

意図の概念を持つ自律エージェントモデルを用いた 実世界ロボットの知的制御

藤田 恵^{†1} 片山 寛子^{†1} 小島 侑子^{†1}
隅田 麻由^{†2,*1} 小山 由^{†2,*1}
新出 尚之^{†2} 高田 司郎^{†3}

本研究では、自律エージェントを用いた実世界上のロボットの制御について述べる。エージェントの設計モデルには BDI モデルを用い、これによりエージェントが目標達成のために一貫性のある、合理的な行動をとることが可能である。我々はこれを用いて、複数の機能の異なるロボットが予測しないタイミングで相互にコミュニケーションを取って協調行動をとる実験を行った。この実験では、シミュレーション上でのエージェントと実世界上のロボットの振る舞いとの間の差異により生じる問題点があり、それに対処する必要があった。我々はこの実験を通じて、我々の実装におけるモデルの適用が現状のロボティクス分野への応用に対して有効であることを述べる。

The intelligent control of robots in the real world using the autonomous agent model system with the concept of intention

MEGUMI FUJITA,^{†1} HIROKO KATAYAMA,^{†1} YUKO OJIMA,^{†1}
MAYU SUMIDA,^{†2,*1} YOSHI KOYAMA,^{†2,*1} NAOYUKI NIDE^{†2}
and SHIRO TAKATA^{†3}

We describe a way of robot control in the real world using autonomous agent. Since we use the BDI model for designing agents, we can develop agents which can act rationally and consistently toward their goals. We conducted the experiment to make robots with different functions cooperate using communications between each other at unpredictable times. In that experiment, we found problems which occurred from the disparities between the agents in the computer simulation and the robots in the real world, and we dealt with these problems. Through that experiment, we show that the way we apply the BDI model to our implementation is effective against the current application of the robotics area.

1. はじめに

近年のロボット技術の進展に伴い、生活や産業の場で、多様なロボットが実用化されている。そうした中で、特に現在、必要性和期待が高まってきているのは、ロボットの知的制御に向けた技術、たとえば、多様に変化する実世界において、障害を検知しそれ避けて目的地への経路を探索したり、物体を特定の条件に従って配置し直したりするなど、ロボットが柔軟に対処できる知的な能力の実現である。具体的には、ロボットが環境と相互作用しながら、多様に変化する環境に適応して柔軟に行動できるプランを随時、構築・実行することで、与えられた目標を達成するような知的制御技術である。

これまでに実用化されているロボットの多くは、単一の特定機能に特化されたものであり、自分の持つ能力をうまく発揮することが使命で、目標達成のために行動を組み立てることは要求されていなかった。また、ロボティクス分野でも、従来の研究の主流は、歩くために体の釣り合いを取るなど、自分の持つ技術をうまく適用することに重点が置かれている。現実のロボット開発の分野では、知的行動は多数の条件分岐の連鎖を手作業でプログラムすることで実装されている現状もあり、ロボットの知的行動の実現に向けた研究は十分とは言えない段階である。

我々は、多様な環境下で柔軟に目標を達成する知的制御技術として、BDI モデル⁹⁾ に基づいた BDI エージェント¹⁰⁾ による制御が有効であると考えている。BDI エージェントは、自律エージェントの実現の 1 つであり、目標を達成するための行動の選択として「意図」を保持し、これに沿って行動することで一貫性を保った目標達成への行動が行えるものである。

2.2 節で詳述するが、意図を保持することの利点は、目標を設定だけでは状況によって目標達成までの経路が変わってしまい、複数の経路の間を振動して目標へ到達できないことが起きうるのに対し、意図の保持によって特定の経路に専念し、一貫して目標へ向かえる点にある。

^{†1} 奈良女子大学大学院
Graduate School of Humanities and Sciences, Nara Women's University

^{†2} 奈良女子大学
Nara Women's University

*1 現在、奈良先端科学技術大学院大学
Presently with Nara Institute of Science and Technology

^{†3} 近畿大学
Kinki University

BDI エージェントは広く用いられているが、その応用はこれまでソフトウェアエージェント主体であり、実世界のロボットの行動制御への応用例は後述するようなわずかなものしか見当たらない(対話ロボットへの応用としては文献 13) などもある)。しかし、近年は LEGO MINDSTORMS NXT³⁾ に代表される、プログラムで制御可能でしかも安価なロボットが入手可能になり、今後はそうしたロボットの日常生活などへの浸透も期待される。従って、実世界ロボットの知的制御に BDI エージェントを応用することは有望と考えられる。

我々は、BDI エージェント構築のための言語 AgentSpeak⁸⁾ のフリーな処理系である Jason¹⁾ と、Python で書かれフリーで配布されている NXT の低レベル制御のライブラリを用いて、BDI エージェントによる行為の決定によって目標を達成するロボットの制御方式の実装を行った。本論文ではそれについて述べ、またその有用性を示すため、2 台の能力の異なるロボットの協調によって目標を達成する実験の例について述べる。

本論文で示す実装は、実世界のモデル化としてはよく用いられるグリッドワールド上での行動を前提として設計している。そこで、まず上述の Python による NXT 制御ライブラリを用いて、1 マスの移動や 90 度の方向転換、センサによる障害物知覚などを実装し、Jason 側での基本行為として利用可能にする。この基本行為を用いて Jason 側では、エージェントの行動のためのプランを記述する。Jason と Python は TCP/IP で通信し、Jason 側から命令を送信することで Python 側では対応するメソッドを実行し、Python は知覚結果を Jason に渡す。これにより、基本行為を実行する部分と高レベルでの行為列決定の部分を分離し、それぞれを独立に改良できる。詳しくは 3 節で述べる。特に、基本行為を実行する部分については、文献 11) で別途述べるように実行時の誤差の問題が発生することは避け難いが^{*1}、この点を制御技術の向上などで改良すれば、我々の制御方式の有用性もそれだけ上がることが保証されることになる上、高レベル行動の部分はその影響を受けずに汎用的に設計することも可能である。

