2015 年度 卒業論文

ROSによる自律ロボットの目標地点への到達行動の実現について

奈良女子大学 理学部 情報科学科 4 回生 新出研究室
12251619 兼松明未

平成 28 年 2 月

目 次

概 要

実世界において自律的に目的を達成するロボットの実現が望まれている中で、我々はこのようなロボットの実現を目指した研究課題に取り組んでいる。本研究ではロボット開発のフレームワークである ROS を用いて、自律ロボットの能力の一環として目標物を認識してそこに到達する行動の実装を行った。

1 はじめに

1.1 研究背景

近年自律ロボットの普及が進んでいる。自律ロボットとは、動的に変化する環境下でも自らの知能で判断し、計画を立て、行動することにより目的を果たすことができるロボットである。人間の指示を介さずに行動することができるため、自動お掃除ロボットなどの一般家電からコミュニケーションロボット、災害救助ロボット、宇宙開発と、様々な分野で活躍している。

我々は、このような自律ロボットの研究を行っている。そうしたロボットは一般的に目標達成のために様々な汎用的な能力を要することが多く、そうした能力の一環として目標物を認識してそこに到達する行動がある。そこで本研究ではそうした行動の実現を目指した。

そうした能力の実現においては、複雑化するロボットの機能を制御するために、ロボット用フレームワークである ROS[1] を用いて統合的な制御を行うことが適していると考えられる。そのため目標物の認識と到達行動をともに ROS で制御する。なお、前者には OpenCV を用いて、後者には先行研究において構築された NXT Python+ライブラリ [2] を用いる。

本研究は、奈良女子大学理学部 4 回生柿原との共同研究である。ROS によるカメラからの環境情報の取得については [3] で述べられており、本論文では目標地点に到達するために行ったカメラによる物体追跡とロボットの目標物への到達行動の実現について述べる。

1.2 ROS

ROS(Robot Operating System) とは、ロボット用ソフトウェアフレームワークの 1 つである。複雑なロボットのふるまいを簡単にプログラミングするために開発され、幅広いロボット用プラットフォームを備えている。ROS の機能にはハードウェア抽象化やメッセージ通信、パッケージ管理、可視化ツールなどがあり、これらはオープンソースで提供されている。世界中の開発者が作成したソフトウェアがフリーで公開されており、これを利用することもできる。

1.2.1 メッセージ通信

ROS ではノード同士がトピックを介してメッセージを配信・購読することで通信を行う。ROS におけるノードとは実行プログラムのことで、トピックとは特定のメッセージの受け渡しを行うバスである。またメッセージとはデータ型のことで、各メッセージ型には受け渡す情報のデータ構造が定義されている。ROS で定義されているメッセージ型の他に、ユーザーが固有のメッセージ型を作成することも可能で、本研究でも自作のメッセージ型の定義を行った。ノードは Publisher（配信者）と Subscriber（購読者）を用いてメッセージ通信を行うが、このとき Publisher と Subscriber は互いに同じメッセージ型を送受信しなければならない。1 つのノードに複数の Publisher と Subscriber を作成することもできる。

図 1: トピックを介したメッセージ通信

1.2.2 rviz

ロボット開発において重要なのは、ソフトウェアの状態を視覚化することである。ROS は強力な可視化ツールを備えており、中でも rviz と呼ばれる 3D 可視化ツールは非常に優れたものである。これを用いることでロボットの位置・姿勢などの状態の可視化や、カメラ画像の表示を行うことが可能で、シミュレータとしても機能する (図 2)[4]。今回は使用しなかったが、今後複雑になるロボットの行動を管理する際にこのようなツールが有用であると考えられる。

