2023 年度　修士論文

自律型システムの行動の根拠を可視化することによる情報の獲得と自律型システムの利便性の向上

奈良女子大学大学院 人間文化総合科学研究科 情報衣環境学専攻
生活情報通信科学コース 博士前期課程 新出研究室
木下美咲

2024 年 2 月 13 日

概要

近年、人間に指示されたとおりに行動するのではなく、システム自身で思考し、その結果から行動を決定する自律型システムの研究が様々な分野で進んでいる。自律型システムでは、自身の周囲の環境情報や過去の行動の結果から学習を行うことで、最適と思われる行動を選択する。ほとんどのケースでは、自律型システムに選択される行動は、人間にとっても周囲の環境情報などから考えて、最適と思われる選択である。しかし、自律型システムによって選択された行動が人間の想定していた最適な行動と異なることがある。そしてそのようなケースでは、なぜそのような行動を自律型システムが選択するに至ったのか、周囲の環境情報から考えても人間にわからないことが多い。これでは、人間が自律型システムの行動選択を改善することが難しくなる。そこで本研究では、自律型システムがある行動を選択したときにどの様な環境に置かれていて、その選択を行う決定材料として何を参照していたのかを人間が視覚的に理解することができ、その自律型システムの行動の根拠が人間にとって明らかになることを目指す。

自律型システムには様々な種類が存在し、最終的にはどのような自律型システムにも応用できるような仕組みの実現が望まれる。しかし、本研究では、その第一歩として、自律型システムの中から自律走行型ロボットを挙げ、その行動の根拠を視覚的に明らかにすることで、ある行動が選択されたときに人間が自律型システムの思考を理解できることを目標とする。

目次

1　はじめに

本研究では、概要で述べたように自律型システムの例として、自律走行型ロボット（以下、ロボットと呼ぶ。）を挙げる。ロボットは、人間の定めたスタートからゴールまでのルートを周囲の環境情報や、事前に作成された地図などから決定し、自律走行を行うシステムである。

基本的に、ロボットにより作成されたルートは、人間が考えるようなルートと同様の合理性を持ち、障害物やロボットが通ることができないような細い道などを避けたものとなっている。しかし、その過程では、なぜロボットがそのようなルートを選択したのか、人間が理解できない箇所が存在したり、目的を上手く達成できないルートが選ばれることもある。そこで、そのような箇所が存在するルートが選択されたときに、ロボットがどのような周囲の環境情報を参考にし、どのような思考をしていたのかという、**自律走行型ロボットの選択の根拠を可視化**することを本研究では目標とする。

ロボットのルート決定の根拠を可視化することで、人間が発見できていなかった最適なルートをロボットが発見したとき、その理由を明らかにすることができ、さらに、自律走行時に失敗が起こった場合には、「なぜその失敗が起こってしまったのか」という原因も視覚的に明らかにすることができる。これにより、同じ失敗を繰り返さないように人間が対策を行うことも可能であると考える。

また、本研究では、地図やシミュレーション空間を用いて実験を行う。この地図やシミュレーション空間は、現実世界でロボットに自律走行させたい環境から作成したものである。実際に自律走行させたい環境から作成した地図やシミュレーション空間を使用することで、実際に自律走行を行う前に、ルート選択の失敗の起こりそうな箇所の発見や、その失敗の起こりそうな箇所に対する対応などを、人間が事前に考えることができる。これにより、実際の環境においての実験の際に生じる人間の負担を減らすことができると考える。

以上のようにシミュレーション空間を用いて、ロボットの選択の根拠を可視化することで、従来の実験で自律走行達成の壁となっていた問題を解決することができる。

例えば、従来の実験では、ロボットに自律走行をさせるためにロボットのセッティングや地図の確認などの準備に多くの時間を割いていた。しかし、シミュレーション空間で実験を行うことにより、これらの準備にかける時間を短くすることができる。また、実際の環境では、１回の自律走行に時間を要するために何度も自律走行を試すということができなかったが、シミュレーション空間では短時間で何度でも実験が可能になる。

実際の実験環境では、人間がロボットの行動やその根拠になったものを逐一確認することは難しい。その理由としてロボットを一時停止させて再発進させると、自己位置推定のズレが起こる可能性があり、走行失敗の原因になりやすいため、自律走行中に一時停止してロボットの周囲の環境の認識について調べにくいことや、走行時に人間はロボットの安全確保に意識を集中しなければならないため、ロボットの環境認識に十分な注意を払えないなどの事情がある。またそのため、自律走行の失敗が起きたとき、原因をその場で探ることも困難であった。しかし、これをシミュレーション空間で行い、さらにそのときのロボットの選択の根拠を可視化できることで、人間がロボットの行動などを注視することができ、かつロボットがどのようにルートを決定しているのかという根拠を合わせて見ることで実際の環境で走行させるときに失敗の起こりそうな地点を発見することができるのである。

このようにシミュレーション空間を用いることで、実際の環境における自律走行時の人間の負担を減らし、ロボットの選択の根拠を可視化することで、事前に自律走行時に起こり得る可能性がある失敗に備えることができるのである。

本論文では、特にロボットの実験を始めたばかりの人や、実験したことのない未知の環境で行う自律走行を

念頭に置いた可視化を行う。これは、我々が２０２１年に行った自律走行ロボットを使用した初めての実験での失敗とそこから得た課題の解決を目的としているためである。この実験では、自律走行をロボットに行わせた環境はロボットにとってもロボットを操作する人間にとっても初見の環境であり、一度地図を作成しただけではその環境の状況を把握したり、自律走行時に注意する点を割り出すことが不可能であった。さらに、ロボットの操作に不慣れであると、効率よく何度実験を行うことが難しい。そこで、本研究では、そのような環境を想定し、自律走行を行う環境がそのようなものであるかを簡単に理解できたり、シミュレーション空間を用いて何度も実験を行うことができることを目指す。２０２１年に我々が行った実験については、4.3.4 節で詳しく述べる。

以下、2 節では、本研究で使用するロボットでの自律走行の実現方法とその過程で発生する課題と解決策について、3 節では、ロボットの選択の根拠を可視化するシステムについて述べる。また、4 節では、実験を行う地図とシミュレーション空間の獲得についてと実験の結果について述べる。そして、最後に 5 節で今後の展望と課題についてまとめる。

2　自律走行型ロボット

1 節で述べたように、本研究では自律型システムの具体例として自律走行型ロボットを挙げる。本節では、2.1 節で、ロボットの仕様について、2.2 節で、自律走行の実現方法について、2.3 節で、自律走行時の課題と解決策について述べる。

2.1　自律走行型ロボットの仕様

本研究で、想定するロボットは、大阪市で行われる自律ロボットの走行大会「中之島ロボットチャレンジ [7]」で奈良女子大学新出研究室と北陽電機が走行させている「Arno（アルノー）」とする。Arno は、前に２つ、後ろに１つの計３つのタイヤによって移動する。そして、機体の前後に１つずつ２次元センサ [8] を設置しており、この２つのセンサの情報から周囲の環境を認識している。図 1 に自律走行型ロボット Arno を示す。

2.2　自律走行の手順

Arno が自律走行を行うには、**地図作成**、**ウェイポイント作成**、**ナビゲーション**、の３つのステップが必要である。以下にそれぞれの手順について詳しく述べる。

2.2.1　地図作成

自律走行時に用いられる地図は、実際の環境から作られたものであり、ロボットが自律走行を行う際に、ルートを決定する過程で参照するものである。地図の作成は、ロボットに接続されているコントローラによって人間が操作することによって行う。

まず、自律走行のスタート地点としたい場所を決定し、その地点で地図作成プログラムを起動させる。次に、地図を作成したい範囲でロボットを走行させることで、2.1 節で述べたセンサが周囲の環境を記録する。このとき、ロボットを走行させるルートは、自律走行させたいルートと一致しなくても良い。最後に自律走行させたい範囲の地図ができていることをロボットに搭載された PC で確認し、地図を保存して終了する。また、自律走行のゴールは後で指定することができるため、地図作成プログラムを終了させる地点に気を使う必要はない。

