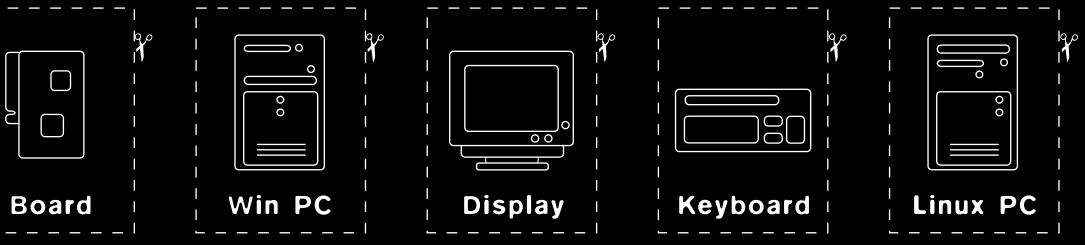
Special Linuxで電子工作シリーズ

LinuxでLEGO（前編）



有限会社ハンブルソフト
成松宏

はじめに

なんとか続いているLinuxで電子工作シリーズですが、なかなかLinuxと電子工作が両立しません。今回もLinuxだけで工作はナシです。代わりに「LEGO MINDSTORMS」を使います。

LEGO MINDSTORMS

LEGO MINDSTORMS**は各種雑誌で取り上げられることが多いので、ご存知の方も多いと思います。**LEGO**は、**LEGO**グループの登録商標であると同時に、**LEGO**ブロックのことを指しています(記事末RESOURCE[1]を参照)。**LEGO**の製品群はいくつかのシリーズに分類されており、幼児用の**DUPLO**、モータや**ギアを使う**TECHNIC**などがあります。**MINDSTORMS**は1999年に新たに加わったシリーズで、マイコンを内蔵したコントローラを使用することを特徴とします。**

MINDSTROMS**は、正確にはシリーズ名にあたり、さまざまな商品が含まれています。その中でも最も**MINDSTORMS**らしいのが、「**ROBOTICS INVENTION SYSTEM**」(以下**RIS、写真1**)で、**RCX**というマイコンを内蔵したコントローラを使用しています(写真2)。さらに、RISの拡張用の商品として、「ROBO SPORTS」から最新の「VISION COMMAND」まで多くの製品があります。また、パソコンを介せずにプログラム可能な、「MICRO SCOUT」というコントローラを使う「DROID DEVELOPER KIT」(DDK)や、「DARK SIDE DEVELOPER KIT」(DSDK)などもあります。**

日本でも、DDKは、割りとアチコチのオモチャ屋さんで見

写真1 ROBOTICS Invention System

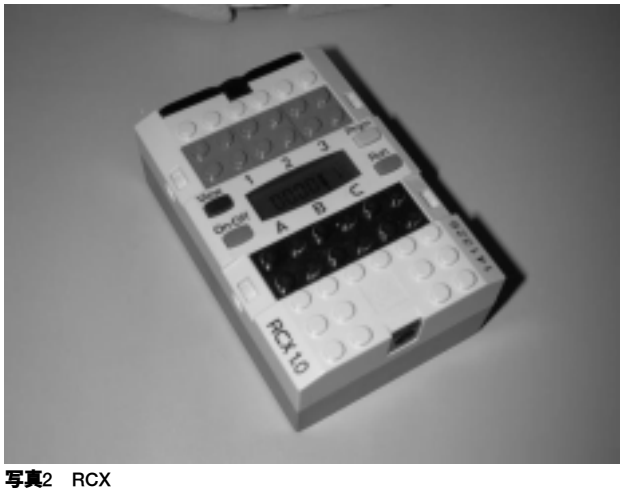


写真2 RCX

ることができます。RISは扱っている店が随分増えてきましたが、それでもまだまだ限られています。LEGOユーザー定番のホームページである、Jin Sato氏の「MINDSTORMS情報局」([2])には、RISを扱っているショップの広告が多く掲載されています。近所に扱っているお店のない方は、「Wise Web Trading」([3])から、オンラインショッピングで購入することも可能です。これらのサイトには商品の写真がたくさん並んでいるので、眺めているだけでも楽しいものです。

