UNIVERSIDAD SIMÓN BOLÍVAR
DECANATO DE ESTUDIOS PROFESIONALES
COORDINACIÓN DE INGENIERÍA DE LA COMPUTACIÓN

Captura de movimiento de personas con múltiples sensores Kinect

Por:
Alfonso Ros, Ismael Mendonça

Realizado con la asesoría de:
Prof. Carolina Chang

PROYECTO DE GRADO
Presentado ante la Ilustre Universidad Simón Bolívar
como requisito parcial para optar al título de
Ingeniero en Computación

Sartenejas, Abril 2012

UNIVERSIDAD SIMÓN BOLÍVAR
DECANATO DE ESTUDIOS PROFESIONALES
COORDINACIÓN DE INGENIERÍA DE LA COMPUTACIÓN

ACTA FINAL PROYECTO DE GRADO

Captura de movimiento de personas con múltiples sensores Kinects

Presentado por:
Alfonso Ros, Ismael Mendonça

Este Proyecto de Grado ha sido aprobado por el siguiente jurado examinador:

Prof. Carolina Chang

Prof. Alexandra La Cruz

Prof. Eduardo Roa

Sartenejas, 16/04/12

RESUMEN

La captura de movimiento se basa en un conjunto técnicas y métodos para el registro de movimientos de un actor que se desenvuelve en una escena para su posterior análisis. Estos métodos son empleados en una gran variedad de áreas como la industria del entretenimiento, el deporte, la medicina, la seguridad, entre otras.

El sensor *Kinect* desarrollado por *Microsoft* es un dispositivo diseñado con el propósito de realizar captura de movimiento para su consola de videojuegos *XBOX*. La introducción de dicho sensor al mercado y el desarrollo de controladores libres para el mismo han impulsado la creación de nuevas aplicaciones por parte de desarrolladores en todo el mundo.

Actualmente, el software dedicado a esta tarea es de alto costo o de código cerrado. El presente trabajo comprende el desarrollo de un sistema para realizar captura de movimiento de un individuo utilizando múltiples dispositivos *Kinect*. Para esta tarea se implementó un algoritmo de aprendizaje de árboles de decisión llamado *Randomized Decision Forests* [25].

Este algoritmo es capaz de aprender ciertas características de un conjunto de imágenes de profundidad con el fin de clasificar los píxeles de una imagen y así obtener la pose de la persona. Para el entrenamiento del algoritmo, se sintetizaron imágenes utilizando programas de diseño como *Maya* y *Motion Builder*. El clasificador arrojó resultados de clasificación superiores al 60 % de los píxeles de una imagen, además de soportar corridas en tiempo real. Las imágenes sintéticas como datos de entrenamiento resultaron convenientes por su facilidad y precisión de etiquetamiento y por su capacidad de generalización para los datos reales.

El clasificador implementado puede ser utilizado no solamente para reconocer poses de personas, sino para cualquier otro problema de reconocimiento de objetos. El clasificador se adapta dependiendo de los datos de entrenamiento suministrados, no es necesario ningún cambio en la estructura lógica del mismo.

Para estimar la posición de las articulaciones del esqueleto se utilizó *mean-shift* [28] sobre las regiones clasificadas para obtener dichas posiciones. Una vez obtenidos dichos puntos por cada sensor, se integran las vistas de los mismos, utilizando una aproximación basada en descomposición de valor singular [2].

Índice general

Índice de cuadros

Índice de figuras

Glosario

CMOS *Active Pixel Sensor.* Sensor que detecta la luz en las cámaras. 7

IR *Infrared.* Luz infrarroja. 7

Kernel *Núcleo.* En estadística se refiere a una función ponderada usada en estimación de densidad por kernel. 14

Offset *Desplazamiento.* Representa un desplazamiento respecto a un valor fijo. 49

RF *Random Forest.* Algoritmo de aprendizaje de máquina, utilizado en problemas de clasificación o regresión. También es conocido con el nombre de *Bosques aleatorios*. 17

RGB *Red, Green, Blue.* Modelo de color rojo, verde, azul. 6

SVD *Singular value decomposition.* Descomposición de valor singular. Técnica de factorización de matrices. 38

Threshold *Umbral.* Valor fijo respecto al cual se realizan comparaciones con el fin de separar conjuntos de datos. 26

VGA *Video Graphics Array.* Sistema gráfico de pantallas. Usado para denotar una pantalla de computadora estándar. 7

Capítulo 1

Introducción

La captura de movimiento ha venido desarrollándose desde la década de 1870 [17, 21]. No fue sino hasta un siglo después, a finales de los años 70 que con el desarrollo de la computación y la invención de nuevas tecnologías, se inició el estudio de la captura de movimiento de individuos utilizando técnicas de visión por computadora.

Formalmente, la captura de movimiento consiste en poder registrar y digitalizar los movimientos de un actor para su posterior análisis. Es importante señalar que este término se ha utilizado mayormente en relación a la captura de movimientos de gran escala, es decir, movimientos de la cabeza, brazos, torso y piernas; dentro de esta definición no se incluyen movimientos de pequeña escala como son expresiones faciales o gestos con las manos [20].

