2011 年度 修士論文

小型ロボット制御のための
汎用ライブラリの構築

奈良女子大学 大学院人間文化研究科
博士前期課程 情報科学専攻
10840024 小島侑子

平成 24 年 1 月

概要

本論文では、LEGO MINDSTORMS NXT(以下、NXT) という小型で安価なロボットを使って、上位レベルの記述で NXT を制御できるライブラリ、NXT Python+ を構築したことについて述べる。

近年、比較的安価で、ある程度の性能が保障されているロボットが市場に広がってきており、特に NXT はロボット制御の研究などにも用いられている。例えば、目標を持ったロボットが、その達成のために行動を決定するような研究に用いる場合、ある程度上位レベルの、例えば一定量移動する、向きを変える、などの動作を基本行為として記述できるライブラリがあることが望まれるが、現状では不十分なものしか見つからなかった。

そこで、今後そうした研究を NXT を使って行うことを容易にするために、既存の NXT Python という NXT 制御ライブラリを基に、上位レベルの記述で NXT を制御できるライブラリ、NXT Python+ を構築した。また、NXT Python+ のライブラリとしての有用性を検証する実験も行った。

目次

1　はじめに

1.1　研究の目的と背景

近年、ロボットの開発が進み、比較的安価でその性能がある程度保障されているロボットが市場に広がってきている。特に、LEGO 社が販売している LEGO MINDSTORMS NXT（以下、NXT）という小型ロボットは、現在、教育現場や個人で使用されることもあり、さらにはロボット競技会 WRO(World Robot Olympiad)[4] や RoboCup[5] のようなサッカー競技などにも NXT での参加が見られるなど、ロボットの行動制御に関する研究にも用いられている。しかし、NXT をそのような目的に使う場合、中でも、目標を持ったロボットがその達成のために行動を決定するような研究に用いる場合、ある程度上位レベルの、例えば一定量移動する、向きを変える、などの動作を基本行為として記述できるライブラリがあることが望ましい。だが、現状での NXT を制御する環境を調べたところ、1.3 節に述べるように、限られた環境のもとでしか動作しなかったり、モーターの回転数を直接指定するなど下位レベルの制御命令を基にしたライブラリなど、現状では不十分なものしか見つからなかった。

そこで本研究では、上位レベルの記述で NXT を制御できるライブラリ、NXT Python+ を構築し、さらに、このライブラリを用いて NXT を動作させて実験を行うことにより、ライブラリの評価を行う。

1.2　LEGO MINDSTORMS NXT

本研究で使用した小型ロボットは、LEGO 社が 2006 年に発売した教育用 LEGO MINDSTORMS NXT[6](図 1) である。NXT にはサーボモーターを接続するポートが 3 個、そしてタッチセンサや超音波センサなどのセンサを接続するポートが 4 個あり、どのポートにも自由に接続ができる。また、LEGO ブロックを使って様々な形のロボットに組み立てることも可能で、本研究では図 1 のように組み立て、ライブラリ検証の実験を行った。

NXT を動かすには、PC から USB による通信か、Bluetooth による通信で制御プログラムを転送し、実行する。実験では、Bluetooth による通信で NXT を動作させた。

図 1　LEGO MINDSTORMS NXT

1.3 NXT を制御する現状のライブラリ

現在、NXT を制御する開発環境やプログラミング言語には、次のようなものがある。

1. GUI による制御プログラムの開発環境
   - NXT-SW
     NXT ソフトウェア (以下、NXT-SW) は、プログラミングブロックと呼ばれるアイコンをマウスで操作し、画面上で組み立てることで制御プログラムを作成することが出来るソフトウェアである。命令を入力する必要が無いため、ブロックを組み立てるような遊び感覚でプログラムを作成することが可能である。また、視覚的にプログラムを作成するため、プログラミングに慣れていない人でも容易に制御することができる。しかし、複雑で長いプログラムを書くほど、プログラム全体の設計が視覚的に大きくなってしまい、見にくくなってしまう。また、NXT-SW のような GUI のソフトウェアは OS に依存し、その多くが有償である。
2. プログラミング言語
   - NXC
     Not eXactly C(以下、NXC)[2] は John Hansen 氏によって開発された言語で、C 言語に非常によく似た文法を持ち、OS に依存しないフリーのプログラミング言語である。特に Windows 上で使用する場合には、Bricx Command Center(以下、BricxCC)[1] という無料の開発環境があり、プログラムを記述する上で、BricxCC が NXC の予約語を太字や色付けなど、装飾して表示する。この色付けにより、事前にバグを修正しやすくしている。しかし、“前進する”というようなプログラムを書く場合、NXT-SW では 1 つのブロックだけで表現できるのに対し、NXC では 3 行の命令を必要とする。同じ動作をするプログラムを比較すると、NXC の方が記述が多くなる。また、モーターの回転数などを直接制御しなければならず、下位レベルの制御命令を基にしたライブラリであると言える。