RISの普通(!?)の使い方

RISを購入すると詳しい説明書が付いてきます。最初は、この説明書に掲載されている作例通りにブロックを組み立て、RIS添付のソフトでプログラムを作成します。LEGOブロックで思い通りの機構を作るのは意外と難しいので、最初は作例で練習するのが良いでしょう。作例を実際に作ってみると、よくできているので感心します。写真3の作例は2個のモータと2個のタッチセンサーを搭載し、障害物回避やテーブルから落ちない(テーブルの端を検出できる)機能を持ったロボットです。

追加すると楽しい作品

RISにはたくさんの部品がついてきますが、LEGOにはその他にも追加すると楽しい部品がたくさん販売されており、サービスパーツとして購入できるものもあります。お勧めのパーツとしては チェーン、ターンテーブル、回転センサ、モーター、リモコンなどがあります(写真4)。リモコンがあると、プログラムしなくてもRCXに接続されたモーターを動かせるので、とても便利です。回転センサとリモコンは、

写真3　作例のTop Secret Plan

写真4　チェーン、ターンテーブル、リモコン

「ULTIMATE ACCESSORY SET」(国内価格で8800円くらい)にも含まれていますので、こちらで入手するのも良いかもしれません。他のパーツはWise Web Tradingなどからも購入できるはずです。

RCX内部構造

RCXは、H8/3292を内蔵したコンピュータで、16KbytesのROMと32KbytesのRAMを備えています。RCXは3つのモータードライバ、3つのセンサインターフェイスと、1つの赤外線ポートを持っています。ROMにはバイトコードのインタプリタが含まれ、赤外線ポートを経由した通信により、バイトコードで命令を送ったり、バイトコードで書かれたプログラムをダウンロードできます(図1)。

通信プロトコル

RCXとIRタワーの通信プロトコルについて説明します。IRタワーからRCXへ送るデータは、ヘッダ、コマンド、コマンドの引数とチェックサムからなります。ヘッダは、16進数で「`0xff 0x55 0x00`」という3bytesのデータです。コマンドは1byteです。コマンドの引数の数は、コマンドによって異なり、全くない場合もあります。チェックサムは、コマンドと引数の合計値の下位8bitを送ります。コマンド、引数、チェックサムは1byteごとに、ビットを反転した1byteを続けて送ります。

例を挙げて説明しましょう。モータのオン／オフを制御するコマンドは16進数で`0x21`で、1byteの引数を取ります。モーターAをオンにする場合、引数は`0x81`になります。チェックサムは`0x21`＋`0x81`＝`0xa2`ですから、`0xa2`です。IRタワーか

らRCXの送るコマンドは、ヘッダと反転バイトを加えて図2のようになります。

RCXにデータを送ると応答が帰ってきます。まずIRタワーが送信したデータをそのままエコーバックしてきます。その次にRCXからの返事が続きます。RCXからの応答も送信データと同じ構造を持っており、ヘッダ、データ、チェックサムからなります。ヘッダは「0x55 0xff 0x00」、または「0xaa 0xff 0x00」です。データは、送信したコマンドごとに異なります。データとチェックサムには、1byteごとにビット反転したバイトが入ります。

先ほどのモータのオン／オフを制御するデータを送った場合、返事として0xd6または0xdeというデータが戻ってきます。

また、赤外線による通信では通信エラーが頻繁に発生します。正しい応答が戻ってこなかった場合には通信エラーが発生したとみなして、再度同じデータを送信します。ただし、RCXは続けて同じコマンドが送られてくると、それを再送によるものとみなして無視します。そこで、同じコマンドを続けて送る必要がある場合には、0x08ビットを反転させて送ります。反転させてもコマンドとしての意味は変わりません。先ほどの例の場合ですと、0x21コマンドと0x29コマンドは両方ともモーターのオン／オフを切り替えるコマンドになります。

