Diseño e implementación de movimientos en el Sony Aibo ESR-7

Los tiros y movimientos distintos a la caminata son ejecutadas por el AIBO como una secuencia de posiciones previamente establecida por el equipo de programación. Cada una de dichos movimientos debe cumplir con ciertos criterios.

En primer lugar, la primera y la última posición en el movimiento debe ser idéntica a la posición de inicio y final del ciclo de caminata o muy parecida a dicha posición. Esto garantiza que el movimiento pueda ser ejecutado en cualquier momento ya sea que el AIBO se encuentre estático, este caminando o al final de otro movimiento.

Segundo, dado que algunos de los movimientos implican el movimiento abrupto de la cabeza, el movimiento tiene que ser breve a fin de garantizar que tras su ejecución no se pierdan demasiados cuadros de la cámara y con ellos información valiosa sobre el estado de juego. La mayoría de los movimientos, con excepción, del festejo, tienen un tiempo de ejecución del orden de dos segundos o inferior.

Tercero, los movimientos deben ser tales que no comprometan la integridad del robot, esto implica que la distancia entre cada una de las posiciones no debe ser muy grande pues ante ese escenario uno o mas motores podrían resultar dañados o, con el objetivo de evitar dichos daños, el robot podrá apagarse.

Diseño de movimientos.

Con el fin de facilitar diseñar los movimientos, se utiliza la herramienta *MEdit for ERS-7.1.0.7* proporcionada por Sony. Este software permite observar la posición de las extremidades y cabeza del AIBO para cada configuración válida en los actuadotes del mismo.

MEdit cuanta con cuatro ventanas, de ellas solo tres son de interés para el diseño de movimientos. Una de ellas, llamada *Pose Control* nos muestra el valor del ángulo en que se encuentra cada uno de los actuadotes de la cabeza, las 4 patas y la cola. Los valores desplegados por esta pantalla están dados en grados.

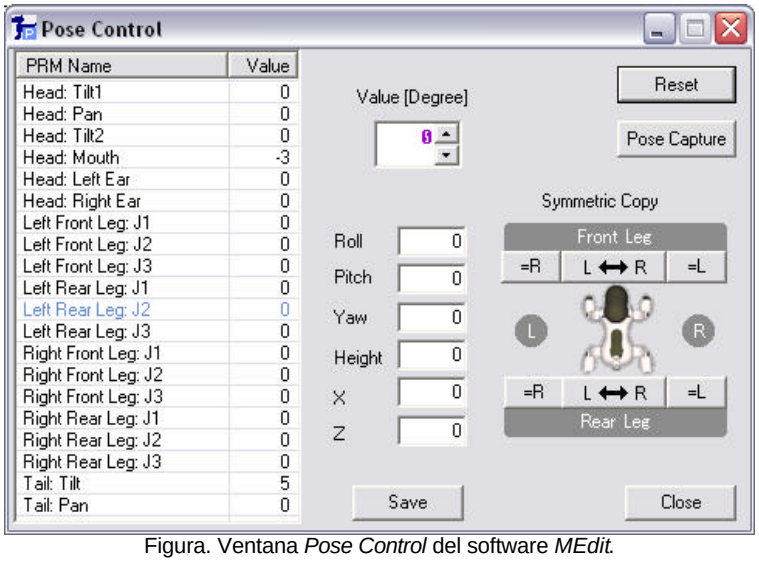


Figura. Ventana *Pose Control* del software *MEdit*.

La ventana principal de programa despliega gráficamente el efecto de las modificaciones de los ángulos en la ventana *Pose Control*. Tambien es posible, con ayuda del mouse modificar la posición de los motores en esta pantalla y ver el valor de ellos en la pantalla Pose Control. Es

posible, utilizando los controles ubicados en estas dos primeras ventana guardar los valores de los motores en un archivo para su uso posterior. Esto es muy importante dado que, como se mencionó, los movimientos están basados en la ejecución secuencial de varias posiciones.

