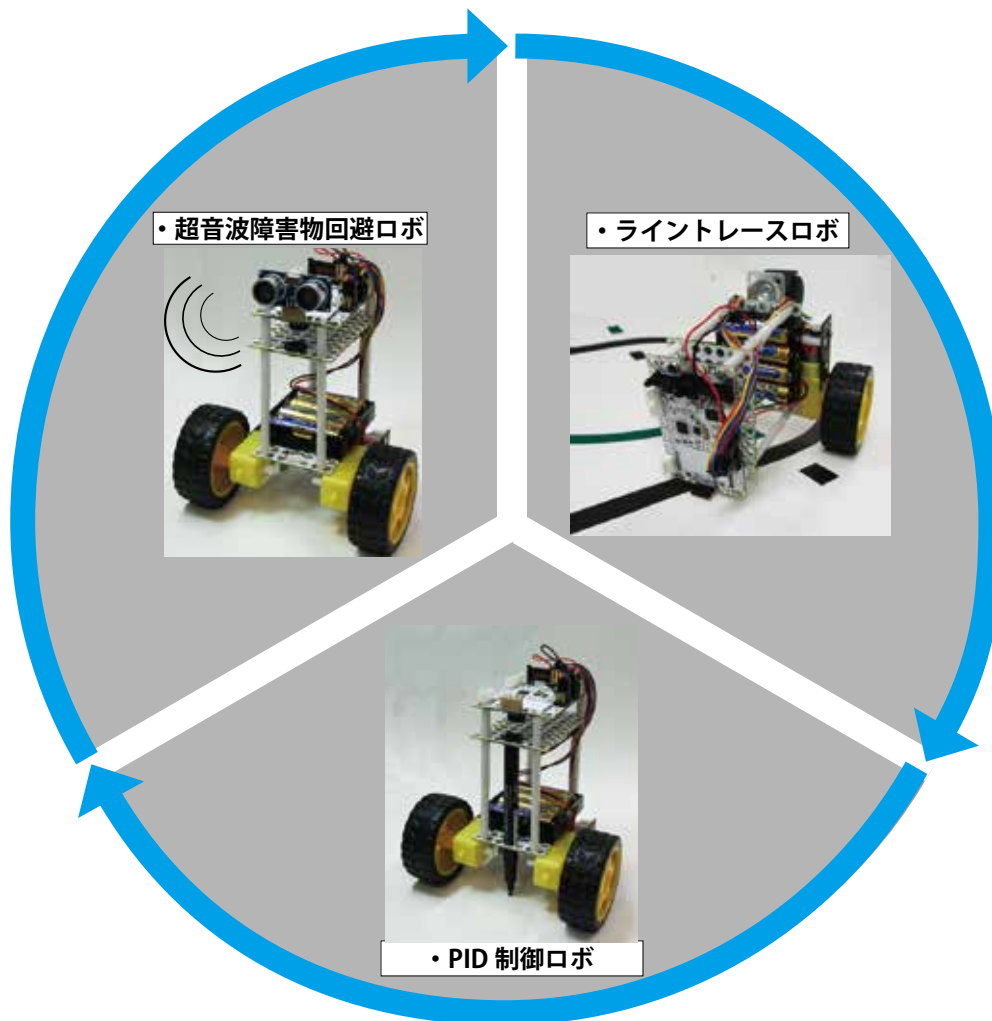


11. PID制御ロボ PID Control Robo RDS-X25



この項では、3種のロボットを作ります。

- ライントレースロボ
- 超音波衝突回避ロボ
- PID制御描画ロボ

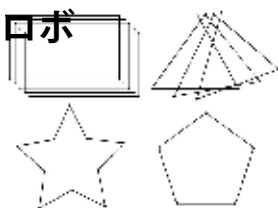
さらに機能拡張し

- IoTロボ を作り

制御則では

- 軌道制御(オドメトリ)

に取り組みます。



PID制御スターターロボット概要

制御方法に比例制御を使って、目標に合わせてロボットを制御し移動する描画ロボットを作ります。

- RDO-502EN エンコーダ付ギアードモータを使用します。
- サインペンをセットした2輪ロボットをモータの回転数によって制御することで図形を描画します。
- モータ停止時のオーバーラン回避のため、比例制御(P制御)で減速します。
- エンコーダの出力値と到達目標の差(残りカウント数=偏差)に比例して、モータへの入力値(PWMのデューティ比=操作量)を変化させます。

搭載コントローラボードにより

Typel (液晶なし、使用可能モータ数(DCモータ×2、サーボモータ×8)、超音波センサ増設可能)

Typell (液晶付、使用可能モータ数(DCモータ×2、サーボモータ×8)、超音波センサ増設可能)のモデルがあります。

すべてのモデルに

・音センサ ・明るさセンサ ・加速度/ジャイロセンサ ・スライダ を搭載しており、これらを利用して各種制御を行えます。

11-1. RoboDesigner RDS-X25構成部品

11.1.1. PID制御ロボRDS-X25パーツリスト

<p><input type="checkbox"/>コントローラボード RDC-103 1枚</p>  <p>セットモデルで、付属コントローラが変わります。 RDS-X25_Type1 :RDC-103_TYPE I RDS-X25_Type2 :RDC-103_TYPE II</p>  <p>下記の部品を取り付けています。 <input type="checkbox"/>樹脂スペーサ M3x10mm 4本 <input type="checkbox"/>ネジ M3x6mm 4本</p>	<p><input type="checkbox"/>エンコーダ付ギアードモータRDO-502EN</p>  <p>組立てに必要なネジが袋に入っています。 <input type="checkbox"/>エンコーダ付ギアードモータ 2個 <input type="checkbox"/>タイヤホイール 2個 <input type="checkbox"/>モータケーブル25cm 2本 <input type="checkbox"/>エンコーダケーブル25cm 2本</p> <p>下記の部品を取り付けています。 <input type="checkbox"/>マウント金具 1個 <input type="checkbox"/>ナベネジ M3x30 2本 <input type="checkbox"/>ナベネジ M3x26 2本 <input type="checkbox"/>ナット M3 4個</p> 	<p><input type="checkbox"/>描画用筆記具取付部品 <input type="checkbox"/>ゲロメット 14mm丸 x H4mm 1個</p>  <p><input type="checkbox"/>スタビライザ作成部品</p> <table border="1" data-bbox="1043 607 1481 763"> <tr> <td></td> <td>全ネジ M3x30mm</td> <td>1個</td> </tr> <tr> <td></td> <td>樹脂スペーサ M3x20mm</td> <td>2本</td> </tr> <tr> <td></td> <td>樹脂スペーサ M3x5mm</td> <td>1本</td> </tr> </table> <p><input type="checkbox"/>ケース</p>  <p>ロボット組み立て時収納ケース</p>		全ネジ M3x30mm	1個		樹脂スペーサ M3x20mm	2本		樹脂スペーサ M3x5mm	1本
	全ネジ M3x30mm	1個									
	樹脂スペーサ M3x20mm	2本									
	樹脂スペーサ M3x5mm	1本									
<p><input type="checkbox"/>ユニバーサルシャーシプレート 2枚</p>  <p>取付に必要なネジが袋に入っています。 <input type="checkbox"/>樹脂スペーサ M3x20 16本 <input type="checkbox"/>樹脂スペーサ M3x15 4本 <input type="checkbox"/>長ネジ M3x95 4本 <input type="checkbox"/>超低頭ネジ M3x4 4本</p>	<p><input type="checkbox"/>後輪キャスタ 1個</p>  <p>組立てに必要なネジ一式が袋に入っています。 <input type="checkbox"/>30mm双輪キャスタ 1個 <input type="checkbox"/>座金組込ネジ M3x10mm 2本 <input type="checkbox"/>平ワッシャー M3x8mm 4個 <input type="checkbox"/>ナット M3 2個</p> <p><input type="checkbox"/>電池ボックス 単3x4 RDP-8093 x 4P 1個</p>  <p>取付に必要なネジが袋に入っています。 <input type="checkbox"/>皿ネジ M3x10mm 2本 <input type="checkbox"/>ナット M3 2本</p>	<p><input type="checkbox"/>補修用ネジ・ナット 1袋 <input type="checkbox"/>ナット M3 10個 <input type="checkbox"/>ネジ M3x10mm 4本 <input type="checkbox"/>ネジ M3x15mm 4本</p> <p><input type="checkbox"/>開発環境 CD-ROM 1枚</p>  <p><input checked="" type="checkbox"/>一緒に使う別売品案内… I²Cコンパスセンサ RDI-5883L</p>  <p>I²Cセンサケーブル付</p>									
<p><input type="checkbox"/>超音波センサ RDI-HCSR04 1個</p>  <p>ピン差し込み式(コントローラにソケット装備)</p>	<p><input type="checkbox"/>マイクロUSBケーブル RDP-824 1本</p> 	<p><input checked="" type="checkbox"/>一緒に使う別売品案内… サーボモータ SG92R</p> 									

11.1.2. 部品の見方、使い方

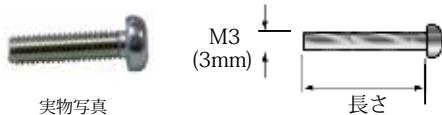
1. 部品サイズ表示

ネジやナットのサイズ表示は以下のとおりです。

■ ^S_C^R_{E^W ナベネジ：頭がナベを伏せたような形の名称です。}

表記：M3 × 10mm

「太さ（直径）3mm、長さ 10mm」という意味です。



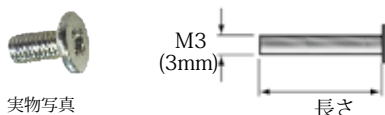
※ネジが切つてあるところの長さです。

Low Head Machine Screw

■ 超低頭ネジ：頭が低く平らな形のネジの名称です。

表記：M3 × 4mm

「太さ（直径）3mm、長さ 4mm」という意味です。



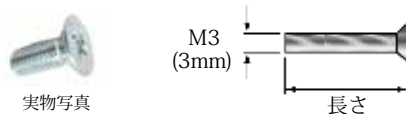
※ネジが切つてあるところの長さです。

Flat Head Screw

■ 皿ネジ：頭が皿のように平らな形のネジの名称です。

表記：M3 × 10mm

「太さ（直径）3mm、長さ 10mm」という意味です。



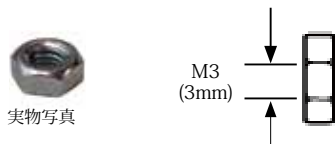
※ネジの頭からの長さです。

Nut

■ ナット

表記：M3

「太さ（直径）3mm のネジ用」という意味です。

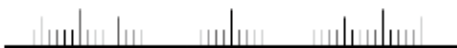


■ 共通で使う上記以外のパーツ



・座金付組ネジ ・スペーサ ・バネ座金 ・平座金

2. 長さ測定用スケール



3. 電子基板使用時の注意

①. 基板表面のピン同士をショートさせないこと。

	表面に装備しているサーボピン等をショートさせないようにしてください。ショートさせると基板が壊れます。
--	--



②. 裏面には必ず隙間を空けて使う。

	電子基板は裏面にも微小な部品や回路パターンが配置されています。圧力を加えると破壊され、また金属製の物体に触れるとショートして基板が壊れるなどの原因になります。
	ネジ・ナットを使って隙間を設けるなど工夫して、裏面の部品や回路パターンが、取付個所などに接触しないように注意してください。

③. 規格範囲内の電圧で使用する。

	電子回路に使用されている電子部品は、規定値の電圧で動作するように設計されています。電源電圧 2V ~ 5V と指定されている場合、指定範囲より電圧が低いと動作が誤ったり、高すぎると回路が破損したりしますので指定範囲内の電圧でご使用ください。
--	--

本製品のコントローラボードの電源電圧規定値は

回路用： 4.5V ~ 6.0V

④. 電源電圧極性を間違えない。

	電子回路に接続する電源電圧のプラス / マイナスを間違えないように注意ください。間違えると部品が壊れます。
--	---

⑤. 水分大敵！ 電子回路は、水をかけると壊れます。

	ロボット製作を行っているとき、夢中になって気がつかないうちに、近くに置いていたカップ容器などを倒したりすると、中に入っていた液体が電子回路にかかり、回路がショートして壊れるなどの事故があります。同じテーブルや机の上に、液体が入っているカップ容器は置かないようにします。
--	--

11-2. コントローラボード概要

11.2.1. RDC - 103TYPE I

- LED / 光センサー、音センサー、ジャイロ/加速度センサ、スライダをボード上に搭載しており、これらを利用して各種制御を行います。
- 外部アナログセンサー 2 個まで接続可能。
- サーボモータ 8 個まで接続可能
- RDC-103TYPE I には、M3、M4、小型液晶モジュールは搭載していません。
- Scratch を使って 2 つのモータまで動かすことができ、常にパソコンと接続して使用します。※ 1
- ※ 1 ・ USB 端子からの電源供給でモータ 1 個が動作可能です。モータ 2 個を動かすには、電池を接続して電源を供給してください。

COMNTROL BOARD RDC-103TYPE I OUTLINE

- It's even possible to move 2 motors using Scratch, and always it's connected with a PC and it's used. ※ 1.
- It's equipped with LED / Light sensor-, Sound sensor-and the Acceleration/a Gyro sensor and a Slider on the board, using these, you can control variously.
- It's even possible to connect 2 of outside analogue sensor (A1,A2).
- It's even possible to connect 8 servomotors.
- M3, M4 and a LCD module aren't loaded into RDC-103TYPE I.
- ※ 1)1 motor can move by power supply from a USB terminal. Please connect a battery and supply me a power supply to move 2 motors.

仕様	The specification	DataSheet URL
マイコン / ATMEGA32U4、発信周波数 8MHz	MCU / ATMEGA32U4 Clock 8MHz	http://media.digikey.com/pdf/Data%20Sheets/Atmel%20PDFs/ATmega16U4.32U4.pdf
加速度センサ / ジャイロ	Acceleration / Gyro sensor MPU-6050	http://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/
音センサ	Sound Sensor SPI XCM6035P	http://www.buzzer.com.hk
スライダボリューム	SlidePotentiometers Alps RS30H121	http://www.alps.com/WebObjects/catalog.woa/HTML/Potentiometer/SlidePotentiometers/
明るさセンサー	Light sensor Everlight PT12-21C	http://www.everlight.com/file/ProductFile/PT12-21C-TR8.pdf

- RDC-103 には、いろいろな文字や記号が描かれていますが、大きく分けると、センサコネクタ、モータコネクタ、電源コネクタ、USB コネクタの 4 つです。

デジタル入出力 Digital in/out

入出力端子を使用したい時はピンで接続します。

ピン番号	記号	解説
13		サーボ / 白色 LED / PWM 出力可能 R/C servo motor / White LED / PWM output
12		サーボ / ボタン R/C servo motor / Button
11		サーボ / 超音波 / 赤外線 LED R/C servo motor / UltraSonic / InfraRed
0		サーボ / ブザー / シリアル RX R/C servo motor / Buzzer / Serial RX
1		サーボ / LCD RS / シリアル TX R/C servo motor / LCD RS / Serial TX
10		サーボ / LCD CS / PWM 出力可能 R/C servo motor / LCD CS / PWM output
6	M1 (0.5A 程度 / 1 個)	サーボ / M1 PWM 制御 / PWM 出力可能 R/C servo motor / M1 PWM control / PWM output
5		サーボ / M1 制御 / PWM 出力可能 R/C servo motor / M1 PWM control / PWM output
4		M1 制御 M1 control
7	M2 (0.5A 程度 / 1 個)	M2 制御 M2 control
8		M2 制御 M2 control
9		M2 PWM 制御 / PWM 出力可能 M2 PWM control / PWM output
電源コネクタ		
	V1	電源コネクタ Power Conctor
	-	電源スイッチ Power Swith

RESET スイッチ

増設可能
★超音波センサ Ultrasonic sensor (別売品) 差し込んで使用します。



赤外線 LED
Infrared Sensor



明るさセンサ Light sensor

発光 LED 白色 / 受光 LED

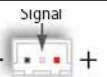
温度コネクタ 3 SCL, 2 SDA

アナログ入力 Analog input

A1 (A は Analog の A) と A2 の 2 ポートがあります。0 から電源電圧 (3.3V) までの入力電圧を 1024 段階で読み取ります。センサやボリュームなどを接続することができます。また、スケッチで設定を変更するとデジタル入出力ピンとして使うことができます。

ピン番号	記号	解説
A0	A0	音センサ Sound Sensor
A1	A1	アナログ入力コネクタ Analog input
A2	A2	アナログ入力コネクタ Analog input
A3	A3	みの虫クリップ Signal Clip terminal
A4	A4	明るさセンサ Light Sensor
A5	A5	スライダ Slider

接続コネクタ JST connector XH3B



LED

記号	解説
ON	青色 電源確認 Blue LED
RX	赤色 通信確認 Red LED
TX	緑色 通信確認 Green LED

加速度 / ジャイロセンサ (I2C) accelerometer

ボタン button

スライダ (可変抵抗) Slider resistance

みの虫クリップ用端子 Terminal for clips

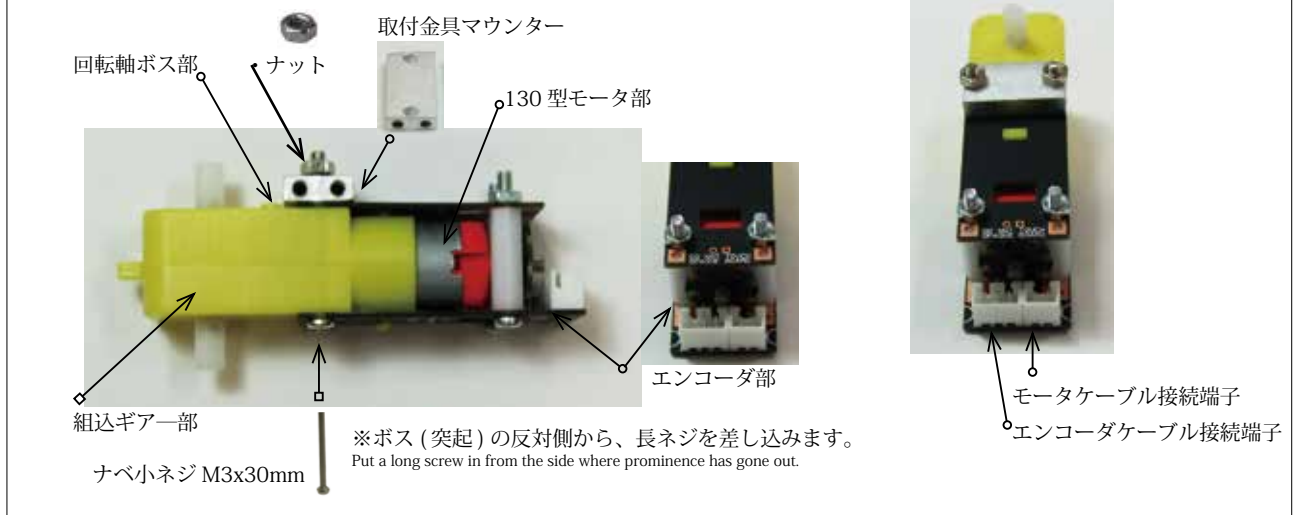
(抵抗等を測ります)

音センサ Sound sensor

USB コネクタ USB connector

11-3. 走行台車組み立て [Assemble of a Vehicles]

11.3.1. エンコーダ付ギアードモータ構造図



11.3.2. 走行台車基部組み立て

1. ギアードモータの準備

①マウンター取付

- ・検査であらかじめつけてある取付金具マウンター片方のナットを外し、天地さかさまにして、取り付けます。
- ・図のように配置したときにマウンターネジ取付穴が上を向くようにし、左右を対称にします。

②左右連結

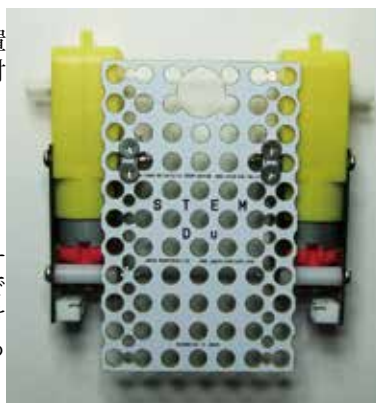
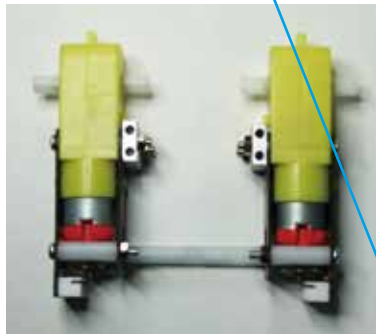
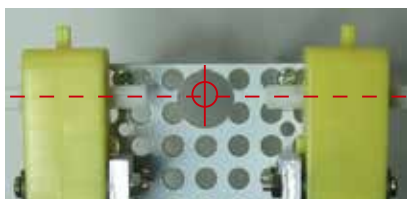
- ・エンコーダ部ナットを左右ともに1か所外し、作成したスタビライザーを図のように取り付けます。

2. シャーシプレート取付


- ・右図シャーシプレートの赤丸位置にマウンター位置を合わせて取り付けます。

	使用部品名	使用数
	超低頭ネジ Low Head Machine Screw M3x4mm	4本

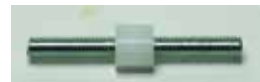
- ・シャーシプレートの穴が、ホイールシャフト軸上になることが重要です、位置がずれないように取り付けます。ロボット回転時の中心になる位置です。



[準備] スタビライザー作成

	使用部品名	使用数
	全ネジ M3 × 30mm	1本
	樹脂スペーサ Resin spacer M3x20mm	2個
	樹脂スペーサ Resin spacer M3x5mm	1個

- ①全ネジに樹脂スペーサ 5mmをはめ込み中央までねじ込みます。



- ②両側から、樹脂スペーサ 20mmをねじ込みます。

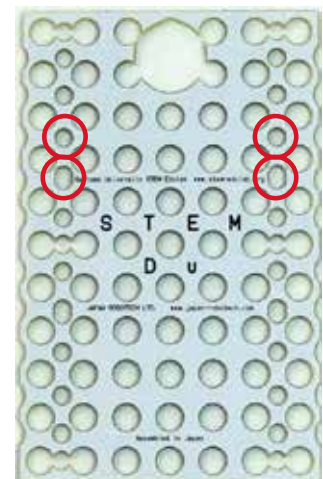


- ③完成したスタビライザー



仕上げ長さ 45mmに調整します。

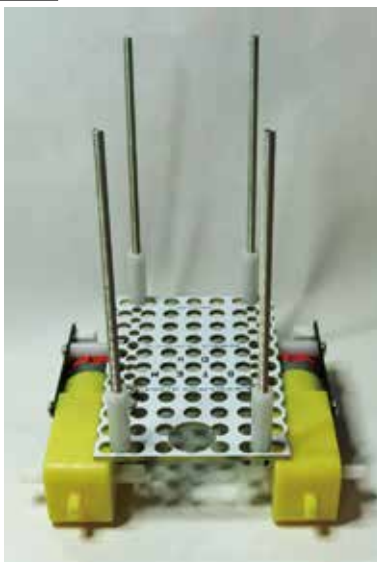
モータマウンター取付位置



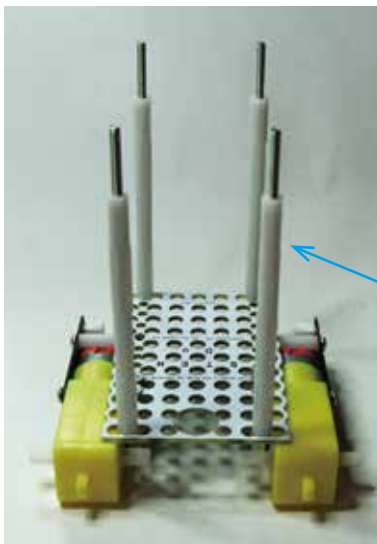
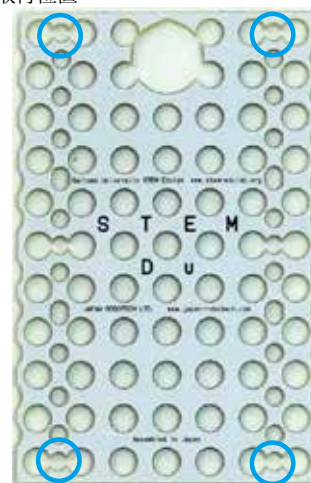
11.3.3. 支柱の組み込み [Assemble of a pillar]

1. 右図シャーシプレートの青丸位置に、支柱を組付けます。
 - 1.1 全ネジ95mmを差し込み、樹脂スペーサ20mmで固定します。
 - 1.2 樹脂製スペーサー20mmを、2段目と3段目に重ねてはめ込みます。
 - 1.3 樹脂製スペーサー15mmを4段目にはめ込みます。

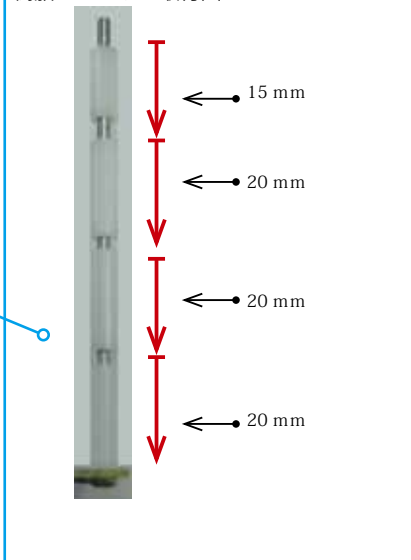
	使用部品名	使用数
	支柱：樹脂スペーサ Resin spacer M3x20mm	12 本
	支柱：樹脂スペーサ Resin spacer M3x15mm	4 本
	長ネジ Screw M3x95mm	4 個



支柱取付位置

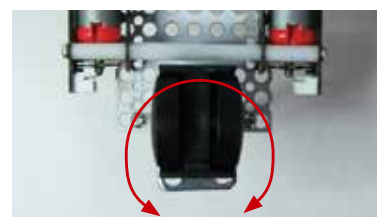
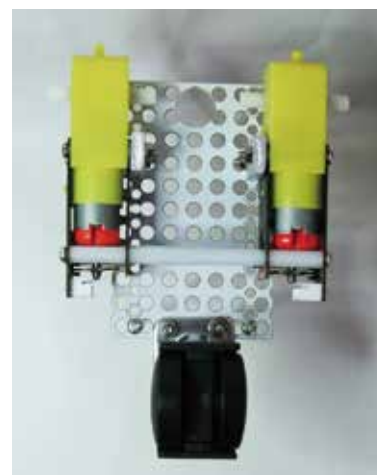


樹脂スペーサー取付図



11.3.4. キャスタ取り付け [Assemble of a caster]

使用部品名	使用数
双輪キャスター Caster 車輪径 30mm	1 個
座金付ネジ Screw M3x10mm	2 本
平座金 Washer	4 個
ナット Nut	2 個



🌀 回転します。

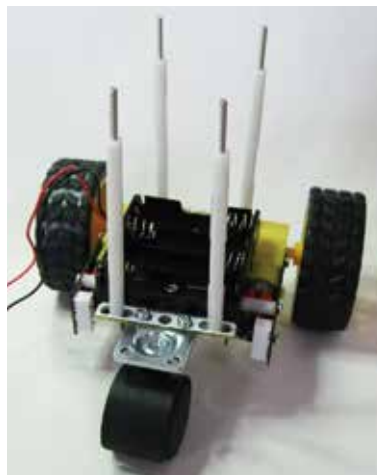
11.3.5. タイヤ取り付け [Assemble of a wheel]

- 車軸にタイヤホイールを差し込みます。
- ※回転軸（シャフト）を支えてホイールを差し込みます。ギアボックス内部ギアに無理な力を加えないようにご注意ください。



11.3.6. 電池ボックス取り付け [Assemble of a battery housing]

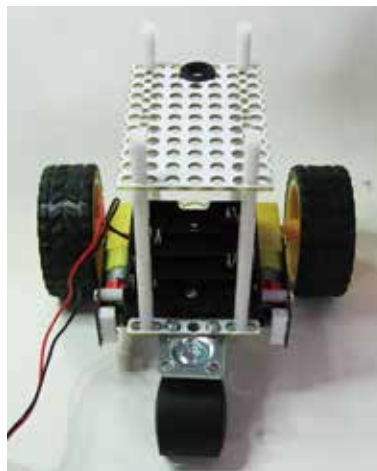
使用部品名	使用数
ネジを差し込んだ電池ボックス Battery housing	1 個
	
ナット Nut	2 個
	



11.3.7. ペンホルダー取り付け [Assemble of a pen holder]

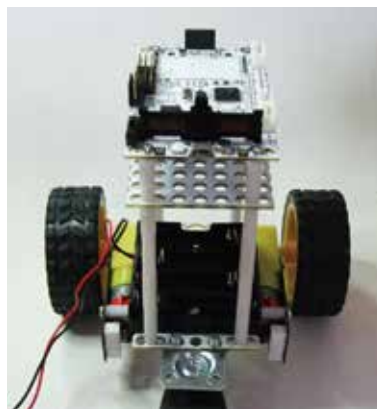
使用部品名	使用数
支柱：樹脂スパーサ Resin spacer M3x20mm	4 本

- 準備で作製したペンホルダーを取り付けます。



11.3.8. コントローラを取り付け [Assemble of a controller]

使用部品名	使用数
超低頭ネジ Low Head Machine Screw M3x4mm	4 本



準備 電池ボックスにネジを取付

Attaching a screw in the battery housing.

