AIBO Sony – Oggetto MoveLegs

Alessandro Quattro
Laboratory of Applied Intelligent Systems (AIS-Lab)
http://ais-lab.dsi.unimi.it
Version 2.6 – 10.Gennaio.2005

Sommario

1. L’oggetto MoveLegs

1.1 Descrizione

È un oggetto OPEN-R che si occupa della gestione del movimento delle zampe dell’Aibo.

Accetta dei comandi sottoforma di stringhe e comunica con l’oggetto di sistema OVirtualRobotComm per far eseguire al robot i movimenti corrispondenti al comando ricevuto.

L’oggetto MoveLegs accetta i seguenti comandi:

```
"WAKE_UP"
"STANDUP"
"ELBOWLK"
"STPEWLK"
"STRTWLK"
"STOPWLK"
"TRNRGHT"
"STP_T_R"
"TRNLEFT"
"STP_T_L"
"SITDOWN"
"LAYDOWN"
```

Al ricevimento del comando "WAKE_UP" attiva i motori dell’Aibo e lo fa muovere fino al raggiungimento della posizione a cuccia.

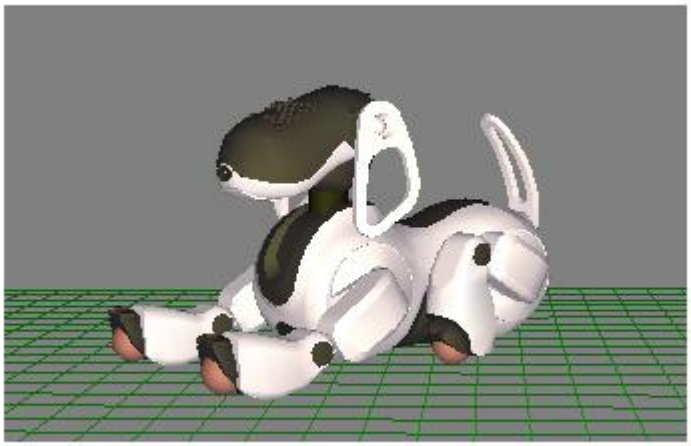


Fig.2 - Posizione a cuccia.

Al ricevimento del comando "STANDUP" fa alzare il robot.

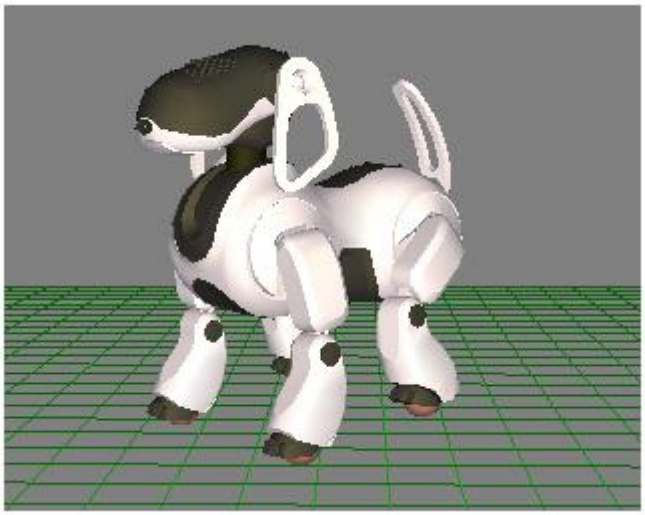


Fig.3 - Posizione in piedi.

Al ricevimento del comando "ELBOWLK" fa camminare il robot sui gomiti delle zampe anteriori, fino a quando non riceve il comando "STPEWLK".

