Carnegie Mellon Motion Object (CMPack'04)

Manuale d'uso

Indice generale

1 Premessa....1
2 Introduzione....2
3 Utilizzo....2
- 3.1 Struttura dei Sorgenti....2
- 3.2 Servizi offerti....3
- 3.3 Il Sistema di Riferimento....4
- 3.4 I Comandi (MotionCommand)....5
- 3.5 Creare i propri parametri....14
- 3.6 Esempio di Utilizzo....15

ATimer in OPENR....23

1 Premessa

CMPack'04 è il codice della Carnegie Mellon University utilizzato dalla loro squadra di Aibo per partecipare alla RobotCup 2004. Tutto il codice è distribuito sotto le condizioni della GNU Lesser GPL license; il codice completo può essere scaricato dall'indirizzo http://www-2.cs.cmu.edu/~robosoccer/legged/.

Il sorgente è composto da vari oggetti Open-R con compiti precisi (gestione della Telecamera, gestione dei behavior, main object...) tra questi c'è MotionObject, l'oggetto Open-R per la gestione del movimento delle zampe, della testa, della coda e altri funzioni aggiuntive per gestire i LED e l'audio. MotionObject è considerato tra gli sviluppatori di codice per Aibo uno dei migliori oggetti per la gestione del movimento attualmente disponibile.

Questo documento intende spiegare come utilizzare MotionObject nei propri oggetti Open-R.

2 Introduzione

MotionObject offre una gestione pressoché completa dei joint degli Aibo, permettendo di gestire le zampe, la coda e la testa con semplici comandi spedibili attraverso la comunicazione inter-object standard di Open-R. E' considerato un oggetto molto avanzato, utilizza la cinematica inversa per la gestione delle zampe e della testa ed utilizza algoritmi genetici per la ottimizzazione dei movimenti (gli algoritmi genetici vengono utilizzati nella fase di creazione dei movimenti, vengono quindi applicati una sola volta prima di memorizzare il movimento nella memory stick dell'Aibo).

3 Utilizzo

In questo documento verrà utilizzato un oggetto chiamato ProvaCM di esempio per spiegare in maniera più semplice come utilizzare MotionObject.

3.1 Struttura dei Sorgenti

La struttura del codice (compreso ProvaCM) è la seguente:

```
+->headers
+->shared_code
+->Motion
    +->genmot
+->MS
    +->motion
        +->ers210
        +->ers7
    +->sounds
    +->OPEN-R
        +->MW
            +->CONF
            +->OBJS
+->ProvaCM
+->SharedMemMgr
```

In [headers] sono presenti gli header condivisi utilizzati dai vari oggetti di CMPack; attualmente in questa directory sono presenti solo gli header necessari a MotionObject e all'oggetto Open-R SharedMemMgr necessario a MotionObject.

In [shared_code] sono contenuti i file sorgente di alcune classi utilizzate dai vari oggetti di CMPack, come nel caso della directory [header] nella versione attuale sono presenti solo i file “.cc” utilizzati da

MotionObject e da SharedMemMgr.

In [Motion] sono presenti i sorgenti di MotionObject ed è presente anche la sotto directory [genmot] che contiene i sorgenti di una utility, genmot, per la creazione di movimenti personalizzati.

In [MS] è presente la struttura dei file da copiare nella memory stick; MotionObject utilizza una struttura in parte diversa dalla standard, nella root della memory stick oltre alla directory OPEN-R, che rispetta la struttura standard definita dalla Sony, è presente anche la directory [motion] che contiene i file di dati necessari a MotionObject, più precisamente in [ers210] sono presenti i file con i movimenti per gli Aibo modello ERS 210, mentre in [ers7] sono contenuti i movimenti per gli Aibo modello ERS 7.

Nella sottodirectory [sound] sono presenti alcuni file audio che è possibile utilizzare attraverso MotionObject.

In [ProvaCM] è contenuto il sorgente dell'oggetto Open-R di esempio ProvaCM.

3.2 Servizi offerti

MotionObject offre i seguenti servizi:

```
Service : "MotionObject.MoveJoint.OCommandVectorData.S"
Service : "MotionObject.MotionComplete.MovementInfo.S"
Service : "MotionObject.Speaker.OSoundVectorData.S"
Service : "MotionObject.VelParam.VelParam.S"
Service : "MotionObject.Control.MotionCommand.O"
Service : "MotionObject.Params.WalkParam.O"
Service : "MotionObject.RegisterRegion.SMMSharedMemRegion.S"
Service : "MotionObject.SensorFrame.OSensorFrameVectorData.O"
```

Tra i vari servizi offerti da MotionObject uno è di particolare interesse: **Control** che ha il compito di ricevere i comandi (**MotionCommand**) da parte dell'oggetto Open-R subject che intende far eseguire un movimento all'Aibo.
Esiste poi il servizio MotionComplete, il nome suggerisce sia un servizio per informare un oggetto Open-R del completamento dell'esecuzione di un'azione, ma analizzando il sorgente di MotionObject ci si accorge che tale servizio non viene utilizzato, il sorgente contiene solo un implementazione dummy delle funzioni necessarie.
I servizi **MoveJoint**, **Speaker** e **SensorFrame** collegano MotionObject con OVirtualRobotComm per la gestione rispettivamente dei joint, dello speaker e dei sensori. **RegisterRegion** collega MotionObject con l'oggetto Open-R SharedMemMgr per la gestione delle zone di memoria condivisa.
VelParam è un servizio che permette di spedire i parametri di velocità dell'Aibo, la struttura VelParam è contenuta nel file WalkParam.h (nella home di MotionObject) e contiene tre double che

corrispondono alla velocità massima lungo l'asse X, Y e la velocità radiale A (vedi oltre per il significato degli assi).
Infine WalkParam è l'altro servizio utile ai fini dell'utilizzo di MotionObject da parte di un proprio oggetto Open-R; permette di impostare i parametri di movimento degli Aibo (vedi creare i propri parametri).

