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
新出研究室 中川香奈美

平成25年2月13日

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る。

先行研究では2台のロボットにBDIエージェントを搭載し実験を行っていたが、作業の並列性と衝突回避のアルゴリズムに不具合が見られた。そこで、本研究ではそれらの点を改善した。その結果、解決可能な問題が増加した。本論文では、それらの手法について述べる。

1 はじめに

BDIエージェントとは、信念(B)、願望(D)、意図(I)という3つの心的状態パラメータを持つことで人間の思考をモデル化し、熟考しながら自律的に行動選択を行うものである。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。そのため、BDIモデルに基づくロボットの開発が望まれる。特に安価で入手しやすいロボットを使っての実験は高価あるいは特殊なハードウェアを使わない知能ロボットの実現に向けて有意義である。

我々は実世界でのBDIロボットの研究を進め、その有用性を示してきた[1]。その際に具体的に取り上げた問題は、BDIモデルに基づいて2台のロボットがそれぞれの目標と意図を持ち、協調して問題解決に当たるというものであった。しかしこの事例では、ロボットが実行する作業が比較的単純であったり、並列性が十分でないなどの問題があり、これらを改善することにより、BDIモデルの優位性をより示せる事例にできると考えられる。また、2台のロボットの衝突回避が不完全で、問題解決としても改善の余地があるものであった。従って、本研究ではそれらの改善を行った。

本研究は奈良女子大学理学部4回生の神志那との共同研究である。

2 問題設定

我々が本研究で設定した例題は2台のロボットが迷路内を知覚、移動することでそれぞれの目標を達成するというものである。それぞれのロボットは目標達成のために相手に作業や情報の伝達を必要最低限依頼することができるものとする。

2.1 ロボットの目標

使用するロボットはExplorer、Liftと呼ぶ。それぞれのロボットの目標は以下の通りである。

- Explorer: 可能な限り迷路の状態を把握すること
- Lift: 箱(Box)を特定の台に運搬すること

相手の目標を達成するために相手から依頼されることで発生する目標は以下の通りである。

- Explorer: 台を識別をすること
- Lift: Explorer が到達不可能である地点を探索して Explorer に情報を伝達すること

2.2 環境設定

環境として以下を設定する (図 1)。

- 迷路は升目空間であり、水平方向を X 軸、垂直方向を Y 軸とし、左上の座標を $(0, 0)$ とする。
- $(1, 0)$ の方向を 0°、$(0, 1)$ の方向を 90°、$(-1, 0)$ の方向を 180°、$(0, -1)$ の方向を 270° とする。
- 迷路上の各地点には Box を置ける台がある地点 (Stand)、Box を置けない台がある地点 (Obstacle)、何もない地点 (Empty) の 3 種類があり、ロボットは何もない地点にのみいることができる。
- ロボットは知覚、回転、移動により迷路の状態を把握することができる。
- Explorer は知覚時に Obstacle を、移動時に Stand を認識できる。一方、Lift は知覚時に台の有無を認識できるが、台の種類を識別することはできない。(実世界のロボットの性能に関しては [2] を参照)
- Lift は Box を Stand に置くためのフォークリフトを持つ。
- ロボットは自分の現在の向き、自分や相手の現在地を常に把握している。ただし、これらの情報は回転、移動終了時に更新される。
- ロボットは作業開始時には迷路の状態を全く知らない。ただし、範囲外への知覚、移動を避けるため例外的に範囲外は Wall と認識している。
- 作業開始時に Lift は Box ををフォークリフト上に載せているものとする。

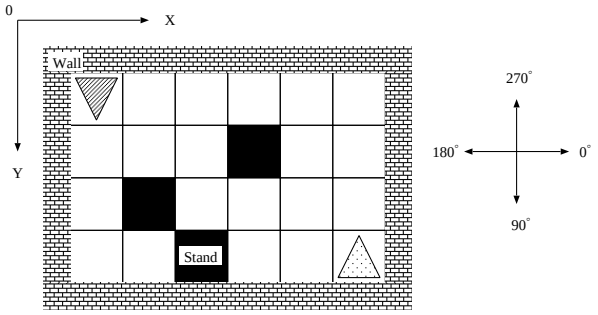

図 1: 迷路の例

2.3 先行研究からの変更点

先行研究からの変更点は以下の通りである。

- ロボットの目的に関して

- Explorer の目的を再定義した
  先行研究での Explorer の目的に盤上を掃除して回ることがあった。しかし、この目的は電池が消耗するまでの間盤上を動き続けるというもので、BDI エージェントの目的としては動作が単純であった。そこで、本研究では Explorer の目的を迷路地図の作成とし、一度探索した地点を再び探索することのないようにした。
- Explorer の目的を廃止した
  別の人間からの依頼により作業を行うとき、相手の目標達成のための目的を依頼される前から持っているのは不自然である。先行研究では Explorer は Stand 発見時にそれを Lift に知らせるという目的を持っていたが、上記の理由により本研究では廃止した。
- Lift の目的を追加した
  先行研究では Lift から Explorer へのみ作業の依頼を行っていた。本研究ではお互いに相手からの依頼に応えることができることを示すべきだと考えた。そこで、Explorer から Lift への依頼として Explorer が到達できない地点の探索を Lift の目的に追加した。

- 環境設定に関して
  - 盤面サイズを任意とした
    先行研究では盤面のサイズを 5×4 に限定していたが、より多くのパターンに対応するためその制約を廃止した。なお、サイズの変更は Java および Python のサイズに関する変数の値を変更することにより行える。
  - Stand の個数を任意とした
    先行研究では Stand の個数が 1 個に限定されていたが、Lift が様々な行動を選択できるようにするため、Stand を複数を置くことができるものとした。

