OPEN-R SDKによるAIBOのプログラミング

AIBO Programming using OPEN-R SDK

ネットワーク情報学部　石原 秀男
School of Network and Information　Hideo ISHIHARA

Keywords : AIBO, Robot, OPEN-R SDK, C++, Distributed Object-Oriented Programming

1．はじめに

SONYが発売している動物型ロボットAIBOは、「ロボットに癒されるなんて……」というある意味本質的かも知れない議論を置いておけば、面白い玩具である。一般的なAIBOの楽しみ方は、メモリスティックで提供される専用のソフトウェア（AIBOウェアと呼ばれる）を本体スロットに挿入して起動し、プログラムに応じた動作を楽しむことであるが、筆者のような犬好きの人間の目には、AIBOの動きはなかなかに愛らしく見える。

さて、ハードウェアとしてのAIBOは、制御用CPUとしてMIPS4000系列のプロセッサを搭載した組み込みシステムであり、15箇所の関節[1]などの可動部や、タッチ、加速度、距離、温度、各関節角度のセンサを有している。ソフトウェアとしては、C++による開発環境であるOPEN-R SDK（Open Architecture for Entertainment Robot）が無償で提供されるともに、APIの仕様などが記載されたマニュアルやサンプルプログラムなども公開されている。つまり腕さえあれば、かなりプリミティブな部分からのロボット制御にチャレンジできるのである。もっとも「腕さえあれば」というところが問題であり、通常のプログラミング開発とはかなり異なる部分もあるために、それなりに敷居が高いことは否定できない。

本稿ではユーザのキーボード入力に応じて、AIBOが首を前後（YESを表現する）、あるいは左右（NOを表現）に振るという簡単なアプリケーションYESNOの作成法を通じて、AIBOプログラミングのための入門的知識を解説することとする。

2．開発のための環境

OPEN-R SDKによるAIBOのプログラミング開発には以下のものが必要になる。

1）AIBO本体とメモリスティック

SONYによれば、OPEN-Rによるソフトウェア開発の対象となるのはAIBO ERS-7シリーズ、ERS-200シリーズであり、8万円台と比較的安価で入手できるERS-300シリーズは非対応とされている。その理由として、SONYはERS-300シリーズには無線LAN環境が存在せずデバッグが困難なためであると説明している。確かにERS-7シリーズ、ERS-200シリーズは無線LAN機能を内蔵しているか、もしくはオプションで内蔵可能であり、リモートPCとの間で無線通信を行うことが可能である。これを用いればAIBO本体からのメッセージをリモートPCのコンソールに表示することや、リモートPCから無線LAN経由でAIBOにデータを送信することができる。このことは制御ロジックの計算をデスクトップPCの強力なCPUで行い、AIBOの内蔵CPUにはセンサ情報の取得やモータのコントロールだけを行わせるというシステム構成を可能にしており、本格的な制御への道を開くという意味を持つ。しかしながら、市販のAIBOウェアのように内蔵CPUのみでAIBOをコントロールするアプリケーションにおいては無線LAN機能が必要でないことは言うまでも無い。確かにAIBO本体からのメッセージはデバッグに多少役立つものの、プログラム上の通過点を確認できる程度のもので、それがなければプログラム開発が不可能であるというほど強力なものではない。ハードウェアなのだから、動作は目で見て確認できるわけで、メッセージ表示ができなければ開発が不可能ということにはならない。

さて、各AIBOのシリーズは以下のようなスペックの内蔵CPUを持っているが、SONYによれば高クロックのCPUを持つAIBOではプログラムの実行速度が速くなるということだけのことであり、それ以外の違いは特にないようである。

Table.1　AIBOの内蔵CPU

AIBO本体	内蔵CPU	動作クロック	メインメモリ
ERS-210/220	MIPS	192MHz	32MB
ERS-210A/220A	MIPS	384MHz	32MB
ERS-7	MIPS	576MHz	64MB

ERS-300シリーズがどのようなCPUを使用しているかについては公開されていないが、クロック、メインメモリなどに違いがあったとしてもMIPS系列の全く同じアーキテクチャを有するCPUであるのはほぼ確実であろう。であるとすると、OPEN-Rが開発されたソフトウェアが動作しな

[1] 4本の足に各3関節と首に3関節で計15関節である。

い理由は何もない。そういうわけで、もしERS-300シリーズを所有しているのならばOPEN-Rによる開発にチャレンジしてみる価値はあるだろう。もちろん、新規に入手するのならERS-7シリーズもしくはERS-200シリーズが絶対である。残念ながら、現在市販されているAIBOでこの系列に属するものはERS-7M2のみであり、価格も185,000円と相当に高い。なお、本稿において筆者が使用しているのはERS-220Aであるが、プログラムの開発法ならびに動作については他の200シリーズ[2]あるいは7シリーズでも全く同様である。

また、作成したプログラムをAIBOで動作させるためには、AIBO専用のプログラミングメモリスティックが必要となる。このメモリスティックは形状については一般的なものと同じであるが、AIBO専用品であり汎用品で代用[3]することはできないとされている。

2) OPEN-R SDK

開発キットであるOPEN-R SDKは、文献1)のOPEN-Rオフィシャルウェブサイトから無償でダウンロードできる。OPEN-R SDKには

インストールガイド
(OPEN-R開発環境のインストール法)
プログラマーズガイド
(OPEN-Rによる開発法)
レベル2リファレンスガイド
(OPEN-Rのクラスと APIの解説)
OPEN-Rインターネットプロトコル バージョン4
(ネットワーキングの解説)
モデルインフォメーション ERS-210
(ERS-210の機種情報)
モデルインフォメーション ERS-220
(ERS-220の機種情報)
モデルインフォメーション ERS-7 (ERS-7の機種情報)

の各マニュアルと

OPEN_R_SDK-1.1.5-r2.tar.gz (OPEN-R SDK本体)
OPEN_R_SDK-sample-1.1.5-r1.tar.gz
(サンプルプログラム)
cygwin-packages-1.5.5-bin.exe (cygwinとGNUツール)
mipsel-devtools-3.3.2-bin-r1.tar.gz
(MIPSクロスコンパイラ)

の各ファイルが付属している。これ以外にもcygwinとGNUツール、MIPSクロスコンパイラのソースなども含まれており、コンパイルの手間さえいとわなければ、Linuxを含むUnix系のOSやMacOS上での開発も可能なようである。

3) 開発用マシン

SONYが推奨している開発環境は

OS : Windows 2000 Professional/XP
CPU : Pentium 233MHz以上
メモリ : 64MB以上
ハードディスク : 200MB以上の空き容量
無線LAN環境 : IEEE802.11b準拠のWireless LANカードもしくはアクセスポイント

である。OSとしてはLinux、FreeBSD、Solaris、MacOS Xもサポートされているが、その場合にはツールをソースからコンパイルし直さなければならない。Windows版についてはバイナリが付属しているので、Windowsマシンがあればわざわざ余計な手間をかけることもないだろう。要求されるマシンのスペックはかなり低く、いまどきこのスペックを満たしていないマシンはほとんどないはずである。無線LANに関してはノートならばPCカード、デスクトップマシンならばUSB接続などで簡単に用意できるだろう。なお無線LANカードのみでAIBOと通信する場合にはアドホックモードでの接続となる。筆者の場合にはMELCO製の無線LANアクセスポイントを導入しているので、それを利用しているが、アクセスポイントを使う場合にはインフラストラクチャーモードでの接続となる。またメモリスティックを読み書きするための機器も必要になるが、数種類のメディアに対応するUSB接続のマルチカードリーダライタが安価に売られている。

