屋外自律走行ロボットの行為決定に関する考察

奈良女子大学 生活環境学部 情報衣環境学科
生活情報通信科学コース 4 回生 新出研究室
18480279 木村文香

令和 4 年 2 月

目次

概要

環境が常に変化する屋外環境における自律走行ロボットには、環境に応じた柔軟な行為決定が求められる。本研究では屋外の自律走行ロボットの行為決定に BDI モデルを適用し、実際の実験をもとにした仮想的な例でロボットの行動プランの例を示すことによりその有用性を論じる。

1　はじめに

近年、与えられた目標を自ら達成する自律走行ロボットの研究が活発に行われている。特に屋外では、人の往来や物の移動が活発であり、周囲の多様な環境変化に対応した確実でスムーズな走行が求められる。そのためにはロボットが自ら状況に応じて適切な行動を決めることが必要である。本研究では、実際の屋外環境における走行実験を通して従来の制御システムの問題点の検証を行った。それをもとに屋外自律走行ロボットの行為決定に関する考察を行い、BDI モデルの適用が有効であると考え、実際の実験をもとにした仮想的な例でロボットの行動プランの例を示すことによりその有用性を論じた。

2　使用したロボットと技術について

2.1　Arno

本研究では北陽電機制作の自律走行ロボット Arno を用いて実験を行った。オープンソースで提供されているロボット操作のオペレーティングシステムである ROS[1] を用いて動作制御を行っており、手動で地図を作成しウェイポイントを設定してルートを指定することで自律走行を行う。前後に搭載された 2D 測域センサ [2] で周辺環境の情報を収集している。

2.2　ROS

ROS は、ロボット操作のプログラムを開発するプラットフォームである。モーターやセンサなどロボットのハードウェアとの通信を、トピックという通信チャンネルを通したメッセージのやりとりとして、C++ や Python などの言語から利用できる機能を提供する。ROS によるロボット制御プログラムはノードと呼ばれ、Arno は、地図とセンサ情報を元にロボットの走行をコントロールする既存ノードで動作している。

3　ロボットによる実験と考察

3.1　奈良女子大学構内

G 棟 4 階廊下で自律走行実験を行ったところ、前から向かってくる人やロボットが通れるような幅であれば、経路上の障害物は回避することができた。そこで、経路を完全に阻むような壁を設置した状態で自律走行を行った。

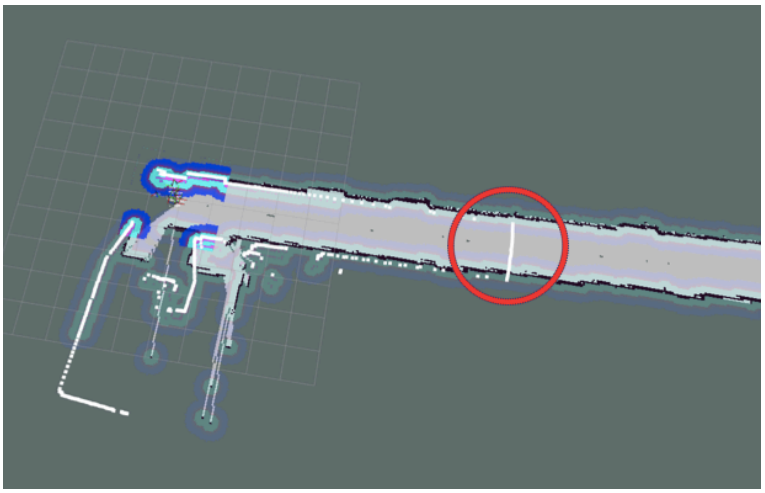

図 1　奈良女子大学 G 棟 4 階廊下の地図と壁を設置した様子

結果、ロボットは目標達成は不可能とみなして自律走行を中断した。これは将来的にこの壁が移動または無くなる可能性がある場合、目標を達成できる可能性が残っているので、問題となる。

3.2　中之島ロボットチャレンジ

人々の往来する実環境において自律移動ロボットが問題無く行動できる技術開発の公開実験の場を大学や企業向けに提供する実験走行大会である中之島ロボットチャレンジ [4] においては、中之島中央公会堂周辺一周 477m のコースを走行した。ここでの実験では、人の流れを壁と認識してしまう誤認識による自律走行の中断が起こった。

