UNIVERSIDAD NACIONAL AUTONOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERIA DE LA COMPUTACIÓN

DESARROLLO DE UN SISTEMA DE CAPTURA DE MOVIMIENTO PARA EL CUERPO HUMANO

TESIS
QUE PARA OBTENER EL GRADO DE:
MAESTRO EN INGENIERIA

PRESENTA:
HERNANDO ORTEGA CARRILLO

DIRECTOR DE LA TESIS: DR. JESÚS SAVAGE CARMONA

MÉXICO D.F. **2001**

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

Resumen

En la actualidad existe un notable incremento de la capacidad de procesamiento y despliegue gráfico de las computadoras personales. Este hecho, aunado a la aparición de tecnologías cada vez más depuradas para el desarrollo de ambientes virtuales (que incluso funcionan en Internet) como: java3D, VMRL y QTVR entre otras, nos permite prever la expansión de los ambientes virtuales. Este tipo de sistemas permitirán a las personas interactuar con fines colaborativos o de entretenimiento en ambientes cuyo comportamiento se apegan a la realidad. Esta tecnología permitirá, (y a su vez requerirá) cada vez de una mayor interactividad de los usuarios, y es en este punto donde las interfaces tradicionales comenzarán a ser obsoletas.

Una posible solucion a este problema es el desarrollo de nuevas interfaces que contemplen la captura de los movimientos del usuario. El proceso de captura de movimiento consiste en el registro de la posición y orientación de objetos a través del tiempo. En lo particular este proceso puede ser aplicado por una computadora para registrar los movimientos de una persona. Esta tecnología es utilizada en diferentes campos como: dispositivos de entrada e interfaces, animación, realidad virtual y control de robots entre muchos otros. El tema de esta tesis se enfoca precisamente a los dispositivos de entrada y las interfaces hombre/maquina. La propuesta es desarrollar un dispositivo de entrada "genérico" que permita a la computadora registrar expresiones corporales para que posteriormente sean interpretadas en el contexto de una interfaz (esto ultimo se ejemplifica a lo largo del trabajo, pero no se desarrolla en extenso).

La flexibilidad que ofrece este dispositivo permitirá al usuario tener una interacción más directa con la computadora ya que ésta podrá registrar todos sus movimientos y entonces interpretarlos de acuerdo a las necesidades de cada aplicación. Con esto en mente es posible pensar en un dispositivo de entrada único, que permita al usuario comunicarse libremente con la computadora sin necesidad de ningún otro dispositivo físico. Es decir, al conocer la computadora los movimientos del usuario, ésta puede asumir la interacción con un determinado dispositivo sin que este exista en realidad.

Contenido

Referencias

Introducción

Los dispositivos de entrada para las computadoras, son aquellos que les permiten adquirir información de su entorno. El proceso de captura de movimiento es utilizado por muchos de estos dispositivos para permitir la comunicación del humano con la computadora. Esta comunicación se logra registrando acciones específicas del cuerpo como son: la opresión de los dedos sobre un teclado o el desplazamiento de un apuntador (mouse) sobre un plano. Una vez que la computadora tiene acceso a esta información entonces puede interpretarla de acuerdo a la actividad que esté realizando el usuario. Bajo este esquema puede observarse que las interfaces entre el humano y la computadora están limitadas en cierta medida por la capacidad de captura de información del dispositivo de entrada [1].

En la actualidad la mayoría de las interfaces con las computadoras no requieren otros dispositivos aparte de los ya mencionados (*mouse* y teclado), o bien si se requiere una mayor interactividad, entonces el problema suele resolverse de formas ingeniosas usando estos mismos dispositivos junto con interfaces de comportamiento variable [1]. Existe, sin embargo, un acelerado avance tecnológico para el hardware gráfico de las computadoras. Lo cual nos indica que en un futuro cualquier computadora de escritorio tendrá la capacidad de representación en tiempo real de ambientes virtuales. De hecho muchas de las tecnologías para Internet nos previenen sobre este cambio:VRML, Java 3D, QTVR entre otros, sobre todo los sistemas de entretenimiento ya están haciendo uso extensivo de esta tecnología.

El principal objetivo de utilizar realidad virtual es el proveer al usuario de un ambiente con características que se apegan al mundo real. De esta manera las interfaces pueden ser similares a las herramientas y utensilios que usamos cotidianamente, por lo que su uso resulta más intuitivo. Para entender mejor esta propuesta debemos recordar que nuestra actividad mental está profundamente ligada al concepto de espacio, obtenido a lo largo de nuestra vida ("experiencia espacial"). Por lo que si el humano tiene oportunidad de interactuar en un ambiente tridimensional entonces podrá aprovechar de mejor manera sus conocimientos y habilidades previas [2]. Aún falta mucho por investigar en cuanto a la mejor manera de interactuar con este tipo de ambientes. Por el momento solo se han adaptado las interfaces existentes para permitirnos algunas actividades como es el navegar o examinar. Sin embargo con estos recursos no es posible explotar todas las ventajas que la representación de ambientes virtuales puede ofrecer [3].

El sistema de captura de movimiento que hemos desarrollado funciona como un dispositivo de entrada genérico, que permite registrar expresiones corporales para que después éstas puedan ser interpretadas en el contexto de una interfaz. Nuestro sistema básicamente es un exoesqueleto[1] que permite acoplar una serie de sensores flexibles a las articulaciones del cuerpo. Con él es posible medir el ángulo en cada

[1] Semejante a una armadura, en nuestro caso se trata de un traje de material flexible que se ajusta al cuerpo.

articulación para después reconstruir los movimientos y expresiones a través de cinemática directa.

En el primer capítulo de este trabajo se describe al lector en que consiste la captura de movimiento, los elementos que componen a los sistemas de captura, su caracterización y clasificación. Se mencionan también en forma breve el estado actual de los sistemas mas utilizados, principales características, ventajas y desventajas.

El segundo capitulo se enfoca al desarrollo del sistema en cuestión, en este punto se retoma mucho del trabajo que se ha hecho en el diseño de otros sistemas y que en el primer capitulo se describieron en forma breve. Se propone una arquitectura y un esquema de trabajo que permita salvar las desventajas de los actuales sistemas de captura. Se definen los elementos electrónicos y las herramientas de programación a utilizar y se comenta el trabajo de construcción del sistema. El capitulo finaliza con un resumen de las principales características observadas en el sistema durante su desarrollo y en pruebas preliminares.

En el tercer capítulo se describen los resultados obtenidos al utilizar el sistema, desde la prueba de los sensores, la reconstrucción y visualización del movimiento capturado, hasta la aplicación del sistema como un dispositivo de entrada.

En el cuarto y último capítulo se mencionan las mejoras al sistema y el trabajo a futuro que se pretende realizar. Se hacen algunos comentarios y conclusiones finales acerca del proyecto.

Capítulo 1

Antecedentes

La captura de movimiento consiste en el registro de la posición y orientación de objetos a través del tiempo. Este proceso puede ser aplicado para registrar los movimientos de una persona. Para este caso en particular la información capturada puede ser tan general como la posición del cuerpo en el espacio, o bien tan compleja como las gesticulaciones faciales o las deformaciones de la masa muscular [4].

1.1. Componentes de un sistema de captura

Un sistema de captura de movimiento para el cuerpo humano se compone de los siguientes elementos [5], algunos de ellos pueden omitirse de acuerdo a la tecnología utilizada (ver figura 1.1):

- Sensores y/o marcas[2] o fuentes[3], con sus respectivas interfaces electrónicas (todos ellos sobre el cuerpo del sujeto).
- Fuentes o marcas y/o sensores, con sus respectivas interfaces electrónicas (todos ellos externos).
- Interface electrónica para la computadora.
- Computadora.
- Modelo computacional y datos.

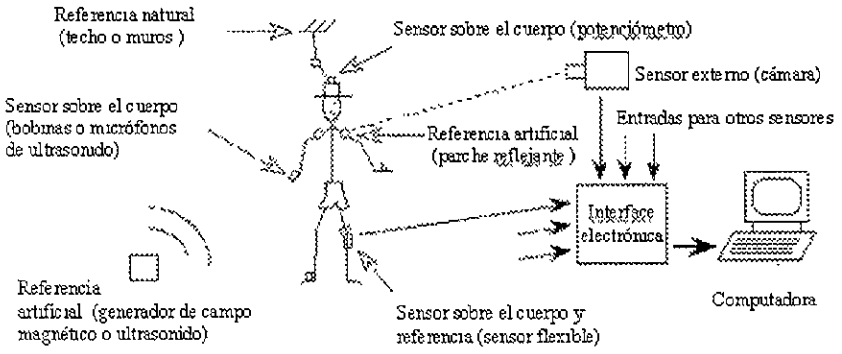

Figura 1.1. Ejemplos de componentes de los sistemas de captura de movimiento para el cuerpo humano [5].

[2] Su función es reflejar la energía emitida por una fuente. Para el caso de sistemas ópticos estas pueden ser de material reflejante u otro cuyo color contraste con su entorno.

[3] Una fuente es un emisor de luz, ultrasonido, campo magnético o cualquier otra manifestación de energía cuya magnitud o comportamiento pueda ser afectado por el movimiento y de esta forma sea posible medir su variación

El movimiento de un objeto siempre es relativo a un punto de referencia. Para medir dicho movimiento se requiere un sensor acoplado al objeto que pueda detectar la distancia, orientación o posición de una fuente fija al punto de referencia. Una configuración equivalente resulta al acoplar la fuente al objeto y el sensor al punto de referencia.

El cuerpo humano consiste en una serie de partes móviles, de tal forma que si tomamos a cada una de ellas como referencia para otras entonces podremos acoplar sensores y fuentes a estas para registrar su movimiento relativo. De forma similar podemos disponer los sensores o las fuentes en un punto de referencia fuera del cuerpo, es decir, nuestro punto de referencia puede ser externo y común para todas las partes móviles del cuerpo.

Las fuentes pueden ser naturales: el campo gravitacional o magnético de la tierra; o artificiales: campo magnético producido por bobinas, luz emitida por un LED, frecuencia de audio proveniente de bocinas, etc. Como podemos observar existen solo 3 configuraciones posibles para la disposición de los sensores y las fuentes en un sistema de captura de movimiento para el cuerpo humano:

- El sensor y la fuente dispuestos sobre el cuerpo.
- El sensor acoplado al cuerpo y la fuente fija a un punto de referencia externo.
- La fuente acoplada al cuerpo y el sensor fijo a un punto de referencia externo.

Con base en este esquema, es posible clasificar a los sistemas de captura de una forma sencilla, que considera a todos los sistemas existentes.

1.2. Clasificación y caracterización de los sistemas de captura de movimiento

Existen varias clasificaciones para los sistemas de captura, algunas de ellas se basan en el principio de funcionamiento, por lo que los sistemas pueden agruparse en: mecánicos, ópticos, magnéticos, etc. [6]. Estas clasificaciones pueden resultar demasiado extensas y fácilmente puede caerse en ambiguedades. Axel Mulder [5] propone una de las clasificaciones, que a nuestro criterio es una de las mas completas. Esta se basa en la ubicación de los sensores, las fuentes y las marcas, con base en estas referencias los sistemas pueden ser clasificados en 3 grandes grupos:

- **Medición interna (*inside-in*)** : el sensor y la fuente están sobre el cuerpo, por lo general estos sensores tienen pequeños factores de deformación[4] por lo que se recomiendan para sensar partes pequeñas del cuerpo, sin embargo, bajo ciertas configuraciones también es posible capturar el movimiento de partes de mayor tamaño. Estos sistemas no tienen limitantes de espacio de trabajo ya que no

[4] Relacion entre el cambio de su forma y el cambio de alguna de sus propiedades físicas, por ejemplo la resistividad.

proveen información sobre su posición relativa al medio ambiente. Su principal desventaja es que obstruyen en mayor o menor medida el libre movimiento del cuerpo. Un ejemplo de estos sistemas son los guantes con sensores piezo-resistivos y los exoesqueletos.

- **Medición del interior al exterior (*inside-out*):** los sensores están fijos sobre las partes del cuerpo y existe una fuente exterior, por ejemplo una bobina moviéndose en un campo magnético generado externamente. Estos sistemas proveen información sobre su posición relativa al medio ambiente. El espacio de trabajo y precisión están limitados por las características de la fuente y los sensores. Por lo regular son utilizados para capturar el movimiento de partes grandes de cuerpo humano. Como ejemplos de estos se pueden mencionar los sistemas magnéticos y los acústicos

- **Medición del Exterior al interior (*outside-in*):** los sensores se encuentran en el exterior detectando fuentes sobre el cuerpo, en este caso las fuentes se denominan como marcas, que pueden ser pasivas (reflectores) o activas (emisores). Estos sistemas son sensibles a la oclusión[5] por lo que generalmente no se ocupan con partes pequeñas del cuerpo ya que esto limita su espacio de trabajo, que de por si se considera reducido. Estos sistemas son los que menos obstruyen el movimiento del cuerpo, pero también los que mayores recursos consumen. Dentro de esta clasificación están los sistemas ópticos.

Como complemento a la clasificación arriba descrita Mulder también propone una serie de parámetros con los cuales pueden ser caracterizados los sistemas de captura. Dichos parámetros nos permiten identificar a cada sistema, sus ventajas y desventajas sobre otros.

Especificaciones descriptivas

- Medio entre el sensor y la fuente (acústico, electromagnético, óptico, mecánico).
- Localización del sensor y la fuente (inside-in, inside-out, outside-in).
- Tipo de fuente (artificial o natural).
- Parte del cuerpo a que se aplica (dedos, manos, brazos, hombros, ojos, cabeza, torso, piernas, pies, etc).

Medidas estáticas

- Rango de resolución espacial.
- Precisión.
- Linealidad, histéresis, calibración.

[5] Esto sucede cuando las marcas quedan ocultas a los sensores, por lo regular por la interferencia de alguna parte del cuerpo

Medidas dinámicas

- Precisión dinámica (si el sistema es lineal este valor es igual a la precisión
- estática).
- Ancho de banda, rango de frecuencias o resolución temporal.
- Latencia, retraso, defasamiento, tiempo de actualización o exactitud temporal.

Medidas de precisión

- Repetibilidad, estabilidad, durabilidad.
- Inmunidad al ruido, interferencia electromagnética (radiofrecuencias, 60Hz, materiales ferromagnéticos, oclusion), capacidad de filtrado.

En la siguiente tabla se muestran las características típicas de los sistemas de captura.

	Medición interna (Inside-in)	**Medición del interior al exterior (Inside-out)**	**Medición del exterior al interior (Outside-in)**
Resolución espacial	~ 0.5 - 1 grados	~ 0.005 - 8 mm; ~0.025 - 0.1 grados	~ 0.0015 - 0.2 % del área de captura.
Precisión espacial	<= 5 grados	~ 0.8 - 5 mm; ~ 0.1 a 3 grados	~ 0.004 - 0.5 % del área de captura.
Ancho de banda	<= 80 Hz	<= 150 Hz	<= 125 Hz
Latencia	~ 1 ms	~ 1 - 40 ms	~ 1 ms
Precisión	Media o alta	Alta	~ 0.0055 - 0.02 % del área de captura.
Marco de referencia	Objeto en movimiento y referencia.	Área de captura.	Área de captura.
Dimensiones	1 cc (sensor y fuente)	1-10 cc (sensor); 10 cc (fuente)	< 1 cc (sensor); 10 cc (fuente)
Area efectiva de trabajo	Sin límite	Fuente natural: ilimitada. Fuente artificial: 1-2 m de radio.	1 a 4 m de radio
Precio (USD)	Guante: $1000- $10,000. Traje: $30,000.	$1000 a $10,000	$20, 000 a $150,000

1.3. Estado actual de los sistemas de captura mas utilizados

El mercado de los sistemas de captura ha crecido notablemente ante al abaratamiento del equipo de cómputo de alto desempeño, ya que con ello es posible costear aplicaciones de animación, entretenimiento, etc. Esto ha dado lugar a que diversas compañías desarrollen y comercialicen una gran variedad de productos, uno para cada necesidad y presupuesto. Los 3 sistemas que a continuación se mencionan representan los más utilizados para la mayoría de las aplicaciones. Cada uno de ellos a su vez sirve como ejemplo de cada uno de los grupos de la clasificación dada por Mulder [5].

1.3.1. Sistemas ópticos para captura de movimiento (outside-in)

El funcionamiento de estos sistemas se basa en el análisis de imágenes de video de alto contraste de marcas reflectoras acopladas al objeto cuyo movimiento se desea capturar. Las marcas son pequeñas esferas cubiertas de material reflectivo similar al de los señalamientos de tránsito. La imagen de las esferas es captada por cámaras de alta velocidad, el número estas últimas depende del tipo de captura de movimiento. La captura de movimientos faciales utiliza por lo regular una cámara. La captura de movimiento del cuerpo completo puede requerir de cuatro o más cámaras, tantas como sean necesarias para cubrir el área donde se realizan los movimientos. Para incrementar el contraste, cada cámara está equipada con emisores (LEDs) infrarojos (IR) y filtros de paso IR en las lentes de éstas. Las cámaras están asociadas a tarjetas controladoras, que por lo regular están instaladas en un equipo PC.

Dependiendo del sistema, durante el proceso de captura la computadora graba el video de alto contraste (1 bit) o bien imágenes mostrando sólo centroide de cada marca. Antes de iniciar esta tarea debe calibrarse el sistema, esto se logra grabando un marco de calibración, el cual está compuesto por un conjunto de marcas distribuidas en un espacio 3D acotado y cuyas medidas son cuidadosamente registradas. Este marco servirá como referencia para la sesión de captura.

Después de la sesión de captura, las imágenes deberán ser procesadas. Si se cuenta con el video de alto contraste, entonces se deberán encontrar los centroides de cada marca, pero si ya se cuenta con las imágenes que los muestran entonces se procede a correlacionar su posición con la de imágenes obtenidas por otras cámaras. Tomando las imágenes de un par de cámaras es posible triangular la posición de cada marca en el espacio. La trayectoria de cada marca es identificada al procesar el conjunto de cuadros del video. Durante este proceso algunos problemas pueden ocurrir, como es la imposibilidad de diferenciar una marca de otra cuando sus imágenes se traslapan, oclusión de una marca con alguna parte del cuerpo, ruido y reflejos falsos en la imagen. Algunos de estos problemas pueden salvarse si nos auxiliamos de las imágenes provenientes de las otras cámaras para inferir la verdadera posición de las marcas. Puede observarse que este proceso puede requerir de tiempo y en ocasiones de intervención humana, todo depende de la calidad de la información capturada y de la

fidelidad requerida. Si los datos están limpios el proceso puede llevar de uno a dos minutos por cada segundo de captura (realizado con un muestreo de 120 imágenes por segundo). Para datos con ruido o que describen trayectorias complicadas el tiempo requerido por segundo de captura, puede dispararse hasta 15 o 30 minutos. Para aplicaciones que requieren menos de 1 minuto de captura de movimiento, puede utilizarse los sistemas ópticos, pues serán prácticos aún en los peores casos. Para aplicaciones que requieran mayor tiempo de captura, el problema de la variabilidad del tiempo requerido para procesar las imágenes puede ser riesgoso. Los proveedores de servicios de captura con esta tecnología pueden cobrar hasta $180 dólares por hora de este proceso [7].

Configuración típica

La configuración usual para capturar el movimiento de una persona utiliza de 20 a 30 marcas adheridas al cuerpo o a ropa ajustada. El tamaño de estas marcas va de 1 a 5cm de diámetro, dependiendo del hardware y del área de captura. La ubicación de las marcas sobre el cuerpo depende de la aplicación que se dará a los datos. Por lo regular solo se utiliza una marca por punto de interés, tales como: pies, rodillas, caderas, codos, etc.

La configuración mas sencilla debe incluir: tres marcas para la cabeza del sujeto, una marca en la base del cuello y otra en la base de la espina dorsal, una más en cada hombro, codo, muñeca, mano, rodillas, tobillos y pies, haciendo un total de 21 marcas. En caso de que se requiera información más detallada como puede ser la rotación de la muñeca, entonces se requerirán marcas adicionales. En el caso de la rotación cubital se necesita sustituir la marca de la muñeca por un par en ambos extremos de ella de forma que pueda apreciarse la rotación relativa entre el brazo y el codo.

Ventajas

- Dependiendo del sistema de captura y de la precisión requerida, la amplitud del área de captura puede ser arbitrariamente grande.

- El sujeto no porta al sistema de captura, por lo que tiene mayor libertad de movimiento. Esto le permite correr, saltar y hace otras actividades que no pueden hacerse si se portan cables y equipo electrónico.

- Las marcas son elementos pasivos dentro del sistema de captura por lo que su costo es mínimo, incrementar número resulta extremadamente barato. En teoría pueden ser incluidos varios cientos de marcas en una misma área de captura, sin embargo el límite puede ser establecido por el software de procesamiento o la capacidad de cómputo del hardware utilizado.

- La frecuencia de muestreo de estos sistemas es de 120 a 250 Hz, la cual es suficiente para registrar la mayoría de los movimientos del cuerpo humano, incluso movimientos rápidos como puede ser el lanzamiento de un objeto, una patada o un golpe. Debemos recordar que para la mayoría de las aplicaciones, como son cine y video, se utilizan de 20 a 30 cuadros por segundo; 120 muestras por segundo ofrecen una mayor calidad de captura pero pueden resultar imperceptibles a la vista.

Desventajas

- Los sistemas de captura de movimiento ópticos son los más costosos, desde a $150,000 hasta $250,000 dólares; inclusive el costo de operación es elevado.

- Dado que los sistemas de captura ópticos se basan en el análisis de imágenes de alto contraste, tanto la iluminación ambiental, reflejos, colores en la ropa y el fondo de la escena pueden afectar el resultado final.

- Los reflejos producidos por joyería, pisos y en general por objetos brillantes pueden inducir errores en el sistema al generar lecturas de marcas donde estas en realidad no existen.

- Dado que las marcas deben ser visibles por al menos 2 cámaras (para poder reconstruir a través de triangulación la posición espacial de una marca), existe el riesgo de que una de ellas quede oculta por otro objeto (como puede ser una parte del cuerpo). Esto puede resultar en perdida de información, ruido, desplazamientos o confusión entre marcas.

- Como se menciono anteriormente, el tiempo de procesamiento de las imágenes registradas en una sesión de captura pueden variar, ello depende de los requerimientos de precisión, complejidad de los movimientos y en general de la calidad de la información capturada.

- Este tipo de sistemas no puede ofrecer resultados en tiempo real, por lo que si la calidad de la información capturada es adecuada o no podrá saberse hasta el momento de su procesamiento. Es común que se deba repetir el proceso de captura 2 o 3 veces para obtener resultados aceptables.

- A pesar de que la adición de un mayor numero de marcas es relativamente barato, debe considerarse que con ello se incrementa también el tiempo de procesamiento, ya que debe darse seguimiento a la trayectoria de cada una de ellas a través del software. Por otro lado debe considerarse que el procesamiento de las imágenes solo nos dará la posición en el espacio de cada marca, y que para reflejar dicho movimiento en una entidad dentro de la computadora como puede ser un personaje virtual debe utilizarse un proceso de cinemática inversa.

- El sistema es muy sensible a la calibración inicial, por lo que cualquier desplazamiento de las cámaras puede producir inconsistencias como duplicidad de puntos o perdida de la trayectoria.