図 1　Arno

図 2 にロボットを人間が操作することによって作成した地図の例を示す。この地図は、大阪市にある ATC（アジア太平洋トレードセンター）で作成したものである。

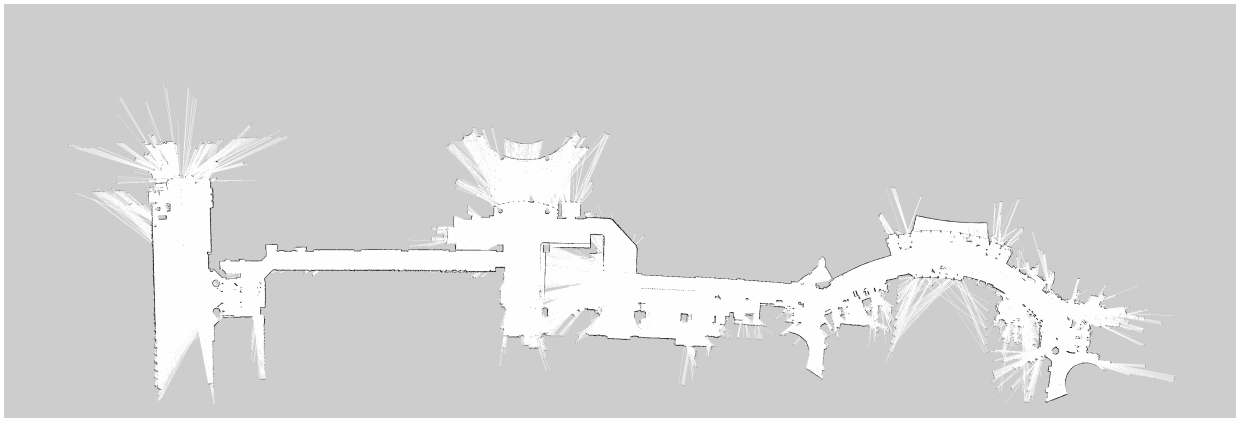

図 2　地図作成プログラムによって作成された地図

2.2.2　ウェイポイント作成

ウェイポイントとは、地図作成を行った範囲の中で、人間が自由に指定することができるポイントであり、自律走行時にロボットはこの指定されたポイントを通過することにより、目的の地点に向かう。ウェイポイントは、地図の範囲内で、複数個指定することができ、間隔や個数を調整することも可能である。これにより、人間が大体のルートを決定したり、ロボットに通過させたい箇所を指定することもできる。また、ウェイポイントの中で最後に指定したものが自律走行のゴールとなる。

ウェイポイントの作成は、手動と自動の 2 通りで行うことが可能である。

手動で作成する場合、2.2.1 節で作成された地図を見ながら、人間がウェイポイントの位置を考える。まず、作成した地図を PC に表示し、次にその中でウェイポイントを打ちたい地点にカーソルを合わせ、最後に向きを調整することで作成できる。

自動で作成する場合は、ロボットを地図作成時と同様にコントローラで操作することが必要である。まず、地図作成をスタートした地点（自律走行時のスタート地点）で地図を開く。そして、その地点で自動でウェイポイントを作成するプログラムを動かし、ロボットに自律走行させたいルートをコントローラーで操作して、ロボットを走らせる。最後に自律走行のゴール地点としたい場所で、プログラムを終了するとウェイポイントが設定されている。

手動で指定した場合、ロボットにコース内で通過してほしいポイントを人間が数箇所のみ指定することにより、そのポイントを通過する以外のルートは、自律走行時にロボットが周りの環境を認識することから決定する。そのため、ルートの作成に自由度が出る一方で、人間が地図のみを見ながら指定することで、ロボットが通過できない誤った場所にウェイポイントを指定してしまうといった問題が起こることがある。自動で指定した場合には、ロボットを実際に動かしてウェイポイントを指定しているので、手動の場合のような誤ったルートの指定はない。また、ウェイポイントも手動で指定する場合に比べ多く指定することが容易であり、それによってロボットの走行ルートをより精密に指定できるが、反面、ロボットによるルート決定に自由度が出ない問題もある。この違いを鑑み、新出研究室での近年の実験走行ではもっぱら後者を用いている。

ウェイポイントは地図上の座標で記録され、１つの地図に対して作成されたウェイポイントの全ての座標がまとめて１つのウェイポイントファイルに書き出され、保存されるようになっている。

2.2.3 ナビゲーション

ナビゲーションは、作成した地図と、その地図上に指定されたウェイポイントを用いて行われる。ナビゲーションシステムに、地図とウェイポイントを渡すと自律走行がスタートするのだが、このときに地図作成をスタートさせた地点にロボットをセットする必要がある。これにより、ナビゲーションシステムが、センサから送られてくる実際の環境情報と地図を照らし合わせることができる。自律走行がスタートすると、ナビゲーションシステムは 2.2.2 節で述べたウェイポイントファイルに記された座標を上から順に１つずつ読み込んでいく。１つの座標を読み込み、その座標の位置にロボットを移動させ、到達したら次のウェイポイントを読み込み、またその座標にロボットを移動させ、到達したら次の座標を読み込み…といった順序である。以上の工程を繰り返し、ウェイポイントファイルに書かれている最後の座標に到達したときに、ゴールに到達したとみなす。

2.3 自律走行時の課題と解決策

2.3.1 課題

2.2.3 節で述べたように、ロボットはセンサによって周りの環境情報を取得し、その環境情報を地図とマッチングすることで自律走行を行う。また、センサから取得される環境情報には、建物などの地図にも記録されていて、マッチングに利用される情報の他に、障害物や人などといった地図作成時にはなかったが、自律走行時には存在するといった情報も含まれる。このように環境情報には様々な情報が含まれていて、そのために自律走行を行うことができる。

その一方で、これらの情報によって自律走行が失敗してしまうという問題点もある。このようなセンサによる認識の問題点として１つ目に挙げられるのが、人や障害物を正確に区別することが難しいという点である。

例えば、多くの人が密集しているのを壁と認識したり、立ち止まっている人を木やポールだと認識するといった誤認識が起き、その結果、ある方向へ行けるか否かの判断を誤ることがある。そして、２つ目に挙げられる問題点が誤ったマッチングである。ロボットは、自律走行を行う際に、常に実際の環境と地図とのマッチングを行っている。そして、地図上で現在走行している地点とより似ている地点があった場合には、そのより似ている地点へと自己位置を変更してしまう。実際に走行していた地点から自律走行型ロボットの自己位置が変更されたしまったことにより、その後の自律走行が上手くいかなくなり、ゴールに辿り着くことができなくなるのである。

ロボットによるある選択によって自律走行の失敗が起こった際に、どのような環境情報が参考とされていたのかを人間が知ることで、不適切な行動の原因も知ることができると考えられる。例えば、地図を特徴ごとに色分けし、人間がその地図を見ることによって自律走行を行う環境がどのようなものかを知ることができるようにしたり、実際の環境で自律走行をする前に何が失敗の原因となり得るかを実際の環境をモデルとしたシミュレーション空間で試すことができることは、自律走行の失敗が起こった際にロボットが不適切な行動を取った原因を人間が明らかにする手がかりとなるだろう。2.3.2 節では、以上で挙げた課題を解決するための解決策をより詳しく述べる。

2.3.2 解決策

本研究では、ロボットによる、ある選択がどのような根拠によって行われたかを知るために、３つのプログラムを作成する。

まず、１つ目に人間が地図から情報を得られるように、地図上の障害物にその特徴ごとに色の振り分けを行い、新たな地図を作成するプログラムである。従来の地図は、障害物を黒の１色のみで示されていたが、これを、障害物の種類を推測してその特徴ごとに色分けすることで、人間が地図を見たときに、自律走行時に失敗の起こりそうな地点を視覚的に把握できるようにする。色の振り分け方など詳しくは 3.2 節に示す。

２つ目に、人間にとっての視認性を上げるために、色分けした地図をシミュレーションを行う際に使用する ROS 付属の可視化ツールである rviz[4] へと立体的に表示するプログラムである。rviz に表示する際には、障害物を立体物として表示することで更に人間にわかりやすいものとする。このプログラムについては、3.3 節で詳しく述べる。

３つ目のプログラムは、シミュレーション空間で、ロボットの通過した地点から一番近い障害物の情報が表示されるものである。多くの場合、ロボットの進路決定に最も影響を及ぶすのが一番近い障害物なので、それを強調することによって、ロボットが自律走行を行う際にどのような障害物を見ていて、行動の根拠になっているかを人間にわかりやすく示す。どのような情報を示すかなど、詳しい内容については 3.4 節で述べる。

3 実装

本節では、2.3.2 節で述べたプログラムについて詳しく述べる。まず、3.1 節では、本研究で使用するプログラムの仕様について述べる。また、3.2 節には、従来の地図を色付けして見やすい地図へと改造するプログラムについて、3.3 節では、色分けした地図を rviz へと立体的に表示するためのプログラムについて、3.4 節には、ロボットが通過した地点の付近にどのような障害物があったかを表示するプログラムについて述べる。

3.1 本研究で使用するソフトウェアについて

本研究では、ロボット用プラットフォームやソフトウェア開発を支援するライブラリツールを備えたソフトウェアフレームワークである、ROS(Robot Operating System) [3] を使用する。ROS では、ノードと呼ばれる実行ファイル間で情報を購読、配信することで、センサの情報やモータの制御信号などの伝達を行い、これによってロボットの動作を制御している。そこで、2.3.2 節で述べた、３つのプログラムをそれぞれ１つのノードとして作成する。

3.2 地図の色分けを行うプログラム

本節では、従来の地図（以下、map と呼ぶ。）を改造して、人間にとって見やすい色分けした地図（以下、colormap と呼ぶ。）を作成するプログラムについて述べる。まず map は、white, gray, black の３色の点の集合で表されている。図 2 の白い点が free、灰色の点が unknown（センサのレーザーが届かないため未判定の場所)、それらの境の黒い部分が occupancy（障害物などで占有されている地点）である。map では、black のみで示されていた障害物を colormap では、red, pink, lightblue, blue の４色に振り分け、white, gray を合わせた計６色からなる地図に改造する。