先述した BDI エージェントのロボットへの制御の他の例としては、同じく NXT に適用した文献 6) や、その前身の RCX を用いた文献 4) などがある。詳しくは 7 節で論じるが、文献 6) は同機能の多数のエージェントが入札により作業を分担するのに対し、我々のものは異なる機能のエージェントの協調を扱う。また、文献 4) においては環境全体を統括する世話役相当のエージェントが存在し、各エージェントはそれとの通信により行動を決めるが、

我々のものは、各自独立した目標のもとで行動しつつ必要時にコミュニケーションを行うエージェントを実現しており、この方が人間の通常の振る舞いに近く、より一般的である。

2. BDI エージェントの概要

本節では、まず、BDI エージェントの概要として、BDI モデルと BDI アーキテクチャについて述べる。次に、BDI エージェントを実装したエージェントプログラムを実行する Jason インタプリタの概要を述べ、最後に、本論文で BDI エージェントによる制御の対象とする教育用ロボット LEGO MINDSTORMS NXT の概要を述べる。

2.1 BDI モデル

BDI モデルとは、B (Belief)、D (Desire)、I (Intention) の 3 つの心的状態とその時間的变化を用いて意思決定を行うエージェントモデルである。

Belief は信念、すなわちエージェントが信じている実世界に関する不確かな知識(計画を含む)、Desire はエージェントの願望、そして Intention は意図、すなわちエージェントが目的を達成するために行おうと目指している心的状態である。

環境に関する知識(信念)に基づき目的を達成しよう振る舞うエージェントを合理的エージェントと呼び、BDI モデルは合理的エージェントの一つである。

この BDI モデルは Michael Bratman の意図の理論に基づいて提唱されたものである²⁾。意図の理論では、われわれ人間は、目的を達成するためには、まずどのような目標を達成すべきかを熟考し、次にその目標はどのような手段(プラン)を用いれば達成できるかという目的-手段推論を行う。この熟考と目的-手段推論を合わせて実践的推論と呼ぶ。合理的エージェントは、目的を達成するために実践的推論を用いて選択されたプランを意図という心的状態で形成し、この意図をいつまで保持すべきかを定めたコミットメント戦略に基づいて、このプランの目標を達成するまで意図を実行することで、目的を達成する。

2.2 BDI アーキテクチャ

BDI モデルに基づくエージェントアーキテクチャを BDI アーキテクチャと呼び、そのアーキテクチャを基に実装されるエージェントが BDI エージェントである。BDI エージェントのインタプリタは一般的に次のようなループで実装される¹⁰⁾。

BDI-interpreter

```
initialize-state();
```

```
do
```

```
options := option-generator(event-queue, B, D, I);
```

*1 本論文では、高レベル制御の部分に議論の主眼を置くため、実行時の誤差については考慮せず、十分なチューニングでのみ対応している。

```
selected-options := deliberate(options, B, D, I);
update-intentions(selected-options, B, D, I);
execute(I);
get-new-external-events();
drop-successful-attitudes(B, D, I);
drop-impossible-attitudes(B, D, I);
```

until quit.

ここで、B は信念、D は願望、I は意図をそれぞれ表すデータ構造である。ただしエージェントが自ら願望を生成することは考えづいため、BDI エージェントの願望とは、具体的には、エージェントに外部から与えられた達成すべき「目標」と考える。

初期状態が与えられると、まず、event-queue を読み、エージェントに対する依頼（願望）であれば、実践的推論を行って、（未来に実行すべき）意図を形成する。そして現在、保持している意図の中から今の状況にて実行可能な意図の候補を options に返す。次に deliberate を用いて、次の時刻に実行すべき意図を options から熟考して一つ決定する。次に、選択された意図が持つプラン本体（行為列）から次に実行すべき行為を決定するために、意図の更新を update-intentions にて行う。そして、決定された行為が基本行為であれば execute にて実行する。そうでなければそれは副目標であり、新たな目標（願望）として event-queue に追加する。次に、get-new-external-events により環境知覚器から環境に関する情報を得て、event-queue に追加する。最後に、コミットメント戦略に基づき、成功裏に目標を達成した意図を破棄する。また、不整合の原因と判断された意図や、これ以上保持することが困難だと判断された意図を破棄する。BDI エージェントは、このループを繰り返すことにより、複数の目標（目的）を達成する（図 1）。

われわれが住む環境は非常に複雑で常に変化しているため予期せぬ事態が発生し、事前に立てた計画は全く無駄に終わるかもしれない。そこで、BDI エージェントは最初から詳細なプランを立てるのではなく、まず環境の変化に柔軟に対応できる程度の粒度からなる副目標から構成されるプランを意図として形成する。そして、その意図を実行すべき時が来たと判断したときに、環境に適応してそれら副目標を達成する（サブ）プランを実践的推論することで、実行すべき意図のプラン本体を詳細化する方式をとる。そのため、多様な環境変化に柔軟に適応した意思決定を行うことができる。

また、意図を持たないエージェントアーキテクチャでは、例えば、目的地への経路が 2 つあり、いずれも一旦目的地から遠ざかる場合、単に目的地に近づくことを目標とするタスク

では、この 2 つの経路の間を振動して、目的地に到達できないような現象も起こる。これに対し、ある 1 つの経路を通して目的地に到達することを実践的推論し意図として形成すれば、下位ではその意図を達成する副目標を詳細化し、順次、そのプラン本体を実行することで、目的地に到達することができる 12)。このように目的を達成する意図を保持することにより、上記の振動するような現象を回避できることも BDI アーキテクチャの利点である。

