BDIモデルに基づいた小型ロボットの開発実験環境の整備について

奈良女子大学理学部情報科学科
新出研究室 神志那未央

概要

近年、単一の目標達成に特化されたロボットではなく、動的に変化する環境のもとで目的を達成するように振る舞うロボットの実現が望まれている。BDI モデルは、行為選択の柔軟性および意図の整合性と再考慮を持ち、複雑で多様性に富んだ実世界における意思決定機構として有効であると考えられる。そのため、BDI モデルに基づくロボットの研究は重要であり、その開発を容易に行えることが望まれる。そこで本研究では開発を容易に行うことができるように開発実験環境の整備を行った。その結果、実際にエージェントの行動改善や衝突回避アルゴリズムの改良などを行った際に見通しよく作業できるという効果があった。

1 はじめに

BDI エージェントとは、信念 (Belief)、願望 (Desire)、意図 (Intention) という 3 つの心的状態パラメータ用いて、熟考しながら自律的かつ合理的な行動を行うエージェントである。BDI モデルは、行為選択の柔軟性および意図の整合性と再考慮を持ち、複雑で多様性に富んだ実世界における意思決定機構として有効であると考えられる。我々は実世界での BDI モデルに基づいた小型ロボットの開発を目標としている。また、高価あるいは特殊なハードウェアによらない知能ロボットの実現を目指すため、ロボットの中でも特に安価で入手しやすいロボットでの開発を進めている。

そこで、我々は BDI エージェントを構築する言語 AgentSpeak 及びその処理系である Jason を用いて、LEGO MINDSTORMS NXT という実世界のロボットを BDI エージェントにより制御することを目指した。

従来の研究 [1] では、迷路を実際に作成し、2 台のロボットを用い、それぞれが異なった目標を達成することを問題として与え、開発にシミュレーションを併用しながら実際のロボットを動かす実験を行った。

この実験には以下のような問題があった。

1. ロボット動作の精度がよくない
2. 開発環境の不便 (シミュレーション環境の機能不足)
3. 実験の設定に BDI モデルの利点を生かせていない部分がある
4. 衝突回避アルゴリズムが不完全

1 については、ロボットはシミュレータ上の挙動どおりに動作したが、周りの環境の影響により行動に誤差が生じた。これによって、実際のロボットとシミュレータ上のロボットとではズレが生じ、手動で実際のロボットを置き直すことでそのズレを修正する必要があった。

2 については、ロボットの作業をコンソール上に出力された文字列からのみで状況把握しなくてはならず、状況把握に時間がかかる上に問題点を発見することが困難であった。

3 については、ロボットのうち 1 台が複数の目標を保持しておらず、BDI モデルの「複数の目標を並列に保持し、その中から行動を選択できる」という利点を実証する実験としては十分でない設定であった。

4 については、待避できない盤面が存在したことや、3 で述べたように実験の設定を変更したことでアルゴリズムを見直す必要が出てきた。

そこで本研究では、これらの問題の改善を目指した。

なお、本研究は奈良女子大学 4 年中川との共同研究である。本論文では以下の点について述べる。

- ロボット動作の精度の改善
- 開発環境の改善

それ以外は、[2] で述べられている。

2 本研究における問題設定

問題設定 マス目の中を Explorer と Lift という異なったロボットがそれぞれ目標持って必要に応じて協調動作を行う。詳しくは 2.1 節で述べる。

使用するロボット LEGO MINDSTORMS NXT を使用する。詳しくは 2.2 節で述べる。

2.1 問題設定

任意のマス目空間で 2 台の異なった性能を持つロボット (Explorer、Lift) を用い, それぞれが異なった目標を必要に応じて 2 台で協調して達成する。Explorer の目標は可能な限り詳細なマップを作成することである。Lift の目標は自分の持っている箱 (以下、Box) を台に置くことである。台には 2 種類あり、1 つは Box を置くことができる台 (以下、Stand)、もう一つは Box を置くことができない台 (以下、Obstacle) である。つまり、Lift は Stand に Box を置くことを目指している。ロボットの性能が異なるため、台の識別方法はロボットごとに異なる。それについては 2.2 節で述べる。