3. 開発環境の構築

まず以下の手順で各ツールのインストールを行う。なお、すべてのダウンロードファイルはc:¥tempに置かれているものとする。

1) cygwinのインストール

(1) ダウンロードしたcygwin-packages-1.5.5-bin.exeをダブルクリックしてUnzipする。
(2) cygwin-packages-1.5.5フォルダが作成されるので、その中にあるsetup.exeを実行。
(3) (3)[Install from Local Directory]を選択し、[Next->]をクリック。
(4) インストールディレクトリをC:¥cygwin(デフォルトのまま)にし、テキストファイル型に[UNIX]を選択して、[Next->]をクリック。
(5) setup.exeがあるフォルダを指定し、[Next->]をクリック。

[2] ERS-210の初期に出荷されたものについてはAIBO内蔵のFlash ROMのアップデートが必要である。この手順についてはOPEN-R SDK付属のOPEN-Rインストールガイドに記述されている。
[3] 手元にあった通常のデータ用メモリスティック(64MB)でテストしてみたが、動作しなかった。

(6) インストールする内容の選択画面が表示されるが、デフォルトのままで[Next->]をクリック。

以上でcygwinがインストールされる。以下の作業はcygwin上で行うことになる。

2) gcc（MIPSクロスコンパイラ）のインストール
(1) デスクトップに作成されたアイコンをダブルクリックしてcygwinを起動する。
(2) cd/usr/localとしてカレントディレクトリを/usr/localに移動する。
(3) tar zxvf c:/temp/mipsel-devtools-3.3.2-bin-r1.tar.gzとしてtarコマンドでパッケージを解凍しインストールする。

これによって/usr/local/OPEN_R_SDKディレクトリが作成され、その下にgccのツール群がインストールされる。続けてcygwin上でOPEN-R SDKのインストールを行う。

3) OPEN-R SDKのインストール
(1) カレントディレクトリが/usr/localであることを確認する。
(2) tar zxvf c:/temp/OPEN_R_SDK-1.1.5-r1.tar.gzとしてtarコマンドでパッケージを解凍しインストールする。

必須ではないが、サンプルプログラムもインストールしておくと重宝するだろう。

4) サンプルプログラムのインストール
(1) ホームディレクトリに移動する。（筆者の場合は cd/home/ishihara）
(2) tar zxvf c:/temp/OPEN_R_SDK-1.1.5-r1.tar.gzとしてtarコマンドでパッケージを解凍しインストールする。

ホームディレクトリの下にsampleディレクトリが作られ、そこに各種のサンプルプログラムがインストールされる。以上でソフトウェアツールのインストールは終了である。

最後にAIBOプログラミングメモリスティックへのシステムプログラムのコピーと無線LANの設定ファイルの変更を行う。

5) メモリスティックへのファイルコピーと無線LANの設定
(1) c:¥cygin¥usr¥local¥OPEN_R_SDK¥OPEN_R¥MS_ERS200¥WCONSOLE¥memprot¥OPEN-Rをフォルダごとメモリスティックのルートにコピーする。
(2) メモリスティックのOPEN-R/SYSTEM/CONFにあるWLANDFLT.TXTの設定項目を
ETHER_IP＝192.168.11.24
IP_GATEWAY＝192.168.11.1（アクセスポイントのIPアドレス）
ESSID＝無線LANのESSID
WEPKEY＝半角5文字[4]のアクセスポイントに割り当てられたWEPキー
と変更する。なお、これ以外の部分についてはデフォルトのままで構わない。上で192.168.11.24はAIBOに割り当てるIPアドレス、192.168.11.1はアクセスポイントのIPアドレスであり、これらの値は筆者の環境のものである。ESSID、WEPKEYを含めて各自の環境に合わせて変更して欲しい。

以上で開発環境の構築は終了である。

4．プログラムの構成

OPEN-Rのアプリケーションはオブジェクトの集合体である。個々のオブジェクトはコンパイルして得られた実行ファイル(*.bin)に対応しており、たとえばプログラムが三つのオブジェクトからなる場合、プログラマは三つのソースファイルを独立に開発することになる。AIBOの電源が投入されると、各オブジェクトは他のオブジェクトと並列に実行されるが、オブジェクト間の情報交換が必要な場合にはメッセージ通信によってそれを行う。

さて、これから作成しようとしているYESNOアプリケーションは、以下のような仕様とする。

1) AIBOの電源を投入すると、首の各関節（前後方向TILT、左右方向PAN、ひねり方向ROLLがある）の制御ゲインを初期化する。
2) 首の位置を原点に移動してユーザからの指示を待つ
3) ユーザが、リモートコンピュータから‘y’と入力すると、首をTILT方向に動かし2)の状態に戻る（YESを表現している）。
4) ユーザが、リモートコンピュータから‘n’と入力すると、首をPAN方向に動かし2)の状態に戻る（NOを表現している）。
5) ユーザが、リモートコンピュータから‘s’と入力すると、AIBOの電源を落とす。
6) AIBOのポーズスイッチが押されるかバッテリーの残量が少なくなるとAIBOの電源を落とす。

この仕様に対して次図のようにオブジェクトを構成することとした。

[4] WEPキーは64bitにしか対応していないため5文字に限られる。13文字のWEPは使えないので注意が必要である。

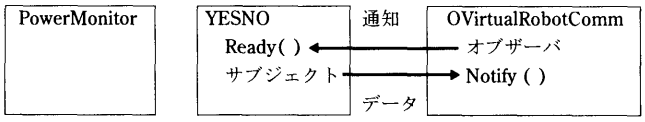

Fig.1 Configuration of YESNO

まずPowerMonitorであるが、これは仕様の6)に相当する。文献3)によるとすべてのアプリケーションはこのオブジェクトを必要とするとされているが、ソースコードを含めサンプルプログラムの中で提供されているので、それをそのまま使えばよい。

次にYESNOであるが、仕様の1)～5)に相当するこのプログラムの主要部分である。一方、OVirtualRobotCommは、AIBOのシステム部に存在するオブジェクトであり実際のハードウェア制御はこの部分が行う。YESNOとOVirtualRobotCommの関係は、YESNOからのメッセージ通信によってOVirtualRobotCommに命令データが渡され、それに応じてOVirtualRobotCommが実際にモータを動かすことになる。分散オブジェクト指向の趣旨に従うのならば、YESNOの中でもユーザの入力を受け入れる部分と、AIBOをコントロールする部分は別のオブジェクトとして実装すべきなのであるが、ユーザ入力についてはscanfが利用できるのでプログラムにして数行に過ぎない。わざわざオブジェクトを分けると通信の手間が増えてプログラムが複雑になるだけなので、今回は同じオブジェクトにまとめている。もちろん、首を動かすだけではなく、歩かせるなどの他の動作もさせる場合には、それぞれに対してオブジェクトを用意し、ユーザからの入力部分は分離するべきであろう。OVirtualRobotCommについてもシステムに存在しているものを利用するだけなので、結局のところ作成しなければならないのはYESNOということになる。

ここでYESNOとOVirtualRobotCommのメッセージ通信の方法について説明しておこう。実際にメッセージ通信を行うのは、各オブジェクトに含まれるサブジェクトとオブザーバである。サブジェクトは他のオブジェクトに対しメッセージを送信し、オブザーバは他のオブジェクトから送られてきたメッセージを受け取る。より具体的にはOVirtualRobotCommでデータ受け入れ準備が完了すると、Assertメッセージが送出され、そのイベントによってYESNOのReadyメソッド（メンバ関数として実装する）が呼び出される。Readyではそれを受けて、OVirtualRobotCommに送るデータを用意し、YESNOのサブジェクトがNotifyObserversメソッドを用いてデータを用意したことを通知する。OVirtualRobotCommでは、通知を受けるとNotifyメソッド（OVirtualRobotCommにメンバ関数として実装されている）が呼び出され、データに応じた処理が行われることになる。通信とは言うものの、実態としては共有メモリを用意しておいて、サブジェクト側でデータを書き込み、オブジェクト側でそれを読み込んでいるということだと推測される。

