コンピュータ制御

2011/09/13

愛知大学

有澤　健治

Lesson 1 (準備)

運営ルール

2人1組 (座席指定、ノートパソコン指定、ロボットセット指定)

注意
- 他のクラスと教材が共用になる
- 毎回課題が出る
- 課題報告の提出で出席扱い (課題をこなして)
- 遅刻の扱い (10分の遅刻は欠席扱い)
- 休みの扱い (4回休むとアウト)

休んだ場合
- 次の週までに補習
- 補習を怠ったら 10 点減点
- 東教室棟センター事務室で補習
- 課題報告を出す (課題をこなして、 次回講義までに、僕に直接渡す事)

毎回の終了時の確認
- ロボットの電源 (ON になっていないか?)
- ロボットの部品 (床に落ちていないか?)

教材

- LEGO Kit
- Users Guide (LEGO Kit 附属のもの)　　# テキスト
- ノートPC

テキスト(PDF版)のダウンロード
- `http://ar.aichi-u.ac.jp/lecture/lego/`　　(学外から)
- `http://ar/lecture/lego/`　　(学内から)

テキスト(PDF版)
- Users Guide
- NXC Guide
- NXC tutorial

今日の作業

- ラベルとタグ (2枚、ノートPCとロボット)
- テキスト(PDF版)ダウンロード
- 実習報告を提出

2011 A3
座席: 05, 06
氏名: 愛知太郎
氏名: 三好次郎

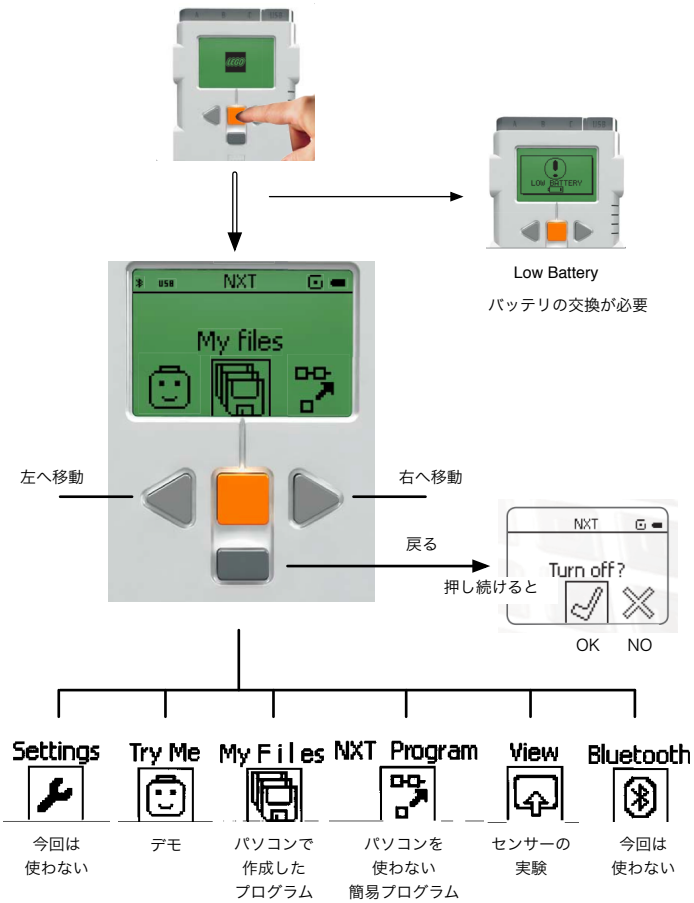
ラベルの例 (タグも同様)

Lesson 2 (組み立て)

今日の作業

組み立て目標: テキスト p.8
パソコンは使わない
バッテリーの確認　　# 赤いボタンでメニューが出るか?
メニューの見方
終了と電源 OFF

メニューチャート



用語

NXT Brick

Lesson 3 (Try Me)

今日の作業

パソコンは使わない
Lego Kit のみ

動作確認

Try Me
1. Try Touch
2. Try Light
3. Try Ultrasonic
4. Try Sound
5. Try Motor
注: テキストとは図が異なる

注意事項

課題の内容に応じて適切なセンサーを取り付け、正しく配線を行う事。
バッテリがへたっている場合には Try Motor でのモーターの動きが悪くなる。(⇒ 電池交換)

課題

以下の Try Me を実行し、観察し気付いた事を書きなさい。

1. Try Touch
タッチした時に起こる事

2. Try Light
白い紙と黒い紙のセンサーの反応の違い

3. Try Ultrasonic
センサーの前に手を置く、
手との距離を変えてみる
音の変化を確認する

4. Try Sound
サウンドセンサーに音を拾わせる(手を叩く、口笛を吹く)

5. Try Motor
車を手で動かして音の変化を確認する
モーターの動きを捉えていることが分かる

Lesson 4 (View)