3 作業全体の流れ

Explorer と Lift はそれぞれ初めに迷路の地図を作成すること、Box を Stand に運搬することという自分の目標を持ち、それを達成するためにプランを選択し実行している。相手から依頼があれば、そこに相手の依頼を達成するという目標が新たに生じ、複数の異なった目標を並列に持つ。そして、それらの目標の中から状況に応じて 1 つを優先させ、その目標を達成するためのプランを選択して実行するようになる。その目標が達成されれば、残っている目標の中から同様に 1 つを選び、その目標を達成するための行動を再開する。

3.1 地点に関する信念

ロボットが持つ地点に関する信念について述べる (表 1)。

表 1: ロボットが作業時に持つ信念

信念	意味
empty	何もない地点
stand	Stand がある地点
obstacle	Obstacle がある地点
unkobj	台の種類が分からない地点
has_looked	台を知覚しなかった地点
has_moved	自分が到達した地点

- empty
  その地点に何もないことを表す。ロボットがその地点に到達したとき、または Explorer が Lift から情報を得たときに追加する。

- stand
  その地点に Stand があることを表す。Explorer が移動時に認識したとき、または Lift が Explorer から情報を得たときに追加する。

- obstacle
  その地点に Obstacle があることを表す。Explorer が知覚したとき、または Lift が Explorer から情報を得たときに追加する。

- onkobj
  その地点に台があることは分かるがその種類が分からないことを表す。Lift が知覚したとき、または Explorer が Lift からの識別依頼を受けたときや Explorer が到達できない地点の情報を得たときに追加する。なお、この信念は Explorer が識別に成功し台の情報が得られれば、stand または obstacle に変更される。

- has_looked
  その地点を知覚済みであり、その地点に台を知覚しなかったことを表す。ロボットがその地点を知覚したときに追加され、その地点に到達、またはその地点の情報 (empty,stand) を得たときに削除される。なお、この信念は到達領域と未到達領域の境界に存在し、この信念が存在する地点に到達、知覚することによって探索を進めることができる。

- has_moved
  その地点に到達済みであることを表す。ロボットがその地点に到達したときに追加される。empty とともに追加され、empty がその地点の状態を表すのに対し、has_moved はそのロボットが実際に到達したかどうかを表す。

3.2 作業の流れ

作業全体の流れを図 2 に示す。

自分の目標を達成するために Explorer は地図の作成 (make_map)、Lift は Box の運搬 (carry_to_stand) を行う。相手からの依頼に応えるために Explorer は識別 (identify_for_lift)、情報伝達 (search_for_explorer) を行う。また、Lift は Explorer に依頼した結果を受けとるために識別結果の取得 (result_identify) を行う。なお、識別 (identify_for_lift)、情報伝達 (search_for_explorer)、識別結果の取得 (result_identify) については 4 章で詳しく述べる。

以下は make_map と carry_to_stand の概要と手順である。

- make_map 可能な限り広範囲の詳細な地図を作成する
  自分で到達できる範囲は全て移動し知覚して、迷路上の各地点の情報を集める。自分で到達できない地点は Lift に依頼し、各地点について台の有無を伝達してもらう。

  手順は以下の通りである。

  1. 4 方向を知覚し、近傍 4 地点の情報を追加する (search)。
  2. has_looked があれば、最近傍の has_looked である地点に移動し、1 に戻る。has_looked がなければ、元の地点に戻る。
  3. Lift に自分の持つ地点情報を送り、それ以外の地点情報の伝達 (search_for_explorer) を依頼する。
  4. Lift が探索終了を通知するまで、地点情報を受け取る。

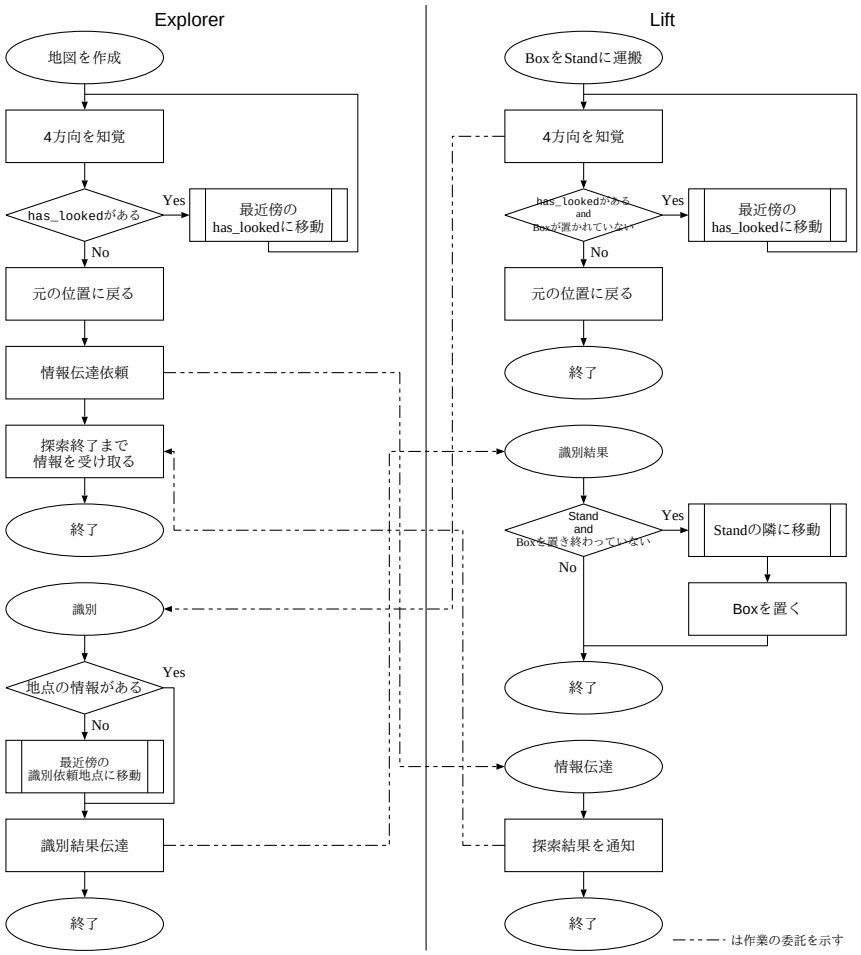

図 2: 作業全体の流れ

- carry_to_stand Box を Stand に運搬する
  探索 (search) を行い、台を発見すれば Explorer に識別依頼をする。Explorer が識別した結果が Stand であれば、その地点に向かい Box を置く。台を発見しなかったり、識別した結果が全て Obstacle だった場合は Box を持ったまま終了する。