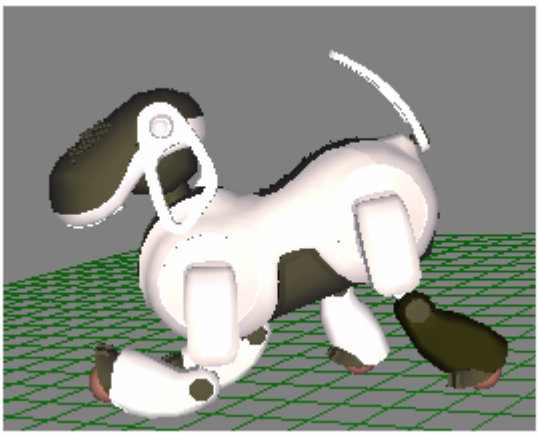


Fig.4 - Cammino sui gomiti.

Al ricevimento del comando "STRTWLK" fa camminare il robot sulle quattro zampe, fino a quando non riceve il comando "STOPWLK".

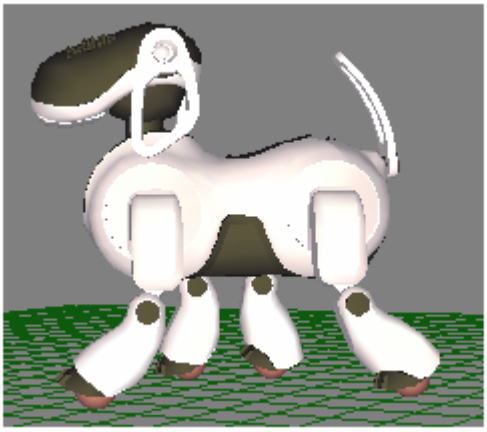


Fig.5 - Cammino sulle zampe.

Al ricevimento del comando "STPEWLK" o del comando "STOPWLK" l'Aibo si rimetterà nella posizione in piedi.

Al ricevimento del comando "TRNRGHT" fa girare il robot verso destra fino al ricevimento del comando "STP_T_R" che farà tornare il robot nella posizione in piedi.

Al ricevimento del comando "TRNLEFT" fa girare il robot verso sinistra fino al ricevimento del comando "STP_T_L" che farà tornare il robot nella posizione in piedi.

Al ricevimento del comando "SITDOWN" fa sedere l' Aibo.

Fig.6 – Posizione seduto.

Al ricevimento del comando "LAYDOWN" fa muovere l' Aibo fino al raggiungimento della posizione a cuccia.

1.2 Utilizzo

Utilizzare MoveLegs è molto semplice.

È sufficiente connettere un servizio subject dell'oggetto che intende utilizzare MoveLegs al servizio observer "ReceiveCommand" dell'oggetto MoveLegs.

Successivamente quando si vorrà far muovere il robot bisognerà inviare una delle stringhe precedentemente elencate al servizio subject precedentemente connesso.

In un'applicazione di test è stato utilizzato l'oggetto Commander per inviare dei comandi all'oggetto MoveLegs. L'oggetto Commander è stato dotato di un servizio SendLegsCommand per inviare comandi a MoveLegs.

```
void
Commander::WakeUp()
{
    char str[8];
    strcpy(str,"WAKE_UP");
    subject[sbjSendLegsCommand]->SetData(str, sizeof(str));
    subject[sbjSendLegsCommand]->NotifyObservers();
}
```