迷路上の各地点には、Stand、Obstacle、何もない地点 (以下、Empty) のうちのどれか 1 つが存在する。迷路上に置ける範囲であれば、個数に制限はない。ただし、ロボットと台が同じマス目に存在することはない。

作業開始時点で、ロボットは迷路の状態をまったく知らないものとする。ただし、範囲外への知覚、移動を防ぐために Wall があると認識している。また、自分の現在地と向き、相手の現在地は常に知っているものとする。

[1] との問題設定の違いについては [2] で述べられている。

2.2 使用するロボット

本研究では、LEGO 社が販売している LEGO MINDSTORMS NXT(以下、NXT)[3] というロボットを 2 台使用した。NXT は教育用に開発されたもので、比較的制御しやすく安価なものであることから実験に適しているといえる。NXT の制御は [1] と同じ方法で行った。

2.2.1 Explorer と Lift の性能の違いについて

ロボットが持つ機能は表 1, Stand および Obstacle とセンサの高さ関係は図 1 に示すとおりである。[1] からの変更点としては、Explorer, Lift ともにコンパスセンサの使用を止め、ライトセンサを 2 つ導入した。

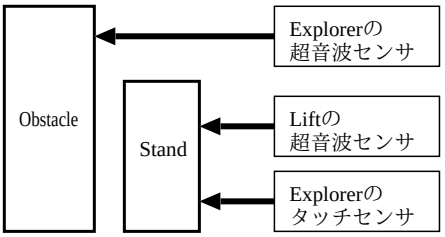


図 1: 台とロボットの高さ関係

表 1: ロボットの性能

	超音波センサ	ライトセンサ	タッチセンサ	フォークリフト	コンパスセンサ
Explorer	○	○	○	×	×
Lift	○	○	×	○	×
目的	障害物を知覚する	ラインとレースを行う	障害物を知覚する	Box を台に乗せる	方向を知覚する
					

2.2.2 Explorer

Explorer は図 2 に示すロボットである。Obstacle の高さは Explorer の超音波センサの位置よりも高いため知覚することができる。それに対して、Stand の高さは Explorer の超音波センサの位置よりも低いため知覚することができない。しかし、下部にタッチセンサがあるので台との衝突によって、Stand を知覚することができる。したがって、Explorer は Stand と Obstacle を知覚することができる。

2.2.3 Lift

Lift は図 3 に示すロボットである。Stand と Obstacle の高さは Lift の超音波センサの位置よりも高いため、Stand か Obstacle のどちらか識別できない台 (以下、UnkObj) は知覚することができる。Lift は自分の目標達成のために UnkObj を Explorer に識別してもらう必要がある。UnkObj が Stand の場合のみ、フォークリフトを用いて Box を台に乗せる。

図 2: Explorer

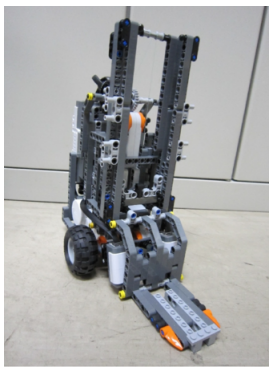


図 3: Lift

3 ロボットの動作環境の整備

3.1 [1] のロボットの動作について

左右のモータの回転数やコンパスセンサを頼りに、マス目に沿って移動を行っていた。

- 直進

  現在向いている方向へ同じ回転数で左右のモータを回転し前進する。

- 1 マス分の移動

  モータの回転数で移動距離を判断し、1 マス分直進する。

- 回転

  コンパスセンサの値を用いて回転する。

3.2 従来研究でのロボットの問題点

従来研究 [1] のロボットの問題点として、周りの環境の影響を受け、正しく動作しない場合があった。
具体的には、モータは床の状態とロボットの重さ、コンパスセンサは回りの磁場の影響を受けた。そのため、モータの回転数に依存している直進と 1 マス分の移動、コンパスセンサに依存している回転は正確に動作しない場合があり、手動で置き直すことでロボットのズレを修正する必要があった。これらの問題を解決するためにラインントレースによる動作を導入した。