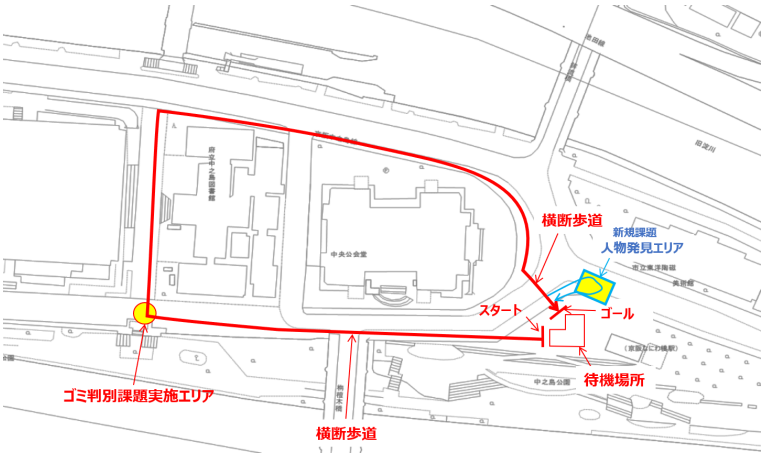

図 2　走行コース

図 3　実験の様子

3.3　エクストラロボットチャレンジ

中之島ロボットチャレンジの延長として、異なる場所での走行実験も行った。八幡屋公園内の一周 535m のコースを走行した。基本コースと、拡張コースである丘コースがあったが、実験では基本コースのみを走行した。ここでの実験では、ロボットの認識と実際の環境とのずれによる自律走行の中断が多く見られた。また、今回は基本コースのみ走行したが、拡張コースにも参加しようとすると、 現在使用している ROS ノードでは、コース全体の走行ルーチンを作り直さなければ ならないという、開発技術上の問題も残されている。

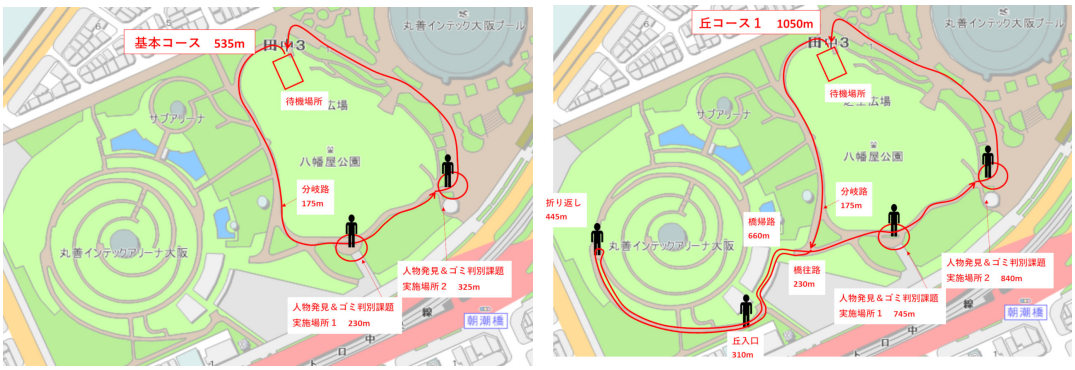

図 4　走行コース

図 5　実験の様子

3.4　考察

以上の実験から、ロボットの行動が、単一のルーチンによって決定されているということが影響を与えていると考えられる。現在はロボットの行為決定には ROS を用いているため、知覚がほぼ直接に行為に結びつく形態であり、 目標や信念の概念を明確に持つモデルを用いる場合に比べ、それらを考慮した柔軟な行為決定を 行うことは難しい。そこで、それらの概念を明確に持つ BDI モデルを用いて、こうした問題の解決が可能なのではないかと考えた。

4　BDI モデル

BDI モデルとは人間の目標達成に向けた行動決定機構の模倣で、Belief(信念)、Desire(願望)、Intention(意図) の 3 つの心的状態を用いて熟考し、自律的に行動選択を行うエージェントモデルである。BDI エージェントは成し遂げたい願望を持つと、その目標を達成するための意図を形成する。エージェントは目標を達成するためのプランを複数持つことができ、その中から信念をもとに最適なプランを意図として選ぶことでエージェントは状況に応じた適切な行動をとることができる。