	使用部品名	使用数
	電池ボックス Battery housing 単 3 x 4 本	1 個
	皿ネジ Flat Head Screw M3x 1 0 mm	2 本



表面 裏面

ネジ取付位置

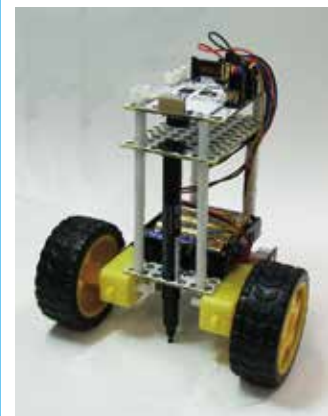
- M3x10mm 皿ネジは、電池ボックスに同梱しています。

準備 ペンホルダー作製

グロメットをシャーシプレートの穴へ差し込みます。



【高さ調整】サインペンの長さに応じて、ペンホルダー高さを調整します。
描画用サインペン：描画させる時に、サインペンを差し込み使用します。テストは Tombow ツインマーカー 33 クロを使用しました。描画用筆記具を変更する場合は、長さに応じてペンホルダーの高さを変更してください。描画する台紙に対し、少し筆圧が必要です。
高さの調整ができるように支柱の芯をネジ式にしています。



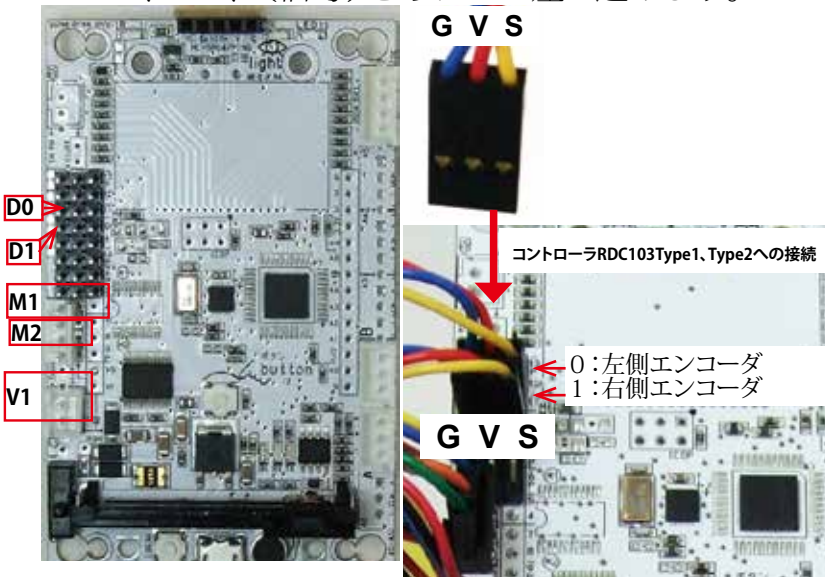
スパーサ長さを変更したり、ネジを回したりして高さの微調整ができます。

11.4. 配線

1. 下記の接続一覧表を参考に、各部品とコントローラの配線をします。

部 品	コントローラボード	接 続
左側モータエンコーダ Encoder Left side motor	D0	エンコーダケーブル** Encoder Cable
左側モータエンコーダ Encoder Left side motor	D1	エンコーダケーブル** Encoder Cable
左側ギアードモータ Geared motor Left side	M1	モータケーブル* Motor cable
右側ギアードモータ Geared motor Right side	M2	モータケーブル* Motor cable
電池ケース(単3 x 4) Battery housing (AA battery x4)	V1	赤/黒ケーブル Red/Black cable

** エンコーダケーブルには、極性があります。写真を参考にG ⊖、V ⊕、S(信号)をあわせて差し込みます。



**エンコーダーの使用

エンコーダはモータ回転軸に設置、取得データスピードが速いため、コントローラでは割り込みでカウントします(ON/OFF)デジタル端子0 と 端子1 を利用します。

コントローラRDC103Type 3、Type3+への接続 →テクニカルガイドを参照ください。

端子0段がM3モータ端子と競合のため、拡張用ケーブルを使用し、基板の改造を行います。

***モータ極性** モータには、回転方向を決める極性がありますが、モータケーブル先端には、どちら方向にでも差し込める無極性のコネクタを付けています、この段階での配線は極性を気にせずに接続します。後ほど、実動テストを行うときにロボットの動きを確認調整します。

2. 接続に間違いがないかどうか再確認後、電池ケース底面に記載されている極性表示に合わせて電池を実装します。

3. プログラムの準備ができるまで、電源スイッチを切っておきます

•この章では、PID制御ロボRDS-X25標準搭載のコントローラRDC103Type1、Type2へエンコーダを接続する方法を記述します。
•RDC103Type3などの他のコントローラへ接続する場合は、テクニカルガイド章_エンコーダ付DCモータの節を参照ください。

WIRING

1. Connection wiring of each part and a controller is done by making reference to the following connection list.

*Motor has the polarity which decides the direction of rotation, but polarity-less connector which can be put in every way is being put on the cable point.

Without worrying about polarity, wiring at this stage is connected.

When doing an actual working test later, the movement of a robot is confirmed and insertion is adjusted.

2. After reconfirming whether it's without mistakes in a connection, a battery is mounted according to the polarity indication indicated on a battery housing base.

3. Power supply switch is cut until We'll be ready for a program.

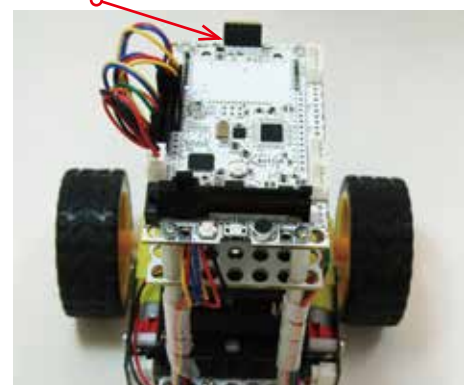
ATTACHING CONTROLLER BOARD

It's used for the purpose that the "slip" tape which is being stuck to an ultrasonic sensor socket side takes out the slip effect which is at the time of floor surface contact.

The sensor page when using a this machine as a Linear tracing robot, and the tape to decide a gap adjustment with resting face.

Please be careful so as not to remove.

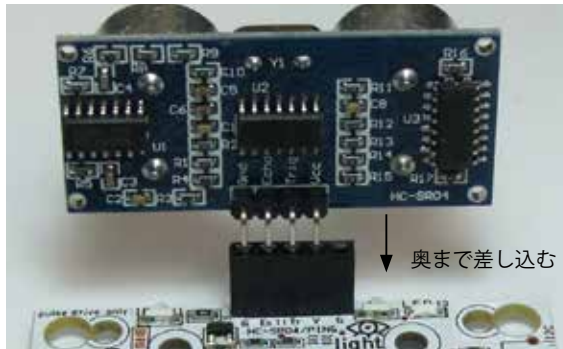
※超音波センサソケット側面に貼り付けている「すべり」テープは、床面接触時のすべり効果を出す目的とともに、本機をライトレースロボットとして使うときのセンサー面と床面との隙間調整を決めるためのテープです。取り外したりなさらないようご注意ください。



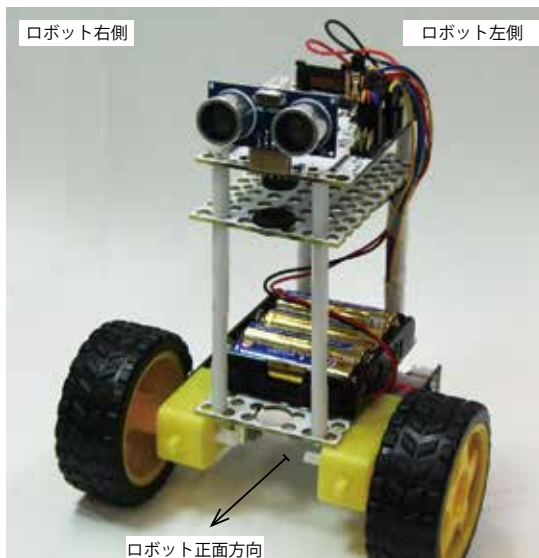
11-5a. 超音波障害物回避ロボ機体

- ・ センサ基板の超音波距離センサ取り付け用ソケットの穴に、表示を合わせてセンサーピンを差し込みます。

超音波 HCSR04 センサ端子表示	RDC-103 ソケット端子表示
Gnd	G
Echo	Ec11
Trig	Tr
Vcc	V
—	G



超音波センサ取り付け後



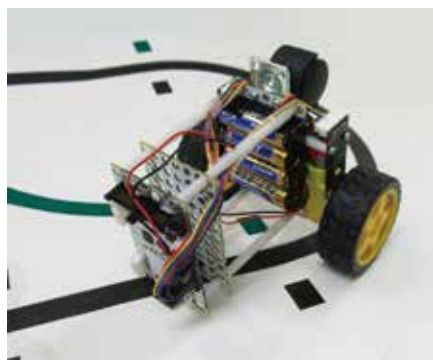
ATTACHING SENSOR

Put a sensor pin in a hole of a socket for ultrasonic range sensor installation of a sensor substrate.

Make a sensor terminal indication symbol and a controllerboard terminal indication symbol agree.

11-5b. ライントレースロボ機体

1. 前項で作成した「超音波障害物回避ロボ」から、超音波センサ HC-SR04 を取り外します。
2. 説明写真のように、コントローラ先端の [明るさセンサ] を床面に向けて、ライントレースロボとして使います。明るさセンサの仕様を考慮し、床面からセンサ面までの距離を 3 mm に設定して構造計算をしています。
3. 搭載センサを利用し、構造を他の方法で組み立てられる場合は、センサ距離を 3 mm 前後になるように調整ください。



Assembly of an Linear tracing robot

1. Ultrasonic sensor HC-SR04 and a rear caster are removed from the "ultrasonic obstacle avoidance robot" made by the preceding clause.

2. [Light sensor] in a controller point is used as a linear trace robot for resting face.

The specification of Light sensor is being considered, the distance from resting face to the sensor face is set as 3mm and structure calculation is being done.

3. When We can use equipped sensor and construct the structure by other ways, please adjust the sensor way as it'll be about 3 mm.

11-6. ロボットの動作確認

11.6.1. コントローラボードの電源スイッチを ON にして、電源を入れます。

11.6.2. 開発環境起動の事前準備…COM ポート番号の確認

1. PCとマイコンボードを、マイクロUSBケーブルで接続します。
2. PCがマイコンボードを感知し、PC側の「COMポート」が設定されますので、次の手順に従い、COM番号を調べてください。
 - ・Windows：マイコンコンピュータ>スタート>コントロールパネル>ハードウェアとサウンド>デバイス マネージャーの順で開いていき、[ポート (COMとLPT) COM]を見つめます。
3. (COMとLPT) COM]に認識出現している STEM Du RDC-102(COM番号)を確認し、COM記号の後ろにある数字(ポート番号)をメモします。
4. プログラム転送処理時にCOM番号が必要となります。

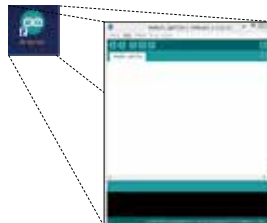


※プログラムが書き込めない場合は、必ずCOM番号を確認してください。

11.6.3. プログラム開発環境の起動

1. デスクトップに作成したショートカット arduino.exe をダブルクリックして arduino を起動します。

Fig. 起動中の arduino 画面



2. Arduino-IDE[ツール]で[マイコンボード]、[シリアルポート]を確認します。

[マイコンボード]: Arduino-IDE の [ツール]▷[マイコンボード] をクリックし、出現するマイコンボードリストで、[STEM Du/RoboDesigner+ RD C-102 w/ ATmega32U4 3.3V 8MHz] を選択・クリック指定を行います。
リスト左端に●印が付きます。



指定を間違うと、マイコンボードが誤動作します。

[シリアルポート]: Arduino-IDE の [ツール]▷[シリアルポート] をクリックして、出現するサブウィンドウで、デバイス マネージャーで調べた COM 番号の通信ポートをクリック指定し、☑マークがついたことを確認します。

指定を間違うと、通信ができなくなります。



エラーメッセージ

Couldn't find a Leonardo on the selected port. Check that you have the correct port selected. If it is correct, try pressing the board's reset button after initiating the upload.

通信エラーが発生し、マイコンボードへの書き込みが失敗しています。Arduino ⇒ [ツール] ⇒ 「マイコンボード」RDC-102 に●マーク、「シリアルポート」接続COM番号に☑マークがついていることを確認してください。

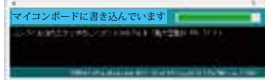
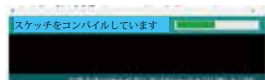
・アップロードがうまくいかない場合は、コンパイルの後、マイコンボードに書き込みが始まる前に、RDC-103 コントローラの RESET スイッチを「ダブルクリック」してください。



↑ RESET



このタイミングです。☑



Operations check of a robot

1). Power supply switch of a controller board is turned on.

2). Preliminary preparations of a development environment start--- The communication port number is confirmed.

1.PC and a microcomputer board are connected by Micro USB cable.

2.PC senses a microcomputer board, and "communication port" on the PC side is established, so please check the COM number with the next procedure.

* Windows : My computer Start ▷ A Control Panel ▷ Hardware and sound ▷ It's being held by the order of the device manager and [port COM(COM and LPT)] is found.

3. You check STEM Du RDC-102 from which recognition emerges in [(COM LPT) COM] (portnumber), and take notes of the number which is behind the COM symbol (portnumber).

4. The COM number is needed at the time of program upload.

※ When a program can't be written in, please be sure to confirm the COM number.

3). Start of a program development environment.

1. The short cut made in a desktop arduino.exe is double-clicked and arduino is started.

2. [Microcomputer board] and [serial port] are confirmed by Arduino-IDE [tool].

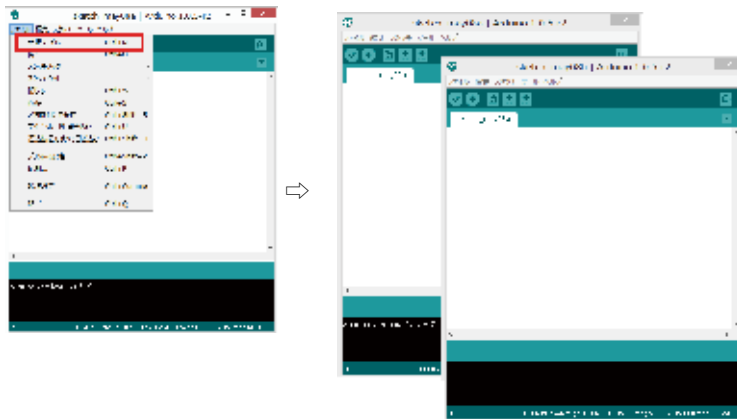
[Microcomputer board]: of Arduino-IDE [Tool] ▷ [STEM Du/RoboDesigner+ RD C-102 w/ ATmega32U4 3.3V 8MHz] is chosen by the microcomputer board list which clicks [microcomputer board] and appears, and click designation is performed. A ● mark sticks to the list left end.

When you make a mistake in designation, a microcomputer board malfunctions.

COM checked by a device manager a short while ago by a subwindow A communication port of the number is designated and it's confirmed that a ☑ mark stuck.

When you make a mistake in designation, you can't communicate any more.

3. Arduino-IDE の [ファイル] > [新規ファイル] をクリックします。新規ファイル [Sketch. 日付] が作成されます。



3. [File] of Arduino - IDE [It's filed newly.] is clicked. A new file [Sketch. Date] is made.

4. 新規ファイルの arduino-IDE で、[ツール]にある [ArduBlock] をクリックします。



4. [ArduBlock] in [tool] is clicked in arduino -IDE of a new file.

5. ArduBlock がスタートします。



5. ArduBlock starts.

A movement test program is made.

1. Make the program to which you just move while consulting a figure above-mentioned.

* An example uses "forward", but it's possible to test by other movements.

* 255 is the greatest for the motor speed. It moves by the quite fast speed, so the beginning, it's rather a little late, I think it's good. You'll designate the speed which is about half.

2. "It's uploaded to Arduino.", it's done.

3. Arduino-IDE compile, an executable file-ized and write in a controller board in Arduino-IDE.

4. Operating state is shown to the lower part message screen of Arduino-IDE.

"I have finished writing notes in a microcomputer board."

11.6.4. 動作テストプログラムを作成します。

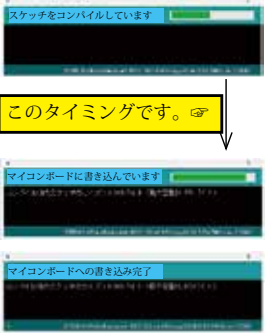



- 上記の図を参考にしながら、前進するだけのプログラムを作成します。
 - 例は、「前進」ですが、他の動きでテストを行なうこともできます。
 - モータスピードは 255 が最大値です。かなり早いスピードで動きますので、最初は、半分くらいの少し遅めが良いと思います。
- 「Arduino へアップロード」します。
- Arduino-IDE では、コンパイルを行い実行ファイル化され、コントローラボードに書き込みます。
- Arduino-IDE の下部メッセージ画面に、動作状態の表示がされます。「マイコンボードへ書き込みが完了しました。」とメッセージが表示されると、書き込み完了です。



エラーメッセージ Couldn't find a Leonardo on the selected port 出現時
通信エラーが発生し、マイコンボードへの書き込みが失敗しています。
 Arduino ▷ [ツール] ▷ 「マイコンボード」RDC-102 に●マーク、「シリアルポート」接続 COM 番号に□マークがついていることを確認ください。

・アップロードがうまくいかない場合は、コンパイルの後、マイコンボードに書き込みが始まる前に、RDC-103 コントローラの RESET スイッチを「ダブルクリック」してください。

When a message is indicated, it's writing in completion.

Couldn't find a Leonardo on the selected port. Check that you have the correct port selected. If it is correct, try pressing the board's reset button after initiating the upload.

* When upload doesn't work, you double-click the RESET switch of RDC-103 controller.

A movement start and all kinds of movement check a first robot.

[1]. Execution of the program forwarded to a substrate

(1). Robot movement preliminary confirmation of preparations

1. A production is moved and it's checked to confirm the state of the connection of the robot.

2. It's confirmed whether an input/output device is connected to the following setting street.

3. It's confirmed that [circuit power supply switch (SW)] becomes off.

4. It's confirmed that a battery housing contains a battery.

(2). A program is executed: [Power supply switch (SW)] is turned on.
 A program is executed.

(3). Program stop: [Circuit power supply switch (SW)] is turned off.

11.6.5. はじめてロボットを動作スタート、各種動作点検

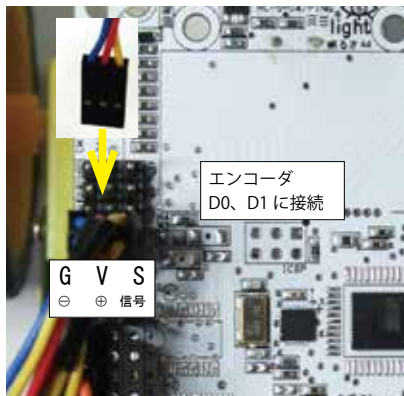
(1). コントローラボードに転送したプログラムの実行

1. ロボットの接続状態を確認するために実機を動かして点検を行います。
2. 下記の設定通りに入出力機器の接続がなされているかを確認します。

部 品	コントローラ	接続
左側モータエンコーダ Encoder Left side motor	D0	エンコーダケーブル * Encoder cable
右側モータエンコーダ Encoder Right side motor	D1	エンコーダケーブル * Encoder cable
ギアードモータ 左側 Geared motor Left side	M1	モータケーブル ** Motor cable
ギアードモータ 右側 Geared motor Right side	M2	モータケーブル ** Motor cable
電池ケース (単 3 × 3) Battery housing (AA battery x 3)	V1	赤 / 黒ケーブル Red/black cable

3. [回路電源スイッチ (SW)] が OFF になっていることを確認します。
4. 電池ケースに電池が入っている事を確認します。

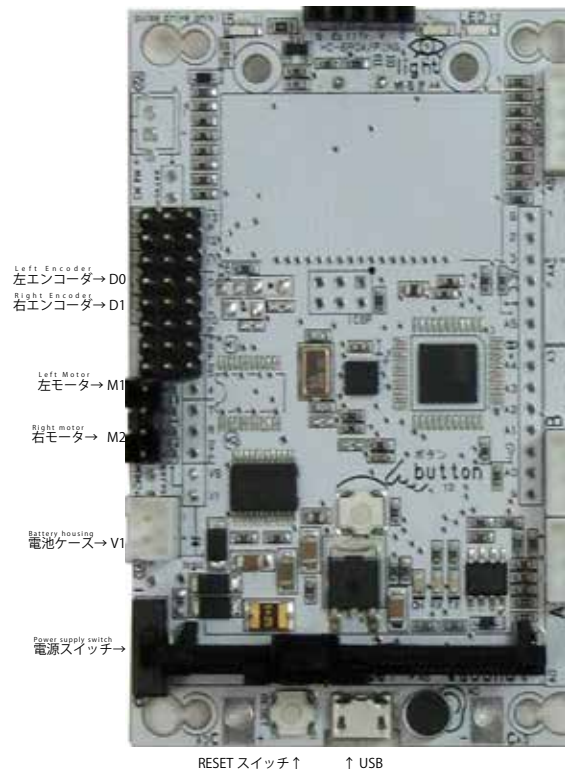
* エンコーダケーブルには、極性があります。下の写真を参考に G ⊖、V ⊕、S (信号) をあわせて差し込みます。



** モータには、回転方向を決める極性がありますが、モータケーブル先端には、どちら方向にでも差し込める無極性のコネクタを付けています、この段階での配線は極性を気にせず接続します。実動テストでロボットの動きを確認して、差し込みを調整します。

(2). プログラム実行: [電源スイッチ (SW)] を ON にします。

(3). プログラム停止: [RESET スイッチ (SW)] を押します。



(4). 初めてのロボット動作点検

- (1). 「前進」プログラムを実行し、ロボットの動作をスタートして、前進するかどうかを確認します。・配線の間違いを含めて、最初の重要な確認ですから、注意して観察しましょう。

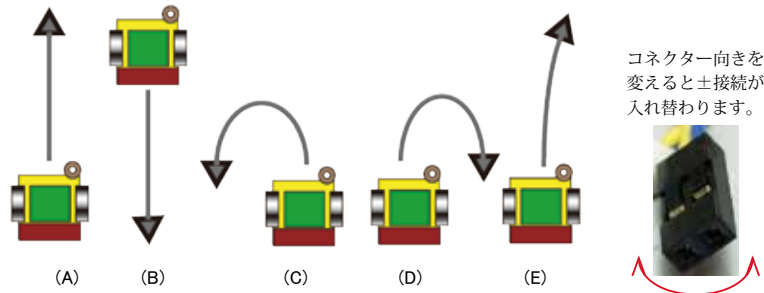


Fig.9.4.5 動作点検 Movement check

(2). 動きに応じた対策

- (A)の動き： 正しく動いています。
 (B)の動き： M1,M2ともに配線の極性(プラス⊕・マイナス⊖)接続間違いです。
 ⇨M1,M2ともにコネクタの向きを反対にして(±接続を入れ替えて)、接続し直してください。
 (C)の動き： M1モータ配線の極性(プラス⊕・マイナス⊖)接続間違いです。
 ⇨左側モーターのコネクタの向きを反対にして(±接続を入れ替えて)、接続し直してください。
 (D)の動き： M2モータ配線の極性(プラス⊕・マイナス⊖)接続間違いです。
 ⇨右側モーターのコネクタの向きを反対にして(±接続を入れ替えて)、接続し直してください。
 (E)の動き： 正しく動いています。左右のモータ個体差の影響で、許容範囲内です。

- *1. 左右モーターの個体差が影響し、完全にどこまでもまっすぐ進むことは、大変難しいことです。
 * 長い距離を走らせると左右のどちらかに少しずつ曲がっていきます。
 ** 違う構造では、たとえば1個のモータの両側に車輪を取り付け走らせるなどの実験を行うとまっすぐ進みますが、プログラムにより自在に方向転換ができなくなります。
 *** 解決策としてモータの個体差をなくす方法も考えられますが、数万個の生産の中から個体差が少ないモータを探し出すことになり、大変高価なコストになります。
 **** 今回取り組んでいるPID制御では、PWMを制御することで直進性を得ます。



RDC-103 コントローラは、電源が入ると、すぐにプログラムが実行されます。言い換えれば、電源スイッチをONにするとロボットの動作が開始します。ロボットが動作しても安全な位置に置いて、動作開始させます。
 ・机の上に置いたまま、電源を入れるとロボットが動き出し、床に落下して衝撃で壊れるなど事故が起こりますので、台座の上に置くなどロボットの設置場所に十分配慮して、ロボットを動かします。

[2]. First robot movement check

1. A program is executed and movement of a robot is started, and it's confirmed whether you move ahead.
 * Because it's the first important confirmation you put into effect including a mistake of your wiring, please check.