View にはセンサーの働きが数値で分かる実験が揃っている。後に、センサーの反応に応じてモーター等を動かすプログラムを作ることになるが、今回の実験はその時に役に立つはずである。
View のメニューに関する詳しいチャートが PDF 版の “Users Guide” p.22 に載っている*。
注 *: ダウンロードした時のファイル名: “9797_LME_UserGuide_US_low.pdf”

Sound

音の強度を測る
Sound (dB)　　# 物理的な音の強さ
Sound (dBA)　　# 人が感じる音の強さ、今回は使わない

Light

光の強さを測る (色は識別できない)
Reflected light (反射光)　# センサーに付いている LED が光る
Ambient light (環境からの光)
Light Sensor

Temperature

温度を測る
省略(温度センサーが無い)
Temperature (°C)　# Celsius
Temperature (°F)　# Fahrenheit's、今回は使わない (今でも英米圏で使用されている単位)

Rotation

モーターの回転量を測る
Rotation
Motor Rotation (R)　# 1回転 = 1
Motor Rotation (D)　# 1回転 = 360 (degree)

Touch

接触した事を感知する
Touch

Ultrasonic

超音波を使って距離を測る
Ultrasonic (inch)　# inch 単位、今回は使わない。 1 inch = 2.54 cm
Ultrasonic (cm)　　# cm 単位

Lesson 5 (NXT Program)

NXT Program では極く簡単なプログラムを初等的な方法で作成でき、そして、それを実行できる。プログラムの作成にはパソコンは必要ではない。

アイコンを並べでプログラムができあがる。

今回の課題

ページ番号は Users Guide

課題1. (p.23)
課題2. (p.27)
課題3. (p.31)
課題4. (p.35)
課題5. (p.39) # light が必要。携帯電話の画面の光で代用してみる。
課題6. (p.45)

NXT program の作り方: Users Guide (PDF版) p.15
NXT Program の読み方: Users Guide (PDF版) p.21

注 *: ダウンロードした時のファイル名: “9797_LME_UserGuide_US_low.pdf”

次の図(アイコン)で示される NXT プログラムは

1. 前進して右ターン
2. 音を感知すると
3. 後進して左ターン
4. 音を感知すると
5. 繰り返す(1から4を)

この例に習って課題1から課題6の NXT プログラムの意味を日本語で書きなさい。
(動作を確認しながら意味を理解したらよい。意味が分かるように書けばよい)

空白アイコン(Empty) の説明が無いのだが.... 推理してみる。
数字が書かれているアイコンがあるが、この数字に関する説明がない。数字の意味は何だろう? 推理してみる。

Lesson 6 (Linux)

今後はロボットのプログラムをパソコン(ノートPC)で行う。ロボット上で直接プログラムするのに比べて、(a) 複雑なプログラムが可能である事、(b) プログラムを残す事ができる事、などの利点がある。ロボットのプログラムは Windows、Mac、Linux の上で可能だが、ここでは Linux を使う。Windows, Mac には子どもでも楽しめるように、初等的な方法でプログラムできるソフトが存在する(Users Guide PDF 版 p.46 ~ p.57)が我々はそのようなソフトには頼らない。

Linux について

Unix の一種
Unix とは
　　1970年頃、Bell 研究所
Unix からの派生 OS　　# Mac OSX を除き、無料、ソース公開
・Linux # Ubuntu など
・Free BSD　# Mac OSX は FreeBSD を Apple 流にアレンジ

Linux 標準教科書

近頃無料で質の良い Linux のテキストが現れた。PDF 版である。
`http://www.lpi.or.jp/linuxtext/text.shtml`

ユーザ名/パスワード

	クラス A	クラス B
ユーザ名	mei	sat
パスワード	totoro	totoro

注意: 打ち込んだパスワードは見えない

コマンド

`startx`　　# Linux の window system の立ち上げ (start X window)
`ls`　　# ディレクトリ(フォルダー)の中の一覧を見る (list)
`cat`　　# ファイルの内容を見る (catenate)
`pwd`　　# 作業ディレクトリを表示する (print working directory)
`mkdir`　　# ディレクトリを作る (make directory)
`rmdir`　　# ディレクトリを削除する (remove directory)
`cd`　　# 作業ディレクトリを変更する (change directory)
`cp`　　# ファイルのコピー (copy)
`mv`　　# ファイルの移動、名称変更 (move)
`rm`　　# ファイルの削除 (remove)
`man`　　# マニュアルを見る (manual)
`halt`　　# 終了する
`gedit`　　# Linux の(初心者向けの)テキストエディタ