OPEN-Rのプログラミングでは、まずこのオブジェクト通信の構成を定め、それをstub.cfgというテキストファイルに記述することになる。YESNOのstub.cfgは以下のようになる。

```
ObjectName : YESNO
NumOfOSubject   : 1
NumOfOObserver  : 1
Service :"YESNO.Move.OCommandVectorData.S",
null, Ready()
Service:"YESNO.DummyObserver.DoNotConnect.O",
null, null
```

Prog.1 stub.cfg

ここでObjectNameはオブジェクトの名称、NumOfOSubjectはこのオブジェクトに含まれるサブジェクトの数、NumOfOObserverはこのオブジェクトに含まれるオブジェクトの数である。YESNOは、OVirtualRobotCommにAIBOの首の関節データを送信するだけなので、必要なのは一つのサブジェクトのみであり、オブザーバは必要ない。しかしOPEN-Rの仕様でstub.cfgには少なくとも一つのサブジェクトとオブザーバを記述しなければならないため、ここではダミーのオブザーバを用意することとしてNumOfOObserverを1としている。次に続くServiceは各サブジェクトとオブザーバに対応するもので、NumOfOSubject+NumOfOObserver行存在することになる。まず上のServiceであるが、"YESNO.Move.OCommandVectorData.S"がサブジェクトのサービス名である。より詳しく説明すると、YESNOはサブジェクトが属するオブジェクトのオブジェクト名、Moveは任意につけたサブジェクトのサブネーム、OCommandVectorDataは任意につけたサブジェクトからオブジェクトへ送信するデータのデータ名、Sはサービスのタイプでサブジェクトを表している。次のnullにはオブザーバとの接続結果を受信するControlメソッドの関数名を記述するが、サンプルプログラムなどを見ると必要ないようなのでここでは使用していない。使用しない場合にはnullを記述する。最後のReady( )はReadyメソッドの関数名で任意であるがここではReady( )とした。下のServiceはオブザーバに対応するものである。サブジェクトはダミーであることからこのようなサービス名にしてあるが、最後のOはサービスのタイプがオブザーバであることを示している。次の二つは、サブジェクトとの接続結果を受信するConnectメソッドと、前述のNotifyメソッドの関数名を記述するものであるが、ここではオブザーバ自体がダミーであるので両者ともにnullにしている。実際のオブザーバでは、Connectを使う必要性はないようであるが、Notifyは必要になるはずである。

さて、サブジェクトとオブザーバの記述はこれで完了だが、YESNOのサブジェクトの通信先がOVirtualRobotCommであるということはどこに記述するかというと、connect.cfgというテキストファイルである。connect.cfgは以下のように

なる。

```
YESNO.Move.OCommandVectorData.S
OVirtualRobotComm.Effector.OCommandVectorData.O
```

Prog.2 connect.cfg

このファイルにはアプリケーション全体の接続をすべて記述することになるが、本アプリケーションではYESNOのサブジェクトがOVirtualRobotCommのオブザーバに通信するだけなので1行のみである。書式としては

サブジェクトのサービス名 オブザーバのサービス名

であるが、OVirtualRobotCommはシステムオブジェクトであるから名称[5]を勝手に変更してはならない。このconnect.cfgはAIBOメモリスティックのOPEN-R¥MW¥CONFディレクトリに置いておく。

一方、stub.cfgは後で作成するYESNOのヘッダファイルYESNO.hとC++によるソースファイルYESNO.ccと同じディレクトリに置いておく。実際のコンパイルはGNU makeによって行うことになるが、 makeファイルの中ではstubgen2コマンドが呼び出され、オブジェクト通信のために必要なファイルをstub.cfgから自動生成する。これについて少し説明をしておこう。stubgen2が作成するファイルの中にdef.hがあるが、その内容は以下のようなもの[6]である。

```
const char* const objectName = "YESNO"
const int numOfSubject  = 1;
const int numOfObserver = 1;
const char* const
subjectService[numOfSubject] =
{
   "YESNO.Move.OCommandVectorData.S"
};
   const char* const observerService
   [numOfObserver] =
{
   "YESNO.DummyObserver.DoNotConnect.O"
};
const int sbjMove = 0;
const int obsDummyObserver = 0;
```

Prog.3 def.h

サブジェクトやオブジェクトは複数存在する可能性があるため配列化されているが、注意しなければならないのは変数sbjMoveである。この名称はサブジェクト"YESNO.Move.OCommandVectorData.S"のサブネームMoveにsbjを付加して自動生成されるが、この変数がこのサブジェクトを表すインデックスになる。つまり、詳しくは後で述べるが、YESNOからOVirtualRobotCommにデータを送る場合には

```
subject[sbjMove]->SetData(rgn);
```

として、オブザーバ用のデータバッファにデータをセットして

```
subject[sbjMove]->NotifyObservers();
```

でOVirtualRobotComm側のNotifyメソッドを呼び出すことになる。変数名まで自動生成されて、それを自分のプログラムの中に記述しなければならないというのは、少し余計なお世話にも思えるが、これはOPEN-Rの仕様であるから素直に従うしかない。

5. プログラムの作成

ここではアプリケーションの主要部分であるYESNOオブジェクトの作成を行う。なおこのプログラムはサンプルプログラムに含まれるMovingHeadアプリケーション[7]をベースに作成したもので、相当部分を流用している。

まずAIBOの取る状態として、以下の8個を考えることとする。

HS_IDLE
: 何もしない状態

HS_START
: AIBOの電源投入直後の各関節位置を読み取り、現在の指示値としてそれを設定する。終了したらモードをHS_ADJUSTING_DIFF_JOINT_VALUEに変更する。

HS_ADJUSTING_DIFF_JOINT_VALUE
: AIBOの各関節のモータ制御におけるPIDゲインを設定し、首の位置を原点へと移動開始する。終了したらモードをHS_MOVING_TO_ZERO_POSに変更する。

HS_MOVING_TO_ZERO_POS
: AIBOの首を原点へ移動している途中状態。首が原点へ移動したらモードをHS_WAITに変更する。

HS_WAIT
: ユーザからの入力を待つ状態。ユーザから'y'が入力されたらモードをHS_SWING_HEAD_YES

[5] どのような名称かは文献4)に記載されている。
[6] 自動生成されるコメント部分は省略してある。
[7] 電源を投入するとAIBOが無限に首を左右に振り続けるというアプリケーションである。