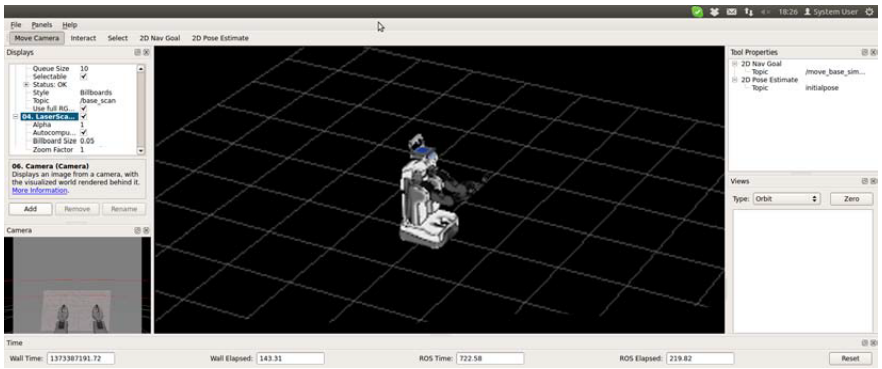

図 2: rviz による可視化 (PR2 ロボット)

2　NXT と NXT Python+

2.1　LEGO Mindstorms NXT

本研究で使用したロボットは、LEGO 社から販売されている教育用 LEGO Mindstorms NXT(以下 NXT)[5] である。NXT は比較的安価で制御が容易である上に、高性能なセンサーやモーターなどを搭載しており、実験に適したロボットである。

付属の LEGO ブロックを組み合わせて自在にロボットを成形することが可能であり、本研究では図 3 のように、組み立てたブロックのフロント部分にカメラとして Xtion PRO LIVE(以下 Xtion)[6] を搭載した。

図 3: Xtion を搭載した NXT

NXT と PC の接続には USB または Bluetooth による通信を用いることが可能である。本研究では Bluetooth によってワイヤレスで制御プログラムを転送することにより NXT を動作させた。また、Xtion と PC の接続には USB 接続を用いている。

2.1.1 構成部品

本研究で使用するロボットを構成する主な部品について述べる。なお、図4〜6はLEGO Shop[7]より引用した。

図 4: インテリジェントブロック

- インテリジェントブロック
  NXTの頭脳となる最も重要な部品。上部の3つの出力ポートにはサーボモーターを、下部の4つの入力ポートにはセンサーを接続する。本体上で簡単なプログラミングを実行することも可能だが、USB2.0とBluetoothを搭載しており、本研究では前述の通り、Bluetooth通信によるPC上でのより複雑な制御を行った。

図 5: サーボモーター

- サーボモーター
  タイヤを取り付けて回転を行うことで、NXTを移動させる行動源となる。モーターのパワーや動作時間をプログラミングで指定し、移動スピードや移動距離を調節することができる。

図 6: コンパスセンサー

- コンパスセンサー
  方角を検出するセンサー。今回の実験には用いていないが、ロボットの行動制御を正確に行うために、今後必要であると思われるため搭載した。

図 7: Xtion

- Xtion
  NXTの付属の部品ではないが、視覚情報の獲得のために用いたカメラである。ASUS社から販売されているRGB-Dカメラで、PC用のモーションキャプチャデバイスである。

2.2 NXT Python+

NXT の行動の実装には、Python 言語によるライブラリを用いた。NXT を制御する既存のライブラリにはいくつか種類があるが、Python を用いたものには NXT Python ライブラリ [8] がある。しかし、これは下位レベルの命令を実装したライブラリであるため、前進や回転など行為レベルでの命令を与えることが困難であった。複雑な命令をまとめてプログラミングを簡単にするために、NXT Python をもとにしたより上位の行為を実装した NXT Python+ライブラリが先行研究にて構築されており、我々はこれを用いた。

