B-004

Cognitive Robotics 向き研究プラットフォーム MAGI の提案

Proposal of a Research Platform MAGI for Cognitive Robotics Studies

今村 圭佑‡ Imamura Keisuke　樋口 裕次郎‡ Higuchi Yujiro　柴田 大星‡ Shibata Taisei　江平 凌太‡ Ehira Ryota　佐藤 玲於奈‡ Sato Reona　田胡 和哉† Tago Kazuya

1. まえがき

深層機械学習技術が急速に発展している。これにより、いわゆるビッグデータ等の大量のデータから法則性を抽出することが、効率的に行えるようになった。しかしながら、よく知られているように、これは、“知能”機能の一部を実現したにすぎない。ここでは、知能に関連するより広い範囲の機能を実現する方法について考えてみる。

このために、まず、自然知能に注目する。たとえば、動物が持つ知能は、自己の生存等の、自然環境における目的達成過程の最適化を図るために発達してきたものと考えられる。そこでは、自然環境と相互作用を行って情報を収集することにより、環境を自己の目的達成に利用するための、モデルとプランを、自動的に構築することが核となる。ただし、ここでの学習対象は複雑系であるから、対象を完全にモデル化することは困難であり、利用モデルが作成できればよいことになる。そこでは、シミュレーションは重要な手段とはなりうるが、最終的な狙いは複雑系から情報を抽出する装置を構築することにあり、シミュレータで真の複雑系を実現することは困難であるから、自然環境下で実際に動作する機構を実現して、モデル構築機構の有効性を確認する必要がある。以下では、このような学習系の実現を目指す活動に限定して、Cognitive Robotics とよぶことにする。

Cognitive Robotics 研究実施のためには、強化学習等の機械学習系だけでなく、ロボットのハードウェアを適切に制御する機構も必要であり、かつ、両者を有機的に連携させなければならない。両者で必要となる技術分野は大きく異なっており、システム全体を統合的に実現し、Cognitive Robotics の研究を推進することは容易でない。

このような認識のもと、Cognitive Robotics を推進するために必要な共通システム基盤を実現することを提案する。その一例として、クラウド上の深層機械学習系と機械系を接続、統合するためのミドルウェアである MAGI (Middleware for Artificial General Intelligence)を開発したので、以下でその詳細を述べる。

2. 必要機能

2.1 Cognitive Robotics の研究環境

Cognitive Robotics のための研究環境として、

- 安価な自律移動ロボットを開発する、
- 大規模な人工知能プログラムを実行し、ロボットを制御できるように、クラウドとロボット接続する、フォグコンピューティング機構を開発する、
- Cognitive Robotics の研究を行うためのフレームワークを開発する、
- IoT やロボティクスに知識がなくとも、研究が行えるようにするための GUI ツールを開発する、
- 自然環境下で、ロボットによる作業や敵対的なゲームを実行できる汎用的な知能を実現することにより、研究を実施する、

ことを試みる。たとえば、１万円程度のロボットを開発し、学生全員がロボットを所有できるようにし、クラウドを用いて人工知能プログラムの開発、試行ができるようにすることにより、多くの人が研究に参加できるようにする。

2.2 対象システム概要

Cognitive Robotics 研究促進のために、MAGI システムを構築する。MAGI で想定する対象システムのハードウェア構成の例を図１に示す。

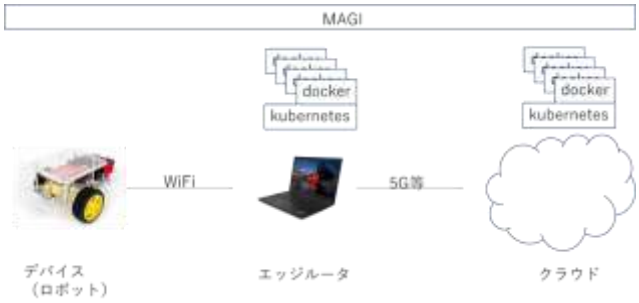

図１　システム構成例.

