AIBO Sony – Movimento, LED, sensori, suono

Alessandro Quattro
Laboratory of Applied Intelligent Systems (AIS-Lab)
http://ais-lab.dsi.unimi.it

Sommario

Il documento si propone di essere una rielaborazione della documentazione reperita su internet, sia proveniente da Sony che non.

1. Scopo del documento

Questo documento è stato scritto con lo scopo di illustrare le possibilità di movimento dei robot Aibo e di spiegare le operazioni necessarie per far muovere i diversi joint.

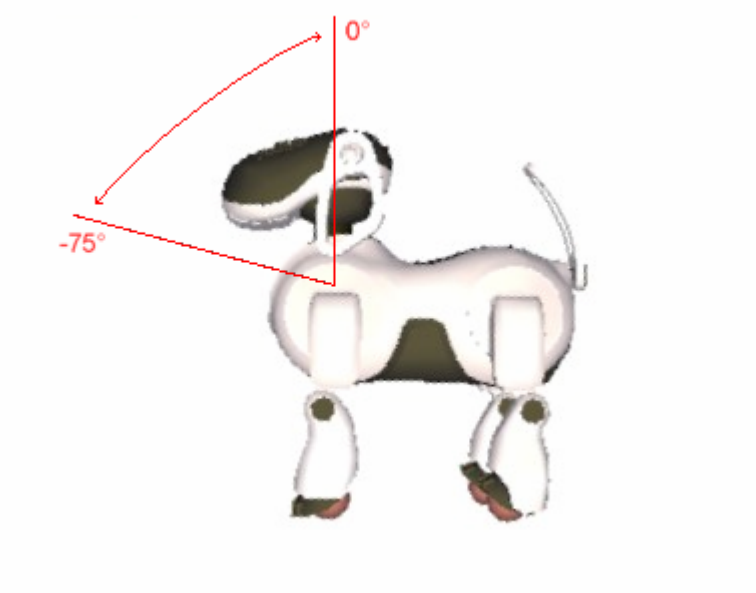
2. Possibilità di movimento

I robot Aibo ers-7 hanno un totale di 18 joint:

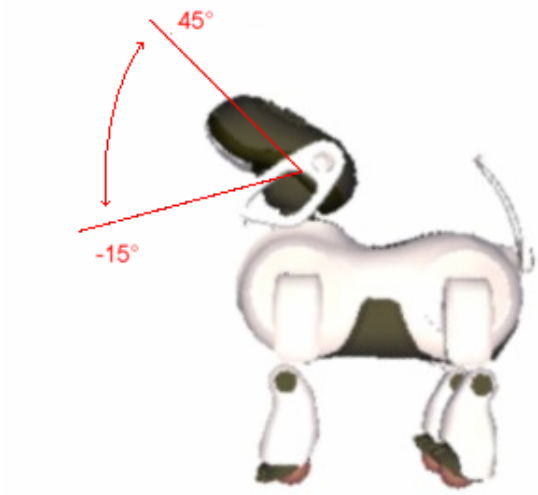
- 3 per muovere il collo e la testa.
- 1 per muovere la bocca.
- 12 per muovere le zampe.
- 2 per muovere la coda.

Nel seguito si riporta un elenco di questi joint specificando, per ognuno, i limiti di movimento.

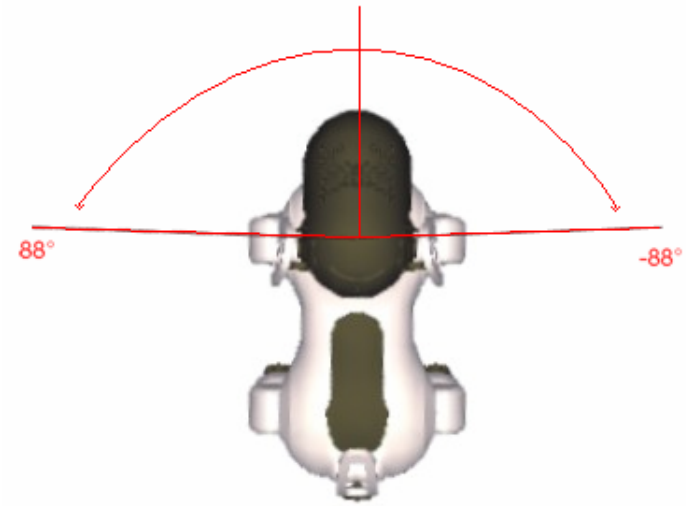
1- Neck tilt1
Permette di muovere il collo avanti/indietro tra -75° e 0°