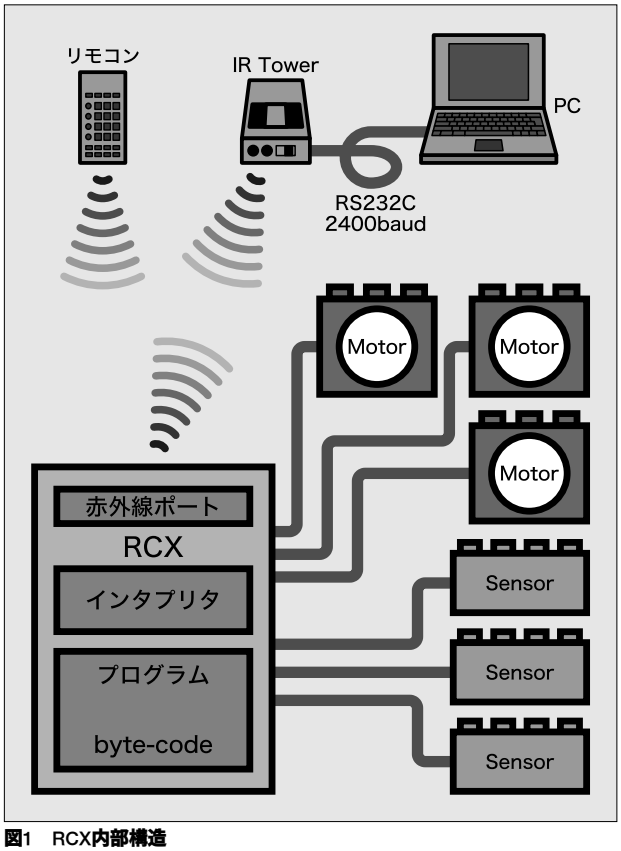


図1　RCX内部構造

RCXのコマンド

RCXのコマンドについては、Kekoa Proudfootさんの「RCX Internals」（[4]）というWebサイトに、詳しい解析結果が掲載されています。記事末の表1に、RCXに送ってみると面白そうなコマンドを抜き出して説明しておきますので、みなさんもぜひ挑戦してみてください。RCXのコマンドすべてについてお知りになりたい方はRCX Internalsをご覧ください。

引数のところで、byteは1byteのデータ、shortは符号付き2bytesのデータ、ushortは符号なし2bytesのデータを示します。なお、2bytesデータでは下位バイトを先に送ります。

シリアルポートを使う

ここまで分かると、コマンドを送ってやればモーターが動きそうです。IRタワーをCOMポートにつなぎ、データを送ってやればいいわけです。

●デバイスの名前

PCのCOMポートは、Linuxでは/dev/ttyS*というデバイスになりますので、COMポートがどのデバイスに対応するのか調べてましょう。私が使用しているカーネル2.2.13のTurboLinux Server 6.0では、カーネルにどのデバイスが認識されているかは、/proc/tty/driver/serialで確認できました。

筆者のシステムでは、

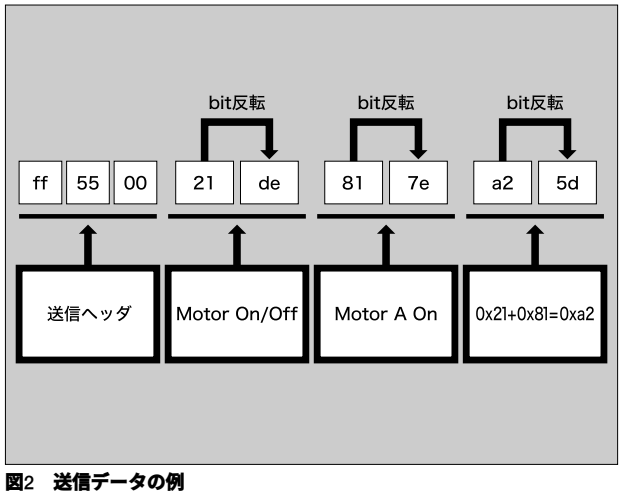


図2　送信データの例

```
$ head -5 /proc/tty/driver/serial
serinfo:1.0 driver:4.27
0: uart:16550A port:3F8 irq:4 baud:9600 tx:0 rx:0
1: uart:unknown port:2F8 irq:3
2: uart:16550A port:3E8 irq:4 baud:9600 tx:0 rx:0
3: uart:16550A port:2E8 irq:10 baud:2400 tx:0 rx:0
```

というように表示され、ttyS00、ttyS02、ttyS03が認識されていることが分かります。これでデバイスに対応するI/Oポートが分かりますので、マザーボードのBIOSの設定などと照らし合わせると、IRタワーを接続したCOMポートのデバイス名が分かります。

●setserial

筆者のシステムにはモデムカードなどが入っており、使用しているIRQが標準的なものでなかったため、setserialコマンドで設定が必要でした。例えば、ttyS03がIRQ 10を使用ようにするには、rootになって以下のように入力します。

```
# setserial /dev/ttyS3 irq 10
```

