1. nxtOSEK特有の表現

* タスクの組み方

タスクは関数とは別個に宣言される必要がある。タスクの宣言はOILファイル上で行う。ファイル内に各タスクの名前、実行周期、優先度等を記述することで宣言することができる。複数タスクを並列で実行する場合は、タスクを複数記述すればよい。

タスクの処理内容は宣言とは別にプログラムファイル内（場所は問わない）に記述する。各タスクに対してその処理内容（関数呼び出しなど）を記述する。

OILファイル上でのタスク宣言とプログラムファイル上での処理内容記述によって、各タスクを決められた周期で動作させることができる。

* 各センサの利用方法

各センサにはそれぞれ固有の関数（API）が用意されており、利用したいタイミングでそれらを呼び出すことによって各センサを利用することができる。例えば、光センサの値を取得したい場合は、プログラム内にecrobot\_get\_light\_sensor(NXT\_PORT)と記述すれば良い。この際、引数にセンサが接続されているポート番号を指定する必要がある。

* 特殊なAPI

現時点では不明。systick\_wait\_ms()などは可能性あり。

* バランサーについて
* 開発環境

前提としてC言語がコンパイルできる環境。cygwin、GNUARM、nxtOSEKが必要。

1. leJOS-NXJ特有の表現

* タスクの組み方

タスクを組む場合は、Runnableインターフェースを実装（implements）する必要がある。Runnableインターフェースではrunメソッドが定義されており、これを実装した物がタスクとなる。よって、各クラスにつきタスクを一つ定義できることになる。

また、runメソッドはインスタンスを通して呼び出すのでは無く、threadクラスのインスタンスを引数にrunメソッドを実装したクラスを指定して呼び出すことで生成し、startメソッドを呼び出すことでタスクが開始する。例えば、

Thread tracerThread = new Thread(tracer);

tracerThread.start();

と記述することで、tracerクラス内のrunメソッドがタスクとして動き出す。

ただし、leJOS-NXJは周期的な動作を提供していないため、while文を利用して繰り返し処理を実装することで擬似的に周期的な動作を実現する必要がある。

複数のrunメソッドを同時に動作させることも可能であるため、並列処理を実現することはできる。

* 各センサの利用方法

各センサはクラスとして表現されており、それらをインスタンス化することによってそれぞれのインスタンスを独立したものとして使用することができる。各クラス毎にメソッドが定められており、それらをAPIとして使用することができる。例えば、光センサの値を取得する場合は次のようになる。

private LightSensor light = new LightSensor(SensorPort.S3);

light.readNormalizedValue();

センサ類は値の取得が目的であるため、インスタンスの生成に関してはあまり考える必要が無いが、値を保持するモータ類やジャイロセンサに関しては、インスタンスを保持する場所について考慮しておく必要がある。

* 特殊なAPI

nxtOSEKは最低限の機能のみを提供しているが、leJOS-NXJはそれに加えてある程度応用的な機能も提供している。例えば、nxtOSEKでは光センサの取得値は0から1024の値に限定されていたが、leJOS-NXJではそれに加えてパーセンテージや最大値、最小値の取得も行うことが可能となっている。また、センサに対していくつかのモードを設定することも可能（例えば超音波センサに対しては、（後述）。

* バランサーについて

leJOS-NXJでは標準の倒立制御APIは提供されていない。よって、自力で倒立制御APIを作成する、誰かが作成したAPIを利用する、倒立制御をしないなどでこの問題を解決する必要がある。

誰かが作成したAPIとしては、「銀河系はねうま」の作成した倒立振子制御 Java APIが広く使われている。このAPIはnxtOSEKと同様に利用することが可能である（オブジェクト化は必要）。

* 開発環境

前提としてJavaがコンパイルできる環境が必要。その環境に加えてleJOS-NXJが必要。eclipseにプラグインを導入することも可能。また、ファームウェアをleJOS-NXJにする必要がある。

1. leJOS-NXJ、nxtOSEK共通の表現