4.1　BDI モデル導入の利点

従来のシステムに対する BDI モデル導入の利点は、エージェントの信念や目標などの概念が明示的に現れ、それらの概念を用いてロボットの振る舞いを設計したりプログラムしたりすることができる点である。制御に ROS だけを用いている場合、ロボットのセンサからの情報やロボットのモーターへの指令など、ハードウェアレベルの情報を用いてロボットの振る舞いをプログラムすることになる。例えば「目的地へ向かう」「障害物がある」などの情報は、「車輪のモーターを回す」「センサが何かを検出する」などと表現される。これに対し BDI モデルを使う場合、これらは「目的地へ向かう」という行為や「障害物がある」という信念で表され、モデルそのものが持っている上位レベルの概念で自然に表現されるため、ロボットにより柔軟な行為選択をさせる場合にも、プログラムの構築が行いやすくなる。また、BDI モデルに基づくエージェント記述言語 Jason によるプランニングによって、1 つの目標に対する達成プランの選択肢拡大、複数の目標行動の設定など、与えられた知覚情報をもとに適切な行動をとる自律ロボットのシステム構築が可能になる。

4.2 Jason

Jason[3] は、BDI モデルに基づくエージェント記述言語および開発環境である。エージェントの信念ベースやプランライブラリは、Prolog 言語に似た節形式で宣言的に記述される。目標や信念の発生というイベントをトリガとして、プランライブラリから熟考によりプランを選択し、そのプランの達成を意図することによって、エージェントは行為を決定する。また、アクションや知覚などの環境モデルを Java で定義することで、周辺環境とのやり取りが可能である。

4.3 Jason によるプランニング

本節以降では、3 節で明らかになった自律ロボットの実環境での走行時の問題を意識し、それをモデル化した仮想的な例における、BDI モデルによる対処について述べる。ロボットのハードウェアは ROS で制御されるものとし、ROS とエージェント記述言語 Jason の結合による仮装空間でのシミュレーションを行った。Jason と ROS の制御プログラムの間で通信を行うことにより、BDI エージェントによる行動決定を Jason 側で行った。通信は TCP プロトコルによる通信で行われる。

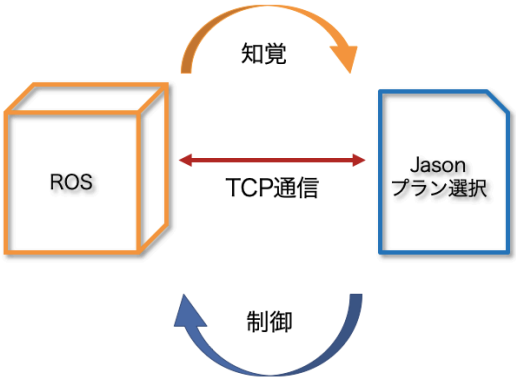

図 6　Jason による ROS の制御方法

ROS 側で知覚した情報を Jason に送ると Jason 側ではそれを信念として受け取ることができ、Jason 側からのアクションで ROS 側へ指令を送信することができる。Jason 側が送信する行動命令は、行動を視覚的に表す文字列で与えられる。ROS の知覚情報が Jason 側に送られると、Jason が行動選択を行いそれを ROS のノードに返すと ROS はロボットの制御を行う。ROS 側で特定のトピック (jason_ros/perception) に配信を行うと、Jason 側に知覚や信念を送ることができる。Jason 側では write_action というアクションを用いて、ROS 側の特定のトピック (jason_ros/action) に指令を文字列の形で配信できる。今回は仮想的な例であるので、制御の様子はコンソールへの文字列による出力で示す。Jason 側では、ROS 側から送られてきた知覚と選択した行動を表示し、ROS 側では Jason 側から送られてきた命令とロボットの行動を表示する。

実験において発生した問題を解決できることを念頭に置くため、ここでは中之島エクストラチャレンジの基本コースと拡張コースの関係を模したコースを走行することを想定する。図 7 のように、地点 0 から地点 0 まで一周する 2 つのコースがあり、コース 1 は地点 0 →地点 1 →地点 2 →地点 3 →地点 0 とたどり、コース 2 は地点 0 →地点 1 を通り、地点 2 と地点 4 を往復したのち地点 3 →地点 0 と移動する。

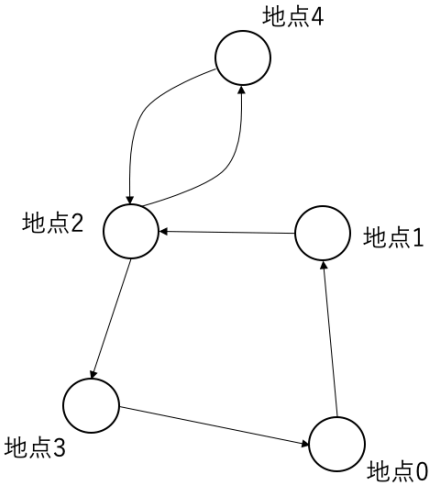

図 7　想定するコース

4.4　副プランの利用

この 2 つのコースを副プランに分割して考える。地点 0 →地点 1 →地点 3 と移動するプラン A、地点 2 →地点 3 →地点 0 と移動するプラン B、地点 2 →地点 4 →地点 2 と移動するプラン C とする。このとき、プラン A とプラン B はコース 1 とコース 2 で共有できる。