3.3 ライントレース

ライントレースとは、地面にあるラインに沿って進む方法のことである。その実現は、ライトセンサを 2 つ導入することで行った。また、迷路を格子状に変更した。ライントレースを行うために 2 台のロボットの幅とタイヤとライトセンサの距離を揃えた。実験方法は任意個のタイルを並べ、ライトセンサが黒のラインを越えたところにロボットを置き、命令通りに動くかを検証した。

- 直進

  ライントレースの動作を導入したことで、ラインに沿って進むため、床の状態やロボットの重さに関わらず、直進することが可能となった。表 2 の様に動作する。

表 2: 直進するときの基本動作

ライトセンサー		基本動作	
左	右	判断内容	動作
白	白	ラインが直線	そのまま前進
白	黒	車体が左にズレた	その場で少し右に回る
黒	白	車体が右にズレた	その場で少し左に回る
黒	黒	1 マス進んだ	一時停止

- 1 マス分の移動

  ２つのライトセンサを用いて、センサが 2 つとも線 (左右のセンサが黒) を認識した場所から次にセンサが 2 つとも線を認識した場所までを 1 マスとし、ラインに沿って直進することで正確に 1 マス分移動できるようになった。

- 回転

  回転するときにライトセンサがラインを何回通ったか (ライトセンサが「白→黒→白」を検知した回数) を認識することで回転することが可能となった。これに伴い、コンパスセンサの使用を中止した。

  また、回転する量を角度で指定していたものを表 3 の様に変更し、動作については左右のライトセンサが両方白を感知するまで前進した後、表 4 の様に動作する。極端に斜めの状態で停止したときに、そのまま左右のライトセンサが両方白を感知するまで前進すると斜めに進みすぎるため、適時修正を加え、まっすぐに停止しようとしている。しかし、その精度はまだ完全ではない。

表 3: 向きの定義

従来の現在地からの向き	新しい現在地からの向き
0 °	foward
90 °	right
180 °	behind
270 °	left

表 4: 回転するときの基本動作

向き	動作
foward	何もしない
right	右のライトセンサが黒を感知し、次に白を感知するまで右に回る
behind	right の動作を 2 回する
left	左のライトセンサが黒を感知し、次に白を感知するまで左に回る

3.4 従来研究との比較

従来研究 [1] では、コンパスセンサで向きを確認し、超音波センサで前方を確認していた。しかし、人は移動するとき、まず現在地を確認し、方位磁針と地図から向きを確認し、前を見て何もないかを確認し、下を見て道を確認してから前へ進む。今回使用するセンサでは、ライトセンサは下を見て道を確認、超音波センサは前を見て何もないかを確認、コンパスセンサは方位磁針で向きを確認することに該当する。[1] では、下を見て道を確認する作業がない。そこで、今回、ライントレースという形でこれを実現した。ただ、今回はコンパスセンサが正しく動く実験場所の確保ができなかったため、コンパスセンサは使用していない。また、現在地を取得する方法がなかったため、向きと現在地は Java を用いて実装している。

[1] ではラインをマーキングとみなし、現実世界にそのようなマーキングがある保証はないためラインに沿って移動することは適切でないと考えていた。しかし、今回はラインを通路とみなすこととした。人は移動する時に通路に沿って移動するため、ラインに沿って移動することは適切であると考える。通路に沿って移動しない場合は、通路がない場所つまり山などの開拓されていない場所になるので、本研究の範囲ではない。

4 シミュレーション環境の整備

シミュレーション環境の整備について述べる前に今回の作業全体の流れと探索方法について説明する。

4.1 作業全体の流れ

作業全体の流れを図に示す。ここでは、主に search、identify、move_to_satnd、retract、search_for_explorer という 5 つの作業が行われる。

Explorer は詳細な地図を作成する (以下、make_map) を実行し、Lift は自分の持っている Box を Stand に置く (以下、carry_to_stand) を実行する。まず、2 台ともに迷路の状態を知るための探索 (以下、search) を行う。