3.3 Il Sistema di Riferimento

Prima di spiegare come creare e impostare le varie opzioni di comando per MotionObject è opportuno specificare gli assi di riferimento degli Aibo, come illustrato nell'immagine seguente.

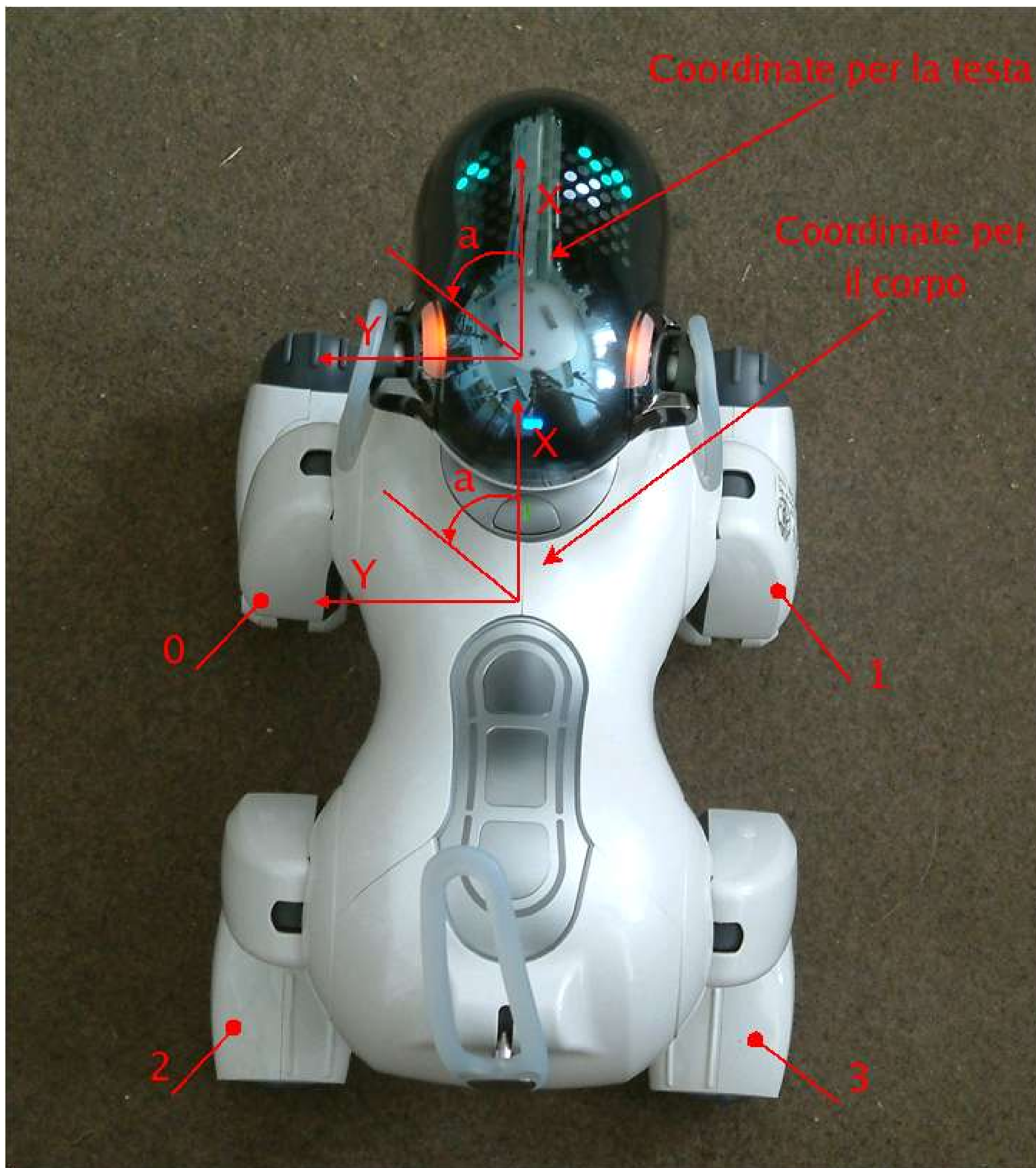

Illustrazione 1 Sistemi di riferimento

MotionObject considera due sistemi di riferimento: uno per la gestione della telecamera, l'altro per la gestione del corpo quindi della camminata. Oltre a specificare le due assi **X** e **Y** specifica anche la velocità angolare **a** per lo spostamento dall'asse X all'asse Y. Nell'immagine sono stati specificati anche i numeri identificativi delle zampe, questi numeri saranno necessari per specificare i parametri di movimento degli Aibo (vedi Creare i propri parametri).

3.4 I Comandi (MotionCommand)

I comandi da spedire a MotionObject per la gestione del movimento sono degli oggetti di tipo

MotionCommand. Nel header file MotionInterface.h è specificata la struttura degli oggetti MotionCommand e i possibili valori assegnabili ai vari attributi; di seguito sono riportati gli attributi di un oggetto MotionCommand:

```
class MotionCommand{
public:
  short motion_cmd;
  short head_cmd;
  short tail_cmd;
  short bound_mode;

  double vx,vy,va;

  vector3d head_lookat;
  double head_tilt_offset;
  double head_tilt;
  double head_pan;
  double head_roll;
  double head_tilt2;

  short tail_pan;
  short tail_tilt;
  short mouth;

  unsigned short sound_cmd;
  unsigned short sound_frequency;
  unsigned long  sound_duration;
  unsigned short sound_file;

  LEDState led;
```