デバイスは、ここでは１台の自律移動ロボットを示してあるが、複数台の自律移動ロボットや、ロボットアーム等も制御対象となる。クラウドを用いて、深層学習技術を用いた大規模な人工知能システムを運用できる。デバイスとクラウドを結びつける機構として、エッジルータを導入する。クラウドに接続されるとともに、デバイスの認証を行い、安全にデバイスと人工知能システムを接続するとともに、レスポンスタイムが問題となるデバイス制御プログラムをここで実行することができる。

エッジルータも複数接続ができる。全体として、たとえば、多数の自律移動ロボットを用いた対戦ゲーム等を課題として、人工知能プログラムの検討を進める想定である。

2.3 組み合わせて利用するソフトウェアの要素

MAGI では、既存のコンポネントを部品として組み合わせて、ロボットを合目的的に行動させる系を実現する。

†東京工科大学　コンピュータサイエンス学部
‡東京工科大学　大学院　バイオ情報メディア研究科

MAGI の管理下で組み合わせされるソフトウェア要素は、以下である。

- ROS (Robot Operating System)[1]

ロボットの制御プログラムとして ROS がもっとも広く用いられており、これを利用する。要素の一部として、以下を持つ。

- Navigation stack
  SLAM と、地図情報を利用した移動制御。カメラによる visual navigation を含む
- シミュレータ (Gazebo)
  実環境のシミュレータ
- 運用監視 (Rviz)
  ロボットの運用状況を視覚化して表示

- ロボット行動制御のための強化学習系

ロボットの行動を立案、実施するための制御は複雑であるので、たとえば、階層的な構造を持った強化学習系を利用する。そこでの下位の制御系は、多種の報酬関数にしたがって動作する複数の強化学習系を切り替えて実行する機能を持つ。それぞれの強化学習が、ROS に命令を出して異なる目標を達成するように、自律的に行動する機能を持つ。

上位の制御系は、抽象的な状態空間上で行動プランを発見し、シミュレータによって検証を行う機能を持つ。行動プランは、データベース化され、再利用される。階層間は、模倣学習によって連携する。

実世界のロボットでは、設定した目標を達成する行動だけでなく、想定外の外部環境にも随時対応できる必要があるため、多数の強化学習系をいつでも動作可能な状態で維持しておく。

このような系を構築することにより、たとえば、多種のゲームを単一の機構で実現できるようにする。

- プランシミュレータ

実環境を簡略化した抽象的な状態空間上で、深層強化学習機構による機械学習系を実行し、目標を達成するための最適な手順を発見する方法を検討するための、専用シミュレータを設ける。抽象化された行動プラン発見のためのシミュレータであり、物理環境を詳細に再現したものではない。

- プログラム開発

Python 言語の開発環境が利用できるようにする。

MAGI は、以上のような多様なソフトウェアをモジュール化して、必要に応じて導入、起動、停止する機能を提供している。

2.4 システムアーキテクチャ

必要に応じて、2.3 で述べた対象システムの要素を柔軟に組み合わせてロボット制御機構を構築できるようにするために、コンテナ技術を採用する。コンテナ技術は、仮想化技術の一つであり、種々のプログラムを互いに独立に運用する機能を提供する。したがって、ここでの目的に、よく適合するシステム基盤機構である。具体的には、docker[2,3]および、そのオーケストレーション機構である、kubernetes[4]を利用する。

2.3 であげた、ROS の navigation 機構、深層強化学習機構、開発環境等要素は、個々、kubernetes の管理下の docker として実行する。kubernetes にリクエストを送る GUI を開発することにより、これらの要素を、ボタンを押すだけで実行することができるようにする。また、ロボット、GPU 等の必要資源も自動的に管理されるようにする。

これに加えて、要素間の連携を容易に行えるようにするために、python による分散オブジェクト指向プログラミング環境を提供する。システム内の任意の docker 内で実現されている python オブジェクトを、他の docker 内の python プログラムから、通常のオブジェクトと同様の方法でアクセスできるようにすることにより、システムが多数の要素からなる複雑な構成になっても、容易にプログラミングが行えるようにする。この機構と iSpace とよぶ。

Kubernetes は、通常、クラスタ内で閉じたネットワーク環境を持つ。ここでは、これを、エッジルータ内でも利用できるように仮想ネットワーク機構を用いて拡張する。さらに、wifi を経由した、デバイスからの通信の認証を行い。デバイスと kubernetes が通信できるようにしている。