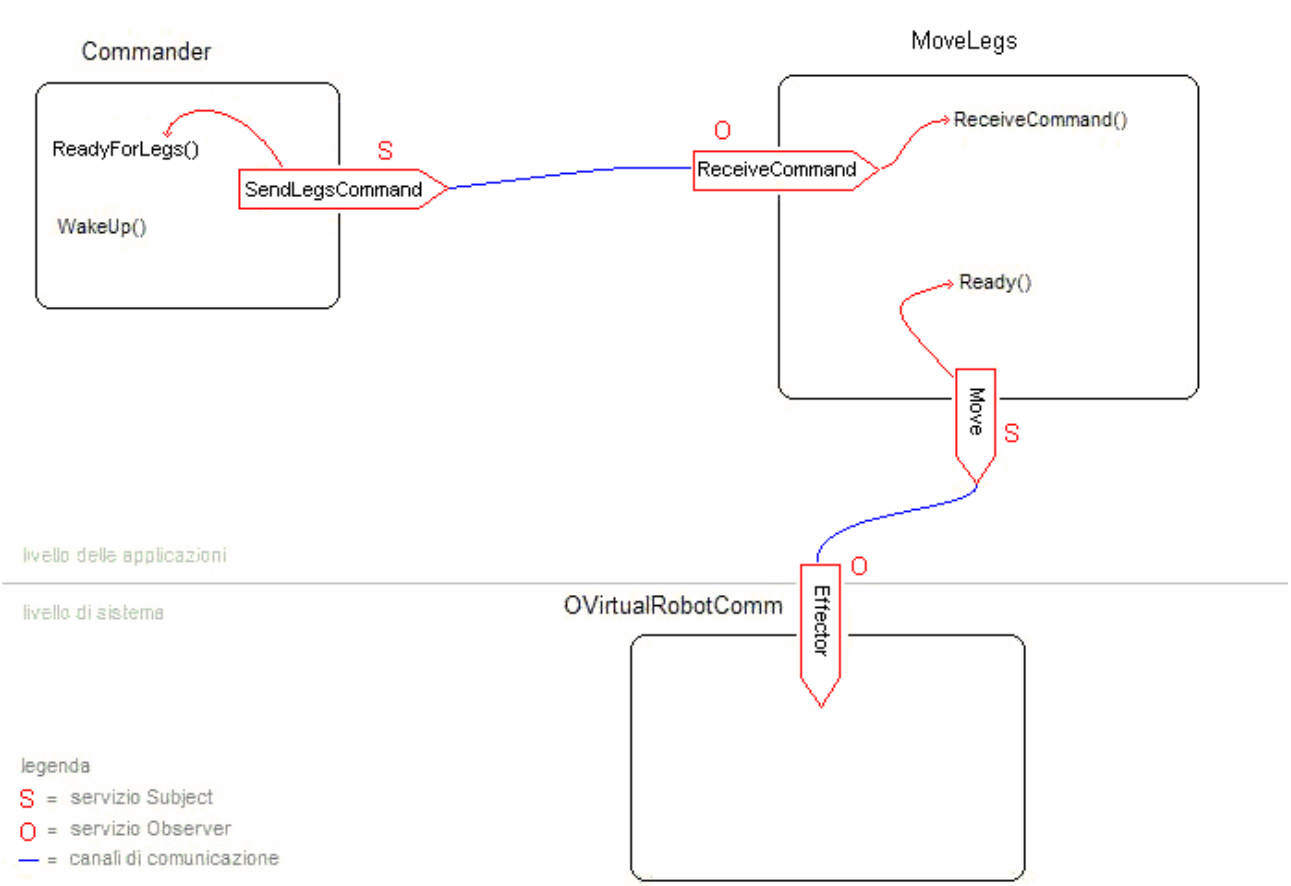


Fig.7 - Applicazione di test.

Bisogna sottolineare che i comandi disponibili dovranno essere chiamati necessariamente nell'ordine.

Infatti perché il robot possa muoversi è necessario che i motori siano accesi e quindi "WAKE_UP" dovrà essere il primo comando da inviare.

"STANDUP" dovrà essere chiamato solo se l'Aibo si trova nella posizione a cuccia.

"ELBOWLK", "STRTWLK", "TRNRGHT", "TRNLEFT", potranno essere chiamati solo se il robot si trova nella posizione in piedi.

Avrà senso chiamare "STPEWLK" o "STOPWLK" solo se l'Aibo sta camminando rispettivamente sui gomiti o sulle quattro zampe, così come avrà senso chiamare "STP_T_R" o "STP_T_L" solo se l'Aibo sta girando rispettivamente a destra o a sinistra.