2. Measure according to the movement
 (A): It's moving right.
 (B): A polarity (plus minus) connection mistake of a motor wiring terminal.
 ⇨ Please replace a + - connection of the left side motor and connect again.
 (C): A polarity (plus minus) connection mistake of a motor wiring terminal.
 ⇨ Please replace a + - connection of the right side motor and connect again.
 (D): It's moving right. It's influence of the motor individual difference in the left and right and is in the latitude.

- *1. Is the individual difference in the motors of left and right influential and is it that it's very difficult even to advance where straight perfectly?
 * The long distance, dispatch if I'll turn to one of them in left and right a little.
 ** For example a wheel is installed in both sides of 1 motor by the different structure, dispatch of when an experiment is made, I advance straight, but you can't turn any more freely by a program.
 *** The way to lose the individual difference in the motors as a countermeasure is also considered, but the individual difference will find a little motor from the inside of tens of thousands of of production, and it'll be the very expensive cost.
 **** The autonomous robot on which you're working this time gets sensor information and changes the constancy direction based on a program, and moves, so a straight line by the long distance won't be a problem big, so please be relieved.

When RDC-103 controller is turned on, a program will be executed immediately.
 When power supply switch is turned on, movement of a robot begins it.
 Even if movement is begun, put a robot in the safe location and let me begin to move, please.

11-7. ライントレースロボのプログラム

11.7.1 条件分岐プログラム

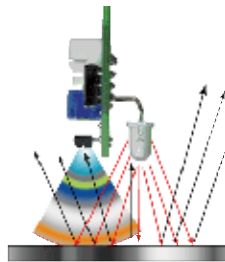


Fig. IR センサ原理図

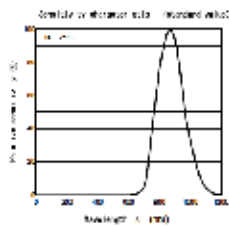


Fig. IR センサ特性図

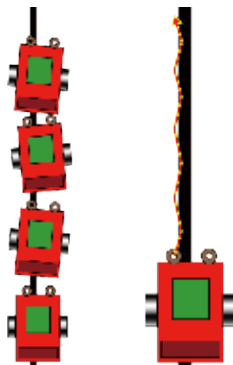
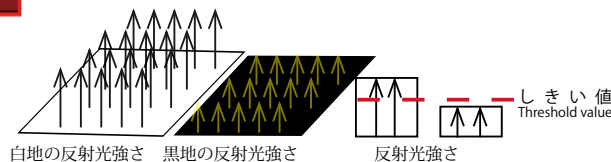


Fig. トレースイメージ図

1. ロボットがどのような方法で黒線を検知し、停止行動をしようとして動いているのか・・・そのアルゴリズムを理解しておきましょう。
2. ロボットに取り付けたコントローラには、発光用LEDと受光用モジュールがあり、左図のように赤外線を発射して、反射光の強さを計測します。
3. 赤外線センサは、フォトIC（光起電力素子）を使用した受光センサです。光の波長の中でも赤外線周辺の帯域をもっともよく検出します。
4. 光の量で出力電圧が変化しますから、光源からの距離を計れば「距離センサ」として、色の反射率を計れば床の図形などの「読み取りセンサ」として使うことができます。より単純に、デジタル入力的に赤外線の「ある・なし」の判定に使うこともできます。
5. このように赤外線センサは届いた光を電気信号に変換してロボット本体へ情報を送ります。
6. コントローラボードは、プログラムに従い、反射光の強さがどのくらい以上であればモータを回転し、以下であれば、モータを停止するなどの動きをします。
7. 光の性質として、白い部分からは強い反射光があり、黒い部分は光を吸収して反射光は弱くなります。動作実験には、白色の床に描かれた黒色の線などを使用する方が、反応が大きく変化しますので、計測がしやすくなります。
8. 反射光の強さの違いによるアナログ赤外線センサの出力の変化を計測して、白なのか、黒なのかを判断し、動くことが必要で、この分岐条件になる値を、「閾（しきい）値」と呼びます。



- ・ロボットが動く環境下で計測して、白色・黒色それぞれの反射光の強さを計測します。センサ値計測のソフトを使ってコントローラで測りません。
- ・白色反射光データ平均値と黒色反射光データ平均値の中間をしきい値（分岐条件）として決定しプログラムに書き込みます。

エッジ走行・・・境界の縁をたどります

分かりやすい一言で説明すると「エッジ走行」です。

- 線上にある。黒から白へ左斜め走行
- ↑
- 線上に無い、白から黒へ右斜め走行
- ↑
- 線上にある。黒から白へ左斜め走行
- ↑
- 線上に無い、白から黒へ右斜め走行

ラインは、直線だけでなく、曲線でもたどれます。

Program of a Linear tracing robot

Conditional branch program

1. A black line will be detected and you're going to do stop behavior, and it's moving or... by what kind of way does a robot understand the algorithm?

2. A controller has an LED for emission of light and a module for receiving light, and infrared rays are launched like a left figure and the strength of the catoptric light is measured.

3. Infrared radiation sensor is a receiving light sensor with photo IC (photovoltaic device). One in the wavelength of the light detects a band in the infrared circumference most often.

4. Because the output voltage changes by the amount of the light, when measuring the distance from the light source, when measuring the reflectivity of the color as "range sensor", it would be possible to use it as "reading sensor" of a figure on a floor. More simply, it can be used for judgment of an infrared presence.

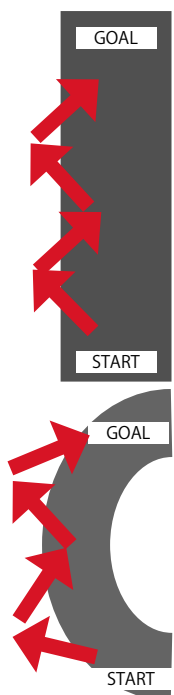
5. Infrared radiation sensor changes the light which has reached to an electronic signal and sends information to the robot main body this.

6. A controller board revolves around a motor according to the size of the sensor input value with a program, it stops, a movement is done.

7. There is strong catoptric light from a white part as the nature of the light, and a black part absorbs light, and catoptric light becomes weak. A reaction is big and the person who uses a drawn black line for a floor of the white floor changes into a movement experiment, so it becomes easy to measure.

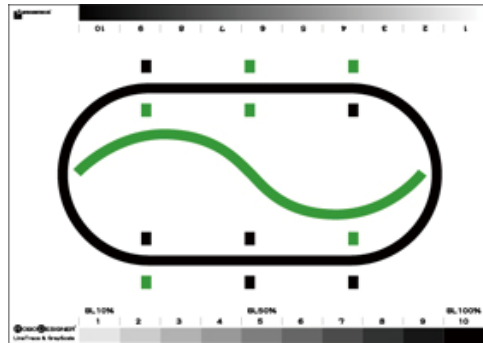
8. A change in output of analog infrared radiation sensor by the difference in the strength of the catoptric light is measured, a white one or a black one is judged, to move is needed and the numerical value which becomes this branch condition is called "the threshold value".

Please decide about the middle of the white catoptric light data mean and the black catoptric light data mean as a threshold value (branch condition) and write notes in a program.



11.7.2. ライントレースのプログラミング

1. 光の反射率により、白色と黒色で取り出せるセンサ出力数値に違いが出ることを利用し、ラインエッジをトレースします。
2. 白い大きなシートに、黒いテープでラインを引いて、そのエッジをたどりながら動くロボットを作ることができます。
3. 黒い線で大きな輪を作り、その一周を走るロボットに挑戦するのはいかがでしょう。
4. トラックを、うまく早いスピードで駆け抜けるロボット作りに挑戦してみてください。



ライントレースシート (A1 サイズ)

[2]. Programming of linear trace

1. That the difference goes out to the sensor output numerical value which can be taken out by white and black is used and a linear edge is traced by the reflectivity of the light.
2. A line is pulled to a white big seat by a black tape, and it's possible to make the robot which moves while following on it.
3. Make the robot which makes a big circle and runs through its one round with a black line.
4. Please try robot making which runs through a truck made of a big black line by the early speed well.

ソースコードは、PC/MyDocuments/Arduino/へ配置したサンプルフォルダ [ArduBlock Examples] に、ファイル名【21_02_Line_board-lightsensor_sample.abp】で格納されています。Arduino/ArduBlockで、PC/MyDocuments/Arduino/ArduBlock Examplesの中に配置したサンプルを開くと確認できます。



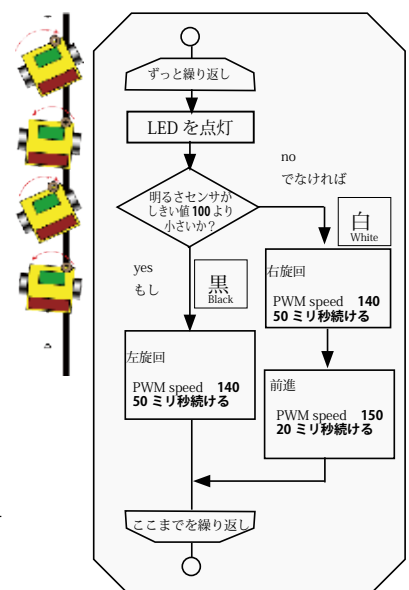
11.7.3. 上図を参考に、プログラムを作成してください。

プログラム中のBlock画像の「？」マークは、コメントがあることを知らせるアナウンスです。

明るさセンサを使用して床の白黒を判断しながら行動します。
 黒線上で、floor 1 (明るさセンサー)の出力が「しきい値 (分岐条件)」以下なら、左旋回します。
 でなければ、右旋回し前進します。
 変数 floor1 明るさセンサーの出力値を格納します。

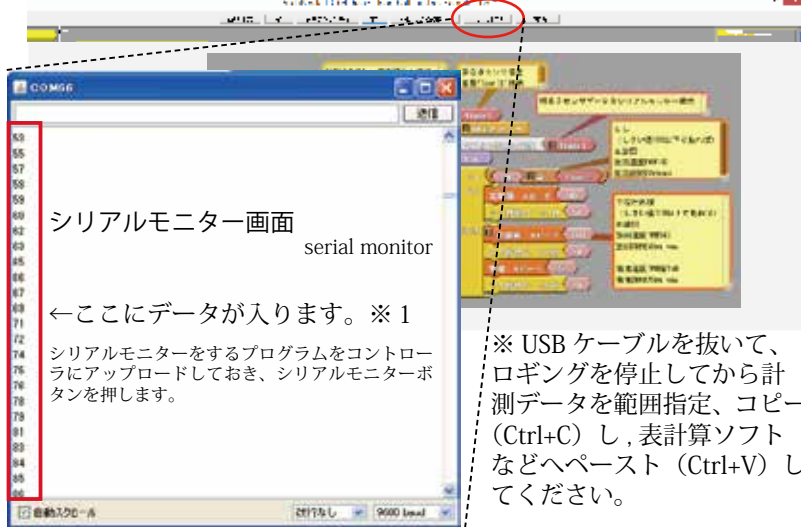
しきい値は、ツール>シリアルモニタを使って floor1 のデータ値を確認して設定します。
 シリアルモニタから計測データをコピー (Ctrl+C) して、表計算ソフトなどにペースト (Ctrl+V) してグラフ化処理すると分かり易くなります。
 ※投射している赤外線と比較して周囲が明るい「前進」と「旋回」をしきい値で分けることができません。

11.7.4. 右側のフローチャートは、上のサンプルプログラム ArduBlock_Line_board-lightsensor_sample をチャート図で表してみました。



11.7.5. プログラム調整ロボット作り込み

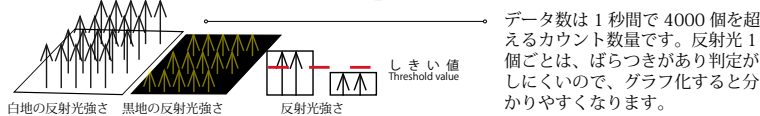
1. センサのデータを調べます。…USB ケーブルを接続して計測します。
 1. コントローラのプログラムが実行されている状態の時に、ArduBlock の [シリアルモニター] を使ってセンサの値を調べることができます。(シリアルモニターできるように sample プログラムを作成しています)
 2. ArduBlock の [シリアルモニター] をクリックするとシリアルモニター画面が立ち上がり、リアルタイムでセンサ値が表示されます。



3. シリアルモニターでセンサ値を確認しながら白床の上、黒線の上、それぞれデータ収集を行います。(USB ケーブルは接続のまま)
4. 計測したデータの間中値を、「しきい値」として分岐条件に使います。
5. 計測したデータをシリアルモニターからコピー (Ctrl+C)、表計算ソフトなどにペースト (Ctrl+V) して数値をグラフ化処理するとセンサー値傾向が分かり易くなり、分岐条件(しきい値) 考察がしやすくなります。…・参照：次頁 - 表計算ソフト使い方

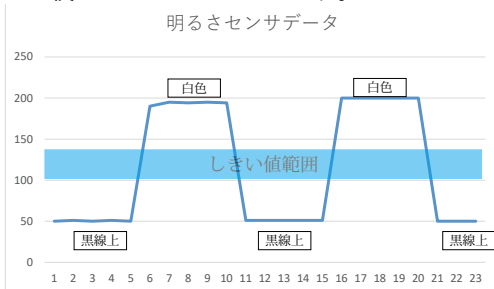
2. 各種パラメータ調整

1. 床面の色の違いで、センサに届く赤外線反射量が変化します。
 - ・シリアルモニターを使って、計測をしながら、ロボットを少しずつ移動していき、黒の線と、白色の床で、色の違いにより、どのように出力が変化するかを調べます。
 - ・データは、数度にわたって調べて、黒色、白色床それぞれに平均値を調べます。
2. センサデータ (黒色からの反射データと、白色からの反射データ) の違いを調べて、その中間値を「しきい値」とします。



3. 「表計算ソフトの利用」が便利です。

・以下は、例として、前頁ライントレースシートで、黒線と白色床上を交互に計測したシリアルモニターのデータを、グラフ化した図です。どれほどの信号の大きさか一目で理解でき、「しきい値」の検討などに役に立てることができます。



大きい値と小さい値の中間値位を「しきい値」とします。
図の例の場合、100 ~ 150 くらいが「しきい値」に適しています。

参照：次頁 - 表計算ソフト使い方

Data of a sensor is checked.... A USB cable is connected and measured.

1. It's possible to check output data of a sensor using [serial monitor] of ArduBlock at the state that a program of the microcomputer board in which "Line_base_floor_sample" was written is executed.
2. When [serial monitor] of ArduBlock is clicked, a serial monitor screen stands up, and the sensor value is indicated in real time.
3. They're on the white floor and a black line while confirming the sensor value by a serial monitor and a data collection is performed respectively. (For a USB cable, condition of a connection)
4. The value of the middle of the measured data is used as "threshold value".
5. When graphing makes spreadsheet software copy from a serial monitor, and deals with a figure, it becomes easy to understand.

All kinds' parameter tuning

1. With the color of the resting face, to be different, more, the amount of the infrared rays which reach a sensor undergoes influence of the reflectivity and changes.

* You're moving a robot a little and measure using a serial monitor, and it's the color in case of a black line and a white floor, please examine how output changes to be different.

* Please check data over several times and check a mean in each of black and white floor.

2. The intermediate value will check the difference in the sensor data (each reflectance data in a black part and a white part), and is "threshold value".

"Practice of spreadsheet software" is needed.

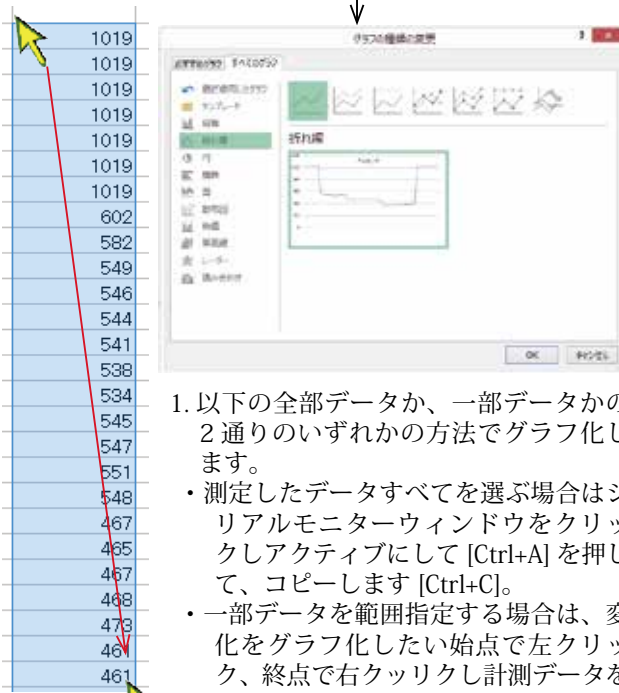
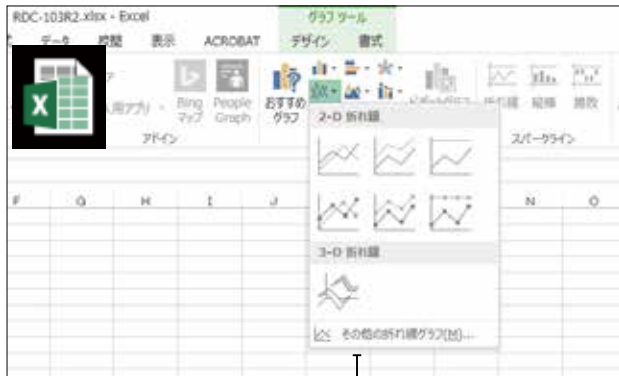
* Below is previous page linear tracing paper and is a figure which graphs data of the cereal monitor who measured black and white as an example.

It can be understood and is it possible to be able to be useful for consideration of "the threshold numerical value" by how much signal size or look?

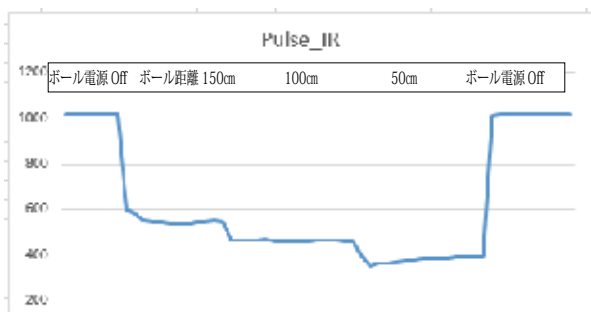
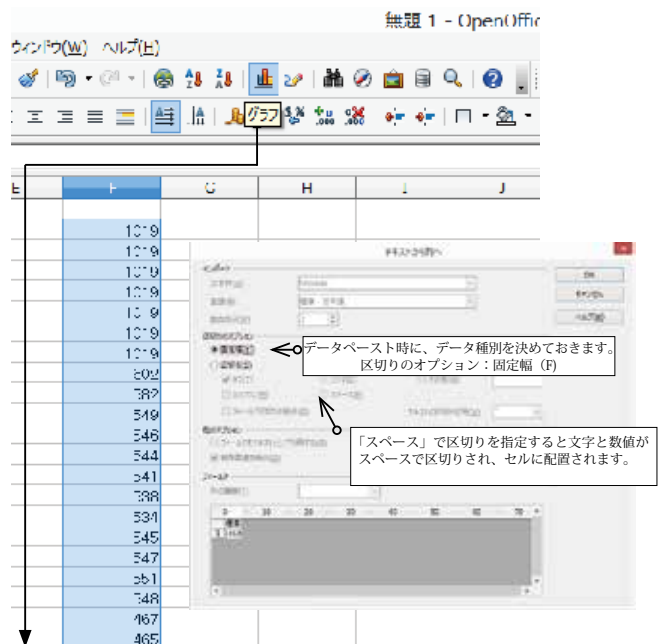
シリアルモニターを長い時間継続すると、取得データがオーバーフローし、PC がフリーズすることがあります。このような場合、データ取得を中止し、コントローラのダブルリセット、Arduino の再立ち上げを行ってください。お使いの P/C によっては、P/C の再起動が必要な場合もあります。When data overflowed, a PC is reset, and there is a case which needs a restart.

4. 表計算ソフトの使い方

- ・シリアルモニタのデータは、1秒で4000個超のカウント数になります。プログラムの分岐条件に使用するしきい値は取得データを「表計算ソフト」などを利用してグラフ化し、分岐点を考察します。
- ・代表的な表計算ソフトとして**EXCEL**(有料ソフトウェア)があります。その他無料でインターネットから入手できるソフトウェアの例として**OpenOffice**もあります。



1. 以下の全部データか、一部データかの2通りのいずれかの方法でグラフ化します。
 - ・測定したデータすべてを選ぶ場合はシリアルモニターウィンドウをクリックしアクティブにして [Ctrl+A] を押し、コピーします [Ctrl+C]。
 - ・一部データを範囲指定する場合は、変化をグラフ化したい始点で左クリック、終点で右クリックし計測データを範囲指定して、コピーします [Ctrl+C]。
2. 貼り付けたい表計算ソフトの位置 (先頭のセル) を指定して、ペーストします。 [Ctrl+V]
3. グラフ化したいデータを範囲指定して、グラフ化処理をします。
4. グラフの形などは自分が分かりやすい形を選びます。

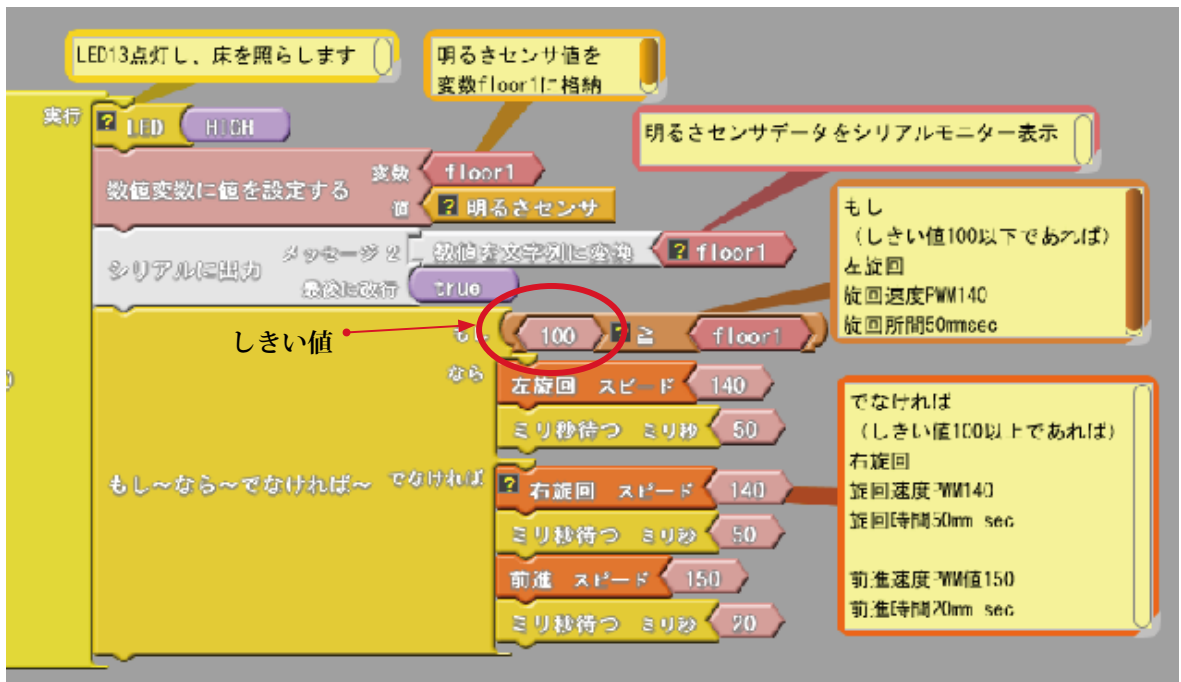


※シリアルモニタの出力を表計算ソフトで処理する場合は、変数の前のテキストをカンマやスペースに置き換えてください。**OpenOffice** の場合は、区切りのオプション「固定幅 (F)」を選択指定してデータをペーストしてください。ペースト時に出現するダイアログのメニューで、「スペースで区切り」を指定すると文字と数値が区別してセルに配置されます。

