わかるから面白い！

TOMOT®

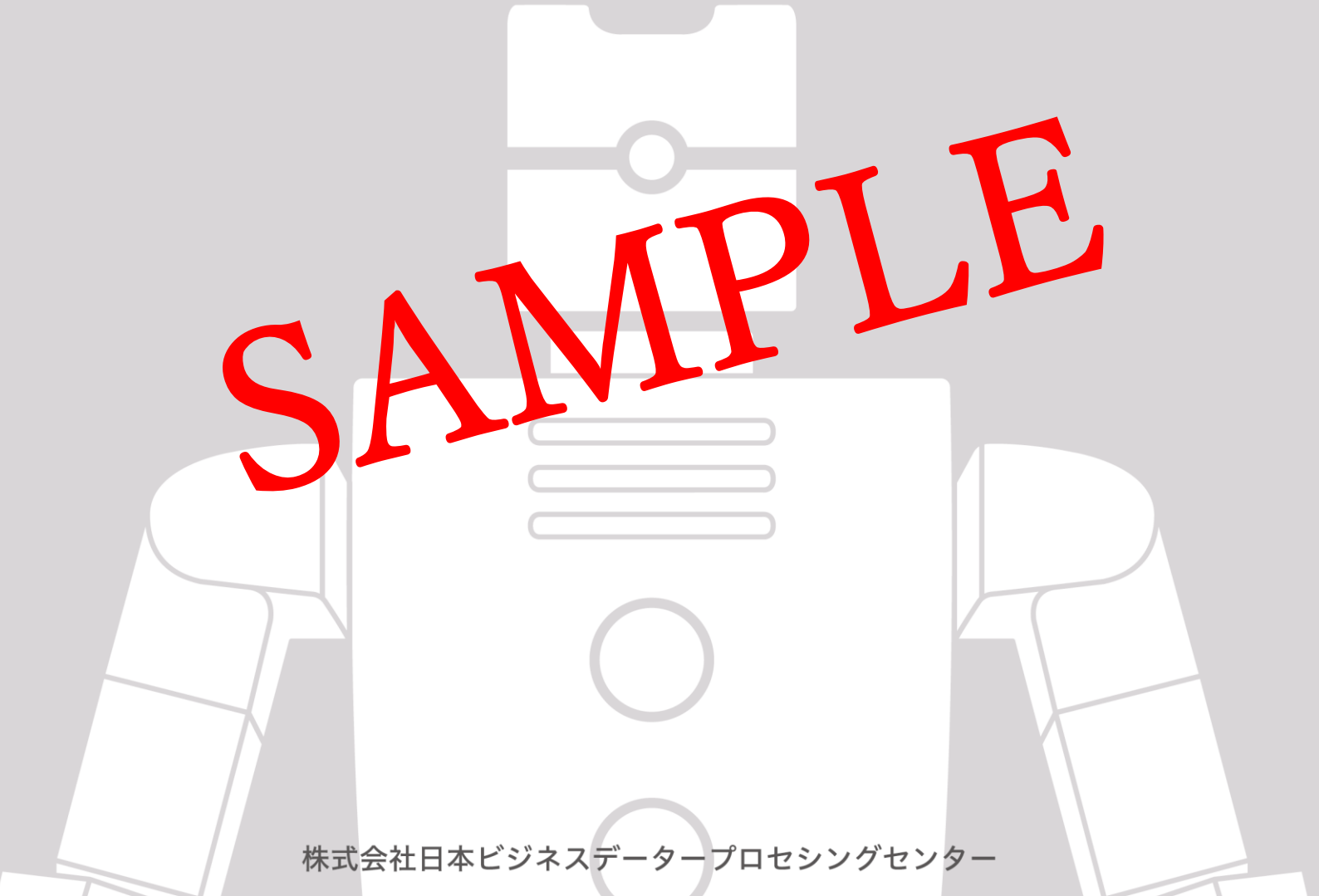
らくらく ロボット工学 ことはじめ 1

Hello,Robotics!

入門編

ロボット工学は難しくない！

未経験から誰でも世界標準に触れ合える、省人化時代で差がつくロボット工学入門書



株式会社日本ビジネスデータープロセシングセンター

サンプル教科書について

らくらく　ロボット工学ことはじめ　1　入門編

- Hello, Robotics! -

本書は「らくらく　ロボット工学ことはじめ　1　入門編　- Hello, Robotics! -」のサンプル版として作成されたものです。実際の教科書からの一部抜粋となっており校正も異なります。実際の教科書の内容を一部ご覧いただくことで、授業などでご活用いただく教材としての導入イメージを高めていただくためのものとなっております。

目　次　　サンプル教科書版

学習の進め方について

学習開始の前に、必ずお読みください。

学習教材の確認

最初に学習教材が揃っていることをご確認ください。

学習環境として、以下が揃っているかどうかご確認ください。
※ 内容物は変更になることがありますので予めご了承ください。

TOMOT-Arol（本機）

梱包付属品

- ☐ TOMOT-Arol（本機）
- ☐ AC アダプター（GST60A07-P1J） … ①
- ☐ 電源ケーブル（アース付） … ②
- ☐ 延長ケーブル（長） … ③
- ☐ 変換ケーブル（短） … ④
- ☐ マイクロ SD カード 32GB
  収納クリアケース … ⑤
- ☐ バッテリー充電器一式
  - ○ AC アダプター（FAD-3） … ⑥
  - ○ 充電器（LBC-3E5） … ⑦
  - ○ 電源ケーブル … ⑧
  - ○ 取扱説明書
- ☐ バッテリー一式 … ⑨
  - ○ バッテリー（PR-4S780P）
  - ○ 取扱説明書
- ☐ ピンセット … ⑩

※ AC アダプターおよび電源ケーブルは、故障の原因となりますので、本製品以外では使用しないでください。

※ マイクロ SD カードは、メインコンピューターである Raspberry Pi3 Model B へ差し込まれています。

※ バッテリーは、すでに本機に装着してありますので、取り外さないことをお勧めします。取り外す場合は、ボディー部のネジを外す必要がありますので、ネジ穴をつぶさないためにも No.0 のプラスドライバー（+ドライバー）のご使用を推奨します。

全ての学習パーツが揃っている前提で学習を進めていきます。不足している場合は、メーカー（前述のお問い合わせ先）にお問い合わせください。

（別売オプション品一覧）

標準の学習教材とは別売で以下のオプションが用意されています。より発展的にROSを学習する際はご検討ください。

※本書の学習では、下記オプションは必須ではありません。

開発用小型PC

□ 開発用小型PC一式

- ◦ 開発用小型PC(LIVA Z)
  ROSカスタムモデル
- ◦ ACアダプター
- ◦ 電源ケーブル
- ◦ VESAマウント
- ◦ 各種ネジ（ネジA：2本、ネジB：4本）
- ◦ ドライバーDVD
- ◦ クイックガイド

無線キーボード・マウスセット

□ 無線キーボード・マウスセット

- ◦ 無線キーボード
- ◦ 静音マウス
- ◦ 取扱説明書

TOMOT-Aro1展示台

□ TOMOT-Aro1展示台

TOMOT-Aro1展示イメージ

目 次

はじめに

日本は、深刻な少子高齢化や労働力不足などの問題を抱え「ロボットニーズ先進国」と呼ばれる一方で、ロボットを使いこなす技術者が不足する問題に直面しています。人材不足が加速する中、ロボット技術者が備えるべき技術は多岐にわたり、AIやクラウドAPI、ロボットの自律制御、ロボット同士の連携制御など、ロボット技術者に求められる知見はますます高度化しています。それに伴い、これらを統合的に制御するための知識も求められるようになってきました。

ROSロボット学習教材「TOMOT-Arol」は、このような日本が直面する社会問題の根本的解決を目指した世界標準化ロボット工学学習プラットフォームとして、兵庫県より「ひょうごNo.1ものづくり大賞 選考委員会特別賞」を受賞しております。基礎から要素技術をテンポよく学ぶことができ、ガラパゴス化が懸念されているロボット技術において、世界標準規格への移行を進め、日本のロボット産業界の国際競争力を高めるとともに、様々なニーズに適応可能な人材を育成することが狙いです。

ロボットが工場の中で黙々と働いていた時代は過去のものになるかもしれません。多くのロボットは既存のITシステムと連携し、ネットワーク上では多くの制御データがビッグデータとして解析されるようになり、AIと連携することでシステムが自主的に効率化を進めることが可能になります。職場では、協働ロボットの普及が進み、人々とともに協力して作業を進めていくようになるでしょう。
このような社会において、世界の第一線で活躍する人材に求められる素養は何か？という観点から本書は執筆されています。

本書では、TOMOT-Arolの環境設定から基本的な操作方法を記し、スムーズな導入を後押しします。学習過程でつまずきやすい部分には、あらかじめ解決に導く手順を備えていますので、本書にそって進めていただくことで知識ゼロからの学習でも無理なく楽しく学習いただけることでしょう。

また、LinuxやROSといった世界標準オープンソース技術に慣れ親しみ、ロボットの全体像を明確にイメージしながら学びますので、部分的知識や学習に偏った結果、「木を見て森を見ず」といった状況に陥ることを防ぎます。

サーボモーターの制御には、ノイズに強く長距離通信が可能なことから産業ロボットなどで広く利用されているシリアル通信規格「RS485」を採用しており、20軸の

関節自由度で様々なポーズやモーション作成が行えます。また、産業ロボットでは一般的なダイレクトティーチングにも対応しており、プログラミングを行わずにモーション制御が可能なため、プログラミングに自信がない方でも安心してご活用いただけるよう配慮しています。

各種センサー制御では、世界中の多くのメーカーが採用しているシリアル通信規格「I^2C」や「SPI」を使用します。センサーからの入力制御や実装方法を学ぶことで、世界中の標準的なセンサーを自由に扱える知識が身につきます。

また、無線端末であることの強みを活かし、様々なモノがインターネットに接続され相互制御する IoT（Internet of Things）社会を見据えて、AI・外部 API 連携の実験や研究に活用してもよいでしょう。センシングデータをクラウドにアップロードし機械学習に活用したり、外部の音声認識・音声合成・会話 API を活用し、サービスロボットやスマートスピーカー開発の研究用プラットフォームとして活用することもできます。

本格的な制御や設計にも十分役に立つ構成で、導入敷居は下げながらも最終的な理解や習得技術を高めることに成功しています。本機ならびに本書を通じて、一人でも多くのロボット技術者を輩出し、世界の第一線で活躍していただくことが「TOMOT-Arol」開発者の願いです。