  手順は以下の通りである。

  1. 4 方向を知覚し、近傍 4 地点の情報を追加する (search)。近傍 4 地点に台を発見すれば、Explorer に識別 (identify_for_lift) を依頼する。
  2. has_looked があれば、最近傍の has_looked である地点に移動し、1 に戻る。has_looked がなかったり、Box を運搬し終えれば、元の地点に戻る。

協調動作の概要を説明する。

- Lift の識別依頼
  Lift が台を知覚すると、Explorer に識別を依頼し、その結果を受け取るというものである。

  以下に手順を述べる。

  1. 台を知覚する。(Lift)
  2. Explorer に識別を依頼する。(Lift)
  3. その地点について既に情報があればその場で伝達し、なければその地点まで移動し知覚した結果を伝達する。(Explorer)

4. その地点が Stand であれば、その地点に向かい Box を置く。(Lift)

- Explorer の地点情報伝達依頼 Explorer が到達できず Lift が到達できる地点があれば、その地点の情報を伝達してもらうというものである。

  以下に手順を説明する。

  1. 到達不可能地点情報の伝達を依頼する。(Explorer)
  2. 現時点で持っている情報を伝達する。(Lift)
  3. 探索を行い、知覚の度に地点の情報を伝達する。(Lift)
  4. 探索終了を伝える。(Lift)

3.3 知覚と移動

図 2 中の知覚 (look_around) と移動 (move_along_route) は以下の手順で行われる。

- look_around ロボットが今向いている向きに対しての前後左右を知覚する (図 3)。

  以下の作業を 4 方向について行う。

  1. その地点の情報が既にあれば、終了する。まだその地点の情報がないなら、2 に進む。
  2. その地点に相手がいれば、その地点の情報を has_looked とする。相手がいないとき、3 に進む。
  3. その方向を向き知覚を行う。台を知覚しなかった場合 (以下、知覚に成功する)、has_looked とする。台を知覚した場合 (以下、知覚に失敗する)、Explorer は Obstacle とする。Lift は UnkObj とし Explorer に識別を依頼する。

- move_along_route 目的地、または目的地に台があるときはその隣までの移動を行う。また、その道中で随時知覚を行う (図 4)。

  手順は以下の通りである。

  1. 現在地が目的地であれば、移動成功とし、処理を終了する。そうでなければ、2 に進む。
  2. 次に進む地点に相手がいれば、衝突回避を行う。そうでなければ、1 歩前進する。
  3. 前進できれば、4 方向を知覚し 1 に戻る。前進できなければ、4 に進む。
  4. Explorer で先の地点に has_looked があれば、その地点に Stand を追加し、目的地までの経路を再検索して 1 に戻る。先の地点に台があれば、目的地までの経路を再検索して 1 に戻る。先の地点に相手がいれば、1 に戻る。

  衝突回避については 5 章で詳しく述べる。

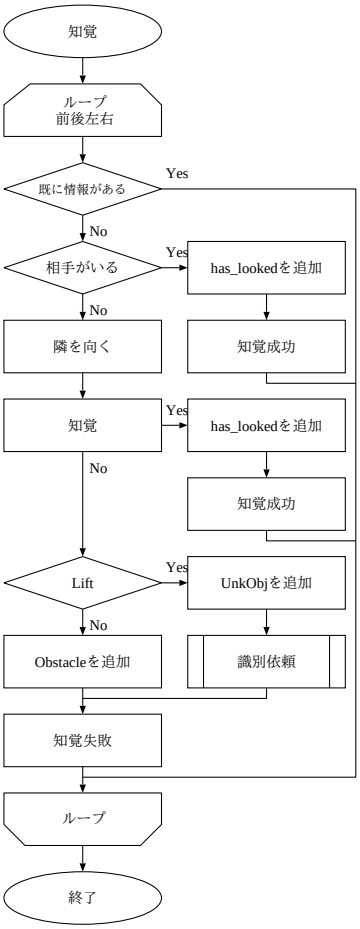

図 3: 知覚の流れ

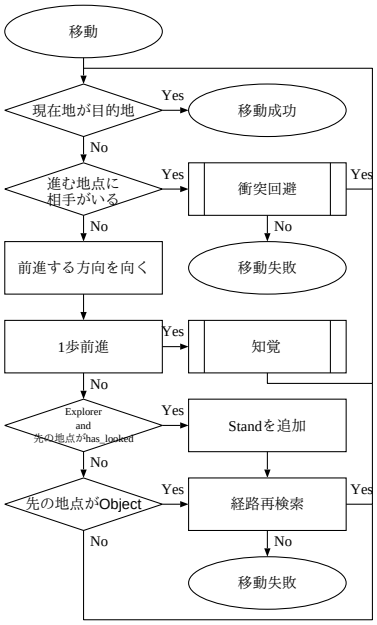

図 4: 移動の流れ

4 協調作業

4.1 協調作業の流れ

- identify_for_lift 台を識別する (図 5)
  Lift から識別を依頼されると、自分の探索を中断し Lift に依頼された地点の識別を行う。既に識別を行っている場合は今向かっている地点と新しく依頼された地点の近いほうから識別する。識別する地点がなくなれば、その地点から自分の探索を再開する。