課題

試して見なさい。login 後

```
startx      # window system を立ち上げる (このノートPCでは Gnome が立ち上がる)
pwd         # ホームディレクトリ(/home/mei あるいは /home/sat)が表示される
ls          # どんな名前が表示されるか? (問1)     # sample.nxc が含まれている事の確認
ls -l       # どんな情報が追加されたか? (問2)
ls /home    # どんな名前が表示されるか? (問3)     # totoro が含まれている事の確認
mkdir lego # ディレクトリ lego を作る
ls -l       # lego が追加されたのを確認せよ
cd lego     # 作業場所を lego に移動
pwd         # 何が表示されたか? (問4)
gedit a.txt      # gedit を使って a.txt を編集する
                 # Hello! と打ち込み、終了する
ls -l       # 問2との違いは? (問5)
cat a.txt  # Hello! が表示される
cp a.txt b.txt  # a.txt のコピー b.txt を作る
ls -l       # b.txt が作成されているのを確認する
cat b.txt  # 確かに Hello! が表示される
rm b.txt   # b.txt を削除する
ls -l       # 問5との違いは? (問6)
cd ..       # 現在のディレクトリの親に戻る (単に cd ならホームディレクトリに移る)
pwd         # 何が表示されたか? (問7)
halt        # 終わる (本来は管理者だけが実行可能だが、ここでは変更してある)
```

問題

上のコメントの中の問1から問7に答えなさい。
(ls については http://ja.wikipedia.org/wiki/Ls_(UNIX) が参考になる)

コマンドとマウス

コマンドと言うのはマウスを使った作業に比べて面倒で大変なように思えるかも知れない
が、Unix のコマンドはプログラム化できて、マウスを使って作業をしていると何日もかかるような事を短時間の内に自動的にこなしてしまう凄い能力を持っている。
例えば次のケースを考えたらよい:
毎日毎日明け方4時に行わなくてはならない作業をマウスを使ってやると苦痛である。(必ず人手が必要なため。)
コマンドを使えばこのような作業を完全に自動化できる。

Lesson 7 (NXC Program 1)

Sample program

```
task main()
{
  OnFwd(OUT_A, 75);
  OnFwd(OUT_C, 75);
  Wait(4000);
  OnRev(OUT_AC, 75);
  Wait(4000);
  Off(OUT_AC);
}
```

図1. a.nxc

出典: “NXC tutorial”

`OnFwd(OUT_A, 75);`
ラベル A に接続されたモーターを 75% の速さで前向きに回転させる

`Wait(4000);`
4秒間プログラムの実行を止める。その間モーターは回転し続ける。

`OnRev(OUT_AC, 75);`
ラベル A と C に接続されたモーターを同時に 75% の速さで逆向きに回転させる。

`Off(OUT_AC);`
ラベル A と C に接続されたモーターを止める。

プログラム作成からロボットでの実行までの流れ

1. プログラムの編集
2. コンパイル(compile) & ロード(load) (USB ケーブルをロボットと接続して)

```
nbct a.nxc        # a.nxc から a.rxe を生成し、ロボットにロード
```

注意: “overwrite existing file “○○.rxe” (y/n)?”
のメッセージが出るかも知れない。
対応: y を打つ。このメッセージは a.rxe が既にロードされている時に出る。

3. ロードされた事の確認

```
t2n -ls           # a.rxe が見える → プログラム名は a
```

4. ラン(run) # ロボット側で実行 (USB ケーブル外して)
My Files → Software Files → プログラム名 a を選ぶ

注: 編集するファイルの名前が XYZ.nxc であれば、プログラム名 XYZ を選ぶ。

プログラムの編集

Lesson 6 での課題が完了していると、次のようにして(Lesson 6 で作成した)ディレクトリ lego の中で、プログラムを作成して行ける。login の後、

```
startx            # window system の起動
cd lego           # 作業場所を lego に移動する
ls                # lego の中のファイル一覧を表示する
gedit a.nxc       # a.nxc を編集する
```

注意: ファイル拡張子は “nxc” にすること

問題1

モータの回転速度を 30% にし、待ち時間を2秒間にしてみよう。

問題2

図1のプログラムで

```
OnFwd(OUT_A, 75);
OnFwd(OUT_C, 75);
```

を1つの命令に纏めてみよう。

問題3

このプログラムは車を1往復させるが、2往復させるプログラムを作ってみよう。
コメント: 戻り方がバックなのは気に入らないが、この問題の解決は後回しにしましょう。

コメント

(1) NXC の書き方はプログラミング言語Cと似ているが完全に同じではない。(NXC = “Not eXactly C”)
なおプログラミング言語Cは現代のコンピュータの主力言語である。コンピュータのあらゆる種類のプログラムはCの上に成立していると言っても過言ではない。

(2) `nbct` コマンドは有澤がこの講義のために作ったものである。このファイルは

```
/usr/local/bin
```

に置かれている。`nbct` の内容は

```
cat /usr/local/bin/nbct
```

で見える。
処理の本質的な部分を例示すると

```
nbct a.nxc
```

の場合には

```
nbc -O=a.rxe a.nxc
t2n -put a.rxe
rm a.rxe
```

が実行されるように作られている。

Lesson 8 (NXC Program 2)