"SITDOWN" potrà essere chiamato solo se l'Aibo si trova nella posizione in piedi.

"LAYDOWN" potrà essere chiamato solo se l'Aibo si trova nella posizione in piedi o nella posizione seduto.

Comunque nel caso venga inviato un comando nell'ordine sbagliato questo non verrà eseguito e l'oggetto MoveLegs si rimetterà in attesa di un nuovo comando.

1.3 Funzionamento

Nel file MoveLegs.h sono state definite delle posizioni raggiungibili dal robot e per ciascuna di queste le angolazioni dei 12 joint.

Sono stati anche definiti i diversi stati in cui può trovarsi il robot: fermo con i motori spenti (MLS_IDLE), fermo nella posizione a cuccia (MLS_SLEEPING), fermo nella posizione seduto (MLS_SITTING), fermo nella posizione in piedi (MLS_STANDINGONBACK) oppure in movimento da una posizione a un'altra (es. MLS_ELBOW_WALK1_TO_ELBOW_WALK1A). Per ogni stato che prevede il movimento tra due posizioni è stato definito il numero di blocchi di

frame da 128ms in cui questo movimento deve essere diviso (es. ELBOW_WALK1_TO_ELBOW_WALK1A_MAX_COUNTER).

nel file MoveLegs.h

```
…
//elenco di posizioni raggiungibili dal robot
...
const double ELBOW_WALK1_ANGLE[] = {
    -20,    // RFLEG J1
    4,      // RFLEG J2
    100,    // RFLEG J3

    20,     // LFLEG J1
    4,      // LFLEG J2
    90,     // LFLEG J3

    -20,    // RRLEG J1
    4,      // RRLEG J2
    80,     // RRLEG J3

    -10,    // LRLEG J1
    4,      // LRLEG J2
    50      // LRLEG J3
};

const double ELBOW_WALK1A_ANGLE[] = {
    -20,    // RFLEG J1
    4,      // RFLEG J2
    100,    // RFLEG J3

    20,     // LFLEG J1
    4,      // LFLEG J2
    80,     // LFLEG J3

    -50,    // RRLEG J1
    4,      // RRLEG J2
    80,     // RRLEG J3

    -10,    // LRLEG J1
    4,      // LRLEG J2
    50      // LRLEG J3
};
…

...
//insieme di stati in cui può trovarsi il robot
enum MoveLegsState {
    MLS_IDLE,
    ...
    MLS_WALK1_TO_WALK1A,
    ...
};

...
static const int ELBOW_WALK1_TO_ELBOW_WALK1A_MAX_COUNTER          = 2;  // 128ms * 2 = 256ms
...
    double START_ANGLE[12];
    double END_ANGLE[12];
    int MAX_COUNTER;

    MoveLegsState     moveLegsState;
…
```

I diversi stati in cui può trovarsi il robot possono essere paragonati a quelli di un automa a stati finiti (fig.8).

La transizione da uno stato a un altro è attivata da un messaggio.

Il movimento da eseguire dipende dal messaggio che è arrivato e dallo stato in cui si trova il robot.