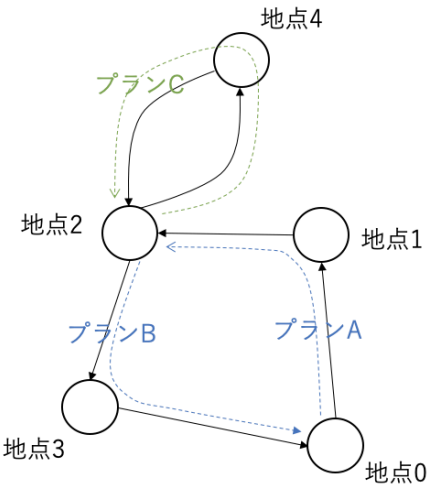

図 8　副プランに分割

作成したプログラムと実行結果

Jason 側を a.asl、ROS 側を a.py としてプログラムを作成した。この例では、a.py で人間がコース 1 と 2 のどちらを選択するかを入力している。 ROS 側が Jason に「ready_1」か「ready_2」のどちらかの信念を送ることにより、それをトリガとして Jason 側ではコース 1 またはコース 2 のいずれかを走るプランを選んで意図とし、その意図に沿って副プランを達成していく。図 9, 10 にプログラム、図 11, 12, 13, 14 に出力例を示す。

```
// 複数のコースで副プランを共有する

// こんな例を想定
// 地点0から地点0まで1周して戻る2つのコースがある
// コース1: 出発地点0→地点1→地点2→地点3→地点0
// コース2: 出発地点0→地点1→地点2まではコース0と同じ
//       続いて地点2から地点4まで往復(地点2→地点4→地点2と走行)
//       そこからはコース0と同じく地点2→地点3→地点0
// このとき
// プランAは地点0→地点1→地点2
// プランCは地点2→地点4→地点2
// プランBは地点2→地点3→地点0
// としてプランAとBをコース1と2で共有

at(0).

// 信念ready_1を得たらコース1を走る
+ready_1
<-      -ready_1;
        !course_1.
// 信念ready_2を得たらコース2を走る
+ready_2
<-      -ready_2;
        !course_2.

// コース1を走るにはプランA、プランBの順に達成
+!course_1
<-      !plan_A;
        !plan_B.

// コース2を走るにはプランA、プランC、プランBの順に達成
+!course_2
<-      !plan_A;
        !plan_C;
        !plan_B.

// プランAは、地点0にいたら、地点1に到達し、次に地点2に到達する
+!plan_A
: at(0)
<-      .print("executing plan A");
        !reach(1);
        !reach(2).
// プランCは、地点2にいたら、地点4に到達し、次に地点2に到達する
+!plan_C
: at(2)
<-      .print("executing plan C");
        !reach(4);
        !reach(2).
// プランBは、地点2にいたら、地点3に到達し、次に地点0に到達する
+!plan_B
: at(2)
<-      .print("executing plan B");
        !reach(3);
        !reach(0).

// 地点Xに動くというプラン
+!reach(X)
: at(W) // 今いる地点をWに入れておく
<-      -at(W); // Wにいるという信念を消す
        .concat("move ", X, Action);
        write_action(Action); // ROSに「move 地点」というアクションを送る
        while(not at(X)){
                .wait(100); // at(X)という信念が得られるまで0.1秒ずつ待つ
        }
        .print("has moved from ", W, " to ", X).

{include("GoalRead.asl")}
```

図 9　Jason 側のプログラム a.asl

```
#!/usr/bin/python3
# -*- coding: euc-jp -*-
import rospy
from std_msgs.msg import String
import time

rospy.init_node('a')

pub = rospy.Publisher('jason_ros/perception', String, queue_size = 1000)
def callback(msg):
        action_string = msg.data
        rospy.loginfo('from Jason: [%s]' % action_string)
        action = action_string.split()
        if action[0] == 'move' and len(action) >= 2:
                rospy.loginfo('moving to %s' % action[1])
                time.sleep(1)
                pub.publish(String('a +at(%s)' % action[1]))
                rospy.loginfo('moved to %s' % action[1])
        else:
                rospy.loginfo('unknown action: %s' % action_string)

rospy.Subscriber('jason_ros/action', String, callback)
print("Which course? 1or2")
num=input()
time.sleep(1)
pub.publish(String('a +ready_%s'% num))
rospy.spin()
```