   - NXT Python
     NXT Python[7] は、Python というオブジェクト指向スクリプト言語を使って NXT を制御するライブラリで、モーターの回転数を直接制御するなど下位レベルの制御命令を基にしている。また、OS に依存しないフリーのものである。

本研究では、プログラミングに慣れていない人に対して、スライダやボタンなどの GUI による入力操作が行えるようにしたいので、GUI 設計が容易に行いやすい、Python 言語でのライブラリ作成が適切であると考えられる。そこでこの NXT Python を使用して、より上位レベルでの制御機能や GUI を提供するライブラリ、NXT Python+ を構築することにした。

2　NXT Python+ について

本章では、2.1 節で NXT Python+ の設計について、2.2 節で NXT Python+ のライブラリにどのような機能があるかを述べる。

2.1　設計概要

上位レベルの記述で NXT を制御できるライブラリ、NXT Python+ を設計するにあたり、本研究では特に以下の 4 点に主眼を置いた。

1. メソッドを簡略化し、プログラムの記述量を減少させる
2. モーターの回転数を直接指定するなど下位レベルの制御命令によらない、一定距離の直進や回転といった上位レベルでの制御命令を可能とする
3. 簡略化された API を持つメソッドを組み合わせ、障害物回避といった複雑なプログラムを容易に作成できるようにする
4. GUI によるメソッドに必要なパラメータの入力

次に、NXT Python+ 全体の概要を図 2 に示す。

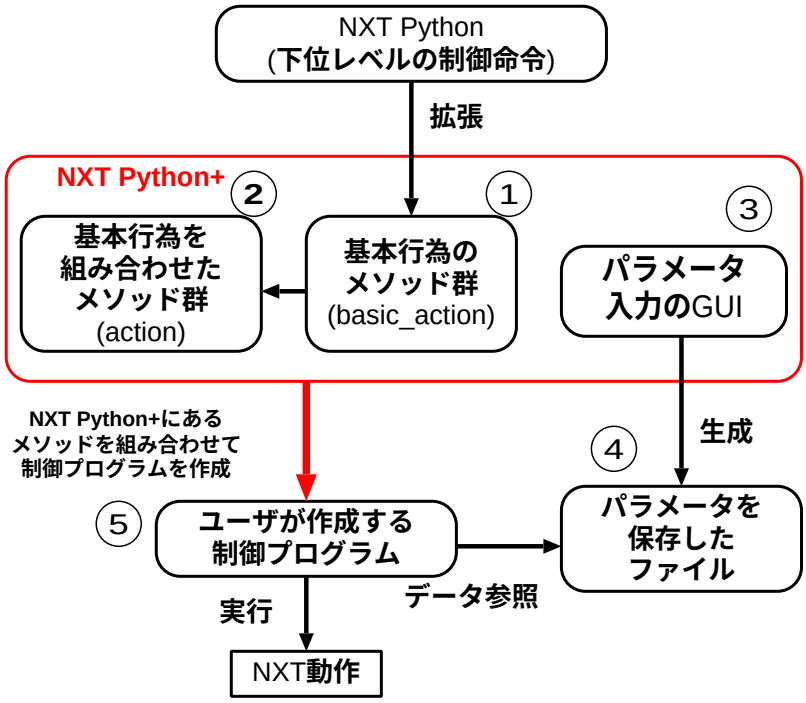

図 2　NXT Python+ の概要

設計の流れとして、まず、NXT Python にある下位レベルの制御命令、例えば、モーターのポートやパワーを指定するといった命令を組み合わせ、直進や回転といった基本行為のメソッド群を作成した (図 2 の①)。さ

らにその基本行為のメソッド群を組み合わせ、例えば、“衝突を検知したら停止する” や、“ライントレースをする” といった、よく利用されると考えられる行動のメソッドを作成した (図 2 の②)。基本行為のメソッド群の組み合わせ方次第で、先述以外の様々な行動制御も可能である。

次に、作成したメソッド内で呼び出される低レベルルーチンが用いるパラメータの調整を、GUI を用いて行えるようにした (図 2 の③)。NXT の個体差や電池の残量により、モーターの回転数やスピードに僅かながら誤差が生じることがあり、例えば電池の残量が少なくなると、モーターを回転させるパワーが落ちることがある。そのため、NXT を起動するたびにモーター等の調整を行う方が、より正確な動作を行う。そこで、少しでも容易にパラメータの調整ができるように、また、プログラミング知識が無い人でも容易に制御プログラムを作成できるように、スライダやボタンでパラメータを入力できるように設計した。