1.3.2. Sistemas magnéticos para captura de movimiento (inside-out)

Los sistemas de captura magnéticos utilizan sensores que miden con precisión el campo magnético producido por una fuente, como puede ser una bobina o un conjunto de ellas. Como ejemplos de estos sistemas están "Bird" y "Flock of birds" de Ascension [8], y "Fastrak" y "Ultratrak" de Polhemus [9]. Este tipo de sistemas pueden operar en tiempo real y proveer de 15 a 120 muestras por segundo (dependiendo del número de sensores). Los datos adquiridos proveen información sobre posición y orientación (6 grados de libertad) con un mínimo de retardo entre el muestreo y la transmisión de datos.

Los sistemas de captura magnéticos tienen una o más unidades de control, a ellas se conectan las fuentes de campo magnético y los sensores. A su vez las unidades de control están conectadas a una computadora, en la mayoría de los casos a través de una red o por medio de un puerto serie. El software de captura puede comunicarse a las unidades de control a través de un manejador (*driver*). Antes de iniciar la sesión de captura los sensores deben fijarse a los objetos cuyo movimiento será rastreado. La fuente puede estar dentro del área de captura o fuera de ella, la restricción mas fuerte de estos sistemas es que no deben existir objetos metálicos que interfieran con el campo magnético, ya que esto alteraría las lecturas de posición y orientación de los sensores.

Configuración típica

Dado que cada sensor provee información acerca de su posición y orientación, es posible utilizar un número pequeño de estos (10 a 20 sensores) para reconstruir el movimiento del cuerpo humano usando cinemática inversa. La restricción de este método es que debe conocerse la longitud de los miembros del cuerpo.

El software "MotionSampler" de AliasIWavefront [10] utiliza un proceso de filtrado denominado "BIPED" para determinar el ángulo de rotación de las coyunturas y la posición en el espacio del cuerpo humano. Este método solo utiliza 11 sensores estratégicamente distribuidos y proporciona una traslación y alrededor de 16 rotaciones. Por otro lado el software "Kinemation" también de AliasIWavefront ofrece 2 métodos para reconstruir el movimiento del cuerpo:

1. Utilizar cinemática inversa para obtener los ángulos de las coyunturas.
2. Obtener estos ángulos directamente utilizando 2 sensores, uno a cada lado de las coyunturas.

El segundo método ofrece mejores resultados, sin embargo requiere un mayor cuidad al calibrar el sistema.

El proceso de captura utilizando un sistema magnético es muy similar al proceso de filmar una escena. Para asegurar un mejor resultado es necesario que los usuarios ensayen y se familiaricen con los sensores, el cableado y el área de captura donde está activo el campo magnético. Dado que estos sistemas pueden trabajar en tiempo real, los pueden apreciarse inmediatamente y dado el caso corregir los movimientos, haciendo el proceso de captura más interactivo.

Ventajas

- Los sensores proveen información acerca de su posición y orientación, por lo que es posible inferir algunos movimientos del cuerpo utilizando un número reducido de estos sensores.

- Estos sistemas miden la distancia y rotación relativa entre el objeto que es fuente del campo magnético y cada uno de los sensores, por lo que para calibrar el sistema solo es necesario conocer la posición de la fuente.

- Estos sistemas son utilizados para una gran variedad de aplicaciones, dada la robustez que presentan pueden usarse incluso para dotar de movimiento a personajes virtuales en programas de televisión en vivo. Por su razonable confiabilidad, fácil mantenimiento y sencillez de transporte e instalación, son los sistemas más utilizados y existe una mayor experiencia entorno a ellos.

- Dado que estos sistemas pueden trabajar en tiempo real el proceso de captura interactivo puede hacerse en forma interactiva.

- El costo del sistema está por debajo de los $40,000 dólares.

Desventajas

- El campo magnético utilizado por estos sistemas se ve afectado por objetos metálicos. Algunos sistemas son menos sensibles que otros a estas interferencias, pero aún así deben evitarse muros, pisos, bases y muebles metálicos cerca del área de captura

- El área de captura está limitada por el área donde sea efectivo el campo magnético producido por la fuente. Esta área suele ser mucho menor que la de los sistema ópticos.

- El número de sensores que pueden acoplarse al cuerpo humano puede estar limitado por la carga de grueso cableado que esto implica.

- La velocidad de muestreo (30 a 60 HZ) puede resultar insuficiente para algunas actividades como son las deportivas, donde las rápidos movimientos de los miembros no pueden ser registrados por el sistema. Si a esto añadimos el proceso de filtrado que comúnmente se aplica en los sistemas magnéticos para reducir el "temblor" producido por pequeñas variaciones en las mediciones, entonces la velocidad de muestreo puede reducirse a 15 HZ.

1.3.3. Sistemas mecánicos para captura de movimiento (inside-in)

El principio de funcionamiento de estos sistemas es uno de los más simples. Se basa en detectar los movimientos de una estructura articulada a través de sensores dispuestos en cada parte móvil de ésta. Los sensores pueden ser potenciómetros, codificadores o cualquier otro con el cual se pueda determinar la posición relativa entre una parte móvil y otra. La información proveniente de los sensores es enviada a la computadora y en ella puede reconstruirse el movimiento de la estructura a través de cinemática directa. Para capturar el movimiento del cuerpo humano la estructura se sobrepone al cuerpo, a manera de un exoesqueleto; sus articulaciones deben concordar con las coyunturas del sujeto de tal forma que cada movimiento de este sea transmitido a la estructura. Un ejemplo de estos sistemas es "Gypsy" de MetaMotion [11] ,el cual provee información de hasta 17 coyunturas del cuerpo utilizando 43 sensores.

Ventajas

- No tiene los problemas de distorsión inherentes en los sistemas que requieren fuentes o sensores externos. La distorsión del campo magnético por objetos metálicos y la oclusión de las marcas en los sistemas ópticos son ejemplos de estos problemas. No requiere de ambientes controlados.

- La inmunidad a interferencias externas permite que estos sistemas puedan fácilmente extenderse para capturar el movimiento de varios sujetos dentro de una misma área.

- El área de captura prácticamente es ilimitado ya que esta se basa únicamente en la posición relativa entre las partes móviles del cuerpo.

- Sencillo de utilizar y calibrar, el entrenamiento del sujeto es mínimo.

- El hardware es extremadamente sencillo, lo cual facilita su fabricación y mantenimiento

- El costo de estos sistemas es el más bajo, alrededor de $20,000 dólares.

Desventajas

- El exo-esqueleto puede obstruir el libre movimiento del sujeto si su estructura es demasiado rígida.

- La anterior restricción ha impedido el uso de estos sistemas para muchas aplicaciones por lo que el mercado y la experiencia de los usuarios se ha enfocado a sistemas como los magnéticos y ópticos a pesar del alto costo que ello implica.

1.4. Conclusiones

En el capítulo que aquí concluye se describió la tecnología para la captura de movimiento del cuerpo humano, los elementos que componen a los sistemas de captura, su clasificación y caracterización. Se mencionó también el estado actual de los sistemas más utilizados en la actualidad, haciendo énfasis en su operación, ventajas y desventajas a fin de retomar todos estos al momento de diseñar nuestro sistema de captura. En el siguiente capítulo retomaremos el esquema de operación de los sistemas de medición interna, cuyas ventajas en precio, sencillez de operación y construcción lo hacen un modelo ideal a seguir para nuestro desarrollo.

Capítulo 2

Diseño y desarrollo del sistema de captura

En el anterior capítulo se describió en forma breve la tecnología de captura de movimiento y se mencionaron las características de los sistemas más utilizados en la actualidad. En el presente capítulo se describe el diseño y desarrollo del sistema de propuesto como proyecto de tesis y cuyo funcionamiento se basa en los sistemas de captura mecánicos (medición interna).

2.1. Arquitectura del sistema

De acuerdo a la descripción de los sistemas de captura mecánicos expuesta en el capitulo anterior, puede observarse que el principio de funcionamiento de estos es uno de los más sencillos. Los sistemas mecánicos se componen básicamente de una serie de sensores de posición, una interfaz electrónica para acondicionar y digitalizar las señales de los sensores, y de un manejador residente en la computadora donde se hará el procesamiento de la información (ver figura 2.1).

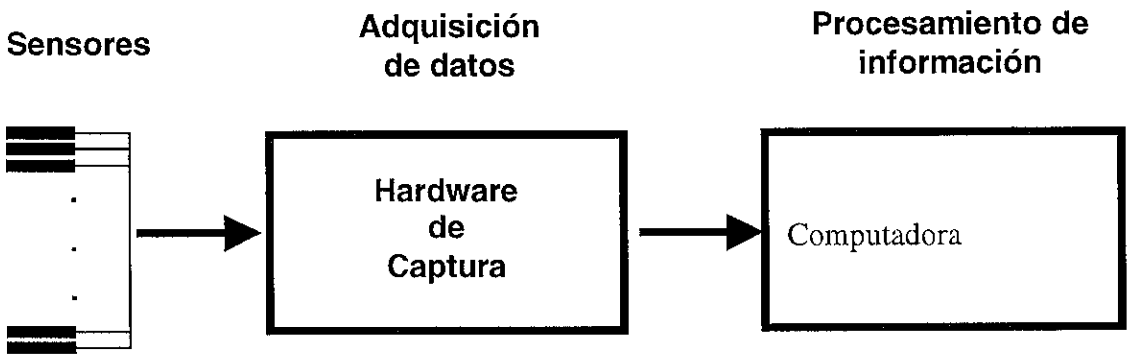

Figura 2.1. Esquema general de un sistema de captura mecánico.

La principal desventaja de los sistemas mecánicos es la obstrucción que representa la estructura que permite mantener a los sensores sobre las articulaciones. La propuesta de solución para este problema es utilizar estructuras flexibles que presenten una resistencia mínima al libre movimiento del cuerpo. En cuanto a la adquisición de datos se propone el uso de un microcontrolador que cuente con puertos de conversión A/D y con algún medio de comunicación estándard (USB, RS232, paralelo) de tal forma que la información pueda ser enviada a una computadora para su procesamiento.

Es fácil ver que el diseño de este sistema puede dividirse en diferentes módulos, lo cual resulta conveniente si pensamos en facilitar las pruebas del prototipo y posteriores modificaciones y mejoras. Una arquitectura modular como la que se muestra en la figura 2.2 permite sustituir cada elemento sin necesidad de modificar a otros. Esta arquitectura no es mas que una extensión del esquema general de la figura 2.1. El hardware abarca los sensores y el circuito de adquisición de datos y el software esta formado por un conjunto de procesos residentes una o más computadoras. Los elementos de la capa superior permiten la programación de nuevas aplicaciones y/o la comunicación con el software de terceros.

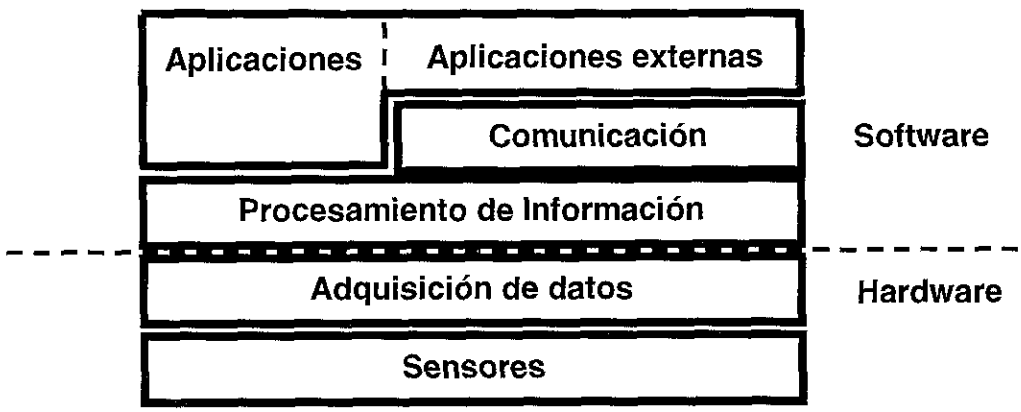

Figura 2.2. Arquitectura propuesta para el sistema de captura de movimiento.

2.2. Descripción general del funcionamiento del sistema

Una vez propuesta la arquitectura del sistema que se pretende desarrollar, procederemos a describir con mayor detalle cada uno de sus componentes. Comenzaremos con la descripción de los elementos del hardware, continuaremos con el software y finalmente se mostrará como utilizar el sistema en el desarrollo de nuevas aplicaciones.

2.2.1. Sensores

La mayoría de las estructuras y sensores utilizados en los exoesqueletos son de materiales rígidos que dificultan los movimientos, y que en algunos casos pueden dañarse por flexiones bruscas. Los sensores de nuestro sistema son de material flexible cuyas las propiedades físicas no afectan las mediciones, por lo que el deterioro por el uso afectan en menor medida sus funciones. Estos sensores tienen la característica de cambiar sus propiedades eléctricas en función de la torsión a que son sometidos. En la figura 2.3 se muestra en forma gráfica el cambio de resistencia en las terminales del sensor de acuerdo al ángulo en que es flexionado. A medida que se deforma el sensor su resistencia eléctrica se incrementa en forma lineal, este comportamiento puede

observarse desde 0 ° con un determinado valor de resistencia, hasta los 90° donde la resistencia toma un valor estacionario.

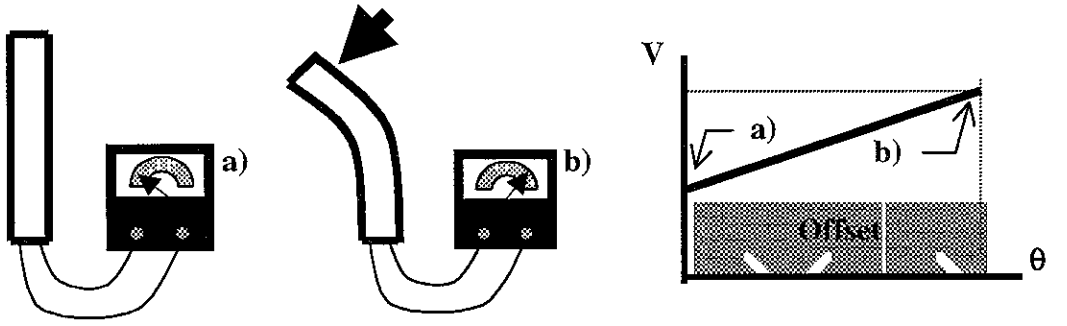

Figura 2.3. Representación del cambio de las características eléctricas del sensor en función de la torsión a que es sometido.

Es fácil ver que si acoplamos estos sensores a las articulaciones, entonces se podrá determinar el ángulo de flexión de las mismas. En la figura 2.4 se muestra la representación de una mano y uno de los sensores acoplado a la articulación del dedo índice.

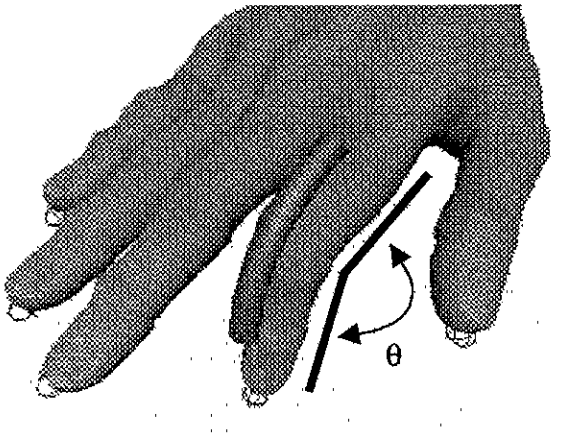

Figura 2.4. Acoplamiento del sensor a una articulación y la correspondencia entre la deformación del primero con el ángulo de flexión de la última.

El ángulo de flexión de las articulaciones puede obtenerse a partir de las lecturas del sensor (resistividad) mediante la siguiente función lineal :

```
AnguloFlexión = LecturaSensor * Factor + Offset
```

El factor (pendiente) y el offset que se indica en la función pueden depender de las características de la anatomía de cada usuario y/o de los elementos del sensor, sin embargo estas variaciones pueden corregirse al calibrar el equipo.

2.2.2. Adquisición de datos

El módulo de adquisición de datos tiene la función de tomar las señales provenientes de los sensores, digitalizarlas y enviarlas a una computadora para su procesamiento (ver figura 2.5). Este módulo se compone básicamente de un microcontrolador HC11 modelo F1 [12] en configuración Single-Chip-Mode [12] y una serie de multiplexores analógicos. Estos últimos son necesarios para extender el número de canales de conversión, ya que el microcontrolador solo cuenta con 8 puertos de conversión A/D. A cada puerto de conversión del F1 se asocia un multiplexor analógico 8X1, y con esto es posible atender hasta 64 canales (sensores S1 a S64 de la figura).

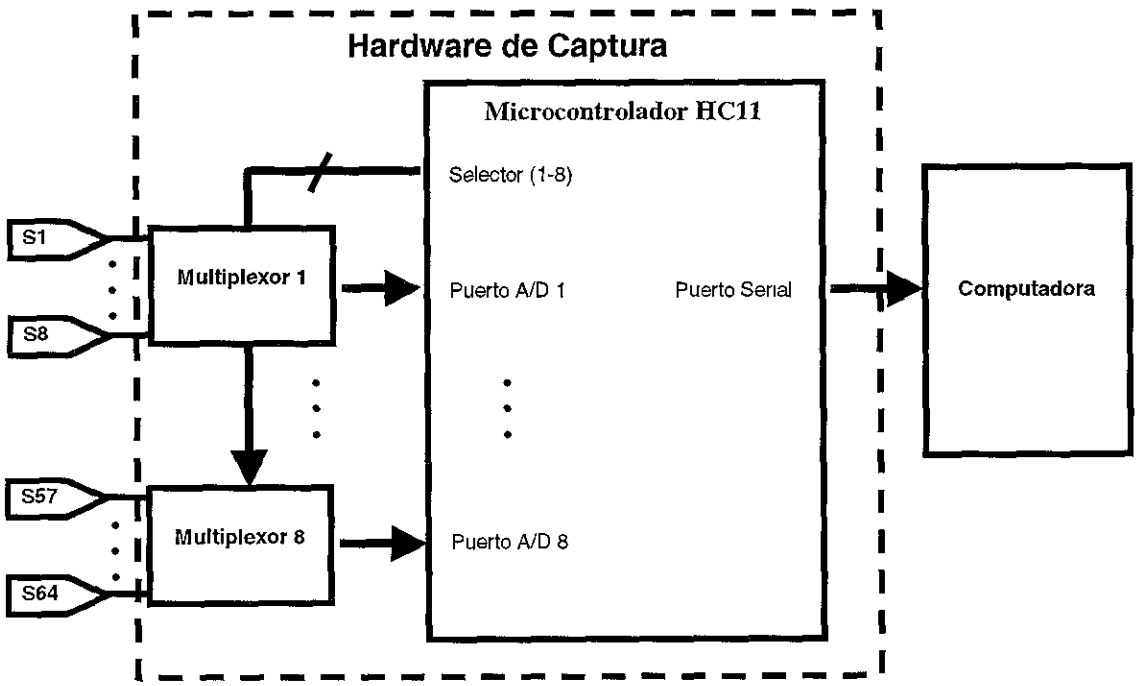

Figura 2.5. Módulo de adquisición de datos.

La información capturada por el microcontrolador es empaquetada y enviada por un puerto serial estándar RS232 a una computadora. Cada paquete está formado por 64 enteros cortos (8 bits) que representan la información registrada, y por un par de caracteres especiales que representan el inicio y final del paquete:

$$\#D_1 D_2 D_3 \ldots D_{61} D_{62} D_{63} \#$$

El esquema de trabajo del módulo de adquisición de datos es el siguiente:

1. Inicializar puertos de conversion A/D.
2. Inicializar puerto serial.

3. Si se inicia un nuevo paquete de información:
   Enviar cabecera del paquete por el puerto serial.

4. Seleccionar el canal correspondiente de cada multiplexor.

5. Realizar conversión A/D.

6. Enviar información por el puerto serial.

7. Si se han barrido todos los canales:
   Enviar carácter de fin del paquete por el puerto serial.

8. Regresar al paso 3.

El programa que provee esta funcionalidad se aloja en la memoria del HC11, el código fuente puede consultarse en el apéndice correspondiente. En la figura 2.6 se muestran los principales componentes del hardware de captura.

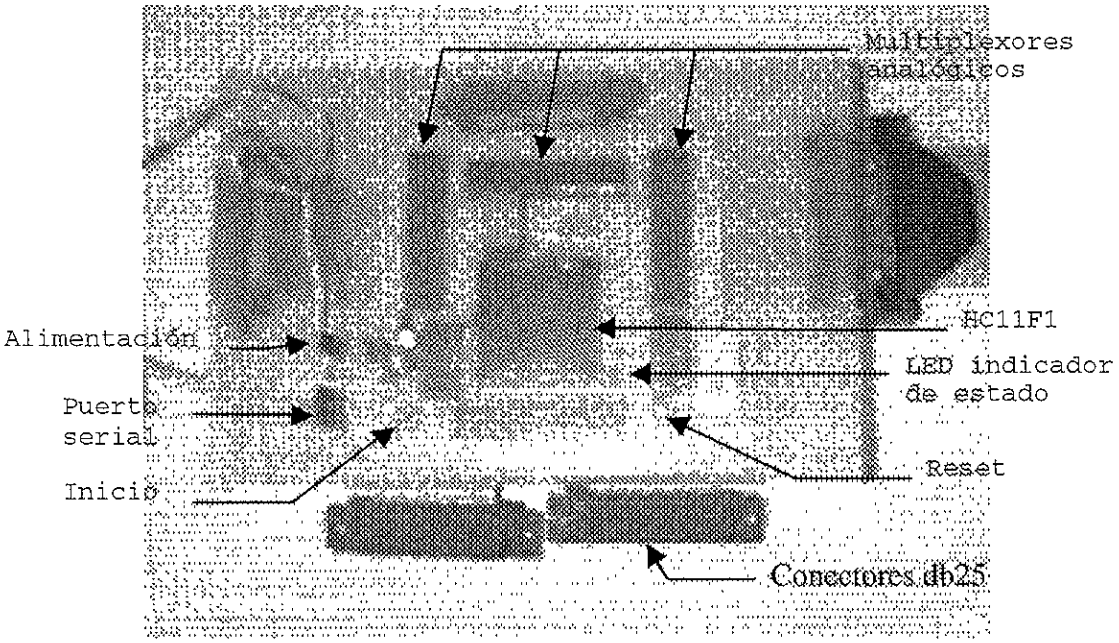

Figura 2.6. Vista del prototipo del hardware de captura.

Para operar este módulo el usuario debe seguir los siguientes pasos:

1. Conectar el sistema a una fuente de 5 Volts.

2. Conectar el cable de puerto serie a una computadora.

3. Conectar los sensores a través de los conectores db25.

4. Oprimir el botón de reset.

5. Oprimir el botón de inicio.

Una vez que se ha pulsado el botón de inicio el LED indicador comenzará a encenderse y apagarse a razón de 2 veces por segundo, y la información empezará a enviarse a través del puerto serie. En la figura 2.7 se muestra el hardware de captura y un guante al cual se le han acoplado una serie de sensores, en este punto el prototipo permite registrar los movimientos del brazo derecho de una persona.