以下、3.2.1 節に地図データの形式と地図データを保存するためのファイルの形式について、3.2.2 節に colormap の色の振り分け方について述べる。

3.2.1 pgm ファイルと ppm ファイル

map は、地図のサイズなどの情報が記述されている yaml ファイルと pgm ファイルから構成される。pgm ファイルには、ファイルの形式、地図の高さ、幅、ホワイトポイントに加えて、地図全体を値に変換した文字列が記述されている。そして rviz 上で地図を表示する際には、上記で述べた pgm ファイルに書かれているデータが ROS の nav_msgs/OccupancyGrid というメッセージ型に変換され、 /map という ROS トピックを介して rviz に配信される。このメッセージ型の data というフィールドに、地図全体の各点のデータが white を 0、gray を −1、black を 100 として表した 8 ビットの整列の配列として格納されている。表 1 は、/map トピックに配信されるメッセージの data フィールドの配列の一部を示したものである。

表 1　data フィールドの配列

...,-1,-1,-1,-1,-1,-1,-1,-1,	...	-1,-1,-1,-1,-1 -1, ...
...,-1,-1,100,100,-1,-1,	...	0,100,-1,100 0, 0, ...
...,-1,-1,-1,100,0,-1,	...	-1,-1,-1,0,0,-1, ...
...,-1,-1,0,100,-1,-1,	...	-1,100,-1,-1,0,1, ...
...,-1,-1,-1,-1,-1,-1,-1,	...	-1,0,-1,-1,-1,-1, ...

/map を介して配信された地図のデータを受け取り、data フィールドの配列の内容を書き換え、新たな画像ファイルを作成することで colormap は完成する。map は pgm ファイルであり、従って色はグレースケールのみであるため、図 1 のように、data フィールドの配列は、１つの値で１つのカラーを表していた。しかし colormap では、red や blue などの他の色を追加するために、colormap を記述するファイルの形式を ppm ファイルとする。ppm ファイルでは、３つの値で１つのカラーを表す。そして、その３つの値は、それぞれ

r(RED), b(BLUE), g(GREEN) に対応しており、その値を調整することで、様々なカラーを表現することが可能である。表 2 に pgm ファイルでの地図データの表現と ppm ファイルでの地図データの表現のカラー値の対応表を示す。

表 2　地図データのカラー値の対応表

pgm ファイル

color	value
white	254
black	0
gray	205

ppm ファイル

color	value(rbg)		
	RED	BLUE	GREEN
white	255	255	255
black	0	0	0
gray	100	100	100
red	255	0	0
pink	215	169	187
lightblue	104	182	231
blue	0	0	255

3.2.2　色の振り分け

black のみで表されていた障害物を red, pink, lightblue, blue のそれぞれの色に振り分ける際に、障害物を示す black で表されている地点のみを変更の対象とするのではなく、map 全体を変更の対象とする。これは、map 作成時のセンサの認識による色の振り分けが必ずしも正しいとは言えないからである。例えば、map で専有されていると判断されている地点では、偶然人が通っただけで、ロボットが自律走行を行う際には、専有されてない可能性がある。また、逆に実際は障害物に専有されていて、ロボットが通過できないのに、map 上で専有がされていないと判断されていることで、自律走行時に map とセンサからの環境情報とのマッチングが上手くいかないことがある。このため、色が black である地点だけでなく map 全体を変更対象とする必要があるのである。また、このような障害物ごとの状況を色分けによって表現する。つまり、map では、ある地点が専有されていると全て等しく black としていたが、colormap では、ある地点とその周囲の地点がどのような状況にあるかに基づいてカラー表現を行う。塗り分けのアルゴリズムについては以下で詳しく述べる。

図 3 は、colormap への変換対象である二次元配列を表で表したものである。

表の縦軸は地図の高さ（height）、横軸は地図の幅（width）を示している。横軸の 2 段目は、3.2.1 節で述べた、RGB の 3 つの値で 1 つのカラーを表していることを示している。(height, width)=(0, 0) には、ファイルの形式、(heigt, width)=(1, 0) には、ホワイトポイント、(height, width)=(2, 0)、(height, width)=(2, 1) には、それぞれ地図の高さと幅が格納されていて、地図自体のデータは、(height, width)=(3, 0) 以降に格納されている。先述したように、地図全体を見て色付けを行うため、(height, width)=(3, 0) 以降のデータ全てが色変更の対象となる。

周囲の環境から色の決定を行うために、データを 1 つずつ見ていくのではなく、グループにまとめて、そのグループごとに色の変更を行っていく。まず、表の height が 1、width が 1 で構成される 1 マスを 1 ブロックとする。そして、そのブロックを 9 個まとめたものを 1 つのグループと定める。1 つのグループの中で中心になるものを Own とし、その周囲 8 ブロックをそれぞれ、上 (Up)、下 (Down)、左 (Left)、右 (Right) 、斜め (DownLeft, DownRight, UpLeft, UpRight) とする。そして、この 9 ブロックから構成される 1 つのグ

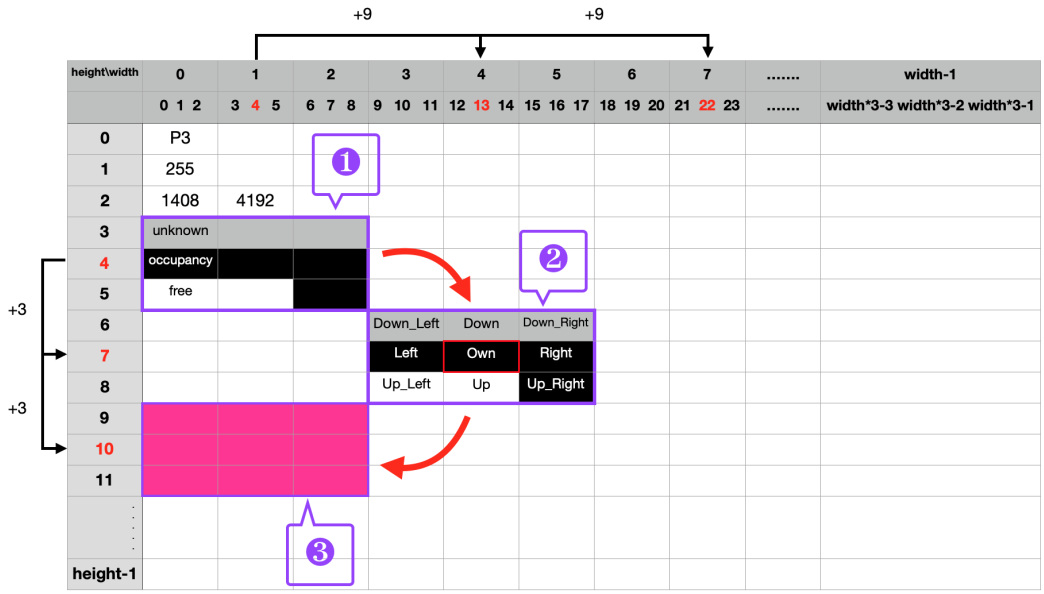

図 3　map から colormap への変換

ループを同じ色に変更する。９ブロックの色を同じ色に一括で変更するのは、周囲の状況から colormap の作成を行う必要があることに加え、１ブロックが地図上では非常に狭い範囲であり、人間が地図を見たときに障害物として見ることが難しくなるからである。

また、図 3 では、map から colormap へと色の変更が行われる具体的な例を示している。この例では、(height, width)=(3, 0) から始まる紫の枠で囲まれた９ブロックで構成されている１つのグループを見る。この時、図ではわかりやすいように紫の枠が移動しているが、実際には表の同じ位置で色の変更が行われているものとする。まず、丸１は、colormap への改造前の white, gray, black で表される map である。そして、丸２では、それが９ブロック、１つのグループであることを示している。最後に丸３では、１つのグループが同じ色に変更されているのがわかる。

次に、red, pink, lightblue, blue のカラーがそれぞれどのような専有を表すかについて述べる。これらのカラーは、専有されている可能性が高い順に red, pink, lightblue,blue とする。例えば、壁や建物といったその地点に存在することが間違いなく、簡単に動く可能性がないものは red、偶然通りかかった人や本来はその地点にないがその時点だけ置いてあったものなど、専有の可能性が低いものは blue、その間の曖昧さが残るものは pink や lightblue とする。これらのカラーの決定は、状態の判断基準となる９ブロックのうちの何ブロックが専有されているか、さらに同じ数だけ専有されていても９ブロックのうちの固まった数ブロックが専有されているのか、あるいは散らばった数ブロックが専有されているのかなど、周囲の状況を調査することによって行う。

3.3　障害物表示の立体化を行うプログラム

2.3.2 節で述べたように、3.2 節で作成した colormap は rviz で表示する。しかし、rviz で colormap を開くと map と同様に white, black, gray のみの平面の地図として表されてしまう。 そこで、rviz 上にマーカーと呼ばれる色付きの物体を置くことができる VisualizationMarker[2] を使用し、colormap のうち一部の色の点

