Docker コンテナとして実行し，スケーラブルかつコンピュータクラスタ的に分散処理を行うことが可能な構造を構築する．

5.5.1 基本構造

実行層の基本構造は，各構成要素が必要な計算リソース量を考慮してコンテナ化され，相互に連携して動作するアーキテクチャとなる．以下では，今回の実装要件を反映した実行層のアーキテクチャを表す図 4 を用いて説明を行う．図中の重なっている要素は，複数存在するサブタスクのそれぞれを表しており，システム内で並行して処理されていることを意味する．

実行層における Sub-tasks Level，Main-task Level は，タスクごとに 2 つの Docker コンテナを持つ構造となる．これらのコンテナは，robo-gym における Environment と Robot Server がそれぞれ対応している．Robot Server のコンテナは，ROS や Gazebo によってタスク環境を実際に動作させるアプリケーションとして動作し，Environment のコンテナは，任意の深層強化学習アルゴリズムを実行するアプリケーションとして動作する．つまり，Sub-tasks Level

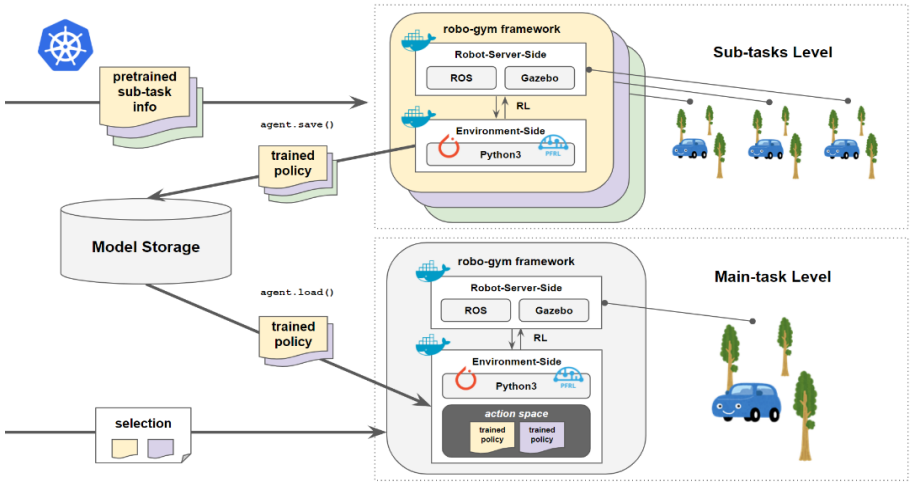

図 4　実行層のアーキテクチャ

は保有するサブタスクごとにこれらのコンテナ対を持ち，Main-task Level はメインタスクに対応する 1 つのコンテナ対を持つことになる．

事前学習されたサブタスクの情報が仮説層から実行層に渡されると，Sub-tasks Level の各サブタスクに対応する Environment コンテナがその情報に基づいてエージェントの関数のパラメータを設定し，エージェントは Robot Server を利用してシミュレーション環境での学習を再開することになる．また，実行層が仮説層からサブタスクの組み合わせ情報を受け取ると，Main-task Level はその情報をもとに Model Storage から学習済み方策を読み出し，Environment コンテナ内のエージェントの行動空間に設定してメタ方策の学習を行う．このとき，仮説層と実行層それぞれの Sub-tasks Level における各サブタスクの平均累積報酬値の比率を計算し，それをパフォーマンス指標としてメタ方策の学習率のパラメータに作用させる．

なお，ここでは Environment コンテナの構成要素として Python3 と PFRL，PyTorch を利用する例を挙げている．PFRL は，PyTorch をベースとしたオープンソースの深層強化学習ライブラリである．図中の「agent.save()」という記述は，強化学習アルゴリズムにおけるエージェントの各関数のパラメータをファイルとして出力できる機能などを用いて，学習状況をコンテナ外部へ出力することを示している．反対に「agent.load()」は，関数の重みなど表すパラメータ情報をコンテナ外部から取り込み，エージェントに適用することを示している．

5.5.2 インタフェース

我々の提案するアーキテクチャでは，仮説層と実行層の構成要素がコンテナとして並行的に動作し，相互に連携しながら自然知能を模倣した学習系を提供する．以下では，2 つの層の間のインタフェースについて説明する．

実行層が仮説層から受け取る情報は，サブタスクの fine-tuning を行うための情報，そして新たなタスクの報酬関数である．前者の内容は，学習を再開させるために必要な各関数のパラメータと，目標指標となる累積報酬値などを示す数値である．後者の情報は，モデル化された報酬関数のパラメータであることが想定される．これらのデータは構造化する必要があるため，機械学習フローで一般的に利用される HDF5 形式を用いて統一的に扱うことが望ましい．

また，今回のシステムでは，実行層側から仮説層側に報酬関数の生成を依頼し，それによってサブタスクの学習環境を新たに作成するときの構造についても考慮する必要がある．ここでは，動作中のシステムに新たなタスクの学習環境を追加する手段として，タスクに対応する新たなコンテナ対のうち Environment コンテナの実行開始時に，あらかじめ読み込んだ報酬関数のパラメータを用いて，ベースの環境クラスに新たな報酬関数をラッピングさせるという方法で実現することが可能である．このとき，仮説層側では，実行層側が発行する報酬関数生成リクエストを受け付けるための API を実装する必要がある．

6. 結論

本研究では，知能を持つ生物が本能的に行っているような知的な行動に着目し，過去の知見である学習済み方策を明示的に再利用することで抽象的なタスクに対して漸近的に対処するためのアーキテクチャを提案した．また，提案内容に適合する実環境向けのタスクを考案し，実際のロボットを利用した検討を行うための実装要件を確認した．今後は，提案したアーキテクチャの有効性を議論し，検証することによって，実世界でより汎用的に活動するための知能の実現を目指す．

参考文献

[1] Andrew Levy, George Konidaris, Robert Platt, Kate Saenko, “Learning Multi-Level Hierarchies with Hindsight”, ICLR, (2019).
[2] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, Pieter Abbeel, “Sim-to-Real Transfer of Robotic Control with Dynamics Randomization”, CoRR, abs/1710.06537 (2017).
[3] S. Kohlbrecher, J. Meyer, O. von Stryk, U. Klingauf. “A Flexible and Scalable SLAM System with Full 3D Motion Estimation”, IEEE, Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR) (2011)
[4] Matteo Lucchi, Friedemann Zindler, Stephan Mühlbacher-Karrer, Horst Pichler. “robo-gym - An Open Source Toolkit for Distributed Deep Reinforcement Learning on Real and Simulated Robots”, CoRR, abs/2007.02753 (2020)
[5] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, Sergey Levine. “ Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”, CoRR, abs/1801.01290 (2018)