例題

```
#define MOVE_TIME    500
#define TURN_TIME    500

task main()
{
    int n;
    n = 0;
    while(n < 4){
        OnFwd(OUT_AC, 75);
        Wait(MOVE_TIME);
        OnRev(OUT_C, 75);
        Wait(TURN_TIME);
        n++;
    }
    Off(OUT_AC);
}
```

図2. b.nxc

出典: “NXC tutorial” (修正されている)

`#define MOVE_TIME 500`
　MOVE_TIME を 500 として定義する。

`int n`
　n が整数(integer)を格納する変数である事を宣言する。

`n = 0`
　n を 0 にする。(n に 0 を代入する)

`while(条件){ .... }`
　条件が満たされている間、繰り返す。
　繰り返しの範囲が { と } で示されている。

`n++`
　n を1だけ増やす。

`OnRev(OUT_C, 75);`
　今回の例題では逆回転するのはCに繋いだモータだけであるから図1の a.nxc とは違った動きをする。

問題1

Lesson 7 に習って図2の b.nxc を実行しなさい。
さらに数字を調整して、90度のターンを4回繰り返して、出発点に戻ってくるようにしなさい。
障害物の周りを1周させてみる。
(調整はトライ&エラーしかないと思う)

いろいろな条件

```
==   equal to (等しい)
<    smaller than (小さい)
<=   smaller than or equal to (小さいか等しい)
>    larger than (大きい)
>=   larger than or equal to (大きいか等しい)
!=   not equal to (等しくない)
```

名前規則

変数には名前を付ける。また define は名前を他のデータに置き換える。名前は英字または数字または下線記号('_')から構成され、数字で始まってはならない。予約語(既に意味付けられている語)(task, main, define, int, while など)は名前の規則に合っているが、変数名や define での名前には使えない。例えば n, n2, xyz などは名前に使えるが、2n は使えない。

Lesson 9 (Light Sensor 1)

例題

```
#define THRESHOLD 60
task main()
{
      SetSensorLight(IN_3);
      OnFwd(OUT_AC, 75);
      while(Sensor(IN_3) < THRESHOLD);
      Off(OUT_AC);
}
```

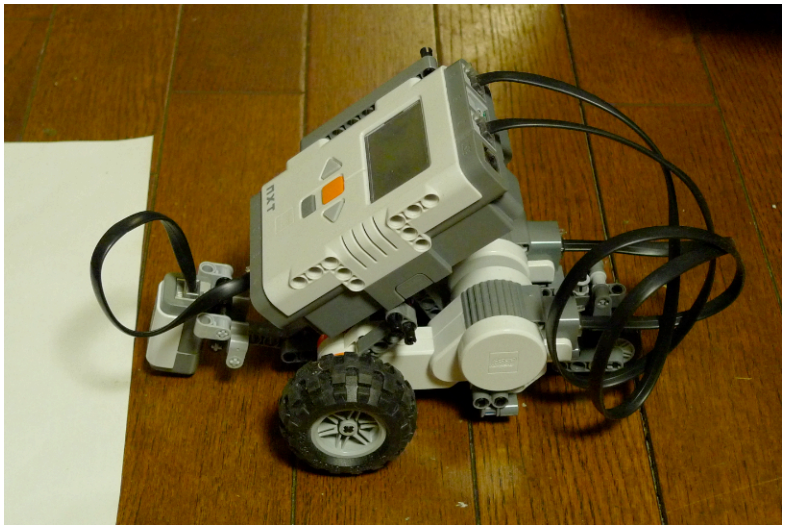
図3. c.nxc

```
# define THRESHOLD 60
    THESHOLD を 60 として定義する

SetSensorLight(IN_3)
   IN_3 にライトセンサーを設定する

while(Sensor(IN_3) < THRESHOLD);
  以下の「説明」を見よ

Off(OUT_AC)
  OUT_AC のモーターを停止する
```

組み立て

光センサーは IN_3
モーターは A と C

光センサーの組み立ては
テキストp.32,33
を見よ。



プログラムの目標

ロボットを白紙に向けて動かす。センサーが白紙を感知したら停止する。

説明

Lesson 8 の例題では

```
while(条件) {  ....  }
```

この場合には条件が満たされている間 { } で囲まれた部分を繰り返す。一般的に言えば { } の中には複数の命令が書かれるが、命令が1個の場合には

```
while(条件) 命令;
```

のように { } を省略できる。命令が存在しない場合には

```
while(条件);
```

である。この場合には条件が破られるまでプログラムはここで停止する。(ロボットが停止する訳ではない。モーターの回転は

```
OnFwd(OUT_AC, 75);
```

でコントロールされており、回転を変化させる命令は実行されてないので、ロボットは前進し続ける。)

今回の例では条件の部分は

```
Sensor(IN_3) < THRESHOLD
```

である。これは「IN_3 のセンサーの値が THRESHOLD (=60) より小さい」の意味である。

注意 ここで使用している言語は本物のC言語ではないので、小数点の付く数は使えない!