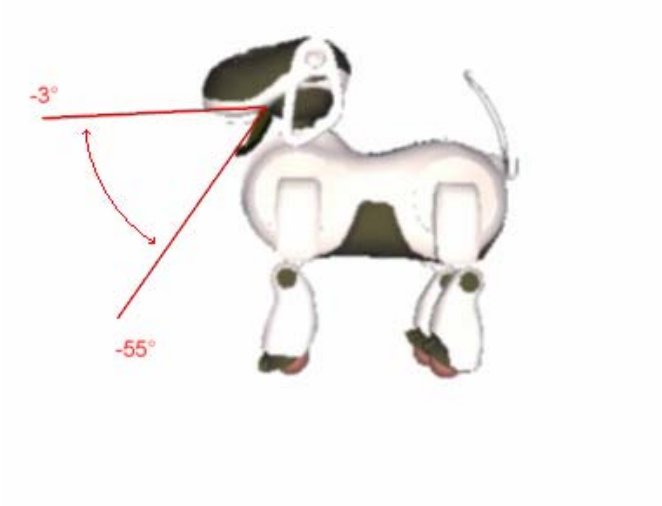
2- Neck tilt2
Permette di muovere la testa verso l'alto/basso tra 45° e -15°



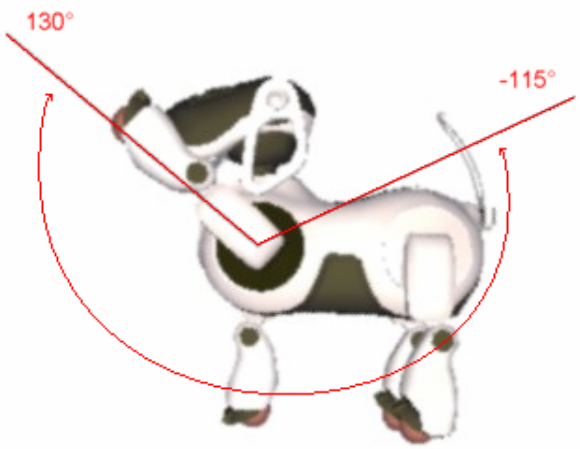
3- Neck pan
Permette di ruotate la testa verso destra/sinistra tra -88° e 88°



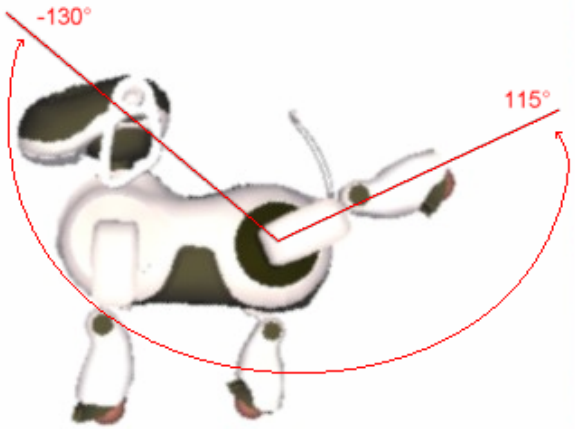
4- Mouth
Permette di aprire/chiudere la bocca muovendo la mandibola tra -55° e -3°



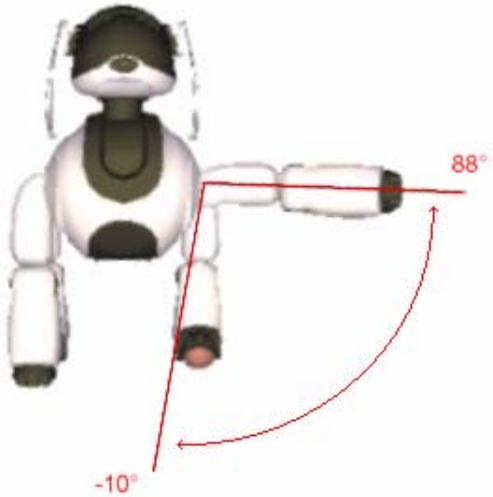
5-6- Front Leg J1 (left/right)
Permettono di muovere le zampe anteriori avanti/indietro tra 130° e -115°