La captura de movimiento ha sido utilizada para un gran número de aplicaciones, entre las que se pueden nombrar: vigilancia inteligente, reconocimiento, control, interfaces humano-máquina, animación de caracteres, realidad virtual, interpolación de vistas, y análisis de movimiento.

Recientemente, gracias a el desarrollo de nuevas tecnologías se ha generado un mayor interés en el área de la captura de movimiento a la vez que nuevos dispositivos como cámaras, rastreadores magnéticos, sistemas infrarrojos, sistemas de radar, entre otros, han permitido afrontar este problema de una forma más conveniente.

El sensor *Kinect* [18] es un dispositivo creado por *Microsoft* para la consola de videojuegos

XBOX 360. Inmediatamente a su salida, se empezaron a desarrollar controladores no oficiales cuyas versiones estables fueron liberadas para noviembre de 2010. Estos controladores permitieron la utilización del *Kinect* en una computadora a través de su puerto USB.

El sensor *Kinect* posee, además de micrófonos y una cámara RGB, una cámara de profundidad o cámara 3D que permite generar imágenes en tres dimensiones de la escena. Este tipo de cámaras ya existían anteriormente a la salida del sensor *Kinect*, pero se trataban de dispositivos costosos que no estaban al alcance de todos, por ende, el número de aplicaciones era limitado.

Con la introducción de cámaras de profundidad de bajo costo como el sensor *Kinect*, se ha creado un gran interés por el desarrollo de todo tipo de aplicaciones, y en nuestro caso particular, para la captura de movimiento.

Para atacar el problema de captura de movimiento existen diferentes aproximaciones dependiendo de las características del problema. Entre ellas podemos encontrar sistemas que emplean uno o múltiples sensores de profundidad [22], sensores de intensidad de luz [32] o sensores de luz infrarroja [3]; así como también existen distintos enfoques respecto al objetivo de la captura como seguimiento, reconocimiento o estimación de pose de una o más personas.

En el presente trabajo se hace uso de múltiples sensores *Kinects* posicionados en lugares fijos alrededor de la escena, enfocándose en una aproximación basada en reconocimiento de pose con el fin de realizar la captura de movimiento.

El objetivo general de este trabajo es diseñar, implementar y probar una aplicación que sea capaz de realizar el seguimiento de un individuo mediante la utilización de tres sensores *Kinect*, y extraer la información de movimiento asociada. Concretamente, se plantea utilizar

el algoritmo de bosques aleatorios ***random forests*** [5] para clasificar los píxeles de una imagen según diferentes partes del cuerpo para realizar el reconocimiento de pose, utilizar el algoritmo *mean - shift* para estimar los puntos correspondientes a las articulaciones del esqueleto y encontrar la transformación geométrica que relaciona un Kinect respecto a los demás con el fin integrar las múltiples vistas.

Para lograr obtener un sistema de captura de movimiento que cumpla con las expectativas de este proyecto, se plantean los siguientes objetivos específicos:

1. Realizar un estudio sobre el problema que representa la captura de movimiento utilizando cámaras 3D y las aproximaciones ya existentes para resolver el mismo.
2. Reconocer poses de un individuo mediante el uso del algoritmo de bosques aleatorios.
3. Proponer un conjunto de puntos 3D correspondientes a las articulaciones del esqueleto de la persona.
4. Integrar los datos generados por tres sensores Kinect a manera de tener una visión de 360 grados de la escena.
5. Evaluar el rendimiento y resultados obtenidos con el sistema desarrollado.

El documento se encuentra dividido en 6 capítulos, incluida la introducción. El capítulo 2 introduce las definiciones necesarias para comprender los detalles más significativos de la implementación realizada. El capítulo 3 presenta una descripción detallada del algoritmo **random forest** y la estrategia de estimación de articulaciones implementados en el presente trabajo. El capítulo 4 ofrece una explicación de la técnica utilizada para integrar las múltiples vistas de los sensores *Kinect*. El capítulo 5 se dedica a explicar los experimentos que se realizaron para evaluar el algoritmo y la integración de vistas. Finalmente, el capítulo 6 contiene las conclusiones y recomendaciones para trabajos futuros.

Capítulo 2

Marco teórico

Este capítulo resume todos aquellos elementos teóricos de interés para desarrollar la presente investigación. Se empieza por dar una explicación sobre la definición del proceso de la captura de movimiento. Posteriormente, se explica la estructura y funcionamiento del sensor *Kinect*, así como los aspectos relacionados con la calibración de cámaras y el problema de calibración estéreo. Finalmente, se describen los algoritmos utilizados para la implementación del presente proyecto, correspondientes a la teoría de árboles de decisión, utilizada para el algoritmo de Bosques aleatorios o *randomized decision forests*, y las técnicas de búsqueda local, como mejoramiento iterativo y algoritmo genético para el problema de múltiples cámaras.

2.1. Captura de movimiento

En [23] se define la captura de movimiento (CM) como el proceso que consiste en traducir movimientos reales a representaciones digitales y se logra mediante el seguimiento de un conjunto de puntos de interés en una escena por un tiempo determinado. Se describe en [23] como fin u objetivo de un sistema de CM: "el seguimiento en tiempo real de un número ilimitado de puntos de interés, sin limitación de espacio, a la más alta frecuencia posible con el menor margen de error".