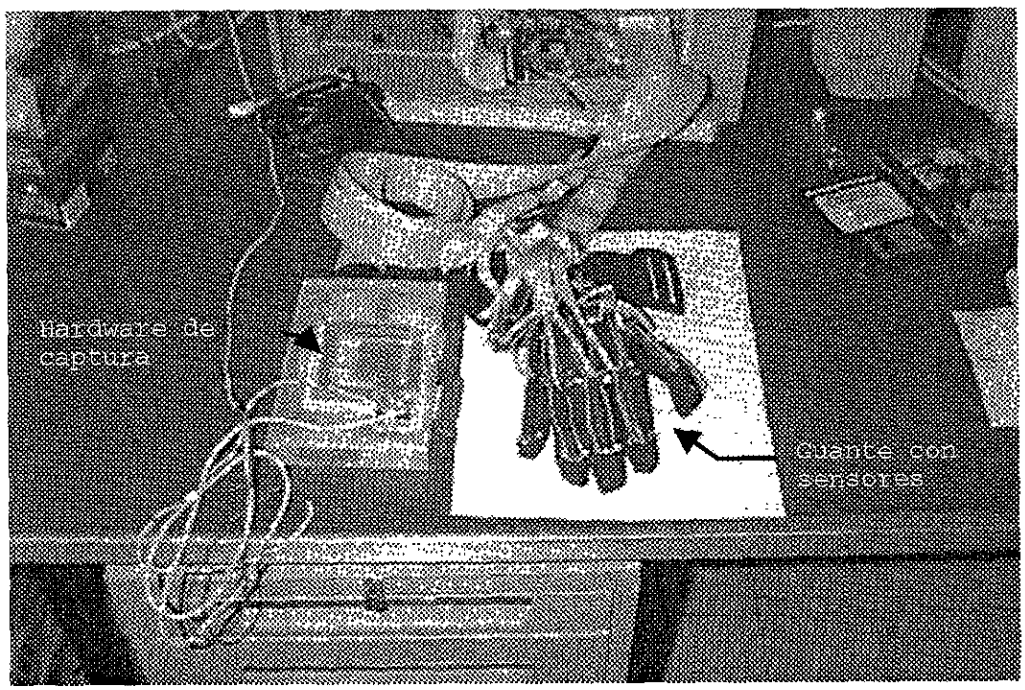

Figura 2.7. Elementos del prototipo del sistema de captura.

2.2.3. Procesamiento de la información

Este módulo es un proceso residente en la computadora que se conecta al módulo de adquisición de datos y tiene la tarea de recibir la información a través del puerto serial y ponerla a disposición de otras aplicaciones. La programación de éste módulo puede realizarse en cualquier lenguaje que proporcione soporte para la comunicación a través de puertos seriales. Independientemente del lenguaje utilizado el esquema de trabajo del módulo de procesamiento de información es el siguiente:

1. Inicializar el puerto serial.

2. Leer información del puerto serial.

3. Desempaquetar los datos.

4. Mapear los datos a grados.

5. Filtrar información (opcional).

6. Poner a disposición de otras aplicaciones la información capturada a través de objetos.

7. Regresar al paso 2.

En esta tesis se desarrollaron 2 prototipos para este módulo, uno programado en lenguaje Java y el otro en C. El código fuente del módulo de procesamiento puede consultarse en el apéndice correspondientes (Servidor.java y ServidorMocap.c respectivamente). En esta sección únicamente se mostrarán fragmentos de código conforme se considere necesario.

La inicialización de los puertos es muy similar en los lenguajes Java y C. Con este proceso se indica la configuración necesaria en cuanto a velocidad, tamaño de los paquetes, corrección de errores etc.

Código en lenguaje Java:

```
serialPort = (SerialPort) portId.open("LectorMocap", 2000);
serialPort.setSerialPortParams(9600,
                SerialPort.DATABITS_8,
                SerialPort.STOPBITS_1,
                SerialPort.PARITY_NONE);
```

Código en lenguaje C:

```
fd = CapSerialOpen("COM1", 9600, 8, CAP_PARITY_NONE, 1,
                        CAP_SERIAL_BINARY, NULL);
```

En el caso de Java, las bibliotecas que proveen el soporte de comunicación con los puertos seriales son distribuidas por SUN bajo el nombre de "Java Communications" [28]. Para el lenguaje C existen muchas bibliotecas para cubrir esta necesidad, en nuestro caso se utilizaron las bibliotecas de captura de movimiento que se distribuyen con el software Maya [14].

En Java el proceso de lectura del puerto serial se lleva a cabo a través de activación de eventos, de esta manera sólo se atiende al puerto cuando es necesario. Al producirse un evento relacionado al puerto serial se activa la función `serialEvent`. En ella deben filtrarse los diferentes eventos provenientes del puerto y dado el caso efectuar la lectura correspondiente:

```
public void serialEvent(SerialPortEvent event)
{
.....
        switch(event.getEventType()) {
         case SerialPortEvent.BI: System.out.println("Break"); break;
         case SerialPortEvent.OE: System.out.println("Overrun"); break;
```

```
.....
        case SerialPortEvent.DATA_AVAILABLE:
            readBuffer = new byte[1];
            try {
                while (inputStream.available() > 0)
        ....
}
```

El proceso de lectura cambia sustancialmente en el caso de C, ya que el esquema de trabajo es utilizar un ciclo en el cual se verificará periódicamente el estado del puerto serial, y su caso se efectuar la lectura de información:

```
while(1)
        {
           n=CapSerialRead(fd, readBuffer,1);
           if(n)
               {
               ...

   }
```

Una vez que se tiene acceso a la información proveniente del módulo de captura a través del puerto serial, entonces es necesario desempaquetar los datos correspondientes a cada uno de los sensores. Este proceso consiste de los siguientes pasos:

1. Esperar cabecera del paquete (caracter especial #).

2. Una vez que se detecta la cabecera se asume que los siguientes 64 dígitos representan la información de cada uno de los canales de captura.

3. La última lectura debe corresponder a un caracter especial #, de ser así se guarda el registro de las lecturas, en caso contrario se asume que se ha perdido la sincronización con el módulo de captura, por lo que se desecha el paquete. En ambos casos se retorna al paso1.

Una vez que se ha capturado la información correspondiente a los canales de captura se procede a mapear cada uno de los dígitos a grados utilizando la función:

```
AnguloFlexion = LecturaSensor*Factor + Offset
```

Dependiendo del lenguaje utilizado, C o Java, la variable *AnguloFlexion* puede ser un tipo de dato (por ejemplo un entero), una estructura de datos o bien un objeto. La representación de la información depende en gran medida de las necesidades de la aplicación y de las facilidades que ofrezca el lenguaje. Una vez que la información es representada con alguna entidad lógica entonces es posible compartirla con algún otro módulo, como es el caso del módulo de comunicación o bien otra aplicación.

2.2.4. Comunicación

Este módulo del sistema provee los recursos necesarios para permitir la transferencia de información entre el módulo de procesamiento de información y alguna aplicación externa. La comunicación se realiza a través de sockets utilizando el protocolo TCP/IP, de esta manera la información puede ser enviada a procesos residentes en la misma computadora o bien en otras computadoras, aún si estas tienen diferentes sistemas operativos. Las condiciones necesaria para establecer comunicación con estas aplicaciones externas es que estas soporten la comunicación por sockets y conocer el protocolo de comunicación. El esquema de comunicación consiste en los siguientes pasos:

1. Iniciar un proceso que atienda los puertos en espera de alguna petición de inicio de comunicación.

2. Al producirse dicha petición se crean uno o más sockets de acuerdo a las necesidades de la aplicación.

3. Asignar a cada socket un controlador de flujo de información.

4. Enviar y recibir información a través de los controladores de flujo en forma de cadenas de texto.

Existen muchos lenguajes que soportan la programación con sockets, entre ellos están java y C. Cabe mencionar que este módulo puede ser programado en un lenguaje diferente al que pudiera estar programado el módulo de procesamiento de información. Los datos provenientes de este último pueden hacérsele llegar de alguna otra forma de comunicación interprocesos que ofrezca el sistema operativo (por ejemplo un pipe en UNIX).

Se describirá primero el módulo de comunicación programado en Java por ser el más general y a continuación el que ha sido programado en C ya que este permite la comunicación en específico con Maya.

Módulo de comunicación en lenguaje Java

El cuerpo principal de este módulo se encuentra en el método servidorMocap del objeto Servidor (para mayor información referirse al apéndice correspondiente). Las variables de instancia que se utilizan en dicho método son las siguientes:

```
ServerSocket listen_socket;          /*Socket para recepción de peticiones*/
Socket socket,                        /*Socket*/
DataInputStream datosDelCliente,      /*Controladores de flujo de
PrintStream datosAlCliente;           información*/
```

El proceso de espera en atención a peticiones de comunicación se levanta utilizando la clase ServerSocket que proporciona Java:

```
listen_socket = new ServerSocket(puerto);
```

Una vez que el proceso en de espera recibe una petición se crea el socket de comunicación:

```
socket = listen_socket.accept();
```

Finalmente se crean los controladores de flujo de información para el socket:

```
datosDelCliente = new DataInputStream(socket.getInputStream());
datosAlCliente = new PrintStream(socket.getOutputStream());
```

Una vez que se han inicializado los controladores de flujo entonces puede enviarse o recibirse información a través de ellos en forma de pequeños mensajes de caracteres, tal y como se haría al enviar información a la pantalla:

```
datosDelCliente.readln(datos);
datosAlCliente.println(datos);
```

El manejo de la información a través de mensajes de caracteres permite la comunicación entre procesos de forma transparente entre diferentes equipos de cómputo e inclusive diferentes sistemas operativos. En lo particular el módulo que se muestra en el apéndice envía los datos capturados a través de un socket en forma tabular. A través del método println() se envía a algún proceso cliente una cadena cuyos elementos están separados por un espacio, cada uno de estos elementos representa la flexión de un sensor en cada momento. El proceso cliente únicamente tiene que leer la línea de datos, tomar cada uno de los elementos de la cadena e interpretarlos como un ángulo. La aplicación BrowserMocap que se describe en el tema de aplicaciones permite visualizar la representación del cuerpo humano a través de modelos 3D. Los datos que describen el movimiento y que permiten animar dicha representación pueden obtenerse directamente del módulo de procesamiento o bien a través de la red utilizando el módulo de comunicación.

Módulo de comunicación en lenguaje C

El cuerpo principal de este módulo reside en ServidorMocap.c, donde trabaja en conjunto al módulo de procesamiento de información dentro de un mismo proceso. Esta versión del módulo de comunicación permite el intercambio de información en forma directa con el software de Maya a través de sockets. Sin embargo, el protocolo que exige este software tiene una mayor complejidad que el ya expuesto en el caso de Java, donde los datos simplemente se transferían en forma tabular. Para los fines de este trabajo no se considera necesario programar dicho protocolo, por lo que se utilizan directamente las facilidades de comunicación a través de sockets que proveen la bibliotecas de captura de movimiento de Maya [14]. El esquema de trabajo es muy similar a la versión en Java, solo que en este caso se crean 64 sockets de comunicación con sus respectivos controladores de flujo de información. El número de

canales de comunicación se debe a que Maya maneja un socket por cada variable de captura, como pueden ser posición en el espacio x, y, z, o bien la rotación sobre alguno de estos ejes entre otros. A continuación se muestran fragmentos del código siguiendo el esquema de trabajo propuesto al inicio de esta sección. Las principales variables que se utilizan en el proceso de comunicación son las siguientes:

```
static CapChannel canales[64];/*Canales de captura (socket y manejador)*/
int client_fd;                          /*Socket para recepción de peticiones*/
```

El proceso de espera en atención a peticiones de comunicación se levanta utilizando la funcion CapServe que proporcionan la bibliotecas de Maya:

```
client_fd = CapServe(server_name);
```

Una vez que el proceso en de espera recibe una petición se crean los socket s de comunicación y los manejadores de información:

```
for(i=0;i<64;i++)                       /* Genera 64 canales de comunicación */
{
  itoa(i,nombreCanal,10);      /* Nombre del canal 1-64 */
  canales[i]= CapCreateChannel(nombreCanal,CAP_USAGE_ZROT, 1);
}
```

Una vez que se han inicializado los controladores de flujo entonces puede transferirse la información correspondiente a cada uno de los canales de captura utilizando la función CapSetData:

```
for(i=0;i<64;i++)                       /* Envía la información de los 64 canales */
       CapSetData(canales[i], &angle);
```

Siguiendo este esquema, la información registrada durante el proceso de captura de movimiento puede verse reflejada en el software de Maya como cambios de valor en determinadas variables. En este punto es posible utilizar dichas variables a nuestra conveniencia, un ejemplo puede ser el rotar una entidad gráfica para emular la articulación cuyo movimiento está siendo capturado. Este sencillo ejemplo es la base para generar la animación de personajes virtuales y será descrito con mayor detalle en los siguientes temas.

2.2.5. Aplicaciones

La arquitectura del sistema de captura permite programar aplicaciones que utilicen la información proveniente del módulo de procesamiento en forma directa y sin recurrir al módulo de comunicación. Esto es posible ya que la información capturada puede manejarse como un entidades (objetos, arreglos, apuntadores, etc.) que pueden ser compartidas con la aplicación para su uso particular. En el caso de las aplicaciones hechas con Java estas pueden utilizar la información del objeto ServidorMocap a través de los métodos que para ello provee, otra opción es consultar directamente las

variables de instancia de este objeto. Para una descripción mas detallada el lector deberá referirse al apéndice correspondiente a ServidorMocap.java.

A fin de mostrar algunas de las características del sistema se desarrollo una pequeña aplicación denominada BrowserMocap. Esta aplicación genera una representación tridimensional del cuerpo humano y a través de la información proveniente del objeto ServidorMocap da movimiento a cada una de las partes móviles de éste.

La representación de los objetos en forma tridimensional se logra utilizando las bibliotecas java3D [15].Para generar la representación del cuerpo esta aplicación utiliza entidades denominadas como eslabones. Estas entidades están formadas básicamente por una representación gráfica (geometría y apariencia) y un punto de rotación (sobre el cual se mapea la flexión de las articulaciones del cuerpo). Los eslabones se acoplan unos con otros formando así un modelo jerárquico.

La calidad gráfica que se logra es limitada, sin embargo esta pequeña aplicación puede utilizarse de muchas maneras: como una sencilla herramienta para la calibración de los sensores, o bien como un software para animación de personajes virtuales en tiempo real. La aplicación cuenta únicamente con 2 ventanas: la de visualización y la de control. En la figura 2.8 se muestra la representación de una mano generada en la ventana de visualización, el movimiento que se utiliza para animarla se obtiene a través del prototipo del sistema de captura (figura 2.7).

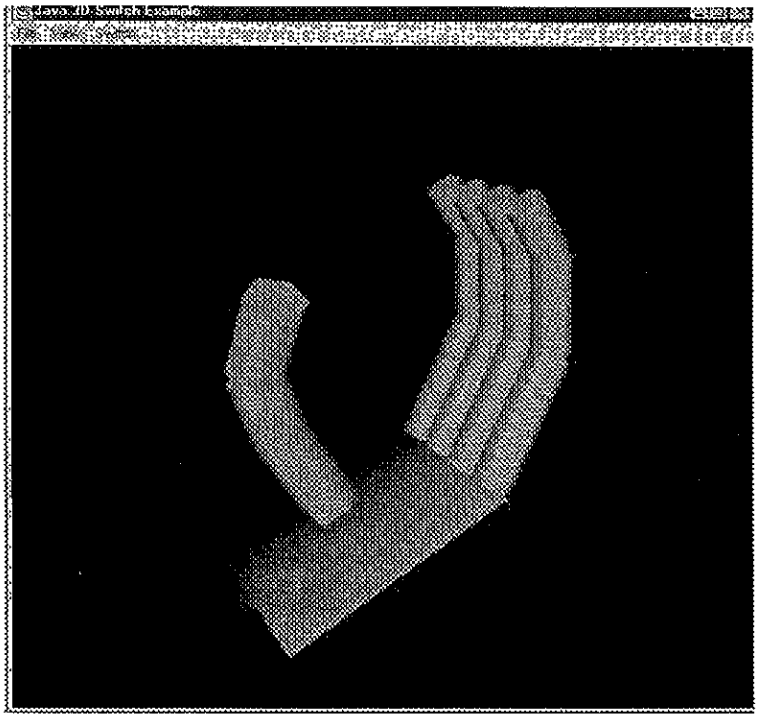

Figura 2.8. Representación gráfica de una mano en Java 3D utilizando geometrías básicas.

La figura 2.9 muestra el panel de control de esta aplicación, donde los atributos de cada elemento móvil que forma la representación gráfica pueden ser modificados. Por el

momento estos atributos se refieren únicamente a los parámetros de la función lineal que permite mapear la información capturada de los sensores. Al tener control de estos atributos es posible calibrar el sistema para diferentes usuarios, al tiempo que puede apreciarse en tiempo real la captura de sus movimientos.

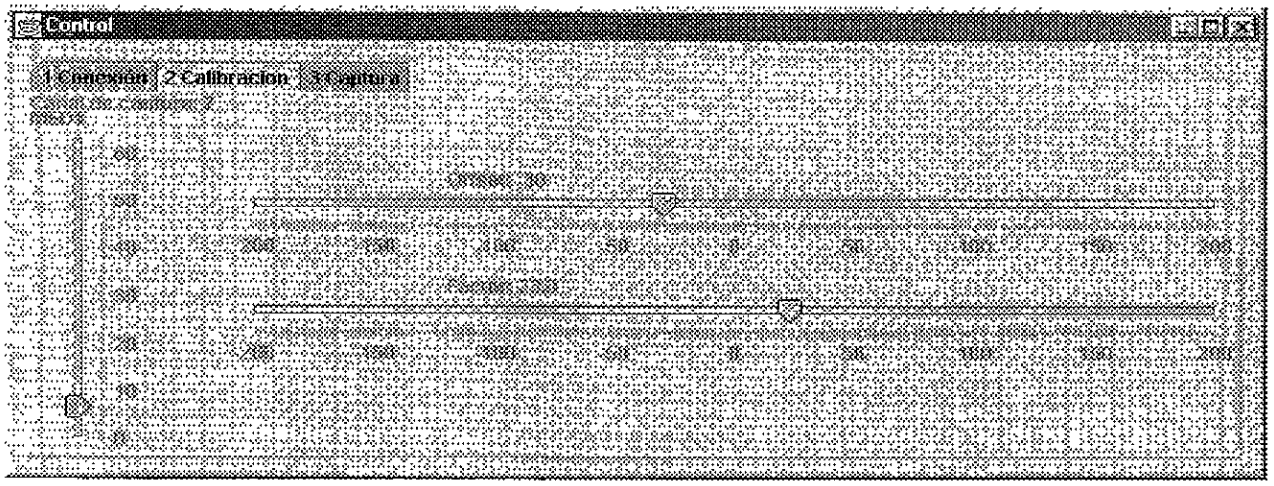

Figura 2.9. Panel de control de la aplicación browserMocap.

La geometría de los eslabones que se utilizan para representar las partes móviles del cuerpo puede ser adaptada fácilmente a cada necesidad. En la figura 2.10 se muestra nuevamente la representación de una mano, los eslabones hexagonales han sido sustituidos por otros elementos cuya geometría y textura aparentan ser huesos .

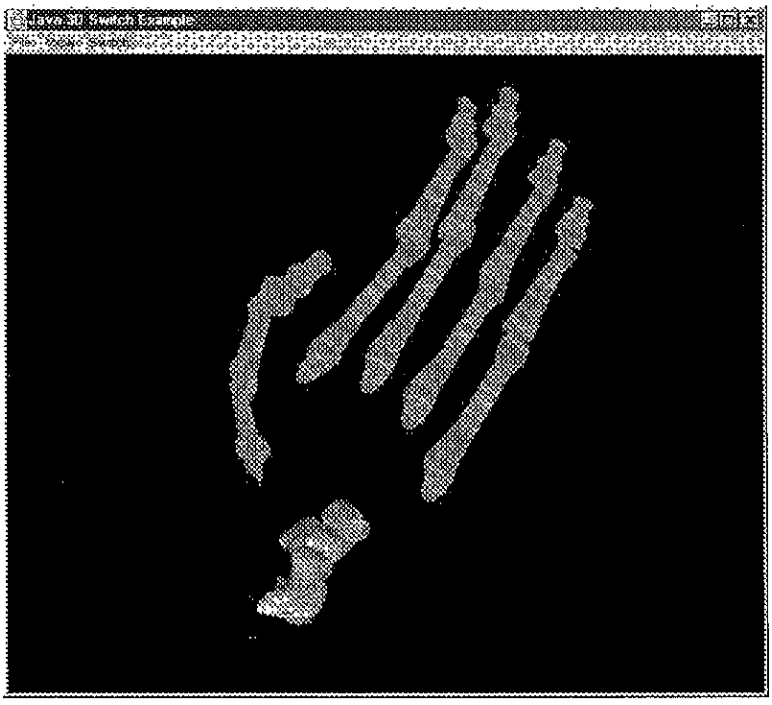

Figura 2.10. Representación gráfica de una mano en Java 3D. En este caso se utilizan geometrías y texturas que proveen mayor realismo y calidad gráfica.

Como se mencionó antes, esta aplicación en su estado actual puede ser utilizada para animar personajes virtuales en tiempo real. Sin embargo la calidad gráfica puede mejorarse utilizando software especializado en animación. En este punto es donde pueden apreciarse de mejor manera las facilidades que ofrece la arquitectura del sistema para comunicarse con aplicaciones de terceros como lo es Maya.

2.3. Aplicaciones externas (comunicación con Maya)

Maya es un software de animación, cuya principal característica es que permite dotar de movimiento a entidades gráficas a través captura de movimiento. El proceso de animación por captura de movimiento utilizado involucra los siguientes pasos:

1. **Generación de esqueletos.** Se definen los eslabones (partes móviles) que formarán a la entidad que se desea dotar de movimiento [18].

2. **Modelado de entidades graficas.** Es la definición de la geometría que dará volumen a la entidad gráfica y sobre la cual se aplicará la apariencia de un material (color, texturas,etc.) [27].

3. **Asignación de elementos del esqueleto con geometría.** Cada uno de los eslabones que forman el esqueleto de nuestra entidad gráfica es asociado con la geometría. De esta manera al dotar de movimiento a los eslabones, estos arrastrarán consigo la geometría asociada a ellos [18].

4. **Asignación de apariencia.** En este paso se define la apariencia que tendrá la entidad gráfica (color, textura, etc.) y se aplica sobre la geometría a manera de una envoltura [17].

5. **Definición de comportamiento.** Es posible que se desee dotar de algún comportamiento en especial para las partes móviles de la entidad gráfica. Este comportamiento puede estar asociado a determinados eventos y por lo regular se utiliza para dotar de un mayor realismo y/o automatizar algunas animaciones. Por ejemplo, uno de los usos más comunes, es la simulación del movimiento muscular bajo la piel de los animales, esto se logra asociando flexores al ángulo de rotación de las articulaciones [18].

Cada uno de los pasos anteriores se muestran de forma gráfica en el apéndice correspondiente a la generación de entidades gráficas con Maya.