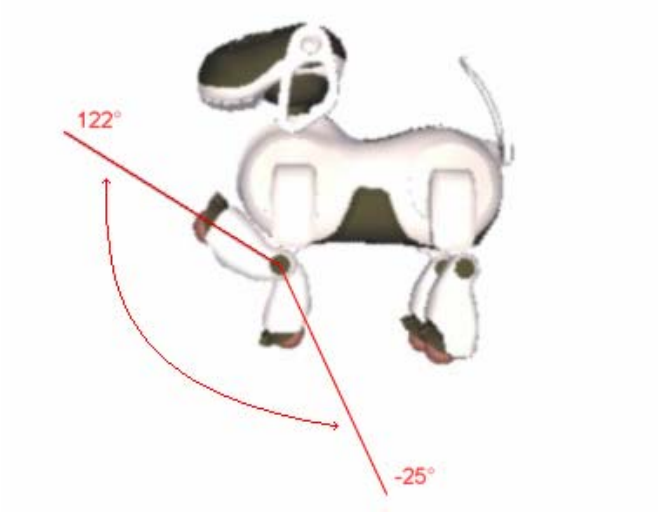
7-8- Rear Leg J1 (left/right)
Permettono di muovere le zampe posteriori avanti/indietro tra -130° e 115°



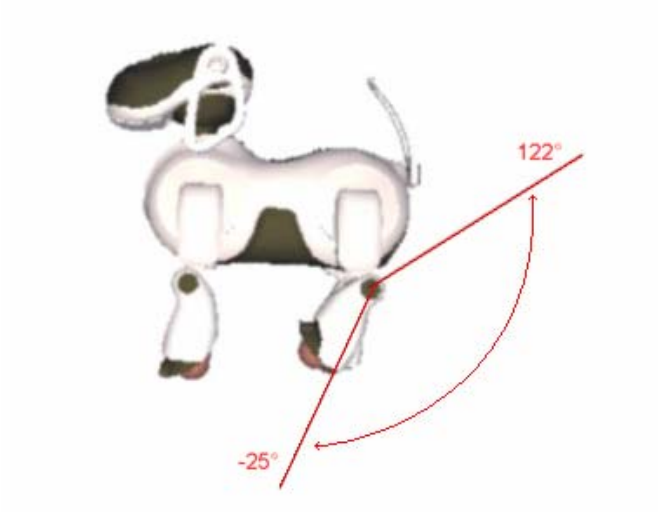
9-10-11-12- Leg J2 (front/rear, left/right)
Permettono di addurre/abdurre le zampe tra -10° e 88°



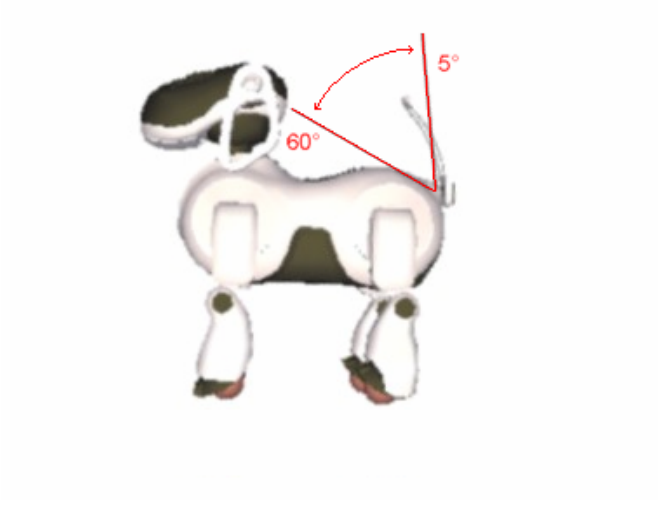
13-14- Front Leg J3 (left/right)
Permettono di piegare le zampe anteriori avanti/indietro tra 122° e -25°



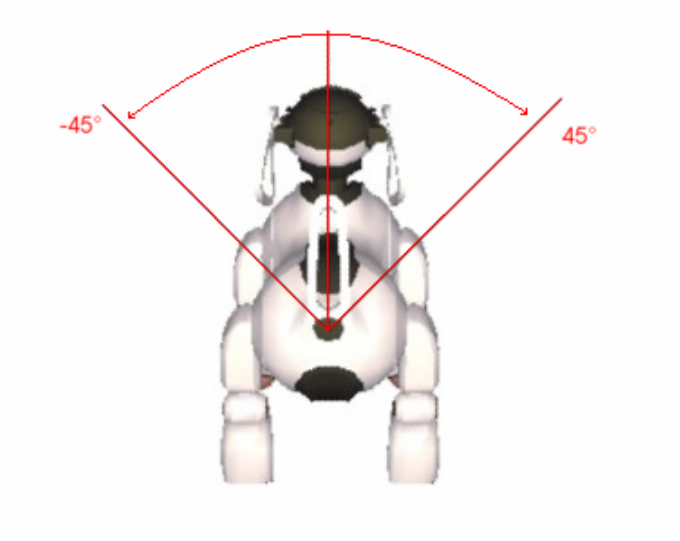
15-16- Rear Leg J3 (left/right)
Permettono di piegare le zampe posteriori avanti/indietro tra -25° e 122°