/etc/rc.d/init.d/serialを読みますと、setserialで行った設定は/var/run/serial.confに保存され、次回起動時に再設定されるはずが、TurboLinux Server 6.0では設定が保存されていませんでした。この原因は、システム停止時に/etc/rc.d/init.d/serialが実行されていないためでした。

このような場合、以下の修正を加えてやるとsetserialでの設定が保存されます。rootで作業してください。

```
# cd /etc/rc.d
# ln -s init/serial rc3.d/K75serial
# ln -s init/serial rc4.d/K75serial
```

nqcで確認

IRタワーも含めて、パソコンとRCXが通信できるかどうかを確認する場合には、nqc([5])というソフトウェアを使用すると便利です。nqcはDave Baum氏が開発したRCXをプログラムするためのソフトウェアです。詳しくは次回紹介することにして、ここでは、nqcのインストール方法とその動作を確認する方法を説明します。

まず、nqcのWebページからnqc-2.2.r1.tar.gzをダウンロードします。このファイルを展開し、readme.txtを読み、Makefileを確認し、makeすればインストールできます。具体的には、次のようになります。

```
$ tar xovfz nqc-2.2.r1.tar.gz
$ cd nqc-2.2.r1
$ ……必要であればMakefileを修正……
$ make
  ……中略……
$ su                    # rootになる
Password: ********
# make install          # インストールする
# exit
```

インストールできたら、以下のコマンドでIRタワーとの接続を確認できます。

```
$ nqc -S/dev/ttyS3 -d test.nqc
```

ttyS3の部分はIRタワーに接続されているシリアルポートのデバイス名を入力してください。/dev/ttyS0に接続されている場合は省略可能です。また、Makefileの

```
DEFAULT_SERIAL_NAME = "/dev/ttyS0"
```

の行を変更するか、環境変数RCX_PORTに設定することでデフォルトのポート名を変更できます。IRタワーと正しく接続され、RCXと通信できれば、以下のようになります。

```
$ nqc -d test.nqc
Downloading Program:..complete
Battery Level = 7.1 V
```

RCXの電源が入っていなかったり、その他の理由で通信できなかった場合は以下のようなエラーメッセージが出力されます。

```
$ nqc -d test.nqc
Downloading Program:error
No reply from RCX
```

コマンド送信実験

コマンドを送信する実験をしてみましょう。このために作成した、Perlのプログラムをリスト1に示します。このプログラムは引数としてRCXの引数とコマンドを取り、ヘッダ、反転ビット、チェックサムを付加してRCXに送信します。コマンドと引数は16進数で指定します。例えば、モーターAをオンにするのであれば、コマンド21か29、引数に81を指定します。オフならば、引数を41にします。

それでは実際に使ってみましょう。RCXのAにモーターをつなぎ、電源を入れてプログラムを実行します。

```
$ perl -w sendTest.pl 21 81
2400
```

2400というのは7行目のsttyコマンドの出力です。sttyコマンドで通信速度の設定を行っています。モーターが動き出したら成功です。動かなかったら、

```
perl -w sendText.pl 29 81
```

を試してみてください。

モーターを止めるには、以下のようにします。

```
$ perl -w sendTest.pl 29 41
2400
```

前に説明した、再送を見分けるルールによって、21コマンドを続けて送信しても、2回目以降は再送とみなされて無視されてしまいます。そこで今度は、コマンドとして29を送ります。モーターをオンにするのに29を使用した場合は、次は21コマンドを送ります。

次に、このときIRタワーからどのようなデータが送られて来ているか見てみましょう。別のウィンドウを開き、そこで「od -t x1 -d /dev/ttyS3」を実行し、再度モーターをオン／オフさせてみますと以下のようになります。

```
$ od -t x1 -v /dev/ttyS3
0000000 ff 55 00 21 de 81 7e a2 5d 55 ff 00 de 21 de 21
0000020 ff 55 00 29 d6 41 be 6a 95 55 ff 00 d6 29 d6 29
                      ⋮
(^Cで中断)
```

送信したデータのエコーバックと、返答のヘッダ「0x55 0xff 0x00」に続いて、返事のdeとd6が返って来ているのが分かります。

通信ライブラリを作る

実験はできましたが、実際に使用する際にはRCXからの返答を見て通信を確認し再送したり、RCXからセンサーの値を読み出すようなことも必要です。

これは面倒な処理ですので、そのような機能を持つPerlの関数trxを定義することにします。この関数はPerlスクリプトの中で以下のように使用します。

```
$res = &rtx(1,0x21,0x81);
```