2.2.1 設計概要

NXT Python+は図 8 のような設計となっている。特に①〜③の項目について以下で述べる。

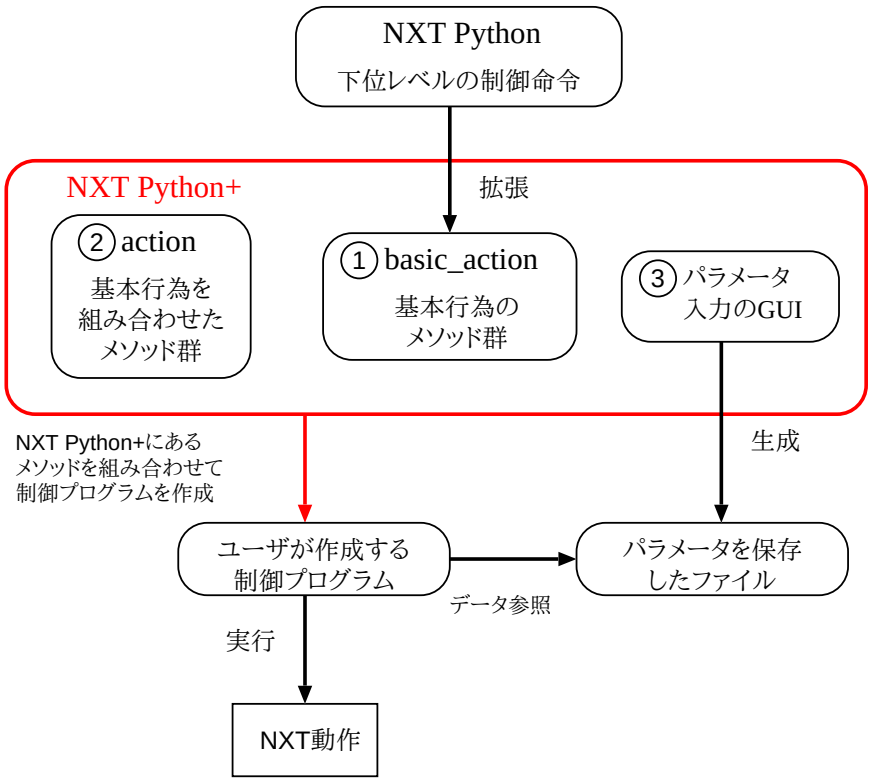

図 8: NXT Python+

① basic_action
下位レベルの制御命令を持つ NXT Python ライブラリ上に、直進や回転などの基本行為のメソッド群を実装したものである。NXT の起動から Bluetooth 通信に至るまで、プログラム実行までに必要な処理もまとめられている。

② action
basic_action の基本行為の命令を組み合わせた、さらに上位の行為のメソッド群を実装したものである。

③ パラメータ入力の GUI
メソッドに必要なパラメータの入力を容易に行うために構築された。必要なパラメータがそろった GUI をユーザーが選んで使用することができる。

2.2.2 クラスとメソッド

本研究では basic_action.py ファイル内に定義されている以下のクラスとメソッドを使用した。

Basic_Action クラス

— __init__(host)
Bluetooth 通信で NXT と PC を接続し、モーターやセンサーの設定を行うメソッド。引数の host には、NXT の MAC アドレスを指定する。

— move(p, m)
直進のためのメソッド。p にはモーターのパワー、m には一定距離の倍率を指定する。

— turn_right(p, m)
右回転のためのメソッド。move(p, m) と同様に引数を指定する。右側のモーターは前転させ、左側のモーターは後転させる。

— turn_left(p, m)
左回転のためのメソッド。turn_right(p, m) と同様であるが、モーターの回転方向を逆にする。

— stop()
モーターのパワーを 0 にして停止するメソッド。NXT と PC の通信は切断しない。

— close()
NXT と PC の通信を切断するメソッド。

3　実装

3.1　実装方針

人間がその目で見て目的の場所に到達するという行動をとる時どのようなプロセスをたどるだろうか。自律ロボットにもそれと同様のプロセスが必要となるため、まずこれについて考える。はじめに行うのは目標の決定である。次にそれがどこにあるのか探索を行う。このとき周囲を見渡して周りの環境を把握するだろう。そしてその中から目標を認識・発見し、そこに向かって移動する。この一連の行動によって目的を達成することができる。人間にとっては単純な行為だが、ロボットが自律的に行うには幾つもの課題がある。