に、'n' が入力されたらHS_SWING_HEAD_NO
に、's' が入力されたらHS_FINISHに変更する。

HS_SWING_HEAD_YES
AIBOが首をYES方向に振っている状態。首を振り終えて原点へ戻ったらモードをHS_WAITに変更する。

HS_SWING_HEAD_NO
AIBOが首をNO方向に振っている状態。首を振り終えて原点へ戻ったらモードをHS_WAITに変更する。

HS_FINISH
AIBOをシャットダウンする

状態遷移の流れは下図のようになる。

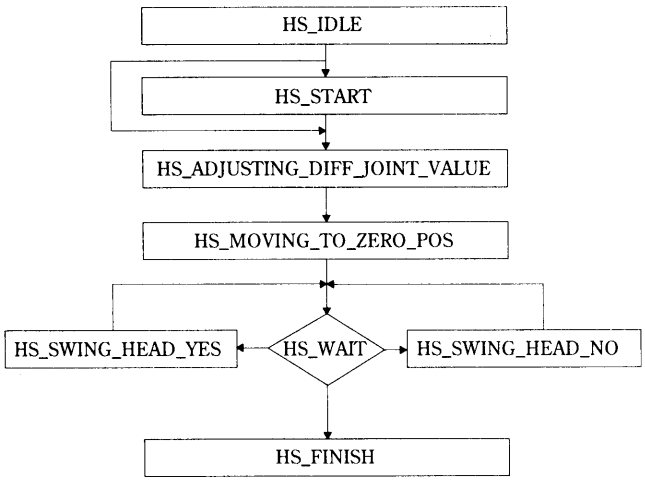

Fig.1 Transitions of States

各状態に対する定数の定義を含めてヘッダファイルYESNO.hは以下のように作成した。

```
#include <OPENR/OObject.h>
#include <OPENR/OSubject.h>
#include <OPENR/OObserver.h>
#include "def.h"

enum HEADState {
    HS_IDLE,
    HS_START,
    HS_ADJUSTING_DIFF_JOINT_VALUE,
    HS_MOVING_TO_ZERO_POS,
    HS_SWING_HEAD_YES,
    HS_SWING_HEAD_NO,
    HS_WAIT,
    HS_FINISH
};

enum MovingResult {
    MOVING_CONT,
    MOVING_FINISH
};

class YESNO : public OObject {
public:
    YESNO();
    virtual ~YESNO() {}

    OSubject*  subject[numOfSubject];
    OObserver* observer[numOfObserver];

    virtual OStatus DoInit
    (const OSystemEvent& event);
    virtual OStatus DoStart
    (const OSystemEvent& event);
    virtual OStatus DoStop
    (const OSystemEvent& event);
    virtual OStatus DoDestroy
    (const OSystemEvent& event);

    void Ready(const OReadyEvent& event);

private:
    void          OpenPrimitives();
    void          NewCommandVectorData();
    void          SetJointGain();
    MovingResult  AdjustDiffJointValue();
    MovingResult  MoveToZeroPos();
    MovingResult  SwingHead();
    RCRegion* FindFreeRegion();
    void SetJointValue(RCRegion* rgn, int
    idx, double start, double end);

    static const size_t NUM_COMMAND_VECTOR = 2;
    static const size_t NUM_JOINTS         = 3;
    static const int    TILT_INDEX         = 0;
    static const int    PAN_INDEX          = 1;
    static const int    ROLL_INDEX         = 2;

    static const word   TILT_PGAIN         = 0x000a;
    static const word   TILT_IGAIN         = 0x0008;
    static const word   TILT_DGAIN         = 0x000c;

    static const word   PAN_PGAIN          = 0x000d;
    static const word   PAN_IGAIN          = 0x0008;
    static const word   PAN_DGAIN          = 0x000b;

    static const word   ROLL_PGAIN         = 0x000a;
```

```
    static const word   ROLL_IGAIN          = 0x0008;
    static const word   ROLL_DGAIN          = 0x000c;

    static const word   PSHIFT              = 0x000e;
    static const word   ISHIFT              = 0x0002;
    static const word   DSHIFT              = 0x000f;

    static const int ZERO_POS_MAX_COUNTER  =  16;

    HEADState headState;
    OPrimitiveID      jointID[NUM_JOINTS];
    RCRegion*         region[NUM_COMMAND_VECTOR];
};
```

Prog.4 YESNO.h

簡単に説明しておこう。<OPENR/OObject.h>、<OPENR/OSubject.h>、<OPENR/OObserver.h>はOObjectクラス、OSubjectクラス、OObserverクラスのためのヘッダファイルであり、"def.h"は前述したようにstub.cfgからstubgen2によって自動的に生成されるヘッダファイルである。次のenum HEADStateはAIBOの取る前述の8個の状態に対応するHEADState列挙型の定数定義であり、enum MovingResultはYESやNOなどの一連の動作が継続中であるか、終了したかを示すMovingResult列挙型の定数定義である。

以降はclass YESNOの定義であるが、OPEN-Rのオブジェクトは基底クラスOObjectから派生させる。YESNO( ),-YESNO( ) {}はYESNOクラスのコンストラクタ、デストラクタである。次のsubject[]とobserver[]はYESNOクラスのサブジェクトとオブザーバの定義であり、サブジェクトおよびオブザーバが複数である場合に対応して配列にしなければならない[8]。実際にはnumOfSubject、numOfObserverともに1であり余計な手間なのだが、これも仕様なので仕方がない。次の4つの関数DoInit( )、DoStart( )、DoStop( )、DoDestroy( )は、OPEN-Rのオブジェクトにおいては必ず実装しなければならない関数である。DoInit( )はこのクラスのエントリポイントであり、AIBOに電源が投入されると最初に呼び出される関数である。通常のアプリケーションには複数のオブジェクトが存在することになるが、すべてのオブジェクトにおいてDoInit( )が呼び出された後にDoStart( )が呼び出される。DoStop( )はAIBOがシャットダウンされるとき(たとえばポーズスイッチが押されたとき) に呼び出される関数で、DoDestroy( )はすべてのオブジェクトでDoStop( )が行われた後に呼び出される関数である。つまりオブジェクトの動作の中心となる部分はDoStart( )に記述されることになる。

Ready( ) はstub.cfgやconnect.cfgに記述したようにOVirtualRobotCommからデータ受け入れ準備完了のメッセージが通知されたときに呼び出される関数である。

続いてプライベート部では、まず

OpenPrimitives( )
: CPCプリミティブ(各関節のモータやセンサなどAIBOにおいて操作対象となるもの) を開く関数

NewCommandVectorData( )
: サブジェクトが送信するデータサイズなどの初期設定をする関数

SetJointGain( )
: TILT、PAN、ROLL方向の関節制御のためのPIDゲインを設定する関数

AdjustDiffJointValue( )
: 関節制御のための初期状態を設定するための関数

MoveToZeroPos( )
: 首を原点へ移動させる関数

SwingHead( )
: 首をYESもしくはNO方向に振る関数

FindFreeRegion( )
: サブジェクトが送信するデータの格納場所の中で空いているものを見つける関数

SetJointValue( )
: 首を移動させる角度を設定する関数

という各関数の定義を行っている。続くNUM_COMMAND_VECTORは使用するOCommandVectorDataの数であるが、ダブルバッファリングによって絶え間なくデータを送れるように2になっている。つまり、データは2個用意しておいて、1個についてAIBOが動作している間に、もう一つのデータを用意することになるのである。NUM_JOINTは動かす関節の数で、ここではTILT、PAN、ROLL方向の3であり、TILT_INDEX、PAN_INDEX、ROLL_INDEXはそれぞれの関節を扱うときに使用する番号である。続くTILT_PGAINからDSHIFTまでは、各関節の制御のためのPIDゲインである。PIDゲインとはフィードバック制御の基本的な手法であるPID制御において、誤差の比例値、積分値、微分値についてフィードバックをかけるときのゲインであるが、どのような値が適切であるかは当然制御系の性質によって異なる。ここではサンプルプログラムMovingHeadにおいて用いられている値をそのまま使用した。これよりも小さな値を用いても位置決めが遅くなる程度で実害はないが、大きな値を用いると機械的な破損につながる可能性があるので充分な注意が必要である。次のZERO_POS_MAX_COUNTERは、AIBOの首を任意の位置から原点へ移動させる場合に何ステップで行うかを定めている定数で、これもMovingHeadに従って16にしてある。たとえば、AIBOの首が初期状態においてTILT方向に+48度傾いていた場合、1ステップで3度ずつ移動して原点へ持ってくることになる。もちろん、ZERO_POS_MAX_COUNTERを1とすれば1ステップで原点へ移動する

[8] stubgen2が自動生成するファイルとの関係上、配列にしなければならない。

わけであるが、ハードウェアの限界があるので最悪の場合、破損につながる可能性[9]がある。この種の値についてはノウハウの領域なので、自信がない限りサンプルなどで示されている値に従っておくべきだろう。つぎのheadStateはHEADState型の変数で、AIBOが8つの中の、どの状態にあるのかを示す変数である。jointID[]はTILT、PAN、ROLLの三つの関節に対応するCPCプリミティブであり、region[]はOCommandVectorDataのための格納場所である。

続いてソースコードの本体であるYESNO.ccについて説明しよう。YESNO.ccでは最初に以下のようなファイルをインクルードする。

```
#include <OPENR/OPENRAPI.h>
#include <OPENR/OUnits.h>
#include <OPENR/OSyslog.h>
#include <OPENR/core_macro.h>
#include "YESNO.h"
```