を立体的にを表現することによって、障害物の視認性を上げる。図 4 は、VisualizationMarker によって様々な物体を並べたものである。

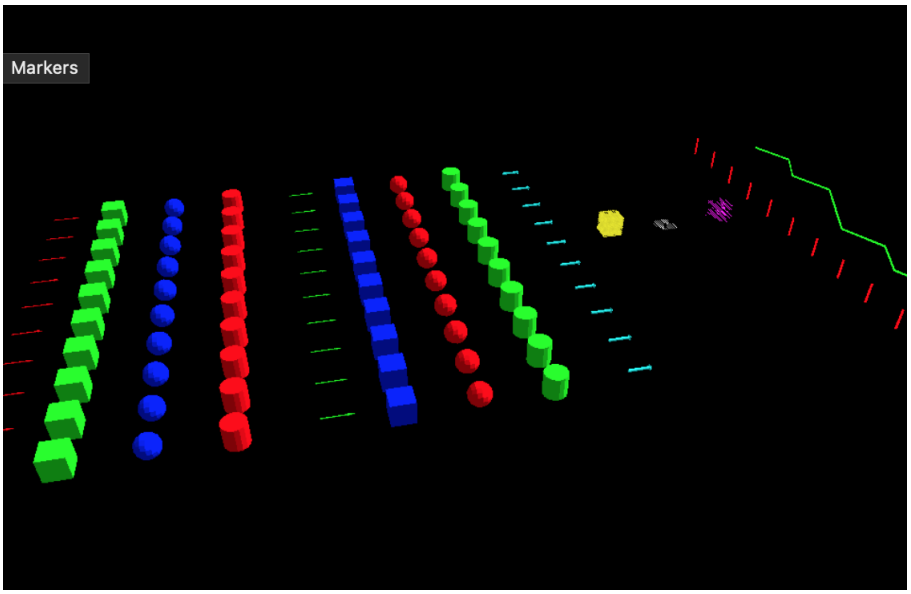

図 4　marker

VisualizationMarker では、図 4 ようにプログラムから簡単にマーカーの形や色、位置などを指定して rviz に表示することができるので、これを用いて colormap の情報をマーカーに渡し、rviz に colormap を出現させる。colormap の情報が記されている ppm ファイルから VisualizationMaker で使用できる形へと変換する手順を 3.3.1 節で述べる。

3.3.1　colormap データをリストに変換

colormap を表す情報として配信されている配列は、地図上の点の横 1 列分が 1 つの配列となり、さらにそれら 1 列ごとの配列をまとめた二次元配列となっている。この二次元配列は、ppm ファイルに書き込まれる際に、1 つの色を表す 3 つの数字が連続したバイトとして書き込まれる。rviz で VisualizationMarker を使用する際には、この ppm ファイルに書き込まれている形では扱いにくいので、Python のリストに変換する。

まず、ppm ファイルのデータをすべて読み込み、そのデータを 1 点ごとに 1 組の RGB データのリストとする。つまり、1 つのリストに 1 つのカラーのデータである 3 つの値が格納されることになる。さらに、ppm ファイルからのデータの最初には、地図データの形式、地図の高さ・幅、地図のホワイトポイントが記されているが、これらのデータはマーカーの作成時には不要なので削除する。

表 3 に、プログラムで使用している、色を表現する二次元配列から ppm ファイルデータ、さらに、colormap リストへの変換を示す。

3.4　rviz に障害物の表示を行うプログラム

本節では、シミュレーション空間でロボットを走行させた際に、ロボットの通過した地点にどのような障害物があったのかという情報を表示するプログラムについて述べる。まず、3.4.1 節では、障害物を表示する仕

表 3　二次元配列→ ppm ファイルデータ→ colormap リストの変換

色付き二次元配列

|-1,-1,-1,　255,0,0,　0,0,0,　-1,-1,-1,　255,0,0 |...

↓

ppm ファイルデータ

-1 -1 -1

255 0 0

0 0 0

-1 -1 -1

255 0 0

...

↓

colormap リスト

|-1,-1,-1 |　|255,0,0 |　|0,0,0 |　|-1,-1,-1 |　|255,0,0 |...

組みについて、3.4.2 節ではどのような情報を表示するかについて述べる。

3.4.1　障害物を表示する仕組み

自律走行を行う際に、環境にはたくさんの障害物が存在し、ロボットはそれらの障害物を避けながら走行を行う。しかし、2.3.1 節で述べたようにセンサの認識の正確性が問題となり、自律走行の失敗が起こることがある。この失敗を防ぐためには、どのような障害物の認識が間違っていて失敗が起こったのかという、失敗の原因を人間が知ることが必要である。そこで、ロボットがある地点を通過した時に、どの障害物が最も近くにあったのかという情報を表示することで、人間が自律走行の失敗の原因を探る糸口になると考える。

どの障害物がロボットの近くにあったのかを知るためには、センサから配信されている情報を使用する必要がある。図 5 はセンサから配信されている情報を示したもので、 /gazebo という ROS トピックに配信されるメッセージのメッセージ型である。

プログラムでは、センサから配信された情報の中で ranges と記されている配列を使用する。この配列には、４００個の要素が格納されている。これは、ラジアンで表されている、*angle*_min（センサのスキャンのスタート地点）から *angle*_max （センサのスキャンのゴール地点）の範囲を４００分割し、分割された地点それぞれからの障害物への距離を表している。

この配列より、センサでスキャンした範囲の中からロボットに最も近い障害物までの距離とその方向を割り出すことができる。最も近い障害物までの距離は、ranges 配列の最小値となる。また、最も近い障害物の方向は図 6 を参考にして計算する。

まず、2.1 節で述べたように、ロボットにはセンサが 2 つある。そして、ロボットの前方にあるセンサを front センサ、後方にあるセンサを back センサとするが、プログラムでは front センサの情報を使用する。front センサの情報を使用するのは、front センサがロボットの前方（進行方向側）に設置されているため、自律走行の失敗の原因となる障害物の検知を行うには有効と考えられるためである。front センサの検知が行われる範囲は図 6 の x 軸より右側の範囲である。

obstacle にロボットから最も近い障害物がある時、*close* はその障害物までの距離、*angle* がその障害物の

```
Header header            # timestamp in the header is the acquisition time of
                         # the first ray in the scan.
                         #
                         # in frame frame_id, angles are measured around
                         # the positive Z axis (counterclockwise, if Z is up)
                         # with zero angle being forward along the x axis

float32 angle_min        # start angle of the scan [rad]
float32 angle_max        # end angle of the scan [rad]
float32 angle_increment  # angular distance between measurements [rad]

float32 time_increment   # time between measurements [seconds] - if your scanner
                         # is moving, this will be used in interpolating position
                         # of 3d points
float32 scan_time        # time between scans [seconds]

float32 range_min        # minimum range value [m]
float32 range_max        # maximum range value [m]

float32[] ranges         # range data [m] (Note: values < range_min or > range_max should be discarded)
float32[] intensities    # intensity data [device-specific units].  If your
                         # device does not provide intensities, please leave
                         # the array empty.
```

図 5　センサに配信されるデータ

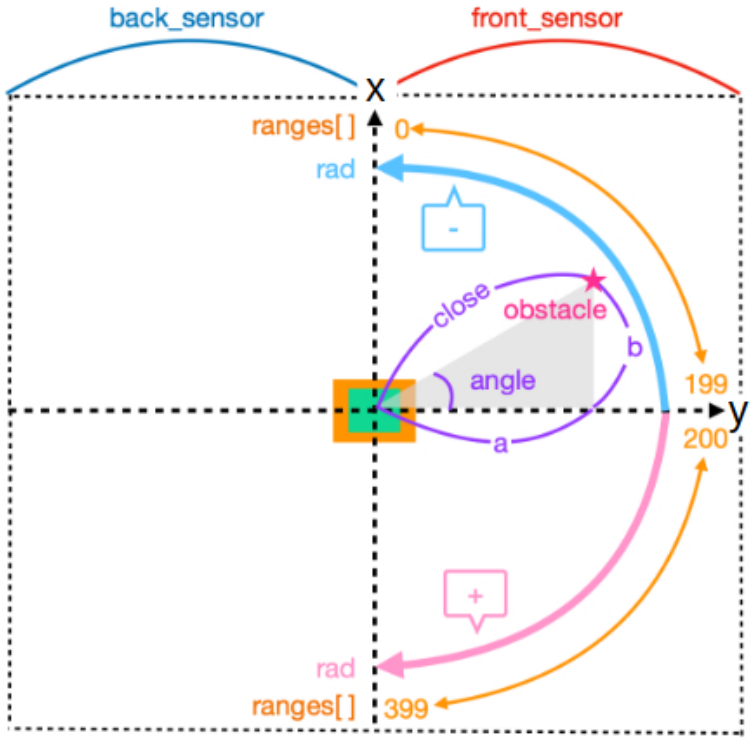

図 6　センサから最も近い障害物の距離と方向（図の右側がロボットの前方）

ある方向を示している。また、*angle* は、ラジアンで表され、y 軸より上の範囲では y 軸を 0 として負の値で、y 軸より下の範囲では、y 軸を 0 として正の値で表される。まず、*angle* の値を求めるには、*close* が ranges 配列の何個目の要素に当たるかを数え、その要素が y 軸より上の範囲か、または、下の範囲に入っているのかを調べる。それぞれの y 軸の上下の範囲がそれぞれ 9 0 度ずつの範囲であるので、その要素が 9 0 度のうちの何度を表すかを求め、その値をラジアンに直すことで *angle* の値となる。*close* と *angle* の値を求めること