Il campo **motion_cmd** specifica il tipo di comando di movimento da eseguire, è importante a questo punto precisare che MotionObject gestisce solo la camminata sui gomiti, stile di camminata preferita per la robotcup in quanto permette una maggior precisione nei movimenti e una maggiore stabilità del robot; nel file MotionInterface.h sono specificati i possibili valori che può assumere e sono:

```
// Standing States
const int MOTION_STAND_NEUTRAL     =  100;
const int MOTION_STAND_LEFT        =  101;
const int MOTION_STAND_RIGHT       =  102;
const int MOTION_STAND_CROUCH      =  103;
const int MOTION_STAND_SKATEBOARD  =  104;
```

```
const int MOTION_STAND_WALL_TURN_L =  105;
const int MOTION_STAND_WALL_TURN_R =  106;

// Walking States
const int MOTION_WALK_TROT        =  200;
const int MOTION_WALK_TROT_FAST   =  201;
const int MOTION_WALK_DRIBBLE     =  210;
const int MOTION_WALK_BEH         =  211; // walk specified by
                                            behaviors

// Kicking States
const int MOTION_KICK_DIVE         =  300;
const int MOTION_KICK_BUMP         =  301;
const int MOTION_KICK_FOREWARD     =  302;
const int MOTION_KICK_HEAD_L       =  303;
const int MOTION_KICK_HEAD_R       =  304;
const int MOTION_KICK_HEAD_SOFT_L =  305;
const int MOTION_KICK_HEAD_SOFT_R =  306;
const int MOTION_KICK_HEAD_HARD_L =  307;
const int MOTION_KICK_HEAD_HARD_R =  308;
const int MOTION_KICK_HOLD         =  309;
const int MOTION_KICK_SWING_L      =  310;
const int MOTION_KICK_SWING_R      =  311;
const int MOTION_KICK_ARM_DIVE_L   =  312;
const int MOTION_KICK_ARM_DIVE_R   =  313;
const int MOTION_KICK_SLAP_BACK_L =  314;
const int MOTION_KICK_SLAP_BACK_R =  315;

const int MOTION_BLOCK_GOALIE_L    =  316;
const int MOTION_BLOCK_GOALIE_R    =  317;
const int MOTION_BLOCK_GOALIE_C    =  318;
const int MOTION_BLOCK_SUPP        =  319;
const int MOTION_BLOCK_SUPP_L      =  320;
const int MOTION_BLOCK_SUPP_R      =  321;

const int MOTION_GOAL_HAPPY1       =  322;
const int MOTION_GOAL_HAPPY2       =  323;
const int MOTION_GOAL_HAPPY3       =  324;
const int MOTION_GOAL_SAD          =  325;
```

```
//Black & White Ball Challenge
const int MOTION_KICK_BW_L        =  326;
const int MOTION_KICK_BW_R        =  327;
const int MOTION_BACKUP_BW        =  328;

// Fallen States
const int MOTION_GETUP_NEUTRAL =  400;
const int MOTION_GETUP_BACK    =  401;
const int MOTION_GETUP_FRONT   =  402;
const int MOTION_GETUP_LEFT    =  403;
const int MOTION_GETUP_RIGHT   =  404;

// Skating States
const int MOTION_SKATE =  500;

//Turn with ball
const int MOTION_TURN_WITH_BALL_L = 600;
const int MOTION_TURN_WITH_BALL_R = 601;
```

I vari comandi hanno nomi sufficientemente esplicativi, ma per chiarire meglio sono stati raggruppati per comandi simili. Gli *Standing state* sono gli stati "in piedi" dell'Aibo. I *walking state* sono gli stati riguardanti la camminata, mandare ad un Aibo uno di questi comandi mette in moto le zampe fino al ricevimento di un comandi di stand. I *kicking state* sono tutti i comandi per calciare la palla. I *Black & White ball challange* sono dei movimenti che richiamano Black & White di Michael Jackson, i due Kick sono dei calci particolari a sinistra e a destra (_L e _R) mentre il BACKUP_BW è una sorta di MoonWalk. I *Fallen State* sono dei comandi che permettono all'Aibo di rialzarsi nel caso si fosse ribaltato. Il comando di *Skating* MOTION_SKATE permette di far camminare l'Aibo muovendo solo le zampe posteriori tenendo le anteriori piegate quindi scivolando sulle giunture delle zampe anteriori. Gli utlimi *Turn with ball* non sembrano essere stati implementati.

Il campo **head_cmd** specifica il tipo di comando per la testa, i possibili valori sono:

```
// Head States
const int HEAD_NO_CMD           =  0;
const int HEAD_LOOKAT           =  1;
const int HEAD_ANGLES           =  2; // Don't use this unless
                                    you have to!
const int HEAD_SCAN_BALL        =  3;
const int HEAD_SCAN_BALL_NEAR   =  4;
const int HEAD_SCAN_BALL_TURN   =  5;
const int HEAD_SCAN_DRIBBLE     =  6;
```

```
const int HEAD_SCAN_MARKERS     =  7;
const int HEAD_SCAN_MARKERS_HIGH=  8;
const int HEAD_SCAN_POINT       =  9;
const int HEAD_SCAN_OBSTACLES   =  10;
const int NUM_HEAD_CMDS         =  11;
```

Il campo **tail_cmd** specifica il comando per la coda e i possibili valori sono:

```
// Tail States
const int TAIL_NO_CMD       =  0;
const int TAIL_AIM          =  1;
const int TAIL_FOLLOW       =  2;
const int NUM_TAIL_CMDS     =  3;
```

Il comando **TAIL_NO_CMD** riposiziona la coda in posizione centrale, mentre il comando **TAIL_AIM** permette di "puntare" la coda nella posizione specificata dai campi tail_pan e tail_tilt spiegati più sotto. Il comando **TAIL_FOLLOW** sembra non essere stato implementato.

E' da precisare che è possibile far eseguire a MotionObject in simultanea tutti e tre i tipi di comando, quindi è possibile, per esempio, far camminare il cane (MOTION_WALK_TROT), muovendo la testa alla ricerca della palla (HEAD_SCAN_BALL) mentre viene puntata la coda in una data posizione (TAIL_AIM).