本書は、これらの本格的な次世代技術学習への導入教材として基本的な制御の学習に留めておりますが、別冊では少し踏み込んだ内容まで学習します。

一昔前まで、パソコンを扱えるのはほんの一部の人だけでしたが、今ではパソコンを扱えないと仕事にならないばかりか、小学生までプログラミングが必修となる時代になりました。同じことが AI やロボットにも起ころうとしています。誰もが AI やロボットを使いこなす時代はもうすぐそこまで来ています。

本書が、AI・ロボット社会で活躍される皆様の一助となれば、これに勝る幸せはありません。

1-1. 本書の目的と位置付け

本章では、学習を開始するに当たり、本書の目的と履修の結果得られる知見について説明を行った後、記述ルールや解説方法、学習のコツなどを説明させていただきます。

【本書の目的】

本書は、近い将来ロボットやAIが社会に深く入り込み、社会の運行システムの多くを担う時代に備え、かつては一部のロボット技術者や研究者にしか必要とされなかったロボット制御やAI活用の知識が多くの人に求められるようになってきており、特殊スキルから現代社会を生きるための一般教養になっていく時代において、社会の第一線で活躍される皆様の一助となることを目的として執筆されました。

このような時代の変遷に伴う一般教養の変化は、ロボットやAIだけに限ったものではなく、外国語教育や情報処理教育においても同様の過程を経てきました。一昔前、コンピューターを操作できる人は、ごく限られた専門家やマニアだけでした。しかし、今では、ほとんどの業界でコンピューターは活用されており、プログラマーやエンジニアでなくても、コンピューターの操作は必須スキルと言われております。
英会話能力においても、航空会社や通訳だけでなく、町の飲食店や販売店でさえ英会話スキルを必須としているお店が多くなってきました。国際化社会の要請により、英語力が一般教養化してきている証拠と言えるでしょう。

本書では、より多くの方にロボット制御を学んでいただくため、専門的な公式や難解な理論を極力減らし、俯瞰的にロボットの構造や仕組み、制御技術などを学んでいただけるように配慮されています。
本格的なロボット設計や開発を行う際は、難解な理論の理解も当然必要になってきますが、高度な情報化社会において、CPU（中央演算処理装置）の仕組みを理解したり、OS（オペレーティングシステム）の設計開発が行えなくても、十分にコンピューターを活用して仕事の効率化を図ることができるのと同様、ロボットを設計開発はできなくても、人の作業をロボットに教えて効率化を進めるスキルと教養が求められると考えられるからです。

これまで、ロボット技術は、極めて難解なものと思われてきました。
制御プログラミングに関しては、コンピューターに慣れ親しんだプログラマーやSE（システムエンジニア）でさえ二の足を踏んでしまう領域だったと言ってよいでしょう。こうしたロボット学習の敷居の高さを取り払い、誰もが手を伸ばせば得られるスキ

ルにしなければならない。これが、本機が開発され、本書が出版された理由になります。

本書は、ロボット工学を学ぶための入門教材として、エンジニアのトライアスロンと言われるロボット技術において、俯瞰的に全体像をつかみ次のステップへと導くと同時に、ロボット学習を実践的に楽しく学ぶ方法が提供されています。
本機と本書で学習するうちに、知らず知らずのうちに世界標準技術に慣れ親しみ、グローバルスタンダードが身に付くことでしょう。最初は少し難解に思える部分もあると思いますが、制御技術は理解よりも慣れが重要な場面も多いため、分からなくても動かしてみて、もう一回読み直してみると、"なるほど"となることが多々あるはずです。まずは、やってみて、ロボットに向き合いながら楽しく学習を進めてみてください。

学び終わる頃には、ロボット制御は、神業のようなスキルではないと気付くはずです。

「ロボットは自分でも扱える!」

こう思えるようになったら、あなたはロボット技術者としての扉を開けたことになります。

ようこそ、こちらの世界へ。「Hello Robotics!」

1-2. 本書での学習の進め方

【学習における前提条件】

本書の学習では、ROS ロボット学習教材「TOMOT-Aro1（以降、「本機」と表記します。）」を使用してロボット制御を学びます。

本機は、背中にメインコンピューターとなる Raspberry Pi3 Model B を搭載しており、ロボットの制御はこのコンピューターが担っています。また、ロボット制御には欠かせない制御プログラム開発環境が最初からインストールされていますので、このメインコンピューターの HDMI ポートにディスプレイを接続し、USB ポートにキーボードとマウスを差し込んで（無線推奨）電源スイッチをＯＮにするだけで、瞬く間に開発環境の準備が整います。
基本的に本書で開発を学ぶ際は、本機はこの状態で起動します。

※ 一旦起動させると、HDMI ケーブルを抜いても大丈夫です。必要なときに HDMI ケーブルを接続すれば、またディスプレイにデスクトップ画面が表示されます。
但し、せっかく無線に対応したロボットを制御するのに、ロボットから沢山のケーブルが出ているというのは、あまり格好のよいものではありません。

開発用 PC がある場合は、基本的に SSH で接続して、リモートコンソールから開発を行います。

※ 開発用 PC を使用すれば、ロボットを完全無線で制御することも可能です（バッテリー駆動時）。但し、バッテリーでの駆動時間は限られているため、腰を据えて学習するには、やはり電源だけは AC アダプターから供給することが好ましいでしょう。

また、本書の中でコマンドを実行する指示がある場合は、基本的には本機がディスプレイと接続されていてターミナルが起動している状態、または、開発用 PC から SSH で本機にリモートログインしていてコンソールが利用可能な状態を想定しています。
本書では学習しませんが、Gazebo などの 3D シミュレーターを使った開発やロボットの動きを RViz で可視化する際には、本機に搭載するメインコンピューターでは処理能力不足となるため開発用 PC が必須となります。

【本書の記述ルール】

本書の中で、次のページにあるように、背景がグレーになっている箇所があります。これは、ソースファイルと言って、Python で記述されたプログラムが記述され

たテキストファイルです。このように、テキストでプログラムが書き込まれたものを Script（スクリプト）と呼びます。

ソースファイルの中で、行頭が「#」から始まる行と、「"」を 3 つ続けて入力したもの、「"""」で前後を囲われた箇所があります。これは、コメントと言って、コンピューターにはプログラムと解釈されず、人がメモ書きやプログラムの中で機能の説明するために使用します。

```
1.  #!/usr/bin/python
2.  # -*- coding: utf-8 -*-
3.  # これはコメントです。
4.  """ これもコメントです。
5.  """
6.  print("Hello Python!")
```

ソースファイルは中身をすべて記載する場合と、説明のため一部を抜粋して記載している場合があります。

また、以下のように背景が黒く、白文字で記載され、冒頭が「$」から始まる部分があります。これは、ターミナルか SSH におけるリモートコンソールで、コンピューターに命令を与えるための操作画面です。

※「$」より左に記述されている文字列は省略します。以降の説明についても、同様に省略いたします。

この箇所を学ぶには、ディスプレイを本機に接続しているか、SSH でリモートログインしている必要があります。

```
$ python ~/test.py
Hello Python!
```

【学習の進め方】

本書を使って学習を進めるにあたり、どのようなペースで進めればよいかについて説明します。

本書では、Lesson1 ～ Lesson8 までの全 8 章で構成されており、1 講座の時間が 90 分である場合、10 ～ 12 回の受講で履修できるようになっています。

Lesson5 と Lesson8 は、内容も少し難しく、ボリュームもあることから、それぞれ 2 講座で学習を進めるとよいでしょう。

それ以外は、前半の簡単なところは 1 講座あたり 1 ～ 2 章のペースで学んでいただき、後半の難しい箇所や学び終わった後のプログラミングの実習などに時間をあてていただくのが理想的です。

巻末には付録が記載され、本書で引用したPythonコードを理解するための簡単なリファレンスや学習に使用されたサンプルプログラムがまとめられています。本機の取り扱い方法なども記載されており、カリキュラムのいずれかのタイミングで学ぶ必要があります。
目安として、「本製品の取り扱い」はLesson1を終えたあたりで学ぶとよいでしょう。「コマンドリファレンス」はLesson5の前に学んでください。「Python基本構文」はLesson5に入る前、もしくは、最初は分からないまま進めて学習に慣れてもらい、Lesson8に入る前に学ぶことを推奨します。

分からないことに出会ったら、立ち止まるのではなく、一旦そのまま進んで、一連の操作を終えた後に調べたり、学び直すと理解が深まるはずです。
プログラミング言語も、英語などの会話用言語の学習と似ていて、まずは知っている語彙でコミュニケーションを繰り返し、慣れていくことが習得の近道です。従って、まずは少ない知識でも動かしてみることがとても重要です。
言語学習を始めるに際して、辞書に載っている語彙をすべて暗記し終えてから会話の練習に入る人はまずいません。
プログラミングでは、コミュニケーションする相手が人ではなくコンピューターであること以外は、他の会話言語学習と何ら変わりありません。

1-3. 本書での履修範囲

本機を使ったロボット工学の学習では、ロボットの全体像を俯瞰的に捉えることに重点が置かれているため、Linux オペレーションや Python 言語プログラミングについては本書だけでは到底マスターできません。
しかし、これらの技術知識は、今後の技術学習を進める上で極めて重要となってくる基本要素であるため、最小限度に留めながらも、本機を扱うのに必要なことはなるべく詳しく説明していきます。

また、ロボット工学では、電気電子工学や機械工学、制御工学なども必要ですが、これらも本書では説明しておりません。これらの学習には、多大な時間を要することに加え、しっかりマスターするには相当な熱意や努力を要すると同時に、これらに慣れてない人にとっては大変な苦痛を伴う学習となりかねません。

本書は、大学工学部におけるロボット工学を履修する前の入門として活用したり、文系大学や専門学校における情報処理教育の発展形として利用されることを想定しています。また、前提知識を必要としない構成となっているため、高校などの情報処理教育として活用することも不可能ではありません。

【学習での履修範囲】

本機を使って本書を学習することで、以下のような知識が身に付きます。

- ロボットの構造と仕組み
- Linux コマンド基本操作
- ロボット制御プログラミングの基礎
- Python プログラミングの基礎
- オブジェクト指向の基礎
- ROS の基本構造の理解と環境構築
- ROS 制御開発の基礎

本書は、1. 入門編という位置付けのため、本格的な開発を学ぶ方には物足りない部分もあるかもしれません。ROS 編や制御工学編なども続々と発売を予定しており、そちらではかなり実践的なロボット制御開発を学んでいただけます。本書の学習を終えた際は、是非とも同じロボットを使ってより発展的な学習に進んでください。