これらは、全体として、クラウドと IoT システムを統合した、リアルタイム性を持ち、かつ、大規模なクラウドコンピューティング機構を含む、フォグコンピューティングシステム[5]である。したがって、システム技術の観点からも、多くの検討を要する。

3. 実装

3.1 MAGI の構成

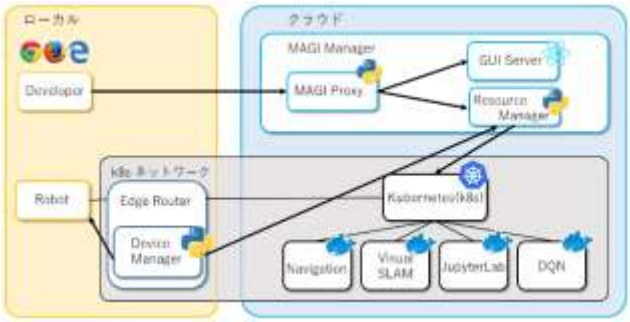

図 2　システム構成図.

MAGI 自体は、

- GUI と MAGI manager
- Resource manager
- Device manager
- エッジルータ

から構成される。

MAGI は、システムを構成する、docker、ストレージ、ロボット等を、リソースとして一括管理している。それらには URL が付加され、REST 構造によって利用を制御することができる。Resource manager は、このリソースを管理する REST サーバであり、python 言語で記述されている。Resource manager は、docker を、リソースの一つとして kubernetes API を利用することによって管理している。したがって、docker の起動、停止、namespace の作成、docker volume のアタッチ等は、すべて、Resource manager が実現している。

Device manager は、通常、エッジルータ上で動作する。ロボットデバイスごとの状態遷移を管理しており、状態の変化を MAGI manager に通知する。任意の時点で、ロボットが wifi に接続されると、それを検知して、以後の状態変化の追跡を開始するとともに、MAGI manager に通知する。これによって、自動的に、GUI 上にリソースとして表示される。開発者は、GUI を経由してこれをプロジェクトに登録することにより、自分が開発したプログラムから利用が可能になるとともに、他の利用者が同時に利用できないようになる。その後、ロボットの再ブート等の状態変化は、自動的に通知される。

Web ブラウザ上の GUI は、React[6]によって実装されており、これを、Magi manager とよぶ。Resource manager に REST によるリクエストを出すことによって動作する。

MAGI manager, Resource manager, Device manager 自体も docker として動作するので、システムのインストールは容易である。MAGI が動作する環境として、kubernetes のクラスタを必要とするが、最近では、Google, AWS 等の商用クラウドベンダがクラウドサービスとして提供しているので、動作環境も容易に準備できる。全体として、これら商用クラウドへの契約と、Windows PC を用意するだけで全体が利用可能になる予定である。

OpenAI の経験等から、このような目的の深層強化学習には、きわめて大量のプロセッサが必要となることが判明している。このため、必要に応じて、スーパコンピュータへの接続等も検討している。

3.2 Resource Manager の実装

Resource manager は、REST サーバであり、種々のリソースを管理している。それぞれのリソースを操作するためのプロキシが Python のオブジェクトによって実装されており、このオブジェクトへの操作を REST インタフェースを経由して操作するための、木構造の名前空間を持った Web サーバによって実現されている。したがって、たとえば、docker の起動は、Resource manager に、リソースのインスタンス名に対応する URI に対して PUT メソッドを送出することによって行うことができる。これによって生成されたリソースプロキシオブジェクトは、kubernetes の API を用いて自動的に docker を起動する。逆に、DELETE メソッドによってこのリソースプロキシオブジェクトを消去すると、対応する docker も自動的に消去されるようになっている。

Resource manager は、このような、直接的な資源だけでなく、メタデータ等も単一の名前空間内で統一的に管理している。この名前空間は、パーシステンシを持つデータベースとしても動作しているので、資源管理は、この名前空間を管理することによって行うことができるようになっている。

以下に、Resource manager が提供する、資源の名前空間の一部を紹介する。

- /project/{project_name}

プロジェクトごとのディレクトリ。PUT によって、作成することにより、プロジェクト自体も作成される。

- /project/{project_name}/namespace/{namespace_name}

namespace ごとのディレクトリ。PUT によって、作成することにより、namespace 自体も作成される。この下に、pvc, docker, device 等のディレクトリが存在する。