で、図 6 の a と b の値を以下の式を用いて求めることができる。

$$a = close \times \cos angle, b = close \times \sin angle$$

ロボットの自己位置は、シミュレーション環境上の座標で表されており、その座標にそれぞれ a と b の値を足すことで、ロボットから最も近い障害物の座標を割り出すことができる。この座標を用いて rviz 上でロボットから最も近い障害物がある位置に obstacle という表示をしている。また、このプログラムでは obstacle と表示される文字の色や大きさを変化させることで、ロボットから障害物への距離や最も近い障害物であると何度も判定されていることなどを示す。これらの詳しい振り分けについては、4.2 節で述べる。

3.4.2 障害物の情報

障害物に関する情報は、シミュレーション空間を再現している rviz での表示に加えて、ターミナルにも出力する。図 7 は、ターミナル出力に障害物の情報が表示されたものである。

```
Find obstacle!!
Coution Front

<Robot info>
 id:4
 position: x 1.0708923004107844
           y 0.06787559440193151

<Distance info>
 ahead:2.1240851879119873
 close:0.10000000149011612
 angle:0.40035000000000004
 number:51
 sin:0.38974068979809656
 cos:0.9209246411708745

<Marker info>
 position: x  1.1629847659001564
           y  0.10684966396250006
```

図 7　ターミナルへの出力

まず、⟨ Robot info ⟩ では、障害物ごとに振り分けられた番号とその時点のロボットの現在地が示されている。そして、⟨ Distance info ⟩ では、ahead がロボットの正面にある障害物までの距離、*close* が最も近い障害物までの距離、*angle* が最も近い障害物の存在する方向、number が最も近い障害物が ranges 配列の何個目に当たるか、sin が sin(*angle*)、cos が cos(*angle*) をそれぞれ表している。最後に ⟨ Marker info ⟩ は、rviz に obstacle と表示される位置である。

4　実験

1 節でも述べたように、本研究では、実際の環境で自律走行を行う前に失敗が起こりそうな箇所を発見し、事前に対策できることを目指す。そこで、実験では、実際の環境から作成した地図を使用し、また、3.4 節で述べたプログラムでは、その環境をモデルとしたシミュレーション空間を用いて実験を行う。4.1 節では、実験で使用する地図を作成し、シミュレーション空間のモデルとした中之島チャレンジについて、4.2 節では、シミュレーション空間の構築方法やシミュレーション空間で走行させるロボットの設定について、4.3 節では、実験結果について述べる。

4.1　中之島チャレンジについて

中之島チャレンジは、２０１８年より自律走行ロボットの実環境実験として行われている。本研究では、２０２２年度に行われた中之島チャレンジで作成された、中央公会堂及び中之島図書館を周回するコースの地図を使用する。このコースは、全長４７７ m の屋外のコースであり、建物などの大きな障害物が目印となりやすい一方で、木や花壇などのセンサで認識するのが難しい障害物が多く存在する。また、公道で多くの人が行き交っているので、人を避けながら自律走行を行う必要がある。

図 8 に中之島公園内のコース、図 9 に中之島公園内のコースから作成した地図を示す。図 9 の右下の地点が図 8 のスタートとゴールの地点に当たる。

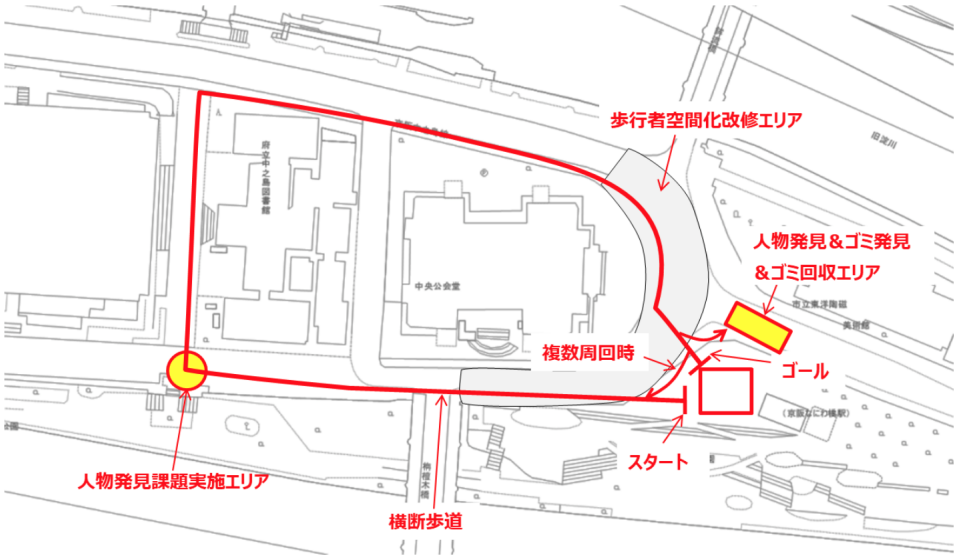

図 8　中之島公園内のコース

4.2　シミュレーション空間の構築

シミュレーション空間は、gazebo[6] と rviz によって構築する。gazebo は、ROS によって動作するシミュレーション空間で、そこでシミュレーションされた結果は、rviz で視覚的に確認することが可能である。

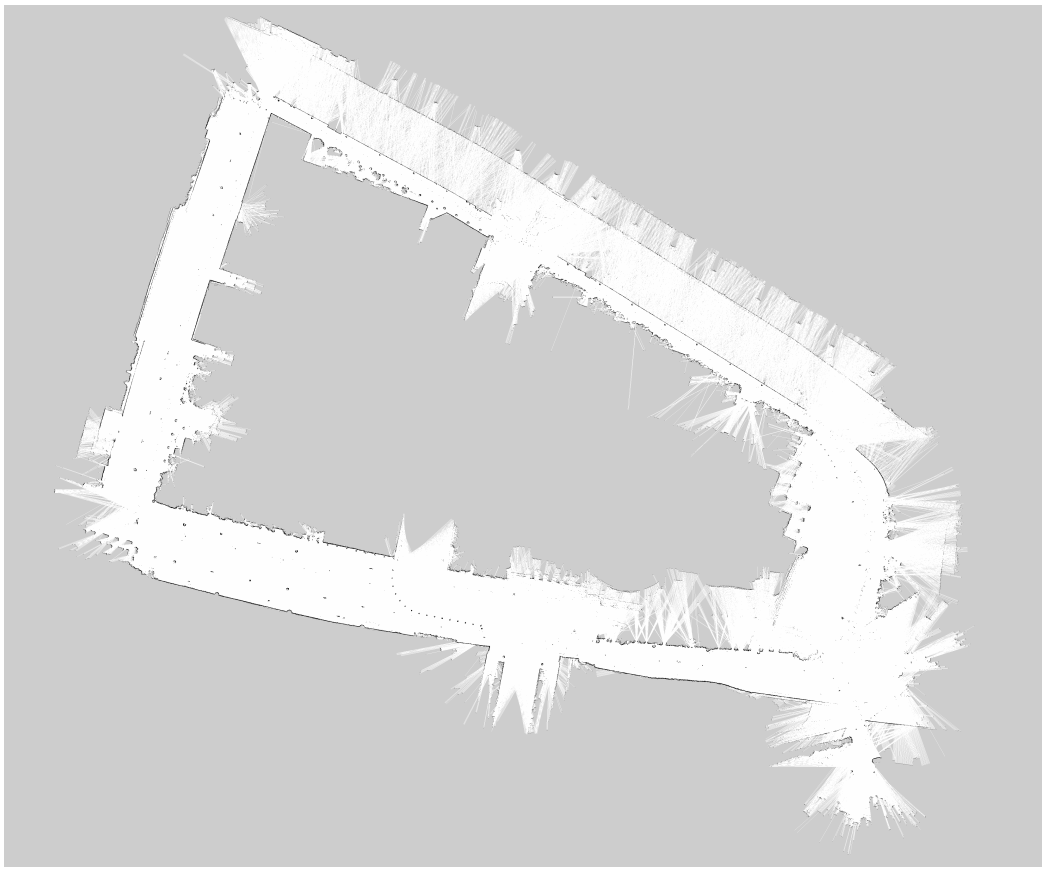

図 9　中之島公園内のコースから作成した地図

gazebo で、シミュレーション空間を作成し、ロボットを用いた実験を行うには、表 4 の 6 ステップが必要である。

表 4　シミュレーション環境の構築ステップ

Step1	URDF[5] でロボットモデルを作成	ロボットの構造を URDF で記述する 色や形、部品の繋がりなどである関節を設定する
Step2	稼働部位の設定	ロボット部品ごとの可動域、ポーズなどの設定
Step3	衝突属性と物理属性の追加	衝突属性で衝突に関する設定やセーフゾーンなどの定義を行う 物理属性で慣性や接触係数などを定義
Step4	Xacro によるロボットモデルの作成	URDF のサイズを縮小したり、保守する 定数、数式、マクロを作成できる
Step5	ロボットモデルのシミュレーション	ロボットを移動させるコントローラーの設定 ロボットのホイールを定義 ROS と gazebo のプラグイン
Step6	地図の設定	地図を gazebo で表示できるよう sdf 形式 [1] に変換