構成要素について

搭載ソフトウェア

3-1. Python とは

本機を操作するにあたり、制御の基本的な部分に関しては Python（パイソン）という開発言語で操作を行います。

Python は、コンパイル不要のインタープリター言語でありながら実行速度が極めて速いことが特徴です。また、多くの言語では、利用用途がある程度限定されているにも関わらず、Python では機械制御からデスクトップアプリや Web 開発など、非常に広範囲でサポートされています。近年では、初心者でも扱いやすい言語として知られ、世界的に爆発的に利用者が増加しています。

Google が正式に採用したことで一気に認知され、Java や C 言語に代わって世界的標準言語としての地位を築きつつあります。最近のクラウド型 API や機械学習・深層学習などとの親和性も高く、最も注目されている言語と言ってよいでしょう。

本書では、初心者でも学びやすく、開発スキルとしても汎用性の高い Python でロボット制御の基本を学びます。本機の特徴である ROS 制御でも Python をベースに話を進めていきますのであらかじめご理解ください。

※ 本書では、サンプルプログラムで使用する構文は解説しますが、Python の言語仕様を詳述するものではありませんので、詳しい解説は Python の専門書をご参照ください。

3-2. Python を試してみよう

Python の実行プログラムは、メモ帳などのテキストエディターで書くことができます。そして、Python がインストールされている環境であればコンパイルせずすぐに実行ができます。

それでは早速、Pluma テキストエディタを起動して「Hello World!」ならぬ「Hello Python!」を書いてみましょう。

※ Pluma テキストエディタの起動は、デスクトップ画面から「アプリケーション」⇒「アクセサリ」の順に進み「Pluma テキストエディタ」をクリックします。

```
1. #!/usr/bin/python
2. # -*- coding: utf-8 -*-
3.
4. print("Hello Python!")
```

テキストエディターで書くのはたったこれだけです。ファイルパスを「~/test.py」

で保存したら、早速以下のコマンドを実行してみましょう。

```
$ python ~/test.py
```

すると、以下のように、先ほど実行したコマンドの下の行に「Hello Python!」が表示されることが確認できるでしょう。

```
$ python ~/test.py
Hello Python!
```

それでは、先ほどテキストエディターで書いた実行プログラムを説明していきます。

1行目の「#!/usr/bin/python」は、Python インタープリターの場所になります。
環境変数にパスが通っている場合は、「#!/usr/bin/env python」と記述することもできますが、複数のバージョンの Python がインストールされている場合など、どのバージョンの Python で実行するのかを指定するためにも最初の説明のようなフルパスで指定することもあります。

※ 本機の Python は 2.7 と 3.5 の 2 つのバージョンがインストールされており、後者でも Python2.7 が指定されるため、1 行目を「#!/usr/bin/env python」と記述しても同様の実行結果が得られます。

2行目の「# -*- coding: utf-8 -*-」は、文字コードの指定になります。
ここでは、UTF-8 という文字コードを指定しています。世界中のほとんどの文字を文字化けせずに表示できる UTF-8 が世界標準として使用されるようになってきており、特別な理由がない限りこの表記で問題ないでしょう。

最後に、4 行目の「print("Hello Python!")」ですが、これは任意の文字列をコンソール（標準出力）に表示する命令です。
ここでは、「Hello Python!」と表示させています。

※ print('Hello Python!') のように、シングルクオーテーションで囲むこともできますが、C 言語の規定で 1 文字をシングルクオーテーション、複数文字をダブルクオーテーションで囲む慣習から、一般的にはダブルクオーテーションでくくる例が多く見られます。

ここでは、1・2 行目で「#!/usr/bin/python」と「# -*- coding: utf-8 -*-」をそれぞれ説明しましたが、次項からは記載を省略しています。しかし、Python プログラムでは、必ず最上部に記載する表記と覚えておいてください。

3-3. Python 文法

【インデント・変数】

Python を学ぶ上でとても重要な文法は、何と言ってもインデントです。インデントとは字下げのことで、半角スペースやタブ [TAB] で字下げを行います。

```
1.  num = 0
2.  while num < 3:
3.      print ("No." + str(num))
4.      num += 1
5.  print ( "end" )
```

上記の内容を「~/test.py」という名前で保存し、以下のようにコマンドを実行してみましょう。

```
$ python ~/test.py
```

すると、以下のような結果が得られるはずです。

```
$ python ~/test.py
No.0
No.1
No.2
end
```

それでは、先ほどテキストエディターで書いた実行プログラムを説明していきます。

while とは、繰り返しを行う制御構文のことですが、「while 条件式 :」という記述ルールで、式の条件が正しい（真）のときには、while 以下の文が実行されます。
ここでの条件式が「num < 3」なので、num が 3 より小さければ、while 以下の文が実行されることになります。

ところで、ここで登場した num（ナム）とは何でしょうか。
これは、num という名前の変数です。
※ 変数とは、値を入れるための入れ物のことで、ここでは num と名付けられた入れ物と考えてください。

1 行目の「num = 0」で、num という名前の変数（Python では、存在しない場合は変数が作られます）に 0 という数字を代入しています。

2 行目の while では、上述のとおり条件式が正しい場合に while 以下の文が実行されることを意味します。
「num = 0」なので while 条件式「0 < 3」は真となり、その下の「print ("No." + str(num))」

と「num += 1」が実行されたのです。
print は、標準出力（ディスプレイ）に文字を出力する命令です。"No." という文字に num の中身である 0 という数字を文字に変換したものを繋げて表示するという意味になります。
※ str は、指定したオブジェクトを文字列に変換して返す関数です。この場合のように、数値を文字列と連結したい場合などに使用します。
その下にある「num += 1」は、インクリメントと言って、num に入っていた値に 1 を足すという命令で「num = num + 1」と同じ意味になります。
ここまでが、繰り返しを行う制御構文 while となり、「num += 1」を実行した後、下の行にある文「print ("end")」に移らず、元の while から繰り返すため、「No.0」に続けて「No.1」「No.2」が出力される実行結果となっています。
このように、while がどこまでの処理を繰り返すのかを定義しているのが、while の下に記述された 2 行のスペース 4 文字によるインデント（字下げ）なのです。

そして、最終行の「print ("end")」では、"end" という文字を出力する命令文で、while での条件式が誤り（偽）となった際に実行結果として出力されています。

インデントが極めて重要な理由が分かったと思います。正しいインデントを行わないと Python は思った通りの処理をしてくれません。
このように、厳しい規則でインデントを行うことで、Python Script はとても読みやすいプログラムになるのです。

【条件分岐】

インデントが理解できたところで、今度は、次のようにファイルを書き換えてみましょう。

```
1.  num = 1
2.  while num > -2:
3.      if num == 0:
4.          print ( "empty" )
5.      elif num > 0:
6.          print ( "exist" )
7.      else:
8.          print ( "error" )
9.      num -= 1
10. print ( "end" )
```

上記の内容を「~/test.py」という名前で保存もしくは上書きをし、以下のようにコマンドを実行してみましょう。

```
$ python ~/test.py
```

すると、以下のような結果が得られるはずです。

```
$ python ~/test.py
exist
empty
error
end
```

それでは、テキストエディターで書き換えた実行プログラムを説明していきます。

まずは、1 行目の「num = 1」で、num という名前の変数に 1 という数字を代入しています。2 行目の while では、条件式が正しい場合に while 以下の文が実行されます。インデントで繰り返す範囲を確認でき、ここでは、3 行目の「if num == 0」から 9 行目の「num -= 1」までが繰り返されることが分かります。

次に、3 行目にでてくる制御構文 if が、このプログラムで注目すべきポイントです。これは、分岐を制御する構文で、「if 条件式 :」という記述ルールで、条件式が真のときには、if の次の行のさらにインデントされた文が実行され、実行されると if の制御を抜けて下の行に移ります。if の制御範囲は、else の下の行、つまり、8 行目の「print ("error")」までであり、if の条件式が偽であった場合、次の elif を評価します。そして、この elif の条件式が存在しなければ else を実行します。

※ elif は、記述がなくても複数でも構いません。else は、記述がなくてもよいのですが、複数存在することは許されません。

実際に while 以下を見ていくと、「num = 1」なので while 条件式「1 > -2」は真となり、その下の「if num == 0」が実行されます。次に、3 行目の「num == 0」は「1 = 0」となり偽なので、次の「elif num > 0」が実行されました。「num > 0」は「1 > 0」となり真なので、その下の「print ("exist")」が実行され、"exist" という文字を出力し表示されたのが見てとれます。そして、if の制御を抜けて「num -= 1」が実行されたのです。
ここまでが、繰り返しを行う制御構文 while となり、「num -= 1」を実行した後、下の行にある文「print ("end")」に移らず、元の while から繰り返すため、「exist」「empty」「error」が出力される実行結果となっています。

そして、最終行の「print ("end")」では、"end" という文字を出力する命令文で、while での条件式が偽となった際に実行結果として出力されています。

このように、条件に合わせて実行されるコードを制御する if は、とても重要で、よく利用される制御構文です。是非覚えておきましょう。また、条件式における＝（イコール）は、「==」のようにイコールが 2 つであることもあわせて覚えておきましょう。

【関数】

関数とは、あらかじめ決まった処理をするために用意された処理のことを言い、引数を受け取り、処理結果を返します。

下記のようなプログラムを作成し、「~/add.py」という名前で保存しましょう。

```
1.  def add ( a , b ):
2.    c =  a + b
3.    return c
4.
5.  val = 3
6.  val2 = 5
7.
8.  while val > 0 :
9.    answer = add ( val , val2 )
10.   print ( str(val) + " + " + str(val2) + " = " + str ( answer ) )
11.   val -= 1
```

以下のようにコマンドを実行してみましょう。

```
$ python ~/add.py
```

すると、次のような結果が得られるはずです。

```
$ python ~/add.py
3 + 5 = 8
2 + 5 = 7
1 + 5 = 6
```

それでは、テキストエディターで書いた実行プログラムを説明していきます。

1行目の「def add (a , b)」では、addという名前の関数を定義しています。これを、関数の宣言と言い、インデントされている3行目まで適用させています。(a , b)については引数（ひきすう）と呼び、aとbという名前の変数として値を2つ受け取ることができます。そして、このaddという関数は、実行されるとreturnで返されるcの値に置き換わります。