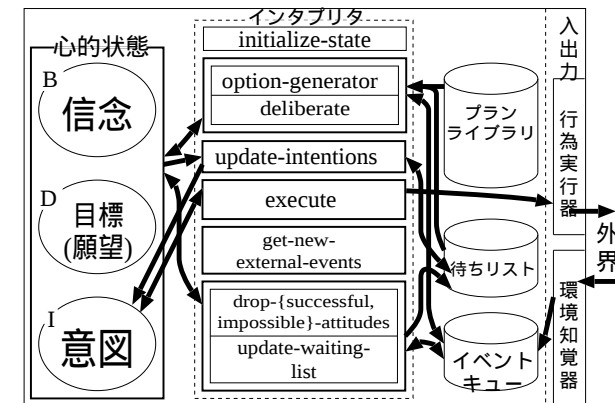


図 1 BDI アーキテクチャの概要図

2.3 Jason インタプリタ

Jason とは、BDI アーキテクチャを基礎とした AgentSpeak 言語（の拡張）のインタプリタであり、Java 言語で実装されている。Jason を用いれば、エージェントの行動やプランを Prolog 風の節形式で宣言的に記述できる。また、提供されている環境に関する API を用いて、環境モデルを Java 言語で実装することも可能である。MAS 定義ファイルによって、システムがユーザの定義した環境のうちのいずれを用いるか、またシステムにどの種類のエージェントがいくつ存在するかを指定する。これにより、その環境とエージェントの相互作用が可能となる（図 2）。この相互作用により、エージェントは環境を知覚することが可能となり、その知覚をエージェントの信念として信念ベースに保存する。これを元にエージェントは自身の行動を計画し、それを意図として扱うことにより合理的かつ自律的に行動することが実現できる。また、エージェントの基本行為（特に、環境とのやりとりを行うようなもの）の定義も上述の環境モデルのプログラム内で行える。

今回我々は、ロボット制御エージェントを Jason で実現するにあたって、環境モデル内に画面上でのシミュレータを併設し、エージェントのおかれた環境の計算機上での可視化 (2次元) を行った。

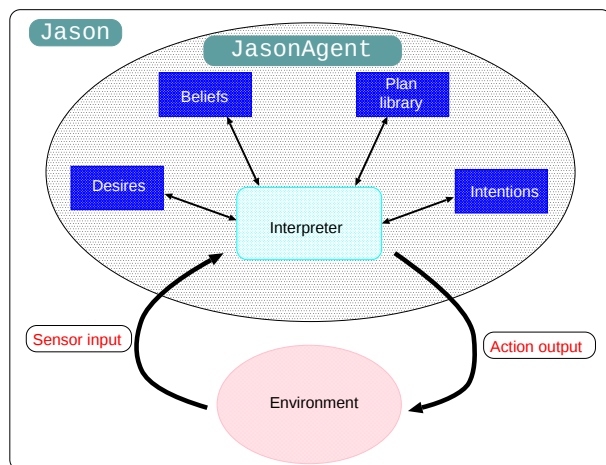


図 2 Jason と環境との相互作用

2.4 LEGO MINDSTORMS NXT

LEGO MINDSTORMS NXT は LEGO 社が開発した、商用教育用のロボットであり、安価で比較的制御が簡単であるので、実験で用いることには適したロボットである。NXT はたくさんのパーツを組み立てて一つのロボットを作成するので、ロボットの形状はさまざまなものに上げることが可能である³⁾。

今回作成したロボットは NXT の組み立て用ホームページを元に作成したもの⁷⁾で、前方にタイヤが二つあって後方に一つタイヤがあり、前方のタイヤを回転させることにより、前進したり回転したりすることが可能である (図 3)。

また、周囲の知覚を得るために、超音波センサをロボットに搭載した。よってロボットは前方に超音波を流すことにより、前方の壁あるいは障害物を感知することができる。他にも、タッチセンサと呼ばれる、衝突すると衝突認識を行うことができるセンサも搭載することができ、これにより衝突したことを知覚することが可能である (図 4, 図 5)。

さらに、NXT は Bluetooth による無線通信をすることが可能である。Bluetooth で PC との通信を行うことによって、PC から遠隔的に指令を送ることができ、また知覚情報を PC 側で受信することが可能である。

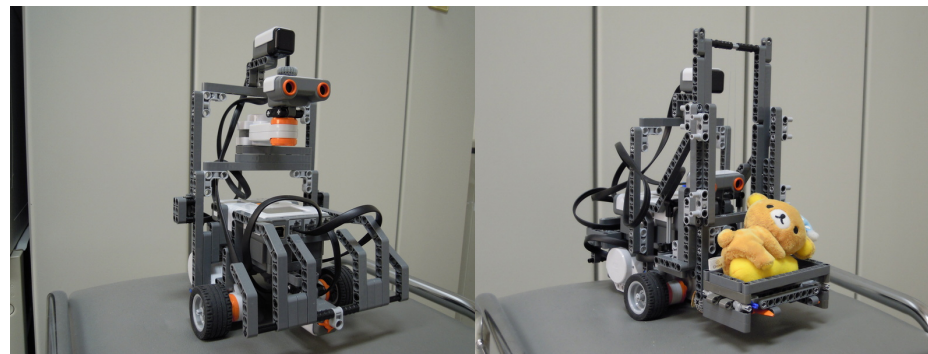


図 3 今回使用したロボット

3. 自律ロボットのアーキテクチャ

ここでは、4 節での実験に用いたロボットと、それを制御するエージェントの実装について述べる。

3.1 ロボットの特徴

我々は次のような 2 種類のロボットを作成した。これは、能力の異なるロボット同士が協調して目標を達成するという問題設定を行うためである。

- Explorer

Explorer は図 4 のような形状をしたロボットである。方向知覚用のコンパスセンサ、障害物知覚用の超音波センサおよびタッチセンサを持つ。

- Fork Lift

Fork Lift (以下、Lift) は図 5 のような形状をしたロボットである。Lift はコンパスセンサおよび超音波センサを持つが、タッチセンサは持たない。また、フォークリフトを備え、フォークリフトの上に乗せている箱を物体の上に置く能力を持つ。

図に示すように、Lift の持つ超音波センサは Explorer の超音波センサよりも下の方に取り付けられている。したがって物体の高さによっては、Lift の超音波センサでは知覚できるが、