- /project/{project_name}/namespace/{namespace_name}/docker/{docker_name}

docker ごとのディレクトリ。これ以下に、docker に関する種々のプロパティが存在する。それを、書き換えることにより、docker の管理が行われる。

- /project/{project_name}/namespace/{namespace_name}/resource

namespace ごとに割り当てられた、資源のディレクトリ。edge ルータ名やデバイス名がこの下に存在する。

3.3 利用環境

利用者は、web ブラウザを用いて docker にアクセスする。X window のプロトコルで表示を行うプログラムも、web 形式に変換してアクセスできるようにしている。したがって、プログラムの開発も、シミュレータの状況監視も、その他の制御も、すべて web ブラウザから行うことができるようになっている。また、プログラミングも Jupyter を用いているので、web ベースで行うことができる。

これらの web サービスは、kubenetes の service として提供されているので、インターネットがあれば、どこからでも利用できる。実機のデバイスを利用せず、ROS の Gazebo シミュレータを利用する場合は、これだけで、研究に必要な環境が揃う。

実際にロボットデバイスを利用する場合は、エッジルータが必要になる。エッジルータは、現状では、IPsec 拠点間 VPN を用いた、いわゆる Virtual Private Cloud(VPC)構成で接続しているが、エッジルータ自体がプライベートアドレス内にあっても kubernetes に接続できるように、リモートアクセス VPN への対応を進めている。

エッジルータは、現状では小型の Linux PC を利用しているが、2020 年後半からは、Window システム上で docker が利用可能になる予定であるので、Windows を用いたノート PC 等を標準的な構成として想定する。この場合は、汎用 IDE として広く用いられつつある VSCode も利用することができるようになる。フレームワークの開発等、大規模な開発の場合は、このような方法で行う。

3.4 ISpace の実装

iSpace は、MAGI のアプリケーションとなる、大規模な人工知能プログラムの実装を容易にすることを目標とする。多数の docker が用いられるだけでなく、デバイス上のプログラムも連携して動作するので、複雑な分散処理プログラムとなる。その形態に一定の枠組みを与える必要性が高い。

iSpace は、オブジェクトを登録することができ、分散共有が可能な Python 言語のディクショナリ変数である。ここにオブジェクトを登録すると、そのクラスにあるメタ情報に従い、iSpace が以下のいずれかの方法でオブジェクトを利用できるようにする。

- オブジェクトの登録側のノードで動作させ、リモートメソッド呼び出しで分散透明に利用する
- iSpace のサーバノードで動作させ、リモートメソッド呼び出しで分散透明に利用する

- 利用側で、オブジェクト実体を新たに作成し、分散共有しない

いずれの方法も、その実装は自動的に選択され、利用側はすべて同一の方法でオブジェクトにアクセスできるようになる。

この機構を基盤とし、分散処理で典型的に頻繁に利用される、同期・情報共有を行うライブラリクラスも用意する。これにより、相互排除、キュー、イベント伝達、ストリーム通信等が行える。また、利用者が作成したオブジェクトも登録することができる。

簡単な例をリスト 1 に示す。

リスト 1　iSpace サンプルコード

```
1  import ispace
2
3  sp = ispace.iSpace()# sp が分散共有されたディクショナリ
4
5  value = sp["sample"]  # value に”sample”という名のオブジェクトが保持された
6
7  value.doTask()  # 取得したオブジェクトは通常の方法でアクセス可能
```

4. 実装されたシステム

MAGI が現在実際に運用されている。そこでは、あらかじめ、要素となるプログラムを docker image の形式で用意しておくことにより、利用者は、個々の要素に関する深い理解がなくても利用できるようになる。特に、python 言語の標準的な知識があれば、全体を構築できるようにすることが目標となる。その利用法の概要を紹介する。

4.1 画面例

MAGI は現在運用可能な状態になっている。その画面例を示す。MAGI は、“namespace”とよばれる単位でプログラムを開発する。namespace には、コンテナ, PVC, デバイス等のリソースが所属する。namespace を管理するための画面を図 3 に示す。コンテナは実行中のプログラムであり、

図 3　namespace の表示.