La CM puede realizarse sobre cualquier actor que posea movimiento. Para realizar el seguimiento del actor, se definen un conjunto de **puntos de interés** sobre el mismo, los cuales se localizan en áreas de éste que posean mayor información sobre sus movimientos. Estos puntos corresponden a conexiones entre partes rígidas del actor, por ejemplo, algunos

de estos puntos pueden corresponder a articulaciones del cuerpo, tales como: las rodillas, codos, hombros, entre otros. Existen diferentes maneras de hacer la CM. Los sistemas ópticos son los mayormente utilizados para esta tarea. Por lo general, dichos sistemas se valen de marcadores reflectivos o constituidos por LEDs, que se colocan sobre el actor para demarcar los **puntos de interés** con el fin de determinar la posición del mismo en la escena. Sin embargo, actualmente existen sistemas ópticos que no requieren de la utilización de marcadores, éstos son los llamados **Sistemas sin marcadores**.

Actualmente, con el desarrollo de técnicas en el área de visión de computadoras se ha dado pie al desarrollo de estos sistemas, como su nombre lo indica, no requieren que el actor vista ningún equipo especial para realizar la CM. Estos sistemas se valen de algoritmos que mediante el análisis de la entrada de datos, dado por las cámaras, son capaces de identificar figuras humanas para la realización del seguimiento. Algunos sistemas utilizan múltiples cámaras que digitalizan diferentes vistas de la escena, las cuales, mediante técnicas de estereopsis, son usadas para unificar las posiciones de los **puntos de interés**.

El sensor*Kinect* de **Microsoft** es un dispositivo que contiene una cámara de profundidad que es utilizada para realizar CM para los videojuegos de la consola **XBOX**.

2.2. Sensor Kinect

El sensor Kinect desarrollado por Microsoft salió a la venta el 4 de noviembre del 2010 en Estados Unidos. Se trata de un sensor de movimiento diseñado para la consola de videojuegos *XBOX*, permite que los usuarios puedan interactuar con la consola sin necesidad de un control, esto lo hace mediante una interfaz natural de gestos o comandos hablados.

El sensor Kinect está constituido por un servomotor, una cámara RGB, una cámara de profundidad y cuatro micrófonos.

La cámara de profundidad está compuesta de dos partes, un proyector de rayos infrarrojos

(IR) y un sensor CMOS monocromático. El sensor que percibe los rayos infrarrojos puede capturar datos de video en 3D bajo cualquier condición de luz. Éste, además, opera en resolución VGA (640 x 480) con 11-bit de profundidad a 30 cuadros por segundo, provee 2048 niveles de sensibilidad.

La cámara RGB posee una resolución de 8-bit VGA (640 x 480 píxeles) que opera a través de un sensor CMOS con un filtro de Bayer igualmente a 30 cuadros por segundo.

El sensor Kinect posee un rango límite recomendado de 1.2 a 3.5 metros utilizado con el software del *XBOX*. Sin embargo, el sensor puede mantener seguimiento en un rango de aproximadamente 0.7 a 6 metros. Posee un campo angular de visión de 57° horizontalmente y 43° verticalmente, mientras que el motor puede girar verticalmente 27° en ambas direcciones.

El Kinect posee 4 micrófonos ubicados a los extremos del sensor, cada canal procesa 16-bit y un rango de muestreo de 16kHz. Estos micrófonos son la única razón por la cuál el sensor es tan ancho.

2.2.1. Controladores para el Kinect

El sensor Kinect posee un puerto USB 2.0 para la conexión con la consola de videojuegos *XBOX 360*. El protocolo con el que se comunica el sensor con la consola no fue protegido con ningún tipo de cifrado, lo que facilitó enormemente la creación de controladores que permitieron el uso del sensor a través de una computadora.

Dado que es relativamente fácil generar un controlador propio para el Kinect, se pueden encontrar gran cantidad de éstos disponibles en internet al igual que tutoriales para la implementación de los mismos. Sin embargo, existen dos que logran destacarse a la hora del uso para el desarrollo de aplicaciones: **OpenKinect** y **OpenNI**.

- **OpenKinect:** Es una librería que se ha venido desarrollando mediante técnicas de ingeniería inversa del protocolo utilizado por el *Kinect* por parte de una gran comunidad

de desarrolladores que colaboran con este proyecto. Esta librería es de código abierto y ofrece control sobre la mayoría de los dispositivos de hardware del sensor, más no ofrece facilidades en cuanto a algoritmos especializados para el procesamiento de imágenes.

- **OpenNI:** se trata de un *framework* desarrollado por la empresa **PrimeSense** que ofrece gran facilidad para el desarrollo de aplicaciones utilizando el sensor *Kinect*. Al igual que **OpenKinect** ofrece control sobre la mayoría de los dispositivos de hardware del *Kinect*, además, posee compatibilidad con la librería **NITE** de **PrimeSense** que contiene algoritmos para el procesamiento de imágenes, detección y seguimiento de individuos. Sin embargo, el código de esta librería no es abierto.

2.3. Sustracción de fondo

La sustracción de fondo o *background sustraction* es una técnica muy común en procesamiento de imágenes. Por lo general es utilizada para extraer de una imagen aquellos elementos de interés para ser analizados de forma aislada. En el presente trabajo, por ejemplo, se utiliza para extraer únicamente los píxeles del individuo de las imágenes producidas por el sensor *Kinect*.