GUI で調整したパラメータは一旦データファイルに保存され (図 2 の④)、新たに制御プログラムを作成した際は、そのデータファイル上のパラメータを参照するようにした。これにより、パラメータを変更するたびに制御プログラムのファイルを開いて修正するのではなく、GUI のスライダを用いて容易に調整できるようになった。

以上が設計の流れである。制御プログラムを作成する際は、パラメータ調整の後、NXT Python+ に用意したメソッドを組み合わせて、ユーザ側が作成する (図 2 の⑤)。

なお、本研究で使用した Python のバージョンは Python2.4.4、そして Python で GUI を構築、操作する際には、Tkinter[3] という標準モジュールを使用している。

次の 2.2 節では、NXT Python+ の機能についてさらに詳しく述べる。

2.2 機能

構築した NXT Python+ には、

- パラメータの入力を容易にする GUI(図 2 の③)
- basic_action ファイル (図 2 の①)
- action ファイル (図 2 の②)

が用意されている。上記 3 つの項目について述べる。

2.2.1 GUI

メソッドに必要なパラメータの入力をより容易に行うため、図 3 のような GUI を構築した。

図 3 の GUI は、直進する際のモーターのパワーと走行時間を調整できる。調整して GUI 画面の一番下にある OK ボタンを押すと、入力した値のパワーで入力した値の時間だけ直進する。この他にも、一定の距離を直進する際のモーターのパワーを調整する GUI、左右のモーターを別途に調整し、かつ走行時間も調整可能な GUI もあり、必要なパラメータが揃った GUI をユーザが選んで使用できるようになっている。

図 3 の GUI を操作する手順について述べる。まず、センサ毎にどのポートに接続するかを選択する。センサを使用しない場合は、“no” を選ぶ。センサは 0〜4 個まで使用でき、どのセンサをどのポートに接続しても問題はない。

次に、NXT を動かす際に欠かせないモーターのポートを選択するが、本研究ではモーターを 2 個使用することを想定してメソッドを作成したので、ポート選択は可能なものの、デフォルトで左側のモーターを Port

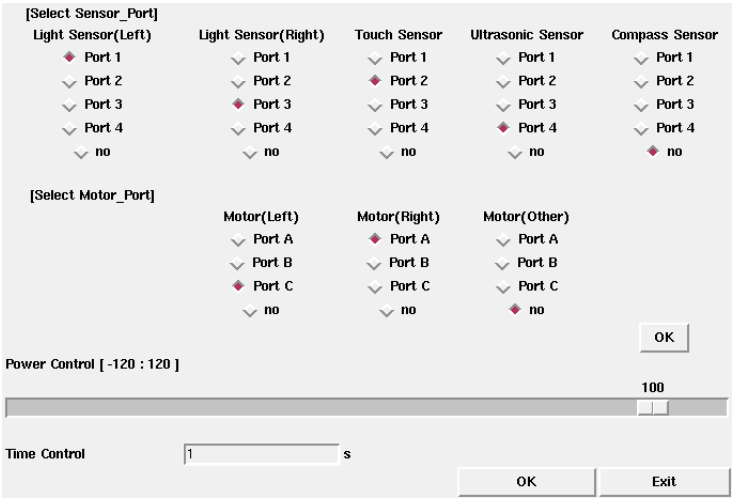

図 3　GUI

C に、右側のモーターを Port A に接続しているものとし、3 個めのモーターは使用しないという前提になっている。なお、モーターを 2 個使用していれば、ポートの組み合わせは自由に変更できる。(Port A と Port B、Port B と Port C などの組み合わせも可能。)

ポートの選択を終えたら、すぐ右下にある OK ボタン (GUI 画面の中央右端) を押す。これで NXT と PC が Bluetooth で接続され、接続できた場合は “Connect” と表示する。なお、この接続は GUI を起動する毎に 1 回だけ可能となっている。

NXT と PC が接続できたら、次はモーターのパワーと走行時間の調整を行う。モーターのパワーはスライダを使い、−120 から 120 までの調整が可能で、正の値だと前進、負の値だと後進する。また走行時間は、例えば 3.1.1 節の例題のように、1 秒間直進する場合では、時間の項目に “1” と入力する。

パラメータの入力を終えたら、GUI 画面の一番下にある OK ボタンを押す。すると、1 秒間 NXT が直進する。動作を確認し、再度モーターのパワーや時間を調整したい場合には、その都度パラメータの入力を行い、OK ボタンを押すだけでよい。