第1引数は、RCXからの返答のbyte数です。第2引数は、RCXの送るコマンドで、第3引数以下はコマンドの引数で、byte単位で書きます。

関数rtxは、これらの引数からヘッダ、反転バイト、チェックサムを追加したコマンド列を作成し、RCXに送信した後、RCXからの返答が期待されるバイト数に達するのを待ちます。達する前にタイムアウト(0.1秒)に達した場合、同じコマンドを再送します。10回再送しても返答がない場合、エラーメッセージを出力し終了します。

返答があれば、返答からヘッダ、反転バイト、チェックサムを除いたデータを文字列として返します。この文字列から必要なデータを取り出すのには、unpack関数を用います。この関数rtxの定義を含むファイルをリスト2に示します。

センサの値を読む

リスト2を使用して、センサを読み出すプログラムを使用してみましょう。最初に、センサを読み出す前に、センサのタイプとモードを設定してやります。まず、タッチセンサを読み出してみます。プログラムをリスト3に示します。

タッチセンサをsenser-1につないだ後でリスト3を実行させ、ポッチを何度か押すと実行例1のようになります。これで、ちゃんとセンサの状態を読み出せているのが分かります。

リスト1　sendTest.pl

```
#!/usr/bin/perl

 1 ($cmd,$arg) = @ARGV;
 2 defined($arg) ||
 3     die "Usage:$0 cmd arg\n";
 4 $tty = "/dev/ttyS3";
 5 system("/bin/stty -F $tty speed 2400 raw");
 6 open(TTY,">$tty") ||
 7     die "open $tty failed.\n";
 8 $c0 = hex($cmd);
 9 $c1 = 255 - $c0;
10 $a0 = hex($arg);
11 $a1 = 255 - $a0;
12 $p0 = ($c0 + $a0)%256;
13 $p1 = 255 - $p0;
14 print TTY pack("C*",0xff,0x55,0x00,
15     $c0,$c1,$a0,$a1,$p0,$p1);
16 close(TTY);
```

リスト2 legoLib.pl

```
 1  #!/usr/bin/perl
 2  $tty = "/dev/ttyS3";
 3  system("/bin/stty -F $tty speed 2400 raw");
 4  open(TTY,"+<$tty") ||
 5      die "open $tty failed.\n";
 6  @header = (0xff,0x55,0x00);
 7  $flip = 0;
 8
 9  sub trx {
10     local($nRes,@txdata) = @_;
11     local($strTx,$nTx,$rd,$rs,
12           $rin,$ee,@data,@res,$i,$retry,$rbuf);
13
14     $retry = 0;
15     $strTx = &txStr(@txdata);
16     $nTx = length($strTx);       # 送信バイト数
17     $nRx = $nRes * 2 + 5;
18
19   retry:
20     while(1){
21         syswrite(TTY,$strTx,$nTx);
22         $rd = "";
23         $timeOut = 0;
24
25         while(length($rd) < $nTx+$nRx){
26             $rin = "";
27             vec($rin,fileno(TTY),1) = 1;
28             $nfound=select($rin,undef,
29                            undef,0.1);
30             if($nfound==0){ # データがこない
31                 $timeOut = 1;
32                 $retry++ < 10 ||
33                     die "No reply from RCX\n";
34                 next retry;
35             }
36             $rbuf="";
37             $c = sysread(TTY, $rbuf, 40);
38             defined($c) ||
39                 die "sysread error.\n";
40             $rd .= substr($rbuf,0,$c);
41         }
42         $ee = substr($rd, 0, $nTx);
43         $rs = substr($rd, $nTx, $nRx);
44         last;
45     }
46     @data = unpack("C*",$rs);
47     @res = ();
48     for($i=3;$i<$nRx-2;$i+=2){
49         push(@res,$data[$i]);
50     }
51     pack("C*",@res);
52 }
53
54 sub txStr {
55    local (@cmd) = @_;
56    local (@data,$c,$checksum);
57
58    push(@data,@header);
59    $c = shift(@cmd);
60    $c |= 8 if $flip;
61    $flip = !$flip;
62    $checksum = $c;
63    push(@data,$c, ~$c);
64
65    foreach $d (@cmd) {
66        $checksum += $d;
67        push(@data,$d, ~$d);
68    }
89    push(@data,$checksum,~$checksum);
70    pack("C*",@data);
71 }
72 1;
```