次に、5・6行目の「val = 3」「val2 = 5」で、valという名前の変数に3、val2という名前の変数に5という数字を代入しています。

そして、8行目のwhileでは、条件式が正しい場合にwhile以下の文が実行されます。インデントで繰り返す範囲を確認すると、最終行まで繰り返されることが分かります。
8行目の「answer = add (val , val2)」は、「answer = add (3 , 5)」と置き換えられ、「add (3 , 5)」は、「3 + 5」つまり「8」となるので、「answer = 8」を実行したのと同じことになります。
その下はもう分かりますね。

10 行目以降では、while 条件式「val > 0」が「3 > 0」となり真なので、「print (3 + 5 = 8)」と「val -= 1」が実行され、while の条件式が偽になるまで繰り返し実行結果を出力しているのです。

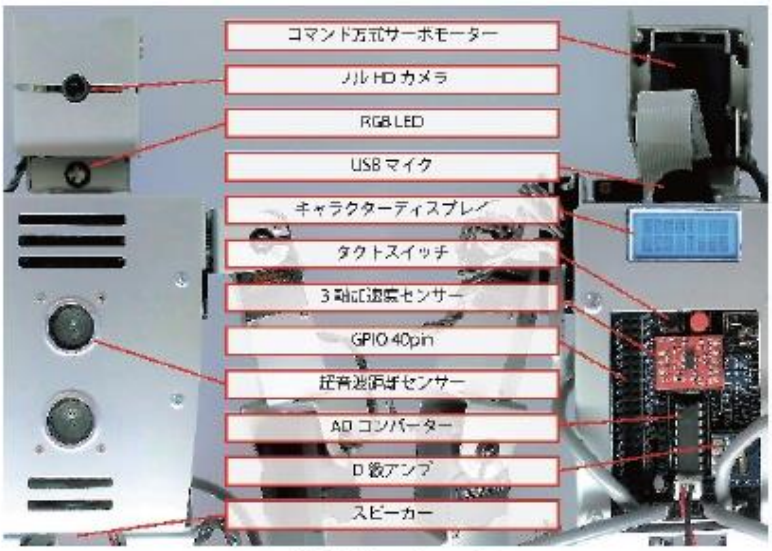
ここまで、駆け足で Python の基本的なプログラミングについて学んできましたが、想像していたほど大変ではないことがお分かりいただけたかと思います。本書では、Python の言語仕様を詳述するものではありませんので、Python の言語学習だけをみると不十分な内容かもしれません。機会があれば、Python の専門書でじっくり学んでみるのもよいでしょう。きっと、Python の学習に費やした時間の何倍もの恩恵を受けられることでしょう。

搭載ハードウェア

4-1. ハードウェアの全体構成

本章では、本機に搭載されたハードウェアの構成と、それぞれのモジュールについて紹介します。

本機には Raspberry Pi3 Model B をメインコンピューターとして、ロボットコントロールボードの日本データー製シールド基板を介して、次のようなモジュールが搭載されています。



（図）4-1-1. モジュール一覧

Raspberry Pi3 Model B

本機のメインコンピューターです。

日本データー製ロボットコントロールボード

「TOMOT-RCB1」

RS485 通信用コントローラーをはじめ、アナログインプットや D 級（デジタル）アンプ、3 軸加速度センサーなどを実装したコントロール基板です。

次項から、各モジュールについて説明します。

制御方法について

5-1. サーボモーターの制御について

本章では、サーボモーターの制御について説明します。

【サーボモーターの種類】

サーボモーターは、大きく分けて「PWM 制御」と「コマンド方式制御」の 2 種類があります。それぞれの特徴について説明していきます。

PWM（Pulse Width Modulation）制御は、パルス幅変調とも言い、デジタル回路から疑似的にアナログ出力のような可変電圧を得るために使われる手法です。つまり、PWM を使うと、ソフトウェア制御で電圧操作を実現できるため、本来、電圧の変化に必要となる専用回路が不要となります。
例えば、1W から 5W まで徐々に電圧が上がっていくような回路では、高速に電圧の ON と OFF を切り替え、その ON となる時間と OFF の時間の比率（デューティー比）によって平均実効電圧を変化させる方法です。この ON の時間と OFF の時間の長短をサーボの角度制御に使用したのが、PWM 制御サーボになります。
PWM はシンプルですが、回路のノイズ源となることに加え、極めてノイズに弱いという性質があり、人型ロボットのような繊細な制御には向きません。

一方で、コマンド方式制御は、目標動作や動作特性がまとまっていることが特徴です。目標角度と移動時間を指示すれば途中の軌道計算はサーボ自身が行うため、ロボット全体の制御装置の負荷を大幅に軽減することができます。

【本機に搭載されているサーボについて】

前章でも説明しましたが、本機では、産業用ロボットなどにも使用され、ノイズにとても強い、RS485 というシリアル通信規格に対応した双葉電子工業社製コマンド方式サーボモーター RS305CR を使用しています。RS485 バスをとお通してコマンドを送信することで、様々な操作が可能となります。

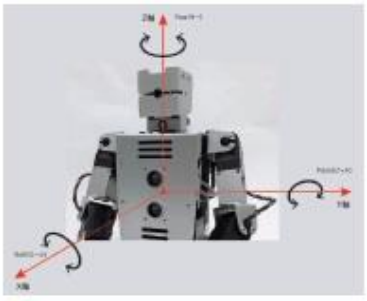
【本機に搭載されているサーボの ID と情報】

これからのサンプルプログラムを使った制御解説では、次ページの表にもとづいて説明します。例えば、2 列目の「ID」は、制御に使われるサーボ識別情報になります。

分類	ID	部位	軸	回転	0度の状態	正方向
頭部	11	首	Z	Yaw	正面を向いている	右回旋
	12	首	Y	Pitch	正面を向いている	前屈
右腕	21	肩	Y	Pitch	腕の内側（手の平）が正面内向き 45°	回内
右腕	22	肩	Z	Yaw	腕を水平に上げている	内転
	23	肘	Z	Yaw	肘を伸ばしている	伸展
左腕	31	肩	Y	Pitch	腕の内側（手の平）が正面内向き 45°	回外
	32	肩	Z	Yaw	腕を水平に上げている	外転
	33	肘	Z	Yaw	肘を伸ばしている	屈曲
右足	41	股関節	Z	Yaw	膝を正面に向けている	内旋
	42	股関節	X	Roll	足を外に開いていない	外転
	43	股関節	Y	Pitch	膝を上げていない	屈曲
	44	膝	Y	Pitch	膝は真っ直ぐ	屈曲
	45	足首	Y	Pitch	つま先が水平方向	伸展
	46	足首	X	Roll	つま先の直角が水平方向	外反
左足	51	股関節	Z	Yaw	膝を正面に向けている	外旋
	52	股関節	X	Roll	足を外に開いていない	内転
	53	股関節	Y	Pitch	膝を上げていない	伸展
	54	膝	Y	Pitch	膝は真っ直ぐ	伸展
	55	足首	Y	Pitch	つま先が水平方向	屈曲
	56	足首	X	Roll	つま先の直角が水平方向	内反

（表）5-1-1. サーボモーターの情報一覧

※ 軸と回転については右図を参照してください。
また、正方向（＋）動作については本書では記しませんが、人体関節の働きを参照してください。



（図）5-1-2. 軸と回転

5-2. サンプルプログラムで学ぶサーボ制御

【開発の前提について】

本機での開発の場合は、背中のメインコンピューターの HDMI ポートにディスプレイ、USB ポートにキーボードとマウスを接続して起動し、本機がログインの状態でターミナルを起動していることを前提として解説します。

また、開発用 PC を通して開発を行う場合は、本機と開発用 PC が同一ネットワークで接続され、SSH で本機にログインしている状態を前提にしています。

【本機の基本姿勢について】

本機では、立った姿勢（以降、「スタンド姿勢」と表記します。）を開発における基本の姿

勢としており、本機を使って学習や開発を行う際は、スタンド姿勢から行うことを推奨します。
スタンド姿勢にするやり方については、以下の 2 通りがあります。

※ すでにスタンド姿勢の場合、変化はありません。また、座り姿勢などの 2 足で立っている状態でない場合は、本機に手を添えるなどをして、急なスタンド姿勢への変化で転倒しないようご注意ください。

≪タクトスイッチでスタンド姿勢にする方法≫

黒のタクトスイッチで「Stand」を選択し、赤のタクトスイッチで決定をするとスタンド姿勢になります。

≪コマンドラインからスタンド姿勢にする方法≫

以下のコマンドを実行するとスタンド姿勢になります。

```
$ python /opt/ndc/servo/pose/stand.py
```

何か動作をした後などに、一度スタンド姿勢に戻って次の学習を進めていくとよいでしょう。

【サーボモーターの制御方法について】

ここでは、ロボットに接続されたサーボモーターを制御する方法を学びます。次の手順にそって、サーボモーターを動かしてみましょう。

≪サーボモーターを動かす手順≫

1. 本機を起動します。
2. 以下のコマンドを実行します。

※ 本機へログインしている状態でコマンドを実行してください。

```
$ python /opt/ndc/sample/servo/servo_01.py
```

※ 開発用 PC を通して開発を行う場合は、本機と開発用 PC が同一ネットワークで接続され、SSH で本機にログインすることが必要となります。接続方法については、巻末の環境構築の「環 -3. リモート接続設定」を参照してください。

3. コマンドを実行すると、以下のようにいくつかの質問が一つずつ順番に表示されます。それぞれ以下のように答えていきます。実行に成功すると、本機の首が動くことが確認できます。

```
$ python /opt/ndc/sample/servo/servo_01.py
動作させるサーボモーターの ID を入力してください：11
角度を入力してください (-1500 ～ 1500)：300
移動時間を入力してください (0 ～ 16383)：100
サーボモーター (ID: 11) を，30.0° まで，1.00 秒で移動しました .
```

4. 本機の首が動いたことを確認できましたでしょうか。

RS305CR は、このように「サーボ ID」「角度」「移動にかける時間」の 3 つを指定することで制御を行います。
今回は、首に搭載されたサーボ ID：11 を指定したので首が動作しています。前項のサーボ情報一覧をもとに、同様の手順で他のサーボモーターも動かしてみましょう。

【サンプルプログラムについて】

本機に搭載されているサンプルプログラムは、ロボット制御の基本をしっかり学んでいただくため、機能を最小限で構成した学習用のプログラムです。
実際の運用では、安全面の配慮など様々な機構を考える必要がありますが、これらを記載してしまうと極めてコードが難解となってしまうため、機能だけにフォーカスし、最小の学習コスト（労力）で学べるように配慮しています。
また、コマンドラインからの入力が前提であるため、キーボードからの入力を待つような記述が中心となります。