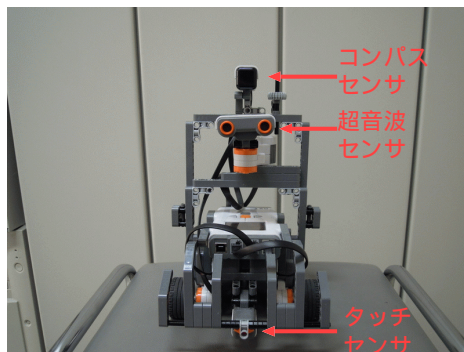


図 4 Explorer 正面図

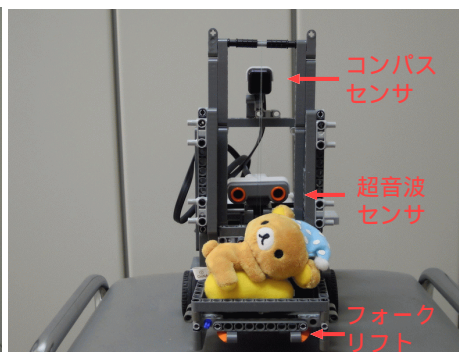


図 5 Lift 正面図

Explorer のそれでは知覚できない。しかし、そのような物体について Explorer はタッチセンサによって知覚可能であり、したがって高い物体と低い物体を知覚し分けることができる。これに対し、Lift はそれができず、知覚した物体の高さの判定は Explorer に委託する必要がある。

3.2 ロボットの行為実行・環境知覚

NXT には制御用のツールがいくつか存在するが、その一つとして Python による低レベル制御のライブラリ NXT_Python⁵⁾ がフリー (GPL) で配布されている。このライブラリは NXT のサーボモータを動かしたりセンサを読み取ったりするなどの低レベルな制御命令を与えるものである。

そこで我々は、一定距離の前進や一定角度 (90 度の倍数) の回転を行うメソッドや超音波センサによる前方をセンシングするというメソッドを、NXT_Python を呼び出すことで作成し^{*1}、これをエージェントにとっての基本行為の実装に用いた。

我々が作成した Python による NXT 制御コードでは、right や left などそれぞれの行動を表す文字列をキーワードとし、キーワードが与えられるとその行動を行うためのメソッドを呼び出して、これがさらに NXT_Python ライブラリを呼び、低レベルの制御命令を NXT に通信する。この行動とは、前進、後退、左回転、右回転、前方知覚と分けており、左回転と右

回転は 90 度回転と 180 度回転を行えるように設計し、実装した。Lift に関してのみ、フォークリフトを持ち上げて台の上に荷物を乗せるという機能にあたる、lift_up というキーワードも使用できるようにしている。これ以外に Explorer にはロボットの正面にタッチセンサを設けており、タッチセンサに物体があたると、衝突したことを認識するようになっている。

また、ロボットの行動及び知覚を表すクラスを設け、すべての行動や知覚に関するメソッドはこのクラスのメソッドとしてまとめている。このクラスのオブジェクトが、コンパスセンサで検知したロボットの最初の方向を保持することによって、常に最初の向きを基準に 90 度単位で回転できるようにしている。回転のメソッドは、回転角の誤差が一定以下になるまでコンパスセンサを用いて回転角の微調整を行う。

3.3 知的制御のための分散処理方法

NXT は小型のロボットであるため計算資源も少なく、知的制御を行うに当たって、エージェント全体を NXT に搭載することは現実的でない。このため我々は、図 1 のうち、行為実行器と環境知覚器のみ NXT のものを用い、残りは PC 上の汎用的なプラットフォームを用いることで自律エージェントを構築している。

Jason で構築されたエージェントと、Python で書かれた我々の NXT 制御プログラムは、ソケットを用いて TCP で通信を行う。これにより、両者の開発マシンや開発環境を分離することが可能である。また、Jason 側が送信する行動命令を、Python 側での NXT の制御の実装とは別に高レベルで設計でき、Jason 側は、NXT が実際にはどのような言語・ライブラリおよび通信手段によって制御されているかを気にすることなく、行動レベルでの制御のみを行えばよい利点が生まれる。また Python 側も、上位とは独立に行動制御の改善などが行える。

Python 側には jasoncomm クラス、Jason (JAVA) 側には nxtcomm クラスを用意した。jasoncomm クラスのインスタンスを作成すると、それがサーバとなって Jason 側からの接続を待ち受ける。nxtcomm クラスのインスタンスを作成すると、クライアントとなって Python 側へ接続する。

通信は行単位でテキストにより行われることを想定しており、一方から他方に 1 行のテキストを書き込むと、もう一方で読み出し操作によりそれを取得できる。これにより、

- (1) Jason 側は行動を決定し、Python 側へ行動命令を 1 行で送信する
- (2) Python 側はそれを受信し、NXT に実際の行動を制御する命令を送信して行動させる
- (3) NXT からの何らかの環境検知を Python で受信
- (4) Python がそれを環境に関する高レベルの知覚として Jason 側に 1 行で送信し、Jason

*1 これらのメソッドは、NXT_Python から見れば高レベル命令であるが、Jason 側 (エージェント側) にとっては (3.4 節に述べるようにラップはされるものの、ほぼ) 基本行為として働く。

側が受信

という手順によって Jason から制御を行っている (図 6)。

現在のところ、通信路の暗号化や認証の機構は設けていないという問題がある。しかし、通信部分も他とは独立して書かれているので、他と独立に改善することは可能である。

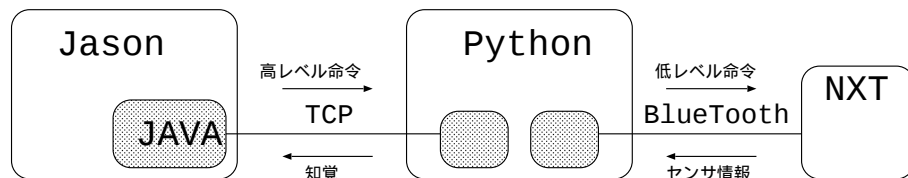


図 6 NXT とエージェントの通信構造

3.4 BDI エージェントの実装

NXT を上位レベルで制御する BDI エージェントは、2.3 節で述べた Jason で実装した。Java 側では、1 節に述べたようなグリッドワールドでの行動制御をにらみ、1 マス前進や 90 度回転、前方に障害物があるかどうかの知覚 (超音波センサによる) など、グリッドワールドでの行動であれば普遍的であると思われる行為を、基本行為として用意した。これらは、基本的には 3.2 節に述べた、我々が作成した Python による NXT 制御メソッドをほぼそのまま用い、多少のラップを行う (前進した場合に位置の信念を更新するなど) 程度で実装している。