Una vez que se ha definido cada una de las partes que componen a la entidad gráfica, entonces es posible dotarla de movimiento utilizando la información proveniente del dispositivo de captura. El procedimiento necesario para mapear los canales de captura a las partes móviles del esqueleto se cita a continuación:

1. **Invocar el software Maya y cargar la entidad gráficas.** En nuestro caso se carga la representación gráfica del brazo derecho, cuya construcción se describió con anterioridad. En la figura 2.11 se muestra la representación gráfica dentro del ambiente de Maya, en ella se han sobre puesto los nombres asignados a los eslabones del esqueleto.

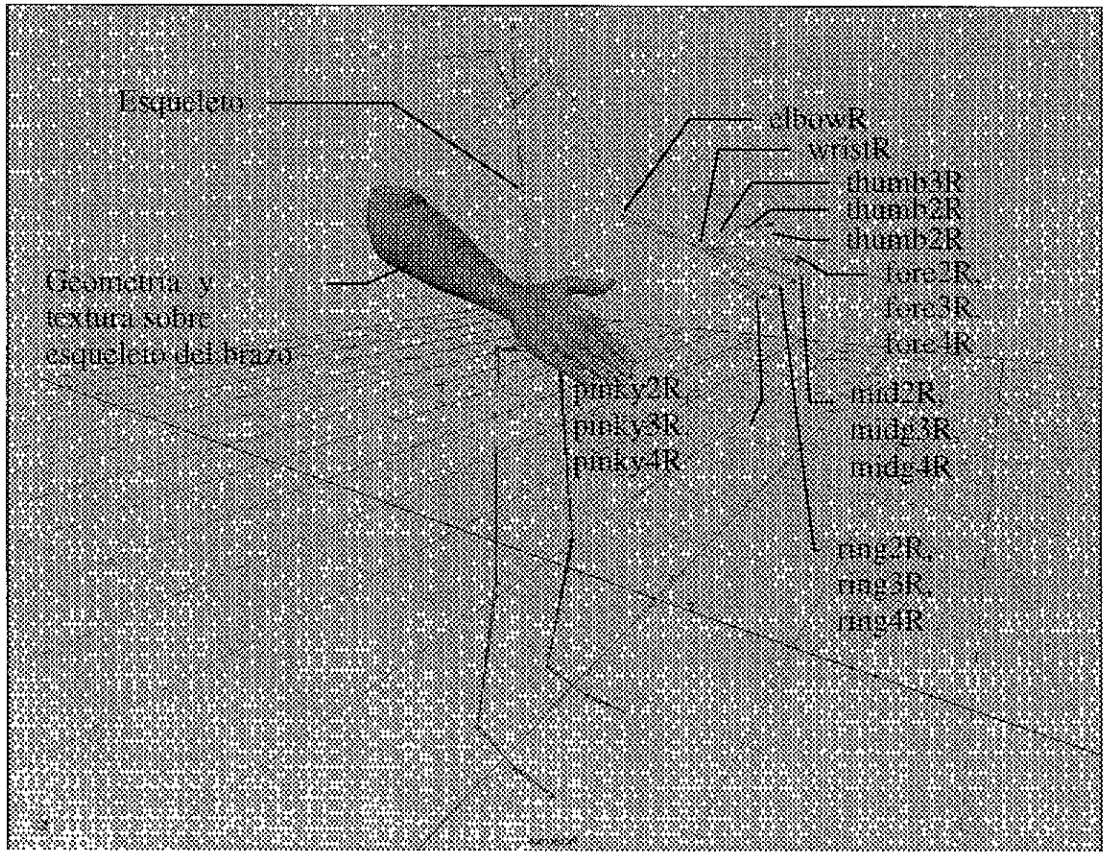

Figura 2.11. Elementos que componen la entidad gráfica que se dotará de movimiento.

2. **Levantar servidor de captura de movimiento.** Se inicia el programa que trabajará como servidor de captura desde el sistema operativo. Este servidor aguardará por un cliente para la transferencia de información a través de sockets, en este caso el cliente será el software de Maya.

3. **Conectar Maya con el servidor y mapear entidades gráficas con los canales de captura.** Una vez que se han cargado las entidades gráficas y se ha iniciado el proceso servidor (pasos 1 y 2), entonces deben asociarse los datos provenientes las partes móviles del esqueleto.

Este último paso puede realizarse desde la línea de comandos de Maya, o bien puede hacerse de forma automática utilizando un pequeño programa en MEL[6] como el que a continuación se describe (el código completo puede consultarse en el apéndice correspondiente):

Función principal desde la cual se inicia el proceso de conexión y mapeo de entidades.

```
global proc string mayaMano()
{
string $dName="mano";  /*Entidad que representa al dispositivo
                                de captura*/

string $sName="c1";    /*Nombre del proceso servidor de datos*/

/*Inicializa el servidor de datos*/

        if ( 0 != initDataServer($dName,$sName,"localhost", "", "",2))
        {
        error("Couldn't launch " + $sName );
        return "";
}

        /*Arreglo de transformaciones a través de las cuales se aplicarán
        las rotaciones correspondientes a los movimientos capturados*/

string $hands[17] = {
                        /*Articulaciones de los dedos*/
                        "fore4R", "fore3R", "fore2R",      /*Indice*/
                        "mid4R", "mid3R", "mid2R",         /*Medio*/
                                "ring4R", "ring3R", "ring2R",      /*Anular*/
                        "pinky4R", "pinky3R", "pinky2R", /*Meñique*/
                        "thumb3R", "thumbR2", "thumb1R", /*Pulgar*/

                        "wristR",                          /*Muñeca*/
                        "elbowR"};                         /*Codo*/

        attachHands($dName, $hands);  /*Asocia los elementos del brazo a la entidad que
                                        representa al dispositivo de captura*/
return $sName;
}
```

En el siguiente fragmento se mapea a cada uno de los elementos del esqueleto con los 64 canales de información provenientes del dispositivo de captura, se incluye para cada uno de ellos un factor de escalamiento y un offset.

```
proc attachHands(string $device, string $hands[] )
{
attach1D($device, "60", 0.005,  -0.3, $hands[0], "rz");
attach1D($device, "60", 0.005,  -0.3, $hands[1], "rz");
attach1D($device, "12", 0.005,  -0.2, $hands[2], "rz");
attach1D($device, "36", 0.005,  -0.4, $hands[2], "ry");

attach1D($device, "52", 0.005,  -0.7, $hands[3], "rz");
attach1D($device, "52", 0.005,  -0.6, $hands[4], "rz");
attach1D($device, "57", 0.005,  -0 3, $hands[5], "rz");

attach1D($device, "9",  0.005,  -0.4, $hands[6], "rz");
attach1D($device, "9",  0.005,  -0.5, $hands[7], "rz");
attach1D($device, "33", 0.005,  -0.3, $hands[8], "rz");

attach1D($device, "49", 0.005,  -0.5, $hands[9], "rz");
attach1D($device, "49", 0.005,  -0.6, $hands[10], "rz");
```

[6] Maya Embedded Language.

movimientos, lo cual es suficiente para atender a la mayor parte de las articulaciones del cuerpo.

- **Construcción sencilla**
  La construcción de los sensores es bastante simple, pues solo requiere de 3 elementos. La precisión no es crítica ya que los errores pueden ser corregidos vía software y/o durante la calibración. Todos los elementos que forman al sistema pueden adquirirse con facilidad en el mercado mexicano.

- **Calibración sencilla y rápida**
  La calibración del sistema se realiza a través de software, este proceso se basa en establecer un offset y un factor lineal para cada sensor. La calibración se efectúa la primera vez que el usuario utiliza el sistema, la anatomía particular de cada sujeto (configuración de sus articulaciones) y las irregularidades de cada sensor son normalizadas utilizando una función lineal donde interviene un offset y un factor.

- **No requiere ambientes controlados**
  Los sensores flexibles se acoplan a las articulaciones y miden en forma directa el ángulo de flexión. El sensor y la fuente están sobre el cuerpo del sujeto (inside in) y ambos elementos están dentro de un encapsulado que los aisla del medio ambiente que los rodea, de esta forma los cambios en este último no alteran el proceso de medición.

- **Independiente de plataforma**
  El software que corre en la computadora y que recibe la información del dispositivo de captura puede ser programado en Java, lo cual garantiza que puede operar en cualquier arquitectura para la cual exista una implementación de la máquina virtual de Java. En caso necesario el software del sistema puede programarse en cualquier otro lenguaje que permita tener acceso a los puertos seriales y comunicación por sockets, por ejemplo: C, C++, Phyton, Basic, pascal, ensamblador, etc.

- **Expandible y actualizable**
  La modularidad del sistema permite sustituir una o más de sus partes para mejorar el sistema, extenderlo o actualizarlo sin tener que afectar el funcionamiento del resto de los elementos que lo conforman.

Capítulo 3

Pruebas y resultados

En el capítulo anterior se describió el desarrollo del sistema de captura de movimiento, este proceso abarcó desde el diseño de los sensores hasta la construcción del hardware y la programación del software respectivo. A lo largo de todo el desarrollo se realizaron una serie de pruebas preliminares para comprobar el buen funcionamiento de cada módulo del sistema. En el presente capítulo se muestran brevemente los resultados obtenidos de estas pruebas, desde la comprobación del funcionamiento de los sensores hasta el uso del sistema como una aplicación final.

3.1. Captura del movimiento

Como ya se había mencionado en el capítulo anterior, los sensores desarrollados para el presente trabajo tienen la característica de cambiar sus resistencia eléctrica al ser flexionados. Dicha resistencia puede ser medida en forma indirecta utilizando un divisor de voltaje(ver figura 3.1). El circuito se alimenta con un determinado voltaje (V), se fija el valor de una de las resistencias (R1) y el sensor actúa como una resistencia variable (R2).

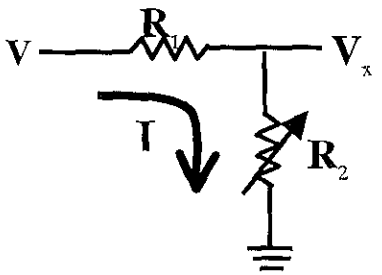

Figura 3.1. Sensor en una configuración de divisor de voltaje.

Bajo esta configuración el voltaje de salida Vx está en función de la resistencia del sensor, y su comportamiento queda definido por la fórmula:

$$V_x = R_2 \cdot \left(\frac{V}{R_1 + R_2}\right)$$

Como podemos observar el voltaje a la salida del circuito está en función de la resistencia del sensor. Si digitalizamos esta señal entonces podemos utilizarla en algún equipo de cómputo. En el experimento que se muestra en la figura 3.2 se utilizó un microcontrolador HC11 [12], el cual cuenta con un convertidor A/D y un puerto serial

La información a cerca del ángulo de flexión de las articulaciones puede ser utilizada para modificar objetos gráficos en la computadora, ya sea rotándolos o doblándolos, tal y como ocurre con la articulación real. Si además del ángulo conocemos las dimensiones de las partes del cuerpo que intervienen en el movimiento entonces es posible reconstruir su dinámica en forma realista a través de cinemática directa. Para simular la dinámica del cuerpo humano es necesario contar con un modelo jerárquico, el cual está formado por una serie de eslabones articulados, donde cada parte móvil depende de otra. Bajo este esquema al mover un eslabón cualquiera los eslabones que dependan de él deberán seguirlo para conservar su posición y orientación relativa.

En el experimento de la figura 3.4 se muestra una aplicación programada con Java 3D [15], con un modelo jerárquico que simula los huesos de la mano. Cada uno de los elementos que lo forman (eslabones) rotan sobre su eje emulando la articulación cuyo movimiento esta siendo capturado. Los resultados de esta prueba nos permiten corroborar en forma visual que la información registrada corresponde con la realidad.

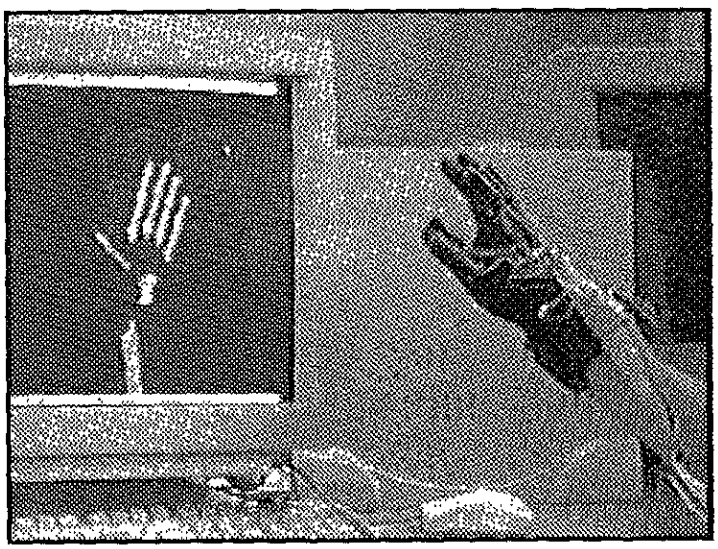

Figura 3.4. Visualización de la información registrada por el sistema de captura.

3.3. Animación por computadora y realidad virtual

Siguiendo con la línea de reconstrucción de movimiento se observó que el sistema de captura podía tener aplicación en el campo de la animación por computadora. Para probar esta posibilidad se decidió aprovechar la capacidad de comunicación del sistema a través de sockets y así transferir información al software de animación Maya [14]. Esta aplicación permite reconstruir el movimiento utilizando modelos jerárquicos y cinemática directa con una mayor calidad gráfica que la que se puede obtener con Java.

En el experimento que se muestra en la imagen 3.5 se observan dos monitores, el primero corresponde a un equipo PC donde se está capturando el movimiento y el segundo aun equipo Octane de Silicon Graphics [16]. En la PC corre la misma aplicación Java de captura y visualización descrita en el experimento anterior, y en la Octane se utiliza el software Maya[7]. Este último recibe la información en tiempo real y muestra el movimiento correspondiente a través de un esqueleto[8]. Maya permite representar en tiempo real la animación o bien grabar los movimientos para después hacer una posproducción de mayor calidad, donde los movimiento pueden ser filtrados o modificados de alguna forma [17].

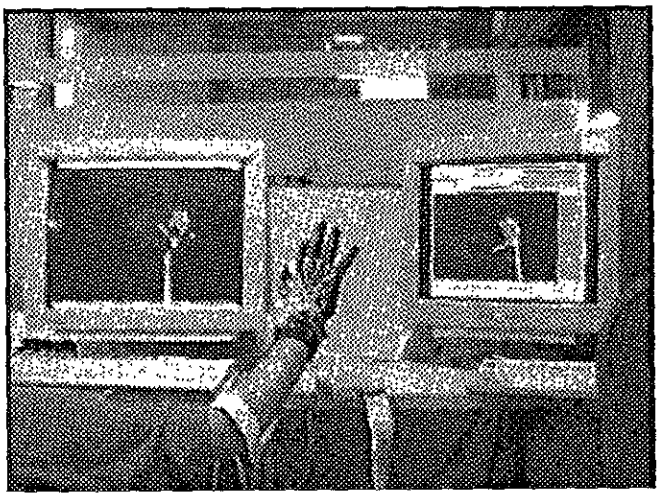

Figura 3.5. Captura de movimiento en un equipo PC y transferencia de la información a un equipo Silicon Graphics.

El esqueleto [18] que provee Maya puede ser recubierto de una geometría y cada vértice de ella puede ser asociado a un eslabón. Al mover los eslabones que forman el esqueleto la geometría sigue dicho movimiento dando la apariencia de volumen y elasticidad. Maya permite también incrementar el realismo a través de los flexores, los cuales permiten cambiar el volumen y forma de ciertas partes de la geometría[18], simulando el movimiento de la masa muscular bajo la piel. En la imagen 3.6 podemos apreciar el esqueleto de la imagen 3.5 recubierto por una malla de vértices que dan forma y volumen a lo que será la reproducción virtual de una mano.

[7] El lector deberá referirse al capitulo de desarrollo y a los apéndices correspondientes al final de este trabajo para una mejor descripción de algunos detalles a cerca de Maya y como se utilizó en este proyecto.

[8] Un esqueleto en el software Maya es un modelo jerárquico, en si es solo una conceptualizacion, sin embargo se representa como una serie de eslabones sobre los cuales se pueden montar objetos gráficos (vértices, nurbs, etc.).

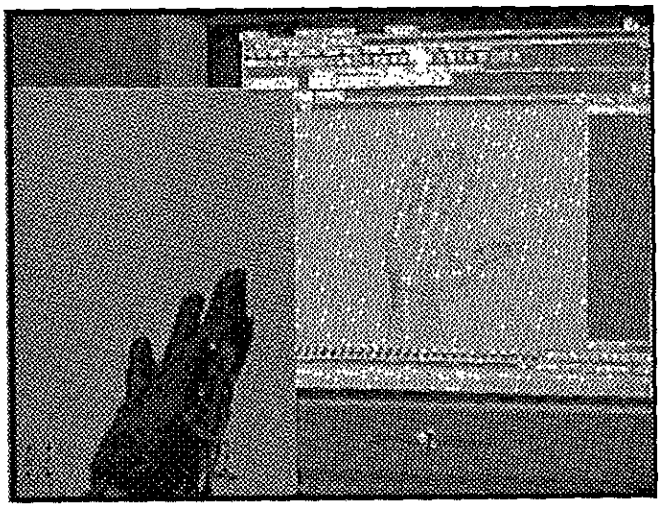

Figura 3.6. Representación del "esqueleto" de una mano recubierto por una geometría simulando el volúmen de esta.

Finalmente, a la malla que recubre el esqueleto se le puede aplicar una determinada textura, esto permite dotar de un mayor realismo al objeto gráfico que se está representando. En la figura 3.7 puede observarse el frente y reverso de la representación virtual de una mano obtenida con el software de Maya. Cabe mencionar que, a pesar del realismo obtenido esta aplicación puede reproducir el movimientos en tiempo real.

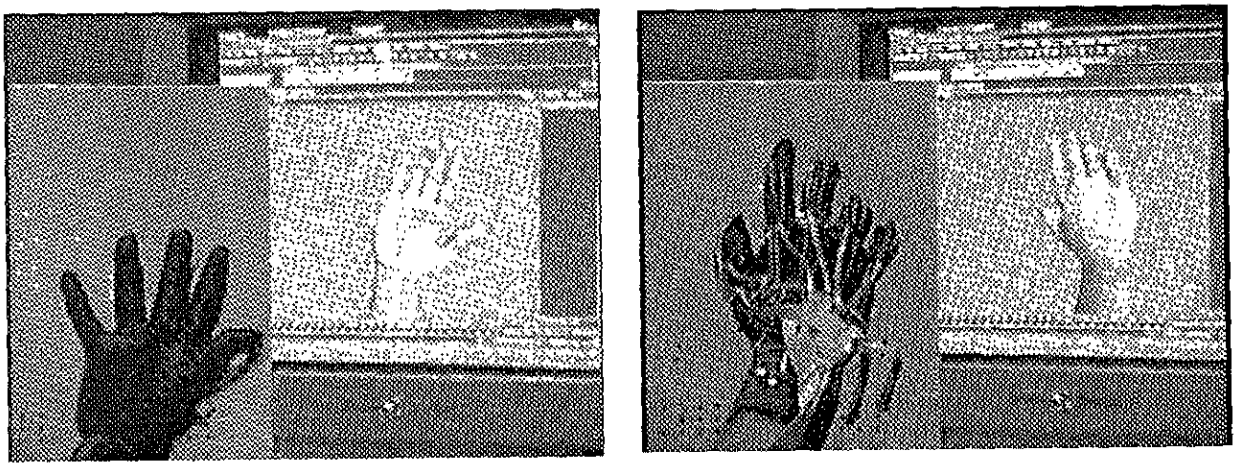

Figura 1.7. Vista frontal y posterior de la representación gráfica de la mano derecha..

El software Maya también permite asignar un determinado comportamiento a las entidades y objetos gráficos, este comportamiento se basa en definir las acciones que deberán ejecutarse al dispararse determinados eventos. Esta funcionalidad que ofrece Maya es de gran utilidad para simular la interacción con ambientes virtuales. Más adelante se describirá su uso para emular algunas interfaces. En el ámbito de la animación el proveer de comportamiento a algunas entidades permite generar

animaciones de personajes ricos en movimientos sin necesidad de postproducción. Dentro de estos casos podemos mencionar la animación de personajes cuyos movimientos no necesariamente corresponden a los movimiento del cuerpo humano, por ejemplo: la animación de algún animal con mas de 4 extremidades, robots con manipuladores de mas de 7 grados de libertad (el brazo humano solo tiene 7 grados de libertad), etc. Para cualquiera de los anteriores los movimientos extras necesarios pueden ser generados a partir de otros y en función de algún comportamiento. En la secuencia que se muestra en la figura 3.8 se observa la representación de una mano a la cual se le ha programado un determinado comportamiento: al doblar los nudillos a determinado ángulo comienzan a salir unas cuchillas de ellos. El efecto se logra a través del escalamiento de las cuchillas en función del ángulo de flexión de los nudillos. Este sencillo experimento nos servirá más adelante para simular el comportamiento de dispositivos como son los botones de un teclado.

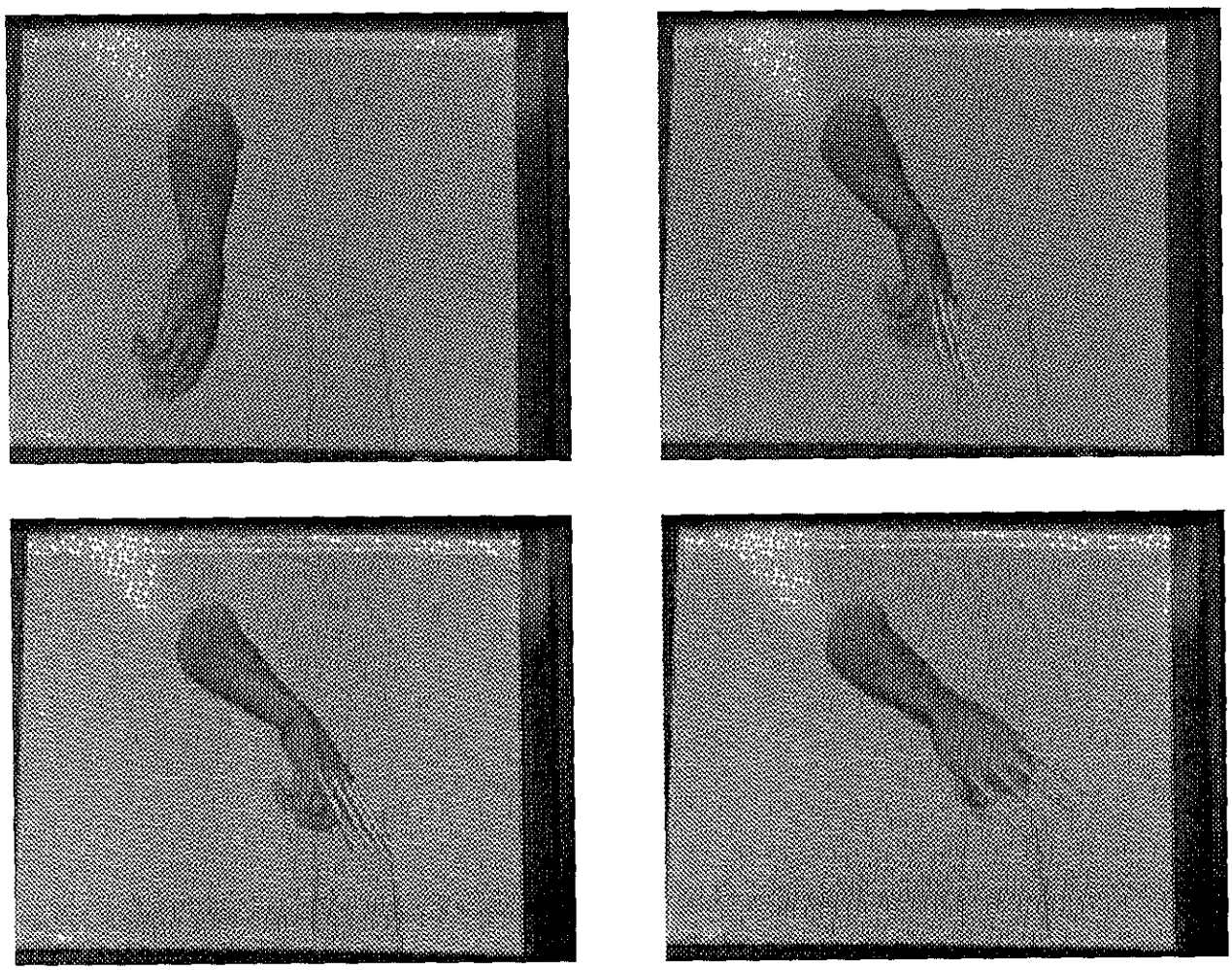

Figura 3.8. Secuencia del comportamiento de unas cuchillas acopladas a la mano de un personaje. El tamaño de éstas es manejado por la magnitud de la flexión de los nudillos.