I possibili valori per **bound_mode** sono:

```
// Bounding modes
const int BOUND_GOTO_EGO = 0;
const int BOUND_NONE     = 1;
const int BOUND_SPEED    = 2;
```

I tre attributi **vx**, **vy** e **va** sono di fondamentale importanza e specificano la velocità di movimento dell'Aibo. In base al sistema di riferimento riportato in figura 1, **vx** specifica la velocità in mm/s lungo l'asse **X**, **vy** lungo l'asse **Y** e **va** specifica la velocità angolare in radianti/s **a**. Questa tecnica per la gestione del movimento permette di specificare con molta precisione la posizione che si vuole far raggiungere all'Aibo specificando le velocità lungo le assi di riferimento e tenendo in considerazione la durata del movimento, per far questo risultano molto utili i **Timer** di Open-R; la loro struttura e il loro utilizzo viene spiegato nell'*Appendice A* di questo documento.
Un altro campo molto utile è **head_lookat** di tipo vector3d. Vector3d, come si può intuire dal nome, è un vettore a 3 dimensioni ed è definito nell'header Geometry.h (presente negli header condivisi), il suo costruttore *vector3d(double x, double y, double z)* permette di specificare il punto lungo gli assi x,y,z che si vuole considerare. Il campo head_lookat permette quindi di specificare un punto nello spazio a 3

dimensione in cui si vuole “puntare” la testa dell'Aibo, tale valore viene preso in considerazione quando head_cmd vale HEAD_LOOKAT.
Il vector3d head_lookat viene elaborato anche nel caso di un comando HEAD_SCAN_POINT. Tale comando posiziona la testa dell'Aibo rivolta verso il terreno in cerca di punti o linee; in questo caso gli assi rispecchiano il sistema di riferimento in figura 1 con l'asse Z uscente dal piano, con l'accortezza di incrementare l'angolo di scansione al crescere della coordinata X.
I campi successivi **head_tilt**, **head_pan**, **head_roll** e **head_tilt2** se utilizzati assieme al comando HEAD_ANGLES permettono di specificare direttamente gli angoli da passare a OvirtualRobotComm per il relativo joint. Nel modello ERS 7 per la gestione della testa sono presenti i joint **tilt**, **pan** e **tilt2**, il joint roll non è presente quindi il suo valore viene ignorato.
Come anticipato in precedenza i due campi **tail_pan** e t**ail_tilt** permettono di impostare i due gradi di libertà pan e tail della coda; stando alle specifiche della Sony il modello ERS7 ha una range per quanto riguarda il pan di [-60, +60] mentre per il titl di [0, +65]. Tali valori vengono presi in considerazione se il campo tail_cmd è uguale a TAIL_AIM.
Il campo **mouth** permette di specificare, con un range da 0 a 100, quanto aprire la bocca dell'Aibo.
Dopo il campo mouth ci sono i campi necessari per la gestione dell'audio. Il campo **sound_cmd** puo' assumere i valori;

```
// Sound States
const int SOUND_NONE = 0;
const int SOUND_NOTE = 1;
const int SOUND_FILE = 2;
```

SOUND_NONE disattiva lo speaker; SOUND_NOTE suona una nota con frequenza e durata specificabile e SOUND_FILE suona un file predefinito.
Il campo **sound_frequency** specifica la frequenza della nota da suonare, mentre **sound_duration** specifica la durata; entrambi questi campi vengono considerati solo se **sound_cmd** vale SOUND_NOTE. Infine il campo **sound_file** specifica il file predefinito da suonare; i possibili valori sono:

```
// Sound Files
const int SOUND_FILE_TEST      = 0;
const int SOUND_FILE_SQUARE    = 1;
const int SOUND_FILE_RECTANGLE = 2;
const int SOUND_FILE_T         = 3;
const int SOUND_FILE_L         = 4;
const int SOUND_FILE_TRIANGLE  = 5;
const int SOUND_FILE_TADA      = 6;
```

Tale campo viene preso in considerazione solo quando **sound_cmd** vale SOUND_FILE.

L'ultimo campo di MotionCommand è **led** di tipo **LEDState** e permette di comandare l'accensione dei

led dell'Aibo. Le varie costanti che semplificano al gestione dei LED sono definite nel file LEDInterface.h dove viene definita la struttura LEDState:

```
struct LEDState{
  unsigned int cmd;
  int intensity[NUM_LEDS];
  int mode;
  int set;
};
```

Cmd contiene la maschera dei LED da accendere; non è possibile generalizzare, in linea di principio ogni LED viene identificato da un singolo bit, ma alcuni LED in base al colore con cui li si vuole accendere possono essere rappresentati da un bit o da un altro, per questo all'interno del header file LEDInterface.h sono specificate una lista di costanti che si possono utilizzare per creare una maschera in modo semplice effettuando un bitwise OR tra le costanti.

L'array **intensity** specifica l'intensità con cui accendere i LED; **mode** viene utilizzato solo per i led sulla faccia e specifica la modalità in cui accenderli (dalla documentazione Sony esistono i seguenti modi: MODE_A o MODE_B); **set** se posta a true attiva l'accensione dei LED specificati dalla struttura, se false la struttura non viene elaborata.

Prima di proseguire con le costanti presenti in LEDInterface.h e con le funzioni necessarie per impostare i LED è necessario precisare che in MotionObject i LED sono divisi in tre categorie:

1. **Bin LED**: sono i LED di cui è possibile specificare solo lo stato di acceso o spento, non è quindi possibile specificare ne l'intensità ne il modo. Questi sono i LED: Sulla testa (vicino al sensore di tatto), il led Blu sulla testa, i LED sulle orecchie.
2. **Back LED**: sono i LED posti sulla “schiena” dell'Aibo, è possibile specificare l'intensità.
3. **Face LED**: sono i LED sul muso degli Aibo; per questi LED è possibile specificare sia l'intensità sia il modo.