リスト3 sensor.pl

```
 1 #!/usr/bin/perl
 2
 3 do 'legoLib.pl';
 4 &trx(1,0x32,0,1); #  set sensor type(1,Touch)
 5 for($i=0;$i<10;$i++){
 6     $s = &trx(3,0x12,9,0);
 7     ($r,$v)=unpack("Cs",$s);
 8     printf "0x%02x val=%5d\n",$r,$v;
 9     sleep 1;
10 }
```

実行例1

```
$ perl -w sensor1.pl
0xe5 val=    0
0xed val=    1
0xe5 val=    1
0xed val=    0
0xe5 val=    0
0xed val=    1
0xe5 val=    1
0xed val=    1
0xe5 val=    0
0xed val=    0
```

指定した量だけモーターを動かす

センサの値を見ながら、指定した量だけモーターを動かすプログラムを作ってみましょう。まず、モーターの移動量をセンサで計測できるような機構を作らなければなりません。筆者は、写真5のようなターンテーブルを作成してみました。なお、このターンテーブルには、RISに含まれていない部品であるターンテーブル、回転センサ、チェーンを使用しています。いずれも便利な部品なので別途購入することをお勧めします。

プログラムをリスト4に示します。このプログラムは引数を

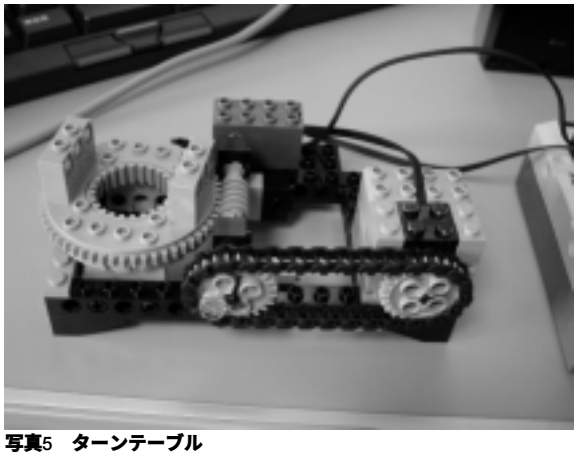


写真5　ターンテーブル

1つ取り、引数が正であればモーターを正転、負であれば逆転させ、回転センサの値が引数の値を超えた時点でモーターを止めます。

LEGOのモーターは、接続ブロックの取り付けの方向によって回転方向が逆になりますので、その場合にはプログラムか接続ブロックの取り付け方向を修正してください。

回転センサは1/16回転単位です。ターンテーブルの外周の歯車は56歯でこれをウォームギアで駆動していますので、56×16＝896で1回転ということになります。実行させると次のようになります。

```
$ perl -w servo1.pl 896
0.14,    0
0.36,    7
0.50,   20
0.64,   34
0.78,   47
0.92,   61
1.06,   74
1.20,   88
  ... 中略 ...
9.32,  875
9.46,  888
9.60,  901
9.82,  916
9.96,  916
10.10,  916
10.24,  916
10.38,  916
```

表示されるのは、左側の値がスタートからの時間で、右側がセンサーからの読み出し値です。時間は、/proc/uptimeから読み出しました。時間を出力させたのはグラフを書くためです。では、出力をファイルにリダイレクトし、リスト5のような設定でgnuplotにグラフを描かせてみます。

リスト4　servo1.pl

```
 1 #!/usr/bin/perl
 2
 3 do 'legoLib.pl';
 4 $/=" ";
 5 ($a)=@ARGV;
 6 defined($a) || die "Usage:$0 arg\n";
 7
 8 &trx(1,0x32,0,4);             # センサー1を回転に
 9
10 if($a > 0){
11    &trx(1,0xe1,0x81);         # モータAを正転
12 }
13 else {
14     &trx(1,0xe1,0x01);        # モータAを逆転
15 }
16 &trx(1,0x13,0x01,2,7);        # モータA Power設定
17
18 @t = `cat /proc/uptime`;
19 $t0 = $t[0];
20 &getData;
21
22 &trx(1,0x21,0x81);            # モータAをON
23 while(1){
24    $v = &getData;
25    if($a > 0){
26 last        if($v > $a);
27     }
28     else {
29 last if($v < $a);
30     }
31 }
32 &trx(1,0x21,0x41);            # モータAをOFF
33 for($i=0;$i<5;$i++){ &getData;}
34 exit 0;
35
36 sub getData {
37     local($s,$r,$v,@t);
38     $s = &trx(3,0x12,9,0);
39     @t = `cat /proc/uptime`;
40     ($r,$v)=unpack("Cs",$s);
41     printf "%.2f,%5d\n",$t[0]-$t0,$v;
42     return $v;
43 }
```

```
$ perl -w servo1.pl 200 > servo1.dat
$ gnuplot servo1.gp
……改行で終了
```