調整し終えたら、Exit ボタンを押す。これによって GUI 画面が終了し、パラメータを保存したデータファイルが生成された状態になる。

2.2.2　basic_action

下位レベルの制御命令をまとめた NXT Python をもとに、NXT との接続を行ったり、モーターやセンサの使用準備に必要な命令をメソッドにまとめたファイルである。NXT Python の上位レベルの機能を提供するファイルにあたる。NXT を起動し、Bluetooth で通信させ、制御プログラムを実行するまでに必要な命令がすべてまとめられている。

以下に、本ファイルで定義されているクラスおよびメソッドを示す。

- Basic_Action クラス
  - __init__(host)

    Bluetooth で PC と NXT を接続し、左右のモーターの設定、センサの設定を行う init メソッド

(コンストラクタ)。引数の host には、NXT の MAC アドレスを指定する。

- set_power(left_power, right_power)
  モーターのパワーを設定するメソッド。left_power は左モーターのパワーを示す。
- move(p, m)
  直進するメソッド。p はモーターのパワー、m は一定距離の倍率を示す。
- rotate(l, r, m)
  パワーの細かい調整ができる回転のメソッド。l は左モーターのパワー、r は右モーターのパワー、m は一定距離の倍率を示す。
- turn_right(p, m)
  rotate(self, p, -p, m) の引数を減らし、右回転用に作成したメソッド。
- turn_left(p, m)
  rotate(self, -p, p, m) の引数を減らし、左回転用に作成したメソッド。
- stop()
  モーターのパワーを 0 にして停止するメソッド。PC と NXT の通信は切断しない。
- close()
  PC と NXT の通信を切断するメソッド。
- rate(x, m)
  一定距離の倍率を変更するメソッド。
- touch_or()
  タッチセンサが検知したかどうかを返すメソッド。返り値は True か False。
- light_value(s)
  ライトセンサが検出した値を返すメソッド。引数 s に 'right' を指定すると右側のライトセンサの値が、'left' を指定すると左側のライトセンサの値が返る。
- ultrasonic_value()
  超音波センサが検出した値を返すメソッド。
- compass_value()
  コンパスセンサが検出した値を返すメソッド。

- power_data(), time_data(), left_power_data(), right_power_data(), time_data_rotate()
  パラメータを保存したファイルからデータを読み取って返すメソッド。

2.2.3 action

basic_action クラスで定義した基本行為の命令を組み合わせ、新たに作成したメソッドを集めたファイルである。このファイルの中でメソッドを定義することで、さらにライブラリを拡張することができる。basic_action ファイルの上位レベルの機能を提供するファイルにあたる。ユーザが制御プログラムを書く際には、action ファイルで定義されているメソッドを呼び出すだけでよい。

以下に、本ファイルで定義されているメソッドを示す。

- move_rotate()
  GUI で調整したモーターのパワーと走行時間の通りに動作するメソッド。
- forward()

GUI で調整したモーターのパワーと走行時間の通りに直進するメソッド。

- turn_right_90()
  GUI で調整したモーターのパワーと走行時間を使用して、右に 90 度回転するメソッド。
- turn_left_90()
  GUI で調整したモーターのパワーと走行時間を使用して、左に 90 度回転するメソッド。
- turn_right_angle(angle)
  引数 angle で指定した角度の方向に、右に回転するメソッド。正確に angle 度回転する保証はなく、あくまでもその方向に向かせるだけである。
- turn_left_angle(angle)
  引数 angle で指定した角度の方向に、左に回転するメソッド。正確に angle 度回転する保証はなく、あくまでもその方向に向かせるだけである。
- move_until_collision()
  タッチセンサが、物体に衝突するのを検知するまで前進するメソッド。衝突したら停止するが、PC との通信は接続したままである。
- linetrace()
  キーボードからのコマンド入力によって、方向転換や停止ができるメソッド。ラインが交差したとき、右折、左折、直進、停止が指示できる。また一定の間隔で、ライントレースを続けるか停止するかの表示が出る。ラインが交差したとき以外で停止をさせるには、この表示で停止を指示する。なお、このメソッドを使う際は、ライトセンサが NXT の左右に 2 個装備していることを前提とする。
- linetrace_forward()
  ラインが交差する箇所まで前進するメソッド。ラインの交差をライトセンサが検知すると停止する。なお、このメソッドを使う際は、ライトセンサが NXT の左右に 2 個装備していることを前提とする。

3　NXT Python+ の有用性の検証

本章では、設計した NXT Python+ のライブラリとしての有用性を検証するため、例題を取り上げて評価を行う。3.1 節では基本的な行動について、メソッドの簡略化によって記述量が減少したかどうか、3.2 節ではそれらを組み合わせた動作の上位レベルでの制御について検証し、3.3 節で評価を、3.4 節で考察を行う。