- ④ 図は、ロボカップジュニア公式競技ボールを変調赤外線センサで計測したデータです。
- ・グラフで見ると、距離ごとに計測数値が違うことも理解でき、プログラムの調整に役立ちそうです。
- ・ボールから離れているとき、中間距離時、近い距離など、ボールからの距離ごとにデータが違うことが分かり、プログラムで工夫をしてロボットの行動を変えることが可能です。

5. 調べたデータで、プログラムの分岐条件「しきい値」を書き換えます。 [パラメータ調整]

- サンプルでは、floor1の「しきい値」を100と仮に決めていますので、今回、調べた実測値に基づき「しきい値」を書き換えます。



・参考プログラム解説

ずっとくり返し

もし 100 (しきい値) \geq floor1 なら
左旋回 モータスピード 140 *50 ミリ秒待つ

でなければ (100 (しきい値) \leq floor1 なら)
右旋回 モータスピード 140 *50 ミリ秒待つ
前進 モータスピード 150 *20 ミリ秒待つ

>>>>>> 計測では、黒だと 50 前後、白であれば 200 前後のデータでした。

- 左図の場合で
黒線上で、明るさセンサが「しきい値 (分岐条件)」以下の数値になったら
PWM140 のモータスピードで左旋回を 50 ミリ秒間行います。----> ①
*[ミリ秒待つ] [ミリ秒] は、結合アイコンの動作時間を設定するパラメータで、継続動作するという意味です。
- 左旋回をすると、黒線上からセンサーが離れ、白色床上に到達するので、明るさセンサ出力が「しきい値 (分岐条件)」以上のデータに変化します。
明るさセンサが「しきい値 (分岐条件)」以上の数値になったら
右旋回を 50 ミリ秒間 PWM140 のスピードで行い
モータスピード PWM150 で 20 ミリ秒間前進します。-----> ②
①と②をくり返して黒と白を境界をたどりながら前へ進んでいきます。
白色床と黒線の違いがわかる「分かれ目 (しきい値) 分岐条件」を調べて、サンプルの明るさセンサ数値を変更することにより、現在実験している環境 (周囲の明るさなどの影響) 下で、黒線をトレースするロボットが作成できるようになります。
- 左旋回、右旋回の時間設定は、左図のロボット振れ幅を設定するパラメータです。
振れ幅を繰り返しながらラインの境界をトレースしますので、振れは必要です。
振れ幅が小さい方が、トレース時間は少なくなります。
- 左旋回、右旋回のモータスピード PWM 値は回転速度を設定するパラメータです。
- 前進のモータスピードと時間設定は、ライントレースコースを進む速さのパラメータです。
- プログラムとロボットの関係が、分かってくると、大変楽しくなります。いろいろなプログラム作りをして、思ったような動きにできるロボットを作りましょう。
- 周回トラックを作成し、3 周を何秒で完走できるかなど、周回時間を測ってみましょう。
- トラックが少し複雑なカーブを持っているなどのコース設定を行うことで、ライントレースプログラミングの難しさにもチャレンジしてみましょう。

5. プログラムを変更し、動きを変えてみます。

1. では、正面から進んでいき、黒線に差し掛かったら、停止して、ほぼ、180 度回転し、他の方向へ進んでいく、ロボットを作成してください。

11.7.6. コンパイル

21-02_ArduBlock_Line_board-lightsensor_sample を Arduino へアップロードし、C 言語へ自動変換したコード ArduBlock で作成したプログラムを「Arduino へアップロード」すると、Arduino IDE 画面に、変換された C ソースが出来上がります。

- 以下、前頁で作成した 21-02_ArduBlock_Line_board-lightsensor_sample を「Arduino へアップロード」し、Arduino IDE 画面に出来上がったソースコードです。
- 直接 C ++ 言語で、コーディングも可能です。・プログラムで使用できる文字は「半角英文字」と「半角数字」のみです。

Program source code.

```

ArduBlock_Line_board-lightsensor_sample | Arduino 1.0.5 r2-
ファイル 編集 スケッチ ツール ヘルプ
Arduino IDE
#include <STEMDU.h>

STEMDU _STEMDU_robot = STEMDU();
int _ABVAR_1_floor1 = 0;

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  _STEMDU_robot.led((bool)HIGH);
  _ABVAR_1_floor1 = _STEMDU_robot.readLight();
  Serial.print(_ABVAR_1_floor1);
  Serial.println();
  if (( ( 100 ) >= ( _ABVAR_1_floor1 ) ))
  {
    _STEMDU_robot.leftM1M2(140);
    delay( 50 );
  }
  else
  {
    _STEMDU_robot.rightM1M2(140);
    delay( 50 );
    _STEMDU_robot.forwardM1M2(150);
    delay( 20 );
  }
}
    
```

Upload

1. Upload-controller board writing in is performed automatically, but it's possible to upload manual.

* When choice clicks an upload button (->) icon of Arduino IDE, writing in to a microcomputer board is begun.

プログラムエラー

- プログラムタイピングミスの場合など、Arduino の該当行が黄色ハイライト表示で警告されます。
- プログラムで使用できる文字は「半角英文字」と「半角数字」のみです。全角文字は、プログラム文ではエラーとなり、該当行が黄色マークで警告されます。

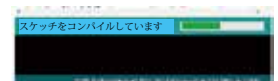


◆ エラーメッセージを確認して、対策を施して、問題を解決した後で、再度、マイコンへの書き込みを行います。

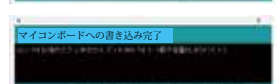
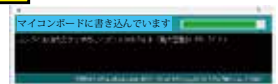
・アップロードがうまくいかない場合は、コンパイルの後、マイコンボードに書き込みが始まる前に、RDC-103 コントローラの RESET スイッチを「ダブルクリック」してください。



↑ RESET

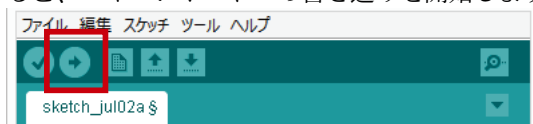


このタイミングです。



11.7.7. アップロード

1. アップロード～コントローラボード書き込みは、自動で行われますが、手動でもアップロード可能です。
 - Arduino IDE のアップロードボタン (→) アイコンを選択クリックすると、マイコンボードへの書き込みを開始します。



2. Arduino IDE 画面では、プログラムの変更、修正も行えます。
 - プログラムで使用できる文字は「半角英文字」と「半角数字」のみです。

11-8. 超音波障害物回避ロボのプログラム

11.8.1. 超音波距離センサー HC-SR04

- 超音波の反射時間を利用して非接触で測距するモジュールです。外部からトリガパルスを入力すると超音波パルス（8波）が送信され、出力された反射時間信号をマイコン（A r d u i n o等）で計算することによって距離を測ることができます。
- 特性を利用して、障害物回避ロボットなどにすることが可能です。

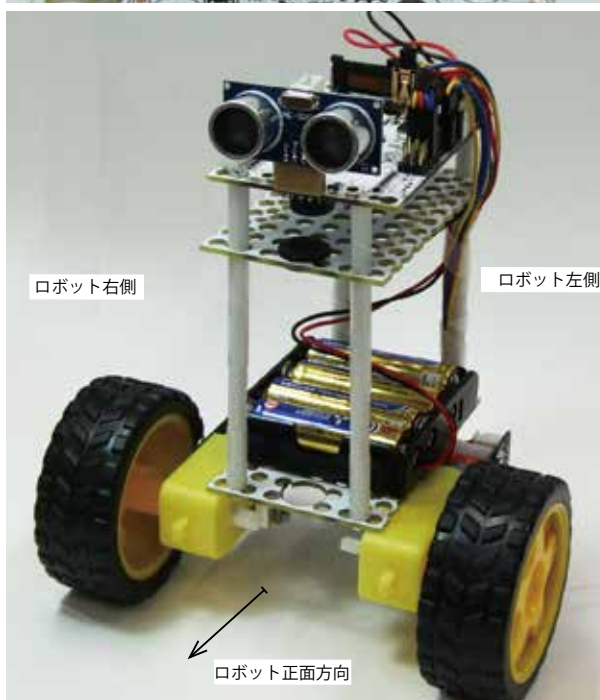
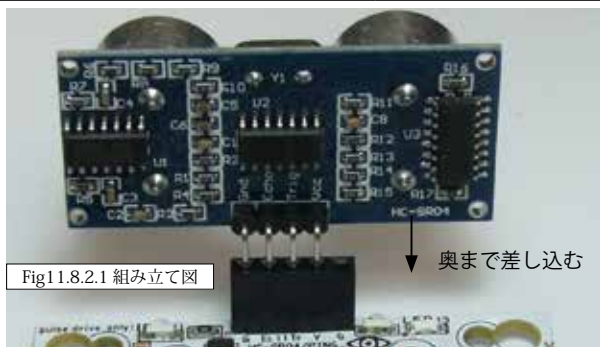
◆主な仕様

- 測距範囲：2～180cm
(センサー基板正面を中心とした15度の範囲、分解能：0.3cm)
 - 電源電圧：DC 5.0V
 - 動作電流：15mA
 - 動作周波数：40kHz
 - トリガ信号：10μs（TTLレベルのパルス波）
 - エコー出力信号：反射（往復）時間
 - サイズ：45×20×15mm
- ※通電時はGND端子が最初に接続されるようにしてください

11.8.2. 超音波距離センサー取付

- センサ基板の超音波距離センサー取り付け用ソケットの穴に、表示を合わせてセンサーピンを差し込みます。

超音波 HCSR04 センサー端子表示	RDC-103 ソケット端子表示
Gnd	G
Echo	Ec11
Trig	Tr
Vcc	V
—	G



Ultrasonic distance sensor HC-SR04

* It's non-contact using the ultrasonic reflective hour and is the module which does ranging.

When a trigger pulse is input from outside, ultrasonic pulse (8 wave) is sent, and it's possible to measure the distance by calculating an output reflective time signal by a microcomputer (Arduino).

◆ The main specification

- * Distance surveying area : 2-180cm
(15 times of area where it was made the center in front of the sensor substrate and resolution :0.3cm)
 - * Line voltage : DC 5.0V
 - * Movement electric current : 15mA
 - * Operation frequency : 40kHz
 - * Trigger signal : 10 μs
(wave pulse of the TTL level)
 - * The echo output signal : reflective
(round trip) hour
 - * Size: 45 x 20 x 15 mm
- ※ When turning on, please make sure that the GND terminal will be connected first.

Assembly of an ultrasonic range sensor.

* A sensor pin is put in a hole of a socket for ultrasonic range sensor installation of a sensor substrate together with indication.

11.8.3 プログラム (ArduBlock) を作成します。



上図を参考にプログラムを作成してください。
超音波センサからの距離情報が変数 threshold よりも小さくなったらモータを停止します。
走行速度との停止位置 (threshold) をうまく調整する必要があります。

・参考プログラム解説

①. 超音波センサの値を「変数」 distance に代入するプログラム部分。



②. 停止位置設定プログラム部分です、動作環境でシリアルモニターにて障害物回避させたい距離を計測して、しきい値を決め、threshold の [値] 数値に書き込みます。



③. シリアルモニタープログラムユニットを挿入すると、超音波センサ距離情報確認が容易になります。



④. 超音波センサ距離情報 [distance] がしきい値 [threshold] 以下になったら [≥], 停止します。でなければ前進 (モータ PWM 値 100) します。



前進スピード (走行速度) と停止位置の調整をしてください。早すぎると、停止させたい位置をオーバーランした状態になり、障害物に衝突します。
・停止した後に、どのように動作させるかは、皆さんでプログラミングしてください。

ソースコードは、PC/MyDocuments/Arduino/へ配置したサンプルフォルダ [ArduBlock Examples] に、ファイル名【21_01_clash_avoidance-HCSR04.abp】で格納されています。Arduino/ArduBlock で、PC/MyDocuments/Arduino/ArduBlock Examples の中に配置したサンプルを開くと確認できます。

● 21_01_clash_avoidance-HCSR04_sample を Arduino へアップロードし、C言語へ自動変換したコード
・直接C++言語で、コーディングも可能です。

```

Program source code
#include <STEMDU.h>

int _ABVAR_1_threshold = 0 ;
int _ABVAR_2_distance = 0 ;
STEMDU_STEMDU robot = STEMDU();
int ardublockUltrasonicSensorCodeAutoGeneratedReturnCM(int pingPin)
{
    long duration;
    pinMode(pingPin, OUTPUT);
    digitalWrite(pingPin, LOW);
    delayMicroseconds(2);
    digitalWrite(pingPin, HIGH);
    delayMicroseconds(5);
    digitalWrite(pingPin, LOW);
    pinMode(pingPin, INPUT);
    duration = pulseIn(pingPin, HIGH);
    duration = duration / 59;
    if ((duration < 2) || (duration > 300))
return false;
    return duration;
}

void setup()
{
    digitalWrite( 11 , LOW);
}

void loop()
{
    _ABVAR_1_threshold = 10 ;
    _ABVAR_2_distance = ardublockUltrasonicSensorCodeAutoGeneratedReturnCM( 11 );
    if ( ( ( _ABVAR_1_threshold ) >= ( _ABVAR_2_distance ) ) )
    {
        _STEMDU_robot.stopM1M2();
    }
    else
    {
        _STEMDU_robot.forwardM1M2(100);
    }
}
    
```

11.8.4. 超音波障害物回避プログラム (C)

Cで作成してみます。

※ サンプルのソースコードは、PC/MyDocuments/Arduino/へ配置したサンプルフォルダ [RDS-X Examples_C] に、ファイル名【ultrasonic_sample_avoidance】で格納されています。

ソースはC言語で記述されています。

※ Arduinoを起動後、[ファイル]→[スケッチブック]→[RDS-X Examples_C]→[ultrasonic_sample_avoidance]で開くと確認できます。



※ After starting Arudino, [file]-> [sketchbook] can confirm that-> [RDS-X Examples_S]-> opens by [ultrasonic_sample_avoidance].

※以下は、サンプルコードの内容です。薄い色の文字はコメント文です。

```

/*
  crash-avoidance_ping
  for JAPANROBOTECH RDC-103
  Souce code of ultra sonic distance sensor control
  from Arduino Sample Sketch "Ping".
  */

// 機体とポート類の接続は 左：小さい番号↔大きい番号：右
//A connection of a fuselage and port kinds The
left: The small number The big number: The right?

// 機体に接続するもの：モータ M1,M2 超音波センサ HC-SR04
// Something to connect to a fuselage: Motor M1,M2,
Ultrasonic sensor HC-SR04
// MotorDriver Pin Assign on RDC-103.
int M1_1 = 4;
int M1_2 = 5;
int M1_PWM = 6;
int M2_1 = 7;
int M2_2 = 8;
int M2_PWM = 9;

// pin number of the sensor's output:
int pingPin = 11;

int LEDPin = 13;

int PWM_Value = 200; // how speed the
Motor is
int distance = 15; // how long from target
object, unit is cm

void setup() {
  // initialize serial communication:
  Serial.begin(9600);
  // initialize the digital pin as an output.
  pinMode(M1_1, OUTPUT);
  pinMode(M1_2, OUTPUT);
  pinMode(M1_PWM, OUTPUT);
  pinMode(M2_1, OUTPUT);
  pinMode(M2_2, OUTPUT);
  pinMode(M2_PWM, OUTPUT);
  pinMode(LEDPin, OUTPUT);
}

```

```

// the loop routine runs over and over again forever:
void loop() {
  // establish variables for duration of the ping,
  // and the distance result in inches and
  centimeters:
  long duration, inches, cm;

  // The PING))) is triggered by a HIGH pulse of 2 or
  more microseconds.
  // Give a short LOW pulse beforehand to ensure a
  clean HIGH pulse:
  pinMode(pingPin, OUTPUT);
  digitalWrite(pingPin, LOW);
  delayMicroseconds(2);
  digitalWrite(pingPin, HIGH);
  delayMicroseconds(5);
  digitalWrite(pingPin, LOW);

  // The same pin is used to read the signal from the
  PING))) a HIGH
  // pulse whose duration is the time (in
  microseconds) from the sending
  // of the ping to the reception of its echo off of an
  object.
  pinMode(pingPin, INPUT);
  duration = pulseIn(pingPin, HIGH);

  // convert the time into a distance
  inches = icrosecondsToInches(duration);
  cm = microsecondsToCentimeters(duration);

  if(cm >= distance){
  // turn the Motor on
  analogWrite(M1_PWM, PWM_Value);
  digitalWrite(M1_1, HIGH);
  digitalWrite(M1_2, LOW);
  analogWrite(M2_PWM, PWM_Value);
  digitalWrite(M2_1, HIGH);
  digitalWrite(M2_2, LOW);
  digitalWrite(LEDPin, LOW); // turn
the LED off by making the voltage LOW
  }
  else{
  // turn the Motor off

```

```
digitalWrite(M1_1, LOW);
    digitalWrite(M1_2, LOW);
digitalWrite(M2_1, LOW);
    digitalWrite(M2_2, LOW);
    digitalWrite(LEDpin, HIGH); //
turn the LED on (HIGH is the voltage level)
}

    Serial.print(inches);
Serial.print("in, ");
Serial.print(cm);
Serial.print("cm");
Serial.println();

delay(100);
}

long microsecondsToInches(long
microseconds)
{
// According to Parallax's datasheet for the
PING))), there are
// 73.746 microseconds per inch (i.e. sound
travels at 1130 feet per
// second). This gives the distance travelled by
the ping, outbound
// and return, so we divide by 2 to get the
distance of the obstacle.
// See: http://www.parallax.com/dl/docs/prod/acc/28015-PING-v1.3.pdf
return microseconds / 74 / 2;
}

long microsecondsToCentimeters(long
microseconds)
{
// The speed of sound is 340 m/s or 29
microseconds per centimeter.
// The ping travels out and back, so to find the
distance of the
// object we take half of the distance travelled.
return microseconds / 29 / 2;
}
```

(2). プログラム動作後、[ツール] → [シリアルモニター]で確認できます。After program movement, it can be checked by [tool]-> [serial monitor].


シリアルモニターウィンドウのデータをコピーした内容です。A measured data example of an ultrasonic sensor by a sample program

サンプルプログラムによる超音波センサの実測データ例（対象物までの距離を in, cm で出力）(The distance to the target thing is output by in, cm.)




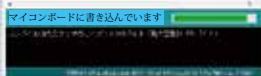
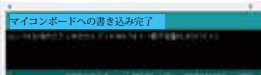
95in, 243cm	48in, 124cm
93in, 239cm	51in, 131cm
94in, 240cm	26in, 66cm
59in, 150cm	25in, 64cm
31in, 80cm	30in, 78cm
28in, 71cm	29in, 74cm
28in, 72cm	26in, 68cm
27in, 69cm	25in, 64cm
31in, 81cm	20in, 52cm
26in, 67cm	19in, 50cm
26in, 68cm	19in, 50cm
19in, 49cm	21in, 54cm
15in, 40cm	23in, 60cm
14in, 37cm	29in, 74cm
19in, 50cm	30in, 77cm
19in, 49cm	23in, 60cm
22in, 56cm	28in, 71cm
31in, 80cm	27in, 69cm
26in, 67cm	7in, 20cm
29in, 76cm	5in, 15cm
31in, 79cm	5in, 13cm
28in, 71cm	5in, 14cm
30in, 77cm	66in, 169cm
32in, 82cm	131in, 334cm
47in, 122cm	
50in, 129cm	

エラーメッセージ Couldn't find a Leonardo on the selected port 出現時
通信エラーが発生し、マイコンボードへの書き込みが失敗しています。
 1)USB ケーブル接続確認 2)Arduino ▷ [ツール] ▷ 「マイコンボード」
 RDC-102 に ●マーク、「シリアルポート」接続 COM 番号に ✓マーク
 を確認してください。

・アップロードがうまくいかない場合は、コンパイル
 の後、マイコンボードに書き込みが始まる前に、
**RDC-103 コントローラの RESET スイッチを「ダブルク
 リック」**してください。



↑ RESET

Error Message
 Couldn't find a Leonardo on the selected port. Check that you have the correct port selected. If it is correct, try pressing the board's reset button after initiating the upload.

11-9. エンコーダを使ったロボ

11.9.1. 直進⇄旋回で、モータ個体差を確認する。

encoder_sample0_forward for RoboDesigner
単相エンコーダ付ギアボックス用サンプルプログラムを使います。

車両モデルで、左右に2つのモータに同じPWM値を指定して直進 - 旋回させます。

実際にはモータの個体差があるため、まっすぐに進まないことを確認してください。

モータの制御は関数にします。

void motorMove (動作時間, M1 速度, M2 速度)

M1 速度、M2 速度をマイナス値にすると逆転します。
moveTime をスピードに応じて調整することで移動距離を決めます。

コントローラ搭載のスライダーでスピードを調整します。

```

encoder_sample0_forward

// 機体とポート類の接続は 左：小さい番号←→大きい番号：右
// MotorDriver Pin Assign on RDC.
int M1_1 = 4;
int M1_2 = 5;
int M1_PWM = 6;
int M2_1 = 7;
int M2_2 = 8;
int M2_PWM = 9;
int maxDuty = 255; //モータPWMデューティ比の最大値
int duty;

// 動作時間設定用変数
int move1 = 2000; // 進む間隔
int move2 = 1300; // 曲がる間隔

// the setup routine runs once when you
press reset:
void setup() {
  // initialize the digital pin as an
output.
  pinMode(M1_1, OUTPUT);
  pinMode(M1_2, OUTPUT);
  pinMode(M1_PWM, OUTPUT);
  pinMode(M2_1, OUTPUT);
  pinMode(M2_2, OUTPUT);
  pinMode(M2_PWM, OUTPUT);

  delay(2000);
}

// the loop routine runs over and over
again forever:
void loop() {
  duty = map(analogRead(A5), 0, 1023, 0,
maxDuty);
  //duty = 50; //スピードを固定する

  // 四角に動く
  motorMove (move1, duty, duty);
  delay(1000);
  motorMove (move2, 0 - duty, duty);
  delay(1000);
  motorMove (move1, duty, duty);

```

```

delay(1000);
motorMove (move2, 0 - duty, duty);
delay(1000);
motorMove (move1, duty, duty);
delay(1000);
motorMove (move2, 0 - duty, duty);
delay(1000);
motorMove (move1, duty, duty);
delay(1000);
motorMove (move2, 0 - duty, duty);
delay(2000);
}

// モータの制御関数
void motorMove (int moveTime, int speedM1,
int speedM2) {
  if (speedM1 >= 0) {
    digitalWrite(M1_1, HIGH);
    digitalWrite(M1_2, LOW);
  } else {
    digitalWrite(M1_1, LOW);
    digitalWrite(M1_2, HIGH);
  }
  speedM1 = abs(speedM1);
  analogWrite(M1_PWM, speedM1);
  if (speedM2 >= 0) {
    digitalWrite(M2_1, HIGH);
    digitalWrite(M2_2, LOW);
  } else {
    digitalWrite(M2_1, LOW);
    digitalWrite(M2_2, HIGH);
  }
  speedM2 = abs(speedM2);
  analogWrite(M2_PWM, speedM2);

  delay(moveTime);

  digitalWrite(M1_1, LOW);
  digitalWrite(M1_2, LOW);
  analogWrite(M1_PWM, 0);
  digitalWrite(M2_1, LOW);
  digitalWrite(M2_2, LOW);
  analogWrite(M2_PWM, 0);
}

```

6 TEM Du/RoboDesigner+ RDC-102 w/ ATmega32U4 3.3V 8MHz on COM3

※ソースコードは、PC/MyDocuments/Arduino/へ配置したサンプルフォルダ[RDS-X Examples_C]に、ファイル名【encoder_sample0_forward.ino】で格納されています。

※ Arduino を起動後、[ファイル]→[スケッチブック]→[RDS-X Examples_C]→[サンプルプログラム名]で開くと確認できます。

11.9.2. エンコーダの出力値をカウントする。

encoder_sample2_count for RoboDesigner
単相エンコーダ付ギアボックス用サンプルプログラムです。

エンコーダの出力をカウントするには、割り込みピンを割り込み関数で監視して、エンコーダの出力変化(HIGH/LOW)で変数をカウントアップします。スライダーでモータの操作量(PWMのデューティ比)を変化させ、単位時間あたりのモータの回転数を調べます。エンコーダの出力変化の間隔はミリ秒以下で非常に速いため、delayやシリアルモニタ出力の処理時間を考慮する必要があります。

while(!Serial)をコメントアウトして実際に走行させてみてください。
モータを左右に2つ使った車両モデルでは、同じPWM値を指定して直進させても実際にはモータの個体差があるため、まっすぐに進みません。

```

encoder_sample2_count

*/
// 機体とポート類の接続は 左：小さい番号←→大きい番号：右
// MotorDriver Pin Assign on RDC.
int M1_1 = 4;
int M1_2 = 5;
int M1_PWM = 6;
int M2_1 = 7;
int M2_2 = 8;
int M2_PWM = 9;
// Encoder Interrupt Pin Assign on RDC.
int encoder1 = 2; //D0
int encoder2 = 3; //D1

int maxDuty = 255; //モータPWMデューティ比の最大値
int duty; //現在のPWMデューティ比
int Ktime = 10; //サンプリング時間K
int KtimeCount; //loopの回数
volatile int enCountNum1; //エンコーダの直近のカウント数
volatile int enCountNum2; //エンコーダの直近のカウント数

// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600
  bits per second:
  Serial.begin(9600);
  while(!Serial); // for 32U4 シリアルモニタを起動
  するまでプログラムはここから進みません

  // initialize the digital pin as an output.
  pinMode(M1_1, OUTPUT);
  pinMode(M1_2, OUTPUT);
  pinMode(M1_PWM, OUTPUT);
  pinMode(M2_1, OUTPUT);
  pinMode(M2_2, OUTPUT);
  pinMode(M2_PWM, OUTPUT);
  attachInterrupt(encoder1, count,
  CHANGE);
  attachInterrupt(encoder2, count2,
  CHANGE);
  //attachInterrupt(digitalPinToInterrupt(0),
  count, CHANGE); //recommended syntax, but not
  function

  KtimeCount = 0;