これで図3のグラフがX上に表示されます。この例では200(200÷896×360＝80.35度)動かそうとしていますが、実際にはプログラムの遅れと惰性で、224(90度)まで動いてしまって

います。目的値に達したらモーターを止めるだけの単純な制御ですのでいたしかたないでしょう。

グラフからは、1秒間に7回データを取得できていることが分かります。フィードバック制御を行うと、センサーの値の読み出しに加えてモーターの方向制御も加える必要がありますので、1秒間に3回ほどしか制御を行えないと考えられます。フィードバック制御を行えば良いのですが、2400baudという遅い通信でつながっている状況で良好な制御を行うのは難しいでしょう。

おわりに

なんとかPerlを使ってRCXを動かすことができました。pack／unpackやselectなど、日ごろあまり使わない(かもしれない)関数を多用していますが、とても便利な関数ですので、ご存知なかった方はこれを機会に覚えると、Perlでできることの幅が広がって良いと思います。使い方は

```
$ perldoc -f select
$ perldoc -f pack
```

などで見ることができます。

また、Linux側からRCXをすべて制御しようとすると、2400baudという遅い通信速度と、エラー対策された冗長なプロトコルのせいで、比較的ゆっくりしたことしかできません。しかし、それでもLinuxマシンにいろんなことをやらせることができると思います。例えば、メールが来ると旗が上がるとか、マシンの負荷(ロードアベレージ、/proc/loadavgで読める)に応じて腕を上下させるとか……。

次回は、nqcを使用してRCXにプログラムをダウンロードし、もっと高速に動作させる方法などを紹介したいと思いますので、ご期待ください。

最後になりましたが、今回の記事では「LEGO MINDSTORMSパーフェクトガイド」([6])を多いに参考にさせていただきました。非常に良い本ですのでみなさんに御一読をお勧めします。

リスト5 servo1.gp

```
set grid
set data style linespoints
set xlabel "time [sec]"
set ylabel "sensor value"
set nokey
plot "servo1.dat"
pause -1
```

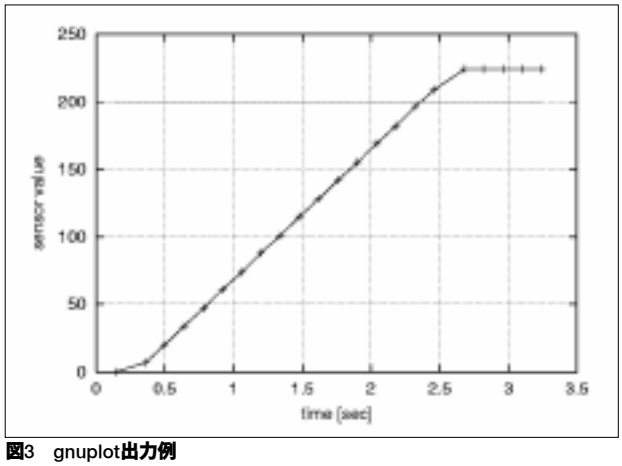


図3 gnuplot出力例

RESOURCE

[1] **LEGOのWebページ**
http://www.lego.com/

[2] **MINDSTORMS情報局**
http://www.mi-ra-i.com/JinSato/MindStorms/

[3] **Wise Web Trading**
LEGOの部品を扱う通販サイト
http://www.wisewebtrading.com/ms/

[4] **RCX Internals**
http://graphics.stanford.edu/~kekoa/rcx/

[5] **nqc Webページ**
http://www.enteract.com/~dbaum/nqc/

[6] **「LEGO MINDSTORMSパーフェクトガイド」**
翔泳社 古川 剛編、Jin Sato、白川 祐記、牧瀬 哲郎、倉林 大輔、衛藤 仁郎著、1999年
ISBN4-88135-769-7