17- Tail tilt
Permette di muovere la coda avanti/indietro tra 60° e 5°



18- Tail pan
Permette di ruotate la coda verso destra/sinistra tra 45° e -45°



3. Oggetti di sistema (OVirtualRobotComm)

I servizi del livello di sistema sono offerti attraverso gli oggetti di sistema OVirtualRobotComm e OVirtualRobotAudioComm.

In particolare ***OVirtualRobotComm*** si interfaccia con i joint, i sensori, i LED e la telecamera del robot.

OVirtualRobotAudioComm si interfaccia con i device audio dell'Aibo.

Come gli altri oggetti OPEN-R anche gli oggetti di sistema comunicano attraverso lo scambio di messaggi (cf. Manuale introduttivo).

Questi messaggi conterranno riferimenti a delle strutture predefinite.

Per comunicare con l'oggetto ***OVirtualRobotComm*** si devono utilizzare le seguenti strutture dati:

- OCommandVectorData: che può contenere comandi per joint e LED.
- OSensorVectorData: che contiene informazioni provenienti dai sensori.
- OFbkImageVectorData: contenente immagini provenienti dalla telecamera.

Per comunicare con l'oggetto ***OVirtualRobotAudioComm*** viene usata la struttura dati OSoundVectorData adatta a contenere dati audio, sia in input che in output.

L'oggetto ***OVirtualRobotComm*** ha un servizio observer che si chiama Effector e due servizi subject rispettivamente nominati Sensor e FbkImageSensor.

Il servizio Effector riceve messaggi contenenti riferimenti a strutture di tipo OCommandVectorData.

Il servizio Sensor invia messaggi riferiti a un tipo OSensorVectorData.

Il servizio FbkImageSensor invia messaggi riferiti a un tipo OFbkImageVectorData.

L'oggetto ***OVirtualRobotAudioComm*** ha un servizio observer che si chiama Speaker e un servizio subject che si chiama Mic.

Il servizio Speaker riceve messaggi riferiti a un tipo OSoundVectorData.

Il servizio Mic invia messaggi riferiti a un tipo OSoundVectorData.

Ricordiamo che le misure dei sensori (ed i comandi) sono campionati ogni 8 ms (frame) e che vengono resi disponibili in blocchi di n frame ogni n * 8 msec (cf. Documentazione, 3.7).

4. Esecuzione del movimento

Il movimento del robot AIBO viene eseguito attraverso l'oggetto di sistema ***OVirtualRobotComm*** che si interfaccia con l'hardware.

Per far eseguire dei movimenti al robot l'oggetto applicativo dovrà interfacciarsi con il servizio Effector dell'oggetto OVirtualRobotComm.

In particolare dovrà inviare al servizio Effector dell'oggetto OVirtualRobotCommon dei messaggi contenenti un riferimento a una struttura di tipo OCommandVectorData.

Nel seguito verrà spiegata la sequenza di operazioni che un oggetto applicativo deve compiere per far eseguire dei movimenti all'oggetto di sistema OVirtualRobotComm.

Come esempio verrà illustrato il codice dell'oggetto applicativo MoveLegs.

4.1 Sequenza di inizializzazione

Prima di muovere i joint dell'AIBO è necessario eseguire una sequenza di inizializzazione.

Per ogni joint che dovrà essere mosso sono necessarie le seguenti operazioni:

4.1.1 Convertire l'indirizzo del joint in un OprimitiveID

Per fare questo e sufficiente chiamare la funzione OPENR::OpenPrimitive().

Occorrono il file MoveLegs.h e la funzione OpenPrimitives().