Explorer が台を発見した場合、Obstacle か Stand の識別できるため、Explorer 自身で台の識別を行いながら、search を実行する。しかし、Lift が台を発見した場合、Obstacle か Stand の識別ができないため、UnkObj の識別を Explorer に依頼し、UnkObj の識別をしてもらう (以下、identify)。依頼を出したら、その場に停止することなく、search を再開する。識別を依頼した UnkObj が Stand であった場合のみ、search を停止し、Stand へ行き、箱を置く作業 (以下、move_to_satnd) を実行する。箱を置き終わったならば、開始地点まで戻り、carry_to_stand を終了する。

Explorer は自分の search が終了し、Lift の carry_to_stand が終了したら、Lift にもう一度 search を実行すること (以下、search_for_explorer) を依頼する。search_for_explorer の実行が完了した時点で make_map は終了する。2 台とも開始位置に戻ったら作業終了である。

この流れの途中でロボット同士が衝突しそうになった時に衝突回避するための待避 (以下、retract) を実行する。

4.1.1 make_map

Explorer が実行する。

Explorer が目標を達成するためのプランである。迷路の地図を可能な限り詳細に作成することを目指している。以下の順番に実行される。search_for_explorer が終了すると make_map は終了する。

1. search
2. search_for_explorer

4.1.2 carry_to_stand

Lift が実行する。

Lift が目標を達成するためのプランである。Box を Stand に運ぶことを目指している。Stand を見つけるために search を実行する。search または move_to_satnd が終了すると carry_to_stand は終了する。

4.1.3 search

Explorer, Lift ともに実行する。

迷路の状態を知るための通常の探索のことである。探索方法については 4.2 節で述べる通りである。迷路上に has_looked がなくなると開始位置に戻り、search を終了する。has_looked については表 5 を参照。

4.1.4 identify

Lift が Explorer に台の識別を依頼する。

Lift が search 中に台を発見した場合に実行される。Lift は Obstacle か Stand かの識別ができないため、UnkObj の識別を Explorer に依頼し、UnkObj の識別をしてもらうプランである。依頼を受けると Explorer は自分の地図を確認し、地図に情報がなければ、台の識別へ向かい、識別の結果を地図に追加する。Explorer は依頼の結果を Stand, Obstacle, たどり着けないの 3 つのうちのどれか 1 つで Lift に教える。UnkObj が Stand であった場合のみ move_to_satnd を実行される。。

4.1.5 move_to_satnd

Lift が実行する。

Box を Stand に運ぶプランである。identify を依頼した UnkObj が Stand であった場合のみ実行される。move_to_satnd が始まると今までのプランを捨て (retract 中は retract が終わってから捨てる)、Stand へ向かう。Stand に Box を置き終わったら、開始位置に戻り、carry_to_stand を終了する。

4.1.6 search_for_explorer

Explorer が Lift に実行を依頼する。

Explorer が自分の目標 (詳細な地図の作成) を達成するために Lift に探索を依頼するプランである。この時、Explorer と Lift の迷路の情報は共有され、Lift は新しい迷路の情報を元に search を実行し、新しい情報を得る度に Explorer に知らせる。search_for_explorer が終わると make_map が終了する。

4.1.7 retract

Explorer, Lift ともに必要に応じて実行する。
ロボット同士が衝突しそうになった時に衝突回避するためのプランである。詳しくは [2] で述べられている。

4.2 基本的な探索方法

4.2.1 探索方法の概要

探索は知覚と移動を繰り返す。知覚、移動を行う度に結果を信念や迷路の情報として追加し、迷路を探索する過程で迷路の地図が完成していく。探索は以下を順番に行う。

1. 知覚
2. 移動
3. 1 へ

4.2.2 知覚

ロボットは迷路内を上下左右に 1 度に 1 方向知覚可能である。知覚によって得られる情報は、台の有無と Explorer のみ台の種類である。

1. 現在地から隣り合う 4 方向を既に知覚しているか (表 5 の信念の有無) を確認 (以下、look_around)