図 10 は、以上の 6 つのステップを実行し、作成した gazebo のシミュレーション空間である。地図を作成

した際に設定したスタート地点には、ロボットがセットされている。

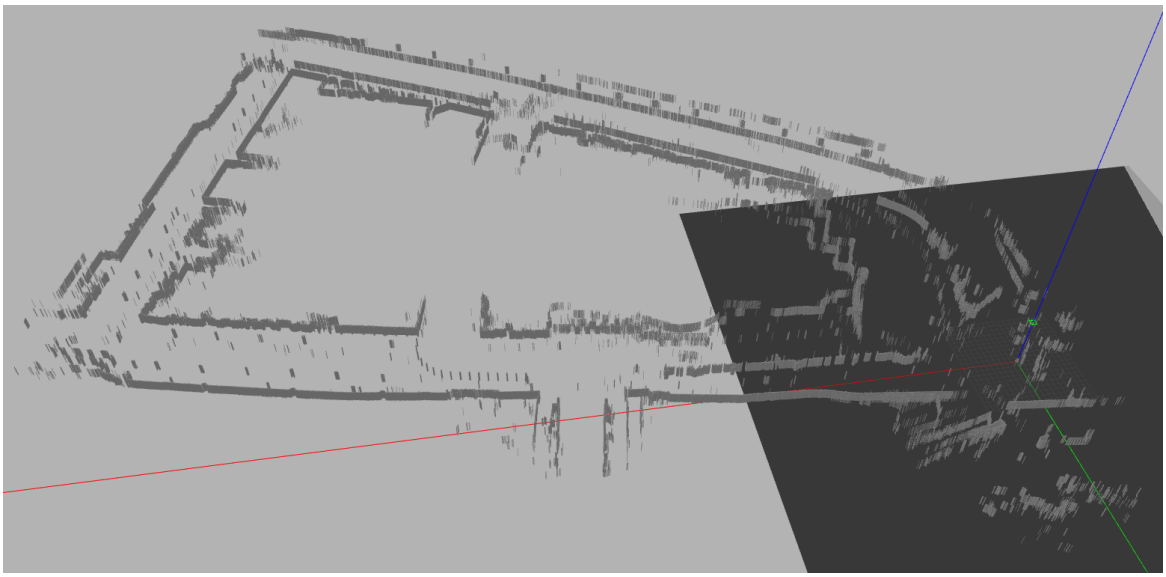

図 10　中之島公園内のコースの地図から作成したシミュレーション空間

4.3　実験結果

4.3.1 節に 3.2 節で述べたプログラム、4.3.2 節に 3.3 節で述べたプログラム、4.3.3 節に 3.4 節で述べたプログラムの実験結果と考察をまとめる。

4.3.1　実験１

■実験結果　図 11 に、図 9 で示した中之島公園内のコースから作成した、colormap を示す。また、図 12 に図 11 の赤い四角で囲まれた部分を拡大したものを示す。

作成した colormap を見てみると、map では、black で表されていた障害物を表す地点が、3.2.2 節で述べた振り分けに従って、色が変更されているのがわかる。地図の中心部分は、unknown の判定になっているが、これは図 8 で示したように建物がある箇所である。この建物が立っている箇所に沿った部分では、色は主に red となっていてロボットが通過できない可能性が高いことを示していることがわかる。また、このコースでは、ポールや木などが多く存在しているが、それは white に囲まれている地点で pink や lightblue になっている地点で表されている。さらに、地図全体に見られる光の反射によってできる細い線上の先では、blue が使われている。

以上より、地図に色をつけることによって、map からは読み取ることのできなかった情報を人間が知ることができるようになった。このプログラムは特に実際に自律走行を行う環境を見たことがない場合に有効であると言える。なぜなら、実際の環境を見れなくても colormap を見ることによって、どのような障害物であるかを人間が想像することが可能であり、そこから自律走行に向けての対策を立てることができるからである。

■考察　中之島チャレンジの際には、色をつける前の従来の地図で実験を行った。そのときに自律走行をさせる前の段階で、ロボットがどのルートを取ることができるかを人間が地図を見て検討するのだが、従来の地図

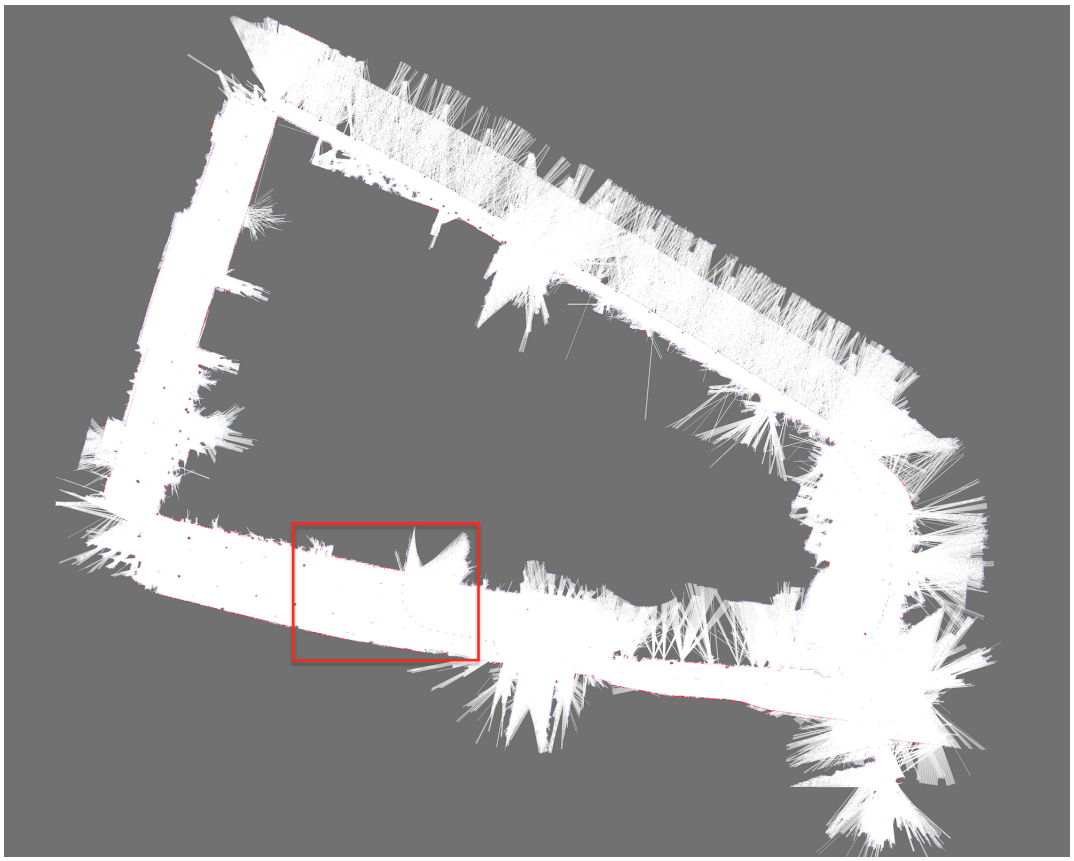

図 11　中之島公園内のコースの地図から作成した colormap

を見ても black で表されている障害物が人間であるのか、木であるのかという判別が困難であった。そこで、もう一度人間がコースを見て歩くということを行わなければならなかった。しかし、本研究で作成した地図を用いることによって、障害物がどのようなものであるかといったヒントを人間が得ることが可能である。これにより、自律走行までに費やす時間を大幅に削減することができる。

4.3.2　実験 2

■実験結果　本節では、まず、図 11 の colormap を立体化し、rviz に表示することを試した。しかし、中之島公園内のコースは距離が長くデータ量が多いため、立体化を行うと使用している PC の処理能力を超えることがわかった。そこで、中之島公園内のコースより短いコースの地図を用いて、rviz に colormap を立体化した地図を作成する。このコースは中之島チャレンジのエクストラチャレンジである、中之島エクストラチャレンジの中で ATC（アジア太平洋トレードセンター）で作成されたものである。図 13 の左側は、ATC で作成した map から作成した colormap、図 13 の右側は、ATC の colormap から作成された立体化した地図を rviz に表示したものである。

colormap を rviz へ立体的に表示したものを見てみると、colormap が見やすくなっており、さらに map を作成した実際の環境を人間が理解しやすくなっている。また、rviz で表示することができるので、シミュレーションを行う際にも使用することができ、より実際の環境に近づけて実験を行うことができる。また、処理能力の超過には、マップを適宜分割して走行することで対応できる。