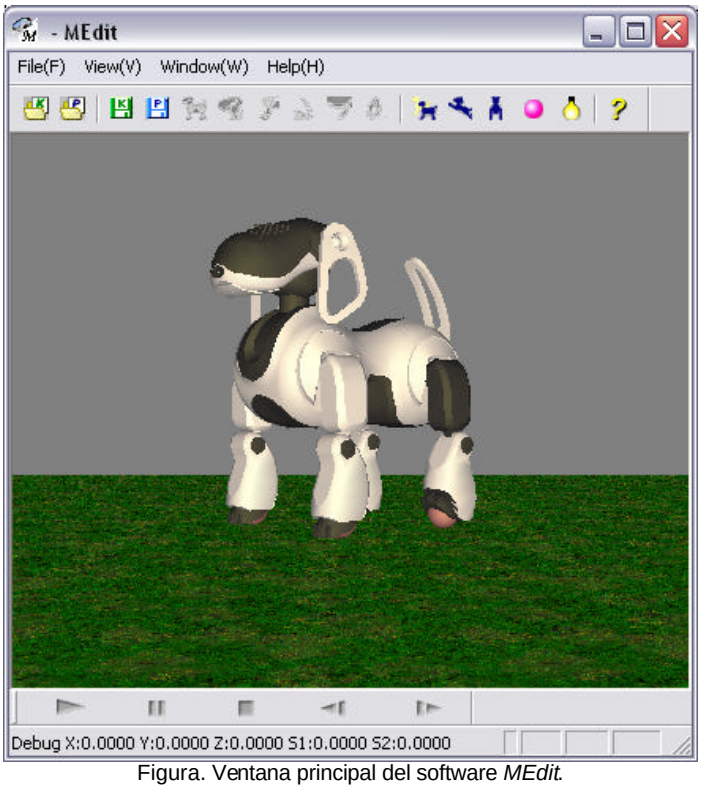


Figura. Ventana principal del software *MEdit*.

La tercera ventana de interés se llama *Motion Control*, en ella podemos cargar varias posiciones guardadas previamente y fijar los tiempos de transición entre una y otra para obtener una animación del resultado de la secuencia de posiciones.

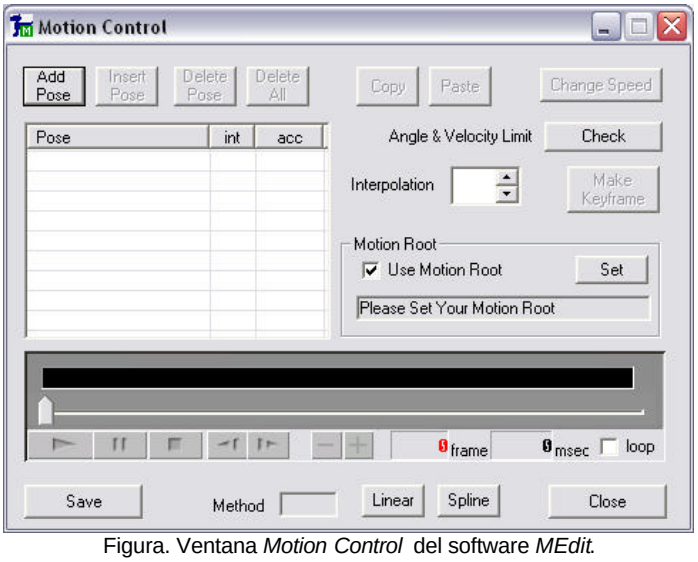


Figura. Ventana *Motion Control* del software *MEdit*.

La siguiente figura muestra las 8 posiciones que conforman movimiento utilizado como tiro a gol en la versión EK2007. Cada una de ellas esta almacenada como un archivo *PSE*. Al cargarlas en la ventana *Motion Control*, es posible observar la simulación de la ejecución de movimiento. Cabe aclarar que *MEdit* no simula colisiones con objetos ni gravedad, sino únicamente movimientos en los motores del AIBO, por tanto no es posible observar los efectos reales de cada movimiento sobre el equilibrio del robot.