3.1　メソッドの簡略化による記述量の減少

本節では例題を 3 題取り上げ、記述量の減少について評価をする。NXT Python+ を用いて制御プログラムを作成した場合と、NXC で作成した場合とで比較した。なお、NXC でのプログラミングの際は [8] を参考にした。

3.1.1　例題 1: 1 秒間直進する

“1 秒間、前に直進する” という動作について、それぞれの言語でプログラムを記述すると、図 4 のようになった。点線で挟まれた部分が、“1 秒間、前に直進する” という命令になり、その記述量を比較すると、NXC では 3 行必要なのに対し、NXT Python+ では 1 行で記述できた。なお、点線で挟まれていない部分は、プログラムを記述する際に必ず必要となる記述なので、今回の検証では対象外とする。

(NXC)

```
task main(){
-----------------------------------------------
    // この部分に命令を記述
    OnFwd(OUT_AC, 100);
    Wait(1000); // 1000 = 1s
    Off(OUT_AC);
-----------------------------------------------
}
```

(NXT Python+)

```
#!/usr/bin/python
# -*- coding: euc-jp -*

import basic_action

m = basic_action.Basic_Action('00:16:53:0D:E5:9F')
-------------------------------------------------
# この部分に命令を記述
m.forward()
-------------------------------------------------
m.close()
```

図 4　1 秒間直進するプログラム (NXC と NXT Python+)

3.1.2　例題 2: 90 度右に回転する

“90 度右に回転する” という動作について、それぞれの言語でプログラムを記述すると、図 5 のようになった。点線で挟まれた部分が、“90 度右に回転する” という命令になり、その記述量を比較すると、NXC では 4 行必要なのに対し、NXT Python+ では 1 行で記述できた。なお、点線で挟まれていない部分は、プログラムを記述する際に必ず必要となる記述なので、今回の検証では対象外とする。

(NXC)

```
task main(){
-------------------------------------------
    # この部分に命令を記述
    OnFwd(OUT_C, 75);
    OnRev(OUT_A, 75);
    Wait(300);
    Off(OUT_AC);
-------------------------------------------
}
```

(NXT Python+)

```
#!/usr/bin/python
# -*- coding: euc-jp -*

import basic_action

m = basic_action.Basic_Action('00:16:53:0D:E5:9F')
-----------------------------------------------------
# この部分に命令を記述
m.turn_right_90()
-----------------------------------------------------
m.close()
```

図 5　90 度右に回転するプログラム (NXC と NXT Python+)

3.1.3　例題 3: ライントレース

図 6 のようなライン上で NXT がライントレースし、ラインが交差する箇所で停止するという動作について、それぞれの言語でプログラムを記述すると、図 7 のようになった。点線で挟まれた部分が、“NXT がラインの交差する箇所までライントレースする” という命令になり、その記述量を比較すると、NXC では 17 行の命令が必要なのに対し、NXT Python+ では 1 行で記述できた。

図 7 のライントレースのプログラムでは、左右のライトセンサが白を検知した場合はラインが直線である (ライトセンサがラインを挟んでいる状態) と判断し、右のライトセンサだけが黒を検知した場合はラインが右に曲がっていると判断、左右両方が黒を検知した場合は交差点であると判断するようになっている。

なお、点線で挟まれていない部分は、プログラムを記述する際に必ず必要となる記述なので、今回の検証では対象外とする。

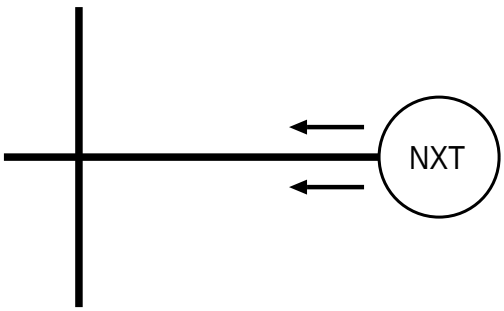

図 6　実験図

3.2　上位レベルでの制御

本節では例題を 2 題取り上げ、上位レベルでの制御について評価する。本質的な行動列に自然に対応する命令列で、行動を制御できているかどうかを、実験を通して評価を行う。