Le costanti definite in LEDInterface.h sono contenute nel namespace Motion e sono:

```
 const int LED_BRIGHT = 255;
 const int LED_DIM    = 100;
 const int LED_OFF    = 0;
```

Per impostare l'intensità dei LED.

```
enum LEDMode { MODE_A, MODE_B };
```

Definisce i due modi di accensione dei LED per il muso.

```
//binary LEDs on head
 const int LED_TOP_ORANGE = 1<<0;
 const int LED_TOP_WHITE  = 1<<1;
```

```
const int LED_TINY        = 1<<5;
const int LED_EARS_RED           = 1<<2;
const int LED_EARS_GREEN         = 1<<3;
const int LED_EARS_BLUE          = LED_EARS_GREEN;
const int LED_EARS_CYAN          = LED_EARS_GREEN;
const int LED_EARS_PURPLE        = LED_EARS_GREEN;
const int LED_EARS_YELLOW        = LED_EARS_GREEN;
const int LED_EARS_WHITE         = 1<<2 | 1<<3 | 1<<4;
```

Queste costanti identificano i vari **Bin LED**.

```
const int LED_BACK_FRONT_COLOR   = 1<<20;
const int LED_BACK_FRONT_WHITE   = 1<<21;
const int LED_BACK_MIDDLE_COLOR  = 1<<22;
const int LED_BACK_MIDDLE_WHITE  = 1<<23;
const int LED_BACK_REAR_COLOR    = 1<<24;
const int LED_BACK_REAR_WHITE    = 1<<25;
```

Queste costanti definiscono i vari **Back LED**.

```
const int LED_A         = 1<<6;
const int LED_B         = 1<<7;
const int LED_C         = 1<<8;
const int LED_D         = 1<<9;
const int LED_E         = 1<<10;
const int LED_F         = 1<<11;
const int LED_G         = 1<<12;
const int LED_H         = 1<<13;
const int LED_I         = 1<<14;
const int LED_J         = 1<<15;
const int LED_K         = 1<<16;
const int LED_L         = 1<<17;
const int LED_M         = 1<<18;
const int LED_N         = 1<<19;

const int LED_VLINE_RIGHT        = LED_A | LED_C | LED_G;
const int LED_VLINE_CENTER       = LED_K | LED_L;
const int LED_VLINE_LEFT         = LED_B | LED_D | LED_H;

const int LED_HLINE_HIGH         = LED_G | LED_I | LED_K |
                              LED_J | LED_H;
const int LED_HLINE_LOW          = LED_E | LED_L | LED_F;
```

```
const int LED_ZIG_ZAG            = LED_G | LED_E | LED_K |
                               LED_F | LED_H;

const int LED_WHITE = LED_A | LED_B | LED_E | LED_F |
                 LED_G | LED_H | LED_I | LED_J | LED_K
                 | LED_L;

//mode A *ONLY*
const int LED_RED_EYES_LOW       = 1<<18;
const int LED_RED_EYES_HIGH      = 1<<19;

//mode B *ONLY*
const int LED_2PURPLE            = LED_N;
const int LED_4PURPLE            = LED_N|LED_M;
const int LED_GREEN              = LED_A|LED_B|LED_C|LED_D;
const int LED_CYAN               = LED_M|LED_C|LED_D;

const int LED_GREEN_EYE_LEFT     = LED_A;
const int LED_GREEN_EYE_RIGHT    = LED_B;
const int LED_GREEN_EYES         = LED_A|LED_B;

const int LED_GREEN_PURPLE       = LED_D|LED_C|LED_N|LED_A|
                               LED_B;
const int LED_GREEN_CYAN         = LED_M|LED_A|LED_B|LED_C|
                               LED_D;
```

Infine queste costanti definiscono i vari **Face LED**; oltre ai LED singoli (LED_A, LED_B...) sono presenti delle configurazione preimpostate per i LED tipo LED_RED_EYES_LOW che accende due occhi rossi sul muso dell'Aibo.

Per impostare il campo led di MotionCommand sono disponibili alcune **funzioni** atte a semplificare il lavoro:

- `void setFaceLEDs(int mask, int intensity, int mode)`: imposta i LED sul muso specificando la maschera, l'intensità e la modalità
- `void setBinLEDs(int mask)`: imposta i Bin LED specificando la maschera mask.
- `void addBinLEDs(int mask)`: aggiunge alla configurazione corrente i Bin LED specificati nella maschera mask.
- `void setBackLEDs(int mask, int intensity)`: imposta i Back LED specificando la maschera, mask, e l'intensità, intensity.

ESEMPIO:

Supponendo di voler accendere:

- le orecchie dell'Aibo con il colore verde
- il led blu sulla testa
- tutti i LED sulla schiena con il colore bianco
- gli occhi rossi sul muso dell'Aibo

I comandi da dare sarebbero:

```
MotionCommand cmd;
...
...
cmd.setBinLEDs(Motion::LED_EARS_GREEN | LED_TINY);
cmd.setBackLEDs(Motion::LED_BACK_FRONT_WHITE |
            Motion::LED_BACK_MIDDLE_WHITE |
            Motion::LED_BACK_REAR_WHITE,
            Motion::LED_BRIGHT);
cmd.setFaceLEDs(Motion::LED_RED_EYES_LOW, Motion::LED_BRIGHT,
Motion::MODE_A);
...
...
```

3.5 Creare i propri parametri

[versione introduttiva]

I parametri per i movimenti sono:

- **Parametri per le gambe** (46):
  - **Neutral Kinematic Position**: 3 per zampa
  - **Lift Velocity**: 3 per zampa
  - **Lift Time**: 1 per zampa
  - **Down Velocity**: 3 per zampa
  - **Down Time**: 1 per zampa
  - **Front Leg Height Limit**: 1 per Aibo
  - **Back Leg Height Limit**: 1 per Aibo
- **Parametri per il corpo** (8):
  - **Height of Body**: 1 per Aibo
  - **Angle of Body**: 1 per Aibo

- **Hop Amplitude**: 1 per Aibo
- **Sway Amplitude**: 1 per Aibo
- **Walk Period**: 1 per Aibo
- **Max Velocities**: 3 per Aibo.