表 5: 探索で重要な信念

信念	意味
has_looked	既に知覚した
has_moved	既に移動した
Stand	Stand(台)
Obstacle	Obstacle(台)
UnkObj	Lift が知覚し、Explorer が識別していない台
Wall	範囲外

2. 知覚していない場合、その方向を知覚 (以下、percept_next)
   (a) その方向へ回転
   (b) その方向を超音波センサーで知覚
   (c) 知覚の結果を信念や迷路の情報として追加する
   - percept_next は何もない場合は真、台がある場合は偽となる
   - 表 6 の様に情報を追加する

表 6: ロボット別の知覚の結果

	真	偽
Explorer	has_looked	Obstacle
Lift	has_looked	UnkObj

4.2.3 移動

ロボットは迷路内を上下左右に 1 度に 1 マス移動可能である。以下を順番に実行する。

1. 目的別に現在地からルート検索 (以下、find_route)
   - 詳しくは 4.2.4 節で述べる。
2. ルートに沿って移動できるか確認 (以下、move_along_route)
   - 移動できるならば 3 へ
   - 移動できないならば 1 へ
3. そのルートに沿って 1 歩ずつ移動する (以下、move_to_next)。
   - その方向へ回転
   - 1 歩前進
   - move_to_next が終わったならば表 7 を行う

表 7: 移動に伴う信念の更新

	move_to_next の結果	現在地	追加	削除	次に行う動作
両方	真	has_looked	has_moved, Empty	has_looked	4 へ
両方	真	has_moved	-	-	4 へ
Explorer	偽	has_looked	Stand	has_looked	1 へ

4. 知覚する
5. 2 へ

4.2.4 find_route

現在地から目的地までの最短ルートを検索する。探索アルゴリズムは幅優先探索を使用している。現時点では、極端に大きい迷路は想定していないのに加え、実装が容易だったので幅優先探索を採用した。目的地の種類は表 8 の通りである。表 8 の使用する主な作業については 4.1 節を参照。目的地別にルート検索することで目的地の台の有無に関わらず検索することが可能となった。

表 8: find_route について

名前	使用する主な作業	目的地
has_looked	search	has_looked
identify	identify と move_to_stand	台 (UnkObj, Stand, Obstacle)
certain	元の場所に戻るとき	台が知覚されていない特定の位置
retract	retract	衝突回避のために相手のルート外のどこか

4.3 シミュレーション環境の改良について

[1] では、シミュレータ上でロボットの向きが分からず、またロボットの信念がコンソール上に文字のみで表示され、正確に状況を把握しにくいという問題があった。これらを解決するために、4.1 節や 4.2 節の作業における重要な信念やロボットの向きを画面上に可視化し、見やすくした。

これにより、ロボットがどこを向き、信念をどのように更新し、何を行おうとしているかの把握が容易になった。図 4 に前回の画面, 図 5 に前回のコンソール、図 6 に今回の画面の例を示す。

今回使用したファイルは表 9 の通りである。実験方法としては、PythonRobots.py を実行してから、Jason 側でエージェントの実行を開始させた。ロボットを動かさない場合には、PythonRobots.py で実世界での linetrace.py の代わりに PythonEnv.py を import する。PythonEnv.py 内のシミュレーション上の迷路を設定する配列 _map と Coop_RobotsModel.java の迷路の高さを示す変数 Height, 迷路の幅を示す変数 Width を変更することで迷路のサイズを変更できる。