一方、Prolog 風の (エージェントプログラムの) 方では、4 節に述べるような、実世界で分担して台を探して Lift が持っているオブジェクトをそこに載せるという問題をにらみ、これに必要なプラン (ただし、4 節の問題に限らずある程度汎用的と考えられるもの) を、上述の基本行為を用いて実装している。回転角でなく向きたい方向を指定して回転、特定の目的地までの経路の決定や移動、また Explorer については、特定の場所にある物体の高さの判定、などである。

3.4.1 シミュレータによるデバッグ

Python 側の実装を、「実際には NXT と通信を行わず、NXT を動作させたふりをしてダミーの実行結果を返す」ものに置き換え、Jason 側をそのまま動作させれば、実際に NXT を動かすことなく、2.3 節で述べたシミュレータの中でのみエージェントを動作させられる。これは、エージェント側のプランの実装の検討やデバッグに効果的であった。

4. 実 験

本節では BDI エージェントを搭載した NXT によるロボットを実際に制御するという実験について記す。

実験は、 4×5 マスのグリッドワールドを実世界上に設置し (図 7・8 は配置の初期状態の例。図の左上のマスの座標が (0, 0)、右下が (4, 3))、2 体の異なる特徴を持つロボットをその上に配置し、2.3 節で述べたシミュレータにおけるグリッドワールド上でのエージェントの動作と対応させて、協調作業を行わせた。

ロボットは 3.1 節に示すように、Explorer と Lift を用意した。また、他に台が配置される (これらは動かない)。台は、荷物を置く台と障害物の 2 種類であり、前者は 1 個だけ、後者は複数個ある。前者の方が低く、Lift は前者にしか荷物を置けない。また、3.1 節に述べたセンサの差異により、Lift はどちらの台も知覚できるが両者の区別はできず、Explorer は両者の区別ができる (前者は、タッチセンサで知覚でき、超音波センサでは知覚できないことで識別可能)。

協調作業の目標は、それぞれのロボットが探索領域を分割し探索を行う中で、荷物を置く台を探し出し、その上に Lift が (初期状態で持っている) 荷物を置くことである。初期状態ではロボットは台がどこにあるかは知らず、台の位置に関する信念は探索に伴って加わっていく。

ここでは、3.4 に述べたプランに加え、特にこの問題に特化した次のようなプランを追加作成し、ロボット間のコミュニケーションを扱うことによって作業を円滑に行えるようにした。

(1) Check 機能

Lift は 3.1 節で述べたように、台の高低差を判別することができないので、Lift が探索を行っていく上で台を知覚した場合は台の種類を特定してもらうために Explorer に台の識別を依頼する。依頼を請け負った Explorer はその依頼を受け、Lift の指示する台まで移動し、判別をする。

台の識別の結果、それが障害物ならば、Explorer はそれを Lift に伝えて自分の探索領域に戻り、探索を続行する。荷物を置く台であれば、Lift に台を発見したことを報告し、その報告を受けた Lift はその台に荷物を置き、作業を終える。

(2) 別のロボットの指示があるまでその場で待つ機能

他のロボットが移動をする際に、2 体同時に移動しようすると衝突が起こる可能性

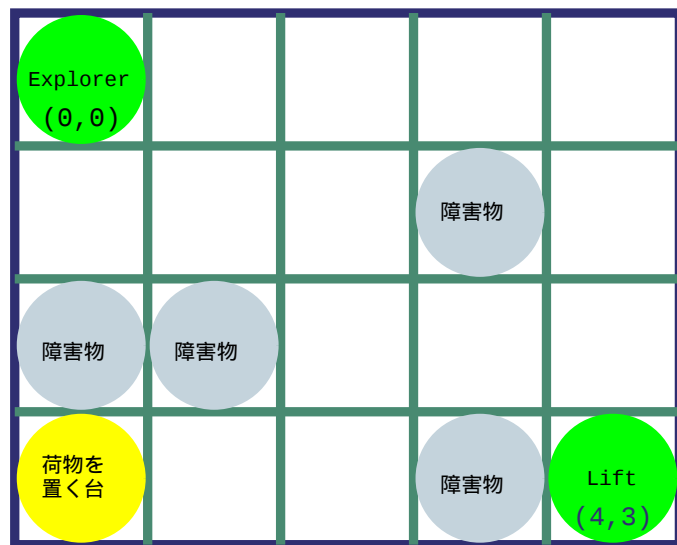


図7 実験におけるグリッドワールドとロボットと台の配置の一例

があり、これを考慮して、1体が移動中である場合、片方を待機させておき、移動完了後に待機解除を命じることにより、衝突を防ぐ。

- (3) 別のロボットの移動経路内にいる場合にその経路に含まれない位置に移動する機能
Explorer が Check の依頼により台の識別に向かう際に、Explorer は最短経路としての自身の移動経路を算出し、それを Lift に伝えることが可能であるが、その移動経路に Lift が存在している場合がある。このようにあるロボットの移動経路内に他のロボットが存在することが確認できる場合には、その移動経路の外に経路内に存在するロボットが移動することによって他のロボットの行動の邪魔にならないようにすることができる。

これらを用いて、Explorer や Lift の動作は以下のように設計した。

- Explorer...初期目標は「台の位置を Lift へ知らせる」で、そのためのプランは「探索を行う」である。このプランの内容は、隣のマスを選ばず、そこに何もなければそこへ進んで再帰的に探索を行う、障害物があればその信念を加えて別なマスを選ぶ、などといったものである。障害物があるという信念を既に得ているマスに対しては、新たな探

索はしない。他に、Lift から台の識別の依頼があったときは、台の識別を行う意図が発生し、そのプランとして上記 (1) が使われる。

- Lift...初期目標は「台に荷物を置く」で、そのためのプランは「探索を行う」(Explorer のものとは別) である。このプランの内容は、Explorer のそれと似た作りだが、隣のマスに何かあれば Explorer に識別を依頼し、その終了まで上記 (2) で待つ。その結果、それが障害物なら信念に加えてから別なマスを選び、荷物を置く台なら荷物を置いて終了する。また、(2) で待っている間に、その位置が Explorer からの移動経路内に入っているならば、Explorer からの通知により (3) が実行される。