I parametri sono contenuti nei file **.prm**; di default nella directory [MS/motion] sono presenti tre file di parametri:

1. **walk_xy.prm**: contiene i parametri per il movimento MOTION_WALK_TROT quando sono specificate solo le variabili vx e vj.
2. **walk_a.prm**: contiene i parametri per il movimento MOTION_WALK_TROT quando viene specificata anche la variabile va.
3. **walk_beh.prm**: contiene i parametri specificati dall'utente e vengono utilizzati quando il comando di movimento è MOTION_WALK_BEH.

Sono possibili due modalità di specifica dei parametri: una versione OFF-LINE e una a RunTime.

OFF-LINE:

A livello intuitivo, in questa modalità i vari parametri vengono passati ad un programma, genmot, che una volta elaborati produce in output un file .prm contenente i parametri specificati. Il vantaggio di questa modalità è che genmot effettua un controllo degli errori che evita possibili malfunzionamenti dell'Aibo, lo svantaggio è che non è possibile aggiornare i parametri a RunTime.

RunTime:

Questa modalità permette di modificare i parametri a runtime mandando quindi un oggetto WalkParam al servizio observer Params di MotionObject. Il vantaggio principale è dato dalla possibilità di modificare i parametri durante la vita del proprio programma permettendo quindi di raffinare i parametri in base all'ambiente di lavoro, lo svantaggio è che non è presente alcun tipo di controllo degli errori.

3.6 Esempio di Utilizzo

Di seguito è riportato il sorgente di ProvaCM un Oggetto Open-R che non fa altro che mandare una serie di comandi, contenuti in una coda, a intervalli regolari a MotionObject.

Di seguito lo stub.cfg di ProvaCM:

```
[stub.cfg]
ObjectName: ProvaCM
NumOfOSubject: 1
```

```
  NumOfOObserver: 1
  Service: "ProvaCM.MotionControl.MotionCommand.S",null,Ready()
  Service: "ProvaCM.Dummy.INT,O",null,null

  Extra: TimerEnd()
```

Come si puo' notare sono definiti due servizi, il servizio observer **Dummy** è necessario solo in quanto richiesto da Open-R, ma non viene utilizzato, mentre il servizio subject **MotionControl** serve per collegare ProvaCM con MotionObject (come si vedrà nel file CONNECT.CFG).
Viene inoltre definita la Extra Entry Point **TimerEnd** necessaria per la gestione dei Timer in Open-R (per informazioni su come gestire i Timer i Open-R vedi *Appendice A*).

Di seguito il file CONNECT.CFG:

```
#
# MotionObject <--> Aperios
#
MotionObject.MoveJoint.OCommandVectorData.S
OVirtualRobotComm.Effector.OCommandVectorData.O
OVirtualRobotComm.Sensor.OSensorFrameVectorData.S
MotionObject.SensorFrame.OSensorFrameVectorData.O
MotionObject.Speaker.OSoundVectorData.S
OVirtualRobotAudioComm.Speaker.OSoundVectorData.O

#
# ProvaCM --> MotionObject
#
ProvaCM.MotionControl.MotionCommand.S
MotionObject.Control.MotionCommand.O

#
# MotionObject <--> SharedMemMgr
#
MotionObject.RegisterRegion.SMMSharedMemRegion.S
SharedMemMgr.RegisterRegion.SMMSharedMemRegion.O
MotionObject.RequestRegion.RequestInfo.S
SharedMemMgr.RequestRegion.RequestInfo.O
SharedMemMgr.SendRegion.SMMSharedMemRegion.S
MotionObject.SharedMemRegionInfo.SMMSharedMemRegion.O
```

Il primo blocco di servizi serve a collegare MotionObject con gli oggetti di sistema di Aperios. Il secondo unisce il servizio MotionControl di ProvaCM con Control di MotionObject per la spedizione

dei comandi di movimento.
Il terzo blocco unisce MotionObject con SharedMemMgr l'oggetto per la gestione delle zone di memoria condivisa utilizzato in CMPack.

Di seguito l'Header ProvaCM.h:

```
#ifndef PROVACM_H_DEFINED
#define PROVACM_H_DEFINED
#include <OPENR/OObject.h>
#include <OPENR/OSubject.h>
#include <OPENR/OObserver.h>
#include <OPENR/RCRegion.h>
#include "def.h"
#include <queue>
#include "../headers/Geometry.h"
#include "../Motion/MotionInterface.h"

using namespace std;

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

  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);

  void TimerEnd(void* msg);
private:
  queue<Motion::MotionCommand> codaComandi;
};

#endif
```

Come si può notare i metodi definiti sono pressoché i metodi base di tutti gli oggetti Open-R; il metodo

Ready viene invocato al ricevimento di un AssertReady da parte di MotionObject (definito solo per la stampa di informazioni di Debug), mentre **TimerEnd** è l'extra entry point invocata allo scadere del timer. Da notare la coda di Motion::MotionCommand codaComandi, coda in cui verranno memorizzati tutti i comandi da mandare ad intervalli regolari a MotionObject.