我々は、上記のプロセスに沿って目標物に到達する行動の実装を試みた。これにはまずカメラでとらえた画像の中から目標とするものを特定し、次に特定した点に対して物体追跡の手法を用いて追跡しながらロボットが自ら動くという手法を取る。目標物の特定としては、人間がマウスで指定する方法と、コンピュータが自動検出する方法が考えられる。コンピュータが自動検出する方法の 1 つとしては、OpenCV のテンプレートマッチングを用いる方法が考えられる。この方法をとることも可能ではあるが、テンプレートマッチングは物体の大きさや見える角度が変わるとうまく検出できないため、物体との距離や位置関係が変わりうる状況ではうまくいかない。そこで今回は人間がマウスで指定することで目標を特定する。

3.2　ノード構成

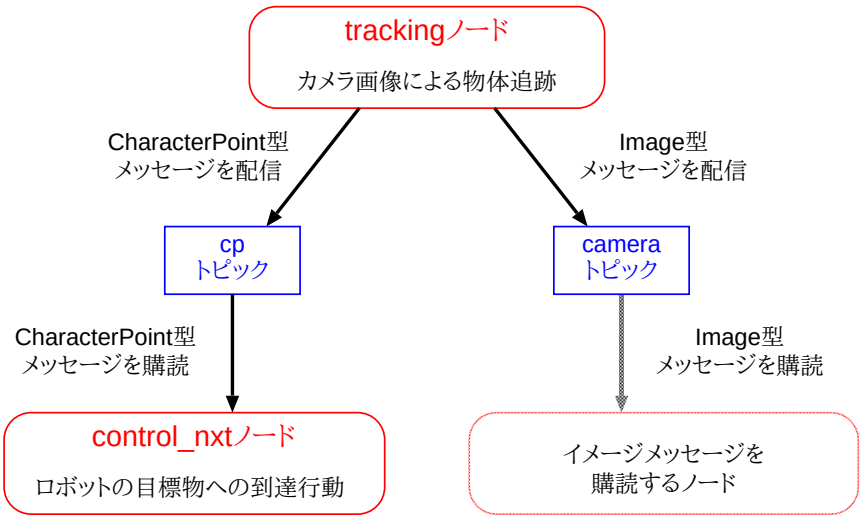

図 9: ノード構成

3.3 tracking ノード

ロボットの動きには誤差があり、目標物に向かってまっすぐ動ける保証はないため、物体追跡を用いて目標物を見失わないようにする必要がある。そこで、カメラ画像による物体追跡を行う tracking ノードを作成した。このノードでは 2 つの Publisher を作成し、それぞれ CharacterPoint 型メッセージと Image 型メッセージを配信している。実装は以下の手順で行っており、このうち 1 と 3 のカメラの処理については [3] で述べられている。

1. Xtion カメラからの画像取得
2. 物体追跡と CharacterPoint 型メッセージの配信

   物体追跡には本研究では OpenCV を用いて指定した点を追跡する方法を用いた。まずウィンドウに表示したカメラ画像から、目標物をマウスクリックで特定し、このときの点を初期座標とする。次に、この点の座標位置とウィンドウのサイズについての情報を持った自作の CharacterPoint 型メッセージをトピックに配信し、点の情報を受け渡す。なお、このときのトピック名は cp トピックとした。そして、移動した物体の動きを検出するために、前後のフレームを比較して、点のオプティカルフローを計算する。これには画像ピラミッド型 Lucas-Kanade 法（3.3.1 節）を実装した OpenCV の関数 (cv2.calcOpticalFlowPyrLK 関数) を用いた。さらに、現在のフレームの点の座標を新しい初期座標として更新する。これを反復して繰り返すことによって、継続的に物体を追跡し続けることが可能となった。
3. 追跡画像を Image 型メッセージとして配信

3.3.1 画像ピラミッド型 Lucas-Kanade 法

Lucas-Kanade 法 (以下 LK 法) とは、物体の特徴的な点を抽出して追跡を行うアルゴリズムで、オプティカルフローの計算方法の一つである。差分の 2 乗により誤差を計算する SSD を用いて前後フレーム間の誤差を計算し、その相違度から追跡物の位置を計算する。計算量が少ないため高速な処理が可能であるが、追跡物が素早い移動をする場合に計算誤差が生じやすい。また、追跡物に特徴的な点が見られない場合に物体を見失うことがあるという欠点がある (図 10)。