La técnica consiste en aprender un modelo estadístico del fondo de una escena, cuyos elementos exhiben, por lo general, un comportamiento regular. El modelo es utilizado para identificar elementos en la imagen que no corresponden al modelo aprendido. Los nuevos elementos en la escena corresponderán a objetos que se encuentran en el primer plano de la imagen o *foreground*.

Mediante la aplicación de la sustracción de fondo es posible segmentar elementos de interés en una imagen. Por lo general, es utilizada en aplicaciones como vigilancia para identificar personas u objetos en movimiento en una escena.

Existen varios factores que pueden influir en la sustracción de fondo, uno de los más determinantes viene dado por el ambiente donde se desarrolla la escena, que puede ser cerrado o abierto (*indoor/outdoor environments*). Otros factores influyentes en la sustracción de fondo son: cambios de iluminación y presencia de elementos dinámicos.

Existen varias aproximaciones para tratar el problema de sustracción de fondo, entre las que se pueden nombrar: **Diferencia de cuadros** [4], **Promedio de fondo** [4], ***Mezcla de Gaussianas*** [26], **estimación basada en *Mean - Shift*** [13], entre otros.

2.3.1. *Mezcla de Gaussianas*

La mezcla de Gaussianas es una técnica que consiste en combinar un conjunto de distribuciones Gaussianas sobre una misma variable. Puede escribirse como una suma ponderada de funciones de densidad Gaussianas.

Recordemos que una distribución Gaussiana multivariada (n-dimensional) viene dada por:

$$p(x;\mu,\Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp^{(-\frac{1}{2}(x-\mu)^t\Sigma^{-1}(x-\mu))} \tag{2.1}$$

Donde μ es el vector de medias y Σ la matriz de covarianza. Se define entonces la suma ponderada de K distribuciones Gaussianas como:

$$pm(x) = \sum_{k=1}^{K} w_k \cdot p(x;\mu_k,\Sigma_k) \tag{2.2}$$

Variando los parámetros K, μ_k, Σ_k y w_k de cada función de densidad Gaussiana, esta combinación de distribuciones puede utilizarse para describir distribuciones de probabilidades más complejas.

En el caso de sustracción de fondo, esta mezcla de distribuciones es empleada para modelar el comportamiento de cada píxel en la escena. Por lo general, en una escena puede cambiar el

grado de iluminación tanto paulatinamente como en un instante, así como nuevos elementos pueden entrar o salir de la misma. Es por ello que múltiples distribuciones Gaussianas son necesarias para modelar o aproximar estos procesos.

Como es descrito en [26] cada cierto tiempo los parámetros de las distribuciones Gaussianas son actualizados, esto con el fin de modelar los cambios que se puedan observar en la escena a través del tiempo. Las distribuciones son utilizadas para evaluar si un píxel pertenece al fondo de la imagen, de no ser así, estos son agrupados usando componentes conexas para delimitar los elementos que pertenecen al primer plano de la imagen. Luego, mediante un algoritmo de seguimiento se mantienen las componentes que conforman los elementos del primer plano de la imagen en cada cuadro.

El resultado final es una imagen con los elementos pertenecientes al primer plano. Comúnmente, un problema en las imágenes resultantes es la aparición de pequeños grupos de píxeles que deberían pertenecer al fondo. Estos grupos se denominan como ruido en la imagen y pueden ser eliminados parcialmente con la técnica que explica en la siguiente sección.

2.4. Transformaciones morfológicas en imágenes (Erosión y dilatación)

El procesamiento morfológico es un campo riguroso de la matemática que se basa en teoría de conjuntos. Comúnmente se utiliza en imágenes digitales para el análisis de estructuras geométricas.

Entre las transformaciones básicas que se le pueden aplicar a una imagen se encuentra la **erosión** y la **dilatación**. Estas transformaciones se pueden utilizar para remover el ruido de una imagen, aislar elementos o unir aquellos que sean próximos en la misma.

Ambas transformaciones se aplican a través de un procedimiento similar. Se utiliza un *kernel* o núcleo que consiste en una región con dimensiones fijas menores que las de la imagen. El *kernel* contiene un llamado "punto de anclaje" que se centra en un píxel que será transformado según un operador definido dentro del mismo *kernel*. El operador para aplicar una erosión es el **mínimo** y para aplicar una dilatación es el **máximo**. Esto quiere decir, por ejemplo, si se aplica un *kernel* con el operador mínimo, el píxel ubicado en el "punto de anclaje" es reemplazado con el valor más bajo de los píxeles contenidos dentro de la región del *kernel*.

La transformación se le aplica a todos los píxeles de la imagen en un proceso iterativo. Este proceso se puede repetir dependiendo de los resultados que se deseen obtener.

En la próxima sección se describe un algoritmo que puede ser utilizado para identificar los elementos aislados y su jerarquía en la imagen.

2.5. Búsqueda de contornos en imágenes binarias

Una imagen binaria es aquella cuyos píxeles están conformados únicamente por ceros y unos. Es necesario a veces encontrar los contornos o componentes conexas formadas por los píxeles de mismo valor.