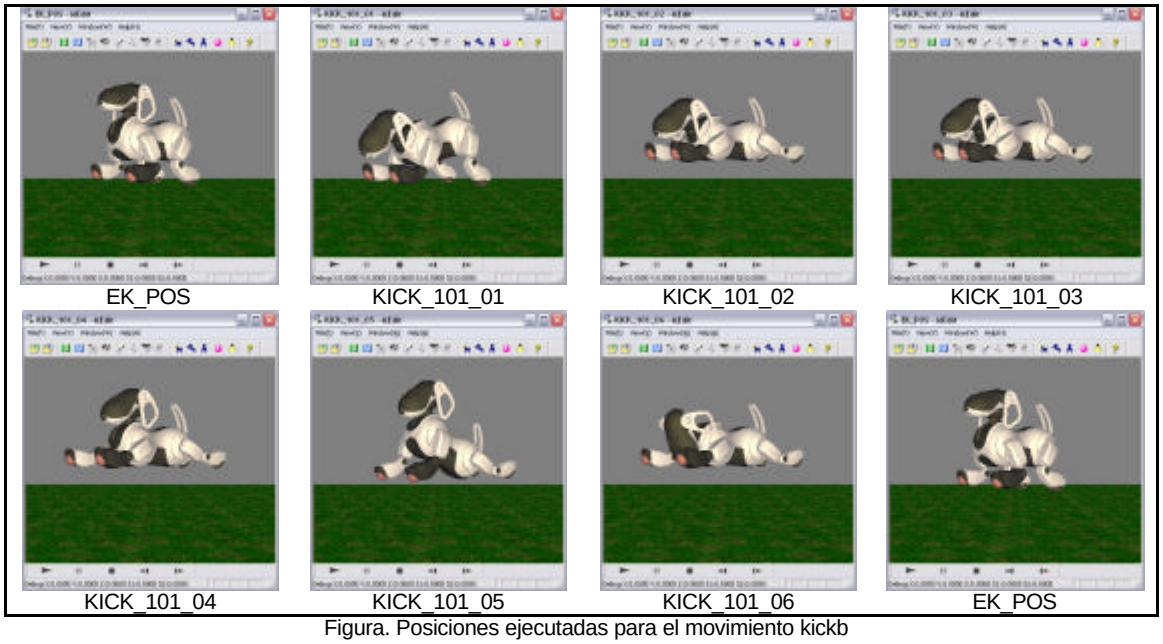


Figura. Posiciones ejecutadas para el movimiento kickb

Una vez creada la secuencia, el movimiento puede ser almacenado en un archivo *KFM*, dicho archivo es una lista con los archivos *PSE* a ejecutar y el intervalo entre cada uno de ellos por lo que es fundamental conservar todos los archivos de las posiciones.

EK2006 - Exportación de los movimientos al archivo MOTION7.ODA.

Para al versión EK2006, los movimientos eran exportados al robot mediante la generación de un archivo *MTN* que posteriormente era incorporado al objeto ODA para, por medio de Monet realizar la ejecución de los movimientos; para ello, en *MEdit* se utiliza la opción File(F) – Export Mtn All(A) para generar el archivo de extensión *MNT*.

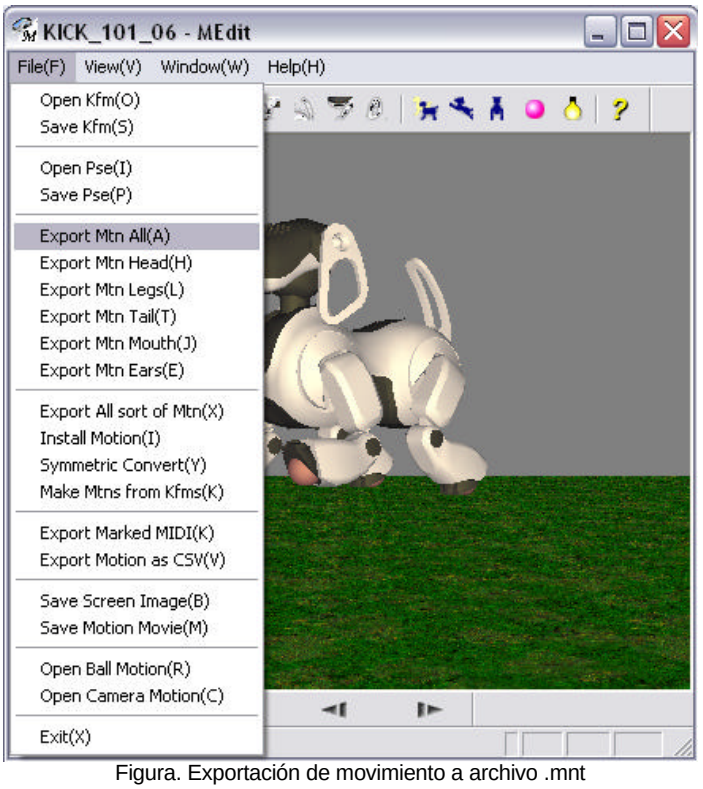


Figura. Exportación de movimiento a archivo .mnt

El archivo debe ser colocado en la carpeta que contiene el archivo ODA utilizado por Monet para realizar movimientos. Suponiendo que la carpeta con el código EK2006 esta en C.\cygwin, entonces el archivo MTN que se desea incorporar debe ser colocado en C:\cygwin\EK2006\util.