Di seguito il sorgente di ProvaCM:

```
#include "ProvaCM.h"
#include <OPENR/OSyslog.h>
#include <OPENR/core_macro.h>
#include <OPENR/OPENRAPI.h>
#include "entry.h"
#include "../Motion/MotionInterface.h"

ProvaCM::ProvaCM(){
  Motion::MotionCommand cmd;

  cmd.motion_cmd=Motion::MOTION_TURN_WITH_BALL_L;
  cmd.head_cmd=Motion::HEAD_ANGLES;
  cmd.tail_cmd=Motion::TAIL_FOLLOW;
  cmd.bound_mode;
  cmd.vx=0;
  cmd.vy=0;
  cmd.va=2;
  cmd.head_lookat=vector3d(0,0,0);
  cmd.head_tilt_offset=0;
  cmd.head_tilt=-20.0;
  cmd.head_pan=0;
  cmd.head_roll=0;
  cmd.head_tilt2=0;
  cmd.tail_pan=60;
  cmd.tail_tilt=0;
  cmd.mouth=0;
  cmd.sound_cmd=Motion::SOUND_FILE;
  cmd.sound_file=Motion::SOUND_FILE_RECTANGLE;
  cmd.setFaceLEDs(0,0,Motion::MODE_A);
  codaComandi.push(cmd);

  cmd.motion_cmd=Motion::MOTION_TURN_WITH_BALL_L;
  cmd.head_cmd=Motion::HEAD_NO_CMD;
  cmd.tail_cmd=Motion::TAIL_FOLLOW;
```

```
cmd.bound_mode;
cmd.vx=0;
cmd.vy=0;
cmd.va=2;
cmd.head_lookat=vector3d(0,0,0);
cmd.head_tilt_offset=0;
cmd.head_tilt=0;
cmd.head_pan=100;
cmd.head_roll=0;
cmd.head_tilt2=0;
cmd.tail_pan=-60;
cmd.tail_tilt=60;
cmd.mouth=0;
cmd.sound_cmd=Motion::SOUND_FILE;
cmd.sound_file=Motion::SOUND_FILE_TADA;
cmd.setFaceLEDs(0,0,Motion::MODE_A);
cmd.setBinLEDs(Motion::LED_TOP_ORANGE);
codaComandi.push(cmd);

cmd.motion_cmd=Motion::MOTION_TURN_WITH_BALL_L;
cmd.head_cmd=Motion::HEAD_NO_CMD;
cmd.tail_cmd=Motion::TAIL_NO_CMD;
cmd.bound_mode;
cmd.vx=0;
cmd.vy=0;
cmd.va=2;
cmd.head_lookat=vector3d(0,0,0);
cmd.head_tilt_offset=0;
cmd.head_tilt=0;
cmd.head_pan=0;
cmd.head_roll=100;
cmd.head_tilt2=0;
cmd.tail_pan=50;
cmd.tail_tilt=50;
cmd.mouth=0;
cmd.sound_cmd=Motion::SOUND_FILE;
cmd.sound_file=Motion::SOUND_FILE_L;
cmd.setFaceLEDs(0,0,Motion::MODE_A);
cmd.setBinLEDs(Motion::LED_TOP_WHITE);
codaComandi.push(cmd);
```

```
cmd.motion_cmd=Motion::MOTION_TURN_WITH_BALL_L;
cmd.head_cmd=Motion::HEAD_ANGLES;
cmd.tail_cmd=Motion::TAIL_NO_CMD;
cmd.bound_mode;
cmd.vx=0;
cmd.vy=0;
cmd.va=2;
cmd.head_lookat=vector3d(0,0,0);
cmd.head_tilt_offset=0;
cmd.head_tilt=0;
cmd.head_pan=0;
cmd.head_roll=0;
cmd.head_tilt2=100;
cmd.tail_pan=50;
cmd.tail_tilt=50;
cmd.mouth=0;
cmd.sound_cmd=Motion::SOUND_NOTE;
cmd.sound_frequency=1000;
cmd.sound_duration=150000;
cmd.setFaceLEDs(0,0,Motion::MODE_A);
cmd.setBinLEDs(Motion::LED_TINY);
codaComandi.push(cmd);

cmd.motion_cmd=Motion::MOTION_TURN_WITH_BALL_R;
cmd.head_cmd=Motion::HEAD_NO_CMD;
cmd.tail_cmd=Motion::TAIL_FOLLOW;
cmd.bound_mode;
cmd.vx=0;
cmd.vy=0;
cmd.va=-2;
cmd.head_lookat=vector3d();
cmd.head_tilt_offset=0;
cmd.tail_pan=0;
cmd.tail_tilt=-100;
cmd.mouth=0;
cmd.sound_cmd=Motion::SOUND_NOTE;
cmd.sound_frequency=10000;
cmd.sound_duration=150000;
cmd.sound_file=Motion::SOUND_FILE_TADA;
```

```
  cmd.setFaceLEDs(0,0,Motion::MODE_A);
  cmd.setBinLEDs(Motion::LED_EARS_CYAN);
  codaComandi.push(cmd);
}

OStatus ProvaCM::DoInit(const OSystemEvent& event){
  OSYSDEBUG(("ProvaCM::DoInit()\n"));

  NEW_ALL_SUBJECT_AND_OBSERVER;
  REGISTER_ALL_ENTRY;
  SET_ALL_READY_AND_NOTIFY_ENTRY;

  return oSUCCESS;
}


OStatus ProvaCM::DoStart(const OSystemEvent& event){
  OSYSDEBUG(("ProvaCM::DoStart()\n"));

  ENABLE_ALL_SUBJECT;
  ASSERT_READY_TO_ALL_OBSERVER;

  OStatus st = OPENR::SetMotorPower(opowerON);

  int dummy=0;
  EventID sincroEvent = EventID();
  RelativeTime period(10, 0);
  TimeEventInfoWithRelativeTime timeInfo
     (TimeEventInfo::Periodic, period);
  sError error=SetTimeEvent(&timeInfo,myOID_,Extra_Entry
     [entryTimerEnd],&dummy,sizeof(dummy),&sincroEvent);
  if(error!=sSUCCESS)
    OSYSLOG1((osyslogERROR, "::DoStart() : ERROR: Sincronization
      Timer not started\n"));

  return oSUCCESS;
}  //DoStart() END


OStatus ProvaCM::DoStop(const OSystemEvent& event){
```

```
  OSYSDEBUG(("ProvaCM::DoStop()\n"));

  DISABLE_ALL_SUBJECT;
  DEASSERT_READY_TO_ALL_OBSERVER;
  OStatus st = OPENR::SetMotorPower(opowerOFF);

  return oSUCCESS;
}  //DoStop() END


OStatus ProvaCM::DoDestroy(const OSystemEvent& event){
  OSYSDEBUG(("ProvaCM::DoDestroy\()\n"));

  DELETE_ALL_SUBJECT_AND_OBSERVER;

  return oSUCCESS;
}  //DoDestroy() END

void ProvaCM::TimerEnd(void* msg){
  if(!codaComandi.empty())
    {
      subject[sbjMotionControl]->SetData(&(codaComandi.front()),
     sizeof(Motion::MotionCommand));
      subject[sbjMotionControl]->NotifyObservers();
      codaComandi.pop();

    }
}

void ProvaCM::Ready(const OReadyEvent& event){
  OSYSDEBUG(("ProvaCM:: ricevuto Assert Ready\n"));
}
```