図 10　ROS 側のプログラム a.py

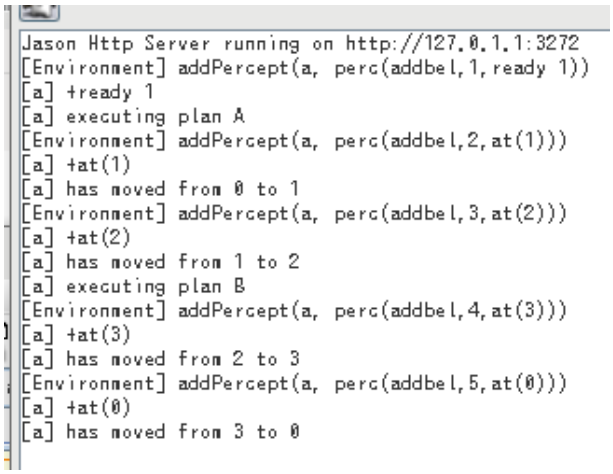

図 11　コース 1・Jason 側の動き

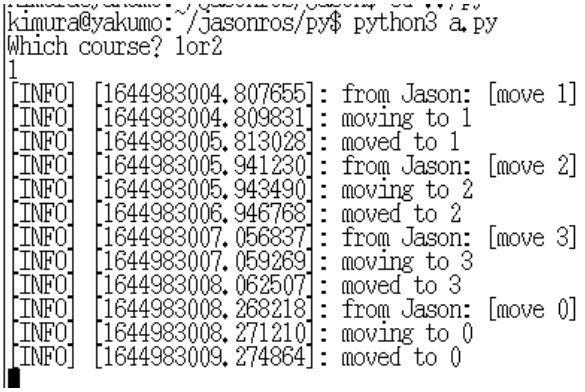

図 12　コース 1・ROS 側の動き

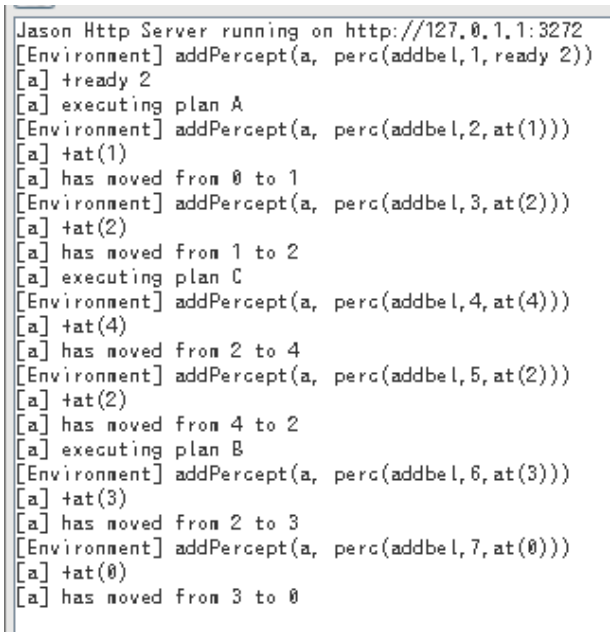

図 13　コース 2・Jason 側の動き

図 14　コース 2・ROS 側の動き

この例は、3.3 節で述べた、コースごとに全体のルーチンを作り直さなければならない問題に対応している。副プランへの分解とそれらの再利用により、新たなルートの構築やルートの修正が行いやすくなる。BDI モデルの適用により、こうしたことは単に「複数のルートに共通する部分を関数としてまとめる」ではなく 「副目標とそれを達成する副プランの共用」と見ることができ、ロボットの行動の設計を上位レベルの 概念で行うことを容易にしている。