El algoritmo descrito en [27] encuentra los contornos de una imagen binaria y su jerarquía. Por ejemplo considere la figura 2.1 donde se muestra una imagen binaria donde las partes negras representan los píxeles con ceros y las partes blancas representan los píxel con unos. Cada una de estas partes conforman una componente conexa que puede ser de dos tipos: **0-componente** y **1-componente** para los píxeles conectados con ceros o unos respectivamente. En la imagen se tiene *S1* y *S3* como **0-componente** y *S2*, *S4* y *S5* como **1-componente**. Cada **1-componente** tiene un **borde externo** y puede tener también un **borde interno** o **borde hueco** como se muestra en la imagen.

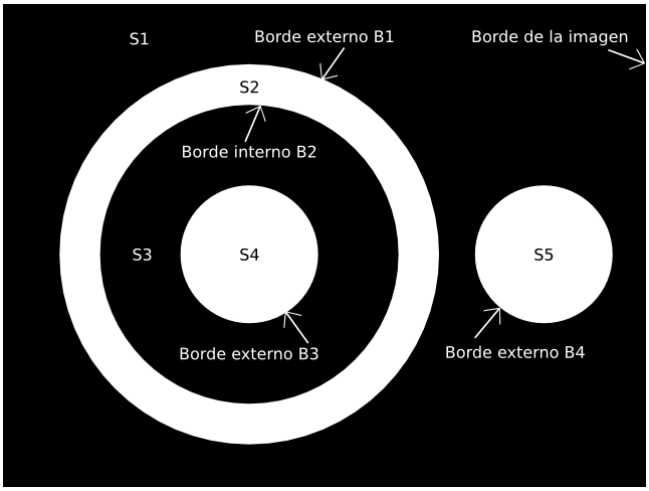

Figura 2.1: Ejemplo de imagen binaria con contornos y componentes conexas.

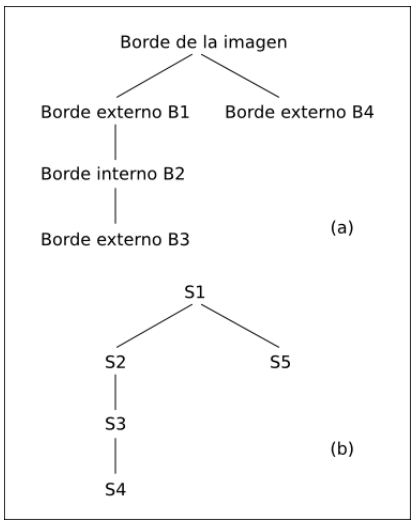

Figura 2.2: Jerarquía de los contornos (a) y jerarquía de las componentes conexas (b).

Los contornos se pueden organizar en una jerarquía topográfica desde el borde de la imagen hasta el contorno más interno. Se puede hacer lo mismo con las componentes conexas. En la figura 2.2 se muestra ambas jerarquías para la imagen 2.1.

El algoritmo marca los bordes externos e internos de cada **1-componente** y construye la jerarquía entre ellos. Para esto, se recorre la imagen píxel por píxel hasta que se encuentre el primer píxel con un valor diferente de cero. Después de esto, se sigue el borde marcando los píxeles con un identificador único hasta que se vuelve al píxel de partida. Cuando se termina de marcar un borde con el identificador, éste se guarda junto con el tipo de borde (externo o interno) para poder deducir los padres de los próximos bordes que se encuentren según la regla presentada en el cuadro 2.1. Se sigue recorriendo la imagen píxel por píxel y repitiendo el procedimiento por cada borde nuevo que aparezca hasta que se llegue al final de la imagen.

Borde anterior / Borde encontrado	Borde externo	Borde interno
Borde externo	Borde anterior	Padre del borde anterior
Borde interno	Padre del borde anterior	Borde anterior

Cuadro 2.1: Regla para deducir el borde padre para un nuevo borde encontrado

2.6. Algoritmo *Ramer–Douglas–Peucker* para reducir los vértices de una curva

El algoritmo comúnmente conocido como *Ramer–Douglas–Peucker* [8, 31] es utilizado para obtener una curva con menor cantidad de vértices a partir de una más compleja. Este algoritmo puede ser utilizado sobre las componentes halladas por el algoritmo presentado en la sección anterior y poder obtener una curva más suave que elimine errores en los bordes del contorno.

El algoritmo consiste en un procedimiento recursivo que divide la curva y selecciona los vértices que serán conservados. Al principio se busca el vértice que esté más alejado de la línea recta formada entre los puntos extremos de la curva. Si la distancia que separa dicho vértice es menor que un ϵ, todos los vértices entre los extremos son descartados. En el caso contrario, el vértice se mantiene para la curva resultante y el procedimiento se repite recursivamente con los vértices ubicados entre el vértice más alejado y los extremos.

Cuando se completa la recursión se obtiene una curva compuesta solo por los puntos que no fueron descartados.

2.7. *Mean Shift*

"*Mean Shift* es un método robusto para encontrar extremos locales en la distribución de densidad de un conjunto de datos" [4]. Este algoritmo puede verse, además, como una

técnica no paramétrica de agrupación o *clustering* que no requiere conocimiento previo sobre el número de *clusters* a utilizar.

Se dice que es robusto ya que no toma en cuenta aquellos puntos fuera de lugar o *outliers* de los datos. El algoritmo opera procesando sólo los puntos que se encuentren dentro de una ventana local. La idea consiste en estimar el vector de medias que indica el desplazamiento de la ventana hasta converger a la moda de la distribución.