```

```

enCountNum1 = 0;
enCountNum2 = 0;
}

// the loop routine runs over and over again forever:
void loop() {
  duty = map(analogRead(A5), 0, 1023, 0,
  maxDuty);

  // サンプリング時間ごとの処理
  if (KtimeCount >= Ktime) {
    KtimeCount = 0;
    enCountNum1 = 0;
    enCountNum2 = 0;
  }
  KtimeCount++;

  // モータの回転
  digitalWrite(M1_1, HIGH);
  digitalWrite(M1_2, LOW);
  analogWrite(M1_PWM, duty); //モータのPWM設定
  digitalWrite(M2_1, HIGH);
  digitalWrite(M2_2, LOW);
  analogWrite(M2_PWM, duty); //モータのPWM設定

  // シリアルモニタチェック
  Serial.print(" duty= ");
  Serial.print(duty);
  Serial.print(" loop= ");
  Serial.print(KtimeCount);
  Serial.print(" encoder count1= ");
  Serial.print(enCountNum1);
  Serial.print(" encoder count2= ");
  Serial.println(enCountNum2);
}

// 割り込み関数 エンコーダ出力1が変化したらカウントアップする
void count() {
  enCountNum1++;
}

// 割り込み関数 エンコーダ出力2が変化したらカウントアップする
void count2() {
  enCountNum2++;
}

```

8\$ TEM Du/RoboDesigner+ RDC-102 w/ ATmega32U4 3.3V 8MHz on COM3

encoder_sample2_count プログラム動作 シリアルモニターデータ例

```

duty= 104 loop= 1 encoder count1= 1 encoder count2= 3
duty= 103 loop= 2 encoder count1= 3 encoder count2= 7
duty= 103 loop= 3 encoder count1= 6 encoder count2= 10
duty= 105 loop= 4 encoder count1= 8 encoder count2= 14
duty= 103 loop= 5 encoder count1= 11 encoder count2= 18
duty= 103 loop= 6 encoder count1= 14 encoder count2= 22
duty= 103 loop= 7 encoder count1= 16 encoder count2= 26
duty= 103 loop= 8 encoder count1= 19 encoder count2= 30
duty= 104 loop= 9 encoder count1= 22 encoder count2= 34
duty= 103 loop= 10 encoder count1= 24 encoder count2= 38

```

エンコーダ1(左側)とエンコーダ2(右側)のデータに違いがあり、左右のモータに個体差があることが確認できます。

※ソースコードは、PC/MyDocuments/Arduino/へ配置したサンプルフォルダ[RDS-X Examples_C]に、ファイル名【encoder_sample2_count.ino】で格納されています。
※ Arduinoを起動後、[ファイル]→[スケッチブック]→[RDS-X Examples_C]→[サンプルプログラム名]で開くと確認できます。

11.9.3. スピードコントロール ON OFF

encoder_sample3_speedControl_ONOFF for RoboDesigner
単相エンコーダ付ギアボックス用サンプルプログラムです。

エンコーダの出力 (HIGH/LOW) は割り込みでカウントします。
単位時間あたりのモータの回転数 (スピード) が目標値に到達したらモータをオフにします。目標値より下がったらオンにします。

```

ファイル 編集 スケッチ ツール ヘルプ
encoder_sample3_speedControl_ONOFF

*/

// 機体とポート類の接続は 左: 小さい番号 ←→ 大きい番号: 右
// MotorDriver Pin Assign on RDC.
int M1_1 = 4;
int M1_2 = 5;
int M1_PWM = 6;
// Encoder Interrupt Pin Assign on RDC.
int encoder = 2; //D0

// 動作の変化にかかわる定数
int Ktime = 10; // サンプルング間隔
int targetSpeedMax = 100; // 到達目標値 (スピード)
// の最大値 目標値はスライダー (A5) で変更できるようにします
// duty が 255 の場合、loop 1 周の間にエンコーダは約 8 カ
// ウントするので、サンプルングが 10 周で 80 を到達目標値とする
// 連動して設定しないと偏差 (deviation) を 0 にできない
// = 最高速度以上の目標は設定できないということ
// また、シリアル出力の時間は長いので、シリアルモニタ
// を接続しないとカウント数が少なくなります
// その他の変数
int KtimeCount; // サンプルング間隔のカウント
int targetSpeed; // 到達目標値
volatile int enCountNum; // エンコーダの直近のカウント数

// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per
  // second:
  Serial.begin(9600);
  while(!Serial); // for 32U4 シリアルモニタを起動す
  // るまでプログラムはここから進みません

  // initialize the digital pin as an output.
  pinMode(M1_1, OUTPUT);
  pinMode(M1_2, OUTPUT);
  pinMode(M1_PWM, OUTPUT);
  attachInterrupt(encoder, count, CHANGE);
  //attachInterrupt(digitalPinToInterrupt(0), count,
  // CHANGE); //recommended syntax, but not function

  enCountNum = 0; // エンコーダのカウント数を初期値にする
}

// the loop routine runs over and over again forever:
void loop() {

  targetSpeed = map(analogRead(A5), 0,
  1023, 0, targetSpeedMax); // スライダーの値をス

```

ピード調整範囲 0~targetSpeedMax にマップする

```

if (KtimeCount >= Ktime) {
  if (enCountNum < targetSpeed) {
    // モータの回転
    digitalWrite(M1_1, HIGH);
    digitalWrite(M1_2, LOW);
    analogWrite(M1_PWM, 255); // モータのPWM設定
  } else {
    // モータ停止
    digitalWrite(M1_1, HIGH);
    digitalWrite(M1_2, HIGH);
    analogWrite(M1_PWM, 0);
  }
  enCountNum = 0;
  KtimeCount = 0;
}
KtimeCount++;

// シリアルモニタチェック
Serial.print(" KtimeCount= "); // エンコーダの出力値
Serial.print(KtimeCount);
Serial.print(" targetSpeed= "); // 目標カウント数
Serial.print(targetSpeed);
Serial.print(" encoder count= "); // エンコーダ
// のカウント数
Serial.println(enCountNum);
}

// 割り込み関数 エンコーダ出力が変化したらカウントアップする
void count() {
  enCountNum++;
}

```

©TEM Du/RoboDesigner+ RDC-102 w/ ATmega32U4 3.3V 8MHz on COM3

※ソースコードは、PC/MyDocuments/Arduino/へ配置したサンプルフォルダ [RDS-X Examples_C] に、ファイル名 [encoder_sample3_speedControl_ONOFF.ino] で格納されています。※ Arduino を起動後、[ファイル] → [スケッチブック] → [RDS-X Examples_C] → [サンプルプログラム名] で開くと確認できます。

11.9.4. PID 制御でスピードコントロール

encoder_sample4_speedControl_PID for RoboDesigner
単相エンコーダ付ギアボックス用サンプルプログラムです
エンコーダの出力変化(HIGH/LOW)は割り込みでカウン
トします。

モータの回転数をPID 制御で一定の目標値(=スピー
ド)に保ちます。

目標値とエンコーダのカウントの差を元にモータへの操作
量(PWM のデューティー比)を変化させます。

デジタル/離散信号でPID 制御するには、以下のような
式で表現できます。

操作量 = 前回の操作量 + 追加の操作量

追加の操作量 = $K_p \cdot \text{偏差} + K_i \cdot \text{偏差} + K_d \cdot \text{前回の偏差}$
と今回の偏差の差

偏差は、目標値 - 一定時間内(Ktime)のエンコーダ
のカウント数です。

K_p が小さいと目標値に到達できず、大きいと目標を超
えてしまいます。目標値近くで出力が足りない状態が起
こります(残留偏差)。

残留偏差をなくすため、偏差に K_i を掛けた数値を操作
量に積算します。比例制御の目標近くで出力が足りない
分を補います。

前回の偏差と今回の偏差の差は、一定間隔をあけた偏差
同士の差です。間隔をとらないと、プログラム処理が速い
ため、常に1 や0 になってしまいます。

偏差の差に適切な K_d を掛けて、まだ目標に遠いときほ
ど大きく比例制御の出力を補い、応答性を向上させま
す。

最初はP のみになっているので、I、D のそれぞれを順に
コメントアウトを外して試してください。

スライダの数値(targetSpeedNew)が大きいと定数の
チューニングの限界を超えるため、うまく動作しません。

```

ファイル 編集 スケッチ ツール ヘルプ
encoder_sample4_speedControl_PID
*/
// 機体とポート類の接続は 左:小さい番号<-->大きい番号:右
// MotorDriver Pin Assign on RDC.
int M1_1 = 4;
int M1_2 = 5;
int M1_PWM = 6;
// Encoder Interrupt Pin Assign on RDC.
int encoder = 2; //D0
// 動作の変化にかかわる定数
//float 型は処理が遅くなるのでゲインKp,Ki,Kd は100 倍で
intで指定します.duty にマッピングする前に1/100 に戻しま
す。
int Kp = 60; // 比例制御のゲイン.停止時の減速の程度が変化します
int Ki = 60; // 積分制御のゲイン.停止時の減速の程度が変化します
int Kd = 30; // 微分制御のゲイン.加速の程度が変化します
int Ktime = 10; // サンプル間隔
int maxDuty= 255; // モータPWM デューティー比の最大値

```

```

int targetSpeedMax = 100; // 到達目標値(スピード)
の最大値 目標値はスライダ(A5)で変更できるようにします
//duty が255 の場合、loop 1 周の間にエンコーダは何カウント
するか、サンプリングが10 周なのでその10 倍を到達目標値とす
る
// 連動して設定しないと偏差(deviation)を0 にできない(=
最高速度以上の目標は設定できないということ)
// その他の変数
int KtimeCount; // サンプル間隔のカウント
int targetSpeedNew; // 最新の到達目標値
int targetSpeedOld; // 変更前の到達目標値
volatile int enCountNum; //エンコーダの直近のカウント数
int deviationNew; // 最新の偏差 到達目標値 - エン
コーダの直近のカウント数
int deviationOld; // 前回の偏差
int deviationDiff; // 前回と最新の偏差の差
int operationVolume; // 操作量
int operationVolumeLast; // 前回の操作量
int nowDuty; // デューティー比の直近の値
// the setup routine runs once when you press reset:
void setup() {
// initialize serial communication at 9600 bits per second:
Serial.begin(9600);
while(!Serial); // for 32U4 シリアルモニタを起動
するまでプログラムはここから進みません
// initialize the digital pin as an output.
pinMode(M1_1, OUTPUT);
pinMode(M1_2, OUTPUT);
pinMode(M1_PWM, OUTPUT);
attachInterrupt(encoder, count, CHANGE);
//attachInterrupt(digitalPinToInterrupt(0), count, CHANGE);
//recommended syntax, but not function
KtimeCount = 0;
enCountNum = 0;
deviationOld = 0;
operationVolumeLast = 0;
}
// the loop routine runs over and over again forever:
void loop() {
targetSpeedNew = map(analogRead(A5), 0,1023,
0, targetSpeedMax); // スライダの値をスピー
ド調整範囲0~targetSpeedMax にマップする
if (targetSpeedOld != targetSpeedNew) //
目標値の変更をチェックする 決定する(変化しなくなる)まで
operationVolumeLast をリセットする
operationVolumeLast = 0;
targetSpeedOld = targetSpeedNew;
//PID でPWM のデューティー比を調整する
if (KtimeCount >= Ktime) {
deviationNew = targetSpeedNew - enCountNum;
deviationDiff = deviationOld - deviationNew;
deviationOld = deviationNew;
enCountNum = 0;
KtimeCount = 0;

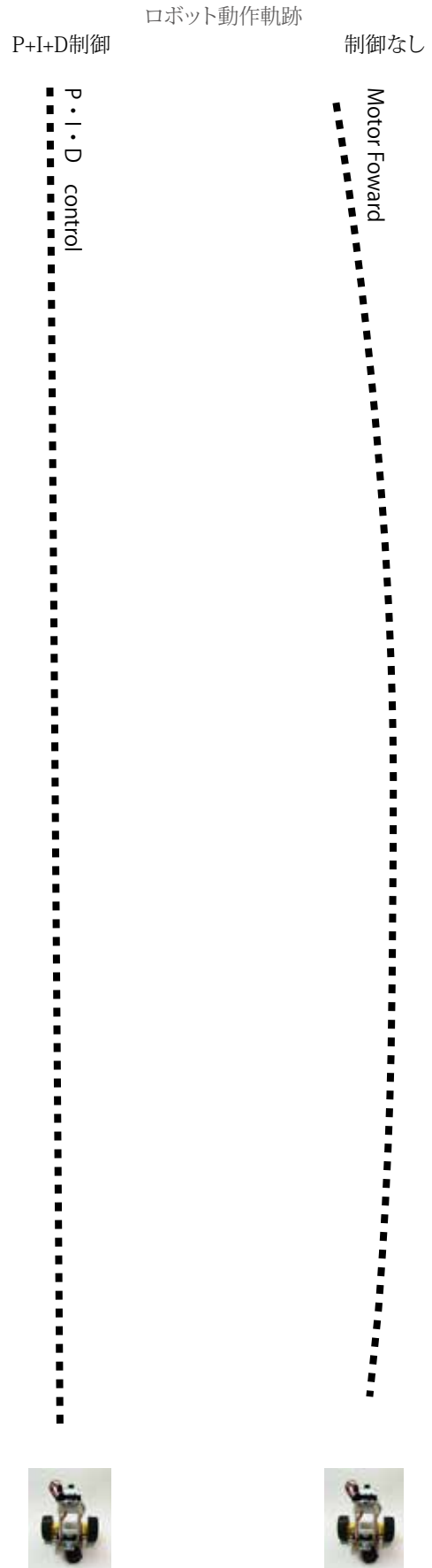
Serial.print(" targetSpeed= ");
Serial.print(targetSpeedNew);
Serial.print(" deviation= ");
Serial.print(deviationNew);
Serial.print(" deviationDiff= ");
Serial.println(deviationDiff);
}
KtimeCount++;
//P、I、D のそれぞれを順にコメントアウトを外して試してください。
operationVolume = deviationNew * Kp /100;
//P 制御

```

```
//operationVolume = operationVolume +
(deviationNew * Ki / 100); //I 制御を加える
//operationVolume = operationVolume +
(deviationDiff * Kd / 100); //D 制御を加える
operationVolume = operationVolumeLast +
operationVolume // 前回までの操作量に今回の操作量
を加える
operationVolumeLast = operationVolume;
// 最新の操作量を前回の操作量に代入する
nowDuty = map(operationVolume, 0, 5000, 0,
maxDuty); //operationVolume の最大値はシリアルモニ
タの実測で決定しています
// モータの回転
digitalWrite(M1_1, HIGH);
digitalWrite(M1_2, LOW);
analogWrite(M1_PWM, nowDuty); // モータのPWM
設定
// シリアルモニタチェック
Serial.print(" Ktime= ");
Serial.print(KtimeCount);
Serial.print(" encoder count= ");
Serial.print(enCountNum);
Serial.print(" operationVolume= ");
Serial.print(operationVolume);
Serial.print(" duty= ");
Serial.println(nowDuty);
}
// 割り込み関数 エンコーダ出力が変化したらカウントアップする
void count() {
enCountNum++;
}
```

6 TEM Du/RoboDesigner+ RDC-102 w/ ATmega32U4 3.3V 8MHz on COM3

※ソースコードは、PC/MyDocuments/Arduino/へ配置したサ
ンプルフォルダ[RDS-X Examples_C]に、ファイル名【encod-
er_sample4_speedControl_PID.ino】で格納されています。※
Arudino を起動後、[ファイル] → [スケッチブック] → [RDS-X
Examples_C] → [サンプルプログラム名] で開くと確認できま
す。



11.9.5. PID制御で描画する

encoder_sample5_lineDraw for RoboDesigner

単相エンコーダ付ギアボックス用サンプルプログラムです
モータを左右に2 つ使った車両モデルの場合、モータの
個体差があるため、同じPWM 値を指定しても直進でき
ません。

モータのPID 速度制御を関数にして、まっすぐに進める
ようにします。

関数 motorPID(動作時間, M1 速度, M2 速度) でモー
タの回転する時間、モータ2 個それぞれの速度を指定
することで軌跡をコントロールして車体に取り付けたペ
ンで描画をします。

M1 速度、M2 速度をマイナス値にすると逆転します。

moveTime をスピードに応じて調整することで移動距離
を決めます。

エンコーダの出力変化(HIGH/LOW)は割り込みでカウ
ントします。

モータの回転数をPID 制御で一定の目標値(=スピー
ド)に保ちます。

目標値とエンコーダのカウントの差を元にモータへの操
作量(PWM のデューティ比)を変化させます。

デジタル/離散信号でPID 制御するには、以下のよう
な式で表現できます。

操作量 = 前回の操作量 + 追加の操作量

追加の操作量 = $K_p \cdot \text{偏差} + K_i \cdot \text{偏差} + K_d \cdot \text{前回の偏差と}$
今回の偏差の差

偏差は、目標値 - 一定時間内(Ktime)のエンコーダのカ
ウント数です。

K_p が小さいと目標値に到達できず、大きいと目標を超
えてしまいます。目標値近くで出力が足りない状態が起
こります(残留偏差)。

残留偏差をなくすため、偏差に K_i を掛けた数値を操作
量に積算します。比例制御の目標近くで出力が足りない
分を補います。

前回の偏差と今回の偏差の差は、一定間隔をあけた偏
差同士の差です。間隔をとらないと、プログラム処理が
速いため、常に1 や0 になってしまいます。

偏差の差に適切な K_d を掛けて、まだ目標に遠いときほど
大きく比例制御の出力を補い、応答性を向上させます。
スライダの数値(targetSpeedNew)が大きいと定数の
チューニングの限界を超えるため、うまく動作しません。

```

ファイル 編集 スケッチ ツール ヘルプ
encoder_sample5_LineDraw
.
// 機体とポート類の接続は 左:小さい番号<-->大きい番号:右
// MotorDriver Pin Assign on RDC.
int M1_1 = 4;
int M1_2 = 5;
int M1_PWM = 6;
int M2_1 = 7;
int M2_2 = 8;
int M2_PWM = 9;
// Encoder Interrupt Pin Assign on RDC.
int encoder1 = 2; //D0
int encoder2 = 3; //D1

```

// 動作の変化にかかわる定数

//float 型は処理が遅くなるのでゲイン K_p, K_i, K_d は100 倍でint で
指定します。duty にマッピングする前に1/100 に戻します。

int K_p = 60; // 比例制御のゲイン。停止時の減速の程度が変化します

int K_i = 60; // 積分制御のゲイン。停止時の減速の程度が変化します

int K_d = 30; // 微分制御のゲイン。加速の程度が変化します

int Ktime = 10; // サンプルング間隔

int maxDuty= 255; // モータPWM デューティ比の最大値

int targetSpeedMax = 100; // 到達目標値(スピー
ド)の最大値 目標値はスライダ(A5)で変更できるようにします

//duty が255 の場合、loop 1 周の間にエンコーダは何カウントする
か、サンプルングが10 周なのでその10 倍を到達目標値とする

// 運動して設定しないと偏差(deviation)を0 にできない(=最高速
度以上の目標は設定できないということ)

// 動作時間設定用変数

int move1 = 300; // 進む間隔

int move2 = 200; // 曲がる間隔

// その他の変数

int KtimeCount; // サンプルング間隔のカウント

int targetSpeedNew; // 最新の到達目標値

int targetSpeedOld; // 変更前の到達目標値

//M1 用

volatile int enCountNum1; // エンコーダの
直近のカウント数

int deviationNew1; // 最新の偏差 到達目標
値 - エンコーダの直近のカウント数

int deviationOld1; // 前回の偏差

int deviationDiff1; // 前回と最新の偏差の差

int operationVolume1; // 操作量

int operationVolumeLast1; // 前回の操作量

int nowDuty1; // デューティ比の直近の値

//M2 用

volatile int enCountNum2; // エンコーダの直
近のカウント数

int deviationNew2; // 最新の偏差 到達目標
値 - エンコーダの直近のカウント数

int deviationOld2; // 前回の偏差

int deviationDiff2; // 前回と最新の偏差の差

int operationVolume2; // 操作量

int operationVolumeLast2; // 前回の操作量

int nowDuty2; // デューティ比の直近の値

// the setup routine runs once when you press reset:

void setup() {

// initialize serial communication at 9600 bits per
second:

Serial.begin(9600);

//while(!Serial); // for 32U4 シリアルモニタを起動するまでプ
ログラムはここから進みません

// initialize the digital pin as an output.

pinMode(M1_1, OUTPUT);

pinMode(M1_2, OUTPUT);

pinMode(M1_PWM, OUTPUT);

pinMode(M2_1, OUTPUT);

pinMode(M2_2, OUTPUT);

pinMode(M2_PWM, OUTPUT);

attachInterrupt(encoder1, count1, CHANGE);

attachInterrupt(encoder2, count2, CHANGE);

//attachInterrupt(digitalPinToInterrupt(0), count,
CHANGE); //recommended syntax, but not func-
tion

KtimeCount = 0;


```

enCountNum1 = 0;
deviationOld1 = 0;
operationVolumeLast1 = 0;
enCountNum2 = 0;
deviationOld2 = 0;
operationVolumeLast2 = 0;
delay(2000);
}
// the loop routine runs over and over again forever:
void loop() {
  // スライダーでスピードを決める
  targetSpeedNew = map(analogRead(A5),
    0, 1023, 0, targetSpeedMax); // スライダー
    の値をスピード調整範囲0~targetSpeedMax にマップする
  //targetSpeedNew = 50; // スピードを固定する
  motorPID(move1, targetSpeedNew,
    targetSpeedNew);
  delay(1000);
  motorPID(move2, 0 - targetSpeedNew,
    targetSpeedNew);
  delay(1000);
  motorPID(move1, targetSpeedNew,
    targetSpeedNew);
  delay(1000);
  motorPID(move2, 0 - targetSpeedNew,
    targetSpeedNew);
  delay(1000);
  motorPID(move1, targetSpeedNew,
    targetSpeedNew);
  delay(1000);
  motorPID(move2, 0 - targetSpeedNew,
    targetSpeedNew);
  delay(1000);
  motorPID(move1, targetSpeedNew,
    targetSpeedNew);
  delay(1000);
  motorPID(move2, 0 - targetSpeedNew,
    targetSpeedNew);
  delay(2000);
}
// モータ制御関数
// スピードをマイナス値にすると逆転
//moveTime はスピードに応じて調整する
void motorPID(int moveTime, int speed-
M1, int speedM2) {
  int speedM1tmp;
  int speedM2tmp;
  // モータの回転方向が逆転でマイナスだったら、偏差の計算用に正数にする
  speedM1tmp = speedM1;
  if (speedM1 < 0) {
    speedM1tmp = abs(speedM1tmp);
  }
  speedM2tmp = speedM2;
  if (speedM2 < 0) {
    speedM2tmp = abs(speedM2tmp);
  }
  KtimeCount = 0;
  deviationOld1 = 0;
  deviationOld2 = 0;
  operationVolumeLast1 = 0;

```

```

operationVolumeLast2 = 0;
enCountNum1 = 0;
enCountNum2 = 0;
while (moveTime > 0) {
  //PID でPWM のデューティ比を調整する
  if (KtimeCount >= Ktime) {
    deviationNew1 = speedM1tmp - enCount-
    Num1;
    deviationDiff1 = deviationOld1 -
    deviationNew1;
    deviationOld1 = deviationNew1;
    enCountNum1 = 0;
    deviationNew2 = speedM2tmp -
    enCountNum2;
    deviationDiff2 = deviationOld2 -
    deviationNew2;
    deviationOld2 = deviationNew2;
    enCountNum2 = 0;
    KtimeCount = 0;
    Serial.print(" deviation= ");
    Serial.print(deviationNew1);
    Serial.print(" : ");
    Serial.print(deviationNew2);
    Serial.print(" deviationDiff= ");
    Serial.print(deviationDiff1);
    Serial.print(" : ");
    Serial.println(deviationDiff2);
  }
  KtimeCount++;
  operationVolume1 = deviationNew1 * Kp
  / 100; //P 制御
  operationVolume1 = operationVolume1 +
  (deviationNew1 * Ki / 100); //I 制御
  //operationVolume1 = operationVolume1
  + (deviationDiff1 * Kd / 100); //D 制御
  operationVolume1 =
  operationVolumeLast1 +
  operationVolume1; // 前回までの操作量に今回の操作
  量を加える
  operationVolumeLast1 =
  operationVolume1; // 最新の操作量を前回の操作量に
  代入する
  nowDuty1 = map(operationVolume1,
  0, 5000, 0, maxDuty); //operationVolume の最大
  値はシリアルモニタの実測で決定しています
  // モータの回転方向
  if (speedM1 >= 0) {
    digitalWrite(M1_1, HIGH);
    digitalWrite(M1_2, LOW);
  } else {
    digitalWrite(M1_1, LOW);
    digitalWrite(M1_2, HIGH);
  }
  analogWrite(M1_PWM, nowDuty1); // モータ
  のPWM 設定
  operationVolume2 = deviationNew2 * Kp
  / 100; //P 制御
  operationVolume2 = operationVolume2
  + (deviationNew2 * Ki / 100); //I 制御
  //operationVolume2 = operationVolume2
  + (deviationDiff2 * Kd / 100); //D 制御

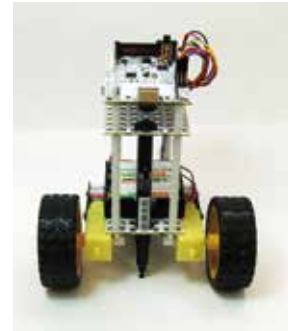
```

```

operationVolume2 =
operationVolumeLast2 + operationVolume2;
// 前回までの操作量に今回の操作量を加える
operationVolumeLast2 =
operationVolume2;
// 最新の操作量を前回の操作量に代入する
nowDuty2 = map(operationVolume2, 0,
5000, 0, maxDuty);
//operationVolume の最大値はシリアルモニタの実測で決定しています
// モータの回転方向
if (speedM2 >= 0) {
digitalWrite (M2_1, HIGH);
digitalWrite (M2_2, LOW);
} else {
digitalWrite (M2_1, LOW);
digitalWrite (M2_2, HIGH);
}
analogWrite (M2_PWM, nowDuty2);
// モータのPWM 設定
//delayMicroseconds(6000); // シリアルモニタを使わない場合は
時間調整用の遅延を入れる
// シリアルモニタチェック
Serial.print (" encoder count= ");
Serial.print (enCountNum1);
Serial.print (" : ");
Serial.print (enCountNum2);
Serial.print (" operationVolume= ");
Serial.print (operationVolume1);
Serial.print (" : ");
Serial.print (operationVolume2);
Serial.print (" duty= ");
Serial.print (nowDuty1);
Serial.print (" : ");
Serial.println (nowDuty2);
moveTime--;
}
// モータ停止
digitalWrite (M1_1, HIGH);
digitalWrite (M1_2, HIGH);
analogWrite (M1_PWM, 0);
digitalWrite (M2_1, HIGH);
digitalWrite (M2_2, HIGH);
analogWrite (M2_PWM, 0);
}
// 割り込み関数 エンコーダ出力1 が変化したらカウントアップする
void count1 () {
enCountNum1++;
}
// 割り込み関数 エンコーダ出力2 が変化したらカウントアップする
void count2 () {
enCountNum2++;
}