Prog.5 includes

次に、YESNOクラスのコンストラクタを記述する。

```
YESNO::YESNO(){
    headState = HS_IDLE;
    for (int i = 0; i < NUM_JOINTS; i++)
    jointID[i] = oprimitiveID_UNDEF;
    for (int i = 0; i < NUM_COMMAND_VECTOR;
    i++) region[i] = 0;
}
```

Prog.6 YESNO( )

コンストラクタではheadStateをHS_IDLEにして、jointIDとregionを初期化している。続いて

```
OStatus YESNO::DoInit(const OSystemEvent&
event){
    NEW_ALL_SUBJECT_AND_OBSERVER;
    REGISTER_ALL_ENTRY;
    SET_ALL_READY_AND_NOTIFY_ENTRY;

    OpenPrimitives();
    NewCommandVectorData();
    OPENR::SetMotorPower(opowerON);
    return oSUCCESS;
}
```

Prog.7 DoInit( )

では、NEW_ALL_SUBJECT_AND_OBSERVERマクロで、YESNOに含まれるすべてのサブジェクトとオブザーバを生成し、REGISTER_ALL_ENTRYでサブジェクトのControl( )とオブザーバのConnect( )[10]をシステムに登録し、SET_ALL_READY_AND_NOTIFY_ENTRYマクロでサブジェクトのReady( )とオブザーバのNotify( )をシステムに登録する。この手順はオブジェクト間通信を使用するすべてのオブジェクトにおいて共通であるので、他のプログラムでもこのまま行えばよい。

ここではOpenPrimitives( )を呼び出し、制御対象であるTILT、PAN、ROLLに関するCPCプリミティブを開くが、そのコードは

```
void YESNO::OpenPrimitives(){
    OPENR::OpenPrimitive("PRM:/r1/c1-
    Joint2:j1",&jointID[TILT_INDEX]);
    OPENR::OpenPrimitive("PRM:/r1/c1/c2-
    Joint2:j2",&jointID[PAN_INDEX]);
    OPENR::OpenPrimitive("PRM:/r1/c1/c2/c3-
    Joint2:j3",&jointID[ROLL_INDEX]);
}
```

Prog.8 OpenPrimitives( )

である。ここで使われているPRM:/r1/c1-Joint2:j1などはAIBOの各部に対して定められているCPCロケータと呼ばれる定数で、値は文献5)に記載されている。OpenPrimitives( )を実行した後には、各関節を指定するためにjointID[]を使用することになる。

続けて呼び出されるのはNewCommandVectorData( )である。

```
void YESNO::NewCommandVectorData(){
    OStatus result;
    MemoryRegionID      cmdVecDataID;
    OCommandVectorData* cmdVecData;
    OCommandInfo*       info;

    for (int i = 0; i < NUM_COMMAND_VECTOR; i++) {
        OPENR::NewCommandVectorData
        (NUM_JOINTS,&cmdVecDataID,&cmdVecData);
        region[i] = new RCRegion(cmdVecData-
```

[9] ゲインを小さくしておかないと、モータに無理がかかる上に、制御結果にもオーバシュートやチャタリングなどを生じる可能性が高い。

[10] 実際はControl( )もConnect( )も使用しないので、NULLが登録される。

```
        >vectorInfo.memRegionID,
                cmdVecData->vectorInfo.offset,
                (void*)cmdVecData,
                cmdVecData->vectorInfo.totalSize);
        cmdVecData->SetNumData(NUM_JOINTS);
        for (int j = 0; j < NUM_JOINTS; j++) {
            info = cmdVecData->GetInfo(j);
            info->Set(odataJOINT_COMMAND2,
            jointID[j], ocommandMAX_FRAMES);
        }
    }
}
```

Prog.9 NewCommandVectorData ( )

では、二つのRCRegion、region[0]とregion[1]の設定を行っている。このregion[]がサブジェクトからOVirtualRobotCommのオブザーバへと送られるデータなのだが、前述のダブルバッファリングのために二つ(NUM_COMMNAD_VECTOR個) 用意されている。後述するが、rgn=FindFreeRegion( )として現在使用中でない(OVirtualRobotCommで実行されていない)region[]であるrgnを見つけ、それについてSetData(rgn)で値を設定し、NotifyObservers( )でそのことをオブザーバに通知する。region[]はYESNO、OVirtualRobotCommの両者からアクセス可能な共用メモリと考えればよいだろう。さて実際のデータの形式であるが、NUM_JOINTS個 (TILT、PAN、ROLLに対応する3個) から構成されるデータ単位がocommnadMAX_FRAMES個(定数で16と定義されている)集まることによって一つのデータとなっている。実は、最小単位のデータは8m秒ごとにOVirtualRobotCommで処理されるのだが、それが16個分、つまり128m秒分のデータを一つのかたまりとして指示するのである。

DoInit( )は最後にAIBOのモータの電源を入れる関数SetMotorPower( )を実行して終了するが、これの呼び出しは、AIBO上で動作しているオブジェクトのうちどれか一つで行えば充分である。

次にDoStart( )である。

```
OStatus YESNO::DoStart(const OSystemEvent&
event)
{
    if (subject[sbjMove]->IsReady()== true){
        AdjustDiffJointValue();
        headState =
         HS_ADJUSTING_DIFF_JOINT_VALUE;
    } else {
        headState = HS_START;
    }
    ENABLE_ALL_SUBJECT;
    ASSERT_READY_TO_ALL_OBSERVER;

    return oSUCCESS;
}
```

Prog.10 DoStart( )

IsReady( )でオブザーバの状態を調べ、準備完了ならAdjustDiffJointValue( )を呼び出し、AIBOのモードをPIDゲインを設定する状態HS_ADJUSTING_DIFF_JOINT_VALUEに変更する。そうでなければ、モードをAdjustDiffJointValue( )を呼び出す状態であるHS_STARTに変更する。すなわち、DoStart( )が呼び出された時点でオブザーバが準備完了なら、すぐにAdjustDiffJointValue( )を呼び出すが、そうでなければ、AdjustDiffJointValue( )を呼び出す状態にしておいて、オブザーバの準備完了を待つことになる。オブザーバの準備が完了すると、Ready( )が呼び出されるので、その時点でモードがHS_STARTであればAdjustDiffJointValue( )が呼び出されることになる。その後で、ENABLE_ALL_SUBJECTマクロですべてのサブジェクトを使用可能とし、ASSERT_READY_TO_ALL_OBSERVERマクロでオブジェクトの準備完了をサブジェクトに通知[11]する。AdjustDiffJointValue( )の説明をする前に、DoStop( )とDoDestroy( )の説明をしておこう。

```
OStatus YESNO::DoStop(const OSystemEvent&
event){
    headState = HS_IDLE;

    DISABLE_ALL_SUBJECT;
    DEASSERT_READY_TO_ALL_OBSERVER;

    return oSUCCESS;
}
OStatus YESNO::DoDestroy(constOSystemEvent&
event){
    DELETE_ALL_SUBJECT_AND_OBSERVER;

    return oSUCCESS;
}
```

Prog.11 DoStop( )、DoDestroy( )

DoStop( )ではモードをHS_IDLE (何もしない状態) に切り替えて、DISABLE_ALL_SUBJECTマクロでサブジェクトを使用不可とし、DEASSERT_READY_TO_ALL_OBSERVER

[11] YESNOはダミーのオブザーバしか持たないので、実際には必要ない。

マクロでオブザーバに対応するサブジェクトに対して通信を行わないというメッセージ[12]を通知する。以上のDoStart()、DoStop()、DoDestroy()については、サンプルプログラムMovingHeadと全く同じものであり、DoInit()についてもSetMotorPower()を呼び出している以外は違わない。

さて、DoStart()で呼び出される関数AdjustDiffJointValue()である。

```
MovingResult YESNO::AdjustDiffJointValue(){
    OJointValue current[NUM_JOINTS];

    for (int i = 0; i < NUM_JOINTS; i++) {
        OPENR::GetJointValue(jointID[i],
        &current[i]);
        SetJointValue(region[0], i,
            degrees(current[i].value/
            1000000.0),degrees(current[i].
            value/1000000.0));
    }

    subject[sbjMove]->SetData(region[0]);
    subject[sbjMove]->NotifyObservers();

    return MOVING_FINISH;
}
```