Para estimar la distribución de los datos, el algoritmo emplea técnicas de **estimación de densidad por *kernel***. Por lo general, se utiliza un *Kernel* Gaussiano para estimar la media ponderada de los puntos vecinos x_i a un punto fijo x dentro de la ventana, con el fin de obtener el vector de desplazamiento que recibe el nombre de *mean shift*. Según [28] el cálculo de dicho vector viene dado por la ecuación:

$$m(\mathbf{x}) = \frac{\sum_i \mathbf{x_i} G(\mathbf{x} - \mathbf{x_i})}{\sum_i G(\mathbf{x} - \mathbf{x_i})} - \mathbf{x} \tag{2.3}$$

Donde G es la función de *Kernel* Gaussiano.

En el área de visión por computadora, *mean - shift* suele ser utilizado para seguimiento de puntos en varios cuadros. En cuanto a la captura de movimiento resulta ser una técnica ideal para tener un registro de los puntos de interés a los que se remita el estudio.

2.8. Matriz dispersa

Una matriz dispersa es una matriz mayormente poblada con ceros. Los datos dispersos, debido a su estructura son de fácil compresión, y es esta compresión la que permite disminuir de gran forma el espacio de almacenamiento de estas matrices. En la computadora, las imágenes son representadas por matrices que poseen información de los elementos de la escena. Por lo general, al sustraer el fondo de una imagen, se espera que una gran cantidad de

elementos queden sin información (aproximadamente dos tercios de la imagen corresponden al fondo). La imagen puede ser almacenada utilizando un formato de compresión basado en matrices dispersas. Existen varios formatos, de los cuales se explicará el formato Yale.

2.8.1. Formato Yale

El formato de compresión Yale [9,10] almacena una matriz M de tamaño $m \times n$ en forma de filas utilizando tres arreglos de una dimensión. Sea NZ el número de elementos distintos de cero de M, se tiene que:

- El primer arreglo A, de tamaño NZ contiene todos los elementos de la matriz M que sean diferentes de cero.
- El segundo arreglo IA es de tamaño $m+1$, y cada elemento $IA(i)$ corresponde al índice de A del primer elemento distinto de cero de la fila i.
- El tercer arreglo JA es del mismo tamaño de A y contiene el índice de la columna en M de cada elemento de A.

A continuación se enumeran algunas propiedades y restricciones asociadas a este formato:

- El primer elemento de la fila i de A corresponde a $A(IA(i))$.
- El tamaño de la fila i se determina por $IA(i+1) - IA(i)$.
- Para algún elemento $A(i)$, su índice de columna es $JA(i)$.
- El formato de Yale ahorra memoria sólo cuando:

$$NZ < \frac{(m(n-1)-1)}{2}$$

- Ningún elemento de la diagonal puede ser omitido del arreglo A de valores.

2.9. Árboles de decisión

Un árbol de decisión es una estructura constituida por un grafo tipo árbol que representa un conjunto de decisiones y sus posibles consecuencias.

En el área de aprendizaje de máquina los árboles de decisión son empleados para problemas de clasificación o regresión.

En los problemas de **clasificación**, se busca estimar un valor en relación a un conjunto de variables dadas que representan un **ejemplo** o **instancia** del problema de interés. Concretamente, como se define en [19], "constituyen una técnica de aprendizaje que consiste en aproximar una función objetivo de valores discretos, en donde la función aprendida es representada por el árbol de decisión".

En los problemas de **regresión**, la aproximación a la función objetivo que constituye el árbol es continua y no discreta. Las variables de un ejemplo son valores continuos, que se comparan con un valor almacenado en el nodo el cual define hacia que rama se bifurcará el ejemplo evaluado.

En un árbol de decisión, cada nodo representa una variable por la que se dividen los ejemplos a clasificar, y cada una de sus ramas representan los distintos valores que puede tomar dicha variable. Los nodos hoja contienen la clasificación estimada del ejemplo.

Una vez entrenado el árbol, se clasifica un ejemplo empezando por el nodo raíz, se evalúa la variable en dicho nodo y luego se mueve hacia abajo por la rama correspondiente al valor de la variable en el ejemplo dado. Este proceso se repite en cada nodo del subárbol hasta llegar a un nodo hoja. En la sección B.1 del apéndice B se explica con detalle el proceso de entrenamiento del árbol.

2.10. Bosques aleatorios

Los **bosques aleatorios o *Random Forests* (RF)** es un algoritmo creado por Leo Breiman y Adele Cutler descrito en [5]. Consiste en una agrupación de varios árboles de decisión que crecen con alguna forma de aleatorización. Constituyen una buena herramienta de predicción ya que pueden tratar tanto problemas de regresión como de clasificación con múltiples clases, son rápidos, robustos y relativamente fáciles de entrenar [16].

Los RF entrenan varios árboles y toman como entrada un ejemplo que consiste en un vector de variables que es clasificado por cada árbol del bosque. Cada árbol emite una respuesta que es considerada como un "voto" por una clase. El bosque escoge la clase que mayor cantidad de votos haya obtenido. En el caso del problema de regresión, la respuesta del clasificador es el promedio de las respuestas emitidas por los árboles del bosque.