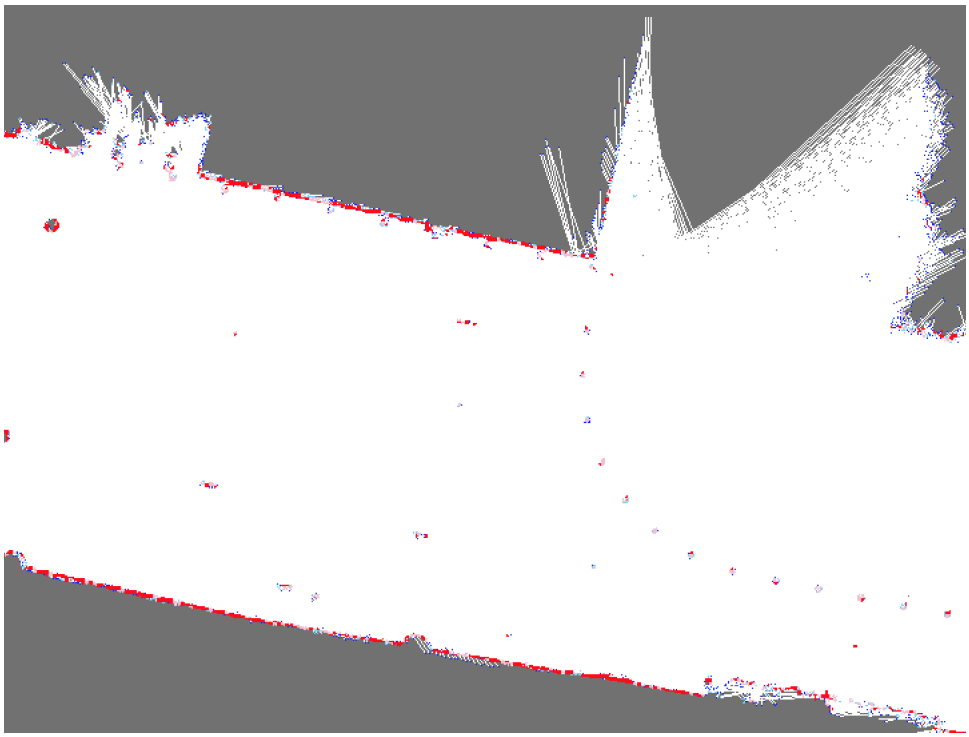

図 12　中之島公園内のコースの地図から作成した colormap の一部

■考察　従来では、rviz には平面の地図しか表示することができなかったが、立体でかつ障害物の特徴ごとに色ついた地図を表示できるようになったことで、4.3.1 節の考察で述べたのと同様に人間が地図を見て、その地図上の障害物などの情報を理解しやすくすることができた。

また、従来のシミュレーションでは、rviz に表示される地図が平面であるために、gazebo でシミュレーションしたものが rviz でも同様に再現されているのかを判断するのに手間がかかっていた。しかし立体化の結果、図 13 を見てもわかるように、特徴が強調されていることで地図の形がつかみやすく、gazebo との照合も容易になった。これにより、人間が従来のシミュレーションで行っていた gazebo と rviz を比べるという作業の手間を省くことができる。

4.3.3　実験３

■実験結果　図 14 は、シミュレーション空間内で、ロボットを走行させ、通過した地点で最も近かった障害物を表示したものである。(地図のサイズが大きく全体を表示すると、わかりにくいので一部を抜粋している。)

また、図 15 は図 14 の赤い四角で囲まれた部分を拡大したものである。

このプログラムでは、ロボットから最も近い障害物を示す際に、ピンク、紫、黒の順にその障害物とロボットの距離が近いというように、**obstacle** の文字の色を変えている。また、表示されている obstacle の大きさが異なるのは、文字が大きいほど何度も最も近い障害物であると判定されたことを示している。また、:obstacle の前にはそれぞれ数字が振られているが、これは obstacle を識別するための番号である。数字が大きいほど、ロボットが直近で発見した障害物となる。そして数字が連続した番号でないのは、何度も同じ障害物がロボットから最も近い障害物となった際には、同じ障害物に振り分けられた番号のなかで最も大きい数字のみが表示

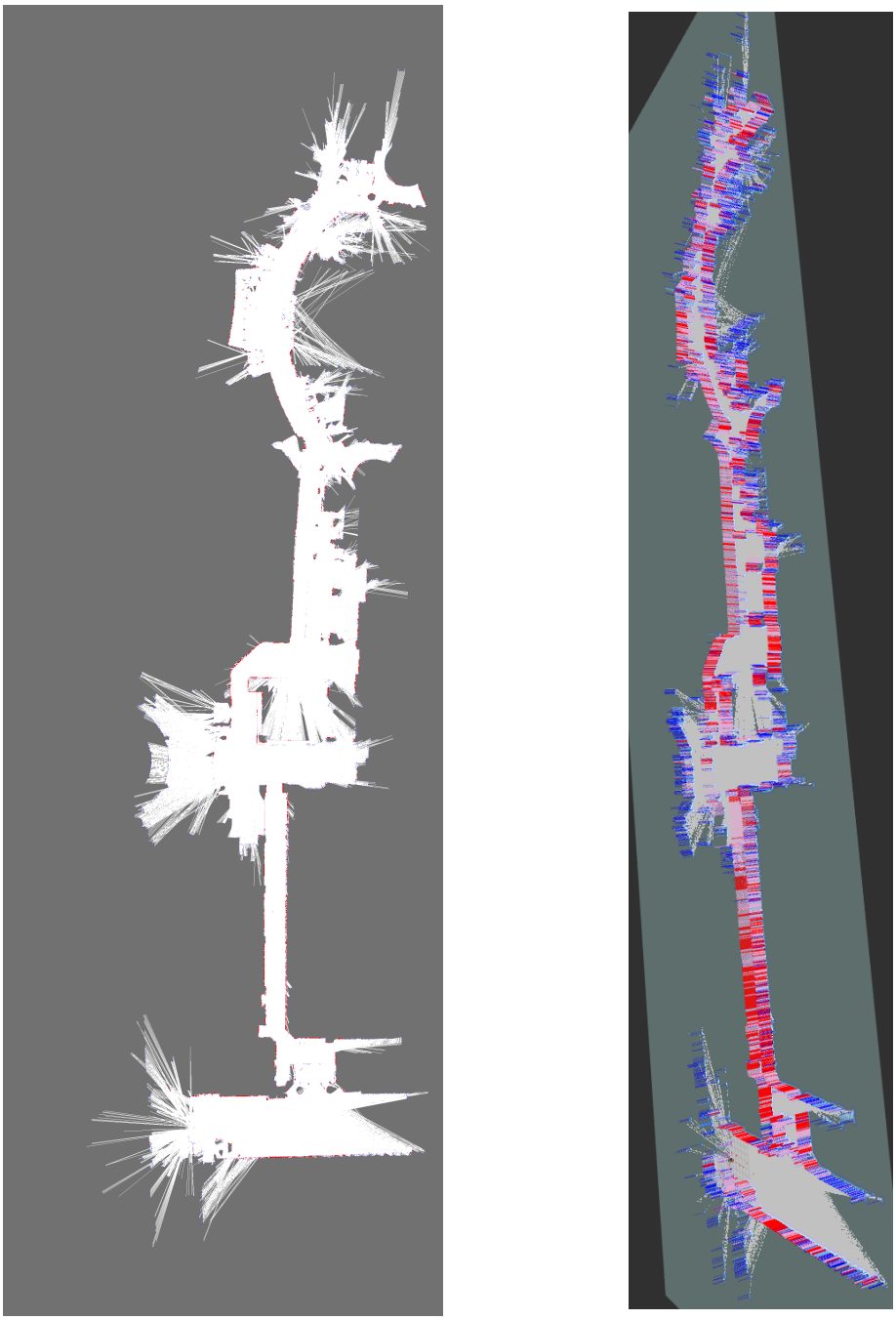

図 13　ATC の map から作成した colormap（左）と colormap を立体化して rviz に表示したもの（右）

されるためである。この数字をターミナルに表示されている、⟨ Robot info ⟩ の id の番号と照合させることでその障害物の情報を簡単に見つけることが可能である。

図 14 と図 15 を見てみると、図 14 の一番左にはピンク色の文字で 0:obstacle と書かれており、さらにそのまわりの赤い四角までの領域では、黒の obstacle がいくつかあるだけで、特に注意するべき障害物がないことがわかる。そして少し進むと、図 15 の赤い四角内で拡大されているようにピンクや紫で示されている障害