4.5　自律走行の妨げとなる障害物に対するプランニング

この例は、3.1 節や 3.2 節で述べた、動く可能性のある壁や人を障害物として認識した時に目標達成ができなくなる問題への対応に相当する。

経路上に、移動または無くなる可能性のある障害物がある場合を考える。図 15 で、地点 1 に障害物があるとする。

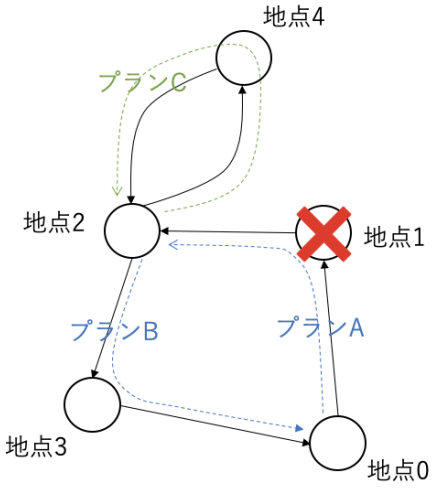

図 15　地点 1 に障害物がある

作成したプログラムと実行結果

Jason 側を b.asl、ROS 側を b.py としてプログラムを作成した。図 16 は、前の例で使用した Jason プログラム a.py に変更を加えた箇所のみを示す。ROS 側は、最初は地点 1 に障害物があるという信念を Jason に持たせておき、エージェントが出発したのちしばらく経ったら地点 1 に障害物があるという信念を消す。Jason 側は障害物があるという信念を得ると地点 1 へ行くという副目標が達成できなくなるが、 ここでコース全体のプランを失敗させるのではなく、 しばらく (プログラム上は 1 秒) 待ってからその副目標の達成を再度試みる。障害物があるという信念が無くなるまでこれを繰り返す。ROS 側から障害物があるという信念が消されると、エージェントは動くという命令を ROS に送ってその副目標を達成し、コース全体を走るというプランの達成に向かうことができる。図 16, 17 にプログラム、図 18, 19 に出力例を示す。ただし、現実の例では障害物が移動する可能性のあるものかどうかをロボット側で判断する過程が必要になるであろう。

この例でも、BDI モデルの適用により、ロボットの行動を制御するループにハードウェアレベルのセンシングによる 分岐を多数記述するのではなく、基本となるプランとそれに対する失敗回復処理とをそれぞれ「目標を達成するプラン」という概念で書き、両者を分離して記述することができている。

```
// 地点Xに動くというプラン
+!reach(X)
: at(W) // 今いる地点をWに入れておく
  & not obstacle_at(X) // Xに障害物があるという信念がなければ
<-      -at(W); // Wにいるという信念を消す
        .concat('move ', X, Action);
        write_action(Action); // ROSに「move 地点」というアクションを送る
        while(not at(X)){
                .wait(100); // at(X)という信念が得られるまで0.1秒ずつ待つ
        }
        .print('has moved from ', W, ' to ', X).

// ゴール!reach(X)が失敗したときのプラン
-!reach(X)
<-      .print('waiting for disappearance of obstacle at ', X);
        .wait(1000); // 1秒待つ
        .print('retrying...');
        !reach(X). // 再び!reach(X)というゴールを生成することにより再トライ
```

図 16　Jason 側のプログラム b.asl(変更点)

```
#!/usr/bin/python3
# -*- coding: euc-jp -*-
import rospy
from std_msgs.msg import String
import time

rospy.init_node('b')

pub = rospy.Publisher('jason_ros/perception', String, queue_size = 1000)
def callback(msg):
        action_string = msg.data
        rospy.loginfo('from Jason: [%s]' % action_string)
        action = action_string.split()
        if action[0] == 'move' and len(action) >= 2:
                rospy.loginfo('moving to %s' % action[1])
                time.sleep(1)
                pub.publish(String('b +at(%s)' % action[1]))
                rospy.loginfo('moved to %s' % action[1])
        else:
                rospy.loginfo('unknown action: %s' % action_string)

rospy.Subscriber('jason_ros/action', String, callback)
print("Which course? 1or2")
num=input()
time.sleep(1)
# 最初は地点1に障害物があるという信念を持たせておく
pub.publish(String('b +obstacle_at(1)'))
time.sleep(1)
# エージェントに出発させる
pub.publish(String('b +ready_%s'%num))
# しばらく待ったら地点1に障害物があるという信念を消す
time.sleep(2.5)
pub.publish(String('b -obstacle_at(1)'))
rospy.spin()
```