Ahora, desde una ventana de terminal de cywin se debe acceder al dicho directorio y ejecutar el script oda para incorporar el movimiento al archivo ODA.

```
oda –a motion7.oda <NombreArchivo>.mtn
```

Una vez hecho esto, el archivo MOTION7.ODA debe copiarse a C:\cygwin\EK2006\MS\OPEN-R\MW\DATA\P\ERS-7. En C:\cygwin\EK2006\MS\OPEN-R\MW\CONF\ERS-7 se debe agregar al archivo MONETCMD.CFG una línea asignando el archivo a un número de identificación, por ejemplo 60.

```
##########################################################################
#
# MOVIMIENTO KICKB
#
##########################################################################
60 1 0
monetagentMTN a_KickB -1
```

Ya en el código que define el comportamiento del robot, se pude ejecutar cualquier movimiento cargado al archivo ODA y registrado en MONETCMD.CFG mediante la función *MoNetExecute(NoID)*, donde NoID es el numero con que se registro el archivo MNT en MONETCMD.CFG.

Este método fue abandonado debido a las limitaciones en velocidad de movimientos del software *MEdit*. El método utilizado para la versión EK2007 es descrito a continuación.

EK2007 – Definición de movimientos en EKMotion

Para la versión EK2007 se decidió utilizar los valores de los ángulos en los motores directamente sobre los actuadotes a través de las funciones proporcionadas por el API de OPEN-R

En la carpeta del código correspondiente al subsistema EKMotion se encuentran los archivos que definen como se realizan los movimientos del robot. El archivo EKMoves.h es un *header* de C/C++ en el que se encuentran definidos arreglos con los valores deseados para cada una de las posiciones utilizadas en un movimiento.

Los arreglos son de tipo *double* y están definidos como constantes, cada uno de ellos contiene 16 valores, uno para cada uno de los motores de importancia siguiendo el siguiente formato:

```
const double <NombrePosicion>[] = {
  #Tilt2,
  #Pan,
  #Tilt1,
  #Mouth,
  #LeftFrontLegJ1,
  #LeftFrontLegJ2,
  #LeftFrontLegJ3,
  #LeftRearLegJ1
  #LeftRearLegJ2
```

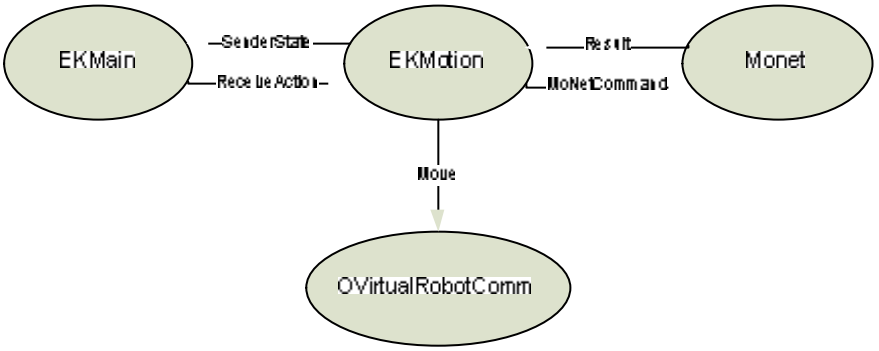
```
    #LeftRearLegJ3
    #RightFrontLegJ1
    #RightFrontLegJ2
    #RightFrontLegJ3
    #RightRearLegJ1
    #RightRearLegJ2
    #RightRearLegJ3
};
```

Figura. Formato del arreglo que define una posición.

Aunque los valores que se establecen en los actuadotes deben darse en radianes, por comediada en la definición de los arreglos se utilizan grados, de esta forma se pueden definir arreglos rápidamente con los posiciones diseñadas en *MEdit* copiando los valores directamente de la ventana *Pose Control* en el orden adecuado. El nombre de cada arreglo debe incluir un número de identificación de su posición en la secuencia de ejecución, por ejemplo: KICK_101_01, en este caso se refiere a la primera posición del movimiento KICK_101.