画像ピラミッド型 LK 法では、画像をいくつかのレベルの解像度の集まりとするピラミッド構造を用いる。低解像度の画像から順に LK 法を実行し、その値をより高解像度の画像の処理の初期座標として用いることで計算量が少なくなり、高精度の追跡が可能となる。追跡物の素早い移動にも対応することができる (図 11)。

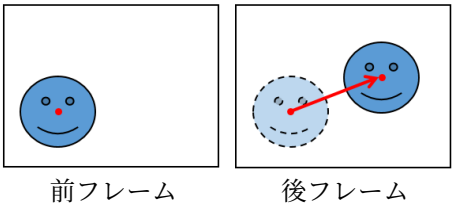

図 10: LK 法

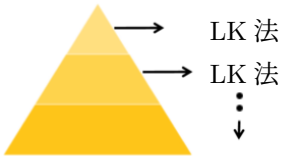

図 11: 画像ピラミッド型 LK 法

3.4 control_nxt ノード

次に、ロボットの目標物への到達行動を行う control_nxt ノードを作成した。実装は以下の手順で行った。

1. NXT と PC の Bluetooth 通信による接続
2. CharacterPoint 型メッセージの購読

   cp トピックから CharacterPoint 型メッセージを購読し、目標物上の点の座標位置とウィンドウのサイズについての情報を取得する。
3. ロボットを移動

   取得した点の x 座標の位置によって場合分けを行い、直進と回転行為によりロボットを目標物に向かわせた。

   - 画面の左側 (0〜 40%)：右回転する
   - 画面の中央 (41〜 60%)：前進する
   - 画面の右側 (61〜100%)：左回転する

これらを反復することで少しずつ目標に近づき、目標物に到達することが可能となった。

3.5 実行結果

実行画面を図 12 に示す。左側は目標物の視点からロボットの動きを撮影した画像で、右側はロボットに搭載したカメラでとらえた画像である。今回追跡する目標物は、図上の赤で囲った黒いオブジェクトとし、目標物に対してロボットが左方向を向いているところから実行を開始した。

まずマウス操作により上段の図のように目標物を特定する点を打つ。この場合は画面の右側に点があるため、ロボットは左回りに回転を行い、中段の図のようにロボットが目標物の正面を向く。最後に、ロボットは目標物に向かって直進し、下段の図のようにロボットは目標物に到達する。