図 17　ROS 側のプログラム b.py

```
Jason Http Server running on http://127.0.1.1:3272
[Environment] addPercept(b, perc(addbel,1,obstacle at(1)))
[b] +obstacle at(1)
[Environment] addPercept(b, perc(addbel,2,ready 1))
[b] +ready 1
[b] executing plan A
[b] waiting for disappearance of obstacle at 1
[b] retrying...
[b] waiting for disappearance of obstacle at 1
[Environment] addPercept(b, perc(delbel,3,obstacle at(1)))
[b] -obstacle at(1)
[b] retrying...
[Environment] addPercept(b, perc(addbel,4,at(1)))
[b] +at(1)
[b] has moved from 0 to 1
[Environment] addPercept(b, perc(addbel,5,at(2)))
[b] +at(2)
[b] has moved from 1 to 2
[b] executing plan B
[Environment] addPercept(b, perc(addbel,6,at(3)))
[b] +at(3)
[b] has moved from 2 to 3
[Environment] addPercept(b, perc(addbel,7,at(0)))
[b] +at(0)
[b] has moved from 3 to 0
```

図 18　Jason 側の動き

```
^Ckimura@yakumo:~/jasonros/py$ python3 b.py
Which course? 1or2
1
[INFO] [1644983167.660524]: from Jason: [move 1]
[INFO] [1644983167.662835]: moving to 1
[INFO] [1644983168.666113]: moved to 1
[INFO] [1644983168.875782]: from Jason: [move 2]
[INFO] [1644983168.878402]: moving to 2
[INFO] [1644983169.881471]: moved to 2
[INFO] [1644983170.087463]: from Jason: [move 3]
[INFO] [1644983170.089928]: moving to 3
[INFO] [1644983171.093729]: moved to 3
[INFO] [1644983171.298852]: from Jason: [move 0]
[INFO] [1644983171.302090]: moving to 0
[INFO] [1644983172.305422]: moved to 0
```

図 19　ROS 側の動き

4.6　複数ルートを所持するプランニング

この例も前節同様ロボットが目標達成の途中で障害物に遭う例であるが、ロボットが代替プランを持っている場合を 想定したものである。図 20 で、経路を塞ぐ障害物があり通行が不可能な場合を考える。地点 3 が通行不可能であるとする。エージェントに、地点 5 を経由して地点 0 に向かう別ルートを所持させておく。

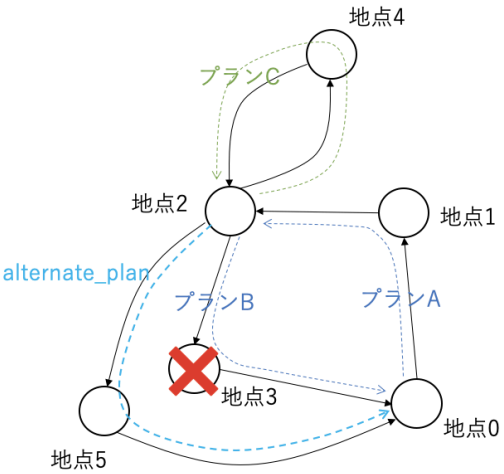

図 20　地点 3 が通行不可・地点 5 を経由して地点 0 に行く経路を追加

作成したプログラムと実行結果

Jason 側を c.asl、ROS 側を c.py としてプログラムを作成した。図 21 は、変更点のみを示す。ROS から地点 3 に障害物があるという信念を持たせておく。地点 2 において Jason は 、地点 0 に向かうプランとしてまずプラン B を選んで達成を試みるが、 地点 3 に障害物があるとの信念により、その達成を不可能と判断する。しかし、プラン B が失敗したときの処理として別なプラン alternate_plan を選ぶことによって経路の再設定を行い、目標を達成に向かうことができる。図 21, 22 にプログラム、図 23, 24 に出力例を示す。

```
+!alternate_plan
:at(2)
<- .print('rerouted');
   !reach(5);
   !reach(0).

-!plan_B
<-      .print('cannot pass');
        !alternate_plan. //別ルートを選択
```

図 21　Jason 側のプログラム c.asl(変更点)

```
#!/usr/bin/python3
# -*- coding: euc-jp -*-
import rospy
from std_msgs.msg import String
import time

rospy.init_node('c')

pub = rospy.Publisher('jason_ros/perception', String, queue_size = 1000)
def callback(msg):
        action_string = msg.data
        rospy.loginfo('from Jason: [%s]' % action_string)
        action = action_string.split()
        if action[0] == 'move' and len(action) >= 2:
                rospy.loginfo('moving to %s' % action[1])
                time.sleep(1)
                pub.publish(String('c +at(%s)' % action[1]))
                rospy.loginfo('moved to %s' % action[1])
        else:
                rospy.loginfo('unknown action: %s' % action_string)

rospy.Subscriber('jason_ros/action', String, callback)
print("Which course? 1or2")
num=input()
time.sleep(1)
# 地点3に障害物があるという信念を持たせておく
pub.publish(String('c +obstacle_at(3)'))
time.sleep(1)
# エージェントに出発させる
pub.publish(String('c +ready_%s'%num))
rospy.spin()
```