Il file MoveLegs.h contiene le definizioni dei joint (in questo caso 3 joint per ogni zampa)

nel file MoveLegs.h

```
...
static const char* const JOINT_LOCATOR[] = {
    "PRM:/r4/c1-Joint2:41",       // RFLEG J1 (Right Front Leg)
    "PRM:/r4/c1/c2-Joint2:42",    // RFLEG J2
    "PRM:/r4/c1/c2/c3-Joint2:43", // RFLEG J3

    "PRM:/r2/c1-Joint2:21",       // LFLEG J1 (Left Front Leg)
    "PRM:/r2/c1/c2-Joint2:22",    // LFLEG J2
    "PRM:/r2/c1/c2/c3-Joint2:23", // LFLEG J3

    "PRM:/r5/c1-Joint2:51",       // RRLEG J1 (Right Rear Leg)
    "PRM:/r5/c1/c2-Joint2:52",    // RRLEG J2
    "PRM:/r5/c1/c2/c3-Joint2:53", // RRLEG J3

    "PRM:/r3/c1-Joint2:31",       // LRLEG J1 (Left Rear Leg)
    "PRM:/r3/c1/c2-Joint2:32",    // LRLEG J2
    "PRM:/r3/c1/c2/c3-Joint2:33"  // LRLEG J3
};
...

static const size_t NUM_JOINTS        = 12;  //numero di joint che verranno controllati
...
OPrimitiveID      jointID[NUM_JOINTS];
...
```

A questo punto si può chiamare la funzione OpenPrimitives() che traduce le definizioni dei joint in codice operativo.

nel file MoveLegs.cc

```
…
void
MoveLegs::OpenPrimitives()
{
```

```
    for (int i = 0; i < NUM_JOINTS; i++) {
        OStatus result = OPENR::OpenPrimitive(JOINT_LOCATOR[i], &jointID[i]);
        if (result != oSUCCESS) {
        ....
        }
}
```

4.1.2 Settare i valori di gain (P,I e Dgains) e shift (P, I e D shifts)

Per attivare i gains viene chiamata la funzione OPENR::EnableJointGain(). Successivamente i valori possono essere settati con la funzione OPENR::SetJointGain()

Di solito si utilizzano i valori standard raccomandati dalla Sony.

P, I e D gains e shifts sono parametri hardware per il servo control. **Un errore nel settaggio di questi parametri potrebbe danneggiare il robot**.

nel file MoveLegs.h

```
...
    static const word   J1_PGAIN = 0x0010;
    static const word   J1_IGAIN = 0x0004;
    static const word   J1_DGAIN = 0x0001;

    static const word   J2_PGAIN = 0x000a;
    static const word   J2_IGAIN = 0x0004;
    static const word   J2_DGAIN = 0x0001;

    static const word   J3_PGAIN = 0x0010;
    static const word   J3_IGAIN = 0x0004;
    static const word   J3_DGAIN = 0x0001;

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

nel file MoveLegs.cc

```
...
void
MoveLegs::SetJointGain()
{
    for (int i = 0; i < 4; i++) {

        int j1 = 3 * i;
        int j2 = 3 * i + 1;
        int j3 = 3 * i + 2;

        OPENR::EnableJointGain(jointID[j1]);
        OPENR::SetJointGain(jointID[j1],
                            J1_PGAIN, J1_IGAIN, J1_DGAIN,
                            PSHIFT, ISHIFT, DSHIFT);

        OPENR::EnableJointGain(jointID[j2]);
        OPENR::SetJointGain(jointID[j2],
                            J2_PGAIN, J2_IGAIN, J2_DGAIN,
                            PSHIFT, ISHIFT, DSHIFT);