  手順は以下の通りである。

  1. その地点の情報があれば、それを伝達して終了する。その地点に has_looked があれば、その地点を Stand だと認識し、それを伝達して終了する。まだ探索していなければ 2 に進む。
  2. 現在行っている作業を中断する。
  3. 識別を依頼された地点があれば、4 に進む。識別を依頼された地点がなければ、終了して中断していた作業を再開する。
  4. 現在地に最も近い識別依頼地点に移動する。
  5. 4 で目指した地点の隣まで移動できていればその地点の情報を、移動できていなければ識別できなかったことを Lift に伝達し (result_identify)、3 に戻る。

- result_identify 識別結果を受け取る (図 6)
  Explorer から識別結果を受け取り、その地点が Stand でまだ Box を置きに行っていなければその地点の隣まで移動し、Box を置く。

  手順は以下の通りである。

  1. Explorer から結果を受け取った地点が Stand でありまだ Box を置きに行っていなければ、2 に進む。そうでなければ、終了する。
  2. 現在行っている作業を中断する。
  3. Box を置く。
  4. 中断していた作業を再開する。

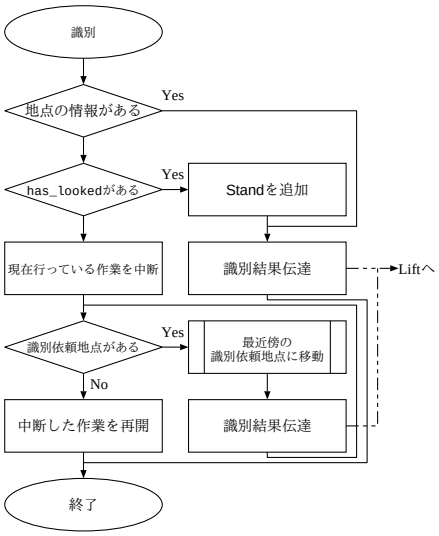

図 5: 識別の流れ

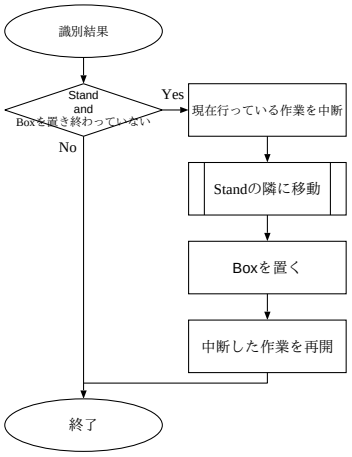

図 6: 識別結果受け取りの流れ

- search_for_explorer Explorer が到達できない地点の探索を行う (図 7)
  自分の目標が終了したら、Explorer が到達できない地点で自分が既に探索した地点の情報を Explorer に伝達し、まだ到達していない地点があれば探索し、各地点の情報が追加され次第 Explorer に伝える。

  手順は以下の通りである。

  1. carry_to_stand の終了を待つ。
  2. Explorer から伝えられた地点の情報を追加する。
  3. Explorer から伝えられた地点以外で現在持っている地点の情報を Explorer に伝達する。
  4. まだ has_looked が残っていれば、最近傍の has_looked である地点に移動、4 方向を知覚し、近傍 4 地点の情報を追加し、5 に進む。has_looked がなければ、Explorer に探索終了を伝え、元の地点に戻る。
  5. Explorer にその地点の台の有無を伝え、4 に戻る。

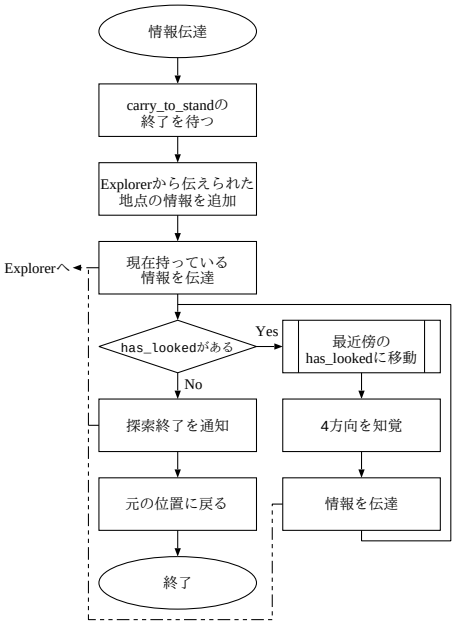

図 7: 情報伝達の流れ

4.2 協調作業の戦略

Explorer は Lift の依頼にすぐに応えるべきであるが、Lift は Explorer の依頼に自分の目標が達成されてから応えるべきである。

その理由は二つある。第一の理由として、Explorer の目標を達成するには可能な限り広く盤面全体を把握しなければならないが、Lift の目標は必ずしも盤面全体を把握する必要はないからである。第二の理由として、Explorer にとって Lift の依頼に応えることは自分の目標達成に役立つが、Lift にとっては役立たないからである。なぜなら、Explorer が移動できる範囲は台を識別できる Explorer が探索するほうが効率が良いし、Explorer の移動できない範囲は Lift も台が何か知ることができないためである。

5 衝突回避

ここでは、移動中に発生する次の進路に相手がいた場合の衝突回避アルゴリズムについて述べる。全体の流れを図 8 に示す。図 8 中の迂回、待避の流れについてはそれぞれ 5.1、5.2 節で述べる。