このようなプランを使用し、実際に実世界におけるグリッドワールド上でロボットに作業をさせた結果、またその時に生じた問題点とその解決について、以下の節で述べる。

5. 結果

実験をおこなった結果、Explorer と Lift は期待通りに行動し、シミュレーション画面上にもその動作が反映された。ここでは図7の初期状態からの実験結果(図9)について述べる。

まず Lift は自分のとりにある台を Explorer に Check してもらうことを依頼し、Explorer は座標 (1,1) へ移動中にそれを受け取り、移動した地点から経路を算出し、台の Check を行った。この台が障害物であることが Explorer によって判断され、Explorer は Lift にそれを伝えて次の探索地点に戻って探索を再開した。

その後 Lift は新たに台を発見し、それに対して Explorer に Check の依頼をし、Explorer は再び経路の算出をした。このとき、Explorer の移動経路内に Lift が存在することが判明し、Explorer は Lift に経路外に移動することを伝え、Lift は経路外に移動することによって台の識別に向かうことができた。

そして、その台が荷物を置く台であることが判明し、Explorer はそれを伝えて Lift は Explorer が自身の移動経路外まで Explorer が戻るまで待機し、その後台まで移動してその台に荷物を置くことによって Lift が自身の荷物を台に置くことができたので、目標を達成することができた。

6. 考察

このようにロボットはシミュレーション通りの行動をとることができたが、初期の実験では、ロボット間の通信を行う際にタイミング次第で期待通りの行動をとることができない場合があることが判明した。

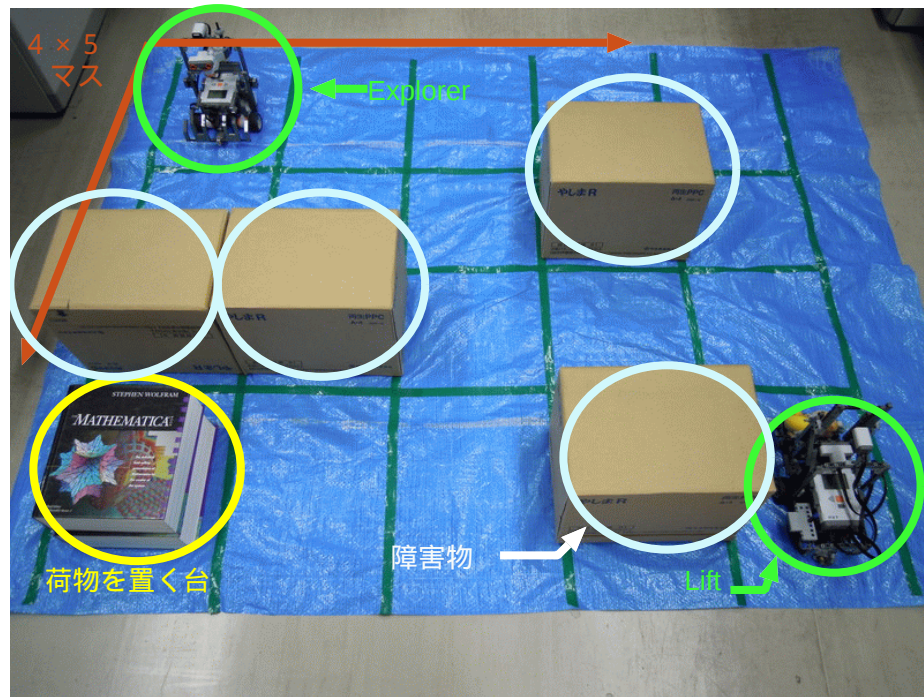


図 8 実験の様子

それは、Explorer の探索中に Lift が Explorer に Check の依頼をするためにメッセージを送るというコミュニケーションを行うところで発生する (図 10)。Explorer は台の識別をするために移動する経路を算出するが、その経路算出をしている間に Explorer が並行して探索を続けてしまい、経路の起点がずれてしまうために算出された経路の移動ができなくなる。

これは、複数の意図の並存に関わる問題である。Explorer は「探索をする」という願望の元に探索を行うための意図を保持しているが、Lift からの Check の依頼によって、新たな信念「台の識別を要する」が Explorer の信念に追加され、その信念によって起動するイベント、「台の識別をする」に対する意図も探索のための意図とは別に生成される。これにより Explorer は、これら 2 つの意図を並列的に持つことになるが、台の識別を終了して台の識別をする意図がなくなるまでは、現在実行すべき意図の熟考による決定 (2.2 節) にあ