3.4. Emulación de interfaces

Una vez que se ha probado el sistema de captura, y que se ha demostrado que la información registrada corresponde con los movimientos ejecutados en la realidad, entonces es posible tratar de interpretarlos dentro del contexto de una interfaz. En esta sección se muestran un par de experimentos, el primero de ellos se refiere a la emulación de un teclado para computadora y el segundo a un apuntador en 3D.

3.4.1. Teclado

En el manejo de un dispositivo como es el teclado de la computadora pueden intervenir muchos movimientos corporales que en realidad no son necesarios. Con la finalidad de simplificar la interfaz para esta prueba se hacen algunas consideraciones:

- Para hacer el recorrido de la mano sobre el teclado se utiliza únicamente el movimiento del codo (figura 3.9 a), por lo ello la representación de este dispositivo virtual toma una forma curva. En un teclado real el sutil movimiento del hombro permitiría seguir su forma recta, sin embargo esto no es necesario para emular su funcionamiento.

- Una vez que se ha seleccionado una de las filas de botones sobre el teclado con el movimiento del codo, entonces es posible seleccionar uno de ellos y activarlo utilizando el movimiento de los dedos y de la muñeca (figura 3.9 movimientos: b,c y d).

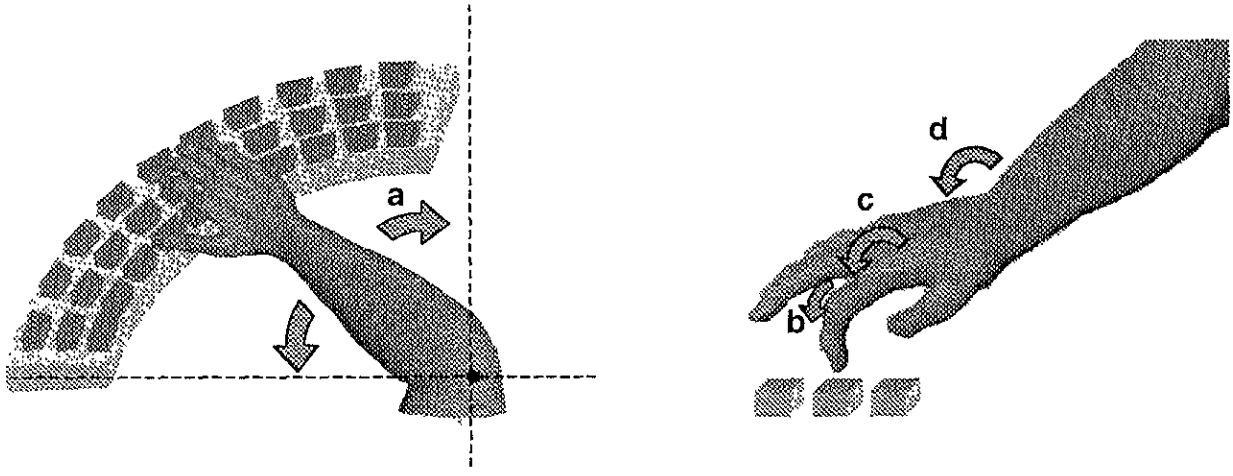

Figura 3.9. Esquema de trabajo del teclado virtual.

El experimento realizado para demostrar el funcionamiento de este teclado virtual se realizó en el ambiente del software Maya, con el cual es posible simular la interacción con objetos en un ambiente virtual. En esta aplicación se probó un sencillo teclado virtual, la opresión de los dedos de la mano sobre los botones fue posible utilizando la

detección de colisiones entre objetos que ofrece Maya. El lector puede consultar el código fuente del programa en MEL [19] que permite emular este dispositivo en el apéndice correspondiente.

En la imagen 3.10 se muestra en primer plano, el guante del sistema y un aditamento que fija un sensor al codo. Al fondo de la misma imagen puede observarse una pantalla con la representación de 4 botones, cada uno de ellos con una "dirección" asignada y que en conjunto representan los botones de control del cursor de un teclado real. Al presionar cada uno de estos botones la cruz que esta frente a ellos gira en la dirección correspondiente con un ángulo proporcional a la presión ejercida.

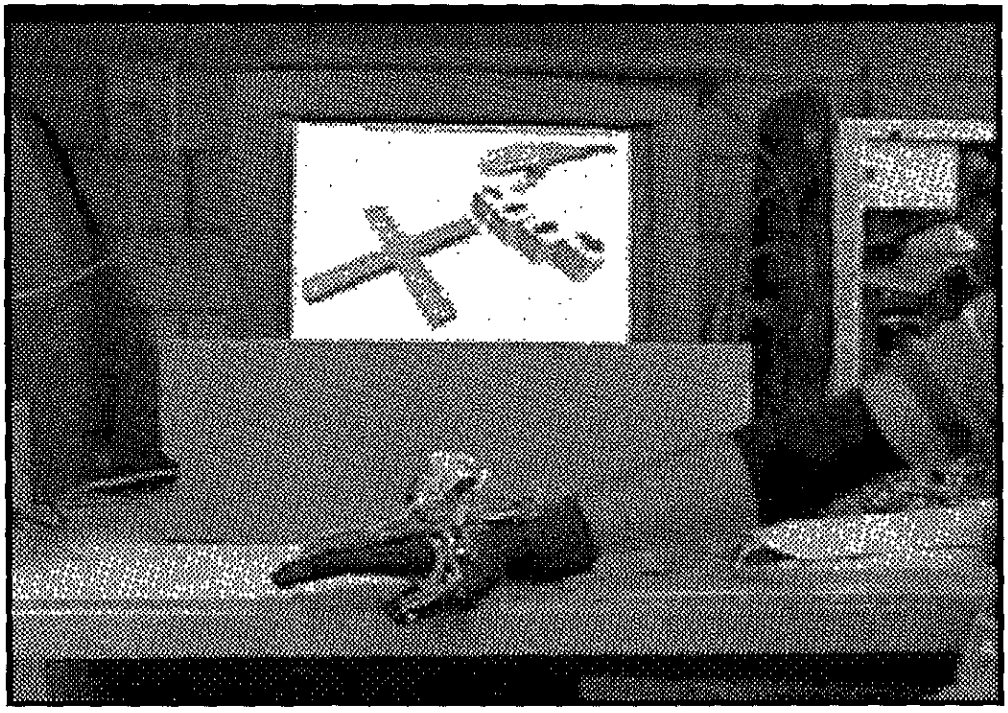

Figura 3.10. Sistema de captura conectado a la aplicación Maya para emular un teclado virtual.

La secuencia de eventos generados al oprimir los botones con las flechas "derecha", "abajo", "arriba" e "izquierda" se muestran en las imágenes de la figura 3.11. Cabe mencionar que con esta prueba se demostró que a través de esta interfaz no solo es posible seleccionar objetos y detectar el contacto con ellos, si no también cuantificar la opresión sobre cada uno y con ello activar determinados eventos. Es fácil observar que la emulación de otro tipo de dispositivos como perillas, válvulas, interruptores, palancas de juegos, monitores sensibles al tacto, etc., etc., puede realizarse con facilidad.

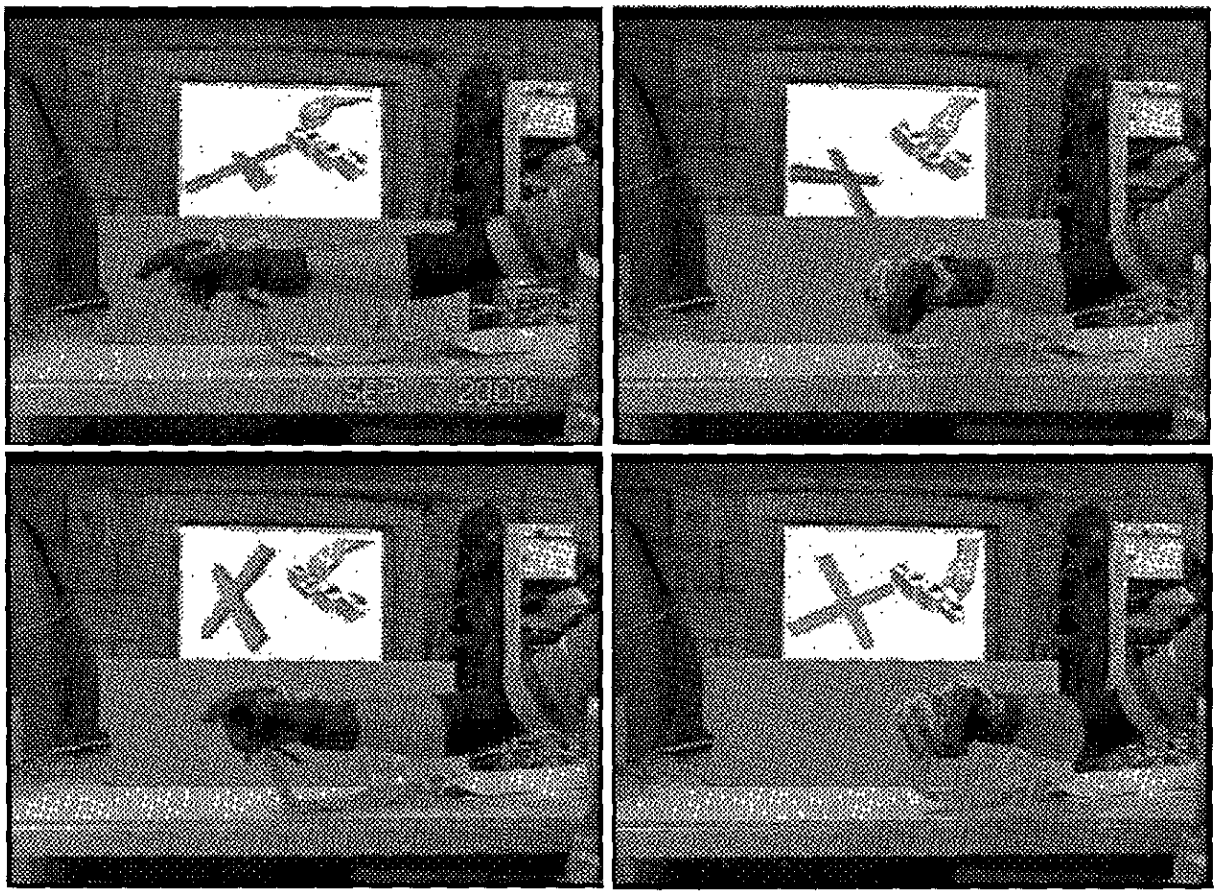

Figura 3.11. Secuencia donde se muestra el teclado virtual y su comportamiento al interactuar con cada uno de sus elementos.

3.4.1. Apuntador 3D

En este experimento se trata de emular el funcionamiento de un apuntador en 3D (un mouse es un apuntador en 2D). El objetivo es demostrar la posibilidad de seleccionar objetos en un espacio tridimensional utilizando el sistema de captura. El esquema de trabajo es bastante sencillo, simplemente se mapea la magnitud de la flexión del codo y la muñeca como una magnitud de desplazamiento en cada uno de los ejes coordenados X, Y y Z (ver figura 3.12). Para seleccionar un objeto simplemente se le toca con la representación del dedo índice, que en este caso es un cursor. En apariencia el mapeo de coordenadas no resulta natural, sin embargo, en la practica el uso de esta interfaz resulta bastante intuitiva. Con este experimento se reafirma la flexibilidad que tiene el sistema para emular cualquier otra interfaz, ya sea que esta exista o bien sea una en proceso de diseño. Con esta última observación queremos hacer ver al lector que el sistema servir incluso como un auxiliar para el diseño y desarrollo de interfaces [20].

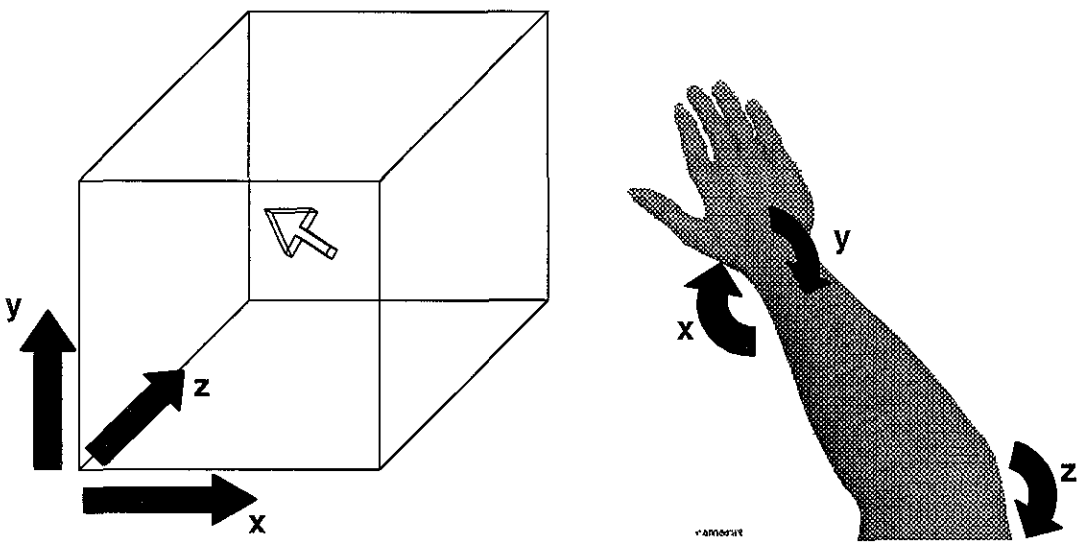

Figura 3.12. Esquema de trabajo del apuntador en un espacio tridimensional.

3.5 Conclusiones

Para finalizar el presente capitulo mencionaremos algunas de las principales características del sistema observadas durante el desarrollo y prueba del mismo:

- En las pruebas mostradas en este capítulo se mostró que las características de los sensores desarrollados para este sistema permiten registrar la flexión de la articulaciones del cuerpo de una forma sencilla, sin necesidad de complicadas armaduras y sin estorbar su movimiento.

- El esquema de trabajo para la reconstrucción del movimiento corporal en la computadora es también una tarea sencilla. Cada uno de los ángulos registrados en las coyunturas del cuerpo es aplicado a las respectivas articulaciones en un modelo jerárquico que representa al humano en la computadora, a fin de que este adopte las mismas posturas.

- A través de los experimentos para visualizar la reconstrucción de los movimientos corporales se observó que el sistema podría tener aplicación en otros campos como lo es la animación por computadora.

- Una vez que el movimiento corporal es capturado y mapeado a un modelo jerárquico en la computadora, entonces es posible tratar de interpretar las diferentes posturas que este adopta. Como se mostró en los experimentos de este capítulo los movimientos del brazo y los dedos pueden interpretarse como pulsaciones en un teclado o un aputandor en 3D. En otras aplicaciones los movimientos del pie pueden activar el freno o acelerador de algún

dispositivo, el movimiento de la cabeza puede dirigir el movimiento de una cámara, y así cualquier otra aplicación similar para emular interfaces mecánica/electrónicas[9].

- Si el sistema permite emular las interfaces mecánica/electrónicas existentes, entonces también puede utilizarse como auxiliar en el diseño de nuevos dispositivos de entrada y probar su desempeño sin necesidad de construirlos físicamente.

- Por sus características como dispositivo de entrada, y por su bajo costo, el sistema de captura aquí descrito puede representar una buena opción como interfaz para juegos y aplicaciones educativas, en especial aquellas que se desarrollan en ambientes virtuales.

- El sistema permite realizar animación en tiempo real.

- A través de su módulo de comunicación por sockets permite transferir información a la mayoría del software de animación comercial (Maya, 3D Studio, etc.).

[9] Aquella que requieren la interacción mecánica de alguna parte del cuerpo humano para producir señales eléctricas, las cuales podrá interpretar la computadora a través de una interfaz electrónica.

Capítulo 4

Conclusiones y trabajo a futuro

Al inicio del presente trabajo se habló de la necesidad de incrementar la capacidad de interacción del usuario con la computadora. Esto a fin de explotar de mejor manera las tecnologías emergentes, y nuevos esquemas de trabajo y comunicación a través de la computación.

Entre las tecnologías emergentes se mencionaron los ambientes virtuales, en especial el problema que se tiene al desempeñar alguna tarea en ellos. La falta de interactividad del usuario con el ambiente virtual no permite explotar del todo la habilidad y destreza que tiene el humano para desenvolverse en un espacio tridimensional. Esta falta de interacción con la computadora se debe principalmente a que la mayoría de los dispositivos de entrada solo registran acciones específicas (oprimir un botón, deslizar un apuntador, etc.). Aun que la finalidad de esto es simplificar la tarea del usuario tiene también el defecto de que disminuye su capacidad expresiva.

Con el fin de ofrecer una posible solución a este problema se propone crear un dispositivo de entrada que permita registrar la mayor parte de los movimientos corporales del usuario, de tal forma que éste cuente con una mayor gama de elementos de expresión para comunicarse con la computadora.

Para demostrar la validez de la propuesta se desarrolló un prototipo del sistema y se realizaron algunas pruebas. El dispositivo desarrollado utiliza técnicas de captura de movimiento y permite registrar los movimientos de una mano y parte del brazo. Se consideró que el registro de estos movimientos era suficiente para la mayoría de las demostraciones posibles en una computadora de escritorio o estación de trabajo, es decir, considerando que no se tiene mas que un equipo de realidad virtual proyectivo.

Entre las pruebas más significativas están aquellas en las que se mostró la emulación de un teclado y de un apuntador tipo *mouse*, ya que con ello se demuestra, entre otras cosas, que es posible manejar con este sistema un conjunto significativo de las interfaces actuales. Considerando que los dispositivos de entrada actuales capturan solo un pequeño subconjunto de las expresiones corporales, entonces es factible pensar que un dispositivo como el que se propone en esta tesis permita al humano tener una mayor interactividad con la computadora. Cabe mencionar que la emulación de dispositivos de entrada puede servirnos para el diseño y mejora de estos sin necesidad de construir prototipos. Otra posibilidad es que en realidad no se necesite otro dispositivo de entrada y que solo se utilicen las emulaciones como un modelo "conveniente" para usuario, como el teclado, y el *mouse* virtuales que se mostraron en este trabajo.

Características generales del sistema

Como resultado del diseño, desarrollo y prueba del prototipo se observaron en él las siguientes características:

- La arquitectura modular del sistema permite la sencilla sustitución de elementos, por lo que es posible extenderlo y mejorarlo a futuro.
- Construcción sencilla y de bajo costo.
- Elementos electrónicos pueden adquirirse a bajo costo en el mercado mexicano.
- Los sensores no requieren ambientes controlados y la obstrucción del libre movimiento que provocan en las articulaciones es minima.
- Calibración rápida y sencilla
- El software del sistema es independiente de plataforma.
- Por su bajo costo, el sistema permite abaratar tareas cuyo precio puede considerarse alto, tales como animación por computadora, incluso en tiempo real y aún con equipo PC convencional.

Como mejoras al sistema y trabajo a futuro podemos mencionar los siguientes puntos:

- **Incremento de la velocidad de captura y transferencia.** Por el momento el cuello de botella del sistema es la transferencia de datos por el puerto serial, ya que la etapa de conversión A/D es capaz de manejar hasta 10,000 conversiones por segundo, sin embargo solo la décima parte son transferidas. La velocidad de comunicación puede incrementarse de 9,600 a 115,200 bps cambiando el reloj del sistema por uno de precisión de 7.3728 MHz (el reloj actual es de 8 MHz).

- **Comunicación inalámbrica:** El prototipo actual del hardware de captura requiere estar conectado a través de un cable serial a una computadora. En algunos casos este cable puede llegar a estorbar el libre movimiento del usuario, por lo que un sistema inalámbrico resulta de mayor utilidad. Para incorporar esta característica a nuestro prototipo hemos pensado en utilizar las tarjetas de comunicación inalámbrica WaveLAN [21]. Esta línea de productos incorpora una solución para dispositivos que requieren comunicación serial, en la cual el diseño de nuestro prototipo no requiere mayores cambios. La comunicación serial es uno de los medios de comunicación más extendidos para dispositivos electrónicos como son estaciones de trabajo, computadoras personales, laptops, palmtops, e inclusive algunas calculadoras. Tal aceptación para este medio de comunicación nos permite garantizar que el dispositivo podrá utilizarse con la mayoría de las computadoras actuales, o bien dado el caso, de que cierta computadora no lo soporte entonces seguramente existirá algún convertidor. Las

máquinas de Apple han adoptado los estándares USB [22] y Fire Wire [23], si se requiriera utilizar nuestro dispositivo de captura en estas computadoras, entonces podemos utilizar un convertidor serial/USB. Conforme las computadoras dejen de dar soporte al estándar de comunicación serial RS232 y se cambie por el de USB entonces podrá pensarse en cambiar el HC11 del sistema por otro chip de Motorola denominado como HC05. Ambos microcontroladores pertenecen a la misma familia, por lo que la migración es transparente y los cambios son mínimos [24]. Existen otras opciones de migración, una de ellas es utilizar los dispositivos para desarrollo de periféricos USB de Cypress [25]. Otro medio de comunicación que es conveniente tomar en cuenta es a través de una red. Para este efecto existen convertidores del estándar RS232 a TCP/IP, una de las principales características de este tipo de dispositivo es el proveer de comunicación inalámbrica [21].

- **Reducción del sistema.** Es posible reducir el tamaño de los sensores utilizando componentes de montaje superficial acoplados a un cableado mas ligero, como pueden ser las fibras conductoras [26], circuitos impresos en acetato o alguna otra tecnología similar. El circuito también es susceptible de reducirse ya que del microcontrolador solo utilizamos el convertidor A/D y un puerto de comunicación. Una opción puede ser usar dispositivos mas especializados como un hc05 [24] o bien un PIC.