EKMotion

EKMotion es un componente del sistema EK2007 y versiones posteriores que se encarga de gestionar los movimientos que realiza el robot. EKMotion no cuenta con una forma directa de indicar a los actuadores el valor que deben tomar sino que utiliza al objeto OVirtualRobotComm para establecer los valore que se desea que el robot tome en cada actuador.



```
    double   y_walk;
    double   ang_walk;
    double   x_head;
    double   y_head;
    double   z_head;
    double   mouth;
};
```

Figura. Estructura EKMotionData

En EKMotion, el servicio ReceiveAction atiende el envió de la estructura desde EKMain y procesa la información arrancando el modulo Push en EKMotion.cc

En Push, si no se está ejecutando otro movimiento, a los datos recibidos se les aplica un cast a tipo EKMotionData para ser accesibles desde el resto del código. Por medio del valor del campo command de la estructura, se verifica que no se trate una instrucción de caminata u otros movimientos de carácter reservado. Cuando se cumplen dichas precondiciones el número del identificador (motData.command) del movimiento a realizar se envía a la función MoveToPos.

```
void
EKMotion::Push(const ONotifyEvent& event) {
        if(movingState != MNT_SPECIAL_MOVE) {
                motData = (EKMotionData)*(const EKMotionData *)event.Data(0);
                if (motData.command == 1000)
                {
                        …
                }
                else
                        if (motData.command == 0) {
                                …
                        } else {
                                if (motData.command == 998) {
                                        ….
                                } else {
                                        if (motData.command == 999) {
                                                ….
                                        } else {
                                                alineate1 = false;
                                                alineate2 = false;
                                                caminaConPelota = false;
                                                movingState = MNT_SPECIAL_MOVE;
                                                MoveToPos(motData.command);
                                        }
                                }
                        }
                }
        observer[event.ObsIndex()]->AssertReady();
}
```

Figura. Código parcial de la funcion Push en EKMotion.cc

MoveToPos recibe el valor del identificador del movimiento a ejecutar. La función SendJointCommandVector es invocada hacia el final de MoveToPos, dicha función es la encargada de enviar al objeto OVirtualRobotComm los datos necesarios para que el robot cambie de posición. Antes de invocar dicha función, es necesario colocar en el arreglo joint_values los valores de las posiciones que conforman el movimiento.

Como se mencionó anteriormente, el archivo EKMoves.h es una librería de vectores que definen los valores, en grados, de los actuadores para distintas posiciones. Esas posiciones en secuencia conforman un movimiento. El identificador de movimiento (cmd) define cual de una seria de funciones invocables sera ejecutada, cada una de esas funciones toma un conjunto de vectores del archivo EKMoves.h de acuerdo al movimiento que se desea realizar. En este caso, se analizará como se ejecuta el movimiento identificado por el número 102; para su ejecución se invoca la función Kickb.

```
MovingResult
EKMotion::MoveToPos(int cmd) {
        MovingResult result;
```

```
        subject[sbjMove]->ClearBuffer();
        switch (cmd) {
                case 110: result = Kickc();              break;
                case 101: result = Kicka();              break;
                case 102: result = Kickb();              break;
                case 103: result = Suelta();             break;
                case 105: result = Kick();               break;
                case 201: result = KickIzq();            break;
                case 301: result = KickDer();            break;
                case 401: result = Bloqueo();            break;
                case 402: result = BloqueoIzq();         break;
                case 403: result = BloqueoDer();         break;
                case 501: result = Festejo();            break;
                case 601: result = Paro();               break;
                case 701: result = Receptor();           break;
                case 702: result = PelotaReceptor();     break;
                case 901: result = TiroR();              break;
        }
        SendJointCommandVector();
        subject[sbjMove]->NotifyObservers();
        return result;
}
```

Figura. Código de la función MoveToPos en EKMotion.cc

Kickb, al igual que el resto de las funciones invocables desde MoveToPos, controla la transición entre cada una de las posiciones que forman el movimiento. Mediante la variable cont, se determina el número de ciclos entre una posición y otra; a menor número de ciclos, el cambio de una posición a otra del movimiento se efectuara en menor tiempo. Para el caso del tiro Kickb se utilizó una distancia de 50 ciclos entre cada una de las posiciones que lo integran.