問題1

THRESHOLD(しきい値) の値は調整が必要かも知れない。反射光の強さの情報は Lesson 4 (View) の “Reflected light” を使えば分かるので、白い紙と床とで各々の反射光の強さを測定し、その中間を THRESHOLD に採用すれば白い紙と床を区別できるはずである。反射光の強さを実際に測定しなさい。そして、この測定によって最適なしきい値を割り出しなさい。

問題2

ロボットを白い紙の中央部に置いて、ロボットを動かし、センサーが紙の外に出たらロボットを停止させなさい。

問題3

例題のプログラムでは、センサーが白い紙を感知すると直ちに停止する。この時にはロボットの本体は紙の外にいることになる。そこでもう少し難しい問題を考える。直ちに停止しないで、さらに紙の上を走り、センサーが紙の外に出たら停止するようにしなさい。(ヒント: 例題と問題2を組み合わせる。)

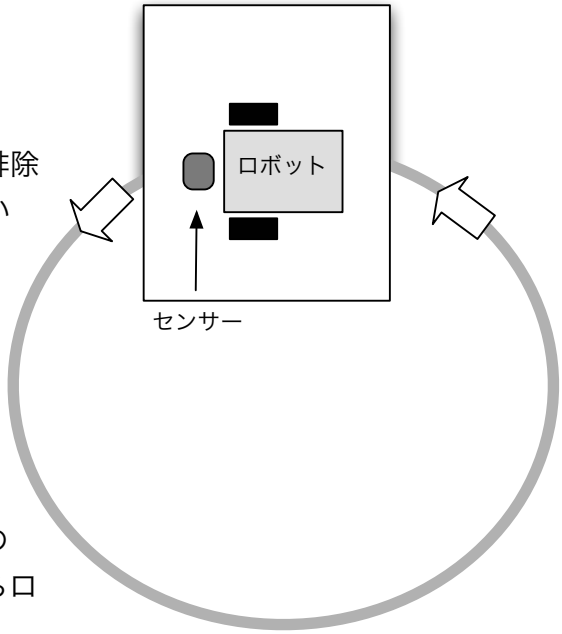
注意:

```
while(Sensor(IN_3) < THRESHOLD);
```

が終了した時に

```
Sensor(IN_3) > THRESHOLD
```

となっている保証はない。つまり等しくなる可能性を排除できない。この現象はセンサーが紙と床の境界を見ている時に発生し得る。



問題4

右図のようにロボットを白い紙の中央部に置いて、ロボットを動かし、センサーが紙の外に出たら(ロボットを停止させなさいで)、ロボットの車の回転数を右と左で変える。するとロボットは円を描いて元の紙の位置に戻って来るはずである。紙の位置に戻って来たらロボットを停止させる。

Lesson 10 (Light Sensor 2)

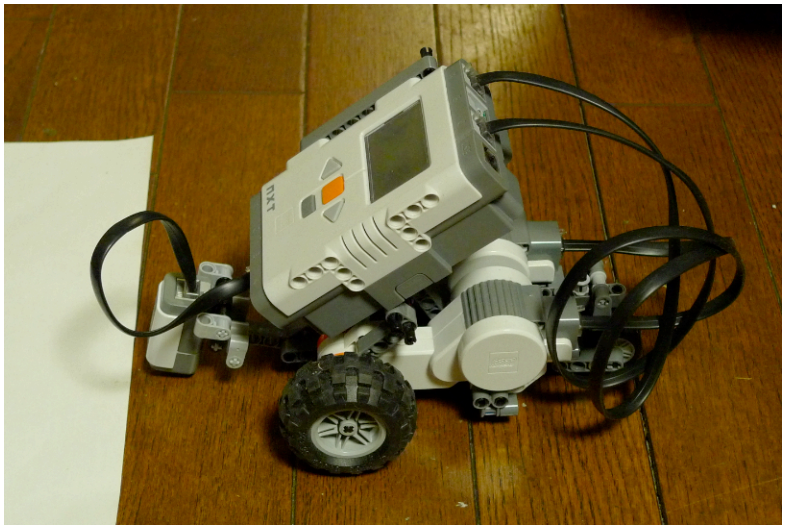
例題

```
#define THRESHOLD 60
task main()
{
    SetSensorLight(IN_3);
    OnFwd(OUT_AC, 75);
    while(true){
        if(Sensor(IN_3) > THRESHOLD){
            OnRev(OUT_AC, 75);
            Wait(1000);
            OnFwd(OUT_AC, 75);
        }
    }
}
```