- **Posición y orientación del usuario.** Actualmente el sistema únicamente captura el ángulo de flexión de las articulaciones, sin embargo nos falta por resolver como obtener la posición y orientación del usuario. Una de las opciones que estamos investigando para obtener la posición, es el calcularla a través de cinemática directa. El proceso se basaría en obtener a cada momento el desplazamiento relativo utilizando el movimiento capturado de las piernas. Para obtener la orientación probablemente utilizaremos un dispositivo semejante a un péndulo o bien un giroscopio.

Apéndice A

Código fuente para el hardware de captura (ensamblador HC11)

Programa fuente para el microcontrolador HC11 F1 del hardware de captura de movimiento.

mocap.asc

```
                        org $fe00              /*Inicio de programa */

                ldaa #$30       /*Inicializa serial a 9600 bits
                                                por segundo*/
                staa $102b
                ldaa #$0c
                staa $102d

                ldaa #$80       /*Inicializa convertidores A/D*/
                staa $1039

        et1     equ *           /*Loop principal*/

                ldaa $1004      /*Selecciona canal en los 8 multiplexores*/
                inca            /*Incrementa contador para el selector, solo como los 3
                                      primeros son usados*/
                staa $1004      /*Selecciona canal en los 8 multiplexores usando el puerto
                                      b del hc11*/

                anda #07        /*Si es el primer canal (00) enviará cabecera*/
                bne  et3        /*Si no es asi entonces continuar en "et3"*/

                ldab #06        /*Envia cabecera del vector de datos */
                bsr writescsr   /*Envia por el serial*/
                ldab #06
                bsr writescsr

        et3     equ *
                ldaa #$b0       /*Inicia conversión (primeros 32 canales)*/
                staa $1030

        et2     equ *           /*Loop en espera del termino de la  conversión A/D*/
                ldaa $1030      /*Checa termino de conversion A/D*/
                anda #$80
                beq et2

                ldab $1031      /*Toma el dato leido en B para ser transferido (canales 1-8)*/

                bsr writescsr   /*Envia por el serial*/

                ldab $1032      /*Canales 9 al 16*/
                bsr writescsr

                ldab $1033              /*Canales 17 al 24*/
                bsr writescsr

                ldab $1034      /*Canales 25 al 32*/
                bsr writescsr


                ldaa #$b4       /*Inicia conversión (últimos 32 canales)*/
                staa $1030
```

```
et4             equ *           /*Loop de espera por conversión*/
                ldaa $1030      /*Checa termino de conversión*/
                anda #$80
                beq et4

                ldab $1031      /*Canales 33 al 40*/
                bsr writescsr

                ldab $1032      /*Canales 41 al 48*/
                bsr writescsr

                ldab $1033      /*Canales 49 al 56 */
                bsr writescsr

                ldab $1034      /*Canales 57 al 64 */
                bsr writescsr

                jmp et1


                /*Rutina de escritura de datos del registro b al serial*/
wcsr            equ *
                psha

rcsr            ldaa $102e      /*Checa disponibilidad de transmisión del
                                        serial*/
                anda #$40
                beq leescsr
                stab $102f      /*Transmite el contenido del registro b*/
                pula
                rts

end
```

Apéndice B

Código fuente del servidor de captura (Java)

Este programa corre bajo el sistema operativo (IRIX, Solaris o Windows 9x/2000/NT). Inicializa y mantiene la comunicación con el hardware de captura a través del puerto serial. La información que recibe es transferida a otras aplicaciones (por ejemplo, Maya) a través de sockets.

Servidor.java

```
import java.io.*;
import java.util.*;
import javax.comm.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.lang.reflect.Array;
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.image.*;
import com.sun.j3d.utils.geometry.*;


public class Servidor implements Runnable, SerialPortEventListener
{
    static CommPortIdentifier portId;
    static Enumeration portList;

    InputStream inputStream;
    SerialPort serialPort;
    Thread readThread;
    int cabecera;
    int[] offsets, lecturasMicro, channels;
    float[]  factores, rotInit,angulos,angulos2;
    String[] nombreCanal;
    static int actualElemento;        /*Ultimo elemento añadido (iniciando desde 0)*/
    static Eslabon[] rot;             /*Lista de rotaciones a afectar*/
    ControlMocap control;

    static int puerto=-1;
    ServerSocket listen_socket;
    Socket socket;
    DataInputStream datosDelCliente;
    PrintStream datosAlCliente;


/**************************************************************************/

    public Servidor(int port) {
          lecturasMicro=new int[100];
          offsets= new int[100];              /*Offset del sensor*/
          factores= new float[100];           /*Relacion grados/lecturas del micro*/
          rotInit= new float[100];
          angulos= new float[100];
          angulos2= new float[100];
          rot=new Eslabon[100];               /*Rotaciones*/
          channels= new int[100];             /*Asociacion de rotaciones y canales
                                              analogicos*/
        nombreCanal=  new  String[100];
          puerto=port;
          for(int i=0;i<100;i++) angulos[i]=0;
          for(int i=0;i<100;i++) factores[i]=1;
```

```
        control = new ControlMocap(this, "Control", nombreCanal, lecturasMicro, offsets,
                                   factores);
      control.pack();
      control.setVisible(true);

        if(puerto>0) servidorMocap();

        dispositivoSerial("COM2");
    }

/*************************************************************************/

   public void dispositivoSerial(String port)
    {
        CommPortIdentifier portId1;
      portList = CommPortIdentifier.getPortIdentifiers();
        cabecera =0;
      while (portList.hasMoreElements()) {
          portId1 = (CommPortIdentifier) portList.nextElement();
          if (portId1.getPortType() == CommPortIdentifier.PORT_SERIAL) {
              if (portId1.getName().equals(port)) {portId =portId1;}
          }
      }

      try {
          serialPort = (SerialPort) portId.open("LectorMocap", 2000);
      } catch (PortInUseException e) {}
      try {
          inputStream = serialPort.getInputStream();
      } catch (IOException e) {}
      try {
          serialPort.addEventListener(this);
      } catch (TooManyListenersException e) {}
      serialPort.notifyOnDataAvailable(true);
      try {
          serialPort.setSerialPortParams(9600,
              SerialPort.DATABITS_8,
              SerialPort.STOPBITS_1,
              SerialPort.PARITY_NONE);
      } catch (UnsupportedCommOperationException e) {}
      readThread = new Thread(this);
      readThread.start();

    }

/*************************************************************************/

   public void servidorMocap()
    {
        try
        {
            listen_socket = new ServerSocket(puerto);
        }
        catch (IOException e)
        {
         control.mensajeServidor(e+ "No se puede crear socket en el puerto " + puerto);
        }
        control.mensajeServidor("Servidor: escuchando en puerto " +puerto);

            try
            {
                socket = listen_socket.accept();
                control.mensajeServidor("Servidor:  cliente conectado ..... ");
            }
            catch (IOException e)
            {
                control.mensajeServidor(e+ "Servidor: error en la conexión del socket");
            }
```

```
        try
        {
            datosDelCliente = new DataInputStream(socket.getInputStream());
            datosAlCliente = new PrintStream(socket.getOutputStream());
        } catch (IOException e)
                {
                  try
                  {
                   socket.close();
                  }
                  catch (IOException e1){  }
                  control.mensajeServidor("Coneccion : error en la conexión del socket " );
                }

    }


/*************************************************************************/

/*Arreglo de transformaciones a rotar, canal asociado, y constantes de calibración
con las cuales interpretar la informacion del micro.*/

    public void addObject(String nom, Eslabon rotacion, int channel, int offset, float
    factor,float rotI)
    {
        nombreCanal[actualElemento]=nom;
        rot[actualElemento]=rotacion;
        channels[actualElemento]=channel;
        offsets[actualElemento]=offset;
        factores[actualElemento]=factor;
        rotInit[actualElemento]=rotI;
        actualElemento++;
    }


/*************************************************************************/
/*Manejador de los eventos proucidos por el puerto serial*/

    public void serialEvent(SerialPortEvent event) {
        int dato,numBytes, i=0;
      byte[] readBuffer;
        Transform3D yAxis;
        float angulo=0;
        switch(event.getEventType()) {
        case SerialPortEvent.BI: System.out.println("Break interrupt"); break;
        case SerialPortEvent.OE: System.out.println("Overrun error"); break;
        case SerialPortEvent.FE: System.out.println("Framing error"); break;
        case SerialPortEvent.PE: System.out.println("Parity error"); break;
        case SerialPortEvent.CD: System.out.println("Carrier detect"); break;
        case SerialPortEvent.CTS: System.out.println("Clear to send"); break;
        case SerialPortEvent.DSR: System.out.println("Data set ready"); break;
        case SerialPortEvent.RI: System.out.println("Ring indicator"); break;
        case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
                                                        break;

        case SerialPortEvent.DATA_AVAILABLE:
            readBuffer = new byte[1];
            try {
                while (inputStream.available() > 0)
                   {
                    numBytes = inputStream.read(readBuffer);
                              if ( cabecera == 0)
                              {
                              if ( (int)readBuffer[0] == 6 )
                                cabecera=1;          /*Delimitador del paquete*/
                              else
                                control.mensajeSerial("Sincronizando dispositivo..." );
                              }
                              else if ( cabecera == 1)
                                     if ( (int)readBuffer[0] == 6 )
                                     {
```

```
                                        cabecera = 2;  /*Delimitador del paquete*/
                                        }
                                        else    /*Perdida del paquete*/
                                        {
                                        control.mensajeSerial("Se perdio sincronizacion" );
                                        }
                                else
                                        {       /*Cuerpo del paquete */
                                        dato=(int) readBuffer[0];
                                        if(dato<0) dato=255+dato;
                                        dato= (dato + 1*lecturasMicro[cabecera-2])/2;
                                        lecturasMicro[cabecera-2]=(int) dato;
                                        cabecera++;
                                        if(cabecera == 66)
                                        {
                                        String datos="";
                                        cabecera=0;     /*final del paquete*/
                                        while (i<actualElemento) /*Actualiza rotaciones */
                                        {
                                        yAxis = new Transform3D();
                                        angulo=rotInit[i]+(lecturasMicro[(channels[i])]+
                                                                offsets[i] )*factores[i];
                                        angulo=(angulo+angulos[channels[i]]+
                                                                angulos2[channels[i]])/3;

                                        angulos2[channels[i]]=angulos[channels[i]];
                                        angulos[channels[i]]=angulo;
                                        rot[i].transRot.getTransform(yAxis);

                                        if(rot[i].ejeRotacion==1)yAxis.rotX(angulo);
                                        else if(rot[i].ejeRotacion==2)yAxis.rotY(angulo);
                                        else yAxis.rotZ(angulo);
                                        rot[i].transRot.setTransform(yAxis);
                                        i++;
                                        if (i>actualElemento) i=0;
                                        }

                                        //Si hay cliente conectado y se desea enviar
                                        for(i=0;i<64;i++)
                                        datos=datos + (angulos[i]/Math.PI*180)+ "     ";

                                        //Envia informacion
                                        if(puerto>0)
                                        datosAlCliente.println(datos);
                                        }
                                }

                        }
                } catch (IOException e) {}

                break;
        }
    }
}
```

Apéndice C

Código fuente del programa para la representación gráfica de los datos provenientes del proceso de captura de movimiento capturado (Java)

Este programa corre bajo el sistema operativo (IRIX, Solaris o Windows 9x/2000/NT). Genera la representación tridimensional del brazo derecho y trabaja junto con Servidor.java para dotarlo de movimiento con la información capturada de un brazo real.

BrowserMocap.java

```
import com.sun.j3d.loaders.objectfile.ObjectFile;
import com.sun.j3d.loaders.ParsingErrorException;
import com.sun.j3d.loaders.IncorrectFormatException;
import com.sun.j3d.loaders.Scene;


import java.awt.*;
import java.util.*;
import java.io.*;
import java.awt.event.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.image.*;
import com.sun.j3d.utils.geometry.*;

public class BrowserMocap
        extends Browser3D
{

        private Switch swtch = null;
        private int currentSwitch = 0;
        private LinearFog fog = null;
        private Background background = null;
        public Servidor servidor;
        Appearance app;

/**************************************************************************/
/*Inicializa la aplicación y crea la conexión con el dispositivo de captura a través del
servidor.java*/

public Group buildScene( )
        {
                Color3f color = (Color3f)colors[currentColor].value;
                float front   = ((Float)fronts[currentFront].value).floatValue( );
                float back    = ((Float)backs[currentBack].value).floatValue( );

                setHeadlightEnable( false );
                setNavigationType( Walk );
                Group sceneMain = new Group( );
               Transform3D yAxis = new Transform3D();
               yAxis.setScale(new Vector3d(new Vector3f(0.02f,0.02f,0.02f)));

               TransformGroup scene = new TransformGroup(yAxis);
               sceneMain.addChild(scene);


               // Add a light
               BoundingSphere worldBounds = new BoundingSphere(
                       new Point3d( 0.0, 0.0, 0.0 ),  // Center
                       10000.0 ),                     // Extent
```

```
                DirectionalLight light = new DirectionalLight( );
                light.setEnable( true );
                light.setColor( new Color3f( 1.0f, 1.0f, 1.0f ) );
                light.setDirection( new Vector3f( 0.5f, -1.0f, -0.5f ) );
                light.setInfluencingBounds( worldBounds );
                sceneMain.addChild( light );


                /*Apariencia del material*/
                app = new Appearance( );
                Material mat = new Material();
                mat.setAmbientColor( 0.2f, 0.2f, 0.2f );
                mat.setDiffuseColor( 0.0f, 0.4f, 8.0f );
                mat.setSpecularColor( 0.7f, 0.7f, 0.7f );

                app.setMaterial( mat );

                /*Crea un objeto Servidor para la conexión con el dispositivo de captura*/
                Servidor= new Servidor(-10002);

                scene.addChild( brazoD(new Vector3f(0f,0f,0f), Servidor) );
                return sceneMain;
}


/*************************************************************************/
/*Construye la representación gráfica de un brazo utilizando una serie de eslabones
asociados en forma gerárquica*/

public Eslabon brazoD(Vector3f puntoInsercion ,Servidor Servidor)
{
        /*Genera palma de la mano*/
        Eslabon mano1= new Eslabon(8f,3f,new Vector3f(4f,2f,4f),2,new
Vector3f(1f,0f,0.0f),3,app,".\\obj_bones\\palma.obj");
        servidor.addObject("Ar/Ab",mano1,4,-25,3.3f/255f,-0.8f);

        Eslabon mano2= new Eslabon(.1f,.2f,new Vector3f(1f,1f,1f),2,new
Vector3f(20f,0f,0.0f),2,app,null);
        servidor.addObject("De/Iz",mano2,41,-36, -2.9f/255f,1f);
        mano2.transRot.addChild(mano1);

        /*Brazo*/
        Eslabon brazoA= new Eslabon(20f,.2f,new Vector3f(10f,1f,1f),2,new Vector3f(0f,-
1f,0f),1,app,".\\obj_bones\\h2d1.obj");
        servidor.addObject("Roll",brazoA,25,-100,-5f/255f,0);
        brazoA.transRot.addChild(mano2);

        Eslabon brazoB= new Eslabon(20f,.2f,new Vector3f(10f,2.8f,2.8f),2,new
Vector3f(0f,-0f,0f),1,app,".\\obj_bones\\h2d1.obj");
        servidor.addObject(brazoB,25,-100,-12f/255f,0);

        /*Codo*/
        Eslabon codo= new Eslabon(0.1f,.2f,new Vector3f(10f,2.8f,2.8f),2,new
Vector3f(20f,-0f,0f),2,app,null);
        servidor.addObject("Codo",codo,17,-31, 2.5f/255f,0);
        codo.transRot.addChild(brazoA);
        codo.transRot.addChild(brazoB);

        /*Muñeca*/
        Eslabon anteBrazo= new Eslabon(20f,.2f,new Vector3f(10f,2.8f,2.8f),1,new
Vector3f(0f,-0f,0f),1,app,".\\obj_bones\\h2d1.obj");
        servidor.addObject(anteBrazo,4,-100,5f/255f,0);
        anteBrazo.transRot.addChild(codo);

        Eslabon hombro1= new Eslabon(0.1f,.2f,new Vector3f(1f,1f,1f),2,new Vector3f(0f,-
0f,0f),2,app,null);
        servidor.addObject(hombro1,41,0,-5f/255f,0);
        hombro1.transRot.addChild(anteBrazo);
```

```
        Eslabon hombro2= new Eslabon(0.1f,.2f,new
Vector3f(1f,1f,1f),2,puntoInsercion,3,app,null);
        servidor.addObject(hombro2,25,170,4f/255f,0);
        hombro2.transRot.addChild(hombro1);

        /*Dedo indice*/
        Eslabon e1= new Eslabon(1.4f,0.6f,new Vector3f(.85f,.85f,.85f),6,new
Vector3f(3.0f,0f,0.0f),3,app,".\\obj_bones\\h3d1.obj");
        servidor.addObject("IND1",e1,60,-30,4.3f/255f,0);
        Eslabon e2= new Eslabon(3.4f,0.65f,new Vector3f(1.6f,1.1f,1.1f),6,new
Vector3f(4.5f,0f,0.0f),3,app,".\\obj_bones\\h2d1.obj");
        servidor.addObject("IND2",e2,60,-30,4.3f/255f,0);
        Eslabon e3= new Eslabon(4.8f,0.7f,new Vector3f(2.6f,2f,2f),6,new
Vector3f(0f,0f,0f),3,app,".\\obj_bones\\h1d1.obj");
        servidor.addObject("IND3",e3,12,-58,3.0f/255f,0);
        Eslabon e4= new Eslabon(0.1f,0.1f,new Vector3f(1f,1f,1f),2,new
Vector3f(7f,0f,2.5f),2,app,null);
        servidor.addObject("IND4",e4,36,-48,-2f/255f,0);
        (mano1.transRot).addChild(e4);
        (e4.transRot).addChild(e3);
        (e3.transRot).addChild(e2);
        (e2.transRot).addChild(e1);

        /*Dedo medio*/
        e1= new Eslabon(1.6f,0.6f,new Vector3f(.9f,.9f,.9f),6,new
Vector3f(3.0f,0.0f,0.0f),3,app,".\\obj_bones\\h3d1.obj");
        servidor.addObject("MED1",e1,52,-30,2.3f/255f,0);
        e2= new Eslabon(3.6f,0.65f,new Vector3f(1.8f,1.1f,1.1f),6,new
Vector3f(4.5f,0f,0.0f),3,app,".\\obj_bones\\h2d1.obj");
        servidor.addObject("MED2",e2,52,-30,2.3f/255f,0);
        e3= new Eslabon(5f,0.7f,new Vector3f(2.8f,2f,2f),6,new
Vector3f(8f,0f,1.0f),3,app,".\\obj_bones\\h1d1.obj");
        servidor.addObject("MED3",e3,57,-100,2.0f/255f,0);
        (mano1.transRot).addChild(e3);
        (e3.transRot).addChild(e2);
        (e2.transRot).addChild(e1);

        /*Dedo anular*/
        e1= new Eslabon(1.4f,0.6f,new Vector3f(.85f,.85f,.85f),6,new
Vector3f(3.0f,0.0f,0.0f),3,app,".\\obj_bones\\h3d1.obj");
        servidor.addObject("ANUL1",e1,9,-30,2.3f/255f,0);
        e2= new Eslabon(3.4f,0.65f,new Vector3f(1.6f,1.1f,1.1f),6,new
Vector3f(4.5f,0f,0.0f),3,app,".\\obj_bones\\h2d1.obj");
        servidor.addObject("IND2",e2,60,-30,4.3f/255f,0);
        servidor.addObject("ANUL2",e2,9,-30,2.3f/255f,0);
        e3= new Eslabon(4.8f,0.7f,new Vector3f(2.6f,2f,2f),6,new Vector3f(7.5f,0f,-
0.5f),3,app,".\\obj_bones\\h1d1.obj");
        servidor.addObject("ANUL3",e3,33,-45,4.5f/255f,0);
        (mano1.transRot).addChild(e3);
        (e3.transRot).addChild(e2);
        (e2.transRot).addChild(e1);

        /*Dedo meñique*/
        e1= new Eslabon(1f,0.6f,new Vector3f(.8f,.8f,.8f),6,new
Vector3f(3.0f,0.0f,0.0f),3,app,".\\obj_bones\\h3d1.obj");
        servidor.addObject("MEÑ1",e1,49,-30,2.3f/255f,0);
        e2= new Eslabon(3f,0.65f,new Vector3f(1.4f,1.1f,1.1f),6,new
Vector3f(4.5f,0f,0.0f),3,app,".\\obj_bones\\h2d1.obj");
        servidor.addObject("MEÑ2",e2,49,-30,2.3f/255f,0);
        e3= new Eslabon(4.4f,0.7f,new Vector3f(2.4f,2f,2f),6,new Vector3f(6f,0f,-
2f),3,app,".\\obj_bones\\h1d1.obj");
        servidor.addObject("MEÑ3",e3,34,-58,3f/255f,0);
        (mano1.transRot).addChild(e3);
        (e3.transRot).addChild(e2);
        (e2.transRot).addChild(e1);

         /*Pulgar*/
        e1= new Eslabon(2f,1f,new Vector3f(.9f,.9f,.9f),6,new
Vector3f(2.5f,0.0f,0.0f),3,app,".\\obj_bones\\h3d1.obj");
        servidor.addObject("PULG1",e1,44,-62,2.5f/255f,0);
```

```
        e2= new Eslabon(3f,1f,new Vector3f(1.3f,1f,1f),6,new
Vector3f(3f,0f,0.0f),3,app,".\\obj_bones\\h2d1.obj");
        servidor.addObject("PULG2",e2,44,-62,1.3f/255f,0);
        e3= new Eslabon(3f,1f,new Vector3f(1.5f,1.5f,1.5f),6,new
Vector3f(0.0f,0f,0f),3,app,".\\obj_bones\\h1d1.obj");
        servidor.addObject("PULG3",e3,20,-68,10.5f/255f,0);
        e4= new Eslabon(0.1f,1f,new Vector3f(0.2f,2f,2f),6,new
Vector3f(4.0f,0.0f,2f),2,app,".\\obj_bones\\h1d1.obj");
        servidor.addObject("PULG4",e4,28,-68,2f/255f,-1.8f);
        (mano1.transRot).addChild(e4);
        (e4.transRot).addChild(e3);
        (e3.transRot).addChild(e2);
                (e2.transRot).addChild(e1);

        return hombro2;
        }
        }
```

Apéndice D

Código fuente del servidor de captura (C)

El código fuente original de este programa es propiedad de Alias Wavefront, se le han hecho modificaciones menores para cumplir con los requerimientos del sistema de captura. Este programa corre bajo el sistema operativo (IRIX o Windows 9x/2000/NT).Inicializa y mantiene la comunicación con el hardware de captura a través del puerto serial. La información que recibe es transferida a otras aplicaciones (por ejemplo, Maya) a través de sockets.