[7] **Linux Serial HOWTO**
http://www.linux.or.jp/JF/JFdocs/Serial-HOWTO.html

表1　RCXコマンド一覧(抜粋)

Alive：RCXの有無を確認	
Command	0x10/10
Reply	0xef/e7

RCXが動作しているかどうかを調べるコマンド。RCXが動作していればreplyを返す。動作していなければ、何も返さない。

Get Battery Power：電源電圧の取得	
Command	0x30/38
Reply	0xc7/cf,ushort millivolts

millivoltsで、ミリボルト単位でRCXの電源の電圧を返す。

Set Motor On/Off：モーターのOn／Offの切替え	
Command	0x21/29,byte code
Reply	0xd6/de

codeの値に応じてモーターのOn／Offを切替える。codeの各bitの意味は以下の通り。

値	説明
0x01	motor Aの状態を変更する
0x02	motor Bの状態を変更する
0x04	motor Cの状態を変更する
0x40	指定されたmotorをOffにする
0x80	指定されたmotorをOnにする

0x40と0x80の両方のbitが0のとき、motorはfloat状態になり、motorは空転する(Offのときはブレーキがかかる)。0x40と0x80の両方が1のとき、指定されたmotorはOnになる。

Set Moter Direction：モータの方向の設定	
Command	0xe1/e9,byte code
Reply	0x16/1e

codeの値に応じてモーターの方向を切替える。codeの各bitの意味は以下の通り。

値	説明
0x01	motor Aの方向を変更する
0x02	motor Bの方向を変更する
0x04	motor Cの方向を変更する
0x40	指定されたmotorの方向を反転する
0x80	指定されたmotorの方向を順方向にする

0x40と0x80の両方のbitが0のとき、motorは逆方向にセットされる。
0x40と0x80の両方が1のとき、指定されたmotorの方向は反転する。

PlaySound：音を鳴らす	
Command	0x51/59,byte sound
Reply	0xa6/ae

引数soundで指定された音を出す。soundは0〜5の値で、対応する音の種類は以下の通り。

値	音の種類
0	Blip
1	Beep beep
2	Downword tones
3	Upward tones
4	Low buzz
5	Fast upward tones

Set Sensor Type：センサの種類の設定	
Command	0x32/3a,byte sensor,byte type
Reply	0xc5/cd

sensorで指定されたセンサの種類を設定する。sensorには0、1、2のいずれかの値を渡す。typeの値によってセットされるセンサのタイプとデフォルトのモードを以下に示す。

値	センサのタイプ	デフォルトのモード
0	Raw	Raw
1	タッチセンサ	論理値
2	温度センサ	温度(摂氏)
3	ライトセンサ	パーセント
4	回転センサ	角度

Set Sensor Mode：センサのモードを設定	
Command	0x42,byte sensor,byte mode
Reply	0xbd/b5

sensor(値は0、1、2)で指定されたセンサのモードをセットする。モードを変更することによって、読み出される値を変換できる。modeの各値の意味を以下に示す。

値	モード名	説明
0	Raw	0〜1023の値
1	論理値	0または1
2	Edge count	論理値が変化した回数
3	Pulse count	論理値が変化した回数÷2
4	パーセント	0〜100の値
5	温度(摂氏)	1／10度単位(-19.8〜69.5度)
6	温度(華氏)	1／10度単位(-3.6〜157.1度)
7	角度	1／16回転、符号付き16bit

Get Value：値の取得	
Command	0x12/0x1a,byte src,byte arg
Reply	0xe5/ed,short value

srcとargで指定される値を取得する。変数やセンサの値を取得できる。

src	arg	種類
0	0〜31	変数
1	0〜3	タイマー
2	0〜255	argumentの値そのまま
3	0〜2	motorの状態
4	0〜255	0〜argumentの値の範囲の乱数
8		現在のプログラムの番号
9	0〜2	センサの値
10	0〜2	センサタイプ
11	0〜2	センサモード
12	0〜2	センサのRAW値
13	0〜2	センサの論理値
14	0	時計の分の値
15	0	メッセージ

モーターの状態の値 0x00：float、0x07：power、0x08：fwd、0x40：0ff 0x80：On

Set Variable：変数へ値をセット	
Command	0x14,byte index,byte src,short arg
Reply	0xe3/eb

index番目の変数(0〜31)に、srcとargで指定される値をセットする。srcとargの値の意味については、Get Valueの項参照。ただしSet Variableではargの値は16bitになる。