```

描画する 筆記具をセットします。



描画用サインペン

描画させる時に、サインペンを差し込み使用します。テストはTombow ツインマーカー33 クロを使用しました。描画する台紙に対し、筆圧が必要です。ユニバーサルシャシープレートの13mmφの穴を利用して、サインペン・筆ペンなどを利用してよいでしょう。

【高さ調整】描画用筆記具を変更する場合は、長さに応じてペンホルダーの高さを変更してください。描画する台紙に対し、少し筆圧が必要です。高さの調整ができるように支柱の芯をネジ式にしています。スプーサー長さを変更したり、ネジを回したりして高さの微調整ができます。

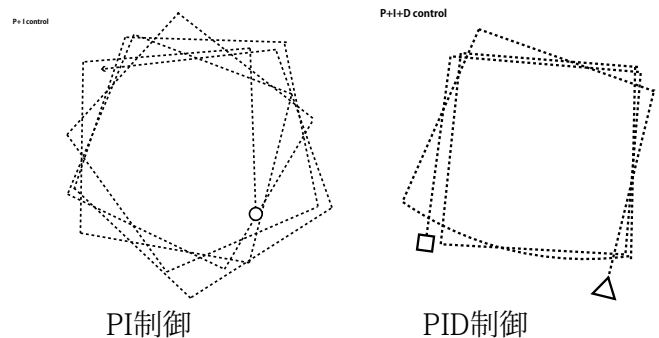
下図は、サインペンでの描画例です。
サンプルの

// 動作時間設定用変数

int move1 = 200; // 進む間隔

int move2 = 180; // 曲がる間隔 に変更し、

//operationVolume2 = operationVolume2 +
(deviationDiff2 * Kd / 100); //D 制御 のコメントアウト//を外して P/I/D 制御をかけてみました。



※サンプルソースコードは、PC/MyDocuments/Arduino/へ配置したサンプルフォルダ[RDS-X Examples_C]に、ファイル名[encoder_sample5_lineDraw.ino]で格納されています。※ソースは、C言語で記述されています。※Arduinoを起動後、[ファイル]→[スケッチブック]→[RDS-X Examples_C]→[サンプルプログラム名]で開くと確認できます。

11.9.6. 指定ポジションで停止する。

encoder_sample6_stop for RoboDesigner
 単相エンコーダ付ギアボックス用サンプルプログラムです。
 エンコーダの出力をカウントするには、割り込みピンを割り込み関数で監視して、エンコーダの出力変化(HIGH/LOW)で変数をカウントアップします。
 エンコーダの出力カウントが目標値に到達したらモータを停止します。
 stopPosition とduty を変更するとモータ停止時のオーバーランの変化がわかります。
 シリアルモニタで確認してください。

```

// 機体とポート類の接続は 左:小さい番号←→大きい番号:右
// MotorDriver Pin Assign on RDC.
int M1_1 = 4;
int M1_2 = 5;
int M1_PWM = 6;
// Encoder Interrupt Pin Assign on RDC.
int encoder = 2; //D0
int stopPosition = 500; //到達目標値
int duty= 100; // モータPWM デューティー比
volatile int enCountNum; // エンコーダの直近のカウント数
// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  Serial.begin(9600);
  while(!Serial); // for 32U4 シリアルモニタを起動するまでプログラムはここから進みません
  // initialize the digital pin as an output.
  pinMode(M1_1, OUTPUT);
  pinMode(M1_2, OUTPUT);
  pinMode(M1_PWM, OUTPUT);
  attachInterrupt(encoder, count, CHANGE);
  //attachInterrupt(digitalPinToInterrupt(0), count, CHANGE); //recommended syntax, but not function
  enCountNum = 0; //エンコーダのカウント数を初期値にする
}
// the loop routine runs over and over again forever:
void loop() {
  // エンコーダの出力カウントが到達目標より小さかったらモータを回転、到達目標値になったらモータを停止する
  if (enCountNum <= stopPosition) {
    digitalWrite(M1_1, HIGH);
    digitalWrite(M1_2, LOW);
    analogWrite(M1_PWM, duty); //モータのPWM設定
  } else {
    digitalWrite(M1_1, LOW);

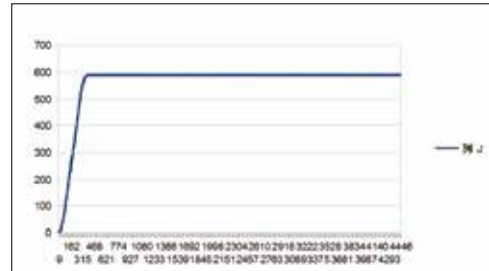
```

```

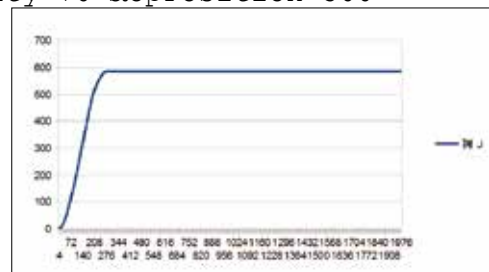
digitalWrite(M1_2, LOW);
analogWrite(M1_PWM, 0); //モータのPWM設定
}
// シリアルモニタチェック
Serial.print(" duty= "); //エンコーダの出力値
Serial.print(duty);
Serial.print(" target count= "); //目標カウント数
Serial.print(stopPosition);
Serial.print(" encoder count= "); //エンコーダのカウント数
Serial.println(enCountNum);
}
// 割り込み関数 エンコーダ出力が変化したらカウントアップする
void count() {
  enCountNum++;
}

```

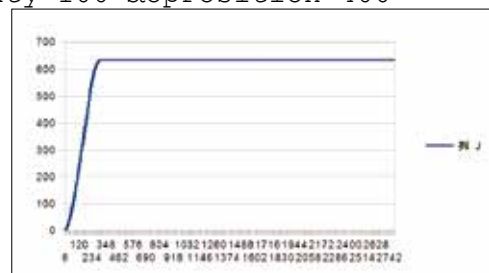
オーバーランの変化



- duty=70 stopPosition=500



- duty=100 stopPosition=400



- duty=100 stopPosition=500

■ソースコードは、PC/MyDocuments/Arduino/へ配置したサンプルフォルダ[RDS-X Examples_C]に、ファイル名【encoder_sample6_stop.ino】で格納されています。
 ■Arduinoを起動後、[ファイル]→[スケッチブック]→[RDS-X Examples_C]→[サンプルプログラム名]で開くと確認できます。

11.9.7. 指定ポジションで停止するPID制御ロボ

encoder_sample7_positionControl_PID for RoboDesigner
単相エンコーダ付ギアボックス用サンプルプログラムです
エンコーダの出力(HIGH/LOW)は割り込みでカウントし
ます。

モータをある位置で停止させたい時に、オーバーランを
回避するため、PID制御を応用して減速します。

エンコーダのカウントと到達目標の差(残りカウント数 =
偏差)を元にモータへの入力値(PWMのデューティ比 =
操作量)を変化させます。

デジタル/離散信号でPID制御するには、以下のような式
で表現できます。

操作量 = 前回の操作量 + 追加の操作量

追加の操作量 = $K_p \cdot \text{偏差} + K_i \cdot \text{偏差} + K_d \cdot \text{前回の偏差と}$
今回の偏差の差

今回の場合、目標値の差が大きくなりますので、通常は
maxDutyで回転し、目標までの偏差がmaxDeviation以下
(停止の直前の一定のカウント数)の間だけPID制御を
かけます。

また、目標値では必ずdutyはモータが回転しない小さな
数値になるので、操作量は加算していかず、今回の操作
量だけを使います。

偏差は、目標値 - 一定時間内(Ktime)のエンコーダのカ
ウント数です。

K_p が小さいと目標値に到達できず、大きいと目標を超え
てしまいます。目標値近くで出力が足りない状態が起こ
ります(残留偏差)。

残留偏差をなくすため、偏差に K_i を掛けた数値を操作量
に積算します。比例制御の目標近くで出力が足りない分
を補います。

前回の偏差と今回の偏差の差は、一定間隔をあけた偏差
同士の差です。間隔をとらないと、プログラム処理が速い
ため、常に1や0になってしまいます。

偏差の差に適切な K_d を掛けて、まだ目標に遠いときほど
大きく比例制御の出力を補い、応答性を向上させます。
最初はPのみになっているので、I、Dのそれぞれを順にコ
メントアウトを外して試してください。

動作手順

- ・実行開始時の位置をカウント数0に設定します(en-Counter= 0)。
- ・割り込みでエンコーダの出力を監視し、変化したらモータの回転方向に応じてカウント数 enCounter を増減します。
- ・プログラムを実行すると配列 targetCountArray[array-Position]で設定した到達目標カウント数までモータを回転させます。
- ・開始カウント数と目標カウント数の大小からモータの回転方向を決めて回転させます。
- ・モータが高速回転のままだと慣性のために目標カウント数で停止できませんので、PID制御で徐々に減速します。
- ・エンコーダ出力がしばらく変化しないかを value-NotChange-Countで判定し、モータの停止を確認します。
- ・次の到達目標カウント数に arrayPosition を進めます。

```

ファイル 編集 スケッチ ツール ヘルプ
encoder_sample7_positionControl

//機体とポート類の接続は 左:小さい番号←→大きい番号:右
// MotorDriver Pin Assign on RDC.
int M1_1 = 4;
int M1_2 = 5;
int M1_PWM = 6;
// Encoder Interrupt Pin Assign on RDC.
int encoder = 2; //D0
//動作の変化にかかわる定数
//float型は処理が遅くなるのでゲインKp, Ki, Kdは100倍見当
//で指定します。dutyにマッピングする前に1/100に戻します。
int maxDeviation; //PID制御を開始する偏差
int Kp = 60; //比例制御のゲイン。停止時の減速の程度が変化します
int Ki = 30; //積分制御のゲイン。停止時の減速の程度が変化します
int Kd = 30; //微分制御のゲイン。加速の程度が変化します
int Ktime = 20; //I, D制御のサンプリング間隔
int maxDuty= 200; //モータPWMデューティ比の最大値(
=回転速度)
int targetCountArray[]={100,200,300,1000,0};
//到達目標カウント数の配列
int arrayPosition; //到達目標配列の現在の要素位置
//その他の変数
int valueNotChangeCount; //エンコーダ出力変化
が無ければカウントアップし、モータの停止の判定に使用する
int startCount; //回転を開始するカウント数
int enCounter; //エンコーダの直近のカウント数
int deviationNew; //カウント数の最新の偏差 到達
目標カウント数 - エンコーダの直近のカウント数
int deviationOld; //カウント数の前回の偏差
int deviationDiff; //カウント数の偏差の前回と今回の差
int KtimeCount; //I, D制御のサンプリング間隔用のカウンタ
int operationVolume; //操作量
int operationVolumeLast; //前回の操作量
int nowDuty; //デューティ比の直近の値
// the setup routine runs once when you press reset:
void setup() {
// initialize serial communication at 9600 bits per second:
Serial.begin(9600);
while(!Serial);
// initialize the digital pin as an output.
pinMode(M1_1, OUTPUT);
pinMode(M1_2, OUTPUT);
pinMode(M1_PWM, OUTPUT);
attachInterrupt(encoder,
count,CHANGE);
//attachInterrupt(digitalPinToInterrupt(0), count,
CHANGE); //recommended syntax, but not function
arrayPosition = 0;
valueNotChangeCount = 0;
startCount = 0;
enCounter = 0;
deviationOld = 0;
deviationDiff = 0;
KtimeCount = 0;
operationVolumeLast = 0;
nowDuty = maxDuty; //初期のデューティ比を

```

```

maxDutyにする
}
// the loop routine runs over and over again forever:
void loop() {
//インクリメントして、エンコーダ出力に変化がない時間を計測する
valueNotChangeCount++; //
//目標値の大きさ(慣性)、制御によって影響が違いますので調整します。
if (targetCountArray[arrayPosition] -
startCount > 1000)
maxDeviation = 200;
if (targetCountArray[arrayPosition] -
startCount > 500)
maxDeviation = 150;
if (targetCountArray[arrayPosition] -
startCount <= 500)
maxDeviation = 100;
//PIDでPWMのデューティ比を調整する
deviationNew = targetCountArray[array-
Position]- enCounter;
deviationNew = abs(deviationNew);
if (deviationNew > maxDeviation) {
nowDuty = maxDuty; //PID制御で減速を始めるま
では最大速度で回転する
} else {
if (KtimeCount >= Ktime) {
deviationDiff = deviationOld -
deviationNew;
deviationOld = deviationNew;
KtimeCount = 0;
}
KtimeCount++;
//P,I,Dのそれぞれを順にコメントアウトを外して試してください。
operationVolume = deviationNew * Kp/10
0; //P制御
//operationVolume = operationVolume +
(deviationNew * Ki / 100); //I制御を加える
//operationVolume = operationVolume +
(deviationDiff * Kd / 100); //D制御を加える
//operationVolume = operationVolumeLast
+ operationVolume; //前回までの操作量に今回の操作量を加える
//operationVolumeLast = operationVolume;
//最新の操作量を前回の操作量に代入する
nowDuty = map(operationVolume, 0,
maxDeviation, 0, maxDuty);
}
//モータの回転方向の設定
if (0 < targetCountArray[arrayPosition]
- startCount) {
//カウントが到達目標より小さかったら正転する
digitalWrite(M1_1, HIGH);
digitalWrite(M1_2, LOW);
}
if (0 > targetCountArray[arrayPosition]
- startCount) {
//カウントが到達目標より大きかったら逆転する
digitalWrite(M1_1, LOW);
digitalWrite(M1_2, HIGH);
}
if (0 == targetCountArray[arrayPosi-
tion]- startCount) {
//カウントが到達目標と同じだったら停止する

```

```

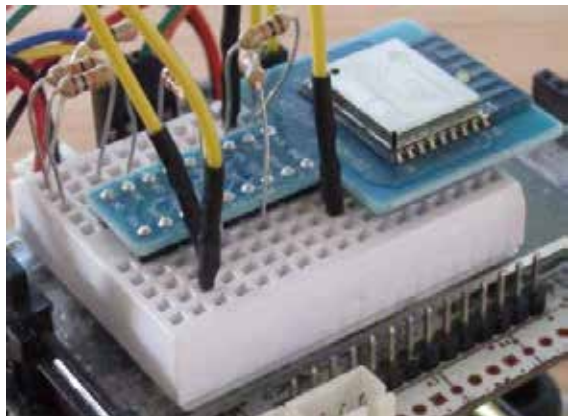
digitalWrite(M1_1, LOW);
digitalWrite(M1_2, LOW);
}
analogWrite(M1_PWM, nowDuty); //モータのPWM設定
//シリアルモニタチェック
Serial.print(" deviation= "); //カウント数の偏差
Serial.print(deviationNew);
Serial.print(" deviationDiff= "); //前回の
偏差との差
Serial.print(deviationDiff);
Serial.print(" operationVolume= ");
//エンコーダの出力値
Serial.print(operationVolume);
Serial.print(" duty= "); //エンコーダの出力値
Serial.print(nowDuty);
Serial.print(" target count= "); //目標カウント数
Serial.print(targetCountArray[array-
Position]);
Serial.print(" encoder count= ");
//エンコーダのカウント数
Serial.println(enCounter);
//モータが停止して一定時間エンコーダ出力に変化が無い
ことを確認したら配列の要素を一つ進める
if (100 < valueNotChangeCount) {
startCount = targetCountArray[arrayPo-
sition]; //スタートカウントは加算する
arrayPosition++;
valueNotChangeCount = 0;
deviationOld = 0;
KtimeCount = 0;
nowDuty = maxDuty;
//配列の最後だったら、最初の要素に戻って、カウント数もリセットする
if(arrayPosition == sizeof(target-
CountArray) / sizeof(targetCountArray[
0])){
arrayPosition = 0;
enCounter = 0;
}
delay(1000); //動作繰り返しの間の待ち時間 短かいとオーバーランします
}
//割り込み関数 エンコーダ出力が変化したら、正転ならカ
ウントアップ、逆転ならカウントダウンする
void count() {
if (0 < targetCountArray[arrayPosi-
tion]- startCount)
enCounter++;
if (0 > targetCountArray[arrayPosi-
tion]- startCount)
enCounter--;
valueNotChangeCount = 0;
}
}

```

■ソースコードは、PC/MyDocuments/Arduino/へ配置したサンプルフォルダ[RDS-X Examples_C]に、ファイル名[encoder_sample7_positionControl_PID.ino]で格納されています。

■Arduinoを起動後、[ファイル] → [スケッチブック] → [RDS-X Examples_C] → [サンプルプログラム名]で開くと確認できます。

11.10. Wi-Fi拡張 IoTロボ作成 RDS-X25enPiT



#用意する拡張用部材 (セットRDS-X25_enPiTには付属)

部品	品名等	入手先
	ESP-WROOM-02	http://akizukidenshi.com/catalog/g/gK-09758/
	ブレッドボード 基板カバー	http://akizukidenshi.com/catalog/g/gP-05155/
	プッシュリベット M3x10mm	ブレッドボードをコントローラへ取付時に使用
	スペーサ 3 mm	プッシュリベット取付時に使用
	接続用オス-メスケーブル	http://akizukidenshi.com/catalog/g/gC-08935/
	10KΩ抵抗 (酸化被膜抵抗)	http://akizukidenshi.com/catalog/g/gR-25103/
	0Ωジャンパー抵抗 (酸化被膜抵抗)	http://akizukidenshi.com/catalog/g/gR-25000/

#ESP-WROOM-02使用前設定作業

USBシリアル変換モジュールを使用して、ESP-WROOM-02の通信速度をデフォルトの115200から9600に変更します(RDC-103では高速だと通信できないため)。

FT232RL USBシリアル変換モジュールキット
<http://akizukidenshi.com/catalog/g/gK-06693/>

ESP-WROOM-02とUSBシリアル変換モジュールの結線

ESP-WROOM-02	FT232RL
3.3V	(外部電源 3.3V)
G	GND(外部電源 G)
EN --- 10KΩ抵抗 --->	(外部電源 3.3V)
RST --- 10KΩ抵抗 --->	(外部電源 3.3V)
TXD --- 0Ωジャンパー抵抗 --->	RXD(0)

ESP-WROOM-02	FT232RL
RXD --- 0Ωジャンパー抵抗 --->	TXD(1)
I00 --- 10KΩ抵抗 --->	(外部電源 3.3V)
I02 --- 10KΩ抵抗 --->	(外部電源 3.3V)
I015 --- 10KΩ抵抗 --->	(外部電源 G)

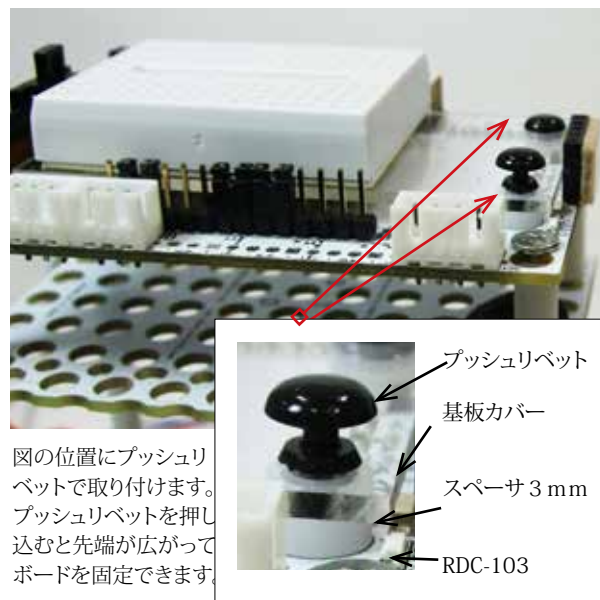
USBシリアル変換モジュールを接続してArduino IDEでポートを選択します。シリアルモニタを開き「CRおよびLF」「115200 baud」に設定して、以下のATコマンドを入力します。
AT+UART_DEF=9600,8,1,0,0

OKが返ってくれば設定完了。
**セットでRDS-X25_enPiTを購入された場合は、ESP-WROOM-02の通信速度をデフォルトの115200から9600に変更しています。

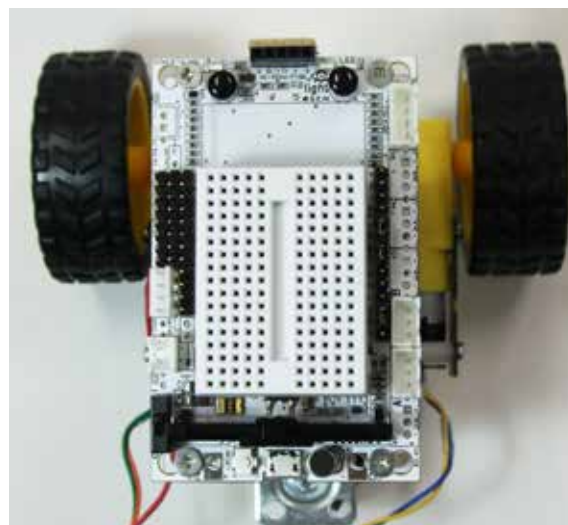
RDC-103 + ESP-WROOM-02でのIoT実験

enPiTモデル組み立て

RDC-103の基板に、ブレッドボードを取付けます。



図の位置にプッシュリベットで取り付けます。プッシュリベットを押し込むと先端が広がってボードを固定できます。

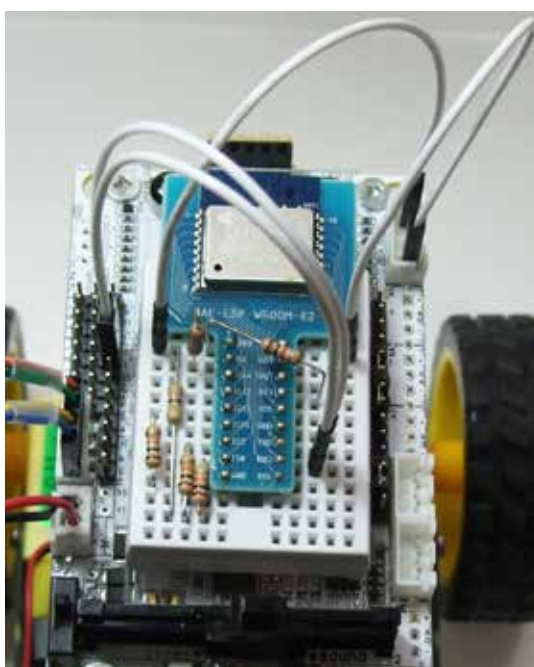
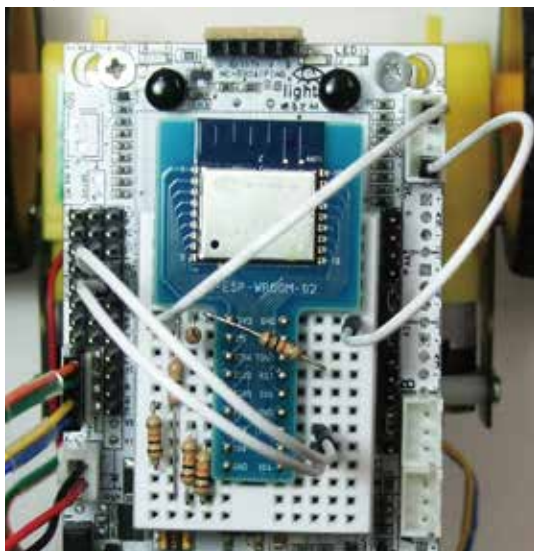


#ハードウェアの接続

RDC-103のブレッドボードにESP-WROOM-02を挿して、以下のように結線します。

ESP-WROOM-02	RDC-103
3.3V --- ジャンパー線--->	⊕
G --- ジャンパー線--->	⊖
EN --- 10KΩ抵抗--->	⊕
RST --- 10KΩ抵抗 --->	⊕
TXD --- ジャンパー線--->	RXD(0)
RXD --- ジャンパー線--->	TXD(1)
IO0 --- 10KΩ抵抗--->	⊕
IO2 --- 10KΩ抵抗 --->	⊕
IO15 --- 10KΩ抵抗 --->	⊖

注意 ⚠️...①抵抗のリード線は電気を通す導体です、リード線同士が接触をしないようにご注意ください。
②ブレッドボードの使い方については、テクニカルガイド章を参照ください。



(参考)

ESP-WROOM-02は以下の各ピンで起動時のモードを設定します。(1は⊕、0は⊖)

	IO0	IO2	IO15
UART Download Mode	0	1	0
Flash Startup Mode	1	1	0
Flashのプログラムを実行(工場出荷時のファームウェアでATコマンドで動作させる)			
SD-Card Boot Mode	0	0	1
外部SDから起動する場合(使用しません)			

RDC-103(32U4/leonard系)のハードウェアシリアルは0 RXD/1 TXD、Arduino IDEではserial1で指定します。

#ATコマンドで接続

ESP-WROOM-02へのプログラム書き込みは行わず、工場出荷時のファームウェアでシリアル通信/ATコマンドで使用します。
サンプルプログラムの「RDC_Serial」(スケッチの例「SoftwareSerialExample」を改変したもの)をRDC-103に書き込みます。
シリアルモニタを開くと、RDC-103とESP-WROOM-02の間で通信が始まります。
「CRおよびLF」「9600 baud」に設定します。
「AT」と送信してOKが返ってくれば正常に接続できています。

```
Goodnight moon!
Hello, world?
ERROR
```

```
AT
```

```
OK
注意:
```

(ESP-WROOM-02の通信速度は初期値115200ですがRDC-103では通信できないため9600に変更してあります。
高速側に変更すると、RDC-103からは戻せなくなりますので注意してください)

(参考)

ESP-WROOM-02/ESP8266の情報です。
Documentationの中にATコマンドのリストもあります。
<http://espressif.com/en/products/hardware/esp8266ex/resources>

ファームウェアバージョンの確認

```
AT+GMR
AT version:0.40.0.0(Aug 8 2015 14:45:58)
SDK version:1.3.0
compile time:Aug 11 2015 17:02:18
```

```
OK
```

ステーションモードのMACアドレス確認

AT+CIPSTAMAC?