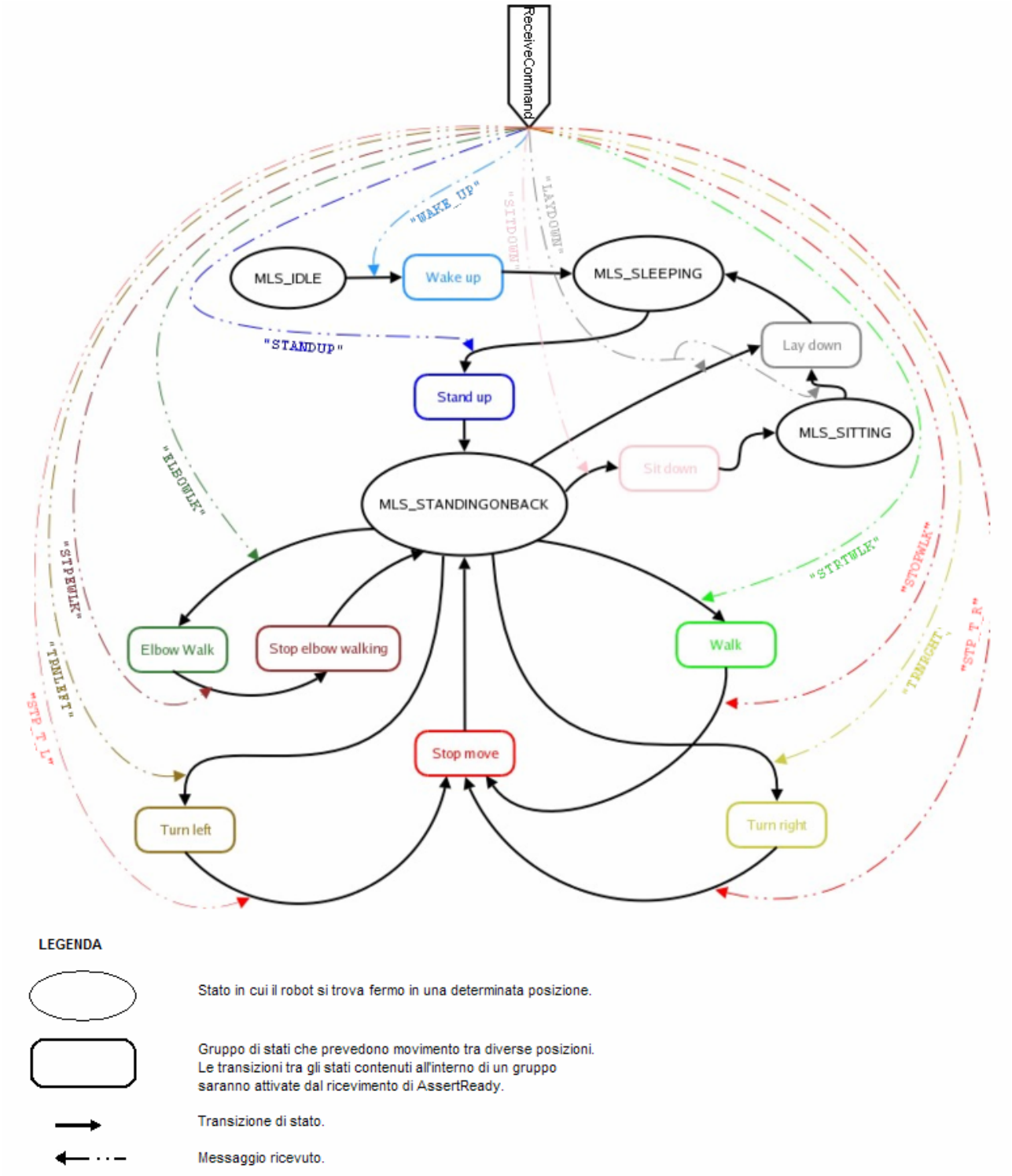


Fig.8 – Stati di MoveLegs (per una visione più dettagliata vedi allegato I).

Il nostro oggetto può ricevere 2 tipi di messaggi: una stringa attraverso il servizio observer “ReceiveCommand” o un AssertReady dall’oggetto di sistema OVirtualRobotComm attraverso il servizio subject “Move”.

Al ricevimento di un messaggio di tipo stringa attraverso il servizio observer “ReceiveCommand” viene chiamata la funzione ReceiveCommand().

Al ricevimento di un AssertReady dall’oggetto di sistema OVirtualRobotComm attraverso il servizio subject “Move” viene chiamata la funzione Ready().

La funzione ReceiveCommand() riceve una stringa.

Se questa stringa corrisponde a un comando valido controlla anche lo stato in cui si trova l'oggetto.