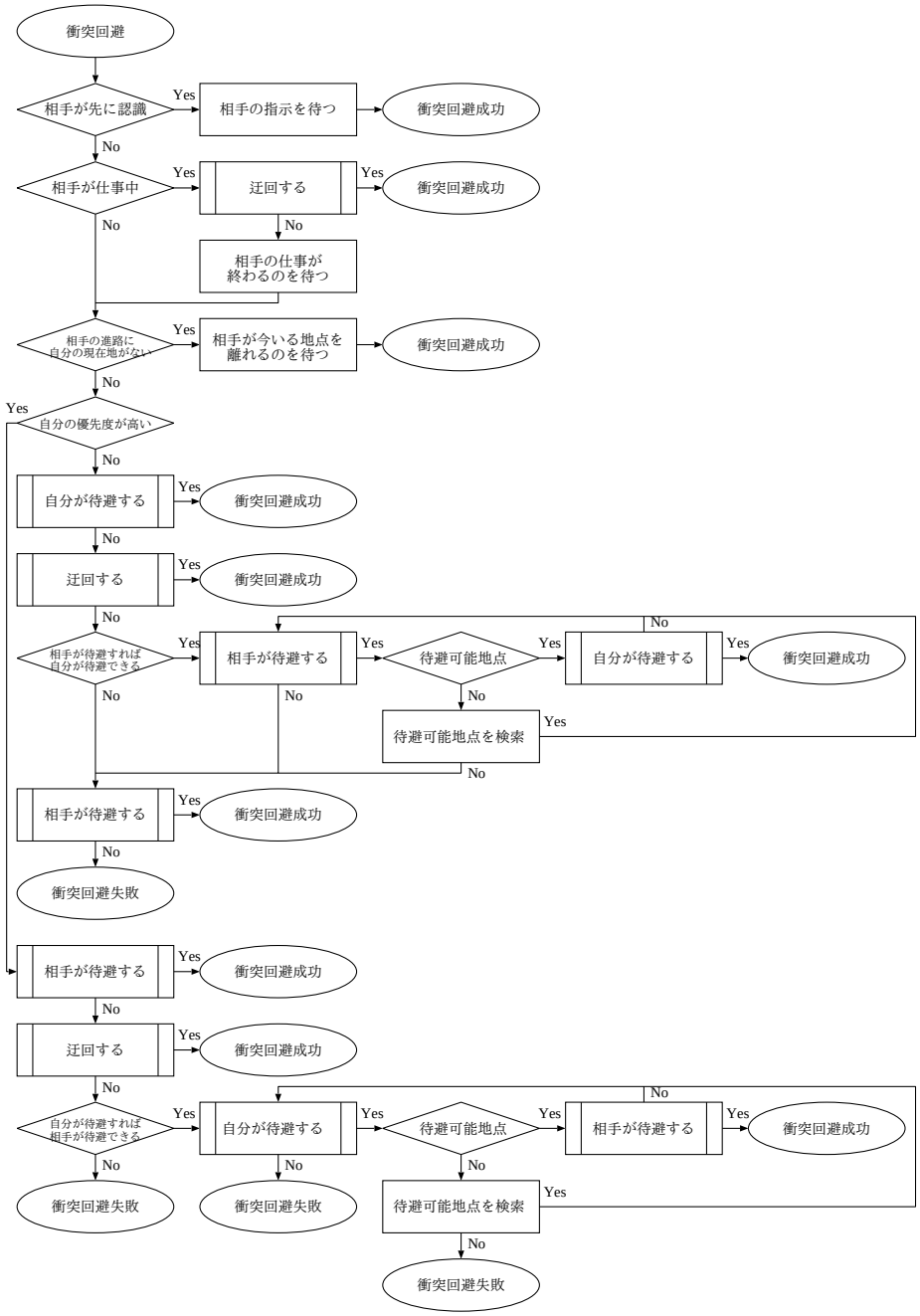

図 8: 衝突回避の流れ

以下に手順示す。

1. 信念 recognized の有無により相手が先に自分を認識しているかどうかを確認する。図 9 のように相手が先に自分を認識していれば、相手からの指示を待ち終了する。相手が自分を認識していなければ、相手に recognized を送り、先に自分が認識したことを伝え、2 に進む。

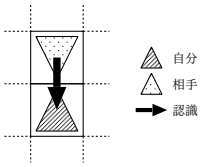

図 9: 先に相手が自分を認識している例

2. 相手に信念 working があるかを尋ねる。(信念 working はその場を動くことができない作業を行うときに追加されるものである。今回は Lift が lift_up の際に追加する。) 信念 working がある場合、相手が動けるようになるまでに時間がかかる恐れがあるため、迂回することを試みる。図 10 のように迂回に成功すれば、衝突回避は終了である。図 11 のように迂回できなければ、相手が動けるようになるのを待ち、3 に進む。

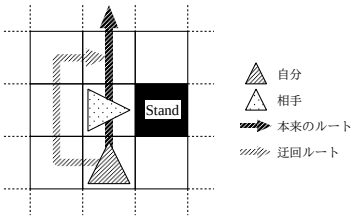

図 10: 相手がその場を動けないが、迂回できる例

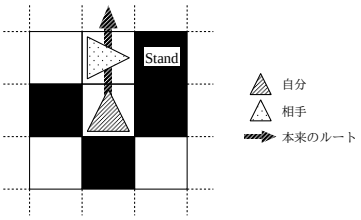

図 11: 相手がその場を動けず、迂回もできない例

3. 改めて相手の状況を確認する。相手が動けないことが分かれば、1 に戻る。そうでないなら、相手のルートを確認する。図 12 のような場合、相手が 2 歩進めば衝突は起こらないため、相手が自分のルートから出るまで待ち、衝突回避を終了する。一方、図 13 や図 14 の場合は少なくともどちらかが待避しなければ衝突を避けることができないため、4 に進む。

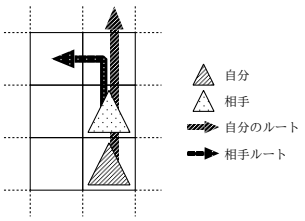

図 12: 自分と相手のルートが一部のみ交わっている例

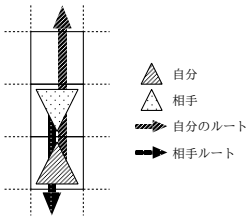

図 13: 相手と鉢合わせしている例

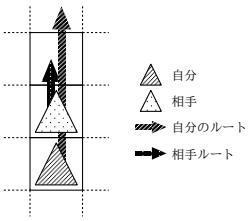

図 14: 相手のルートが全て自分のルート内にある例

4. 優先度を表 2 のように定め、自分と相手の優先度を比較する。(優先度について詳しくは 5.3 節で述べる。)