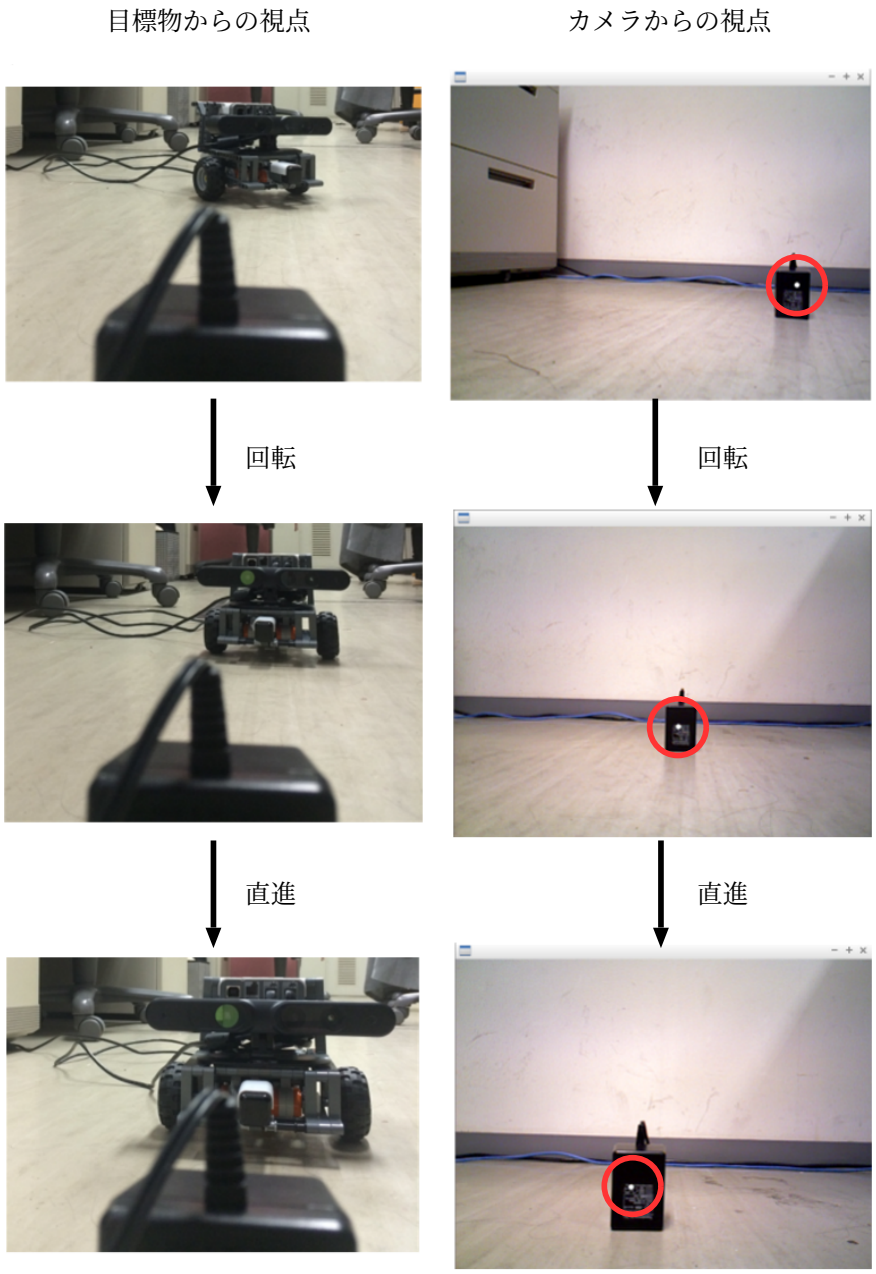

図 12: 実行画面

4　まとめ

ロボットの行動制御にカメラ画像を使用したことで、 ロボットが自律的に目標物に向かって動くことが可能となった。今後の課題を挙げる。物体追跡については、本研究では目標物の決定をマウスクリックで行ったが、将来的にはもっと適切な方法で自動検出をすることが考えられる。そして、本研究で行った物体追跡の手法では追跡する点を見失うことがあるなど精度に問題があったが、これは容易に差し替え可能な仕様にしたため、より高精度な物体認識手法に変更することで、より正確な物体追跡を行うことが課題である。ロボットの行動制御については、今回は Xtion の深度センサーを利用しなかったが、これを利用して目標物との距離の計測精度を上げることで、より正確な回転角度を算出することが期待できる。そして、将来的に、今回獲得した能力を自律エージェントのプランの一部として組み込み、自律的に目的を達成するロボットの開発全体を ROS を用いて行うことが望まれる。

5　謝辞

本論文の執筆および研究にあたり、いつも親身にご指導をしていただい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] ROS.org. http://www.ros.org/.

[2] 小島侑子. 小型ロボット制御のための汎用ライブラリの構築. 2011 年度修士論文, 奈良女子大学大学院人間文化研究科, 2012.

[3] 柿原怜奈. 自律ロボットの実現に向けた ROS による環境情報の取得. 2015 年度卒業論文, 奈良女子大学理学部情報科学科, 2016.

[4] Aaron Martinez and Enrique Fernandez. *Learning ROS for Robotics Programming*. Packt Publishing, 2013.

[5] Lego Group. レゴマインドストーム公式サイト. http://www.legoeducation.jp/mindstorms/index.html.

[6] Xtion PRO LIVE. https://www.asus.com/jp/3D-Sensor/Xtion_PRO_LIVE/.

[7] LEGO Shop. http://shop.lego.com/en-IT/.

[8] Douglas P Lau. NXT Python. http://home.comcast.net/~dplau/nxt_python/.

[9] 桑井博之, 豊沢聡, 永田雅人. 実践 OpenCV 2.4 for Python 映像処理&解析. カットシステム, 2014.