ここからは、実際のサンプルプログラムを見ていきましょう。サンプルプログラムは、指定のファイル閲覧で中身の確認ができますので、ここでは、テキストエディター「nano」を使った操作で説明していきます。

※「nano」以外のテキストエディターでの閲覧やファイルの操作方法については、「2-3. Linux CUI の基本的な操作方法」を参照してください。

それでは、以下のようにコマンドを実行してみてください。

```
$ nano /opt/ndc/sample/servo/servo_01.py
```

すると、以下のようなサンプルプログラムが出てきたことでしょう。

〖 /opt/ndc/sample/servo/servo_01.py〗

```
1.  import servo_rs485 as rs485
2.  import time
3.
4.  # 動作内容の入力
5.  servo_id = int(raw_input(" 動作させるサーボモーターの ID を入力してください： "))
6.  servo_angle = int(raw_input(" 角度を入力してください (-1500 ～ 1500): "))
7.  servo_time = int(raw_input(" 移動時間を入力してください (0 ～ 16383): "))
8.
9.  # インスタンス作成．ポート開放
10. my_rs = rs485.Rs485()
11. my_rs.open_port()
12.
```

```
13. # サーボ操作（トルク ON）
14. my_rs.set_torque(servo_id, 1, 0)
15. time.sleep(0.1)
16.
17. # サーボ操作（位置設定）
18. my_rs.set_target_position(servo_id, servo_angle, servo_time, 0)
19. time.sleep(servo_time / 100)
20.
21. # ポートを閉じる
22. my_rs.close_port()
23.
24. # 動作完了のアナウンス
25. print(" サーボモーター (ID: %d) を，%.1f° まで，% 2f 秒で移動しました ." % (servo_
    id,servo_angle / 10, servo_time / 100))
```

上記は、先ほどコマンドを実行し、本機の首のサーボを動かしたサンプルプログラム「servo_01.py」の中身です。ここでは、文字コードや Python パスの設定、そして、プログラムの説明書きや著作権表記の説明などは省略しています。

それでは、サンプルプログラム「servo_01.py」の説明をしていきます。

サンプルプログラムを見てみると、プログラムの最上部には下記のような 2 行があります。こちらから確認していきましょう。

```
import servo_rs485 as rs485
import time
```

これは、「import 文」と呼ばれ、外部のプログラムを読み込むための方法です。

このサンプルプログラムの import 文は、「time」という時間に関する機能（関数）をまとめてクラスライブラリー化されたものを呼び出しています。これは、Python に標準で搭載されているライブラリーです。
一方で、1 行目に「import servo.rs485 as rs485」という表記がありますが、これは、本機に搭載されているメインコンピューターの Raspberry Pi3 Model B から RS485 シリアルトランシーバーを制御するためのライブラリーです。Python には標準で搭載されていないため、本機用に当社で開発し、あらかじめインストールしたものになります。

Python には、様々な標準プログラムが用意されています。これらを「ライブラリー」と呼びます。また、まとまった処理をどこでも使えるように、汎用的に書いて呼び出して使うことを「関数を作る」と言います。そして、一例として、ファイル操作に関する機能をまとめて関数のグループとして扱うことがあり、そのような手法を「ライブラリー化」すると言い、一般的にはクラスライブラリーとして開発します。クラスに格納された関数は、「メソッド」と呼び、区別されます。
このように、一度書いたプログラムは、以後は呼び出せるようにプログラムをグループ化

して管理し、開発と修正の手間を大きく削減する開発スタイルが現在では一般的です。
このような開発スタイルを「オブジェクト指向開発」と呼び、主な開発言語として Java や C++ が有名です。Python も、もちろんこのオブジェクト指向に対応している開発言語です。

※ 少し複雑なシステムを開発する際、これまでの方法では、いたるところに似たようなプログラムを書く必要がありました。そのほとんどが同じプログラムだったため、コピーをして貼り付けるなどの作業でコードを複製し、プログラムが無駄に長くなってしまい、そのプログラムに修正が必要となった場合には、方々を探して修正しなくてはいけなくなり、大変工数のかかる作業となっていました。

次に、4 行目以降を見てみると、以下のような入力を待つ記述が見られます。

```
# 動作内容の入力
servo_id = int(raw_input(" 動作させるサーボモーターの ID を入力してください : "))
servo_angle = int(raw_input(" 角度を入力してください (-1500 ～ 1500): "))
servo_time = int(raw_input(" 移動時間を入力してください (0 ～ 16383): "))
```

これは、コンソールに「動作させるサーボモーターの ID を入力してください :」「角度を入力してください :」「移動時間を入力してください :」と表示させて、キーボードからの入力を待ち受けるプログラムです。

Windows の操作に慣れ親しんだ方には、このようなコマンドラインに抵抗があるかもしれません。しかし、LINE やメールを使ったことがある方なら、文字で意思疎通をすることは理解できるはずです。コマンドラインとは、コンピューターに仕事をしてもらうためのチャットコミュニケーションだと考えてください。少しは抵抗感も薄れることでしょう。

さて、ここからは、実際のサーボ制御に入っていきます。9 行目を見てみましょう。

```
# インスタンス作成．ポート開放
my_rs = rs485.Rs485()
```

これは、クラスライブラリーの「インスタンス化」という作業です。
「rs485.py」という Python ファイルに記述された「Rs485()」というクラスを、「my_rs」という変数に実行プログラムとして常駐させるという意味の記述になります。
こうして変数に格納された実行プログラムを「オブジェクト」と呼びます。

次は、11 行目から 22 行目を見ていきます。

```
my_rs.open_port()

  ～中略～

# ポートを閉じる
my_rs.close_port()
```

これは、通信系プログラムによく見られる記述です。
通信ポートを開き、処理を行った後ポートを閉じる。電話をかけ、話し、電話を切るという作業をコンピューター同士が行う処理に似ています。

次に、14 行目に注目してみます。

```
my_rs.set_torque(servo_id, 1, 0)
```

ここでは、通電しているサーボの制御トルクを ON にするメソッドを実行しています。
これを行わないとサーボは動作しません。車で言うところのエンジンスタートと同じで、走る前にエンジンをかける作業に似ています。
「servo_id」は、前項のサーボ情報一覧の「ID」を指定します。例えば、首を指定する場合なら ID は 11 ですから、「my_rs.torque_on(11, 1, 0)」と記述しても同じです。
毎回コマンド入力で動かしたいサーボ ID を指定したいため、ここでは変数になっています。

15 行目はどうでしょうか。

```
time.sleep(0.1)
```

これは、最上部で import した time ライブラリーにある sleep メソッドです。
指定された秒数の間（ここでは 0.1 秒）プログラムを一時停止します。先ほど実行した「my_rs.torque_on(servo_id, 1, 0)」が、処理完了前に動作制御命令を出してしまうことを防止するために行います。
このように、通信プログラムでは、通信のタイムラグをある程度考慮しながらプログラミングを行う必要があります。

いよいよ、サーボの駆動です。18 行目を説明していきます。

```
my_rs.set_target_position(servo_id, servo_angle, servo_time, 0)
```

サーボ制御の命令は、たった 1 行です。
これは、先ほどの通信で、電話がつながっている相手に対して要件を伝える作業です。
14 行目では、「何番のサーボのトルクを ON にして」と命令を出しましたが、ここでは、先ほどトルクを ON にしたサーボに動いてほしい場所と速さを指定します。
「servo_id」は、先ほどと同じで首の 11 になります。
「servo_angle」は、-1500 ～ 1500 の範囲を 0.1 度刻みで合計 300 度を指定可能です。
「servo_time」は、0 ～ 16383 の値を指定可能です。単位は 10ms となるので、100 を指定した場合は 1 秒となります。また、サーボ速度よりも速い速度での指定の場合には、最速で処理されます。

そして、最後に実行結果を出力しています。

```
# 動作完了のアナウンス
print("サーボモーター (ID: %d) を、%.1f° まで、% 2f 秒で移動しました。" % (servo_id,
servo_angle / 10, servo_time / 100))
```

本項では、最初の解説となるので、Python の仕組みまで詳しく解説しましたが、次項以後のサンプルプログラムについては、機能のみの解説とさせていただきます。

5-5. デモモーションとティーチング制御

前項では、サンプルプログラム「servo_05.py」でポーズ情報を見てきました。より発展した学習に向けて、ここでは、当社が開発したデモアプリを使い、モーション起動やサーボ角度情報の取得、また、プログラミングなしでモーション作成が行える「ティーチング制御」について説明していきます。

【デモアプリの起動方法について】

それでは、次の手順にそって、デモアプリを起動していきましょう。

1. 本機を起動します。
2. 以下のコマンドを実行します。

※ 本機へログインしている状態でコマンドを実行してください。

```
$ python /opt/ndc/demo.py
```

※ 開発用 PC を通して開発を行う場合は、本機と開発用 PC が同一ネットワークで接続され、SSH で本機にログインすることが必要となります。接続方法については、巻末の環境構築の「環 -3. リモート接続設定」を参照してください。

3. コマンドを実行すると、以下のように「>>> Root Menu」が表示されればデモアプリの起動は完了です。

```
>>> Root Menu

[Motion]
  1. Take a standing pose
  2. Play the motion 'Sit down'
  3. Play the motion 'Stand up'
  4. Play the motion 'Flamingo'
  5. Play the motion 'Walk'
  6. Play the motion recorded in JSON file
  7. Adjust the motion recorded in JSON file
  8. Capture the motion to the JSON file

[Sensor]
 11. 3-axis acceleration sensor
 12. Ultrasonic sensor
 13. RGB LED with PWM

[Meta]
 91. Tell IP Address of the robot
 92. Handle the robot by itself
 99. Exit
```

4. デモアプリの起動が完了したら、「デモモーション」と「ティーチング制御」をそれぞれ実行していきましょう。

【デモモーションの実行方法について】

ここでは、本機にあらかじめ搭載してあるデモモーションの実行方法を説明します。デモモーションについては、すでに搭載されているモーションの他に、新たにモーションを作成して追加することもできます。本項で操作方法を学び、オリジナルモーションの作成を楽しんでみてください。

※ デモアプリの操作については、すでに本機のタクトスイッチでの操作もできますので試してみてください。

それでは、次の手順にそって、デモモーションを実行してみましょう。

1.「>>> Root Menu」で「Input number:」と記載されている行に任意の数字を入力し、JSON ファイルを再生します。それぞれの番号については以下で説明します。