+CIPSTAMAC:"18:fe:34:ee:9d:bf"

OK

APモードのMACアドレス確認

AT+CIPAPMAC?

+CIPAPMAC:"1a:fe:34:ee:9d:bf"

OK

アクセスポイントの確認

AT+CWJAP_CUR?

No AP

OK

モードの確認

AT+CWMODE=?

+CWMODE:(1-3)

OK

(参考)

設定ミスでシリアル通信ができなくなった場合、通信速度がわかっている場合はFactory Resetできる場合があります。

AT+RESTORE

#アクセスポイントに接続する

ステーションモードに設定

AT+CWMODE_CUR=1

OK

アクセスポイントに接続

AT+CWJAP_CUR="SSID","password"

WIFI CONNECTED

WIFI GOT IP

OK

IPアドレスの確認

AT+CIFSR

+CIFSR:APIP,"192.168.4.1"

+CIFSR:APMAC,"1a:fe:34:ee:9d:bf"

OK

アクセスポイントから切断

AT+CWQAP

OK

WIFI DISCONNECT

#webページのデータを取得する

ローカルでサーバを立てるか、研究室など接続して問題ないサイトをnslookupでIPアドレスを入手します。

<http://www.cman.jp/network/support/http.html>

JAPAN ROBOTECHのwebサーバにアクセスしてみます。

AT+CWMODE_CUR=1

OK

AT+CWJAP_CUR="SSID","password"

WIFI CONNECTED

WIFI GOT IP

OK

AT+CIPSTART="TCP","203.180.231.186",80
CONNECT

OK

AT+CIPSEND=21

OK

> / HTTP/1.1

busy s...

Recv 21 bytes

SEND OK

+IPD,391:HTTP/1.1 400 Bad Request

Date: Fri, 30 Dec 2016 06:43:00 GMT

Server: Apache

Content-Length: 225

Connection: close

Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD

HTML 2.0//EN"> <html><head> <title>400

Bad Request</title> </head><body> <h1>Bad

Request</h1> <p>Your browser sent a request

that this server could not understand.
 </p>

</body></html>CLOSED

AT+CWQAP

OK

WIFI DISCONNECT

(参考)

https://trac.switch-science.com/wiki/ESP-WROOM-02_AT

#仮想COMポートソフトウェアに接続して、トランスペアレントモードで文字列をやりとりする

仮想COMポートソフトウェアをWindowsマシンにインストールします。

USR-VCOM (Windows)

<http://www.usr.so/Product/72.html>

<http://www.usriot.com/usr-vcom-virtual-serial-software/>

TCPのポートは999などとします。

COMポート番号を適宜設定し、設定したCOMポートにTera Termなどのソフトウェアで接続します。

ipconfigでWindowsマシンのIPアドレスを調べます。

こちらのソフトは単体で通信内容の確認もできます。

Hercules SETUP utility (Windows)

http://www.hw-group.com/products/hercules/index_en.html

AT+CWMODE=1

OK

AT+CWJAP_CUR="SSID","password"

WIFI CONNECTED

WIFI GOT IP

OK

AT+CIFSR

+CIFSR:APIP,"192.168.4.1"

+CIFSR:APMAC,"1a:fe:34:ee:9d:bf"

+CIFSR:STAIP,"192.168.2.102"

+CIFSR:STAMAC,"18:fe:34:ee:9d:bf"

OK

TCPのポートに接続する

AT+CIPSTART="TCP","192.168.2.103",999

CONNECT

OK

トランスペアレントモードにする

AT+CIPMODE=1

OK

送受信開始

AT+CIPSEND

OK

>aaabbb

(参考)

androidでボタン操作できるような汎用アプリもあります。

- TCP Client (Android)

- WiFi TCP/UDP Controller (Android)

#IFTTTでIoTの実験をする

Web上の様々な機能／サービスを連携させる

「BaaS」と呼ばれるサービスが増えています。

今回はIFTTTを使ってArduinoと連携させてみます。

下記書籍のプログラムソースをRDC-103で動くように修正して使用します。

蔵下まさゆき(2016)『センサーでなんでもできる おもしろまじめ電子工作』p.297-314,秀和システム.

##IFTTTでアプレットを作成する

IFTTTにsign upします。

アカウントができれば、自分のメニューから「New Applet」を選択してアプレットを作成します。

thisをクリックして、Search servicesで「Maker」チャンネルを探します。

Receive a web requestで、Event Nameを決めて Create Triggerをクリックします。

次にthatをクリックして、「Notifications」チャンネルを探します。

Send a notificationを選択します。

先ほどのEvent Nameを入力し、Create actionをクリックします。

Finishをクリックすると完了です。

メニューのServicesから「Maker」を選んで、Settingsをクリックし、Maker ChannelのKeyを控えておきます。

<https://maker.ifttt.com/use/>ここにある文字列がKeyです

##スマートフォンの設定

Notificationsからの通知を受けるため、スマートフォンにIFTTTアプリをインストールしてください。

##RDC-103のプログラム

ATコマンドを直接触らずに操作するためのライブラリをArduino IDEにaddします。

https://github.com/exshonda/ESP8266_Arduino_AT

ESP8266_Arduino_AT-master.zipをダウンロードして、メニューのスケッチ>ライブラリを使用>Add

Library...を選択して追加します。

ライブラリはユーザの作成したファイルが格納されるArduinoフォルダの中の「library」の中に追加されます。

追加されたライブラリの中のESP8266.hをテキストエディタで開き、通信速度を9600に変更します。
void begin(HardwareSerial &uart, uint32_t baud = 9600);

サンプルプログラム「IFTTT_alarm」のアクセスポイント情報、Keyを書き換えて、RDC-103にダウンロードします。
スイッチを押すと、しばらくしてスマートフォンのアプリに通知が届きます。

RDC-103に入力するセンサや、IFTTTアプレットのthatを変えて実験してみましょう。

```
IFTTT_alarm | Arduino 1.0.5-r2
ファイル 編集 スケッチ ツール ヘルプ
IFTTT_alarm
// ライブラリの読み込み
#include "ESP8266.h"

// 転送レート
#define SERIAL_SPEED 9600

// Wi-Fi SSID
#define WLAN_SSID "SSID"
// Wi-Fi パスワード
#define WLAN_PASS "password"

// IFTTTのイベント名
#define IFTTT_EVENT_NAME "alarm"

// IFTTTのホスト名
#define IFTTT_HOST_NAME "maker.ifttt.com"

// IFTTTのシークレットキー
#define IFTTT_KEY "IFTTTのKEY"

// ポート番号
#define PORT_NUMBER 80

// Wi-Fiモジュール
ESP8266 wifi;

// センサーを接続するピン 12はボタンです。
const int sensorPin = 12;

void setup(void) {
```

```
// デジタル12番ピンはプルアップします
pinMode(sensorPin, INPUT_PULLUP);

// パソコンとのシリアル通信のポートを開ける
Serial.begin(SERIAL_SPEED);
while(!Serial); // for 32U4 シリアルモニタを
起動するまでプログラムはここから進みません
// Wi-Fiモジュールとのシリアル通信のポートを開ける
Serial.begin(SERIAL_SPEED);

// Wi-Fi設定
setupWiFi();
}

/*
Wi-Fiを設定します
*/
void setupWiFi() {
Serial.print("Connecting to ");
Serial.println(WLAN_SSID);

// シリアルポートの指定
wifi.begin(Serial1);

// Wi-Fiへ接続(成功するまで処理を繰り返す)
while(!wifi.joinAP(WLAN_SSID,
WLAN_PASS)) {
delay(500);
Serial.print(".");
}

Serial.println("Wi-Fi connect-
ed");
Serial.println("IP address: ");
// Wi-FiのローカルIPアドレスをシリアルモニターへ表示
Serial.println(wifi.getLocalIP().c_str());
}

void loop(void) {
// TCP接続を確立
if(wifi.createTCP(IFTTT_HOST_
NAME, PORT_NUMBER)) {
Serial.println("TCP success");

// 焦電センサーの状態を取得
int sensorState = digital-
Read(sensorPin);

// 動きを検知したら
if(sensorState == LOW) {
Serial.println("Send Data");
// IFTTTへ送信するデータ
char sendData[256] = "";
sprintf(sendData, "GET
```

```

http://maker.ifttt.com/trigger/%s/
with/key/%s HTTP/1.1\r\nHost:maker.
ifttt.com\r\nConnection: close\r\n\r\n
r\n", IFTTT_EVENT_NAME, IFTTT_KEY);
    Serial.println(sendData);

    // IFTTTへデータを送信
    wifi.send((const uint8_t*) send-
Data, strlen(sendData));
}
// TCP接続の確立失敗
else {
    Serial.println("TCP failure");
}

// 1秒処理を止める
delay(1000);
}
    
```

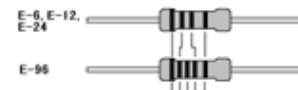


- ソースコードは、PC/MyDocuments/Arduino/へ配置したサンプルフォルダ[RDS-X Examples_C]に、ファイル名【IETTT_alarm.ino】で格納されています。
- ※ソースは、C言語で記述されています。
- Arduino を起動後、[ファイル] → [スケッチブック] → [RDS-X Examples_C] → [サンプルプログラム名] で開くと確認できます。

参考:配線に使用する炭素被膜抵抗(カーボン抵抗)

表示例	3桁表示	0Ω → 0R0(ジャンパー)	4桁表示	0.051Ω → R051
		1.5Ω → 1R5		1.47Ω → 1R47
		15Ω → 150		147Ω → 1470
		150,000(150k)Ω → 154		147,000(147k)Ω → 1473

公称抵抗値及び許容差の色表示 (JIS C 5062:1997)



色	第一数字	第二数字	第三数字	倍率 (10のべき数)	抵抗値 許容差
黒	0	0	0	10 ⁰	-
茶	1	1	1	10 ¹	F(±1%)
赤	2	2	2	10 ²	G(±2%)
橙(黄赤)	3	3	3	10 ³	-
黄	4	4	4	10 ⁴	-
緑	5	5	5	10 ⁵	D(±0.5%)
青	6	6	6	10 ⁶	C(±0.25%)
紫	7	7	7	10 ⁷	B(±0.1%)
灰色	8	8	8	10 ⁸	-
白	9	9	9	10 ⁹ (10 ⁻³)*	-
金色	-	-	-	10 ⁻¹	J(±5%)
銀色	-	-	-	10 ⁻²	K(±10%)
色を つけない	-	-	-	-	M(±20%)

*巻線抵抗器の場合、10⁻³として用いる。

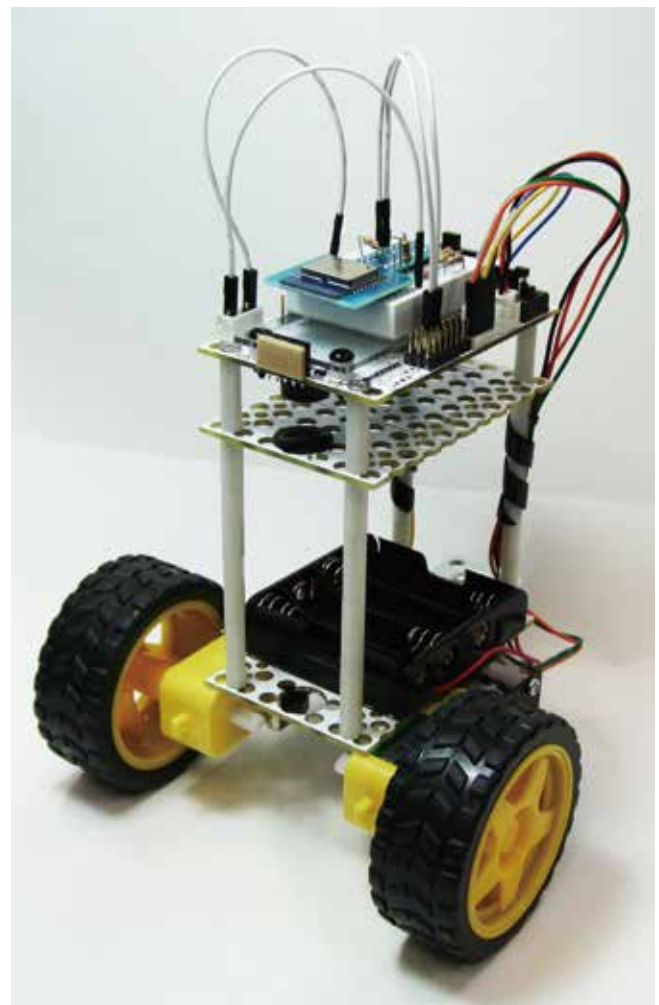
チップ抵抗器 RC1608 E-96 数列の3桁表示 (部品上面印字のみ)

RC1608は本体印字スペースが小さいため、抵抗値記号がE-96数列(4桁)のもの部品上面印字については、表2“E-96数列2桁化対応記号”の2桁数字と表1の乗数記号を組合せた3桁で表示する。

表示例	12.1Ω → 09X	3.320(3.32k)Ω → 51B
	475Ω → 66A	28.700(28.7k)Ω → 45C

表1. 乗数記号

表示記号	A	B	C	D	E	F	G	H	X	Y	Z
乗数	10 ⁰	10 ¹	10 ²	10 ³	10 ⁴	10 ⁵	10 ⁶	10 ⁷	10 ⁻¹	10 ⁻²	10 ⁻³



オドメトリを用いた軌道制御 ～理論にもとづいた制御への発展～

東洋大学 理工学部 機械工学科
山川聡子

ロボットを与えられた目標の軌道に沿って走行させることを考えましょう。これを達成するために、制御工学の「理論」にもとづいた制御プログラムを考えていきます。「理論」と聞くと数学や物理などを思い描いて身構えてしまうかもしれませんが、また、経験的に試行錯誤しながら調整することでロボットが動くようになるのであれば、それで十分だと思えるかもしれません。しかし、そのような方法には限界があります。システムが複雑になると、動きのバリエーションは増えていきますし、変更することができる(=決定しなければならない)制御パラメータの組み合わせも増えていきますから、良い解を勘で見つけられる確率はどんどん低くなっていくでしょう。それに対して、理論にもとづいて考えるというアプローチには利点があります。例えば、理論にもとづいて考えると、ある程度ロボットの動きを予想することができます。また、上手くいかなかった場合にはその理由を想像することができます。理由がわかれば、上手くいかない解を最初から省いて検討することができますし、良い解を得るための道筋が見つかることもあります。その結果、制御方法を定めるための試行錯誤の回数が減って開発期間や労力が削減されます。さらに、理論的に体系だてて考えることで、問題点を明確にし、経験だけでは導けなかった解が得られることもあります。もちろん、現実の問題はすべてが理論通りに行くわけではありません。それでも、理論的なアプローチは論理的な設計の道筋を示してくれることでしょう。闇雲に試行錯誤を繰り返すのではなく、分かっている知識を利用して、無駄を省き、より良い道筋を目指しましょう。

体系化されている制御理論には、古典制御¹⁾、現代制御²⁾、デジタル制御など、いくつかの種類があります。制御したい対象、状況や目的によってどの理論を使うと良いかは変わります。ここでは、古くから用いられ、今も広く使用されているPID制御を用いて、ロボットの軌道を制御してみましょう。

自動制御によって目標軌道に沿ってロボットを動かすわけですが、目標の位置に行きたいと思っても、今、ロボットがどこにいるのかが分からなければ、目標の場所に向かうことはできません。そこで、制御方法を学ぶ前に、ロボット自身が今どこにいるかを推定する方法について学びましょう。

1 ロボットの位置推定

ライントレースロボットでは、ラインのうえを走っているかどうかで自分の位置を判断しています。一方、自動車などではいつでも軌道が目で見えるような線で描かれているわけではありません。このような場合は、例えばGPSを用いて自分の位置を判断することがあります。しかし、トンネルに入ったときなどはGPS衛星からの信号が受信できなくなります。このようなときは最初の位置からどれくらい走行したかによって今の位置を推定します。これをオドメトリ(odometry)といいます。自動車はタイヤが一回転すれば、タイヤの直径×円周率だけ進みます。片輪だけが回転すれば旋回します。つまり、左右のタイヤがどれだけ回転したかを測定して積算すれば、今どこにいるかを推定できるわけです。

対象としているロボットにはエンコーダが搭載されていますから、タイヤの回転角度を測定することができます。この測定値を利用して水平面内での位置を推定しましょう。

1-1 ロボットの運動モデル

まず、ロボットのタイヤは地面に対して滑らないと仮定します。もし、タイヤが滑ってしまうとタイヤが回転していてもロボットの位置が変化してしまいます。この場合にはそもそもタイヤの回転角度からロボットの位置を推定する

ことはできません。別のセンサ（加速度センサなど）が必要となります。タイヤが地面に対して滑らないという仮定は、自動車が凍った路面でスリップする場合や、急発進や急ブレーキをかけたときにタイヤが空転している場合などでは満たされません。一方、建物の廊下などを比較的ゆっくり走るロボットでは満たされると考えることができます。つまり、ゆっくり走行するロボットの制御を考える問題においては、この仮定は「妥当」だと言えます。

さて、この仮定の下、ロボットの速度と位置の関係を考えてみましょう。対象としているロボットには車輪が2つ取り付けられていて、この左右の車輪の回転速度を制御することで、ロボットを前進させたり、旋回させたりすることができます。ロボットの位置を表すために、地上に固定した座標系を定義します。この座標系におけるロボットの中心位置（＝左右の車輪の中心位置）を (x, y) とします。また、この座標系の x 軸に対するロボットの向き（姿勢角）を θ とおきます。中心位置 (x, y) が変化する速度を並進速度と呼び、これを v_1 とします。ロボットの向きが中心位置に対して回転する速度を旋回角速度と呼び、これを v_2 とおきます（図1）。

ロボットの並進速度 v_1 を x 軸、 y 軸方向に分解した成分 v_x 、 v_y は、ロボットの向き θ を用いて、それぞれ、

$$v_x = v_1 \cos \theta \quad (1-1)$$

$$v_y = v_1 \sin \theta \quad (1-2)$$

です。位置を時間で微分したものが速度ですから、 x 軸方向の成分についていえば、 $dx/dt = v_x$ が成り立ちます。したがって、ロボットの位置 (x, y) の時間変化は、

$$\frac{dx}{dt} = v_1 \cos \theta \quad (1-3)$$

$$\frac{dy}{dt} = v_1 \sin \theta \quad (1-4)$$

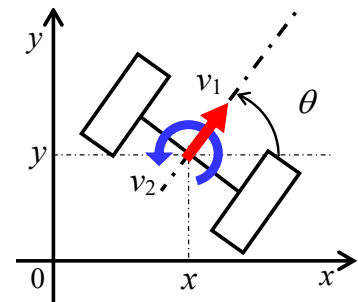


図1 ロボットの位置と姿勢

という式で表わされます。一方、姿勢角は、

$$\frac{d\theta}{dt} = v_2 \quad (1-5)$$

という式にしたがって変化していきます。(1-3)～(1-5)式をあわせたものが二輪車両の運動モデルとしてよく用いられる式です。

ところで、時間によって値が変化する関数 $x(t)$ の時間微分の定義は、

$$\frac{dx(t)}{dt} = \lim_{\Delta t \rightarrow 0} \frac{x(t) - x(t - \Delta t)}{\Delta t} \quad (1-6)$$

です。ここで Δt が十分小さいと仮定して次のように近似しましょう。

$$\frac{dx(t)}{dt} \approx \frac{x(t) - x(t - \Delta t)}{\Delta t} \quad (1-7)$$

つまり、 Δt は十分小さいから、(1-6)式の右辺の極限を取っても取らなくても同じだと考えるわけです。(1-7)式を移項して変形すると、

$$x(t) = x(t - \Delta t) + \Delta t \frac{dx(t)}{dt} \quad (1-8)$$

となります。この右辺2項目に(1-3)式を代入すると、

$$x(t) = x(t - \Delta t) + \Delta t (v_1(t) \cos \theta(t)) \quad (1-9)$$

となります。 $y(t)$ 、 $\theta(t)$ についても同様に、

$$y(t) = y(t - \Delta t) + \Delta t (v_1(t) \sin \theta(t)) \quad (1-10)$$

$$\theta(t) = \theta(t - \Delta t) + \Delta t v_2(t) \quad (1-11)$$

が得られます。(1-9)~(1-11)式の左辺は時刻 t での位置と角度です。一方、右辺は時刻 $t-\Delta t$ での位置と角度、それに時刻 t での速度の項です。つまり、過去の情報と測定した速度の情報から、時刻 t での位置と角度を計算する式です。この計算を繰り返していくと、初期時刻 $t=0$ での位置と角度 $x(0)$, $y(0)$, $\theta(0)$ と各時刻での速度 $v_1(t)$, $v_2(t)$ から、各時刻での位置と角度を得ることができます。

1-2 エンコーダを用いた速度計測

ロボットの速度 v_1 と v_2 を左右の車輪モータについているエンコーダの測定値から算出しましょう。まず、エンコーダは一定の角度回転するごとにパルス信号を発生します。このパルス数をカウントすることでモータの回転角度を測定することができます。多くの場合、モータと車輪の間には減速させるためのギア（歯車）が組み込まれていて、車輪の回転数はモータの回転数の定数倍になります。ここでは、車輪が一回転するときにエンコーダが C 回のパルス信号を発生するとします。この C の値はエンコーダの仕様とモータのギア比で決まる定数なので、対象とするロボットの部品の仕様書などを見て調べてください。エンコーダが発生したパルス数をカウントしたものをエンコーダ値と呼ぶことにします。

エンコーダ値が1増えた場合、その間に車輪は $2\pi/C$ [rad] だけ回転します。時刻 $(t-\Delta t)$ から t までの間に右車輪のエンコーダ値が $encR(t-\Delta t)$ から $encR(t)$ に変化しましたとします。このとき、右車輪は角度、

$$\Delta\phi_r = \frac{2\pi}{C}(encR(t) - encR(t-\Delta t)) \quad (1-12)$$

だけ回転します。車輪の半径を r とすると右車輪の接地点が地面に対して進む距離 ΔL_r は、角度 \times 半径 r なので、

$$\Delta L_r = \frac{2\pi r}{C}(encR(t) - encR(t-\Delta t)) \quad (1-13)$$

です(図2)。同じく時刻 t での左側のエンコーダ値を $encL(t)$ とおくと、左車輪が進む距離 ΔL_l は、

$$\Delta L_l = \frac{2\pi r}{C}(encL(t) - encL(t-\Delta t)) \quad (1-14)$$

です。 Δt の間に ΔL_r , ΔL_l 進むわけですから、左右の車輪の接地点での地面に対する速度は、それぞれ

$$\begin{aligned} v_r(t) &= \frac{\Delta L_r}{\Delta t} \\ v_l(t) &= \frac{\Delta L_l}{\Delta t} \end{aligned} \quad (1-15)$$

です。ロボットの並進速度 v_1 とは左右車輪の中心位置の速度、つまり左右の車輪の平均速度です。したがって、(1-13)~(1-15)式を用いれば、並進速度 $v_1(t)$ は

$$\begin{aligned} v_1(t) &= \frac{v_r(t) + v_l(t)}{2} \\ &= \frac{\pi r}{C} \frac{(encR(t) + encL(t)) - (encR(t-\Delta t) + encL(t-\Delta t))}{\Delta t} \end{aligned} \quad (1-16)$$

と得られます。一方、 Δt での姿勢角度の変化 $\Delta\theta$ は、左右の車輪間の距離が d であるとき、図3から、

$$\Delta\theta(t) = \frac{\Delta L_r(t) - \Delta L_l(t)}{d} \quad (1-17)$$

です。したがって、旋回角速度 v_2 は、

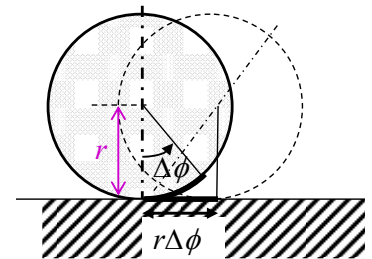


図2 車輪が進む距離

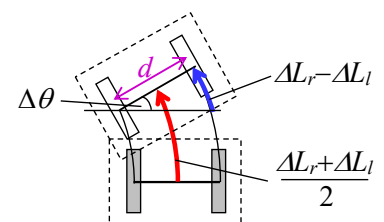


図3 左右車輪の移動距離の平均と差

$$v_2(t) = \frac{\Delta\theta(t)}{\Delta t} = \frac{2\pi r (encR(t) - encL(t)) - (encR(t - \Delta t) - encL(t - \Delta t))}{Cd \Delta t} \quad (1-18)$$

と得られます。以上のように、左右のエンコーダ値からロボットの速度 v_1 , v_2 を算出することができます。

1-3 プログラムでの実現

第 1-1 節では並進速度 v_1 と旋回角速度 v_2 から状態 x , y , θ が計算できることがわかりました。また、第 1-2 節では、エンコーダ値を使って速度 v_1 , v_2 を算出する方法を説明しました。これらを組み合わせて、エンコーダ値から x , y , θ を求めるプログラムを書いてみましょう。

まず、角度を計算する(1-11)式に(1-18)式を代入すると、

$$\theta(t) = \theta(t - \Delta t) + \frac{2\pi r}{Cd} ((encR(t) - encL(t)) - (encR(t - \Delta t) - encL(t - \Delta t))) \quad (1-19)$$