        OPENR::EnableJointGain(jointID[j3]);
        OPENR::SetJointGain(jointID[j3],
                            J3_PGAIN, J3_IGAIN, J3_DGAIN,
                            PSHIFT, ISHIFT, DSHIFT);
    }
}
```

4.2 La struttura OCommandVectorData

Il tipo OCommandVectorData è una struttura che contiene comandi per il movimento dei joint (o per l'accensione dei LED) per un massimo di 16 frame consecutivi. Questa struttura contiene 3

membri: **vectorInfo** di tipo **ODataVectorInfo**, un **array di OCommandInfo** e un **array di OCommandData**.

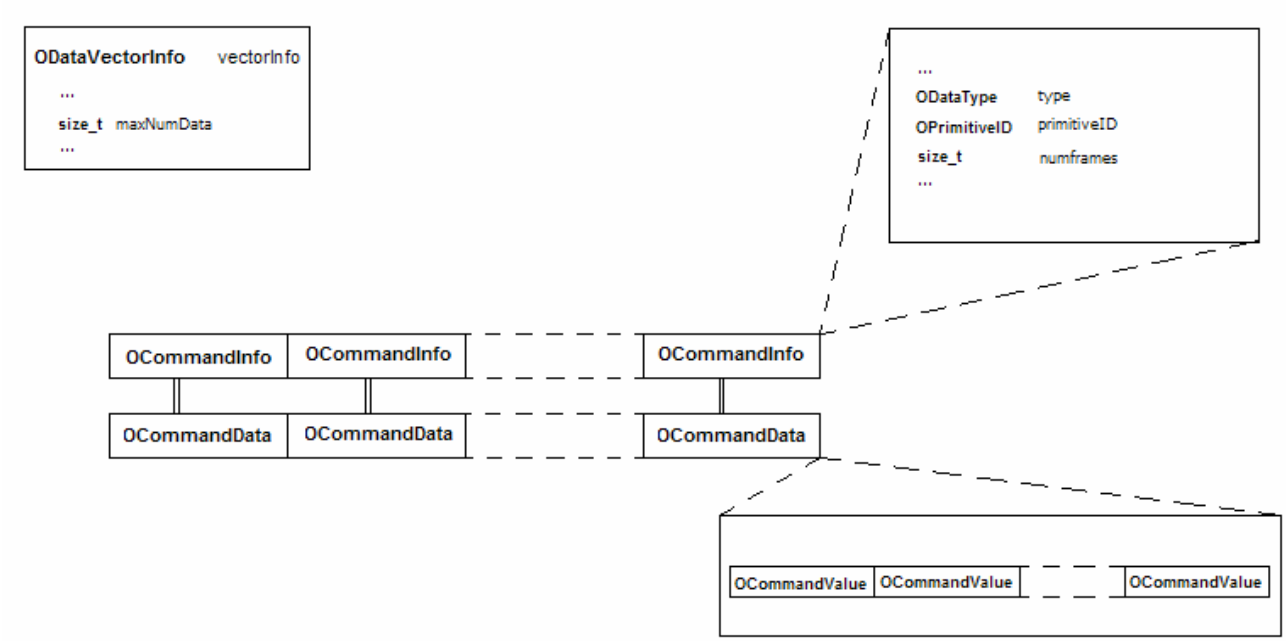


Fig.1 – La struttura OCommandVectorData..

Ad ogni elemento del vettore di OCommandInfo corrisponde un elemento nel vettore di OCommandData. Ogni coppia di elementi dei due array è riferita a un joint (o a un LED). Quindi la dimensione di questi 2 array dovrà essere uguale al numero di joint (o LED) che si vogliono controllare.

Il tipo **OCommandData** è a sua volta costituito da un array di 16 **OCommandValue** (ciascuno riferito ad un frame).

Questo tipo (OCommandValue) è un tipo generale per inviare comandi per un frame (sia a un joint che a un LED).

Nel caso il comando sia riferito ad un joint OCommandData dovrà essere castato al tipo OJointCommandValue2 e conterrà il valore dell'angolazione da fare assumere al joint in quel frame.

ESEMPIO

```
    OCommandData* data; //puntatore all'elemento della struttura dati dei comandi (array di OCommandData) che si vuole settare
    OJointCommandValue2* jval = (OJointCommandValue2*)data->value;

    for (int i = 0; i < numframes; i++) {
        double dval; //valore di tipo double stabilito dal programma
        jval[i].value = oradians(dval);
    }
```

La struttura **OCommandInfo** ha tre membri modificabili mediante la funzione set.

ESEMPIO

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

I primi due descrivono il tipo di harware a cui si riferisce il comando (odataJOINT_COMMAND2 per i joint, odataLED_COMMAND2 per i LED), e l'identificativo dell'indirizzo dell'hardware (OprimitiveID).

Il terzo membro indica il numero di elementi validi contenuti nell'elemento OcommandData corrispondente. Cioè indica quanti dei 16 elementi dell'array di OCommandValue dovranno essere

letti dall'oggetto OVirtualRobotComm.(questo numero sarà compreso tra 0 e ocommandMAX_FRAMES=16)

4.3 Creazione della struttura OCommandVectorData e utilizzo della memoria condivisa

I messaggi hanno una dimensione massima che è inferiore alla dimensione della struttura OCommandVectorData.

Quindi non è possibile inviare messaggi di tipo OCommandVectorData.

Per risolvere questo problema gli oggetti inviano un puntatore a una struttura di tipo OCommandVectorData allocata nella memoria condivisa.

Per creare una struttura OCommandVectorData in una zona di memoria condivisa si può utilizzare la funzione **OPENR::NewCommandVectorData(size_t numCommands, MemoryRegionID* memID, OCommandVectorData** baseAddr)**.

Questa funzione infatti crea una struttura di tipo OCommandVectorData con la dimensione dei due array di OCommandInfo e di OCommandData pari a numCommands elementi.

Inoltre inizializza il valore di baseAddr con un puntatore alla struttura creata e il valore di memID con l'identificativo della zona di memoria condivisa dove questa struttura è contenuta.