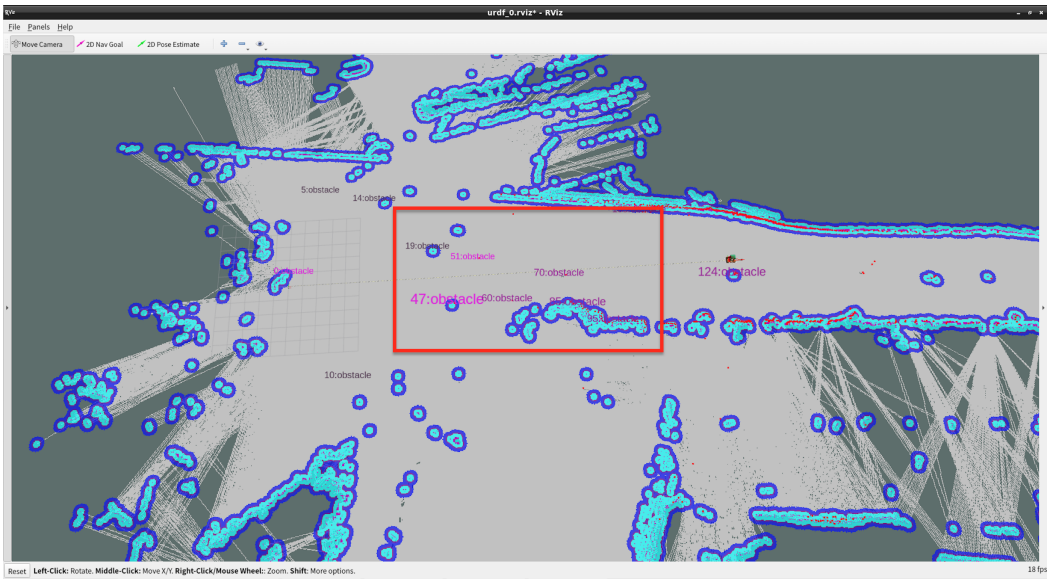

図 14　rviz への表示

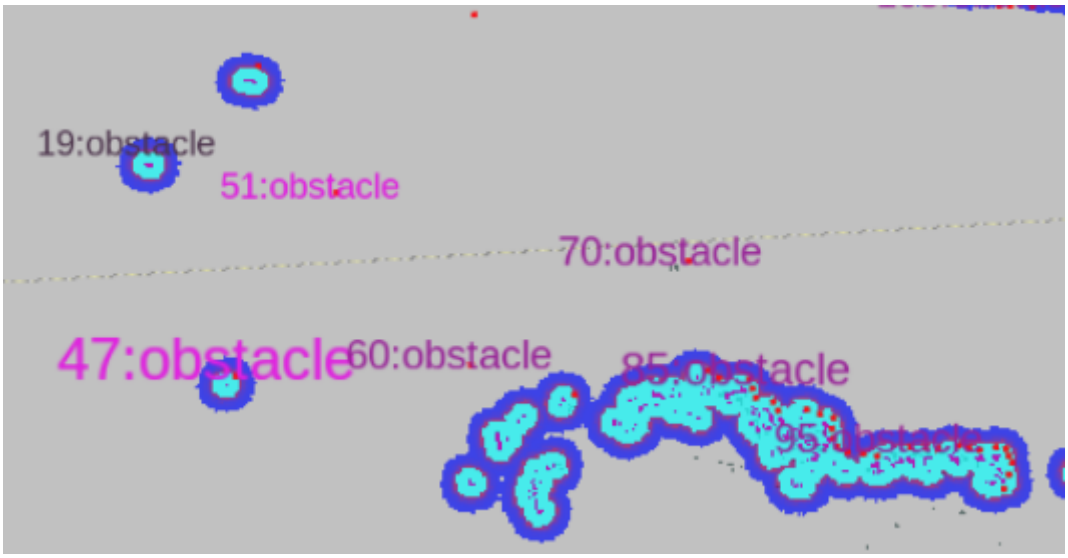

図 15　図 14 の rviz への表示から一部拡大したもの

物がいくつか存在し、特に 47:obstacle は、ロボットが何度も最も近い障害物と判定しており、注意が必要である。

■考察　以上のように、ロボットを動かしながら人間が自律走行を行わせたい環境がどのようなものであるかを知ることができるようになった。また、地図上に情報が記録されているので、あとから見返すことも可能となった。これは、実際の環境で自律走行を行い、失敗が起こったらその対策を考え、また最初から走行させ、また失敗が起こったら対策して試すといった従来の実験方法の手間と時間を短縮できるものであると考えられる。

そして、実際の環境から作成した地図とシミュレーション空間を用いることで、よりリアルな自律走行を試

すことができ、そのシミュレーションの過程でロボットがどの障害物を見ているのかを明らかにすることで、実際の環境で自律走行を行った際に、どの障害物に注意させてロボットを走行させればよいかを人間が事前にわかっているので、走行させる前に自律走行の失敗の可能性に対する対策を立てることが可能である。

4.3.4 実験のまとめ

1 節で述べたように、２０２１年に我々は初めてロボットを自律走行させる実験を行い、かつそれは未知の環境であったが、これが図 8 で示した中之島チャレンジのコースであった。この初めての実験で課題となったのは、作成した地図を見ても特徴がわかりにくく、実際にコースを何度も歩いてコースの確認を行ったり、自律走行をロボットに行わせるためのセッティングに時間を取られて、自律走行を何度も試すことができなかったことであった。

4.3.1 節では、従来の地図から特徴をわかりやすい地図を作成できたことで、未知の環境であっても、実際に何度もコースを確認することなくルートの作成などを行うことができるようになった。次に 4.3.2 節では、中之島のコースの障害物などを rviz で立体的に表すことはできなかったが、ATC のコースで試したことで、より実際の環境をわかりやすく表示でき、自律走行させたい環境を人間が知ることが可能になった。また、4.3.3 では、シミュレーション空間で自律走行させることができたので、実際の環境で自律走行を試す機会が少なくても、シミュレーション空間で何度も試すことができ、２０２１年の実験で我々が感じた課題の１つである、自律走行を試す機会の少なさを解決できた。

以上より、本研究では、特にロボットの実験を始めたばかりの人や、実験したことのない未知の環境で行う自律走行であっても実験が進められることを目指して研究を行い、それを達成することができた。

5 終わりに

5.1 まとめ

本研究では、自律型システムの行動の根拠を明らかにすることを目標とし、自律型システムの具体例として自律走行型ロボットを挙げてプログラムの開発と実験を行った。2.3 節で述べたように、自律走行型ロボットにおける課題は、センサによる認識の過程で誤った認識が起こることにより、作成した地図と実際の環境にズレが発生し、そのズレによって、自律走行時に失敗が起こることである。この問題を解決するために、本研究では、実際に自律走行を行う前に人間が失敗の起こりそうな地点を発見したり、失敗の起こりそうな地点に実際に自律走行を行う前に対策を立てれるように３つのプログラムを作成した。

まず、１つ目の map を colormap に改造するプログラムでは、従来の map では、障害物が存在するのか、しないのか、また未判定の地点であるのかという情報しか得られなかったが、colormap では、それらに加えて、どのような障害物であるかという情報を人間が知ることが可能な地図の作成を行うことができた。

２つ目の colormap を rviz に立体的に表示するプログラムでは、地図を立体することでより実際の環境に近づけた地図を獲得でき、シミュレーションを行う際の再現度を挙げることが期待できる。

また、３つ目のシミュレーション空間においてロボットを走行させ、ロボットの通過した地点で最も近い障害物を表示するプログラムでは、本来であれば実際の環境で自律走行を行わなければわからない、失敗の原因となる可能性のある障害物を可視化することができた。このプログラムにより、今後さらなる実験の効率化が図れると考える。

5.2 今後の課題について

今後の課題として、以下の 3 点が挙げられる。

まず、colormap を作成する際の色の振り分け方についてである。現在は、black から色分けを行った 4 色に加え、white と gray の 6 色の地図としているが、black から振り分ける色を増やしたり、色の振り分け方の条件を変えたりすることで、さらに人間にとって情報を得られる地図を作成できると考えられる。

次に、colormap を rviz に表示する際に、使用している Marker の種類を増やすことが挙げられる。現在は Marker の中でも、line という 1 種類の Marker のみを使用しているが、図 4 で示したように、Marker には様々な種類がある。これらの他の Marker を使用することで、さらに実際の環境に近いシミュレーション空間を再現することでシミュレーションの精度も上がることが期待される。

そして、最後にシミュレーション空間でロボットを走行させた際に、obstacle 以外の情報を rviz に表示することである。現在は、obstacle を表示する色や大きさで障害物の情報を示していたが、さらにどのような形状や性質の障害物であるかといった具体的な情報を示すことで、より簡単に人間が失敗の起こりそうな地点の発見やその地点に対する対策を立てることが可能であると考える。

6 謝辞

本研究を進めるにあたり、最後まで熱心にご指導いただいた、新出尚之准教授、並びに、ロボットの提供で多大なるご協力をいただいた北陽電機の皆様に、深く感謝の意を表します。

付録 A 研究業績

合同エージェントワークショップ&シンポジウム 2023(JAWS2023), 自律型システムの推論過程の可視化, 木下美咲

参考文献

[1] Shiloh Curtis. map2gazebo. https://github.com/shilohc/map2gazebo.

[2] Open Source Robotics Foundation. Marker. http://wiki.ros.org/rviz/DisplayTypes/Marker.

[3] Open Source Robotics Foundation. ROS. https://www.ros.org.

[4] Open Source Robotics Foundation. rviz. http://wiki.ros.org/ja/rviz.

[5] Open Source Robotics Foundation. urdf. http://wiki.ros.org/ja/urdf.

[6] GazeboSim.org. gazebo. https://gazebosim.org/home.

[7] 中之島ロボットチャレンジ実行委員会. Nakanoshima Robot Challenge. https://www.nakanoshima-rc.jp.

[8] 北陽電機株式会社. 測域センサ URM-40LC/LCN-E. https://www.hokuyo-aut.co.jp/search/single.php?serial=189.