Se l'oggetto si trova in uno stato compatibile con il comando rappresentato dalla stringa viene chiamata la funzione preposta all'esecuzione di quel comando.

Ad esempio se viene ricevuta la stringa "STANDUP" e il robot si trova nello stato MLS_SLEEPING verrà chiamata la funzione StandUp().

```
...
char* cmd;      //comando da eseguire
char* rcvcmd;   //comando ricevuto
...

void
MoveLegs::ReceiveCommand(const ONotifyEvent& event)
{
    rcvcmd = (char*)event.Data(0); //stringa ricevuta
    //se la stringa ricevuta è compatibile con lo stato in cui si trova il robot
    //la copio in cmd (comando da eseguire) ed eseguo il comando
    //se no invio un ASSERT_READY all'oggetto che ha inviato il comando senza eseguirlo
    ...

    } else if(strcmp(rcvcmd,"STANDUP")==0){
        if(moveLegsState == MLS_SLEEPING){//si alza
            cmd = rcvcmd;
            positionsSetted = false;
            StandUp();
        } else {                          //non è nello stato SLEEPING
            observer[event.ObsIndex()]->AssertReady();
        }

    } else if(strcmp(rcvcmd,"ELBOWLK")==0){
...
```

La funzione Ready() non fa altro che richiamare la funzione preposta all'esecuzione dell'ultimo comando ricevuto. Essa viene chiamata ad ogni ricevimento di un AssertReady dall'oggetto di sistema OVirtualRobotComm, potrà quindi essere chiamata ripetutamente durante la transizione dallo stato iniziale allo stato finale del movimento previsto da un comando.

Perciò anche la funzione preposta a eseguire il comando ricevuto potrà essere chiamata più volte fino al raggiungimento dello stato finale previsto dal comando.

```
MoveLegs::Ready(const OReadyEvent& event)
{
...
        //STANDUP
        else if(strcmp(cmd,"STANDUP")==0){
            StandUp();
        }//end STANDUP
...
}
```

Nelle funzioni preposte a eseguire i comandi ricevuti (es. StandUp()) viene controllato in quale dei possibili stati si trova il robot.

Se lo stato del robot è appena cambiato e corrisponde a uno stato che prevede il movimento tra due posizioni vengono settati i valori degli array rappresentanti le posizioni di inizio e di fine movimento e della variabile che rappresenta il numero di blocchi da 16 frame in cui deve essere diviso il movimento stesso.

Poi, sempre se lo stato in cui si trova il robot corrisponde a uno stato che prevede il movimento tra due posizioni, viene chiamata la funzione MoveJoints().

Quando viene raggiunta la posizione finale la funzione MoveJoints() restituisce il valore MOVING_FINISH e lo stato del robot viene cambiato con quello relativo allo stato successivo.

Nella funzione MoveJoints() il movimento viene diviso in intervalli regolari, viene cercata l'RCRegion libera, con la funzione FindFreeRegion(), e poi chiamata la funzione SetJointValue() che provvede a valorizzare la struttura OCommandVectorData dell'RCRegion libera appena trovata.

Infine viene inviato il messaggio all'oggetto OVirtualRobotComm.

Quando questo riceve il messaggio incrementa di 1 il reference counter dell'RCRegion, processa il messaggio muovendo i joint e poi decrementa di 1 il reference counter liberando l'RCRegion.

Una volta eseguito il movimento l'oggetto di sistema OVirtualRobotComm invia un Assert Ready all'oggetto MoveLegs.

Verrà quindi chiamata nuovamente la funzione Ready() che richiamerà a sua volta la funzione preposta all'esecuzione dell'ultimo comando ricevuto. Questo ciclo si interromperà quando sarà stato raggiunto uno stato che non prevede movimento, cioè uno dei 4 stati che corrispondono alle posizioni fermo nella posizione a cuccia (MLS_SLEEPING) , fermo nella posizione seduto (MLS_SITTING), fermo nella posizione in piedi (MLS_STANDINGONBACK) oppure fermo con i motori spenti (MLS_IDLE).