Come si può notare il codice è molto semplice. Il costruttore di ProvaCM aggiunge nella coda codaComandi i vari MotionCommand da spedire a MotionObject. I metodi DoInit, DoStart, DoStop e DoDestroy richiamano le macro predefinite in Open-R per inizializzare l'oggetto e la comunicazione Inter-object in più il metodo DoStart inizializza il Timer utilizzato per schedulare la spedizione dei comandi. Il Timer ha durata di 10 secondi, allo scadere viene invocata la funzione TimerEnd che prende la testa della coda e la spedisce a MotionObject, questo fino a quando sono presenti comandi. La funzione Ready viene invocata al ricevimento di un AssertReady spedito da MotionObject, come spiegato tale funzione non effettua alcuna elaborazione a parte stampare sulla console Telnet il

messaggio *“ricevuto Assert Ready”*.

A Timer in OPENR

I timer risultano molto utili nella programmazione robotica, purtroppo però nella documentazione standard di Open-R non è presente alcun riferimento alla possibilità di impostare timer, anche se all'interno degli header standard sono presenti le funzioni necessarie per farlo.
Da un punto di vista intuitivo in Open-R i timer vengono impostati attraverso una chiamata di sistema che richiede come parametri, un oggetto che specifica la durata e il tipo di timer (se periodico o non periodico) e il riferimento ad una funzione che si vuole venga invocata allo scadere del timer. Questa metodologia permette di impostare un timer, quindi dilazionare l'esecuzione di parte del codice, senza bloccare l'oggetto Open-R in un ciclo di attesa, permettendo quindi all'oggetto di proseguire nella sua elaborazione.
I passi da seguire per impostare un timer in Open-R sono:

1) Dichiarare nel file **stub.cfg** una *extra entry point* con la parola chiave **Extra: <nome funzione>**:
   p.es:
   ```
   [stub.cfg]
   ObjectName: ProvaCM
   NumOfOSubject: 1
   NumOfOObserver: 1
   Service: "ProvaCM.MotionControl.MotionCommand.S",null,Ready()
   Service: "ProvaCM.ActionCompleted.MovementInfo.O",null,Notify()

   Extra: TimerEnd()
   ```

   Quindi definire all'interno del proprio oggetto Open-R la funzione appena definita in stub.cfg rispettando la seguente segnatura:
   ```
   void TimerEnd(void* msg)
   ```

2) Impostare il timer attraverso la chiamata di sistema **SetTimeEvent**:
   ```
   int dummy=0;
   EventID sincroEvent = EventID();
   RelativeTime period(25, 0);
   TimeEventInfoWithRelativeTime timeInfo
                               (TimeEventInfo::Periodic, period);
   sError error=SetTimeEvent(&timeInfo,
                              myOID_,
                              Extra_Entry[entryTimerEnd],
                              &dummy,
   ```

```
                                    sizeof(dummy),
                                    &sincroEvent);
  if(error!=sSUCCESS)
    OSYSLOG1((osyslogERROR, "Timer not started\n"));
```

sincroEvent rappresenta un evento di sistema, viene creato con il costruttore di default di conseguenza viene creato un evento “not defined”, è necessario come parametro per la partenza del timer, ma è di poco interesse pratico.
period di tipo RelativeTime specifica la durata del timer, il costruttore riceve due parametri, il primo specifica i secondi, il secondo specifica i milli secondi.
timeInfo specifica le caratteristiche del timer, nell'esempio viene creato un timer periodico (attraverso la costante *TimeEventInfo::Periodic*, specificando *TimeEventInfo::NonPeriodic* si sarebbe creato un timer non periodico) della durata specificata dall'oggetto period.
Infine la funzione **SetTimerEvent** imposta e fa partire il timer; riceve come argomenti:

1. l'oggetto **timeInfo** che specifica le caratteristiche del timer
2. **myOID_**, una variabile presente in tutti gli oggetti Open-R che contiene un identificativo dell'oggetto stesso.
3. Extra_Entry è l'array delle extra entry point (specificate nel file stub.cfg) definito all'intero del file def.h generato in automatico dallo script stubgen sulla base dei valori contenuti nel file stub.cfg. L'array contiene i riferimenti alle varie extra entry point definite nell'oggetto. Nel nostro esempio l'extra entry point definita per la gestione del timer è la funzione TimerEnd, quindi passando come parametro **Extra_Entry[entryTimerEnd]** SetTimeEvent viene informato che la funzione da invocare allo scadere del timer è TimerEnd.
4. **dummy** (nell'esempio è un intero, ma può essere un qualunque tipo predefinito) rappresenta il parametro con cui verrà invocata la extra entry point allo scadere del timer. Tale parametro verrà passato alla funzione come puntatore void, sarà quindi necessario un cast all'interno dell'extra entri point prima di poter utilizzare il parametro.
5. **sizeof(dummy)** è la dimensione in byte del parametro che si vuole passare alla extra entry point, informazione necessaria per la gestione della memoria.
6. **sincroEvent** è l'oggetto EventID necessario a Aperios per la gestione degli eventi.

3) Se la funzione ritorna un sSUCCESS allora il timer è partito, allo scadere verrà invocata la funzione passata come terzo parametro a SetTimerEvent con il parametro passato come quarto parametro alla funzione SetTimerEvent. Se la funzione ritorna un sERROR allora c'è stato un errore nella creazione del timer.