表 2: エージェントの状態とそのときの優先度

優先度	状態
1	Lift が Stand に向かっている
2	Explorer が識別に向かっている
3	Explorer が探索を行っている
4	Lift が探索を行っている
5	作業が終了し Explorer が元の位置に向かっている
6	作業が終了し Lift が元の位置に向かっている
7	目標を達成し元の位置で停止している

5. 優先度の低いほうが待避を試みる。図 15 のように待避に成功すれば、衝突回避は終了である。待避に失敗した場合、6 に進む。

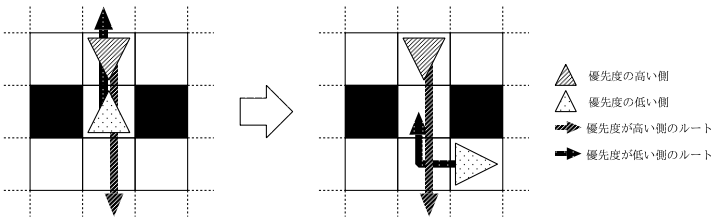

図 15: 優先度の低い側が待避する例

6. 指示しているほうが迂回を試みる。迂回に成功すれば、衝突回避は終了である。失敗した場合、7 に進む。(7 の前に迂回を行うのは図 16 のような例に対応するためである。この場合、どちらも待避することはできないが迂回することによって目的地に到達できる。)

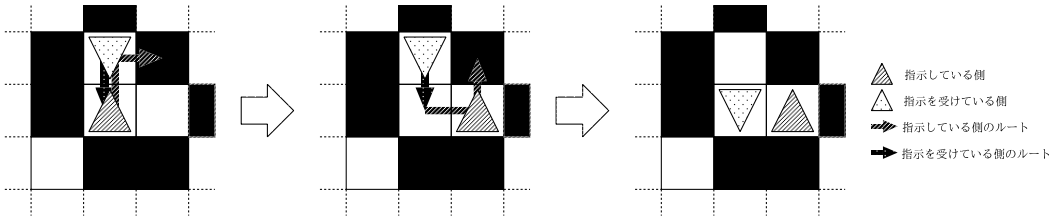

図 16: どちらも待避することはできないが、指示している側が迂回できる例

7. 優先度の高いほうが一度待避して優先度の低いほうを待避させる。図 17 のように成功すれば、衝突回避は終了である。失敗すれば、8 に進む。

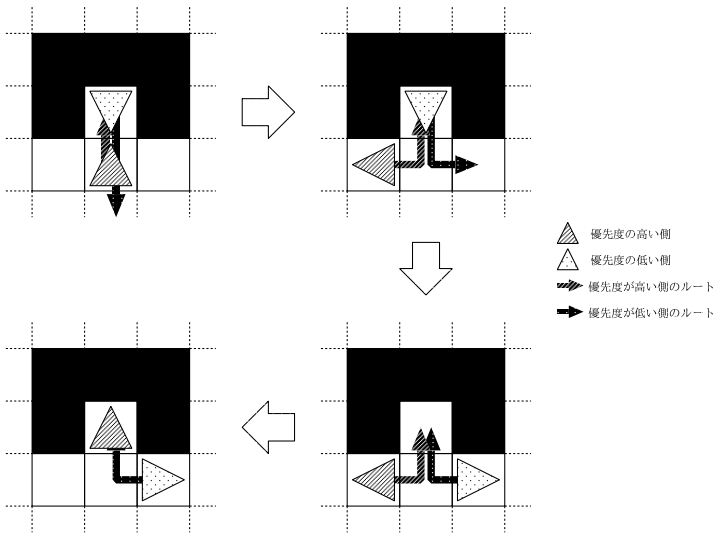

図 17: 優先度の高い側が一度待避して優先度の低い側を待避させる例

8. 優先度の高いほうは待避地点を再検索する (図 18)。再検索に成功すれば、7 に戻る。失敗すれば、9 に進む。(このような処理をするのは、図 18 のように少なくともどちらか一方が持つ地点の情報が不完全なため、優先度の高いほうが待避できても低いほうが待避できない場合があるからである。)

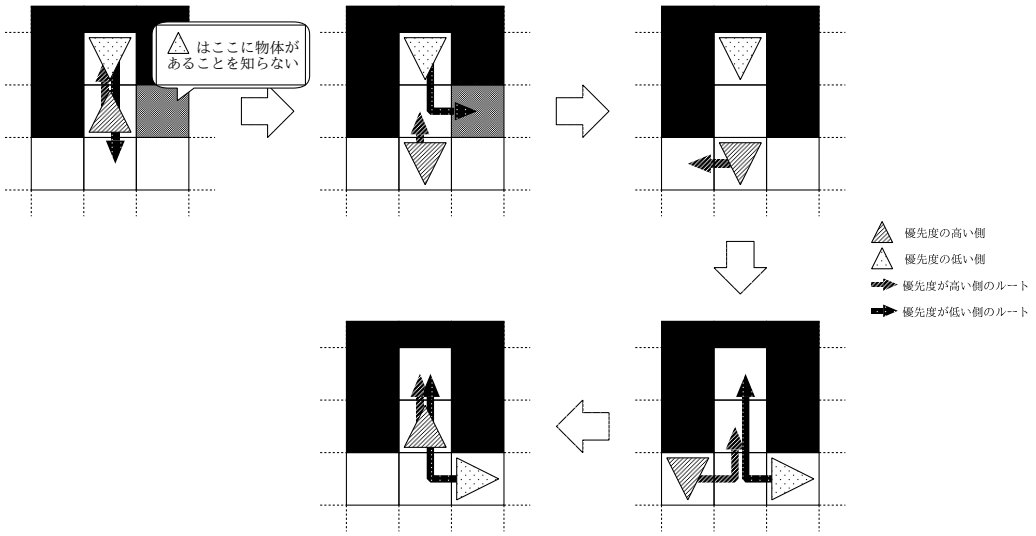

図 18: 待避地点を再検索する例

9. 指示している側の優先度が低い場合、相手を待避させる。待避に成功すれば、衝突回避を終了する。失敗、または指示している側の優先度が高い場合は衝突回避は不可能と判断して終了する。(このような処理をするのは、図 19 のような例に対応するためである。)