Come esempio si riporta il codice della funzione privata StandUp() che fa passare il robot dallo stato MLS_SLEEPING (corrispondente alla posizione fermo nella posizione a cuccia) allo stato MLS_STANDINGONBACK (corrispondente alla posizione fermo nella posizione in piedi) passando attraverso gli stati intermedi MLS_SLEEPING_TO_STANDINGFRONT e MLS_STANDINGFRONT_TO_STANDINGONBACK.

Si riporta anche il codice delle funzioni private MoveJoints(), FindFreeRegion() e SetJointValue() .

```
void
MoveLegs::StandUp()
{
    if (moveLegsState == MLS_SLEEPING_TO_STANDINGFRONT) {
        if(!positionsSetted) {
            for (int i = 0; i < NUM_JOINTS; i++) {
                START_ANGLE[i] = SLEEPING_ANGLE[i];
                END_ANGLE[i] = STANDINGFRONT_ANGLE[i];
            }
            MAX_COUNTER = SLEEPING_TO_STANDINGFRONT_MAX_COUNTER;
            positionsSetted = true;//nuove posizioni settate
        }
        MovingResult r = MoveJoints();
        if (r == MOVING_FINISH) {
            moveLegsState = MLS_STANDINGFRONT_TO_STANDINGONBACK;//stato successivo
            positionsSetted = false;//posizioni non settate (vengono settate all'inizio della
funzione)
            counter = -1;         //azzero il contatore del frame raggiunto
        }
    } else if (moveLegsState == MLS_STANDINGFRONT_TO_STANDINGONBACK) {
        if(!positionsSetted){
            //setta le nuove posizioni
            for (int i = 0; i < NUM_JOINTS; i++) {
                START_ANGLE[i] = STANDINGFRONT_ANGLE[i];
                END_ANGLE[i] = STANDINGONBACK_ANGLE[i];
            }
            MAX_COUNTER = STANDINGFRONT_TO_STANDINGONBACK_MAX_COUNTER;
            positionsSetted = true;//nuove posizioni settate
        }
        MovingResult r = MoveJoints();
        if (r == MOVING_FINISH) {
            moveLegsState = MLS_STANDINGONBACK;//stato successivo
            positionsSetted = false;//posizioni non settate (vengono settate all'inizio della
funzione)
            counter = -1;         //azzero il contatore del frame raggiunto

            subject[sbjMove]->ClearBuffer();

            observer[obsReceiveCommand]->AssertReady();
        }
    }
}
```