```
MemoryRegionID      cmdVecDataID;
OCommandVectorData* cmdVecData;
…
    OPENR::NewCommandVectorData(NUM_JOINTS,
                                        &cmdVecDataID, &cmdVecData);
```

Dopo che è stata chiamata la funzione OPENR::NewCommandVectorData() è possibile istanziare la classe RCRegion che viene fornita da OPEN-R SDK per semplificare l'accesso alla memoria condivisa garantendo mutua esclusione.

Questa classe può essere istanziata utilizzando il seguente costruttore:

RCRegion(MemoryRegionID memID, size_t offset, void* baseAddr, size_t size).

I valori degli argomenti da passare possono essere recuperati dalla struttura OCommandVectorData precedentemente allocata.

```
region[i] = new RCRegion(cmdVecData->vectorInfo.memRegionID,
                         cmdVecData->vectorInfo.offset,
                         (void*)cmdVecData,
                         cmdVecData->vectorInfo.totalSize);
```

A questo punto sarà necessario inizializzare i componenti della struttura OCommandVectorData.

Prima si dovrà settare il numero di elementi degli array di OCommandInfo e di OCommandData, corrispondente al numero di joint che dovranno essere controllati (nell'esempio NUM_JOINTS = 12).

```
cmdVecData->SetNumData(NUM_JOINTS);
```

Poi si dovranno settare tutti gli elementi dell' array di OCommandInfo.

Nell'esempio si vogliono controllare 12 joint i cui ID sono contenuti nell'array jointID[], inoltre si è scelto di sfruttare tutti gli elementi (ocommandMAX_FRAMES = 16) di ogni array di OCommandValue cioè di inviare comandi per 16 frame alla volta. Quindi gli elementi dell' array di OCommandInfo vengono settati col seguente ciclo:

```
for (int j = 0; j < NUM_JOINTS; j++) {
    info = cmdVecData->GetInfo(j);
    info->Set(odataJOINT_COMMAND2, //tipo del dato(odataJOINT_COMMAND2 per i joint)
              jointID[j],          //ID del joint
              ocommandMAX_FRAMES); //numero di frame
}
```

Nel nostro esempio si è scelto di utilizzare 2 strutture OCommandVectorData allocate in 2 zone di memoria condivisa così che mentre l'oggetto OVirtualRobotComm sta leggendo da una di queste l'oggetto MoveLegs può scrivere nell'altra.

nel file MoveLegs.h

```
…
static const size_t NUM_COMMAND_VECTOR = 2;   //numero di RCRegion (zone di memoria condivisa)
…
```

nel file MoveLegs.cc

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

    for (int i = 0; i < NUM_COMMAND_VECTOR; i++) {
        //inizializzo la struttura OCommandVectorData
        result = OPENR::NewCommandVectorData(NUM_JOINTS,
                                             &cmdVecDataID, &cmdVecData);
    ...

        //creo un RCRegion che punta alla struttura OCommandVectorData
        region[i] = new RCRegion(cmdVecData->vectorInfo.memRegionID,
                                 cmdVecData->vectorInfo.offset,
                                 (void*)cmdVecData,
                                 cmdVecData->vectorInfo.totalSize);
        //setto alcuni membri della struttura OCommandVectorData
        cmdVecData->SetNumData(NUM_JOINTS); //numero di elementi del vetore
                                                                    //di OCommandData(n. di joint)
        //setto gli elementi del vettore di OCommandInfo
        for (int j = 0; j < NUM_JOINTS; j++) {
            info = cmdVecData->GetInfo(j);
            info->Set(odataJOINT_COMMAND2, //tipo del dato(odataJOINT_COMMAND2 per i joint)
                      jointID[j],          //ID del joint
                      ocommandMAX_FRAMES); //numero di frame
        }
    }
}
```

4.4 Accensione dei motori e correzioni iniziali

Prima di poter muovere i joint è necessario assicurarsi che i motori siano accesi, per far questo è sufficiente chiamare la funzione OPENR::SetMotorPower(OPower power) passando come valore del parametro opowerON.

```
OStatus st = OPENR::SetMotorPower(opowerON); //accende i motori
```

Quando i motori dell'AIBO vengono accesi potrebbe esserci una differenza tra la posizione reale del joint e la posizione data dal sensore.

E' quindi necessario correggere questa differenza prima di muovere il joint.