光センサーは IN_3 へ
モーターは A と C

光センサーの組み立ては
テキストp.32,33
を見よ。

図4. d1.nxc
出典: “NXC tutorial”



動作

光センサーが白い紙を捉えるとロボットは 1秒間 バックし、また前進する。また白い紙を捉えて....
但し THRESHOLD の値を適切に設定しなくてはならないだろう。
(Lesson 9 課題1 を見よ)

解説

true (真)

```
while(true){ .... }
```

繰り返しの条件が常に true (真) なので、無制限に { } の中を繰り返す。

if(条件){ .... }

```
if(Sensor(IN_3) > THRESHOLD){ .... }
```

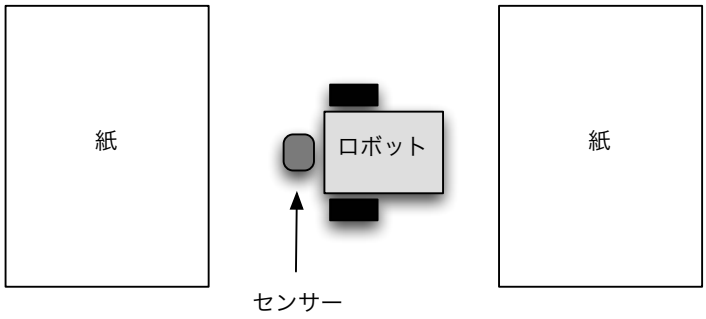
条件が成立する時に { } の中を実行する。ここでは条件として

```
Sensor(IN_3) > THRESHOLD
```

が指定されている。

問題1 2枚の紙を床の上に40cm程離して置き、その間にロボットを置く。ロボットの動作は、2枚の紙の間の往復運動の繰り返しとする。

次に、この動作を可能にするプログラムの例を示す。




```
#define THRESHOLD 60
task main()
{
      SetSensorLight(IN_3);
      while(true){
            OnFwd(OUT_AC, 75);
            Wait(500);
            while(Sensor(IN_3) < THRESHOLD);
            OnRev(OUT_AC, 75);
            Wait(500);
            while(Sensor(IN_3) < THRESHOLD);
      }
}
```

魔の境界問題: このプログラムでは

```
Wait(500);
```

がないとうまく動作しない。(500 に関してはもっと適切な値があるかも知れないが...)
これを外して実行してみよ。そして何故うまく動作しないか考えてみよ。
ヒント: while から抜け出した直後には

```
Sensor(IN_3) >= THRESHOLD
```

となっているはずである。

問題2 問題1において、ロボットが白い紙に出会ったら、向きを変えて戻ってくるようにしなさい。ロボットの向きを変えるには左右の車の回転を逆にすればよいであろう。(正確に U ターンさせるのは難しく、そのためにロボットはしだいに紙から離れて行くと思われる。2週回ぐらいできれば可とする。)

問題1では左の紙と右の紙でのロボットの動き方が異なるが、問題2ではUターンなので、ロボットから見た時には左の紙での動作と右の紙での動作には違いはないはずである。そのように考えると問題1のプログラムよりもシンプルにプログラムできるはずである。

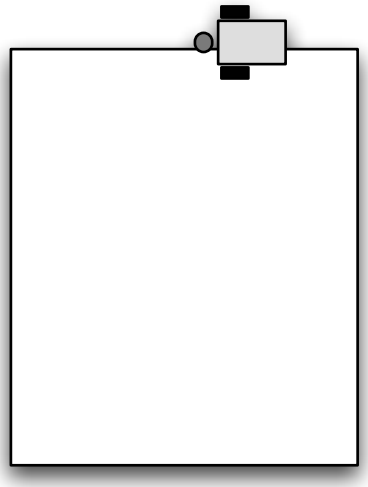
Lesson 11 (Light Sensor 3)

例題

大きな白紙を左回り(反時計回り)に周回させる。白紙は灰色の床の上に置かれている。

プログラム例

```
#define THRESHOLD 60
task main()
{
      SetSensorLight(IN_3);
      while(true){
            OnFwd(OUT_A, 75);
            while(Sensor(IN_3) < THRESHOLD);
            Off(OUT_A);
            OnFwd(OUT_C, 75);
            while(Sensor(IN_3) >= THRESHOLD);
            Off(OUT_C);
      }
}
```



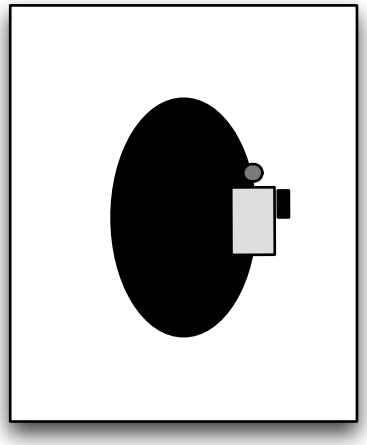
光センサーは IN_3 へ
モーターは A(右モーター) と C(左モーター)