(NXC)

```
#define TH_LIGHT 40
#define POW 50
#define TIME 1

task main(){
------------------------------------------------------
    # この部分に命令を記述
    SetSensorLight(IN_1);
    SetSensorLight(IN_3);

    while(true){
        if((Sensor(IN_1) > TH_LIGHT) && (Sensor(IN_3) > TH_LIGHT)){
            OnFwd(OUT_AC, POW);
            Wait(TIME);
        } else if((Sensor(IN_1) > TH_LIGHT) && (Sensor(IN_3) < TH_LIGHT)){
            OnFwd(OUT_A, POW);
            Off(OUT_C);
            Wait(TIME);
        } else if((Sensor(IN_1) < TH_LIGHT) && (Sensor(IN_3) > TH_LIGHT)){
            OnFwd(OUT_C, POW);
            Off(OUT_A);
            Wait(TIME);
        } else if((Sensor(IN_1) < TH_LIGHT) && (Sensor(IN_3) < TH_LIGHT)){
            Off(OUT_AC);
            break;
        }
    }
------------------------------------------------------
}
```

(NXT Python+)

```
#!/usr/bin/python
# -*- coding: euc-jp -*

import basic_action

m = basic_action.Basic_Action('00:16:53:0D:E5:9F')
------------------------------------------------
# この部分に命令を記述
m.linetrace_forward()
------------------------------------------------
m.close()
```

図 7　ライントレースのプログラム (NXC と NXT Python+)

3.2.1　例題 4: 壁にそって走行し、1 周する

2.2.3 節の move_until_collision(タッチセンサが、物体に衝突するのを検知するまで前進するメソッド) と、turn_right_90(もしくは turn_left_90) のメソッドを使うことで、壁にそって走行し、1 周するという動作 (図 8) をつくることができた。まずは前進し、タッチセンサが壁の衝突を感知したら右 (もしくは左) に 90 度回転する。この動作を 4 回繰り返せば、元の位置に戻るという動作である。制御プログラムに記述すると、図 9 のようになった。

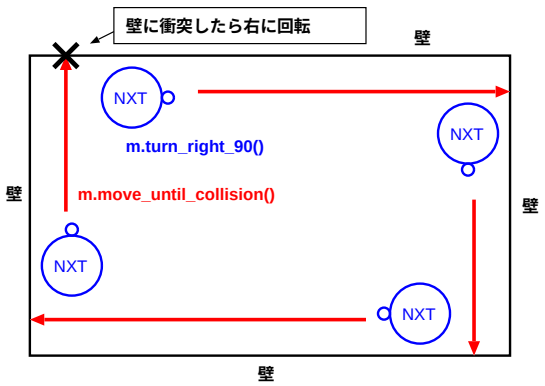

図 8　壁にそって走行し、1 周する動作

(NXT Python+)

```
#!/usr/bin/python
# -*- coding: euc-jp -*

import basic_action

m = basic_action.Basic_Action('00:16:53:0D:E5:9F')
------------------------------------------------
# この部分に命令を記述
m.move_until_collision()
m.turn_right_90()
m.move_until_collision()
m.turn_right_90()
m.move_until_collision()
m.turn_right_90()
m.move_until_collision()
m.turn_right_90()
------------------------------------------------
m.close()
```

図 9　壁にそって走行し、1 周するプログラム

3.2.2 **例題** 5: **目的地までライントレースする**

2.2.3 節の linetrace_forward(ラインが交差する箇所まで前進するメソッド) と、turn_right_90、turn_left_90 のメソッドを使うことで、目的地までライントレースで移動するという動作 (図 10, 図 11) をつくることができた。制御プログラムに記述すると、図 12 のようになった。

図 11 では、目的地までの経路はいくつか考えられるが、今回の実験では行き止まりを 3 箇所設定し、目的地への経路は実質 1 通りだけにしている。そのため、制御プログラム (図 12) に記述する際は、目的地までの経路が分かっているものとして、直進 → 右に 90 度回転 → 直進 → 左に 90 度回転 → 直進 と順番に記述している。この実験では、進む手順を実用的に与えているが、さらなる応用として経路選択のアルゴリズムを別に与えれば、自律的に経路選択をすることも可能である。