1. ：立っている姿勢を取ります
   すでに立っている場合は、そのままの状態を保ちます
2. ：立っている状態から座り込むモーションを実行します
   すでに座っている場合は、そのままの状態を保ちます
3. ：座っている状態から立ち上がるモーションを実行します
   すでに立っている場合は、そのままの状態を保ちます
4. ：片足立ちのモーションを実行します
5. ：二足歩行のモーションを実行します
6. ：モーションを再生します
7. ：モーションを編集します
8. ：モーションを録画します

11. ：3 軸加速度センサーの値をコンソールと本機の LCD に表示します
12. ：超音波距離センサーの値をコンソールと本機の LCD に表示します
13. ：LED を使用した PWM 制御のデモを実施します

91. ：本機に割り当てられている IP アドレスをコンソールと本機の LCD に表示します
92. ：本機のタクトスイッチを使ったデモに切り替えます
99. ：デモアプリを終了します

2.「6. Play the motion recorded in JSON file」を選択した場合、実行したいファイル名のファイル番号を「Input number:」に入力しモーションを再生します。

※ モーションによっては転倒することがありますので、モーション再生時は、必ず近くで手を添

えるなど本機をサポートしてください。

【ティーチング制御の実行方法について】

次に、プログラミングなしで操作が可能なティーチング制御の方法を学んでいきましょう。

ティーチングとは、主に産業用ロボットに実際の動作をさせることで、制御プログラムとして動作を記憶させることです。つまり、その名の通り「Teaching = 教えること」です。
本機に搭載されている双葉電子工業社製のコマンド方式サーボモーターならではの機能として、ロボットに記録したポーズの角度情報が取得でき、この機能により人間がロボットに教えたポーズが数値化されているのです。

ティーチングには、「モーションの作成」「モーションの再生」「モーションの編集」の3段階があります。それぞれの方法について、簡単に説明していきますので、実際に動作をして、簡単にロボット制御ができることを確認してみてください。

≪モーションの作成方法≫

モーションは、一つ一つのポーズの連続で成り立つものであり、ここで言うサーボモーターの角度の保存とは、ポーズを記録することを意味します。

1.「>>> Root Menu」で「Input number:」と記載されている行に「8」を入力します。
※「8. Capture the motion to the JSON file」を選択すると、即座に全身のサーボモーターがブレーキングモード（半脱力状態）になるので、手で支えるなどのサポートをして、転倒しないよう注意してください。

```
>>> Root Menu

～ 中略 ～

Input number: 8
Press enter key to capture angles of all servo-motors.
When you get off capturing, press Ctrl + C.
```

2. 上記のような説明文が出てきたらモーションの作成に進みます。
3. 最初のポーズを記録します。
※ 全身のサーボモーターがブレーキングモードになっていますので、重心や全体のバランスを考え、無理なく転倒しないポーズを意識して、任意のポーズを取らせてください。

4. 記録したいポーズが決まったら、[Enter] キーを押して記録します。

※ ポーズ決定時にキー入力が必要となるので、2人以上で行うと安全かつスムーズに進みます。

5. 連続してポーズを記録していきます。
   最初のポーズを登録したら、同様に次のポーズを取らせ、[Enter] キーを押して記録していきます。そして、モーションが終了するまでこの作業を繰り返します。

※ 慣れるまでは、3～4ポーズで1つのモーションを作成してみてください。


6. すべてのポーズを記録し終えたら、[Ctrl] + [C] キーを押して JSON ファイルに保存します。その際、「Enter title of the motion:」と保存名を尋ねられますので、分かりやすい名前を入力します。

```
>>> Root Menu

～ 中略 ～

Capturing complete. Press enter key to capture next posing.
Press enter key to capture angles of all servo-motors.
When you get off capturing, press Ctrl + C.
^C
Enter title of the motion:
```

≪モーションの再生方法≫

1. ティーチングで保存したモーションを再生するには、「>>> Root Menu」で「Input number:」と記載されている行に「6」を入力し、JSON ファイルを再生します。
2. 「6. Play the motion recorded in JSON file」を選択したら、次に、JSON ファイルに保存したファイル名のファイル番号を「Input number:」に入力し、モーションを再生します。

※ モーションによっては転倒することがありますので、モーション再生時は、必ず近くで手を添えるなど本機をサポートしてください。

≪モーションの編集方法≫

1. ティーチングで保存したモーションを編集するには、「>>> Root Menu」で「Input number:」と記載されている行に「7」を入力し、JSON ファイルを編集していきます。
2. 「7. Adjust the motion recorded in JSON file」を選択したら、次に、JSON ファイルに保存したファイル名のファイル番号を「Input number:」に入力し、モーションを編集します。以下のような選択されたモーションの編集画面が表示され、最初のポーズを取ります。

※ モーションによっては転倒することがありますので、モーション再生時は、必ず近くで手を添えるなど本機をサポートしてください。

```
>>> Root Menu

[Motion name] Flamingo.json              [Poses]  1/11              [Time]  50

[1*: Head]      [2*: Arm(R)]      [3*: Arm(L)]      [4*: Leg(R)]      [5*: Leg(L)]
 11:     0       21:    -900       31:    -900       41:     -50       51:     -50
 12:   100       22:    -750       32:    -750       42:     -40       52:     -40
                 23:    -150       33:    -150       43:       0       53:       0
                                                     44:       0       54:       0
                                                     45:     -25       55:     -25
                                                     46:     -40       56:     -40

                  << How to adjust >>
        7         : << Previous pose
        9         : Next pose >>
        0         : Focus on the time to take the pose
        1-5&10?   : Focus on the servo motors belonging to the group
                    (1: Head,  2: Arm(R),  3: Arm(L),  4: Leg(R),  5: Leg(L),
                     100: ALL,  101: TOPS,  102: LEGS)
        11 to 56  : Focus on the servo motor
                    (the servo motor focused will be in breaking mode)
        99        : Save the motion to fileHandler and exit adjustment
        999       : Edit frames in motion
        9999      : Destroy changes and exit adjustment

Input number:
```

それでは、編集画面の記載内容を以下で説明していきます。

[Poses] では、11 個のポーズで構成されたモーションのうち、1 つ目のポーズを表示していることを意味しています。

[Time] では、このポーズを取るのにかける時間を [10ms] で表しています。

※ 値が「50」の場合、50 × 10ms = 500ms となり、0.5 秒を意味します。

また、中段にある 11 ～ 56 の数値については、各サーボモーターの角度情報を表示しています。単位はそれぞれ [0.1 度] です。

※ 値が「900」の場合、900 × 0.1 度 = 90.0 度を意味します。

そして、<< How to adjust >> 以下では、それぞれの編集方法を記載しており、「Input number:」に数字を入力してモーションを編集していきます。数値（角度情報）を画面上から編集することで、モーションの細かいチューニングが可能となります。
それぞれの番号の説明については以下のとおりです。

7	：前のポーズ
9	：次のポーズ
0	：このポーズを取るのにかける時間を編集します
1	：頭系（ID:11, 12）のサーボを一括編集します
2	：右腕系（ID:21, 22, 23）のサーボを一括編集します
3	：左腕系（ID:31, 32, 33）のサーボを一括編集します
4	：右足系（ID:41, 42, 43, 44, 45, 46）のサーボを一括編集します
5	：左足系（ID:51, 52, 53, 54, 55, 56）のサーボを一括編集します
100	：すべてのサーボを一括編集します
101	：上半身のサーボを一括編集します
102	：下半身のサーボを一括編集します
11 to 56	：該当する ID のサーボに設定されている角度情報を編集します
99	：変更を保存して終了します
999	：ポーズの複製や削除を行います
9999	：変更を破棄して終了します

3. 最初のポーズを編集したら、「Input number:」に「9」を入力して次のポーズを取らせ、同様に、連続して編集していきます。すべての編集が終了するまでこれを繰り返します。また、前のポーズに戻る場合は、「Input number:」に「7」を入力して前のポーズを取らせます。
4. すべてのポーズを編集したら、「Input number:」に「99」を入力して変更を保存します。

※ 変更を保存せずに終了してしまうと、編集した角度情報はすべて無効となりますので注意してください。

5. 編集したすべてのポーズの角度情報が変更されているかを確認します。

※ 編集が反映されていない場合は、何らかの操作ミスが考えられます。もう一度、モーションの編集をを行ってください。

少しずつ理解が深まってきたことでしょう。次の章では、センサーの制御を学んでいきます。

ROS について

ここまでは、Python を使ったロボット制御を学んできました。しかし、本機にはロボット用ミドルウェア「ROS（ロス）」を搭載しており、本機の最大の特徴である ROS による制御も可能となっています。

ここからは、ROS によるロボット制御を実際に体感いただきます。まず、本章では、ROS の概念を学んでいきます。

7-1. ロボット制御用ミドルウェア「ROS」とは

Robot Operating System（ROS）は、ロボット用ソフトウェアを作成するための柔軟なフレームワークであり、ミドルウェアです。様々なロボットプラットフォームで、複雑かつ頑強なロボット制御プログラムを作成するにあたり、作業の簡素化を目的としたツールやライブラリーの集まりとも言えます。
具体的には、ハードウェア抽象化、デバイスドライバー、ライブラリー、視覚化ツール、メッセージ通信、パッケージ管理などが提供されています。

ロボットには、様々なセンサーや駆動系が搭載されており、これらを複雑に非同期で同時に制御することが求められます。もちろん、これは簡単なことではありません。目的のロボットを制御する基幹システムを作成し、そこからすべてを制御するプログラムを書くことは、非常にコストのかかる作業となります。そして、残念なことに、そうやって作成した特定メーカー製のロボット用プログラムは、他メーカーのロボットでは使えません。ロボットのメーカーを変更するたびに、プログラマーは、制御用ソフトウェアの作成に追われることになり、ロボットを導入することで特定メーカーへの依存が大きくなります。
一見、当たり前とも思えますが、あなたが最初に買った自動車がトヨタ製だったとします。次に、ホンダ製に乗り替えようとするときに、運転免許を取り直す必要があったとしたら不便だとは思いませんか。メーカーを替えるたびに全く違う操作を学び、免許を取り直すなんて考えられないと思います。
このように、他のものに置き換えて考えてみると、特定企業依存度が高い製品が、どれほど購入者にとって不利益かが理解できたと思います。
これまで、日本メーカーの多くは、このように、自社の技術に深く依存するものづくりを得意としてきました。皆さんは知らないかもしれませんが、1995 年頃の日本は家電大国と呼ばれ、世界中が日本製の家電を欲しがりました。同時に、半導体大国、携帯大国、ロボット大国と何をつくっても日本製は世界から注目されました。これらを総じて、「ものづくり大国」と呼ばれていたのですが、ちょうどこの頃から世界