2.10.1. Entrenamiento

Los árboles se construyen de arriba hacia abajo, empezando en el nodo raíz, y en cada nodo se realiza una prueba que separa el espacio de ejemplos de entrenamiento a medida en que se desciende en el árbol. Cada árbol se construye hasta su mayor extensión posible. Al llegar a un nodo hoja, se almacena la distribución de probabilidades de las clases de los ejemplos que han llegado a dicho nodo. Los detalles sobre el proceso de construcción o entrenamiento de los árboles se explican con mayor profundidad en la sección B.2 del apéndice B, así como las métricas para medir los errores de clasificación del algoritmo.

2.11. Calibración de una cámara

Según [29] el proceso de calibración de una cámara consiste en estimar los parámetros intrínsecos y extrínsecos asociados a ésta. El objetivo central de la calibración es definir las ecuaciones que relacionan coordenadas conocidas de un conjunto de puntos 3D con sus proyecciones, la solución a estos sistemas corresponde a los parámetros de la cámara. Para ayudar a entender este proceso es preciso conocer los aspectos básicos sobre geometría de cámara, los cuales se detallan en la sección A.2 del apéndice A.

Para realizar la calibración de una cámara, es necesario contar con un conjunto de imágenes de un objeto que funcione como "patrón de calibración". Este objeto debe poseer una geometría y una posición conocida en el espacio con el fin de poder extraer los puntos de interés para formar el sistema de ecuaciones.

Por lo general, como patrón de calibración se utiliza un tablero de ajedrez, o cualquier objeto que facilite la detección de puntos en una imagen. El tablero de ajedrez es un buen candidato ya que posee la característica de tener una disposición en forma de cuadrícula de un conjunto de cuadros negros sobre un fondo blanco, de esa forma, es fácil conocer la posición de los vértices de cada cuadrado gracias al alto contraste y la geometría simple del patrón.

A continuación se explica con detalle en qué consisten los parámetros de la cámara.

2.11.1. Parámetros extrínsecos

Hallar los parámetros extrínsecos de la cámara significa hallar la matriz de rotación (ver apéndice A.1.1) R y el vector de traslación $\mathbf{T}$ que definen la transformación entre un punto en el marco de referencia del mundo real y un punto en el marco de referencia de la cámara. En el proceso de calibración, el marco de referencia del mundo real es conocido, mientras que

el de la cámara no lo es.

Sea un punto P dado por las coordenadas $X = (x, y, z)$ en el marco de referencia del mundo real, y sea ese mismo punto P dado por las coordenadas $X' = (x', y', z')$ en el marco de referencia de la cámara. Conocer los parámetros extrínsecos significa hallar la matriz de rotación R en $\mathbb{R}^{3\times3}$ y el vector de traslación $\mathbf{T}$ en $\mathbb{R}^3$ que serán utilizados para aplicar una transformación entre los puntos X y X' (consultar la sección A.1.2 del apéndice A).

2.11.2. Parámetros intrínsecos

Los parámetros intrínsecos son aquellos que definen el modelo de distorsión de los lentes y el modelo de geometría de la cámara [29]. Ambos modelos están conformados por una serie de parámetros que deben ser estimados durante el proceso de calibración.

Para el modelo geométrico, se precisa estimar los siguientes parámetros:

- Distancia focal f.
- Relación de aspecto α dada por la relación entre la distancia horizontal efectiva s_x y la distancia vertical efectiva s_y de la imagen.
- Coordenadas del centro de la imagen (o_x, o_y).

El modelo de distorsión de los lentes comprende un conjunto de parámetros que pueden dividirse en dos grupos de acuerdo al tipo de distorsión (explicado en la sección A.3 del apéndice A), como son:

- **Distorsión radial:** Comprende los parámetros k_1, k_2 y k_3 que representan los primeros tres términos de una serie de Taylor que se utilizan para corregir la distorsión radial con las ecuaciones A.7 y A.8 expuestas en el apéndice A.

- **Distorsión tangencial:** Comprende los parámetros p_1 y p_2 que son utilizados para corregir la distorsión tangencial con las ecuaciones A.9 y A.10 expuestas en el apéndice A.

Transformación de coordenadas en el plano de la imagen al mundo real

Una vez obtenidos los valores de los parámetros intrínsecos de la cámara, es posible a partir de ellos, obtener la transformación entre coordenadas en el plano de la imagen y el mundo real. Dadas las coordenadas x, y en el plano de la imagen y un valor de profundidad z del mundo real, se obtiene la transformación de la siguiente manera:

$$x = (x - o_x)zf_x \tag{2.4}$$

$$y = (y - o_y)zf_y \tag{2.5}$$

Con (o_x, o_y) las coordenadas del centro de la imagen, f_x la distancia focal en x y f_y la distancia focal en y.

2.12. Visión estéreo

La visión estéreo o estereopsis es el proceso que a través de la geometría de varios puntos de vista de una escena, extrae la información sobre su estructura tridimensional [29]. Se divide a su vez en otros dos grandes problemas: el problema de correspondencia y el problema de reconstrucción.