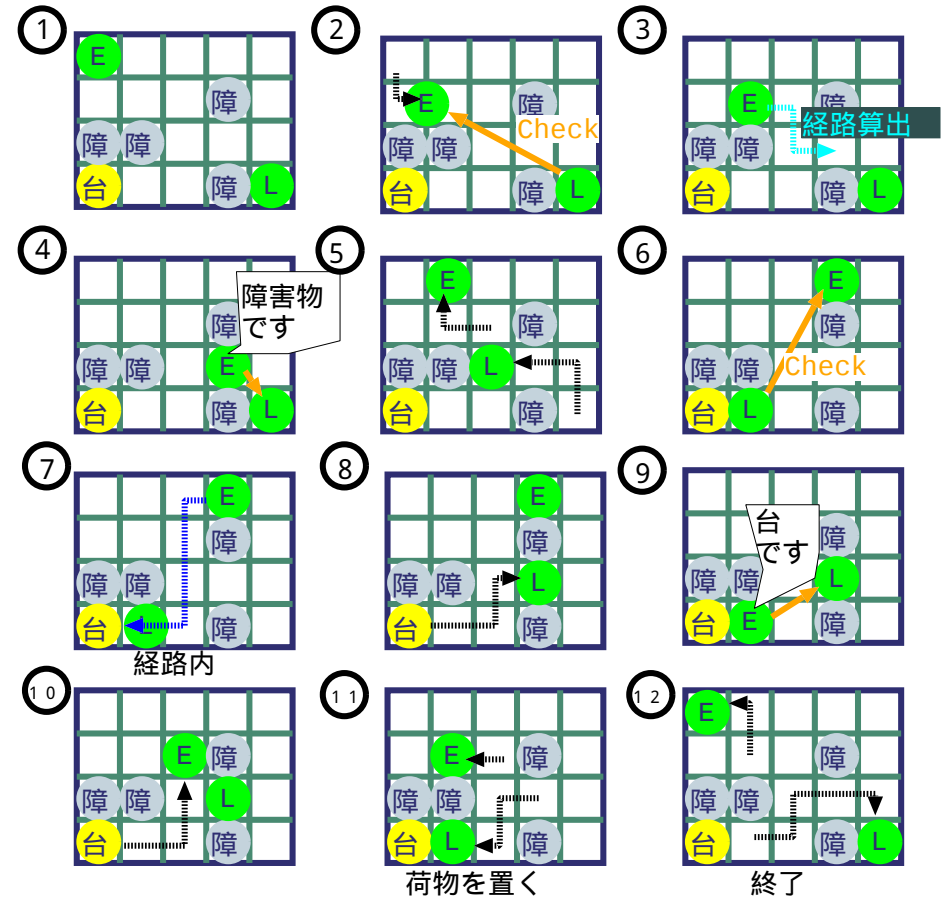


図 9 実験の結果の一例

て、台の識別をする意図の方が優先されるようになっており^{*1}、その結果、探索を行う意図は保留され、Explorer は台の識別のための行動に専念することができる。このように、複数

^{*1} 実装上は、効率上の観点から、毎回熟考で選択するのではなく、探索を行う意図の方を Jason の組み込みアクション .wait で待たせることで同等のことを行っているが、概念的には毎回熟考で選択が行われていると理解しても差し支えない。

の目標に対する具体的な行動の切り替えを、意図の選択という形で明確に扱えることは、意図の概念を明示的に持つ BDI モデルを用いたことによる、設計上の大きな利点である。

しかし、タイミング次第では、Lift から Check 依頼が来て Explorer が台の識別の意図を選択し、そのプランを実行に移して台への経路算出を始める時点が、探索のためのプランを構成する基本行為である隣のマスへの移動の途中であることがあり、この場合、Explorer の現在位置に関する信念は移動前のものである。一方、選択された経路によって実際に台への移動を開始するのは隣のマスに到着してからであるため、起点が不適切になってしまう。

これは、実世界では基本行為の実行に時間がかかること、またそのため、その間に他の要因でのイベントとそれに対する処理が発生してしまうこと (Jason はそうなっている) に起因する。

そこで我々は、これに対する改良として、Explorer の経路算出の開始は隣のマスへの移動終了に伴う Explorer の現在地の信念の更新によってトリガされることとした。これによって上記の問題は解消できた。

しかし、残された問題点として、たまたま隣のマスへの移動終了直後に Check の依頼が来た場合、その次のマスへの移動が始まらないうちに探索の意図が保留されてしまい、次のマスへの移動が起きないため現在地の信念の更新が発生しなくなって、Explorer が立ち往生する問題が発生した (1)。これに対しては、現時点では、いくらかの時間が経てばタイムアウトをする設定にし、その場合にも現在地の取得を行うことで回避している。今回は、一マスの移動に費される時間が概ね 8 秒以内であったことが実験的に得られたため、これを過ぎても現在地の信念の更新が発生しない場合は Explorer が停止しているものと見なし、この 8 秒をタイムアウト時間として設定した。

この部分に関しては、ロボットの動作にかかる時間に依存しており、この実装よりもより効率のよいものがあるかどうかは現在検討中である。

このような信念参照を行うことにより現在地からの経路選択が行えるようになり、実験上でも問題無く動作することが確認できた。しかし、ここで述べたような問題は、現実世界での行動の不安定性を反映したものであり、一般に対処の難しい問題である。例えば今回の実験を、2.3 で述べたシミュレータのみ (ロボットなし) で行った場合、通常の実行モードでは図 10 の状況のみ、ステップ実行では上記 (1) の状況のみが起きた。しかし、現実のロボットを伴う実験ではいずれも起きるため、両方を考慮しないと正しい対処が行えない。

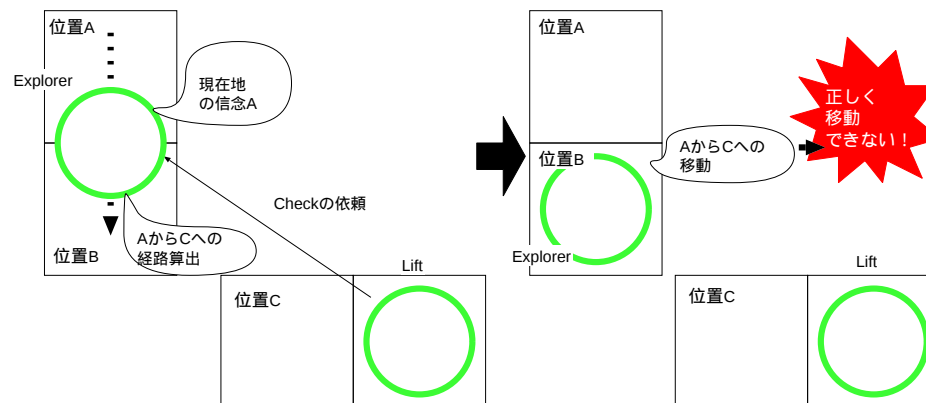


図 10 タイミングによるズレの問題

7. 関連研究

本研究に関連する研究として、文献 6) と文献 4) が挙げられる。どちらもマルチエージェントをロボットに搭載し、実際に実験を行った研究である。

このうち文献 6) は、複数のロボットがグリッドワールド上を探索してオブジェクトを収集するもので、主に、衝突問題をロボット間のコミュニケーションによって回避することや、同じオブジェクトを複数のロボットが拾いにいくことを避けるために、入札システムを介してコミュニケーションをとることを提案している。また、ロボットの移動が格子状に張りめぐらされたラインにそって行動する、ライントレース仕様になっているが、その格子間を複数のロボットが移動する際に、ロボットが移動前の位置と移動後の位置の両方を一時的に占有することからデッドロックの可能性が生じる。これに対して、格子間の間をあけて物理的に行動時間を伸ばすことによる回避や、あるロボットが別のロボットの移動のために停止する待機機能などを実装することによる回避を提案している。