図 22　ROS 側のプログラム c.py

```
Jason Http Server running on http://127.0.1.1:3272
[Environment] addPercept(c, perc(addbel,1,obstacle at(3)))
[c] +obstacle at(3)
[Environment] addPercept(c, perc(addbel,2,ready 1))
[c] +ready 1
[c] executing plan A
[Environment] addPercept(c, perc(addbel,3,at(1)))
[c] +at(1)
[c] has moved from 0 to 1
[Environment] addPercept(c, perc(addbel,4,at(2)))
[c] +at(2)
[c] has moved from 1 to 2
[c] executing plan B
[c] cannot pass
[c] rerouted
[Environment] addPercept(c, perc(addbel,5,at(5)))
[c] +at(5)
[c] has moved from 2 to 5
[Environment] addPercept(c, perc(addbel,6,at(0)))
[c] +at(0)
[c] has moved from 5 to 0
```

図 23　Jason 側の動き

```
^Ckimura@yakumo:~/jasonros/py$ python3 c.py
Which course? 1or2
1
[INFO] [1644983332.809256]: from Jason: [move 1]
[INFO] [1644983332.812328]: moving to 1
[INFO] [1644983333.815801]: moved to 1
[INFO] [1644983333.928291]: from Jason: [move 2]
[INFO] [1644983333.930873]: moving to 2
[INFO] [1644983334.933954]: moved to 2
[INFO] [1644983335.143265]: from Jason: [move 5]
[INFO] [1644983335.145308]: moving to 5
[INFO] [1644983336.147969]: moved to 5
[INFO] [1644983336.354244]: from Jason: [move 0]
[INFO] [1644983336.357020]: moving to 0
[INFO] [1644983337.359942]: moved to 0
```

図 24　ROS 側の動き

5　まとめ

BDI モデルを従来の自律走行ロボットに適用することで、現実世界の問題に柔軟に対応するためのロボットの行動決定の構築が、目標や信念といった上位概念を用いて見通しよく行え、結果的に、より確実でスムーズな自律走行の実現が期待できると考える。今回は仮想的な例で、実験で起こった問題に対応する行動プランの例を示した。今後の課題としては、実際に屋外ロボットに対する BDI エージェントの実装と実環境での実験を行い、本手法の有用性を示すことが挙げられる。それには、今回は抽象的に扱ったロボットの動作部分を実際のロボットの走行ルーチンで 置き換えたり、障害物の種別判定などのロボットの思考部分を実際に構築するなどの過程が必要になる。

6　謝辞

本研究及び本論文の執筆にあたり、自律走行ロボットを提供してくださった北陽電機株式会社様、また、最後まで親身にご指導くださった新出尚之准教授に深く感謝いたします。

参考文献

[1] Open Robotics. ROS Wiki. https://wiki.ros.org/ja. (2022 年 2 月 16 日閲覧).

[2] 北陽電機. URM-40LC/LCN-EW. https://www.hokuyo-aut.co.jp/search/single.php?serial=189. (2022 年 2 月 16 日閲覧).

[3] Rafael H. Bordini, Jomi Fred Hübner, and Michael Wooldridge. *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons, 2007.

[4] 中之島ロボットチャレンジ　https://www.nakanoshima-rc.jp

[5] 片山寛子「環境認識の動的な変化に応じて プランを変更する BDI エージェント 」, 2009 年度卒業論文, 奈良女子大学理学部情報科学科, 2010

[6] 藤田 恵, 片山 寛子, 小島 侑子, 新出 尚之.「動的環境における BDI ベースのロボットの再プランニングによる再行動決定法」, The 24th Annual Conference of the Japanese Society for Artificial Intelligence, 2010.

[7] 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」, 情報処理学会研究報告 IPSJ SIG Technical Report, Vol. 2011-MPS-85 No.10, 2011.

[8] Megumi Fujita, Yuki Goto, Naoyuki Nide, Ken Satoh, Hiroshi Hosobe, Toward a robot that acquires logical recognition of space, Information Engineering Express International Institute of Applied Informatics 2017, Vol.3, No.4, P.1-10, 2017.