表 9: 今回使用したファイルについて

ファイル名	ファイルの内容
PythonRobots.py	Coop_RobotsModel.java と、linetrace.py または PythonEnv.py を仲介する役割
linetrace.py	Jason からの基本行為を Bluetooth を用いて NXT に伝えて動作させ NXT のセンサ入力や動作の結果を Bluetooth を用いて受け取り、java ファイルを経由し、Jason に返す役割
PythonEnv.py	NXT の動作や迷路の環境を仮想的に実現しているプログラム
Coop_RobotsModel.java	ロボットの動作の詳細を記述したプログラム。ロボットの初期位置や角度、名前の設定や変更をこのプログラム内で行う。
Coop_RobotsView.java	シミュレーション画面の設定が記述されているプログラム
Coop_RobotsEnv.java	プログラムの環境設定を指定しているプログラム。Jason からの命令は、まずここで処理される。Jason と Coop_RobotsModel.java との仲介としての役割
asl ファイル	Jason 側でのエージェントの動作が記述されている
GridWorldModel.java	Jason であらかじめ用意されているプログラム。グリッドワールド内での基本的な設定が記述されている
GridWorldView.java	Jason であらかじめ用意されているプログラム。グリッドワールド内での基本的なシミュレーション画面の設定が記述されている

4.3.1 ロボットの向きの可視化

ロボットの向きを把握するために、ロボットを表す形を円から三角形へ表 10 のように変更し、方向が画面上で把握できるようにした。実装方法としては、Coop_RobotsView 内の public void drawAgent(Graphics g, int x, int y, int id) で fillOval を用いていたものを三角形の 3 点を計算して求め、fillPolygon で描くことで行った。三角形の 3 点を回転行列を用いた計算で求めているため、正三角形を二等辺三角形に変えたり、0°, 90°, 180°, 270° 以外に回転させることができる。

4.3.2 信念の可視化

4.1 節や 4.2 節の内、状況把握するために重要な信念を画面で表示できるようにし、表 10 のように可視化した。実装方法としては、前回まで、asl ファイルで管理していた迷路探索で重要な信念を asl ファイルと java ファイルの両方で管理するようにすることで実現した。具体的には、asl ファイルで信念を更新するときに時に Jason の信念追加オペレータ+を直接用いるのではなく信念追加を行うためのゴールとして新設した!add で行うようにした。そうすることで、Coop_RobotsEnv.java, Coop_RobotsModel.java を経由することができ、Coop_RobotsView.java を用いた可