また文献 4) は、複数のロボットがコンベア上の小包を指定位置まで運ぶというものである。ロボットは PDA (Personal Digital Assistant) と LEGO の RCX (NXT の旧バージョン) をつなげて実装している。ロボットの仕様については 2 種類用意しているが、1 つはコンベアベルトに相当するロボットであり、もう 1 つはフォークリフトに相当するものである。コンベアベルトは用意された小包をパレットの上に置くことを目標とし、フォークリフトは指

定された位置に運ぶことを目標としており、これらのタスクは独立して扱うことができる。ロボットの制御は、コミュニケーション部分と実際に行動するための制御部分に分けて実装され、その設計に関しては詳細な議論が展開されている。一方、実験についてはプロトタイプにとどまり、実験結果に関する考察は少ない。

これらと我々の研究の主な差異は、我々のものでは、異なる機能のエージェントがそれぞれの目標の達成のために行動しつつ、必要に応じてコミュニケーションをとって協調を行っている点である。

文献 6) では、コミュニケーションは衝突を避けるために用いられ、複数のロボットが一つのタスクに集中することを避けたりすることに用いられているが、いずれも競合回避を目的としたもので、特に後者では入札システムを介する中央集約的コミュニケーションであり、複数台のロボットが協調して一つのタスクを達成するための相互コミュニケーションはおこなっていない。

また、文献 4) においても、天井からロボットを監視しロボットに推奨される経路を命令するエージェントが存在し、このエージェントがロボットとコミュニケーションをとることにより衝突回避をすることが可能である。他にも、コンベアベルト同士とフォークリフト同士で、どちらのロボットが仕事を代表して行うかということを入札システムで管理している。これらのように、全体を統率する存在のあるシステムでは、1 つの空間内でタスクを円滑に行える利点はあるが、個々のエージェントの自律性は損なわれる。

これに対し、我々の研究では個々のエージェントが自分の行動決定や経路の選択を行い、そのエージェント同士がお互いに必要であるときに、相互にコミュニケーションをとって協調して作業をすることが可能である。そのため、エージェントはより自律的であり、実世界でのロボットの行動制御にとってはより望ましいと考えられる。

また、文献 6) や文献 4) は独自実装の BDI エージェントを用いており、意図の柔軟な変更についてあまり考慮されていないが、我々は BDI エージェントの汎用的プラットフォーム Jason を実現に用いているため、意図の変更などをより柔軟に扱える。

さらに、文献 6) では Windows/MacOS にのみ対応する NXT-G や、Visual Studio などを開発環境としているため動作環境に制約があるが、我々のものは Python と Java さえ動けば OS やプラットフォームを選ばない点も有利である。

8. おわりに

本論文では、BDI モデルによる自律エージェントを用いたロボットの知的制御の実現につ

いて述べた。自律エージェントとロボットの結合、特に BDI モデルの適用によって、問題解決に向けた一貫した行動がとれ、人間に近い行動決定のできるロボットの実現が期待できる。今後の課題としては、ロボットの基本行為の精度に関する問題を解消するため、ロボティクス分野の成果によるロボット制御技術との結合による実験の展開や、また、プランニングなどを必要とするより大きな問題へ本論文の手法を適用し、より実用的な課題への応用を図ることなどが挙げられる。

参 考 文 献

- 1) Bordini, R. H., Hübner, J. F. and Wooldridge, M.: *Programming Multi-Agent Systems in AgentSpeak using Jason*, John Wiley & Sons (2007).
- 2) Bratman, M. E.: *Intention, Plans, and Practical Reason*, Harvard University Press (1987).
- 3) Claudia and Thomas: LEGO Mindstorms NXT: Welcome to the NeXT generation, http://www.tik.ee.ethz.ch/tik/education/lectures/PPS/mindstorms/sa_nxt/index.php?page=print (2006).
- 4) Jensen, L. K., Kristensen, B. B. and Demazeau, Y.: FLIP: Prototyping multi-robot systems, *Robotice and Autonomous Systems*, Vol.53, No.3-4, pp.230-243 (2005).
- 5) Lau, D. P.: NXT_Python, <http://home.comcast.net/~dplau/nxt-python/index.html> (2007).
- 6) Lemke, A., Laidlaw, J. and Zilmer-Pedersen, L.: Developing Multi-Agent Lego Robotics, <http://homepage.mac.com/piyajk/text/2007/0707.html> (2007).
- 7) Parker, D.: nxtprograms.com, <http://www.nxtprograms.com/index.html>.
- 8) Rao, A. S.: AgentSpeak(L): BDI agents speak out in a logical computable language, *Proc. of MAAMAW-96*, LNAI, Vol.1038, Springer-Verlag, pp.42-55 (1996).
- 9) Rao, A. S. and Georgeff, M. P.: Modeling Rational Agents within a BDI-Architecture, *Proc. of International Conference on Principles of Knowledge Representation and Reasoning* (1991).
- 10) Singh, M. P., Rao, A. S. and Georgeff, M. P.: Formal method in DAI: Logic-Based Representation and Reasoning, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, The MIT Press (1999).
- 11) 藤田 恵, 小島侑子, 片山寛子, 新出尚之: 実世界での BDI ベースのロボットの柔軟な行動決定の実現に向けて, 合同エージェントワークショップ&シンポジウム (JAWS2009) 講演論文集, pp.354-361 (2009).
- 12) 山川 宏, 高田司郎, 鮫島和行ほか: 特集: 意図研究のスペクトル, 人工知能学会誌 (特集: 意図研究のスペクトル), Vol.20, No.4, pp.357-455 (2005).
- 13) 高田司郎, 川戸慎二郎, 間瀬健二: 利用者と協調して写真を撮る対話エージェント, 人工知能学会全国大会, pp.4-5 (2002). 1A1-04.