La variable pos, por otro lado, es utilizada para determinar cual de las posiciones es la que se va a mandar a los actuadores. Para efector de facilitar la programación, se decidió dividir el establecimiento de una posición en dos partes, por un lado las patas y por el otro la cabeza. Para el caso de los tiros se utilizan dos funciones *MoveHeadKick* y *MoveLegsKick*; como su nombre lo indica, la primera se encarga de los 4 motores de interés en la cabeza y la segunda se encarga de los 12 actuadores de las patas.

```
MovingResult
EKMotion::Kickb() {
        MovingResult result;
        cont += 1;
        result = MOVING_CONT;
        switch (cont) {
                case 1:   pos = 3;
                          MoveHeadKick(pos);
                          MoveLegsKick(pos);
                          break;
                case 50:  pos = 4;
                          MoveHeadKick(pos);
                          MoveLegsKick(pos);
                          break;
                case 100: pos = 5;
                          MoveHeadKick(pos);
                          MoveLegsKick(pos);
                          break;
                case 150: pos = 6;
                          MoveHeadKick(pos);
                          MoveLegsKick(pos);
                          break;
                case 200: pos = 7;
                          MoveHeadKick(pos);
                          MoveLegsKick(pos);
                          break;
                case 250: pos = 8;
                          MoveHeadKick(pos);
                          MoveLegsKick(pos);
                          break;
                case 300: pos = 1;
                          MoveHeadKick(pos);
                          MoveLegsKick(pos);
```

```
                        break;
                case 310:       cont = 0;
                        result = MOVING_FINISH;break;
        }
        return result;
}
```

Figura. Código de la funcion Kickb en EKMotion.cc

Como se mencionó, los valores a establecer para cada actuador deben estar dados en radianes así que es necesario convertirlos a dichas unidades. En ambas funciones, se utiliza el parámetro n para determinar que posición será cargada a los arreglos temporales jointValue o leg_joints, según se trate de MoveHeadKick o MoveLegsKick, respectivamente.

```
void
EKMotion::MoveHeadKick(int n) {
        double jointValue[NUMBER_HEAD_JOINTS];
        switch (n) {
                case 1: for (int i = 0; i<4; i++)
                                jointValue[i] = (EK_POS[i]*PI)/180;
                        break;
                …
                case 4: for (int i = 0; i<4; i++)
                                jointValue[i] = (KICK_101_02[i]*PI)/180;
                        break;
                case 5: for (int i = 0; i<4; i++)
                                jointValue[i] = (KICK_101_03[i]*PI)/180;
                        break;
                case 6: for (int i = 0; i<4; i++)
                                jointValue[i] = (KICK_101_04[i]*PI)/180;
                        break;
                case 7: for (int i = 0; i<4; i++)
                                jointValue[i] = (KICK_101_05[i]*PI)/180;
                        break;
                case 8: for (int i = 0; i<4; i++)
                                jointValue[i] = (KICK_101_06[i]*PI)/180;
                        break;
                …
        }
        SetHeadJoints(jointValue);
        return;
}
void
EKMotion::MoveLegsKick(int n) {
        static double leg_joints[NUM_LEG_JOINT];
        switch (n) {
                case 1: for (int i = 0; i<12; i++)
                                leg_joints[i] = (EK_POS[i+4]*PI)/180;
                        break;
                …
                case 4: for (int i = 0; i<12; i++)
                                leg_joints[i] = (KICK_101_02[i+4]*PI)/180;
                        break;
                case 5: for (int i = 0; i<12; i++)
                                leg_joints[i] = (KICK_101_03[i+4]*PI)/180;
                        break;
                case 6: for (int i = 0; i<12; i++)
                                leg_joints[i] = (KICK_101_04[i+4]*PI)/180;
                        break;
                case 7: for (int i = 0; i<12; i++)
                                leg_joints[i] = (KICK_101_05[i+4]*PI)/180;
                        break;
                case 8: for (int i = 0; i<12; i++)
                                leg_joints[i] = (KICK_101_06[i+4]*PI)/180;
                        break;
                …
        }
        SetLegJoints(leg_joints);
        return;
}
```

Figura. Código parcial de las funciones MoveHeadKick y MoveLegsKick en EKMotion.cc