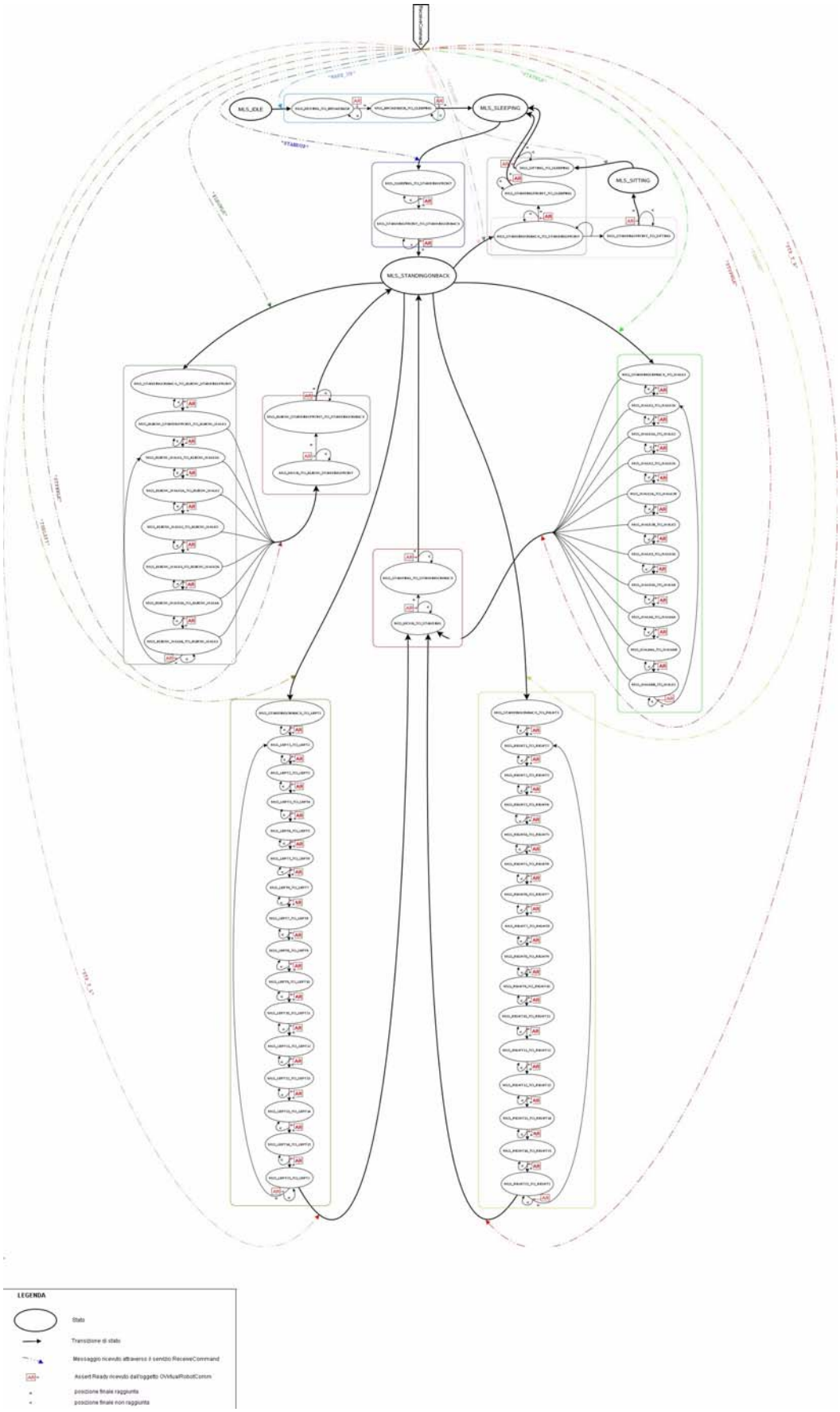
```
MovingResult
MoveLegs::MoveJoints()
{
    static double start[NUM_JOINTS];
    static double delta[NUM_JOINTS];
    double ndiv = (double)MAX_COUNTER;
    if (counter == -1) {
        for (int i = 0; i < NUM_JOINTS; i++) {
            start[i] = START_ANGLE[i];
            //il movimento viene diviso in delta costanti(velocità costante)
            delta[i] = (END_ANGLE[i] - start[i]) / ndiv;
        }
        counter = 0;
    }
    RCRegion* rgn = FindFreeRegion();
    for (int i = 0; i < NUM_JOINTS; i++) {
        SetJointValue(rgn, i, start[i], start[i] + delta[i]);
        start[i] += delta[i];
    }
    subject[sbjMove]->SetData(rgn);
    subject[sbjMove]->NotifyObservers();
    counter++;
    return (counter == MAX_COUNTER) ? MOVING_FINISH : MOVING_CONT;
}

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


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);
    }
}
```

Allegato 1.


MLS_IDLE
MLS_SLEEPING
MLS_SITTING
MLS_STANDINGONBACK
LEGENDA
Stato
Transizione di stato
Messaggio ricevuto attraverso il servizio ReceiveCommand
Assert Ready ricevuto dall'oggetto OVMutualRobotComm
posizione finale raggiunta
posizione finale non raggiunta