画面の”コンテナ一覧“に実行中のプログラムのリストが表示される。PVC は、namespace 内で共有されるストレージである。デバイスは、ロボット等の外部装置を示す。ロボットの状態は定期的に監視されており、画面上に、Connect (利用可能)、Disconnect(利用不可)等の状態が表示される。

Namespace に属するプログラムを追加するのは、新たなコンテナを起動することによって行う。図 4 に示すように、起動可能なコンテナイメージのなかから、必要なものを選択するだけで、コンテナが起動する。この例では、プログ

図 4　コンテナの起動画面.

ラム環境(Jupyter)、ROS navigation stack(gopigo3-navigation)等が選択可能になっている。

ここで、Jupyter コンテナを選択すると、プログラム開発環境画面が起動するので、そのまま開発を始めることができる。また、新しいライブラリの開発等、大規模な開発では、エッジルータ上で実行される汎用 IDE である VSCode を用い、遠隔の開発用 docker を操作しながら作業を行えるようになっている。

4.2 利用手順

MAGI は、利用者個人が kubernetes を用意して、数人のグループで利用することを想定している。大規模なサービスサーバとして運用することは想定していない。したがって、その設置が容易に行えることは重要である。MAGI は、単純なスクリプトにより、kubernetes 上で主要コンポネントを起動することによって、実行することができる。

これによって、web によるシステム管理画面が、アクセス可能になる。利用者を登録し、再度アクセスすると、4.1 に示した画面が利用可能になる。プロジェクトと namespace を作成し、Jupyter もしくは VSCode を用いたプログラミングが可能になる。namespace 内ではファイルが共有できるので、Git も併用してプロジェクト管理を行うことにより、大規模なソフトウェア開発にも対応可能である。

4.3 実装の状況

MAGI は、主に Python 言語で記述され、実際に運用可能な状態になっている。実際には、GoPiGo3 ロボットを数台用意し、必要に応じて接続試験を行っている。また、マニュアル等、広く利用するための準備を進めている段階である。この過程で、アプリケーションプログラムの開発に関しても、試行を行った。

今後、オープンソース化することを検討している。

5. フォグコンピューティング向け資源割り当てのスケジューリング機構の検討

MAGI の運用の過程で、その動作に関する種々の知見が得られつつある。ROS で制御されるロボットの場合、ROS によるナビゲーションプログラムと ROS の接続においては、数ミリ秒の遅延しか許容されないため、クラウドのサーバの設置位置によって、動作する場合と動作しない場合が出てくる。実際に、海外のデータセンターを利用した場合、動作しなかった。

また、たとえば、ROS の Gazebo シミュレータは、シミュレーション実行時の CPU 消費が大きい。自動的な CPU 割り当てでは、この点が考慮されず、複数利用者がシミュレーションを行うと、不適切な CPU 割り当てが行われ、複数のシミュレータが同一の CPU 上で実行されてしまうことが判明した。

MAGI 運用の経験に基づき、現在、フォグコンピューティング向き資源割り当てスケジューリング機構について検討してみる。フォグコンピューティング環境の、資源割り当て上の問題点として、

- 多種の計算機が関与する環境の下、リアルタイム性が重視され、システム性能の指標が従来のものより複雑である
- 複数のシステム基盤にまたがって構築されるために、全体として適切な資源割り当てを保証する機構が存在しない
- 個々のソフトウェアの特性を考慮した資源割り当てが行われない

等があげられる。これに対応するためには、新たなアプローチを検討する必要がある。

このために、まず、アプリケーションプログラム側も資源割り当てスケジューリング機能の一部を分担することにする。複雑なアプリケーションが全体として所定の性能を得ているのか、否かは、システムでは判定することが不可能で、アプリケーションプログラム自体が判定を行う必要がある。アプリケーションプログラム自体が、性能判定を行い、それに基づいて不足する資源を判定し、システムに要求を行う。

たとえば、通信機構の性能に関して、TCP/IP の特定のコネクションのレスポンスタイムが、全体の性能に大きく影響を与えることはありうる事態である。アプリケーションプログラム自体がそれを観測して、特定のコネクションに関するレスポンスタイムの改善をシステムに要求する。