servidorMocap.c

```
/*
 * ****************************************************************************
 * Copyright (C) 1997 Alias|Wavefront Inc.
 *
 * These coded instructions, statements and computer programs contain
 * unpublished information proprietary to Alias|Wavefront Inc. and are
 * protected by Canadian and US federal copyright laws. They may not be
 * disclosed to third parties or copied or duplicated, in whole or in part,
 * without prior written consent of Alias|Wavefront Inc.
 *
 * Unpublished-rights reserved under the Copyright Laws of the United States.
 * ****************************************************************************
 */

#include <stdio.h>
#ifndef _WIN32
#include <unistd.h>
#endif //WIN32
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <time.h>
#include <math.h>
#include <sys/types.h>
#ifndef _WIN32
#include <sys/time.h>
#include <sys/socket.h>
#include <getopt.h>
#else
#include <windows.h>
#include <io.h>
#include <winsock2.h>
#endif //wIN32
#include "/AW/Maya2.5/include/maya/mocapserial.h"

#ifdef _WIN32
char *   optarg = NULL;
int        optind = 1;
int        opterr = 0;
int     optlast = 0;
#define GETOPTHUH 0
#else
#define closesocket close
#endif

/*
 * Missing prototypes
 */
#ifndef _WIN32
extern void bzero(void *b, int length);
extern int select (int nfds, fd_set *readfds, fd_set *writefds,
```

```
                fd_set *exceptfds, struct timeval *timeout);
#include <maya/mocapserver.h>
#include <maya/mocapserial.h>
#else

#define bzero  ZeroMemory
#include <mocapserver.h>
#include <mocapserial.h>
#endif

#ifndef lengthof
#define lengthof(array)       (sizeof(array) / sizeof(array[0]))
#endif /* lengthof */

static int daemon_mode = 0;
static int inetd_mode  = 0;
static int show_usage  = 0;
static int verbose     = 0;
static int debug_mode  = 0;

static char program[80];
static char *server_name = program;
static CapChannel      canales[64];
static int handle_client(int client_fd);
static void get_data(int client_fd);
static int create_channels(int client_fd);
float angulo[64],anguloAnterior[64];
int fd; /* File descriptor for the serial port */


/*****************************************************************************/

int main(int argc, char **argv)
{
  int opt;
  char *cptr;

  int status;
  int client_fd = -1;

  /* Grab a copy of the program name   */
#ifdef _WIN32
  _splitpath (argv[0], NULL, NULL, program, NULL);
#else
  cptr = strrchr(argv[0], '/');
  if (cptr)
  {
    strcpy(program, (cptr + 1));
  }
  else
  {
    strcpy(program, argv[0]);
  }
#endif //WIN32

  /* Parse the options   */
#ifdef DEVEL
  while ((opt = getopt(argc, argv, "hdvDin:")) != -1)
#else /* DEVEL */
  while ((opt = getopt(argc, argv, "hdn:")) != -1)
#endif /* DEVEL */
  {
    switch (opt)
    {
      case 'h':
        show_usage++;
        break;

      case 'd':
        daemon_mode++;
```

```
        break;

      case 'D':
        debug_mode++;
        break;

      case 'i':
        inetd_mode++;
        break;

      case 'n':
        server_name = optarg;
        break;

      case 'v':
        verbose++;
        break;

      case GETOPTHUH:
        show_usage++;
    }
  }

  /* Check for errors  */
  if (daemon_mode && inetd_mode) show_usage++;
  if (optind < argc) show_usage++;

  if (show_usage)
  {
    fprintf(stderr, "Usage:\n");
#ifdef DEVEL
    fprintf(stderr, "    %s [-hdvDi] [-n name]\n", program);
#else /* DEVEL */
    fprintf(stderr, "    %s [-hd] [-n name]\n", program);
#endif /* DEVEL */
    fprintf(stderr, "\n");
    fprintf(stderr, "        -h        Print this help message\n");
    fprintf(stderr, "        -d        Run as a daemon in the background\n");
#ifdef DEVEL
    fprintf(stderr, "        -i        Run from inetd\n");
    fprintf(stderr, "        -D        Set the debug flag\n");
    fprintf(stderr, "        -v        Set the vebose flag\n");
#endif /* DEVEL */
    fprintf(stderr, "        -n name   Set the UNIX socket name to name\n");
    fprintf(stderr, "\n");
    fprintf(stderr, "Defaults:\n");
    fprintf(stderr, "    %s -n %s\n", program, program);
    fprintf(stderr, "\n");

    exit(1);
  }

  if (inetd_mode)
  {    /* The socket to the client is open on file descriptor 0.
       * Handle one client and then exit.
       */
    client_fd = 0;
  }

  if (daemon_mode)
  {    /* Convert this process into a standard unix type daemon process.
       * It will be running in the background with no controlling terminal
       * and using syslog to report error messages.
       */
    status = CapDaemonize();
    if (status < 0)
    {
      CapError(-1, CAP_SEV_FATAL, program, NULL);
      exit(1);
    }
  }
```

```
  if (inetd_mode)
  {
    /* The socket to the client is already open on file descriptor 0.
     * Handle one client and then exit.
     */
    status = handle_client(client_fd);
    exit(status);
  }
  else
  {

    /*abre puerto serial*/
    open_port();

    /*loop de recepcion de clientes*/
    while (1)
    {
      /*Set up the server socket and wait for a connection.
       */
      client_fd = CapServe(server_name);
      if (client_fd < 0)
      {
              CapError(-1, CAP_SEV_FATAL, program, NULL);
              exit(1);
      }

      /* Handle client requests */
      status = handle_client(client_fd);

      if (status < 0)
      {
              CapError(-1, CAP_SEV_FATAL, program, NULL);
      }

      /* Shutdown the client */
      closesocket(client_fd);
      client_fd = -1;

      /* Go back and wait for another connection request */
      continue;
    }
  }
  return 0;
}


/*****************************************************************************/
/*Procesa la cadena dada como lista de argumentos en la línea de comandos*/
int getopt(int argc, char **argv, char *pargs)
{
        if (optind >= argc) return EOF;

        if (optarg==NULL || optlast==':')
        {
                optarg = argv[optind];
                if (*optarg!='-' && *optarg!='/')
                        return EOF;
        }

        if (*optarg=='-' || *optarg=='/') optarg++;
        pargs = strchr(pargs, *optarg);
        if (*optarg) optarg++;
        if (*optarg=='\0')
        {
                optind++;
                optarg = NULL;
        }
        if (pargs == NULL) return 0;   /*error
        if (*(pargs+1)==':')
```

```
        {
                if (optarg==NULL)
                {
                        if (optind >= argc) return EOF;
                        // we want a second paramter
                        optarg = argv[optind];
                }
                optind++;
        }
        optlast = *(pargs+1);

        return *pargs;
}


/*************************************************************************/
/*Inicializa el puerto serial*/
 int open_port(void)
 {

   struct termios save_attr;

   fd = CapSerialOpen("COM1",9600,8,CAP_PARITY_NONE,1,CAP_SERIAL_BINARY,NULL);
   if (fd == -1)
   {
        /* Could not open the port. */
        fprintf(stderr, "open_port: Unable to open COM1 - %s\n",
        strerror(errno));
   }
   else
   {
                fprintf(stderr, "Com1 abierto\n");
   }
   return (fd);
 }


/*************************************************************************/
/*Recibe datos del dispositivo de captura, se verifica periódicamente en un ciclo el
estado del puerto, no se libera el control hasta no recibir un paquete completo (64
enteros)*/

 int recivePaqueteSerial(void)
 {
   char readBuffer[1];
   int n; /*es cuantos se recibieron del serial?*/
   int dato, cabecera=0;
   float ang;

   while(cabecera!=66) /*hasta recibir un paquete completo*/
   {
        n=CapSerialRead(fd, readBuffer,1); /*Lee un caracter del buffer buffer*/
        if(n) /*Si hay leyó algo del puerto entonces actualiza*/
        {
                if ( cabecera == 0) /*Checando paquete*/
                {
                        if ( (int)readBuffer[0] == 6 )
                                cabecera=1;                     /*Primer caracter especial de
la cabecera*/
                        else    fprintf(stderr, "Sincronizando dispositivo...\n" ),
        /*Sincronizando*/
                }
                else
                        if ( cabecera == 1)
                                if ( (int)readBuffer[0] == 6 )
                                {
                                        cabecera = 2;  /*Segundo caracter especial  de la
        cabecera*/
                                        fprintf(stderr, "Dispositivo sincronizado.\n");
                                }
```

```
                                else            /*Pérdida de sincronizacion, vuelve a
empezar*/
                                {
                                        fprintf(stderr, "Se perdio sincronizacion ...\n" );
                                cabecera = 0;
                                }
                        else
                        {       /*Para cada uno de los datos del cuerpo del paquete */
                                dato=(int) readBuffer[0];
                                if(dato<0) dato=255+dato;
                                angulo[cabecera-2]= (float) 360.0*dato/255.0;
                                ang=(angulo[cabecera-2] + anguloAnterior[cabecera-2]) / 2.0;
                                CapSetData(canales[cabecera-2], &ang );
                                anguloAnterior[cabecera-2] = angulo[cabecera-2]; /*Actualiza
valor */
                                cabecera++;
                        }
        }
    }

    return 1;
 }


/*************************************************************************/
/*Crea los canales de comunicacion entre el servidor y el cliente*/

static int create_channels(int client_fd)
{
  int i;
  char nombreCanal[5];
  for(i=0;i<64;i++)
        {
          itoa(i,nombreCanal,10);
          canales[i]= CapCreateChannel(nombreCanal,CAP_USAGE_ZROT, 1);
        }

  return 0;
}


/*************************************************************************/
/*Manejador del cliente, mantiene comunicacion e interpreta comandos*/

static int handle_client(int client_fd)
{
  int status, size;
  CapCommand cmd;
  char ruser[64], rhost[64], realhost[64];
  float rate, time;
  fd_set rd_fds;
  struct timeval base_timeout, timeout;
  int recording = 0;

  static int channels_created = 0;

  base_timeout.tv_sec  = 1;
  base_timeout.tv_usec = 0;

  /*loop del cliente*/ /*buscar donde se va a poner el chequeo de lo que llega del chunce
de captura*/
  while (1)
  {
    FD_ZERO(&rd_fds);
    FD_SET(client_fd, &rd_fds);
    timeout.tv_sec  = base_timeout.tv_sec;
    timeout.tv_usec = base_timeout.tv_usec;

        status = select(FD_SETSIZE, &rd_fds, NULL, NULL, &timeout); //temporizador??
    if (status  0)
```

```
    {
#ifndef _WIN32
      if (errno == EINTR)
      {
       /* Ignore signals and try again */
       continue;
      }
#endif /*WIN32*/

      /* Otherwise, give a fatal error message */
      CapError(client_fd, CAP_SEV_FATAL, program, "select failed");
      CapError(client_fd, CAP_SEV_FATAL, "select", NULL);
      exit(1);
    }
    else if (status == 0)
    {
      /* We got a timeout */
      if (recording) get_data(client_fd); //recivePaqueteSerial(void)?? o fuera para que
no vaya de acuerdo al temporizador...

      /* Try again */
      continue;
    }
    else
    {
      /* There is data on the client file descriptor */
      cmd = CapGetCommand(client_fd);
      switch (cmd)
      {
       case CAP_CMD_QUIT:
         return 0;

#ifdef _WIN32
       case CAP_CMD_ERROR:
              return -1;
#endif

       case CAP_CMD_AUTHORIZE:
         status = CapGetAuthInfo(client_fd, ruser, rhost, realhost);
         if (status < 0)
         {
           return -1;
         }

         /*If user@host is not authorized to use this server then:
          *
          * status = CapAuthorize(client_fd, 0);
          */
         status = CapAuthorize(client_fd, 1);
         break;

       case CAP_CMD_INIT:     /* Initial client/server handshake */
         status = CapInitialize(client_fd, program);
         break;

       case CAP_CMD_VERSION:  /* Send version information */
         status = CapVersion(client_fd, program, "1.0",
                             "Clock capture server - v1.0");
         break;

       case CAP_CMD_INFO:
         if (!channels_created)
         {
           /* Only create the channel data once */
           status = create_channels(client_fd);
           if (status < 0)
           {
             break;
           }
           channels_created = 1;
         }
```

```
#ifdef SERVER_SIDE_RECORDING
        /* The server side recording returns incorrect time in
         * the current version
         */
          status = CapInfo(client_fd, 1.0, 100.0, 10.0, 512 * 1024, 1);
#else
          status = CapInfo(client_fd, 0.0, 0.0, 0.0, 512 * 1024, 1);
#endif

          break;

        case CAP_CMD_DATA.     /* Send frame data */
          get_data(client_fd);
          status = CapData(client_fd);
          break;

#ifdef SERVER_SIDE_RECORDING
        /* The server side recording returns incorrect time in
         * the current version
         */
        case CAP_CMD_START_RECORD:     /* Start recording */
          rate = CapGetRequestedRecordRate(client_fd);
          size = CapGetRequestedRecordSize(client_fd);

          /* Convert rate in Hz to a timeout value in a timeval structure */
          if (rate <   1.0) rate =   1.0;
          if (rate > 100.0) rate = 100.0;

          time = 1.0 / rate;
          base_timeout.tv_sec  = time;
          base_timeout.tv_usec = (time - base_timeout.tv_sec) * 1000000;

          status = CapStartRecord(client_fd, rate, size);
          if (status != -1)
          {
            recording = 1;
          }
          break;

        case CAP_CMD_STOP_RECORD:              /* Stop recording */
          status = CapStopRecord(client_fd);
          recording = 0;
          base_timeout.tv_sec  = 1;
          base_timeout.tv_usec = 0;
          break;
#endif /* server side recording */

        default:                          /* Ignore unknown commands */
          status = CapError(client_fd, CAP_SEV_ERROR, program,
                            "Unknown server command.");
          break;
      }

      if (status < 0)
      {
        return -1;
      }
    }
  }

  /* return 0; */
}


/*************************************************************************/
/*Envia la informacion actualizados al cliente*/

#ifndef M_PI
#define M_PI 3.14159265358979323846
```

```
#endif

#define ANGLE(t, a) (((t) % (a)) * 2 * M_PI / (a))
static void get_data(int client_fd)
{
  time_t now, midnight,
  struct tm *local;
  long elapsed;
  float angle;
  int i;

  now = time(0);
  local = localtime(&now);
  local->tm_hour = 0;
  local->tm_min  = 0;
  local->tm_sec  = 0;
  midnight = mktime(local);
  elapsed = now - midnight,

  angle = -ANGLE(elapsed, 1);

  for(i=0;i<64;i++)
        CapSetData(canales[i], &angle);

recivePaqueteSerial();

}
```

Apéndice E

Generación de entidades gráficas en Maya

Las gráficas que a continuación se muestran ilustran el proceso descrito en el capitulo 3 referente a la generación de animaciones con el software Maya:

1. **Generación de esqueletos**. Se definen los eslabones (partes móviles) que formarán a la entidad que se desea dotar de movimiento.

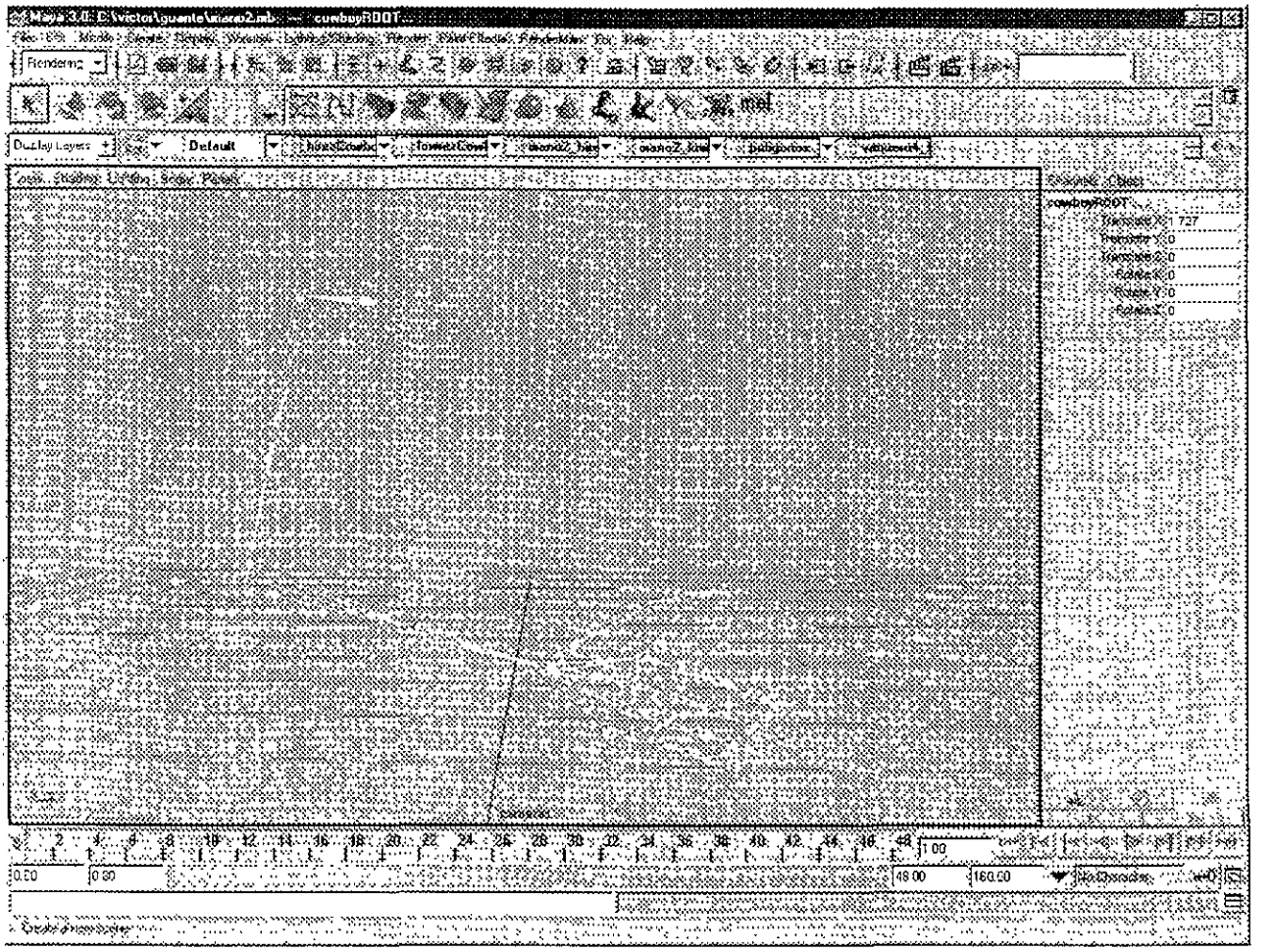

Figura E.1. Esqueleto donde se definen las partes móviles del brazo derecho.

2. **Modelado de entidades graficas.** Es la definición de la geometría que dará volumen a la entidad gráfica y sobre la cual se aplicará la apariencia de un material (color, texturas,etc.).

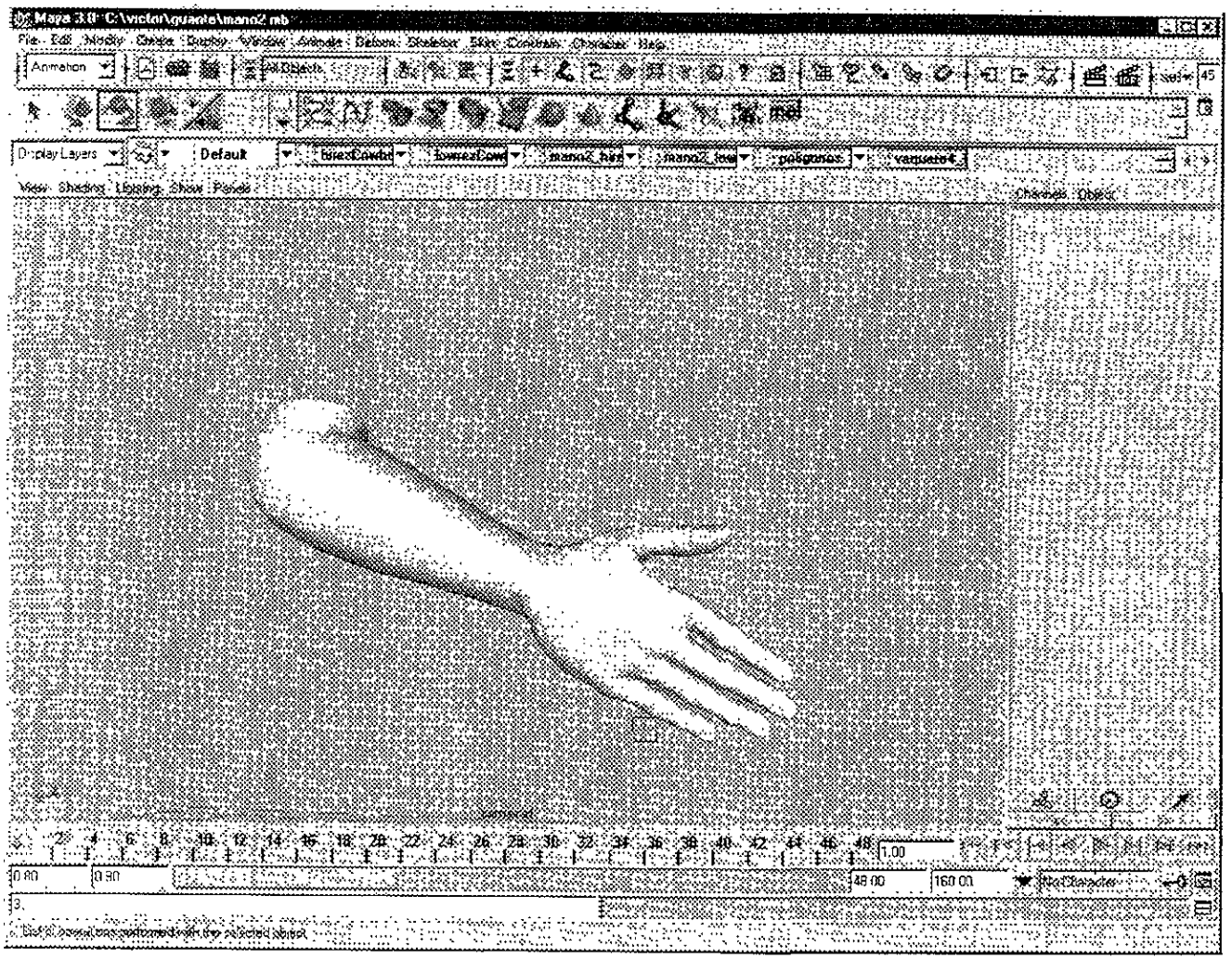

Figura E.2. Geometría, volumen del brazo .

3. **Asignación de elementos del esqueleto con geometría.** Cada uno de los eslabones que forman el esqueleto de nuestra entidad gráfica es asociado con la geometría. De esta manera al dotar de movimiento a los eslabones, estos arrastrarán consigo la geometría asociada a ellos.