Una vez que se tiene los valores en radianes se deben establecer estos en el arreglo joint_values siguiendo el orden descrito de los arreglos en EKMoves.h. Como se decidio dividir la asignación en dos funciones, SetHeadJoints y SetLegJoints, se utilizan los arreglos HEAD_JOINT_INDEX y LEG_JOINT_INDEX para acceder al índice correcto en joint_values

```
const int HEAD_JOINT_INDEX[] = {0, 1, 2, 3};
const int LEG_JOINT_INDEX[] = {4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};
```

Figura. Declaración de los arreglos HEAD_JOINT_INDEX y LEG_JOINT_INDEX en EKMotion.h

La función SetJoint asigna el valor al arreglo joint_values en el indice que se le indique.

```
int
EKMotion::SetHeadJoints(double *x)
{
	for (int i = 0; i < NUMBER_HEAD_JOINTS; i++)
	{
		SetJoint(HEAD_JOINT_INDEX[i], x[i]);
	}
	return 1;
}

int
EKMotion::SetLegJoints(double *x)
{
	for (int i = 0; i < NUMBER_LEG_JOINTS; i++)
	{
		SetJoint(LEG_JOINT_INDEX[i], x[i]);
	}
	return 1;
}

void
EKMotion::SetJoint(int index, double val)
{
	if ((index >= 0) && (index < NUMBER_JOINTS))
	{
		joint_values[index] = int(1e6*val);
		send_joint_values[index] = true;
	}
	send_joints = true;
}
```

Figura. Funciones SetHeadKick y SetLegJoints y SetJoint en EKMotion.cc

Una vez que se tiene en joint_value los valores de la posición deseada, en MoveToPos la función SendJointCommandVector es invocada. SendJointCommandVector es la responsable de enviar a OVirtualRobotComm la información del movimiento a efectuar, dicha información es proporcionada mediante un objeto OCommandVectorData llamado cmdVecData que es almacenado en memoria y accesado por medio de apuntadores. Los apuntadores jvalue2 y jvalue4 de tipo OJointCommandValue2 y OJointCommandValue4 acceden a los valores del vector cmdVecData. Para controlar la ejecución del movimiento, en este punto se realiza la interpolación entre la posición que guarda el robot y la posición a la que se desea llegar. Cuando se tiene todos los datos para todos los motores, el servicio *Move* es invocado para enviar a OVirtualRobotComm el vector de datos, en este caso se envía el apuntador rgn que hace referencia a la localidad de memoria donde se almaceno cmdVecData.

```
void
EKMotion::SendJointCommandVector() {
  RCRegion* rgn = NULL;
  // Find free joint command vector
  rgn = FindFreeRegion();
  OCommandVectorData* cmdVecData = (OCommandVectorData*)rgn->Base();
  int numData = 0;
  for (int i = 0; i < NUMBER_JOINTS; i++) {
    OCommandInfo* info = cmdVecData->GetInfo(i);
    OCommandData* data = cmdVecData->GetData(i);
```

```
        if (send_joint_values[i] == false) {
            info->numFrames = 0;
        } else {
            info->numFrames = number_frames;
            if (i < NUMBER_JOINT2) {
                OJointCommandValue2* jvalue2 = (OJointCommandValue2*)data->value;
                for (int j = 0; j < number_frames; j++) {
                    jvalue2[j].value = joint_values_current[i] +
                        (j+1)*(joint_values[i]-joint_values_current[i])/(number_frames);
                }
            } else {
                // Joint4 type
                OJointCommandValue4* jvalue4 = (OJointCommandValue4*)data->value;
                for (int j = 0; j < number_frames; j++) {
                    jvalue4[j].value = joint_values_current[i] +
                        (j+1)*(joint_values[i]-joint_values_current[i])/(number_frames);
                }
            }
            joint_values_current[i] = joint_values[i];
            numData = i;
        }
        send_joint_values[i] = false;
    }
    cmdVecData->SetNumData(numData+1);
    subject[sbjMove]->SetData(rgn);
    send_joints = false;
}
```

Figure. Función SendJointCommandVector en EKMotion.cc

Bibliografía y referencias:

'How To Make AIBO Do Tricks'. Wijbenga.2004.
http://www.ai.rug.nl/~gert/as/download/bachelor/thesis_monet.pdf

'EagleKnights 2007: Four-Legged League, Team Description Paper'. Weitzenfeld, Martínez, Muciño, Serrano, Ramos & Rivera. 2007.
ftp://ftp.itam.mx/pub/alfredo/PAPERS/ektdp2007_4L.pdf

Código:

EK2007. ITAM EagleKnights. 2007.