システムは、要求をそのまま鵜呑みにするのではなく、アプリケーションプログラムの性能と、要求された特定資源の関係を別途観測し、要求を参考としながら、あらたな割り当てを行うべきか否かを決定する。このように、アプリケーションプログラム側のスケジューラと、システム側のスケジューラが交渉を行いながら資源割り当てを行う方法をとる。

このような割り当ての対象となるのは、

- CPU
- メモリ
- 通信回線
- ファイルシステム

が主となる。CPU とメモリの割り当ては、OS の機能によって制御することができる。アプリケーション側は、要素となる各ソフトウェアの特性を把握した上で、必要資源の予測値や、CPU 共有の可能性を加味した資源要求を、システム基盤、ここでは、kubernetes に対して行う。Kubernetes は、システム基盤としての資源割り当てを、OS の機能を利用しながら実現する。

通信回線の帯域幅割り当てやレスポンスタイム管理に関しては、VPN 機構と統合することを検討している。通信のレスポンスタイムは、通信媒体自体の遅延による絶対的な条件と、通信資源の多重化のスケジューリングに依存して決まる。絶対的な遅延は、アプリケーションが測定し、オペレータに情報適用する以外にできることはない。一方、通品媒体を多数のコネクションが共有している場合、コネクション間での帯域幅の割り当ては、制御することが可能である。具体的には、一定時間ごとに、コネクションごとの通信量を監視し、レスポンスが問題になるコネクションに関しては、必要な通信量を割り当てる方法をとることにより、レスポンスを保証することができる。VPN では、全コネクションの通信量を集約的に把握できるので、このようなスケジューリング機構が実現できる。

ファイルシステムに関しては、バージョン管理を導入することにより、レコードを不変化した分散ファイルシステム Elton を開発した。バージョン管理機能を持つファイルシステムは、コンテナ機構ときわめてよく整合するので、今後利用してゆく予定である。レコードが不変になることにより、ssd 等の高速二次記憶装置をキャッシュに利用することが可能になる。このような、二次記憶装置によるキャッシュのスケジューリングも、連携して運用できるようにする。

これらの資源に関して、アプリケーションのスケジューラが、過不足を判定し、優先度を付けてシステムに要求できるような、API を作成する。アプリケーションの、資源利用状況は、MAGI の Resource manager が把握している。したがって、Resource manager に、スケジューリング機能も付加し、動作基盤となる kubernetes 上の専用スケジューラと交渉を行うような機能を検討する予定である。

以上のような、実行時のスケジューリングと同時に、システム構成に関する設計支援も重要である。たとえば、最近では、ロボットに搭載できる小型の深層学習アクセラレータも出現している。このような装置の利用も想定した、性能の異なる計算機による分散処理環境を最適化するためには、アプリケーションがログをとり、後に分析できるようにする必要がある。また、機械学習技術の利用も有効であると考えられる。システム設計の支援機構の実現を検討している。

6. 人工知能研究への利用例

MAGI を用いた研究の一例として、現在進行中のプロジェクトである、汎用性のある知能で制御された自律移動ロボットによる対戦ゲームの実現をとりあげる。２台、もしくは複数のロボット間で対戦を行う。ロボットから特定のパターンを持った赤外線をビーム発光し、そのパターンを受光した側に負の報酬が課せられることにより、対戦ゲー

ムを構成できる。対戦の実施には、索敵、対戦等の行動を効率よく実行できる必要がある。

ここでは、たとえば、全体を一つの強化学習によって実現するのではなく、階層的な構造を試みる。下位の層では、ROS に行動コマンドを発行することにより、ロボットの行動を直接制御する。複数の、内発的報酬を含む多数の報酬関数を設定し、それぞれに関する強化学習を並列的に実行することを考える。実際の行動は、この中から一つの報酬関数を、上位層が選択し、強化学習によって実行される。

報酬関数を複数用いることにより、複数の、ロボットがとりうる行動パターンを報酬関数単位で記号化することができる。上位の層はプランシミュレータを用いて、目的を達成するための行動の手順を発見する。これを、複数の報酬関数を特定の順番で切り替えることによって実現する。ここで用いられた報酬関数の列を、手順とよぶことにする。