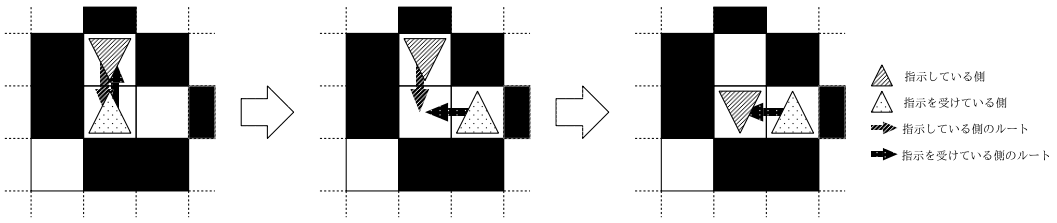

図 19: 指示している側が優先度が低く、相手を待避させることで目的地に到達できる例

7 で優先度の高いほうが待避するだけでなく優先度の低いほうも待避させるのは、優先度の低いほうはその地点で停止している可能性があり優先度の高いほうが待避してもその地点から動かず衝突回避ができないことを防ぐためである。それでも図 20 のような場合はそのままでは元の地点に戻れないため、待避場所で待機する時間には制限を設けるべきである。

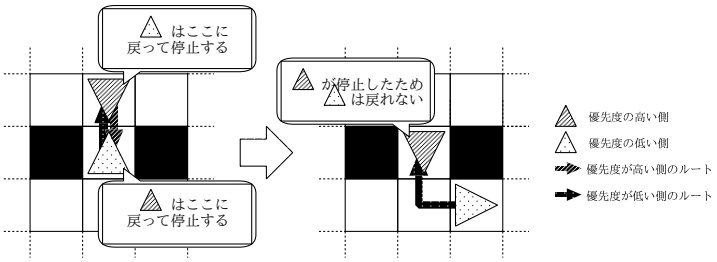

図 20: 相手が停止したため元の地点に戻れない例

5.1 迂回

以下に迂回の手順を示す (図 21)。

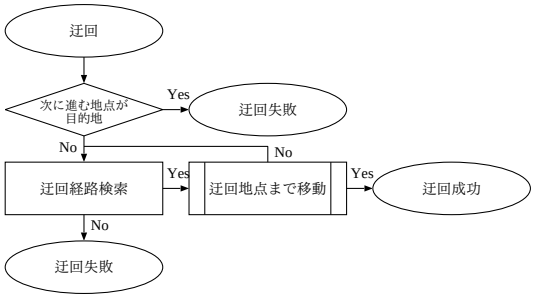

図 21: 迂回の流れ

1. 次に進む予定の地点が目的地のとき、迂回は不可能と判断し終了する。そうでないとき、2 に進む。

2. 迂回経路を検索する。(迂回経路とは、自分のルート中で次に進む地点より先の現在地から最も近い地点に次に進む地点を通らずに行ける経路のことである。) そのような経路が存在すれば、3 に進む。存在しなければ、迂回は不可能と判断し終了する。

3. 2 で検索した経路にそって迂回地点まで移動する。図 22 のように移動に成功すれば、その地点から目的地までの経路を新しいルートとし、迂回を終了する。失敗すれば、2 に戻る。(このような処理

をするのは、ロボットが持つ地点の情報が不完全なため図 23 のように迂回地点に未知の台がある場合があるからである。迂回地点に向かう経路上に未知の台が存在した場合は move_along_route の 4 により再検索が行われ、再考慮を行わなくても迂回地点に到達している可能性が高い。)

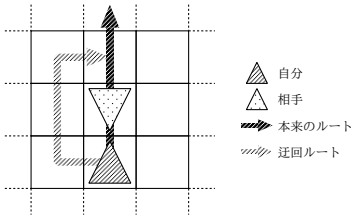

図 22: 迂回に成功する例

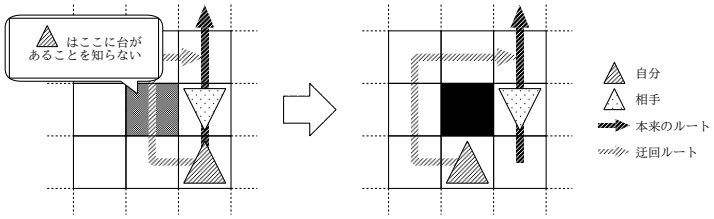

図 23: 迂回路を再検索する例

5.2 待避

以下に待避の手順を示す (図 24)。

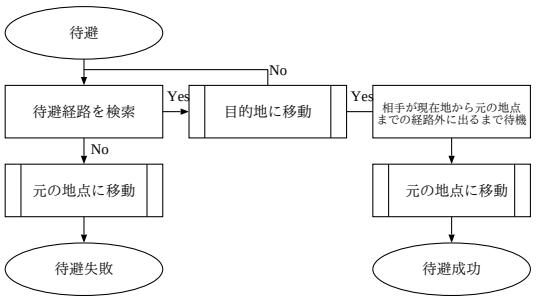

図 24: 待避の流れ

1. 待避経路を検索する。(待避経路とは、待避する相手のルート外となる地点を目的地とするルートである。) 検索に成功すれば、2 に進む。失敗すれば、元の地点に戻り、待避不可能と判断して終了する。

2. 1 で検索した経路にそって目的地まで移動する。図 25 のように移動に成功すれば、3 に進む。失敗すれば、1 に戻る。(図 26 は目的地に台があったため移動には成功しているが、待避できていない例である。)