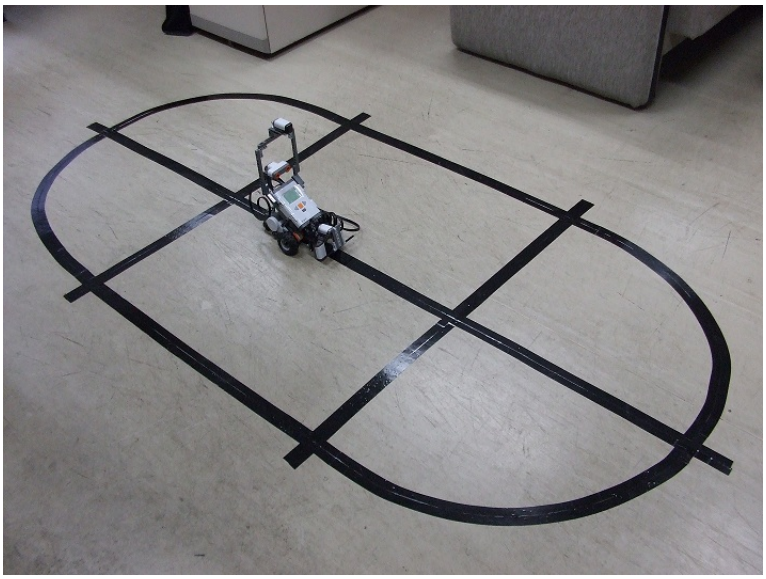

図 10　実験風景

3.3　例題の結果

3.3.1 **例題** 1

“1 秒間、前に直進する” という動作について、それぞれの言語でプログラムを記述し、その記述量を比較すると、NXC では 3 行必要なのに対し、NXT Python+ では 1 行で記述できた。

NXC では、直進するという動作一つを実現するたびにポートとパワーの指定、そして走行時間が必要になる。しかし、NXT Python+ ではその指定も含めて一つのメソッドとしてまとめたことにより、“直進する” という、ユーザから見て単一の動作を一つのメソッドとして実現できた。よって記述量は減少し、命令語も簡潔になった。

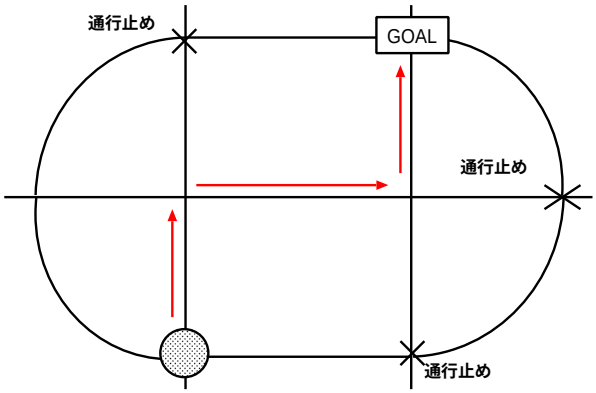

図 11　目的地までライントレースで移動する動作

```
(NXT Python+)

#!/usr/bin/python
# -*- coding: euc-jp -*

import basic_action

m = basic_action.Basic_Action('00:16:53:0D:E5:9F')
------------------------------------------------
# この部分に命令を記述
m.linetrace_forward()
m.turn_right_90()
m.linetrace_forward()
m.turn_left_90()
m.linetrace_forward()
------------------------------------------------
m.close()
```

図 12　目的地までライントレースするプログラム

3.3.2　例題 2

“90 度右に回転する” という動作について、それぞれの言語でプログラムを記述し、その記述量を比較すると、NXC では 4 行必要なのに対し、NXT Python+ では 1 行で記述できた。

NXC では、90 度右に回転するという動作一つを実現するのに、左右のモーター別々にポートとパワーの指定、そして 90 度回転するまでの走行時間が必要になる。しかし、NXT Python+ ではその指定も含めて一つのメソッドとしてまとめたことにより、“回転する” という、ユーザから見て単一の動作を一つのメソッドとして実現できた。よって記述量は減少し、命令語も簡潔になった。

3.3.3　例題 3

NXT がラインの交差する箇所までライントレースするという動作について、それぞれの言語でプログラムを記述し、その記述量を比較すると、NXC では 17 行の命令が必要なのに対し、NXT Python+ では 1 行で記述できた。

ライントレースしてラインの交差箇所まで到達するという動作は、一つの動作とは見なしにくいが、よく現れると考えられる有用な応用のため、NXT Python+ では一つのメソッドとして用意した。その結果、このように少ない記述量で実現できた。

3.3.4　例題 4

action ファイルにあるメソッドを組み合わせた例として、壁にそって走行し、1 周するという動作を実現した。NXC や現状の NXT Python で同じような動作をする制御プログラムを作成すると記述量が非常に多くなるが、NXT Python+ ではまず基本行為を簡単なメソッドにまとめ、さらにそれらのメソッドを組み合わせて上位レベルのメソッドを作成しているので、ユーザ側はモーターの指定等を意識することなく、動作に関するメソッドのみを記述すれば良い。

3.3.5 例題 5

action ファイルにあるメソッドを組み合わせた例として、目的地までライントレースするという動作を実現した。3.3.4 節同様、NXC や現状の NXT Python で同じような動作をする制御プログラムを作成しようとすると記述量が非常に多くなるが、NXT Python+ ではモーターの指定等を意識することなく、動作に関するメソッドのみを記述すれば良い。