複数の強化学習系を同時に運用することは、予想しない外部環境の変化に対応する目的も有している。ロボットが実環境で動作する場合、自発的な行動の結果だけでなく、他者の行動の結果にも対応しなければならない。これは、通常の機械学習の運用条件とは大きく異なる点である。

手順の発見は、行動プランニングの一種と考えることができる。ロボットの行動プランニング[7]は、古くから検討されてきたが、最近、大規模な深層強化学習機構を用いることにより、直接的に複雑な行動を生成することも試みられている。たとえば、マルチエージェント環境下で、高度な行動戦略が自動的に生成されることを示したものとして、OpenAI による、team-based hide-and-seek game の試み[8]があげられる。そこでは、きわめて高度な行動戦略を敵対的方法で発見することに成功している。しかしながら、逆に言うと、深層学習を用いた方式であり、そこで発見された戦略手法を別の条件で再利用したり、知見をデータベース化したりすることは容易でない。行動戦略考案等の高度な検討では、毎回ゼロから検討を行うことは、計算量の点で現実的でない。戦略に関する何らかの抽象的な共通状態空間の設定と、その上での戦略事例の蓄積が不可欠であると考えられる。このためには、記号化の問題を避けて通ることは不可能である。

したがって、上位層は、単一の方法で実現するのではなく、複数の方法を組み合わせて実現する必要があると考えられる。現在は、主にこの点に関して、MAGI を用いて試行錯誤を行っている段階である。

さらに、上下の層の連携の方法が検討課題となる。たとえば、下位層では、外的な例外イベントが起きる可能性のある現実環境を対象とした行動制御を行いながら、抽象的な戦略に従って行動とれるようにすることが目標となる。このように、実際のロボットでは、単一の機械学習系で制御されるとは考えにくく、現実世界における行動を制御するための一般性のあるアーキテクチャが新たに必要になることが予想される。

このような認識のもと、階層構造を持ったロボット行動の制御機構のフレームワークである、Spear の開発を進めている。下位層では、OpenAI Gym[9]による強化学習系の枠組みを用い、それを複数管理できるようにしている。上位層では、プランシミュレータを実装し、手順の発見を行えるようにしている。

7. むすび

Cognitive Robotics の研究推進を目的として開発された、MAGI を紹介した。CEATEC 2017, 2018 でプロトタイプを展示したが、本番システムの開発も進んでおり、ドキュメント等を整備した後、オープンソースとして公開の予定である。また、ここで用いるロボットは、標準的な Turtlebot や GoPiGo を想定しているが、さらに安価な、1 万円程度のロボットの開発も試みている。

今後、開発基盤の開発と並んで、Animal-AI Olympic[10]にみられるような、研究におけるベンチマークとなる標準アプリケーションの策定も重要になると考えられる。これらを整備することにより、多くの研究者が Cognitive Robotics 研究に参加することがより容易になると期待される。これに向けて、努力を行いたい。

参考文献

[1] Open Sourece Robotics Foundation, “ROS wiki”, http://wiki.ros.org/, (2020/04/30).
[2] Docker, “What is Container”, https://www.docker.com/resources/what-container, (2020/04/30).
[3] Docker , “Docker”, https://www.docker.com/,(2020/04/30).
[4] Kubernetes, ”kubernetes”, https://github.com/kubernetes/kubernetes, (2020/04/30).
[5] P. Bellavista et al, A Survey on fog computing for the Internet of Things, Pervasive and Mobile Computing, No.2 (2018).
[6] React, “React”, https://reactjs.org/,(2020/05/03).
[7] G. Konidaris et al., From Skills to Symbols: Learning Symbolic Representations for Abstract High-Level Planning, JAIR 61 pp.215-280 (2018).
[8] Bowen Baker et al., “Emergent Tool Use from Multi-Agent Autocurricula” , https://arxiv.org/pdf/1909.07528.pdf, (2020/04/30)
[9] OpenAI, “OpenAI Gym”, https://gym.openai.com/, (2020/04/30).
[10] Animal-AI Olympics, “The Animal-AI Testbed”, http://animalaiolympics.com/AAI/, (2020/04/30).