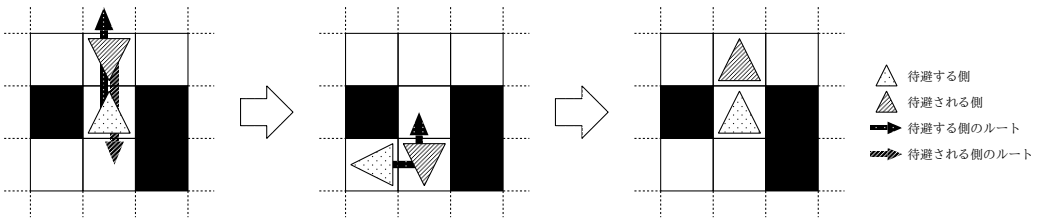

図 25: 待避に成功する例

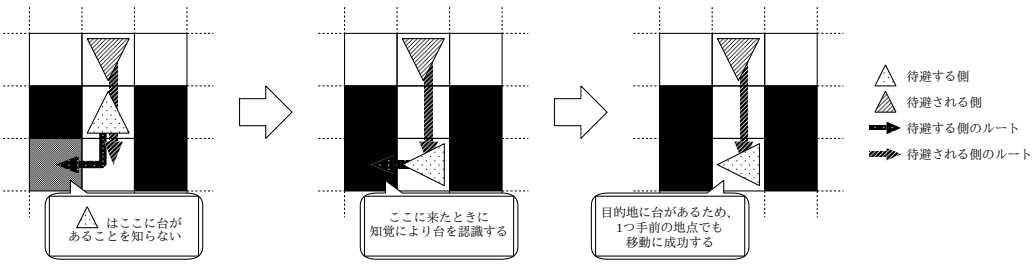

図 26: 待避地点を再検索する必要のある例

3. 現在地から元の地点に戻る経路を検索し、相手がその経路外に出るまで待機する。

4. 元の地点に戻る。

5.3 作業の優先度

表 2(p.12) の優先度について述べる。4.2 節で述べたように、今回の設定では Lift の目標のほうが先に終わる可能性が高い。そこで、Lift が Stand を認識してから Box を置くまでの処理を最優先とした。しかし、Lift の目標が達成されるためには Explorer によって予め知覚されている必要があるため、Explorer による迷路探索は Lift による迷路探索より優先されるべきである。Explorer が Lift の識別依頼に応える際、依頼地点だけでなくその道中の周辺も知覚できるため Lift の識別依頼に応えることは Explorer にとっても有益である。そこで、Lift からの依頼による探索を通常の探索より優先した。

目標を達成し作業が終了した状態は探索中の状態よりも優先度が低くされるべきである。しかし、作業終了時は元の地点に戻っていない場合があり、まだそれぞれのロボットが移動することが考えられるため、移動しているときを停止しているときよりも優先度を上げ、また両方が元の地点に戻る動作を行っているときは Explorer を優先した。

6 プランの取捨選択

4、5 章で述べた台の識別 (identify_for_lift) や識別結果の取得 (result_identify)、待避 (retract) などの処理を行うときには現在持っている目標とは異なる新しい目標が追加される。そのような場合、現在の目標を保持するか放棄するかを決める必要が生じる。そこで、新しい目標が追加されたときに保持または放棄する可能性のあるプラン (表 3) にはそのプランの発生時に running(intention 名) を信念として持ち、そのプランの終了時に削除するようにした。

表 3: プラン名と新しい目標が追加されたときの処理

保持するるプラン	
identify_unkobj	台の識別を行う
move_to_stand	Box を Stand に運搬する
放棄するプラン	
search	迷路の探索を行う
return_from_retract	待避地点から元の地点に戻る
終了を待つプラン	
look_around	現在地から 4 方向を知覚する
retract	待避する

新しいプランが追加されたとき、基本的には今まで行っていたプランを保持し新しいプランを達成したのち再開するべきである。しかし、これには例外がある。1 つ目は search である。これは 3.2 節で述べた通り、最近傍の has_looked である地点に移動しその地点から 4 方向を知覚するという作業を繰り返すことで迷路内を探索するものであった。この作業中に新しいプランが追加され現在地から移動しそのプランの目的地を目指す場合、現在地の周辺の知覚は目的地に向かっている間にある程度行われていることが多い。そこで、新しいプランの実行が終了して search を再開するとき、先ほどいた地点に戻る必要は必ずしもなく新しいプランの実行が終了したときにいる地点から search を再開することができる。そのため、新しいプランが追加されたとき今まで行われていた search を保持しておく必要はなく、search を新しく追加するべきである。

2 つ目は return_from_retract である。これは待避地点から元の地点に戻るプランである。このプランの実行中に新しいプランが追加されそれが終了したとき、保持しているプランを再開するためのプランである return_from_retract を再開してから保持しているプランを再開する必要はなく、直接保持しているプランを再開すればよい。そのため、return_from_retract は新しくプランが追加されたときに放棄すべきである。

保持しているプランを再開する場合は最も後に中断したプランから再開し、そのようなプランがなければ search を実行するようにした。

7　まとめ

今回、ロボットの協調動作の並列性と衝突回避アルゴリズムの改善を図った。これにより、BDI モデルの利点をより活かせるようになり、また解決可能な盤面の種類も増加した。

現時点での問題点として、次の 2 点が挙げられる。1 つ目として、検証した盤面が少なくこれらのアルゴリズムの妥当性が十分に示されていないという点である。2 つ目として、時間の制約により実世界でアルゴリズム改良後のロボットを使った検証を行うことができなかったという点である。しかし、2 つ目の点は先行研究では実世界のロボットを用いた実験が行われていたため、実世界で改良後のロボットを動かすことも比較的容易であると考えている。今後は盤面のサイズや台の配置などを変えアルゴリズムの妥当性を検証するとともに実世界でのロボットでも実験を行い、誤差や知覚誤りに対応できるものにしていきたい。

8　謝辞

本研究を遂行するにあたり、いつも親身に御指導くださった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] 神志那未央. 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 2012 年度卒業論文, 2013.