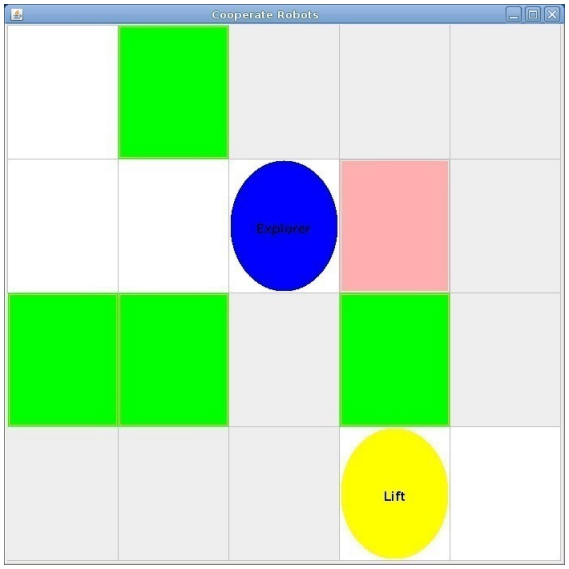


図 4: 前回の画面

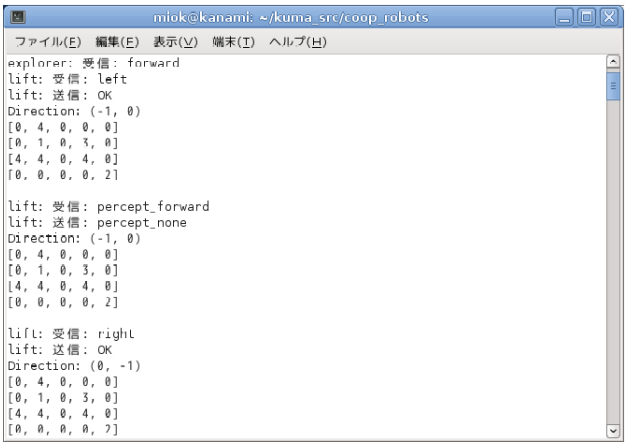


図 5: 前回のコンソール

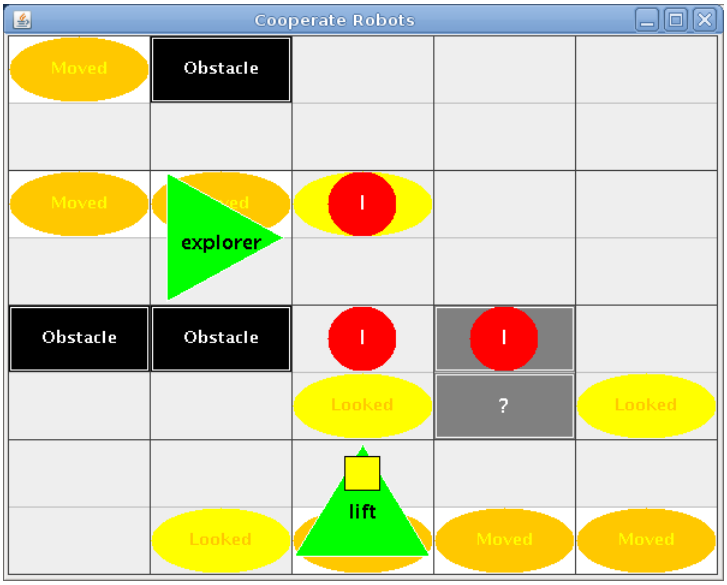


図 6: 今回の画面

視化が可能となった。そのために、Coop_RobotsEnv.java に信念を追加する addbelief、信念を削除する removebelief を、Coop_RobotsModel.java に信念を追加する addBelief、信念を削除する removeBelief を追加した。!add で asl 側に+を用いて信念を追加するのと同時に addbelief を行っている。remove も同様の処理を行っている。

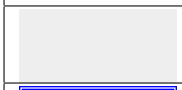
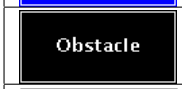
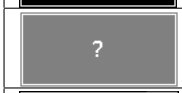
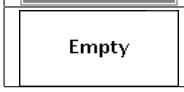
4.3.3 マスの変更

マスを表 12 のように上下 2 段に変更した。

ロボット別の可視化 ひとつのウィンドウで状況を把握できるように、マスを使用するロボットの台数で分割した。実装方法としては、Coop_RobotsView 内で図を描き始める位置と描き終わる位置を調整することで行った。

マスを正方形に どの大きさの迷路でも画面が正方形に表示されていたものを 1 つのマスが正方形になる様にし、必要に応じて画面が長方形になるように変更した。この変更に伴い、Jason の配布 tar ball の中の Jason-

表 10: 可視化を行った情報の一覧

図	表示する文字	内容
ロボット		
	explorer	Explorer
	lift	Lift
		Box
重要な信念		
	looked	has_looked
	moved	has_moved
find_route		
	L	has_looked
	I	identify
	C	certain
	R	retract
	R	retract できるルート
迷路の状態		
		未知
	Stand	Stand
	Obstacle	Obstacle
	?	UnkObj
	Empty	Empty

version/src/jason/environment/grid 内の GridWorldModel.java、GridWorldView.java を書き換え新たに RectangleGridWorldModel.java、RectangleGridWorldView.java を作成した。この変更に伴い、import するものを表 11 の様に変更した。import 以外の変更点は以下の通りである。

- GridWorldModel からの変更点 (RectangleGridWorldModel)
  - view の型を GridWorldView から RectangleGridWorldView に
- GridWorldView からの変更点 (RectangleGridWorldView)
  - model の型を GridWorldModel から RectangleGridWorldModel に
  - コンストラクタの引数を変更
  - makeGridCanvas を追加

表 11: import するものの変更点

	import	変更前	変更後	
ファイル名 (.java)	import jason.environment.	*	Area	Location
Coop_RobotsModel	import jason.environment.	*	○	○
Coop_RobotsView	import jason.environment.	*	○	○
Coop_RobotsEnv	import jason.environment.	*	×	○
RectangleGridWorldModel	import jason.environment.	*	○	○
RectangleGridWorldView	import jason.environment.	*	○	○