解説

紙の周囲を左回り(反時計回り)で周回させる。そのために、
センサーが一紙の中に入れば、右のモーターを止め、左のモーターを動かす。
センサーが一紙の外に出れば、左のモーターを止め、右のモーターを動かす。
以上を繰り返す。

問題1

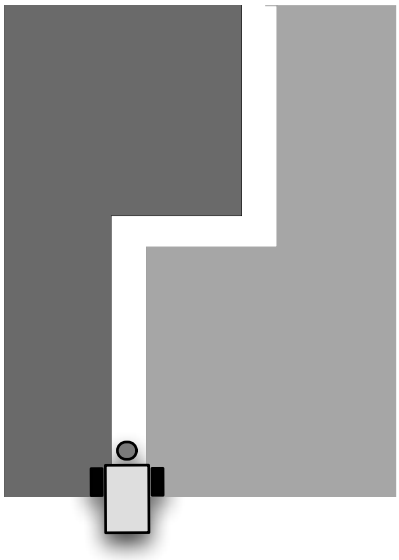
右図のように白紙の中央に描かれた黒い楕円の周囲をロボットを左回りに周回させる。
注釈: この問題は Lesson 5 (NXT Program) 課題4 で行ったが成功しなかった。(NXT ではロボットの旋回能力に問題があったのである。) しかし今回はきめの細かい制御ができるので成功する可能性がある。



問題2

白い紙の中でロボットをスタートさせる。
紙の中でのロボットの動きは直進とする。
紙からセンサーがはみ出したら、センサーが紙の中に入るまでロボットの左の車輪を止める。右の車輪はその間、前進方向に回転したままとする。(従ってロボットは左に向きを変える)
センサーが紙の中に入ったら再び直進させる。
以上を繰り返す。(ロボットは紙の中を左回りにぐるぐる回るはずである)

問題3



紙から白い場所に置いて、白い部分からできるだけはみ出さないで紙から脱出させるプログラムを考えてみよう。
ロボットのセンサーは、紙の白い場所、紙の濃い灰色の場所、淡い灰色の場所の各々を区別しなくてはならない。そこで次のように考えれば良いと思われる。

1. 白い部分で直進
2. 淡い灰色に出会ったら左向きに (白になるまで) 旋回し直進
   濃い灰色に出会ったら右向きに (白になるまで) 旋回し直進

まず、この問題に現れる3つの色(白、淡い灰、濃い灰)を区別するための適切なしきい値を決定する。各色の反射率は Lesson 4 (View) の “Reflected light” を使えば分かる。白、淡い灰色、濃い灰色で各々の反射光の強さを測定し、白と淡い灰色の中間値を `THRESHOLD1`、淡い灰色と濃い灰色の中間値を `THRESHOLD2` とする。すると

白では　`Sensor(IN_3) > THRESHOLD1`
淡い灰では　`Sensor(IN_3) < THRESHOLD1` かつ `Sensor(IN_3) > THRESHOLD2`
濃い灰では　`Sensor(IN_3) < THRESHOLD2`

の条件が成立する。次のプログラムで ?? に数字を書きなさい。

```
#define THRESHOLD1 ??
#define THRESHOLD2 ??
task main()
{
      SetSensorLight(IN_3);
      while(true){
            OnFwd(OUT_AC, 75);
            while(Sensor(IN_3) > THRESHOLD1);
            Wait(??); /* consider why this is required */
            if(Sensor(IN_3) < THRESHOLD2){
                  /* black, turn right */
                  Off(OUT_A);
                  OnFwd(OUT_C, 75); /* required? */
            }
            else{
                  /* gray, turn left */
                  Off(OUT_C);
                  OnFwd(OUT_A, 75); /* required? */
            }
            while(Sensor(IN_3) <= THRESHOLD1);
      }
}
```

注意: ここでもまた**魔の境界問題**が登場する。`Wait` がその処理のために必要になる。

C言語のコメント

`/*` から `*/` まではコメントと言い、実行に影響は無い。メモとして使う。

`if(条件){ .... }else{ .... }`

`else` の後の命令は `if` の条件が成立しないときに実行される。

Lesson 12 (Rotation Sensor)

例題1 ロボットを前に2秒間進め、後に向きを変えて戻ってくるようにする。

```
void rotate(int angle, int pwr)
{
      int motor_angle;
      motor_angle = 500*angle/100; /* replace 500 by a suitable value */
      ResetRotationCount(OUT_AC);
      Wait(100); /* wait until the counter is really reset */
      if(motor_angle > 0){
            OnFwd(OUT_A,pwr);
            OnRev(OUT_C,pwr);
            while(MotorRotationCount(OUT_A)
            - MotorRotationCount(OUT_C) < motor_angle);
      }
      else{
            OnRev(OUT_A,pwr);
            OnFwd(OUT_C,pwr);
            while(MotorRotationCount(OUT_C)
            - MotorRotationCount(OUT_A) < -motor_angle);
      }
      Off(OUT_AC);
}

task main()
{
      OnFwd(OUT_AC,75);
      Wait(2000);
      rotate(180,30); /* turn 180 degrees with 30% power */
      OnFwd(OUT_AC,75);
      Wait(2000);
      Off(OUT_AC);
}
```