Prog.12 AdjustDiffJointValue( )

ここでは、GetJointValue( )で各関節の角度を読み込み、その値をSetJointValue( )を利用してregion[0]に設定し、SetDataで共有メモリに設定した上で、そのことをNotifyObserversによってオブザーバであるOVirtualRobotCommに通知している。AdjustDiffJointValue( )が呼び出されるのは初期状態なので、region[0]、region[1]の両者とも使用されておらず、どちらを使っても構わない。ここで行っていることは、関節角度の現在値を読み取り、それをそのまま指示値としてAIBOに命令しているわけで、当然のことながら外見的な動作は何も行われない。AIBOの制御系はPID制御であるため、初期の指示値に現在値と異なる値が与えられた場合、PIDゲインを設定したとたんに過大な初期応答が発生する可能性がある。AdjustDiffJointValue( )はそれを避けるための関数である。なお、GetJointValue( )が読み出す角度はマイクロラジアン単位である。この関数から呼び出されるSetJointValue( )には、第一引数に目標値を設定するデータ、第二引数に目標値を設定する関節の番号、第三引数に目標角度の初期値、第四引数に目標角度の最終値を与えるが、SetJointValue( )に与える角度は度単位であるためdegrees( )を利用して単位の変換を行っている。

```
void YESNO::SetJointValue(RCRegion* rgn,
int idx, double start, double end){
    OCommandVectorData*cmdVecData=
    (OCommandVectorData*)rgn->Base();
    OCommandInfo* info=cmdVecData-
    >GetInfo(idx);

    info>Set(odataJOINT_COMMAND2,
    jointID[idx], ocommandMAX_FRAMES);

    OCommandData* data = cmdVecData-
    >GetData(idx);
    OJointCommandValue2* jval =
    (OJointCommandValue2*)data->value;

    double delta = end - start;

    for (int i = 0; i < ocommandMAX_FRAMES;
    i++) {
        double dval = start + (delta * i) /
        (double)ocommandMAX_FRAMES;
        jval[i].value = oradians(dval);
    }
}
```

Prog.13 SetJointValue( )

SerJointValue( )では、rgn->Base( )として使用するregion[]に対応するOCommandVectorDataを取り出す。OCommandVectorDataには、データタイプ、CPCプリミティブID、OCommadDataのフレーム数を設定するOCommandInfoと、フレーム数分のデータを設定するOCommandDataが含まれているが、まずGetInfo( )でOCommandInfoを取り出し、コマンドの種類にodataJOINT_COMMAND2、CPCプリミティブIDにjointID[]、フレーム数にocommandMAX_FRAMES（値は16）を設定する。次にGetData()でOCommandDataを取り出し、実際に値を入れる配列部分を取り出して、それをOJointCommandValue2型の配列にキャストする。これは64ビットの配列データとなっているOCommandDataに対して、実際に設定する角度データが32ビットであることによる。目標値はこうして取り出したjval[].valueについてjval[0]からjval[15]まで16個設定する。AIBO側ではこのjval[].valueについてjval[0].valueから順に8m秒ごとに処理していくことになる。与える値は、度単位の初期値から最終値までの差を16等分したものとする。すなわち、初期値が4.8度で最終値が0度の場合には、jval[0]に4.8度、jval[1]に4.5度、jval[2]に4.2度というように設定していく。なお、

[12] YESNOはダミーのオブザーバしか持たないので、これも実際には不要である。

実際の指示値はマイクロラジアン単位になるため、oradians()によって単位を変換している。

さてAdjustDiffJointValue()でNotifyObservers()が実行されると、データはOVirtualRobotCommで受信され、それに基づく動作（このYESNOでは首の移動）が行われる。OVirtualRobotCommではコマンドの実行が開始され、次のコマンドを受け入れる準備が整うとAssertイベントが発生し、それによってYESNOのReady()が呼び出される。

```
void YESNO::Ready(const OReadyEvent&
event){
    if (headState == HS_IDLE) {
        ;
    }
    else if (headState == HS_START) {
        AdjustDiffJointValue();
        headState = HS_ADJUSTING_DIFF_JOINT_VALUE;
    }
    else if (headState ==
    HS_ADJUSTING_DIFF_JOINT_VALUE) {
        SetJointGain();
        MoveToZeroPos();
        headState = HS_MOVING_TO_ZERO_POS;
    }
    else if (headState ==
    HS_MOVING_TO_ZERO_POS) {
        MovingResult r = MoveToZeroPos();
        if (r == MOVING_FINISH) {
            headState = HS_WAIT;
        }
    }
    else if (headState == HS_WAIT){
        OSYSPRINT(("command(y/n/s) -> "));
        char c[2];
        scanf("%s",c);
        if(c[0]=='y') headState =
        HS_SWING_HEAD_YES;
        if(c[0]=='n') headState =
        HS_SWING_HEAD_NO;
        if(c[0]=='s') headState = HS_FINISH;
        RCRegion* rgn = FindFreeRegion();
        SetJointValue(rgn, TILT_INDEX, 0.0, 0.0);
        SetJointValue(rgn, PAN_INDEX,  0.0, 0.0);
        SetJointValue(rgn, ROLL_INDEX, 0.0, 0.0);
        subject[sbjMove]->SetData(rgn);
        subject[sbjMove]->NotifyObservers();
    }
    else if (headState == HS_SWING_HEAD_YES ||
    headState == HS_SWING_HEAD_NO){
        MovingResult r = SwingHead();
        if (r == MOVING_FINISH) {
            headState = HS_WAIT;
        }
    }
    else if (headState == HS_FINISH){
        OSYSPRINT(("SHUTDOWN ...¥n"));
        OBootCondition bootCond(obcbPAUSE_SW);
        OPENR::Shutdown(bootCond);
    }
}
```

Prog.14 Ready()

Ready()はYESNOの動きを管理する働きをしており、各時点のAIBOの動作モードheadStateに応じて、必要な関数を呼び出し、場合によってはモードの変更を行うことになる。まずHS_IDLEの場合に何もしない。HS_STARTの場合にはAdjustDiffJointValue()を呼び出すことになるが、この場所へ到達したということはOVirtualRobotCommでAssertイベントが発生したということなので、DoInit()のようにIsReady()のチェックを行う必要はない。AdjustDiffJointValue()がDoInit()で行われたにせよ、HS_STARTで行われたにせよ、モードはHS_ADJUSTING_DIFF_JOINT_VALUEに変更される。AdjustDiffJointValue()の処理がOVirtualRobotCommで行われてASSERTイベントが発生すれば、次はHS_ADJUSTING_DIFF_JOINT_VALUEに対応するSetJointGain()が実行される。

```
void YESNO::SetJointGain(){
    OPENR::EnableJointGain(jointID[TILT_INDEX]);
    OPENR::SetJointGain(jointID[TILT_INDEX],
    TILT_PGAIN,TILT_IGAIN,TILT_DGAIN,
        PSHIFT, ISHIFT, DSHIFT);
    OPENR::EnableJointGain(jointID[PAN_INDEX]);
    OPENR::SetJointGain(jointID[PAN_INDEX],
    PAN_PGAIN,PAN_IGAIN,PAN_DGAIN,
        PSHIFT, ISHIFT, DSHIFT);
    OPENR::EnableJointGain(jointID[ROLL_INDEX]);
    OPENR::SetJointGain(jointID[ROLL_INDEX],
    ROLL_PGAIN,ROLL_IGAIN,ROLL_DGAIN,
        PSHIFT, ISHIFT, DSHIFT);
}
```

Prog.15 SetJointGain()