Per fare questo il valore del joint viene letto con la funzione OPENR::GetJointValue(OPrimitiveID primitiveID, OJointValue* value), poi il joint viene settato con il valore letto e infine viene inviato il comando per far eseguire l'aggiustamento.

nel file MoveLegs.cc

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

```
    for (int i = 0; i < NUM_JOINTS; i++) {
        OPENR::GetJointValue(jointID[i], &current[i]);   //lettura dei valori dei joint
        SetJointValue(region[0], i,
                      degrees(current[i].value/1000000.0),
                      degrees(current[i].value/1000000.0)); //settaggio con i valori letti
    }
    //esecuzione della correzione
    subject[sbjMove]->SetData(region[0]);
    subject[sbjMove]->NotifyObservers();
...
 }
```

4.5 Settaggio dei joint ed esecuzione del movimento

Dopo aver creato una struttura OCommandVectorData e la corrispondente RCRegion è possibile settare i valori delle angolazioni da fare assumere ai joint nei successivi n frame (con n compreso tra 0 e ocommandMAX_FRAME=16).

Cioè è possibile riempire n elementi di ogni array di OCommandValue della struttura OCommandVectorData (come già detto, nel nostro esempio sono stati sfruttati tutti i 16 elementi di ogni array di OCommandValue).

Innanzi tutto bisogna recuperare un' RCRegion associata ad una zona di memoria condivisa che contiene una struttura OCommandVectorData alla quale nessun altro oggetto stia accedendo.

La classe RCRegion ha una funzione NumberOfReference() che restituisce il numero di oggetti che stanno accedendo all'area di memoria allocata. Se questo numero è 1 significa che solo l'oggetto corrente sta leggendo la memoria allocata ed è quindi possibile scriverci.

nel file MoveLegs.cc

```
…
RCRegion*
MoveLegs::FindFreeRegion()
{
    for (int i = 0; i < NUM_COMMAND_VECTOR; i++) {
        if (region[i]->NumberOfReference() == 1) return region[i];
    }
    return 0;
}
...
```

A questo punto è possibile riempire la struttura OCommandVectorData.

Per ogni elemento dell'array di OCommandData, corrispondente ad un joint, verranno settati i valori degli elementi dell'array di OCommandValue, corrispondenti ciascuno ad un frame.

Nel nostro esempio questo viene fatto chiamando la funzione privata SetJointValue(RCRegion* rgn, int idx, double start, double end) per ogni elemento dell'array di OCommandData (quindi per ogni joint). Il parametro idx è l'indice dell'array di OCommandData, start e end sono rispettivamente i valori dell'angolazione iniziale e finale del joint mentre rgn è un puntatore all'RCRegion precedentemente recuperata.

In questa funzione il movimento dall'angolazione iniziale (start) all'angolazione finale (end) viene diviso in ocommandMaxFrame = 16 intervalli di uguale ampiezza in modo da ottenere una velocità di movimento costante. Vengono quindi identificati 16 valori di angolazioni ognuno dei quali viene assegnato a un elemento dell'array di OCommandValue.

nel file MoveLegs.cc

```
…

void
MoveLegs::SetJointValue(RCRegion* rgn, int idx, double start, double end)
{
    //setta il movimento per i successivi 128ms (16*8)
```

```
    //ottengo un puntatore alla struttura OCommandVectorData dalla RCRegion
    OCommandVectorData* cmdVecData = (OCommandVectorData*)rgn->Base();
    //setto i membri della struttura OCommandInfo
    OCommandInfo* info = cmdVecData->GetInfo(idx);
    info->Set(odataJOINT_COMMAND2, jointID[idx], ocommandMAX_FRAMES);
    //setto i valori dei frame della struttura OCommandData
    OCommandData* data = cmdVecData->GetData(idx);
    OJointCommandValue2* jval = (OJointCommandValue2*)data->value;

    double delta = end - start;
    //il movimento viene diviso in 16 frame di 8ms (ocommandMAX_FRAMES=16)
    for (int i = 0; i < ocommandMAX_FRAMES; i++) {
        double dval = start + (delta * i) / (double)ocommandMAX_FRAMES;
        jval[i].value = oradians(dval);
    }
}
```

Una volta settati i valori della struttura OCommandVectorData contenuta nella zona di memoria riferita da un' RCRegion sarà sufficiente inviare un messaggio contenente un riferimento a questa RCRegion al servizio Effector dell'oggetto di sistema OVirtualRobotComm per far assumere ai joint dell'Aibo nei successivi n frame (nell'esempio ocommandMaxFrame = 16) i valori precedentemente settati.

nel file MoveLegs.cc

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