標準が声高に叫ばれるようになり、国内特定の技術や特定企業依存技術でつくられた製品が、世界標準技術に置き替わる形で姿を消していきました。かつて、国内PC市場を席捲していたNEC PC-88、PC-98シリーズやフューチャーフォンなどが分かりやすい例と言えます。日本は、今もロボット大国としての地位を維持していますが、前述したとおり、特定企業依存については今も根強く残っています。

この状況を打破すべく開発されたロボット用ミドルウェアが「ROS」です。ROSに対応するモジュール化されたドライバーを作成すると、多くのROS対応ロボット上で動作する制御用ソフトウェアになり得ます。様々な分野のスペシャリストが知恵と技能を持ち寄り、ロボット制御用ソフトウェアの生産性を大幅に向上させることに成功しているのです。すでに日本でも、多くのFAにおいてROSが採用される事例が増えてきています。

※ FAとは、Factory Automationの略で、工場における生産工程の自動化を図るシステムのことを意味します。

ROSは、まだまだ発展途上ではありますが、ロボティクスの分野では、すでに次世代の世界標準制御方式と目されており、世界中のロボットメーカーが対応機種の発表を急いでいます。こうした世界標準化への対応を各国のメーカーが進める中で、自社技術に特化したものづくりに固執する形でマイナー製品化していくことを「ガラパゴス化」と呼び、現在では非常にリスクの高い製品戦略と考えられるようになってきました。

最新のROSに関する情報については、公式のROS Wikiを参照してください。

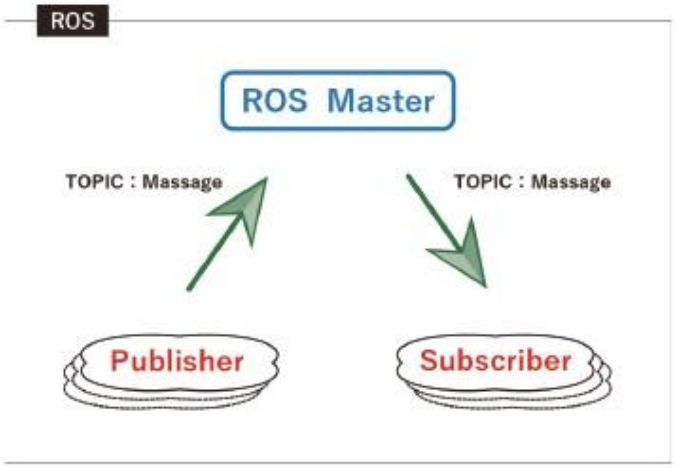
http://wiki.ros.org/ja

7-3. ROS の最小構成

次章からは、ROS を使ってセンサーからの値を取得し、ダイナミックにロボットを制御していきますが、このように、ROS 使って複数のモジュールを制御するには、以下の最小構成が必要になります。

- ROS マスター（ROS Master / ROS Core）
  ⇒ ROS 上に存在するノード間の通信を取りまとめる管制塔の役割を担います。
  必ずネットワーク上に 1 つ起動している必要があります。
- 配信者（パブリッシャー Publisher）
  ⇒ ROS にメッセージを配信するノードの総称です。
  トピック名と値をセットで配信します。
- 購読者（サブスクライバー Subscriber）
  ⇒ ROS 上のメッセージを購読するノードの総称です。
  購読したいメッセージのトピック名を指定して、必要な値を取得します。

図に示すと以下のようになります。



（図）7-3-1. ROS の最小構成の例

ROS は、ネットワークを介しての分散コンピューティングをサポートしています。従って、ネットワーク上の複数端末で通信しながら処理を分散させることが可能となっています。しかし、今回は分散コンピューティングは行わず、すべて本機内で動作させることとします。

それでは、本機を使って ROS を使った具体的な制御を体験していきましょう。

公式サイトのチュートリアル

http://wiki.ros.org/ja/ROS/Tutorials/WritingPublisherSubscriber%28python%29

「シンプルな配信者（Publisher）と購読者（Subscriber）を Python を使って書く」プログラムの他にも様々な機能に関するチュートリアルが紹介されています。

8-6. 次のステップへ

本章では、これまで、ROS の簡単な制御を学んできました。しかし、例で示したノードの構成は最小限のもので、ROS には、まだ多くの機能・API が用意されています。充実したライブラリーとその設計思想により、応用的に色々なことが実現可能となります。以下に、一部を紹介します。

【機能】	【API】
• 分散コンピューティング • 3D 可視化（RViz） • センサーデータの記録（rosbag） • リアルタイム・プロット（rxplot） • システムの可視化（rxgraph）	• 各種ドライバー • 画像処理 • 座標変換 • シミュレーション（2D/3D） • モーションプランニング

8-7. RViz を用いて動かしてみよう

ROS に付属してインストールされるソフトウェアの一つに「RViz」があります。RViz には、以下のような機能が備わっています。

- 3D 可視化
- センサーのデータ表示
- ロボットモデルの表示
- その他の 3D データの表示

（図）8-7-1.
センサーのデータをグラフ化した例

【開発用 PC を使用した 3D 可視化】

開発用 PC から本機へリモート接続することで、ROS の様々な機能を用いた本格的な開発ができるようになります。ここでは、オプションとしてご提供している開発用小型 PC を用いて ROS アプリケーションの 3D 可視化ツールである RViz を使い本機を動かしてみましょう。

それでは、次の手順にそって、RViz を試してみましょう。

1. 本機と開発用小型 PC を起動しログインます。

※ ログインに必要となるユーザー名とパスワードについては、巻末の環境構築「環 -3. リモート接続設定」を参照し、あらかじめ設定しているユーザー名とパスワードを入力してください。

2. 本機と開発用小型 PC を同一ネットワークで接続します。

※ ネットワーク環境の設定方法については、巻末の環境構築の「環 -2. ネットワーク環境設定」を参照してください。

3. 以下のコマンドを実行し、ROS マスターを起動します。

※ 本機へログインしている状態でコマンドを実行してください。

```
$ roscore
```

※ ROS のノード間通信を実現するためには、ROS マスターが動作していることが必要条件となります。開発用小型 PC（LIVA-Z-***）が ROS マスター、つまり、Publisher（配信者）側になります。

4. 以下のような起動ログが表示され、ROS マスターの正しい起動が確認できたら、[Ctrl] + [C] キーで ROS マスターを停止します。

※ ここでは、「ROS_MASTER_URI=http://LIVA-Z-***.local:11311/」として起動を確認します。

```
$ roscore

～ 中略 ～

auto-starting new master
process[master]: started with pid [18481]
ROS_MASTER_URI=http://localhost:11311/

setting /run_id to 315c43d4-395a-11e8-b2ac-f83441a5b4a0
process[rosout-1]: started with pid [18494]
started core service [/rosout]
```

5. ROS マスター起動準備として、IP アドレスの設定を下記コマンドで指定します。

```
$ export ROS_MASTER_URI=http://LIVA-Z-***.local:11311/
                              （http://192.168.***.***:11311/ でも可）
$ export ROS_IP=192.168.***.***
```

※ IP アドレスは、開発用小型 PC に割り振られた IP アドレスを入力してください。

6. ROS パッケージをビルドし、動作を確認します。

```
$ cd ~/catkin_ws/
$ catkin_make
$ source devel/setup.bash
$ roslaunch ndc-hn01 ndc-hn01.launch
```

7. それぞれのコマンドの実行に成功すると、RViz が起動し、Publisher 画面をマウスで操作すると画面内のロボットモデルが動くことを確認できたことでしょう。

8. 開発用小型 PC から SSH で本機にログインします。

※ 接続方法については、巻末の環境構築の「環 -3. リモート接続設定」を参照してください。

9. ROS マスター起動準備として、IP アドレスの設定を下記コマンドで指定します。

```
TOMOT-Aro1-00*:$ export ROS_MASTER_URI=http://LIVA-Z-***.local:11311/
                                        （http://192.168.***.***:11311/ でも可）
TOMOT-Aro1-00*:$ export ROS_IP=192.168.***.***
```

※ IP アドレスは、開発用小型 PC に割り振られた IP アドレスを入力してください。

10. 以下のコマンドを実行して、ROS マスターとの接続を確認します。

```
TOMOT-Aro1-00*:$ rosnode list
```

11. 以下のような内容が書き出されれば、ROS マスターとの接続確認できたこととします。

```
TOMOT-Aro1-00*:$ rosnode list
~~~ 実行結果例 ~~~~~~~~~~~~~~~~~~~
/joint_state_publisher
/robot_state_publisher
/rosout
/rviz
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

12. トピックメッセージの配信状況を確認します。

```
TOMOT-Aro1-00*:$ rostopic echo /joint_states
```

13. 実行画面が出力され、配信経過時間などが表示されます。

14. 購読できていることが確認できたら、[Ctrl] + [C] キーで停止します。

※ ここでは、「ROS_MASTER_URI=http://LIVA-Z-***.local:11311/」として起動を確認します。

15. 以下のコマンドで Subscribe を実行し、本機を動かします。

※ コマンドを実行すると、即座に本機が動き出しますので、手で支えるなどのサポートをして、転倒しないよう注意してください。

```
TOMOT-Aro1-00*:$ rosrun ndc_hn01 subscribe_joint_state.py
```

16. 本機が、RViz 画面内のロボットモデルと同じ姿勢（左右の腕を開いた立ち姿勢）になり、Publisher 画面をマウスで操作すると、画面内のロボットモデルと本機が連動することを確認できます。

（図）8-7-2.
3D モデルを生成し実機と同じ角度情報を可視化した例

（図）8-7-3. ロボットを一括制御した例

おわりに

最後までご覧いただきありがとうございます。ROS ロボット学習教材「TOMOT-Arol」を使ったロボット制御の指南書はいかがでしたでしょうか。本書で紹介したサンプルプログラムをとおして、ロボット制御の基本的な内容はご理解いただけたかと思います。ぜひとも、次は、TOMOT-Arol の性能をフルに活用し、ご自身のアイデアを形にするアプリケーションを作成してみてください。

本書では、ROS を使ったロボット制御への導きとして、ROS でロボットを制御するための最小構成に必要なモジュールの基本的な扱い方を中心に解説しました。しかし、Lesson8 でも解説したように、ROS が本領を発揮するのは以下のようなケースです。