となります。この代入を繰り返して遡っていくと、途中の時刻でのエンコーダ値は打ち消しあって、

$$\theta(t) = \theta(0) + \frac{2\pi r}{Cd} (encR(t) - encL(t)) \quad (1-20)$$

が残ります。ただし、初期時刻の $encR(0)$, $encL(0)$ は 0 であるとしています。角度は繰り返し計算をしなくても初期時刻の角度 $\theta(0)$ と現時刻のエンコーダ値から求めることができます。

一方、位置については(1-9)(1-10)式で各時刻の角度 $\theta(t)$ に依存した $\cos\theta(t)$ や $\sin\theta(t)$ が掛け算されていますので、(1-20)式のように途中の値は打ち消されません。そのため、繰り返し計算を行って積算していきます。プログラムに何度も同じ計算を書くのは見づらくなるので、現時刻 t での左右エンコーダ値の和を sum とおくことにします。

$$sum = encR + encL \quad (1-21)$$

繰り返し計算の一回分の処理にかかる時間を Δt とし、 $(t - \Delta t)$ 時刻での値を sum_0 とおくことにします。同様に、時刻 t での状態を x , y とし、1 回前の $(t - \Delta t)$ 時刻の状態をそれぞれ x_0 , y_0 と添え字 0 をつけて表わすことにします。プログラム開始時には、 $t = 0$ での位置を x_0 , y_0 として定義しておきます。これを初期値と言います。つぎに、(1-9)(1-10)式、および(1-16)式を用いて、現時刻 t の状態をエンコーダの値からつぎのように計算します。

$$x = x_0 + \left(\frac{\pi r}{C} (sum - sum_0) \right) \cos \theta \quad (1-22)$$

$$y = y_0 + \left(\frac{\pi r}{C} (sum - sum_0) \right) \sin \theta \quad (1-23)$$

この式では Δt は打ち消し合ってあられなくなりませんから、処理にかかる時間 Δt を計測する必要はありません。しかし、実際には Δt が十分小さくない場合には(1-7)式の近似が成り立たなくなり、実際の位置と計算値に誤差が生じるので気にとめておく必要があります。

さて、繰り返し計算を行なうとき、つぎの計算を行う時点では現時刻の状態を一回前の状態として扱います。そのため、繰り返し計算の最後で変数の更新を行っておきます。すなわち、 x_0 , y_0 および sum_0 に現時刻の状態を代入して更新し、つぎの計算に備えます。

$$sum_0 = sum \quad (1-24)$$

$$x_0 = x \quad (1-25)$$

$$y_0 = y \quad (1-26)$$

以上の(1-21)～(1-26)式の計算を繰り返せば、自己位置を計算することができます。

2 目標軌道への追従制御

自己位置の情報を用いて、水平面内の関数として与えられた目標軌道 $y_r(x)$ に追従させることを考えましょう。 $y_r(x)$ を定数にすると、図4のように自動車のレーンチェンジのイメージになります。

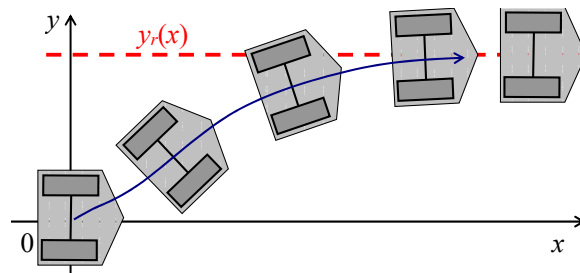


図4 目標値への追従制御

2-1 並進速度のPI制御

まず、並進速度の制御を考えます。ここでは一定の目標速度での走行を目指すこととし、この目標速度を v_d とします。今の並進速度が目標の v_d よりも遅ければ加速させ、速ければ減速させることで速度を目標に近づけます。ロボットの今の並進速度 $v_1(t)$ はエンコーダの値を用いて(1-16)式で求められるのでした。この値を用いて並進方向の入力 $u_1(t)$ を

$$u_1(t) = K_P (v_d - v_1(t)) \quad (2-1)$$

とすることにします。 K_P は正の定数で、比例ゲインと言います。(2-1)式のように、目標値との差(偏差)に定数を掛けて制御入力を決める方法を比例制御 (*proportional control*) と言います。(2-1)式の制御法を用いると、速度 $v_1(t)$ が目標値 v_d よりも小さいとき、入力 $u_1(t)$ は正の値になってモータの速度を加速させます。逆に $v_1(t)$ が v_d より大きいときは、負の値になってモータの速度を減速させます。また、 v_d と $v_1(t)$ の差が大きいときは $u_1(t)$ の絶対値は大きくなりますし、 v_d と $v_1(t)$ の差が小さくなれば $u_1(t)$ の絶対値は小さくなります。

比例制御(2-1)はとても基本的な制御方法です。この方法を使えば、目標から離れているときはたくさん入力を与え、近づいたら入力を小さくするので調子が良さそうです。しかし、目標に近づいて入力を小さくしたせいでずっと平行線を辿って目標に到達しないことがあります。こんなときは過去の履歴を見てみましょう。例えば、偏差が一定値であっても偏差の積算値は時間とともに大きくなります。ずっと目標に一致しないまま時間が経過している場合にはこの偏差の積算値を用いることで状況が良くなる可能性があります。

$$u_1(t) = K_P (v_d - v_1(t)) + K_I \int_0^t (v_d - v_1(\tau)) d\tau \quad (2-2)$$

(2-2)式の二項目を積分制御 (*integral control*) と言います。 K_I も正の定数であり、積分ゲインと言います。(2-2)式は比例制御と積分制御を組み合わせているので、頭文字を用いてPI制御 (*proportional-integral control*) とよべれます。

2-2 旋回角速度のPID制御

つぎに、目標軌道と y 座標の差に応じてロボットの向きを変えるように旋回方向の入力 u_2 を決めましょう。目標 y_r と今の位置 $y(t)$ との差、つまり偏差 (*error*) を

$$e(t) = y_r - y(t) \quad (2-3)$$

とおきます。前節の並進速度の制御と同じようにPI制御を用いると、

$$u_2(t) = k_p e(t) + k_i \int_0^t e(\tau) d\tau \quad (2-4)$$

です。 k_p , k_i はそれぞれ、比例ゲイン、積分ゲインです。

ところで、もし順調に目標に近づいている状況ならば、それ以上入力を増やさなくても自然に目標に到達するかもしれません。逆に目標から遠ざかっているときには、もっと大きな入力を入れたほうが早く目標に近づくかもしれません。このように偏差の変化にもとづいて制御入力の値を決める方法が微分制御 (*derivative control*) です。現時点での偏差の変化、つまり微分を用いて、

$$u_2(t) = k_p e(t) + k_i \int_0^t e(\tau) d\tau + k_d \frac{de(t)}{dt} \quad (2-5)$$

とします。(2-5)式の3項目の k_d を微分ゲインといいます。このように比例制御、積分制御、微分制御を組み合わせた方法を PID 制御 (*proportional-integral-derivative control*) と呼びます。PID 制御は古典制御の代表的な制御方法であり、その簡便さから実際によく用いられている方法です。

PID 制御で用いられる偏差、偏差の積分値、偏差の微分値を図5のようにになります。偏差 $e(t)$ は現在の測定値から決まります。積算値は過去の情報を使います。そのため、実際に使う際には過去の情報を保存するメモリが必要になります。一方、微分値は厳密には一瞬未来の値が必要です。しかし、現実的には未来の値はわかりませんので、プログラムで実現するときは、第 1-1 節での説明と同様に Δt が十分小さいとして近似的に直前の値と現在値との差を使って求めたりします。

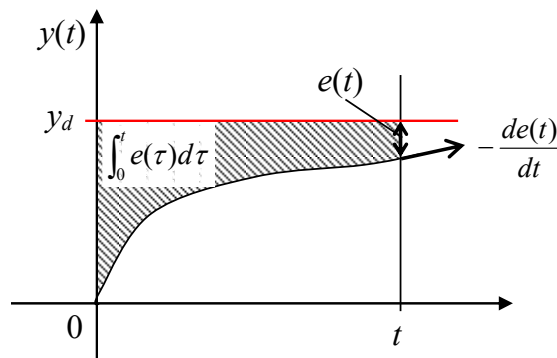


図5 比例、積分、微分要素

(2-5)式に含まれる制御ゲイン k_p , k_i , k_d の選び方によって、目標軌道への追従の仕方は変化します。これを実験で調べてみましょう。実際に使用したプログラムは第 2-4 節に示します。このプログラムでは並進入力の制御ゲインを $K_P=2.0$, $K_I=8.0$ としています。旋回入力の制御ゲインを① $k_p=30$, $k_i=0$, $k_d=45$, ② $k_p=30$, $k_i=10$, $k_d=45$ として実験を行ったときのロボットの軌道を図6に示します。この実験例では PD 制御では定常偏差が残ってしまいました(黒線)。そこで、積分制御を加えて PID 制御にすると、定常偏差がほぼ0になりました(青線)。ただし、積分制御を入れた場合は一度目標値を行き過ぎています。自転車や自動車の運転でコース変更を行なうときに、急ハンドルになって軌道が揺れてしてしまうことに似ています。よりうまく軌道に追従するように、制御ゲインを調整してみてください。

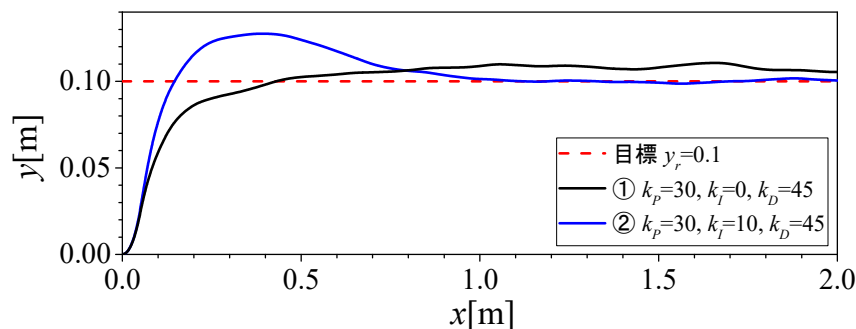


図6 実験でのロボットの軌道例 (PID 制御)

2-3 軌道制御の理論的な考察

PID 制御では、多くの場合、経験にもとづいてゲインを決めます。例えば、一般に積分制御を入れることで十分時間が経ったときの偏差（定常偏差）の改善が期待できます。比例制御や微分制御は目標値に近づく速さを調整します。ゲインの組み合わせによっては、ロボットは全く軌道に追従しなくなってしまうこともあります。PID ゲインを決める方法には、経験的な知見にもとづいた限界感度法¹⁾や、運動モデルをたてて理論的な知見にもとづいて決める極配置法²⁾などがあります。制御理論を学ぶと、例えば、微分制御を入れるべきかどうか、おおよそどのようにゲインを選んだら目標を行き過ぎる現象（オーバーシュート）を抑えられるかなどが、実験を行う前に予想できます。

ここでは、古典制御を学んだ人のために、古典制御の考え方と運動モデル(1-3)~(1-5)式にもとづいてロボットの軌道制御について考察してみましょう。まず、古典制御の考え方を用いるために、

仮定 1 : 並進速度は一定値であり、正の値とする。

仮定 2 : 姿勢角度 θ は十分小さい。

とします。仮定 1 は第 2-1 節の制御が十分機能している場合、ある程度の時間以降では妥当だと考えていいでしょう。仮定 2 は急な旋回をすることなく、直線に沿って移動している場合には満たされると考えていいでしょう。

仮定 1 および 2 が成り立つとすると、(1-3)式は、

$$\frac{dx(t)}{dt} = v_1 \text{ (const.)}$$

となるので、 x 方向の位置は $x(t) = v_1 t$ で変化します。 y 方向の運動は、(1-4)式の両辺を微分すると、

$$\begin{aligned} \frac{d^2 y(t)}{dt^2} &= \dot{v}_1(t) \sin \theta(t) + v_1(t) \cos \theta(t) \cdot \frac{d\theta(t)}{dt} \\ &\approx v_1 \frac{d\theta(t)}{dt} \\ &= v_1 v_2(t) \end{aligned} \tag{2-6}$$

という式で表わされます。したがって、 v_2 から y までの伝達関数は、

$$\frac{Y(s)}{V_2(s)} = v_1 \frac{1}{s^2} \tag{2-7}$$

であり、図 7 のように PD 制御、すなわち、

$$v_2 = k_p (y_r - y(t)) + k_d (\dot{y}_r - \dot{y}(t)) \tag{2-8}$$

を用いると、 y_r から y までの伝達関数は、

$$\begin{aligned} \frac{Y(s)}{Y_r(s)} &= \frac{(k_p + k_d s) v_1 \frac{1}{s^2}}{1 + (k_p + k_d s) v_1 \frac{1}{s^2}} \\ &= \frac{v_1 k_d s + v_1 k_p}{s^2 + v_1 k_d s + v_1 k_p} \end{aligned} \tag{2-9}$$

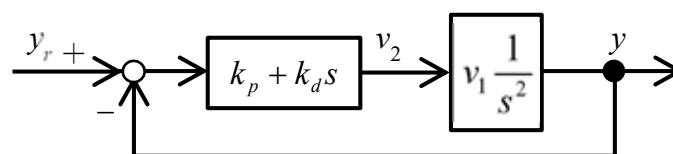


図 7 フィードバック制御系

となります。特性多項式が $s^2+v_1k_d s+v_1k_p$ となりますから、システムを安定にするためには $k_p, k_d > 0$ であることが必要です。もし、制御則(2-8)に微分項を入れない ($k_d=0$ とする) とシステムが安定限界となり、軌道が振動して目標値に収束しないことが予想できます。

(2-7)式のシステムにPD制御(2-8)を用いた場合は、十分時間が経ったときに一定の目標軌道 y_r に収束することが期待できます。これは、(2-9)式に最終値の定理を用いれば確認することができます。しかし、実際に実験を行うと前節の図6の①の場合のように、偏差が残ってしまうことがあります。この原因の一つとして、前述のモデルではモータの動特性や摩擦などを考えていないことが考えられます。そこで、前節の実験では積分項も加えて(2-5)式のようなPID制御を用いて定常偏差を0に近づけています。

ところで、実際には並進速度 v_1 は一定値ではありません。目標値 v_d に収束するまでの間や外乱（例えば床の摩擦の変化）などによって v_1 が変動した場合は仮定1が満たされず、(2-5)式の制御則では軌道が変動してしまいます。また、図6の実験例では途中で角度 θ が60度近くになっており、仮定2が満たされず、実際のロボットの y 方向の変化は(2-7)式の線形モデルの挙動とは異なっています。そのため、線形モデルでの知見にもとづいて制御ゲインを選定することには限界があります。しかし、このような仮定1、仮定2が満たされない場合でも、(1-3)~(1-5)式に含まれる非線形性を考慮して設計した制御則を用いるとよりうまく軌道制御を行うことができます。興味がある人は参考文献3を見てください。文献3の制御法を適用した場合の走行軌道例を図8に紫の太線で示します。PID制御の青線よりも定常状態での軌道の変動が減少しています。また、この実験例ではオーバーシュートをしないように制御ゲインを調整しましたが、その調整も比較的容易でした。同じロボットでも制御法や制御パラメータをうまく選ぶことで走行軌道を改善できる可能性があります。いろいろな制御法を学んで試してみてください。

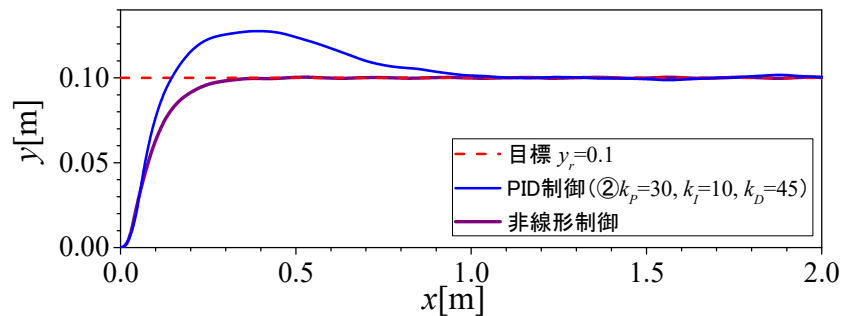


図8 別の制御法を用いた場合の軌道の例（非線形制御）

2-4 PID制御のプログラム

制御則(2-5)を実現するためのプログラムを示します。

今回用いるロボットでは速度を直接入力できるわけではないので、以下の制御プログラムでは計算した制御入力 u_1, u_2 に km という定数をかけて入力としています。 km は実際の入力値にあわせて選んでいます。

繰り返し計算について第 1-3 節で説明しましたが、C 言語のプログラムでは $x = x + \alpha$ という記述が右辺の値を左辺に代入するという処理であることを利用して、(1-22)式と(1-25)式をまとめて処理しています。そのため、 x_0 の定義は不要です。 y_0 についても同様です。

シリアル通信でデータを PC へ送信しています。送信するデータ数やデータの型によって通信に要する時間が変わりますので、繰り返し計算において 1 回の処理にかかるサンプリング時間 Δt が変わります。シリアル通信を行わない場合のように Δt が短すぎると、今回用いるロボットではエンコーダの読み取りが間に合わず、エンコーダの読み飛ばしが生じることがあります。この場合、オドメトリで推定した値に対して、実際の走行距離が長くなってしまいます。一方、通信量を増やしすぎてサンプリング時間 Δt を長くすることは、第 1-1 節で説明した位置計算の精度を下げるだけでなく、制御入力の更新を間引くことにもなりますので、軌道追従性などの制御性能が悪くなります。

このプログラムは、ロボットの基板の中ほどにあるボタンを押してから 0.5 秒後にロボットが動き始めるようにしています。PC でデータを取得するときは、PC の Arduino のメニューでシリアルモニタを起動してからロボットのボタンを押してください。ロボットはプログラムで指定した $xd[m]$ だけ走行して停止します。この後、同じプログラムで実験を繰り返してデータを取りたい場合には、PC 上のシリアルモニタを終了してからロボットのリセットスイッチを押してください。その後、改めてシリアルモニタを起動してロボットのボタンを押せばロボットが再スタートします。

```
/*
*****
Tracking controller for Robo Designer   by S.Yamakawa 20170721
RDS-X25 用, 開発環境は Arduino 1.0.5-r2
PID 制御を用いて目標軌道  $y_r(x)$  に追従させる
*****
//=====ロボットのパラメータ (※ロボットに合わせて変更) =====
float d = 0.118;           //タイヤ間距離[m]
float r = 0.064;           //タイヤ直径[m]
float a = r*3.14159265/768.0; //1pulse あたり進む距離[m], ギア比 48×1 回転 16pulse
//=====ロボット制御の目標値と制御ゲイン (※自分で設定・調整する) =====
float vd = 0.1;           //並進速度目標値 [m/s]
float yr = 0.1;           //目標軌道 [m]
float xd = 2.0;           //x 軸方向目標位置 [m]
float kp = 30;            //比例ゲイン
float ki = 10;            //積分ゲイン
float kd = 45;            //微分ゲイン
//=====基板のピン設定=====
int M1_1 = 4;
int M1_2 = 5;
int M1_PWM = 6;
int M2_1 = 7;
int M2_2 = 8;
int M2_PWM = 9;
int P_PUSH = 12;
int encoder1 = 2;
int encoder2 = 3;
//=====オドメトリ・制御用パラメータ=====
volatile int enCounter_r, enCounter_l; //エンコーダの直近のカウント数
int sum = 0; // (右エンコーダ) + (左エンコーダ) のカウント値
int sum0 = 0; //1 時刻前の値
float vr = 0.0; //右タイヤの目標速度 [m/s]
```



```

float v1 = 0.0; //左タイヤの目標速度[m/s]
float x = 0.0; //車軸中心位置[m]
float y = 0.0; //車軸中心位置[m]
float theta = 0.0; //車両向き[rad]
float theta_old = 0.0;
float v1 = 0.0; //並進速度[m/s]
float v2 = 0.0; //旋回角速度[rad/s]
float u1, u2; //並進, 旋回角速度入力[m/s]
float v1i = 0.0; //v1の偏差の積算値
float e; //偏差
float e0 = yr; //1時刻前の偏差
float ei = 0.0; //偏差の積算値
float ed = 0.0; //偏差の微分値
long time; //時刻[ms]
long time_old = 0; //一回前の時刻[ms]
long dt = 0; //サンプリング時間[ms]
float temp;

/*****
初期化处理
*****/
void setup(){
  Serial.begin(9600);
  pinMode(M1_1, OUTPUT);
  pinMode(M1_2, OUTPUT);
  pinMode(M1_PWM, OUTPUT);
  pinMode(M2_1, OUTPUT);
  pinMode(M2_2, OUTPUT);
  pinMode(M2_PWM, OUTPUT);
  pinMode(P_PUSH, INPUT_PULLUP);
  attachInterrupt(encoder1, count_l, CHANGE); //エンコーダ値取得開始
  attachInterrupt(encoder2, count_r, CHANGE);
  enCounter_r = 0; //エンコーダ初期値設定
  enCounter_l = 0;
  //=====ボタンを押してから0.5秒後にスタート=====
  while( digitalRead(12) == 1 ){};
  delay(500);
  time_old = millis();
}

/*****
繰り返しメインループ
*****/
void loop(){
  //=====自己位置計算=====
  time = millis();
  dt = time - time_old; //経過時間[msec]
  time_old = time;
  theta = float(enCounter_r - enCounter_l)*a/d; //旋回角度[rad]
  if(dt != 0){
    v2 = (theta-theta_old)/dt*1000.0; //旋回角速度[rad/s]
    sum = enCounter_r + enCounter_l;
    temp = float(sum-sum0);
    v1 = temp/2.0/float(dt)*1000.0*a; //並進速度[m/s]
    x = x + (temp*cos(theta)*a/2.0); //車軸中心x座標[m]
    y = y + (temp*sin(theta)*a/2.0); //車軸中心y座標[m]
  }
}

```

```

    sum0 = sum;
};
theta_old = theta;
//=====制御則=====
v1i = v1i + (vd - v1)*float(dt)/1000.0;
u1 =2.0*(vd - v1)+ 8.0*v1i;           //並進方向速度入力(PI control)
e = (yr - y);
ei = ei + e*float(dt)/1000.0;
ed = (e - e0)/float(dt)*1000.0;
u2 = kp*e + ki*ei + kd*ed;           //旋回方向速度入力(PID control)
e0 = e;
vr = u1 + u2*d/2.0;
v1 = u1 - u2*d/2.0;

if(x > xd){
    v1 = 0;
    vr = 0;
}

//=====モータへの出力=====
motorDrive(M1_1, M1_2, M1_PWM, v1);
motorDrive(M2_1, M2_2, M2_PWM, vr);
//=====シリアル通信=====
Serial.print(x,DEC);                 //DEC は送信に1 つにつき 3ms くらいかかる
Serial.print(",");
Serial.print(y,DEC);
Serial.print(",");
Serial.print(dt);
Serial.print(",");
Serial.print(v1*1000);
Serial.print(",");
Serial.println(v2*1000);
}

/*****
モータ入力の関数
*****/
void motorDrive(int m1Pin, int m2Pin, int PWMpin, float v){
    int maxDuty = 150;                 //大きすぎるとエンコーダの読み飛ばしが起きる
    int km = 600.0;                   //モータデューティ比と速度の比
    int duty;
    if (v > 0){                         //カウントが到達目標より小さかったら正転
        digitalWrite(m1Pin, HIGH);
        digitalWrite(m2Pin, LOW);
        duty=(int) (v*km);             //速度から duty 比への変換
    }else if(v < 0){
        digitalWrite(m1Pin, LOW);
        digitalWrite(m2Pin, HIGH);
        duty=(int) (-v*km);           //速度から duty 比への変換
    }else{
        digitalWrite(m1Pin, LOW);
        digitalWrite(m2Pin, LOW);
        duty = 0;
    }
    if(duty > maxDuty) duty=maxDuty;   //リミッタ
    analogWrite(PWMpin, duty);         // モータの PWM 設定
}

```

```

/*****
 エンコーダのカウンタ割り込み関数
 *****/
void count_l(){
  if (vl >= 0) enCounter_l++; //電圧入力为正ならば正転していると仮定
  if (vl < 0) enCounter_l--;
}
void count_r(){
  if (vr >= 0) enCounter_r++;
  if (vr < 0) enCounter_r--;
}

```

実際に動かしてみてロボットの走行軌道が図6のようにならない場合、原因は大きく分けて2つ考えられるでしょう。まず、計測したデータと実際のロボットの動きがあていない場合ですが、これはエンコーダの計測値が正しくないことが考えられます。つまりタイヤが地面と滑っている、エンコーダの読み飛ばしが起きている、車軸が曲がって固定されているなどの原因が考えられます。遅い速度で直線走行させてエンコーダ値から計測した位置とメジャーなどで測定した実測値が合うかを確認するなど、原因を探ってください。車輪が回転するときにタイヤが波打つような場合にはタイヤの取り付けを確認して下さい。また、第1-2節で説明したようにプログラムの最初で定義している**タイヤ間距離 d** 、**タイヤの直径 r** 、および1回転当たりのパルス数 C (上記のプログラム例では $48 \times 16 = 768$ です) をロボットに合わせる必要があります。つぎに、計測したデータとロボットの動きは合っているけれど、その動きがおかしい場合ですが、これは制御プログラムの問題が考えられます。まずはプログラムの入力ミスがないかを確認してください。つぎに使用しているロボットの特性に対して、制御パラメータが不適切であることが考えられますので、制御ゲイン kp 、 ki 、 kd を調整してください。図6の結果よりも良い軌道になるようなゲインの組み合わせを探ってみてください。

参考文献

- 1) 古典制御の本 (たとえば、吉川著、古典制御理論、コロナ社(2014))
- 2) 現代制御の本 (たとえば、小郷、美多著、システム制御理論入門、実教理工学全書(1980))
- 3) 山川、時間軸状態制御形にもとづいた車輪型倒立振り子ロボットの軌道追従制御、計測自動制御学会論文集 49-10, pp.936-943(2013)

謝辞

本学での実習科目や高校生対象のロボットコンテスト開催などにおいて、2005年から継続して Robo Designer を利用しています。とても扱いやすい教材であり、大変感謝しております。また、今回の実験等は JSPS 科研費 JP17K01093 の助成を受けて行いました。