解説

ここでは指定された角度だけロボットの向きを変える命令を作って問題を処理する。つまり

```
      rotate(180,30)          # 30% のパワーで、正方向(反時計回り)に180度回転
```

のように使える新たな命令を作るのである。このような命令があれば他の問題に応用しやすい。ロボットの向きを変えるためには左右の車輪を逆方向に回転させればよい。指定された回転数だけ正確に車輪を回転させるために、ここでは回転センサー(rotation sensor)を使う。これを使うと、ロボットが前進する方向に車輪Aを90度回転させるには

```
      ResetRotationCount(OUT_A);
      OnFwd(OUT_A,30);
      while(MotorRotationCount(OUT_A) < 90);
```

となる。実験してみると、左右の車輪の回転角を同時に正確にコントロールする事は難しいことがわかる。つまり

```
      ResetRotationCount(OUT_AC);
      OnFwd(OUT_A,30);
      OnRev(OUT_C,30);
      while(MotorRotationCount(OUT_A) < 90);
```

としても実際には車輪Aが90度回転したときに車輪Bも90度(逆に)回転している保証はない。
そこで、ロボットの旋回角を決めるのは左右の車輪の回転角の差である事を利用して旋回角の精度を上げることにする。(車輪が逆方向に回転すると MotorRotationCount の値は負になる)

```
ResetRotationCount(OUT_AC);
OnFwd(OUT_A,30);
OnRev(OUT_C,30);
while(MotorRotationCount(OUT_A) - MotorRotationCount(OUT_C) < 180);
```

仮に車輪Aの回転角が 90 の時に車輪Bの回転角が -90 にならば、この条件は先の条件と一致することに注意しよう。(一致するように作ったのである)

ロボットを逆方向に旋回させる場合には A と C の立場を入れ替えることになる。

例題のプログラムでは ResetRotationCount の直後に Wait(100) を入れている。実験によるとカウンターをリセットしても直ちにはカウンターが 0 にはなってくれないからである。

車輪の回転角とロボットの旋回角は比例する。比例定数は実験的に定める方がやさしい。

さて例題のプログラムでは

```
rotate(180,30)
```

が実行される。すると

```
void rotate(int angle, int pwr)
```

の angle は 180 に、pwr は 30 に設定される。なお int は integer(整数)の意味である。

注 NXC Guide(p.57)によれば、ロボットを旋回させるために車輪を逆回転させる命令は LEGO NXC に初めから備わっています。

```
RotateMotorEx(OUT_AC,30,180,-100,true,true)
```

とすれば左右の車輪が180度だけ逆向きに回転するはずです。その結果ロボットは旋回するはず(180度旋回する保証はない)ですが、正の向き(反時計回り)に旋回するか、負の向きに旋回するかはマニュアルを見てもよく分かりません。実験するしかないでしょう。

問題1

例題のプログラムの

```
motor_angle = 500*angle/100;
```

の 500 を調整して、

```
rotate(180,30)
```

で 180 度旋回するようにしなさい。

問題2

上の問題1の調整で rotate(180,75) でもちゃんと180度旋回するか吟味せよ。

問題3

1m程前に進み、反時計回りに180度旋回し、50cm程進み、時計回りに90度旋回して止まるプログラムを書きなさい。

Lesson 13 (Random Number)

例題1 フィギャースケートをイメージしてプログラムを作ってみました。

```
void rotate(int angle, int pwr)
{
      ここには Lesson 12 の例題のコードを書く

}

task main()
{
      int move_time, rotate_angle;
      while(true) {
            move_time = Random(600); /* 0 to 599 */
            rotate_angle = Random(2161) - 1080; /* -1080 to 1080 */
            OnFwd(OUT_AC, 75);
            Wait(move_time);
            rotate(rotate_angle, 70);
      }
}
```

解説

ランダムな動きを可能にしているのが

```
Random(600)
```

のような関数です。この場合、この関数は 0 から 599 までの間の乱数を生成します。例題のプログラムでは生成された乱数は一旦 move_time とか rotate_angle などの変数に納められています。この方が

```
rotate(Random(2161) - 1080, 70);
```

のように直接書くよりもプログラムの意味が分かりやすいからです。

問題1 ルーレットロボ

1mほど真っ直ぐ進み、乱数によって適当に旋回して、また1mほど真っ直ぐ進むプログラムを書きなさい。(このロボットは半径1mの大きなルーレットとして使えるだろう)