- 複数モジュール間の通信をトリガーとしたリアルタイム制御
- 同ネットワーク上の複数のロボットによる連携作業
- 3D モデルを使った綿密なシミュレーション
- 制御を手助けする強力なライブラリーの利用

このような強みから、今後の IoT 業界において、ROS がますます台頭することは想像に難くありません。実際に、すでに ROS による制御を取り入れている現場も増えてきています。今後、ROS のニーズがさらに増えることが予想される中で、本書で学んだ ROS 制御の知識は、間違いなく皆さんの力になることでしょう。もう少し高度な内容について解説する書籍についても順次発売を予定しており、興味のある方は、ぜひそちらでの発展的な学習に進んでみてください。

最後に、TOMOT-Arol の開発から本書の執筆まで多くの方にご協力いただきました。サーボモーターの制御に的確なアドバイスをいただいた双葉電子工業株式会社の皆様。学術顧問として、本書の企画から執筆の方針に至るまでサポートいただいた兵庫県立大学名誉教授の力宗幸男先生。教育現場におけるロボット教材の現状についてアドバイスをいただいた兵庫県立教育研修所様。本機を研究に使用した上で、研究者の立場から生の声をお聞かせいただいた京都大学大学院様。ROS 教材のあり方についてアドバイスいただいた東京大学大学院様。

皆様のご協力で、このように ROS ロボット学習教材「TOMOT-Arol」を使ったロボット制御の指南書を執筆することができました。改めて、感謝申し上げます

株式会社日本ビジネスデータープロセシングセンター
AI・ロボティクス推進室 一同

サンプルプログラムについて

付録 -3. サンプルプログラム

本書において使用したサンプルプログラムをまとめています。いずれも単体で動作する簡単なプログラムなので、必要な機能の実装時に参照してください。

※ サンプルプログラムは、基本的には本書で説明する順に紹介しています。本書で登場しないサンプルプログラムについては、後半で紹介していますのでご覧ください。

〖 /opt/ndc/sample/servo/servo_01.py〗

```
1.  #!/usr/bin/python
2.  # -*- coding: utf-8 -*-
3.
4.  """
5.  sample.servo.servo_01
6.  ~~~~~~~~~~~~~~~~
7.
8.  コマンド方式サーボ「RS30x シリーズ」を RS485 通信で制御するモジュールを使ったサンプルプログラムです．
9.  サーボを指定位置まで移動させます．
10.
11. :version: 1.2.2 (2018/9/12)
12. :copyright: (c) 2018 Nihon Business Data Processing Center Co., Ltd.
13. """
14. import servo_rs485 as rs485
15. import time
16.
17. # 動作内容の入力
18. servo_id = int(raw_input("動作させるサーボモーターの ID を入力してください： "))
19. servo_angle = int(raw_input("角度を入力してください (-1500 ～ 1500): "))
20. servo_time = int(raw_input("移動時間を入力してください (0 ～ 16383): "))
21.
22. # インスタンス作成，ポート開放
23. my_rs = rs485.Rs485()
24. my_rs.open_port()
25.
26. # サーボ操作 ( トルク ON)
27. my_rs.set_torque(servo_id, 1, 0)
28. time.sleep(0.1)
29.
30. # サーボ操作 ( 位置設定 )
31. my_rs.set_target_position(servo_id, servo_angle, servo_time, 0)
32. time.sleep(servo_time / 100)
33.
34. # ポートを閉じる
35. my_rs.close_port()
36.
37. # 動作完了のアナウンス
38. print(" サーボモーター (ID: %d) を，%.1f° まで，% 2f 秒で移動しました ." % (servo_id,servo_angle / 10, servo_time / 100))
```

〖 /opt/ndc/sample/servo/servo_02.py〗

```
1.  #!/usr/bin/python
2.  # -*- coding: utf-8 -*-
3.
4.  """
5.  sample.servo.servo_02
6.  ~~~~~~~~~~~~~~~~
7.
8.  コマンド方式サーボ「RS30x シリーズ」を RS485 通信で制御するモジュールを使ったサンプルプログラムです．
```

```
サーボのトルクモードを変更します．

:version: 1.2.2 (2018/9/12)
:copyright: (c) 2018 Nihon Business Data Processing Center Co., Ltd.
"""
import servo_rs485 as rs485
import time

# 動作内容の入力
servo_id = int(raw_input(" トルクモードを変更するサーボモーターの ID を入力してください： "))
torque_mode = int(raw_input(" トルクモードを入力してください (0: OFF, 1: ON, 2: ブレーキモード )：
"))

# インスタンス作成，ポート開放
my_rs = rs485.Rs485()
my_rs.open_port()

# サーボ操作 ( トルクモード変更 )
my_rs.set_torque(servo_id, torque_mode, 0)
time.sleep(0.1)

# ポートを閉じる
my_rs.close_port()

# 動作完了のアナウンス
if torque_mode == 0:
    print(" サーボモーター (ID: %d) のトルクモードを OFF に設定しました ." % servo_id)
elif torque_mode == 1:
    print(" サーボモーター (ID: %d) のトルクモードを ON に設定しました ." % servo_id)
elif torque_mode == 2:
    print(" サーボモーター (ID: %d) のトルクモードをブレーキモードに設定しました ." % servo_id)
```

〖 /opt/ndc/sample/servo/servo_03.py〗

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

"""
sample.servo.servo_03
~~~~~~~~~~~~~~~~~

コマンド方式サーボ「RS30x シリーズ」を RS485 通信で制御するモジュールを使ったサンプルプログラム
です．
サーボの各種情報を取得します．

:version: 1.2.2 (2018/9/12)
:copyright: (c) 2018 Nihon Business Data Processing Center Co., Ltd.
"""
import servo_rs485 as rs485
import time

# 動作内容の入力
servo_id = int(raw_input(" 情報を取得するサーボモーターの ID を入力してください： "))

# インスタンス作成，ポート開放
my_rs = rs485.Rs485()
my_rs.open_port()

# サーボ操作 ( サーボ情報取得 )
servo_data = my_rs.get_data(servo_id)
time.sleep(0.1)

# ポートを閉じる
my_rs.close_port()

# 動作完了のアナウンス
print(" サーボモーターの情報を取得しました .")
print("ID¥t¥t¥t¥t: %d" % servo_data[0])
print("Angle( 現在位置 [0.1°])¥t¥t: %d" % servo_data[1])
print("Time( 現在時間 [10ms])¥t¥t: %d" % servo_data[2])
```

```
print("Speed( 現在スピード [deg/sec])¥t: %d" % servo_data[3])
print("Load( 現在負荷 [mA])¥¥t: %d" % servo_data[4])
print("Tempersture( 現在温度 [℃ ])¥t: %d" % servo_data[5])
print("Voltage( 現在電圧 [10mV])¥t: %d" % servo_data[6])
```

〖 /opt/ndc/sample/servo/servo_04.py〗

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

"""
sample.servo.servo_04
~~~~~~~~~~~~~~~~~

コマンド方式サーボ「RS30x シリーズ」を RS485 通信で制御するモジュールを使ったサンプルプログラムです .
複数のサーボを同時に制御します .

:version: 1.2.2 (2018/9/12)
:copyright: (c) 2018 Nihon Business Data Processing Center Co., Ltd.
"""
import servo_rs485 as rs485
import time

# 動作内容の入力
servo_angle = int(raw_input(" 角度を入力してください (-1500 ～ 1500): "))
servo_time = int(raw_input(" 移動時間を入力してください (0 ～ 16383): "))

# サーボ操作情報作成
servo_ids = [11, 12, 21, 22, 23, 31, 32, 33, 41, 42, 43, 44, 45, 46, 51, 52, 53, 54, 55, 56]
data = []
for servo_id in servo_ids:
    data.append([servo_id, servo_angle, servo_time])

# インスタンス作成 , ポート開放
my_rs = rs485.Rs485()
my_rs.open_port()

# サーボ操作
my_rs.set_multi_position(data)
time.sleep(servo_time / 100)

# ポートを閉じる
my_rs.close_port()

# 動作完了のアナウンス
print(" サーボモーターを移動しました .")
print(data)
```

〖 /opt/ndc/sample/servo/servo_05.py〗

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

"""
sample.servo.servo_05
~~~~~~~~~~~~~~~~~

コマンド方式サーボ「RS30x シリーズ」を RS485 通信で制御するモジュールを使ったサンプルプログラムです .
JSON ファイルを読み込み , ファイル内の情報をもとに複数のサーボを制御します .

:version: 1.2.2 (2018/9/12)
:copyright: (c) 2018 Nihon Business Data Processing Center Co., Ltd.
"""
import servo.rs485 as rs485
```

らくらく ロボット工学ことはじめ 1　入門編

- Hello, Robotics! –

次世代向け AI･ロボット･IoT ブランド「TOMOT®（ともっと）」

TOMOT（ともっと）の語源は「ともだちロボット」。
猫のロゴマークには、人類最大の「ともだち」という意味が込められています。

人間とともに暮らし、最も身近な存在である猫をモチーフにすることで、
「生活の一部として親しまれ、一緒に学びながら成長して行く、ともだちのような存在になれたら…」
そんな願いを込めています。

また、猫の右目のウインクには「親愛の情」という意味があり、おしゃれな蝶ネクタイには「オス猫」であることを表現しています。

TOMOT のブランド名は、世界中の誰もが馴染み深く感じていただけるよう名付けました。
左右どちらから読んでも TOM（トム）となる TOMOT は、オス猫の総称である TOMCAT（トムキャット）に由来します。英語圏では、トムという名称が最も象徴的な男性名であり、そのことからオス猫を総称してトムキャットとも呼ぶそうです。

TOMOT は、ともだちロボットをコンセプトに「人に寄り添う身近な AI ロボット」を目指してブランドを立ち上げました。

AI･ロボット･IoT の普及が進み、ますます省人化が加速する社会において、人類のともだちとして、乳幼児期から老年期にいたるまで世代を超えて寄り添っていけるような様々なプロダクトを生み出していきます。

TOMOT の製品やサービスが世界の隅々にまで普及していくことで、新たなライフスタイルの提案から、新業態の創出まで、広い視野で開拓し、必要に応じて世界初となる製品の開発にも取り組んでまいります。

これからの社会と人々に、安心と豊かさをご提供することが TOMOT ブランドに込められた私たちの願いです。

TOMOT 公式サイト

YouTube チャンネル