これによって、各関節のPIDゲインがヘッダファイルで定められた値に設定される。SetJointGain()の次には、首位置を原点に移動する関数MoveToZeroPos()が呼び出され、モードはHS_MOVING_TO_ZERO_POSに変更される。動作モードはこれまでNotifyObservers()を1回呼び

出すごとに変更されてきたが、MoveToZeroPos( )はZERO_POS_MAX_COUNTER回呼び出されるため、その間モードはHS_MOVING_TO_ZERO_POSのまま維持される。

```
MovingResult YESNO::MoveToZeroPos(){
    static int counter = -1;
    static double s_tilt, d_tilt;
    static double s_pan,  d_pan;
    static double s_roll, d_roll;

    if (counter == -1) {
        OJointValue current;
        OPENR::GetJointValue(jointID[TILT_INDEX],
        &current);
        s_tilt = degrees(current.value/1000000.0);
        d_tilt=(0.0-s_tilt)/(double)ZERO_
        POS_MAX_COUNTER;
        OPENR::GetJointValue(jointID[PAN_INDEX],
        &current);
        s_pan = degrees(current.value/1000000.0);
        d_pan = (0.0 - s_pan) /
        (double)ZERO_POS_MAX_COUNTER;
        OPENR::GetJointValue(jointID[ROLL_INDEX],
        &current);
        s_roll = degrees(current.value/1000000.0);
        d_roll = (0.0 - s_roll)/(double)ZERO_
        POS_MAX_COUNTER;
        counter = 0;
    }
    RCRegion* rgn = FindFreeRegion();

    SetJointValue(rgn, TILT_INDEX, s_tilt,
    s_tilt + d_tilt);
    SetJointValue(rgn, PAN_INDEX,  s_pan,
    s_pan  + d_pan);
    SetJointValue(rgn, ROLL_INDEX, s_roll,
    s_roll + d_roll);

    subject[sbjMove]->SetData(rgn);
    subject[sbjMove]->NotifyObservers();

    s_tilt += d_tilt;
    s_pan  += d_pan;
    s_roll += d_roll;

    counter++;
    return (counter == ZERO_POS_MAX_COUNTER) ?
    MOVING_FINISH : MOVING_CONT;
}
```

Prog.16 MoveToZeroPos( )

この関数ではstatic変数counterを使い、ZERO_POS_MAX_COUNTER(=16)回データを送信する。一つのデータは実際にはocommandMAX_FRAMES(=16)個のデータを含んでいるので、結局のところ256ステップ(2048m秒)を経て首の位置を初期値から原点へと移動させていることになる。最初にGetJointValue( )で関節の角度の初期値を得て、目標である0との差を16分割し、d_tilt、d_pan、d_rollに与える。そして、初期値と最終値をその分だけずらしながらSetJointValue( )を呼び出し、SetData( )でデータを書き込み、NotifyObservers( )でそのことをOVirtualRobotCommに通知する。このことによりcounterがZERO_POS_MAX_COUNTERに達した時点で首は原点に移動しているはずであり、そのときにはMOVING_FINISH、それ以外ではMOVING_CONTを返すようになっている。ここで呼ばれている関数

```
RCRegion* YESNO::FindFreeRegion(){
    if (region[0]->NumberOfReference() == 1)
    return region[0];
    else return region[1];
}
```

Prog.17 FindFreeRegion( )

は、現在OVirtualRobotCommで使用されていないregion[]を返す関数で、region[]が使用中でなければNumberOfReference( )が1、使用中ならば2となっていることを利用している。

MoveToZeroPos( )の実行中は、モードはHS_MOVING_TO_ZERO_POSのままであるから、ASSERTイベントが発生しReady( )が呼ばれるごとにMoveToZeroPos( )が実行されるが、ZERO_POS_MAX_COUNTER回呼ばれてMOVING_FINISHが返されると、モードはHS_WAITに変更される。

HS_WAITではOSYSPRINT( )によって画面にcommand->と表示され、ユーザからの入力待ちになる。ユーザのキーボードからの入力に対し、'y'についてはHS_SWING_YES、'n'についてはHS_SWING_NO、's'についてはHS_FINISHとモードが変更される。HS_WAITモードでは首の位置は原点にあるが、OVirtualRobotCommにASSERTイベントを発生させReady( )を呼び出させるために、ダミーで目標値を原点に設定してNotifyObservers( )を実行している。

モードがHS_SWING_HEAD_YESもしくはHS_SWING_

HEAD_NOの場合には、SwingHead()が呼び出される。

```
MovingResult YESNO::SwingHead(){
    static int phase = 0;
    static int prevphase;
    static int counter = 0;

    if(headState == HS_SWING_HEAD_YES){
    if(counter == 32)  counter = 0;
    if(counter < 16){
        counter = counter + 1;
        prevphase = phase;
        phase = phase - 3;
    }
    else if(16 <= counter && counter < 32){
        counter = counter + 1;
        prevphase = phase;
        phase = phase + 3;
    }

    RCRegion* rgn = FindFreeRegion();

    SetJointValue(rgn, TILT_INDEX,
    (double)prevphase,(double)phase);
    SetJointValue(rgn, PAN_INDEX,  0.0, 0.0);
    SetJointValue(rgn, ROLL_INDEX, 0.0, 0.0);
    subject[sbjMove]->SetData(rgn);
    subject[sbjMove]->NotifyObservers();
}

if(headState == HS_SWING_HEAD_NO){
    if(counter == 32)  counter = 0;
    if(counter < 8){
        counter = counter + 1;
        prevphase = phase;
        phase = phase + 4;
    }
    else if(8 <= counter && counter < 24){
        counter = counter + 1;
        prevphase = phase;
        phase = phase - 4;
    }
    else if(24 <= counter && counter < 32){
        counter = counter + 1;
        prevphase = phase;
        phase = phase + 4;
    }

    RCRegion* rgn = FindFreeRegion();
    SetJointValue(rgn, TILT_INDEX, 0.0, 0.0);
    SetJointValue(rgn, PAN_INDEX,
    (double)prevphase,(double)phase);
    SetJointValue(rgn, ROLL_INDEX, 0.0, 0.0);
    subject[sbjMove]->SetData(rgn);
        subject[sbjMove]->NotifyObservers();
    }
    return (counter == 32) ? MOVING_FINISH :
    MOVING_CONT;
}
```

Prog.18 SwingHead( )

SwingHead( )は、モードがHS_SWING_HEAD_YESの場合にはAIBOに首を縦に振らせてYESという動作を、HS_SWING_HEAD_NOの場合には首を横に振らせてNOという動作をさせる関数であり、一連の動作はいずれも32回の呼び出しによって完了する。HS_SWING_HEAD_YESの場合には、TILT方向の角度を0度から－48度まで3度ずつ減らしながらSetJointValue( )、SetData( )、NotifyObservers()を呼び出し、それからTILT方向の角度を0度まで3度ずつ増やしていく。HS_SWING_HEAD_NOの場合には、PAN方向の角度を0度から32度まで4度ずつ増やし、次に－32度まで4度ずつ減らし、最後に0度まで4度ずつ増やしていく。SwingHead()が一連の動作を終えてMOVING_FINISHを返すと、モードは再びHS_WAITに変更される。

HS_WAITでユーザが‘s’を入力するとモードはHS_FINISHに変更される。この状態でReady()が呼ばれると画面にSHUTDOWN... と表示し、Shutdown()によってAIBOの電源が落とされる。