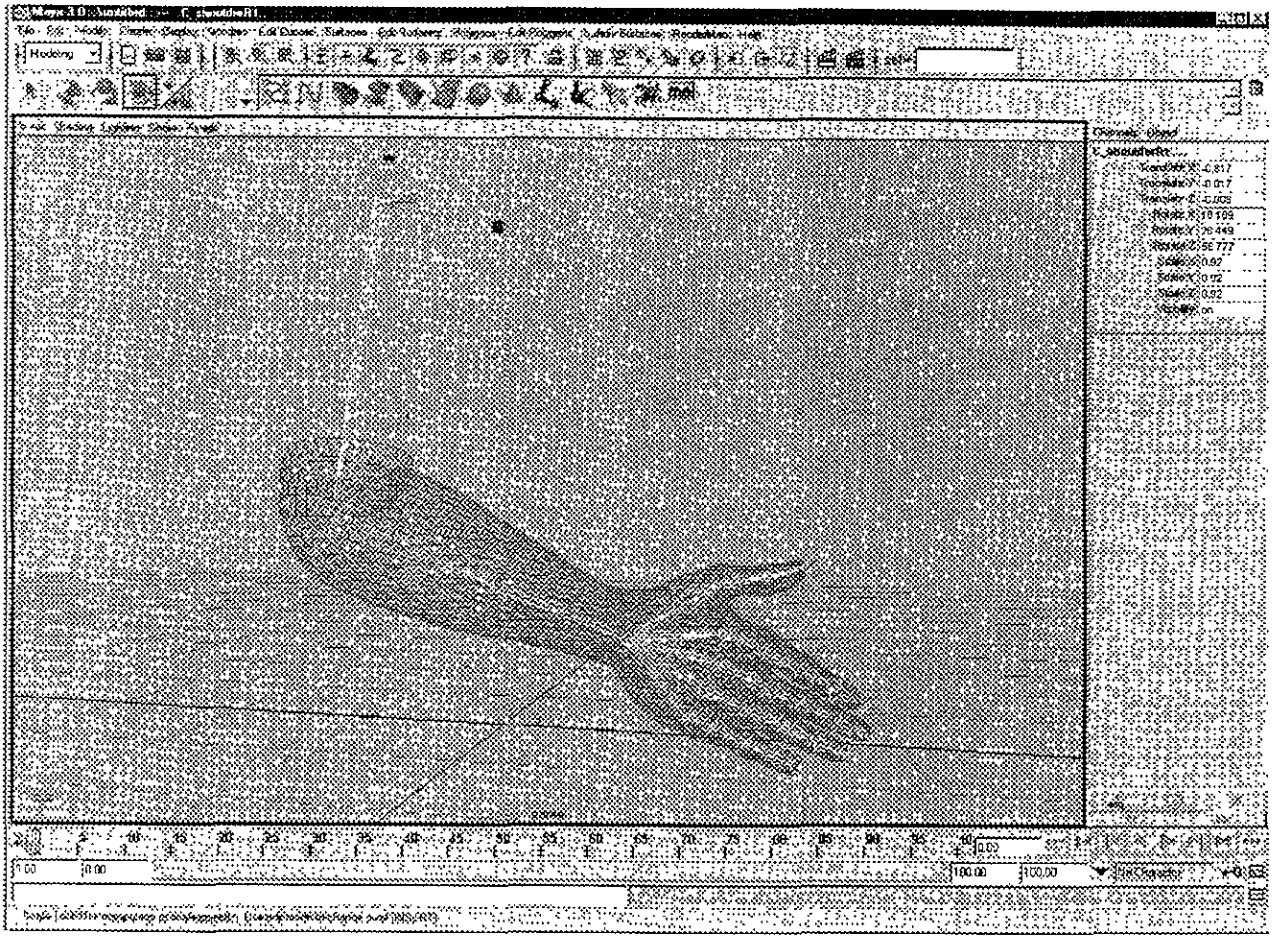

Figura E.3. Asociación de la geometría del brazo con el esqueleto.

4. **Asignación de apariencia.** En este paso se define la apariencia que tendrá la entidad gráfica (color, textura, etc.) y se aplica sobre la geometría a manera de una envoltura.

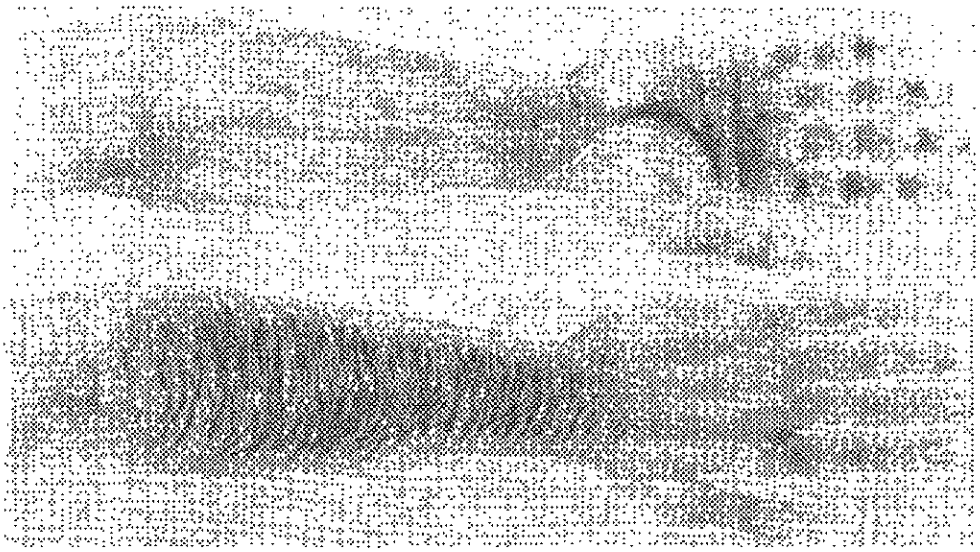

Figura E.4. Textura plana a mapear en la geometría del brazo.

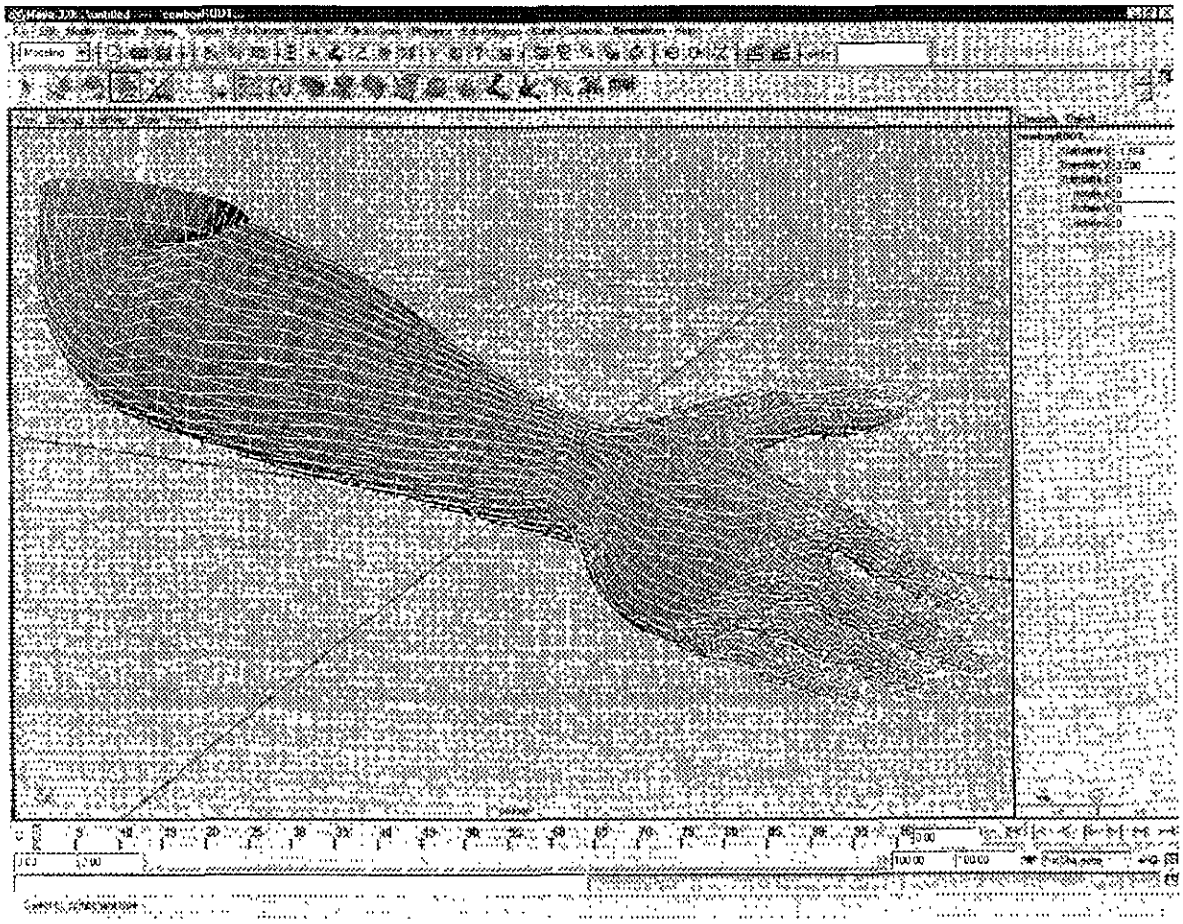

Figura E.5. Textura mapeada sobre la geometría del brazo.

5. **Definición de comportamiento.** Es posible que se desee dotar de algún comportamiento en especial para las partes móviles de la entidad gráfica. Este comportamiento puede estar asociado a determinados eventos y por lo regular se utiliza para dotar de un mayor realismo y/o automatizar algunas animaciones. Por ejemplo, uno de los usos más comunes, es la simulación del movimiento muscular bajo la piel de los animales, esto se logra asociando flexores al ángulo de rotación de las articulaciones.

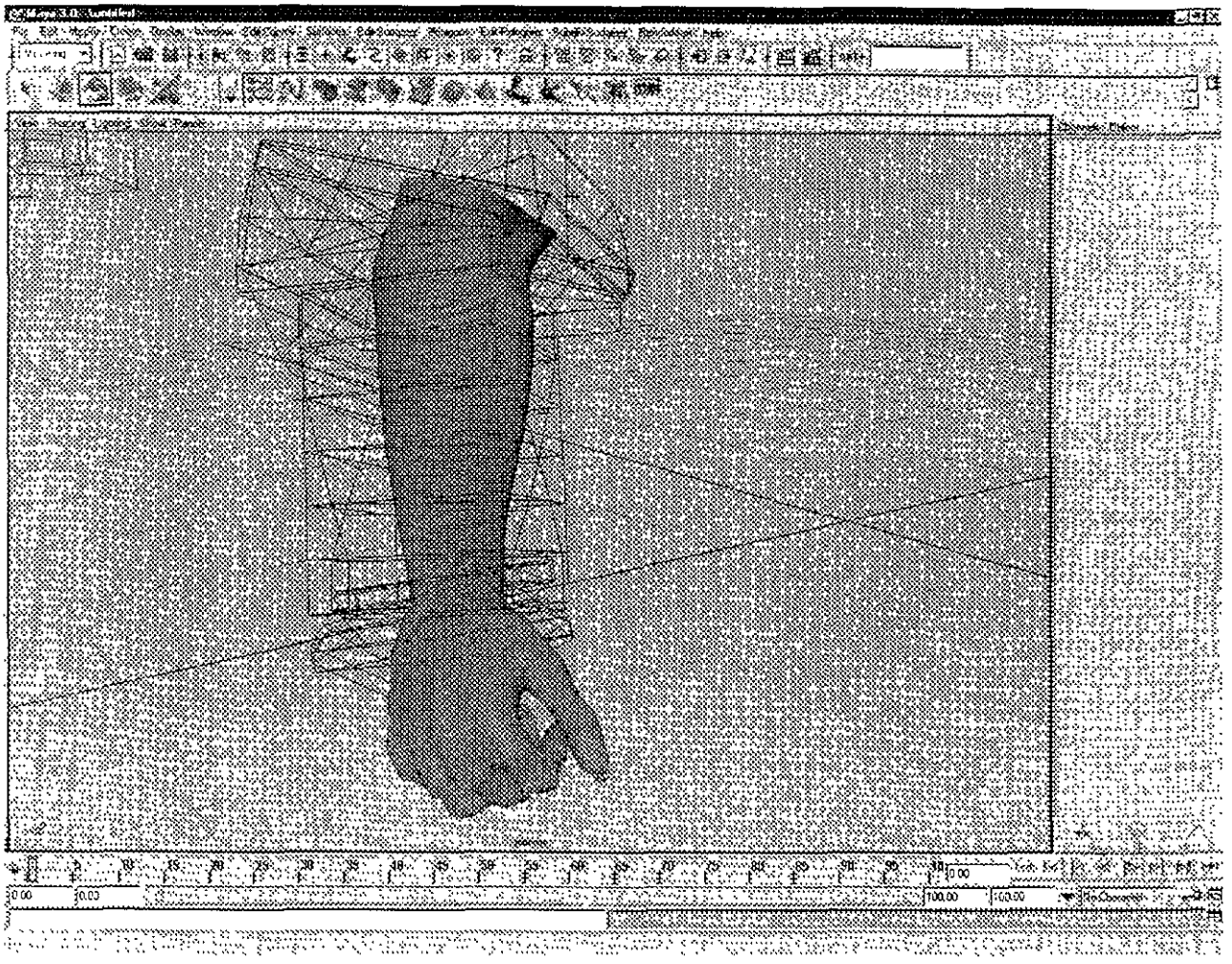

Figura E.6. Asignación de flexores sobre la geometría del brazo.

Apéndice F

Programa para la conexión con el servidor de captura y el mapeo de entidades en Maya (MEL)

Este programa corre bajo el ambiente de Maya. Inicializa y mantiene la comunicación con el servidor de captura (servidorMocap.c), y mapea los datos provenientes cada uno de los canales de captura a los eslabones del esqueleto de la entidad gráfica.

brazo.mel

```
/*Inicializa las estructuras de datos para la entidad gráfica */

global proc string mayaMano()
{
        string $dName="mano";  /*Entidad que representa al dispositivo
                                 de captura*/
        string $sName="cl";    /*Nombre del proceso servidor de datos*/

        /*Inicializa el servidor de datos*/
        if ( 0 != server($dName, $sName) ) {
                error("Couldn't launch " + $sName );
                return "";
        }

        /*Arreglo de transformaciones a través de las cuales se aplicarán
        las rotaciones correspondientes a los movimientos capturados*/

        string $hands[17] = {"fore4R", "fore3R", "fore2R",
                                "mid4R", "mid3R", "mid2R",
                                "ring4R", "ring3R", "ring2R",
                                "pinky4R", "pinky3R", "pinky2R",
                                "thumb3R", "thumbR2", "thumb1R",
                                "wristR",                          /*Muñeca*/
                                "C_elbowR"};                              /*Codo*/


        /*Mapea los canales de captura*/
        makeHands($dName, $hands);

        return $sName;
}


/*************************************************************************/

/*Inicializa la conexión con el servidor de datos*/
proc int server(string $dName, string $sName)
{
        //defineDataServer -d $dName -s $sName;
        return initDataServer($dName,$sName,"localhost", "", "",2);
}


/*************************************************************************/

proc makeHands( string $dName, string $hands[] )
{

        attachHands($dName, $hands);
}
```

```
/*****************************************************************************/

/*Asocia los eslabones del esqueleto con cada uno de los canales de datos provenientes del
hardware de captura*/
proc attachHands(string $device, string $hands[] )
{
        attach1D($device, "60", 0.005,  -0.3, $hands[0], "rz");
        attach1D($device, "60", 0.005,  -0.3, $hands[1], "rz");
        attach1D($device, "12", 0.005,  -0.2, $hands[2], "rz");
        attach1D($device, "36", 0.005,  -0.4, $hands[2], "ry");

        attach1D($device, "52", 0.005,  -0.7, $hands[3], "rz");
        attach1D($device, "52", 0.005,  -0.6, $hands[4], "rz");
        attach1D($device, "57", 0.005,  -0.3, $hands[5], "rz");

        attach1D($device, "9",  0.005,  -0.4, $hands[6], "rz");
        attach1D($device, "9",  0.005,  -0.5, $hands[7], "rz");
        attach1D($device, "33", 0.005,  -0.3, $hands[8], "rz");

        attach1D($device, "49", 0.005,  -0.5, $hands[9], "rz");
        attach1D($device, "49", 0.005,  -0.6, $hands[10], "rz");
        attach1D($device, "34", 0.005,  -0.3, $hands[11], "rz");

        attach1D($device, "44", 0.007,  -1.0, $hands[12], "rx");
        attach1D($device, "44", 0.003,   0.0, $hands[13], "rx");
        attach1D($device, "20", 0.050,  -3.5, $hands[14], "rx");
        attach1D($device, "28",-0.0025, -0.3, $hands[14], "ry");

        attach1D($device, "4",  0.005,  -0.5, $hands[15], "rz");
        attach1D($device, "41", 0.005,  -0.5, $hands[15], "ry");
        attach1D($device, "17", 0.005,   0.0, $hands[16], "ry");
}


/*****************************************************************************/
/*Mapea uno de los canales del dispositivo de captura a los elementos móviles del
esqueleto */

proc attach1D( string $device, string $channel, float $scale, float $offset,
                              string $target, string $attribute)
{
        string $axisName;
        $axisName = $channel;
        string $atrName = $target+"."+$attribute;

        attachDeviceAttr -d $device -ax $axisName $atrName;
        setAttrMapping -d $device -ax $axisName -at $attribute -s $scale -o $offset -a
$target ;
}


/***********************************************************************/
```

Apéndice G

Programa para definir el comportamiento de los botones para la emulación del teclado en Maya (MEL)

Este programa corre bajo el ambiente de Maya. En él se define el comportamiento de los botones de un teclado virtual, es decir, se especifican los eventos que serán activados para determinadas acciones del usuario.

teclado.mel

```
float $botonesx0[4];
float $botonesz0[4];
float $botonesx1[4];
float $botonesz1[4];
float $botonesy1[4];
int $cual;
float $dedox[3];
float $dedoy[3];
float $dedoz[3];
float $minimo;

matrix $b[4][3];
int $cuantos[4];

int $i, $j, $ind;

$botonesx0[0]= boton1.translateX - 1;
$botonesx0[1]= boton2.translateX - 1;
$botonesx0[2]= boton3.translateX - 1;
$botonesx0[3]= boton4.translateX -1;

$botonesz0[0]= boton1.translateZ - 1;
$botonesz0[1]= boton2.translateZ - 1;
$botonesz0[2]= boton3.translateZ - 1;
$botonesz0[3]= boton4.translateZ - 1;

$botonesx1[0]= boton1.translateX + 1;
$botonesx1[1]= boton2.translateX + 1;
$botonesx1[2]= boton3.translateX + 1;
$botonesx1[3]= boton4.translateX + 1;

$botonesz1[0]= boton1.translateZ + 1;
$botonesz1[1]= boton2.translateZ + 1;
$botonesz1[2]= boton3.translateZ + 1;
$botonesz1[3]= boton4 translateZ + 1;


$botonesy1[0]=2;
$botonesy1[1]=2;
$botonesy1[2]=2;
$botonesy1[3]=2;


$dedox[0]=dedo1.translateX;
$dedox[1]=dedo2.translateX;
$dedox[2]=dedo3.translateX;

$dedoy[0]=dedo1.translateY;
$dedoy[1]=dedo2.translateY;
$dedoy[2]=dedo3.translateY;

$dedoz[0]=dedo1.translateZ;
```

```
$dedoz[1]=dedo2.translateZ;
$dedoz[2]=dedo3.translateZ;

$cuantos[0]=0;
$cuantos[1]=0;
$cuantos[2]=0;
$cuantos[3]=0;


for ( $i=0; $i<3; $i++ )
   for ( $j=0; $j<4; $j++ )
   {
       if ( ($dedox[$i] > $botonesx0[$j] && $dedox[$i] < $botonesx1[$j] )
            && ($dedoz[$i] > $botonesz0[$j] && $dedoz[$i] < $botonesz1[$j] )
            && ($dedoy[$i] > 0 && $dedoy[$i] < $botonesy1[$j] ) ){
          $b[$j][$cuantos[$j]]=$i;
          $cuantos[$j]+=1;
       }
    }


for ( $j=0; $j<4; $j++ ){
   if ($cuantos[$j] == 0 ) {
       if ( $j ==  0 ) boton1.scaleY=1;
       if ( $j ==  1 ) boton2.scaleY=1;
       if ( $j ==  2 ) boton3.scaleY=1;
       if ( $j ==  3 ) boton4.scaleY=1;
   }
   else{
       $cual=$b[$j][0];
       $minimo = $dedoy[$cual];
       $ind = $cual;
       for ( $i=1; $i<$cuantos[$j]; $i++ ){
          $cual=$b[$j][$i];
          if ($dedoy[$cual] < $minimo ){
             $minimo = $dedoy[$cual];
             $ind=$cual;
          }
       }
       if ( $j ==  0 ) boton1.scaleY=$dedoy[$ind]/2;
       if ( $j ==  1 ) boton2.scaleY=$dedoy[$ind]/2;
       if ( $j ==  2 ) boton3.scaleY=$dedoy[$ind]/2;
       if ( $j ==  3 ) boton4.scaleY=$dedoy[$ind]/2;

    }
}// for para barrer cada uno de los botones
```

Referencias

[1] "Designing Special-Purpose Input Devices", W. Bradford Paley, Digital Image Design Incorporated 72 Spring Street; New York, NY 10012 .

[2] "Sketching in 3D", Robert Zeleznik , Brown University, Department of Computer Science, Providence, RI 02912, (401) 863-7653; bcz@cs.brown.edu August 26, 1998.

[3] Virtual Reality Systems, R. A. Earnshaw, Academic Press 1993.

[4] Motion Capture White Paper; Scott Dyer, Windlight Studios; Jeff Martin, Alias|Wavefront; John Zulauf, Alias|Wavefront, December 12, 1995.

[5] "Human movement tracking technology", Axel Mulder, Technical Report 94-1, School of Kinesiology, Simon Fraser University, July 1994.

---

[6] "A Practical Approach to Motion Capture: Acclaim's optical motion capture system", Wes Trager, Advanced Technologies Group.
[7] http://www.biovision.com/products.htm.
[8] http://www.ascension-tech.com/
[9] http://www.polhemus.com/home.htm
[10] http://www.aliaswavefront.com
[11] http://www.metamotion.com/
[12] M68HC11 Family, 68HC11 8-bit Microcontrollers, http://www.motorola.com/
[13] http://www.javasoft.com
[14] http://www.aliaswavefront.com/maya
[15] http://www.javasoft.com/products/java-media/3D/
[16] http://www.sgi.com
[17] Learning Maya. Alias Wavefront, Toronto Canada 1998.
[18] Using Maya, Animation. Alias Wavefront, Toronto Canada 1998.
[19] Using MEL. Alias Wavefront, Toronto Canada 1998.
[20] "User Performance in Relation to 3D Input Device Design", Shumin Zhai, IBM Almaden Research Center, San Jose, CA 95123, USA +1 408 9271112 , zhai@almaden.ibm.com.
[21] http://www.wavelan.com/support/software/y2000.html.
[22] http://www.usb.org/
[23] http://www.firewire.com/
[24] M68HC05 Family, 68HC05 8-bit Microcontrollers, http://www.motorola.com/
[25] http://www.cypress.com/.
[26] "Smart Fabric, or Washable Computing", Digest of Papers of the First IEEE International Symposium on Wearable Computers, pp. 167-8, October 13-14, 1997, held in Cambridge, Massachusetts. (http://www.media.mit.edu/~rehmi/fabric/index.html)
[27] Using Maya Modeling. Alias Wavefront, Toronto Canada 1998.
[28] http://www.javasoft.com/products/javacomm/