- initComponent 内の drawArea に代入するのを new GridCanvas → makeGridCanvas に変更
- GridCanvas に myDraw を追加
- GridCanvas の paint の draw を myDraw に変更
- GridCanvas の外側の線の色を代入する変数を追加

表 12: マスの変更点

変更前	変更後
	 上：Explorer 下：Lift

5 まとめ

本論文では、BDI エージェントを搭載したロボットの開発やそれを用いた実験が容易に行えるように開発実験環境の整備を行ったことについて述べた。その結果、ライントレースにしたことでロボット動作の精度が上がり、修正する手間が減った。また、シミュレーション環境も今回に似た問題設定であれば、Java ファイルを少し変更するだけで表示するものを変えることが可能である。今回、ロボットの動作やシミュレーション環境といった開発実験環境を整備したことで、実験やアルゴリズムの改良が容易になった。

本論文の今後の展望としては以下の 4 つがあげられる。

1. ロボットの改良
2. ライントレースの精度の改良
3. 通信環境の改善
4. シミュレーション環境の実世界への拡張

1 については、現在確認されている問題としてはロボットの重さと超音波センサの位置の大きく 2 つの問題がある。まず 1 つ目のロボットの重さの問題についてである。ライントレースを行うためにライトセンサを追加したことで、ロボットが重くなった。ロボットは重いほど、床との摩擦が強くなり、動き出すために大きな力が必要となる。そのため、摩擦に負けることなく動かすには、モータの回転数をあげる必要がある。しかし、モータの回転数をあげると、ライントレースによる制御が難しくなる。また、Explorer と Lift の重さが違うため、モータの回転数をロボットごとに調整する必要がある 2 つ目は、超音波センサの位置についてである。2.2 節で述べたよう

にロボットを変更したため、Lift の構造が大きく変更された。この変更に伴い、超音波センサが前回のものより位置が下になったため、フォークリフトが邪魔で正しく知覚できなくなり、知覚の際にフォークリフトを超音波センサよりも上にあげる必要が出てきた。しかし、1 つ目の問題に関係するが、Lift は前方にライトセンサをつけたことで、前方にライトセンサとフォークリフトがあり、前方に傾きやすくなっている。そこに Box として重いものを与え、上に持ち上げると前方に傾き、ライトセンサが床に擦る恐れがある。また、Explorer と Lift の超音波センサから台までの距離が違うため、値をロボットごとに調整する必要がある。よって、ロボットの軽量化とセンサの位置を調整する必要がある。

2 については、ライントレースが回転の動作で正しく動作しない場合があることが確認された。交差点に極端に斜めに入った (交差点に入る直前に修正がかかった) 場合、その逆の方向に回転した時に交差点の中心部を回転し、線を識別できず、正しく回転できない場合があった。この問題を解決しようと試みたが、3 で述べる通信の問題により、この問題を再現する前に通信が切れてしまうため、問題の解決には至っていない。

3 については、ある程度の時間、ロボットを動かすと通信が切れることが確認された。しかし、その原因についてはまだわかっていない。

4 については、このシミュレーション環境はグリッドワールドのみに対応しており、実際のロボットがいる連続な実世界には対応していない。連続な実世界へ応用できるようにさらに環境を改善したい。

今後は、これらの問題について取り組むことが課題である。

6 謝辞

本研究を遂行するにあたり、熱心にご指導下さった指導教官の新出尚之准教授に深く感謝し、厚く御礼申し上げます。また、論文作成にあたり、新出研究室の皆様に感謝の意を表します。ありがとうございました。

参考文献

[1] 藤田恵, 片山寛子, 新出尚之, 高田司郎. 実世界の多様性に適応した BDI ロボットについて. 情報処理学会論文誌 数理モデル化と応用, Vol. 5, No. 1, pp. 50–64, 2012.

[2] 中川香奈美. BDI モデルに基づいた小型ロボットの協調による問題解決方法の改善について. 奈良女子大学理学部情報科学科 2012 年度卒業論文, 2013.

[3] 教育用レゴマインドストーム NXT 紹介. http://www.afrel.co.jp/mindstorms/nxt