以上で、ソースは完成である。Prog.5からProg.18までをYESNO.ccという一つのファイルにまとめて保存する。

6．Makeファイルとプログラムの実行

コンパイルにはgccを使うことになるが、手順が複雑なためMakeファイルを使うことになる。Makeファイルについてはサンプルに含まれているものをベースに若干の変更を行えばよいが、本アプリケーションの場合には以下のようになる。

```
OPENRSDK_ROOT?=/usr/local/OPEN_R_SDK
INSTALLDIR=../MS
CXX=$(OPENRSDK_ROOT)/bin/mipsel-linux-g++
STRIP=$(OPENRSDK_ROOT)/bin/mipsel-linux-
strip
MKBIN=$(OPENRSDK_ROOT)/OPEN_R/bin/mkbin
STUBGEN=$(OPENRSDK_ROOT)/OPEN_R/bin/
stubgen2
```

```
MKBINFLAGS=-p $(OPENRSDK_ROOT)
LIBS=-L$(OPENRSDK_ROOT)/OPEN_R/lib
-lObjectComm -lOPENR
CXXFLAGS= ¥
    -O2 ¥
    -g ¥
    -I. ¥
    -I$(OPENRSDK_ROOT)/OPEN_R/include/R4000 ¥
    -I$(OPENRSDK_ROOT)/OPEN_R/include/MCOOP ¥
    -I$(OPENRSDK_ROOT)/OPEN_R/include

.PHONY: all install clean

all: YESNO.bin

%.o: %.cc
    $(CXX) $(CXXFLAGS) -o $@ -c $^

YESNOStub.cc: stub.cfg
    $(STUBGEN) stub.cfg

YESNO.bin: YESNOStub.o YESNO.o YESNO.ocf
    $(MKBIN) $(MKBINFLAGS) -o $@ $^ $(LIBS)
    $(STRIP) $@

install: YESNO.bin
    gzip -c YESNO.bin > $(INSTALLDIR)/OPEN-
    R/MW/OBJS/YESNO.BIN

clean:
    rm -f *.o *.bin *.elf *.snap.cc
    rm -f YESNOStub.h YESNOStub.cc def.h
    entry.h
    rm -f $(INSTALLDIR)/OPEN-R/MW/OBJS/
    YESNO.BIN
```

Prog.19 Makefile

別のアプリケーションを作成する場合には、このMakeファイルの中のYESNOという文字[13]をすべて自分のアプリケーションに合わせて変更すればよい。上のファイルをMakefileという名称で、stub.cfg、YESNO.h、YESNO.ccと同じディレクトリ（仮にYESNOという名称にする）に保存しておく。

コンパイルは、YESNOディレクトリにおいてコマンドラインからmake allと入力すればよい。コンパイルが終わったらmake installとすると、ディレクトリYESNOと同じ場所にMSというディレクトリが作られる。MSの下にはOPEN-Rというディレクトリがあるので、これをディレクトリごとAIBOメモリスティックにコピーしておく。なお、make cleanとするとstub2genが自動生成したYESNOStub.h YESNOStub.cc def.h entry.hなどのファイルやスタックサイズなどが記述されているYESNO.ocf、mkbinが自動生成するYESNO.snap.cc YESNO.snap.elf YESNO.nosnap.elf YESNO.rel.elf YESNO.o YESNO.snap.o YESNO.stub.onなどの中間ファイル、最終的に生成される実行可能ファイルYESNO.binが消去される。

次にサンプルプログラムのPowerMonディレクトリにコンパイル済みの実行ファイルPOWERMON.BINがある[14]ので、それもAIBOメモリスティックの/OPEN-R/MW/OBJSフォルダにコピーしておく。

最後に以下のようなテキストファイルOBJECT.CFGを作成し

```
/MS/OPEN-R/MW/OBJS/POWERMON.BIN
/MS/OPEN-R/MW/OBJS/YESNO.BIN
```

Prog.20 OBJECT.CFG

AIBOメモリスティックの/OPEN-R/MW/CONFフォルダに保存する。以上でメモリスティックへのファイルの保存は完了である。

実行はAIBO本体にメモリスティックを装着した上で、本体のポーズスイッチを押す。しばらく待ってリモートコンピュータのコマンドプロンプトから

```
telnet 192.168.11.24 59000
```

とAIBOのポート59000番にtelnetをかけると、AIBOの状態を示すメッセージが画面上に表示される。AIBOは首を原点に移動した上で、画面にcommand->と表示し入力待ちになるので、'y'と入力すれば首を前後方法、'n'と入力すれば左右方向に動かした上で、再び入力待ちになる。終了させたい場合には、's'を入力すればよい。

7. おわりに

本稿では、動物型ロボットであるAIBOのプログラミングについて、環境構築から、プログラムの作成法、実行法などについて述べた。

さて、本稿が掲載されるネットワーク&インフォメーション第8号は、故高津信三教授の追悼特集号である。実は筆者がAIBOに取り組んだきっかけは、高津先生のアドバイスであった。高津先生のご専門は経営システム工学であっ

[13] もしNOYES.ccからアプリケーションを作成するのなら、たとえば2行目については
rm -f NOYESStub.h NOYESStub.cc def.h enrty.h
とすればよい。

[14] Makefileもあるのでコンパイルして作ってもよい。

たが、ハードウェアを好まれパソコンの自作などにも古くから取り組まれていた。「これからのネットワーク情報学部の学生は、経営・経済系のプログラミングだけでなく、もっとハードウェア寄りのプログラミングを経験する機会があるべきだ」というお考えをお持ちで、「SONYのAIBOを少し調べてみたらどうか」というご教唆がきっかけなのである。本稿を読んでこの種のプログラミングに興味を抱いてくれる学生がいたならば、きっと高津先生も喜ばれることと思う。

最後に少しだけ、高津先生の思い出を記しておこう。ネットワーク情報学部の設立は、筆者が経営学部選出の委員として「新・21世紀構想会議」に提出した「新学部設置に関する意見書」を契機としている。意見書を提出したのは、平成11年の7月であったが、その内容はそれに先立つ3月に、筆者の研究室において高津先生、齋藤雄志先生、佐藤創先生らと相談をした結果に基づいて作成されたものであった。その後、高津先生は、当時の出牛正芳学長を委員長とする「新学部設置委員会」の委員長代理として、学部設置の実質的な中心として尽力され、学長の指名により初代ネットワーク情報学部長に就任されたわけである。典型的な学者肌であった先生は、学内政治には全く興味を持たれていなかった方であったので、学部長就任を命ぜられたときの当惑された表情は今でもはっきりと記憶に残っている。しかしながら、高津先生はリーダーとして傑出した才を持たれていた。物事を決断する前には周囲の意見に真摯に耳を傾け、一旦決断するとどのような抵抗があろうとも決してぶれることがなかった。それは自分のプライドを守るためなどではなく、何よりもこれからの大学のあり方を考え、そして学部の利益を考えてのことであった。大学においては新しいことを始めようとすれば、必ず反対意見がでる。そして、その意見にもそれなりの理があることは多い。しかし、それに屈していたのでは何も変えることはできない。高津先生のリーダーシップがなければ、ネットワーク情報学部が出来ることはなかったろう。同時に筆者は、もし新学部設置に関わらなければ、高津先生が病に倒れられることはなかったのではないかとの思いを禁じえない。

最後にお会いしたのは、昨年の2月だった。病気療養のため学部長を辞任され、非常事態ということで、ご高齢にもかかわらず後任を快く引き受けてくださった坂本實先生にご挨拶に来られたときである。学部設置後は満足なお手伝いも出来ず、ご負担をお掛けしたことをお詫びする筆者に、「いやあ、今度のことは仕事とは関係ありませんよ」と微笑んで下さったが、元々細身であられた先生のズボンの隙間から出ていた足が、女性の手首のようになっておられたのを見て一瞬言葉を失った。お見舞いに伺いたいと思いながらも、とうとう最後までその勇気が持てなかった。最も尊敬する大学人の一人であった先生を失ったことを、心から残念に思う。

参考文献

1) OPEN-Rオフィシャルウェブサイト, http://openr.aibo.com
2) OPEN-R SDKインストールガイド, ソニー株式会社
3) OPEN-R SDKプログラマーズガイド, ソニー株式会社
4) OPEN-R SDKレベル2 リファレンスガイド, ソニー株式会社
5) OPEN-R SDK機種情報 ERS-220, ソニー株式会社
6) C++でAIBOを自在に動かす, OPEN-Rプログラミング Special Interest Group, インプレス