3.4 考察

3.4.1 各例題の結果について

3.3.2 節の 90 度右に回転する動作では、NXC や現状の NXT Python で記述すると、90 度回転するようになるまで制御プログラムのファイルを修正し、モーターのパワーを調整する必要があったが、NXT Python+ では最初に GUI で調整済みなので、90 度の回転がすぐに実現でき、非常に容易に制御できるようになった。

3.3.5 節の目的地までライントレースする動作は、本質的な行動列に自然に対応する命令列で実現できたが、ライントレースを実験中、NXT が少し斜めを向いて交差点に進入した場合に停止しないことが何度か起こった。原因は、左右のライトセンサが同時にラインを検知しないと停止しないため、例えば交差点に対して少し斜め右を向いて進入すると、左のライトセンサが先に黒を検知してしまい、ラインが左に曲がっていると判断、結果として交差点で停止せずに左に回転し、そのままライントレースを続行してしまうためである。

しかし、この問題については、直線距離を長くとればその間に行動補正ができ、交差点に対して垂直に進入することができた。だが、ライントレースをするにあたり、必ずしも直線距離を長くとれるとは限らないので、この問題についてはプログラムの修正が必要である。例えば、交差点に対して少し斜め右を向いて進入すると、左のライトセンサが先に黒を検知してしまい、ラインが左に曲がっていると判断するところで、本当にラインが左に曲がっているのか、あるいは交差点であるのかを確かめる必要がある。この問題を解決すれば、例えばラインの交差が直角ではなく、鋭角だった場合でも対応できるようになると考えられる。

3.4.2 本章の検証実験について

すべての例題に共通することとして、NXC のプログラムではポートやモーターのパワーを直接記述するため、調整するたびにファイルを開いて修正、そして保存、コンパイル、NXT にプログラムを転送、といった順序をふまなければならない。一方、NXT Python+ ではモーターは既に調整済みであり、NXT にプログラムを転送して実行するだけなので、ユーザにとって非常に便利になった。NXT Python+ を構築する際の目的であった、“今後 NXT を使った行動制御に関する研究を行う際に、少しでも制御が容易になること” に適っている。

また制御プログラムを書いたとき、動作一つ一つが短く分かりやすく書けることで、プログラム開発が早く出来るといった効果が期待できる。

以上の結果より、NXT Python+ にはライブラリとしての有用性があると判断される。

4　まとめ

本研究では、上位レベルの記述で LEGO MINDSTORMS NXT という小型ロボットを制御できるライブラリ、NXT Python+ を構築し、その設計や機能について述べた。さらに、構築した NXT Python+ を用いて NXT を制御する実験も行い、ライブラリとしての有用性の評価を行った。実験の結果、制御プログラムの記述量の減少や、上位レベルでの制御が実現できたため、ライブラリとして有用性があるという結果になった。

5　将来展望

本研究で構築した NXT Python+ には、2.2.1 節にあるように、モーターのポート選択に関して制限が設けられている。GUI 設計の際、ロボットが移動することを前提として、モーターのポートを 2 個だけ選択すると仮定したので、今後はこの部分をさらに拡張し、モーターのポートを 0〜3 個まで選択できるようにすることで、制限を取り除くことが考えられる。

また、NXT Python+ にメソッドやアルゴリズムを追加することで、本ライブラリをさらに発展させることも考えられる。例えば、2.2.3 節の action ファイルについて、基本行為の組み合わせ方次第で様々なメソッドが作成できるので、ロボット制御の研究の状況に応じて必要なメソッドを作成し、action ファイルに追加することで、本ライブラリをより充実させることが考えられる。また、3.2.2 節のライントレースの例題では、さらなる応用として、目的地までの経路を自律的に選択するようなアルゴリズム、例えば、遺伝的アルゴリズムや A-star 探索アルゴリズムによる経路選択を制御プログラムに組み込むことで、より知的なロボット制御が可能なライブラリに発展させることも考えられる。

6　謝辞

本研究を遂行するにあたり、熱心にご指導して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、日頃から丁寧な助言を与えてくださっている、奈良女子大学大学院の藤田恵さん、そして論文作成にあたり、鴨浩靖准教授、さらに新出、鴨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] Bricx Command Center 3.3. http://bricxcc.sourceforge.net/.

[2] Next Byte Codes and Not eXactly C. http://bricxcc.sourceforge.net/nbc/.

[3] Tkinter - PythonInfo Wiki. http://wiki.python.org/moin/Tkinter.

[4] WRO JAPAN 公式サイト. http://www.wroj.org/.

[5] ロボカップ日本委員会公式ホームページ. http://www.robocup.or.jp/.

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

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

[8] 藤吉弘亘, 藤井隆司, 鈴木裕利, 石井成郎. 実践ロボットプログラミング. 株式会社 近代科学社, 2009.