2.12.1. Problema de correspondencia

El problema de correspondencia consiste en determinar qué puntos de un par de imágenes corresponden a proyecciones de un mismo elemento de una escena. El problema de correspon-

dencia puede ser explicado con el ejemplo del ojo humano. Se trata entonces de determinar qué elementos vistos por el ojo izquierdo corresponden a qué elementos vistos por el ojo derecho.

2.12.2. Problema de reconstrucción

Dado un conjunto de elementos correspondientes de un par de imágenes, y posiblemente información sobre la geometría del sistema estéreo, el problema consiste en determinar la estructura 3D de la escena observada. Haciendo referencia al ejemplo del ojo humano, este problema es resuelto por el cerebro al calcular la profundidad de la escena a raíz de la diferencia de disparidad entre los elementos correspondientes vistos por cada ojo.

2.13. Transformación rígida entre dos conjuntos de puntos

Para dos conjuntos de puntos que se relacionan a través de una transformación rígida, se desea conocer la matriz de rotación y el vector de translación que resuelven el sistema de ecuaciones. En [2] se describe un método para lograr obtener una solución para este problema.

El método se centra en la idea de que ambos conjuntos poseen los mismos centroides para separar el cálculo de la matriz de rotación. Para hallar la matriz de rotación se usa una descomposición de valor singular sobre una matriz H que contiene la sumatoria de las multiplicaciones de los vectores que se originan en los centroides y llegan a cada punto del conjunto respectivo. Una vez obtenida la matriz de rotación se despeja el vector de translación del sistema de ecuaciones. Los pasos del algoritmo y las definiciones se encuentran detalladas en el apéndice A.5.1.

2.14. Búsqueda Local (Local Search)

Búsqueda local (BL) es una técnica utilizada para resolver problemas de optimización combinatoria. Se empieza con una solución inicial s al problema, esta solución puede ser generada de forma aleatoria o mediante algún otro método. Se busca mejorar s haciendo pequeños cambios en la misma, es decir, eliminando, reordenando o reagrupando elementos de s. Si se logra mejorar la solución s, se obtiene una nueva solución s' sobre la cual se continua el mismo proceso hasta que no se pueda mejorar más, como se describe en [1].

Existe una amplia gama de algoritmos de BL que van desde los más sencillos como mejoramiento iterativo (*iterative improvement*), también conocido como *Hill Climbing* (ver sección B.3 del apéndice B), hasta métodos más complejos como algoritmos evolutivos o recocido simulado (*simulated Annealing*).

2.15. Algoritmo Genético

El algoritmo genético es una meta-heurística perteneciente a la familia de algoritmos evolutivos que "están inspirados en las capacidades que tiene la naturaleza de evolucionar seres vivientes bien adaptados a su entorno" [1]. El algoritmo busca imitar el proceso evolutivo y es usado para problemas de búsqueda y optimización.

El algoritmo genético trabaja sobre una población de individuos, a los cuales se les aplican operadores de cruce y mutación para obtener otros nuevos que formarán una nueva población. A cada una de las iteraciones del algoritmo donde se forman nuevas poblaciones se les conoce como "generaciones". En cada iteración se realiza un proceso de **selección** que determina cuales son los individuos que pasarán a la próxima generación. El algoritmo termina cuando se llega hasta un número fijo de generaciones o se cumple alguna condición de parada. Una explicación más detallada del algoritmo se puede encontrar en la sección B.4 del apéndice B.

Capítulo 3

Diseño e implementación del clasificador y propuesta de esqueleto

El problema a tratar es el referente al reconocimiento de poses humanas en tiempo real con imágenes de profundidad y la formación de un esqueleto 3D. Para ello, se empleó una aproximación que consiste en reconocer las diferentes partes del cuerpo mediante una clasificación por píxeles realizada en tiempo real. Para la resolución del problema se implementó el algoritmo de Bosques Aleatorios (RF) como clasificador de píxeles de una imagen. La implementación se basa en el paper emitido por Microsoft sobre la técnica utilizada para reconocimiento de pose con el sensor Kinect [25].

Una vez clasificadas las partes del cuerpo, es necesario generar propuestas para las articulaciones del esqueleto. Para la resolución de este problema, se implementó el algoritmo *Mean shift* para aproximar un punto en cada parte del cuerpo que funcionará como articulación.

En esta sección se explican todos los aspectos referentes a la implementación del algoritmo RF y el procedimiento aplicado para generar la propuesta de esqueleto. En un principio, se da una descripción del tipo de datos sobre el cual operará el algoritmo RF, así como las modificaciones, estrategias y parámetros utilizados para dicho algoritmo. Se explicarán las etapas de pre y post - procesamiento aplicados para el algoritmo *Mean - shift*, así como los parámetros utilizados para el mismo.

3.1. Imágenes de profundidad

El algoritmo trabaja con imágenes de profundidad generadas por un sensor *Kinect*. Los píxeles en una imagen de profundidad poseen valores que indican la distancia entre la cámara y un punto en la escena expresada en milímetros(mm).

Como no se necesita información temporal para que el clasificador funcione, los datos utilizados consisten en imágenes de profundidad de diferentes poses de una persona. La ventaja de utilizar datos de profundidad es que no son susceptibles a cambios en intensidad de luz ni de colores en la escena.

3.2. Datos de entrenamiento

Como datos de entrenamiento se utilizaron imágenes de profundidad etiquetadas con cada una de las partes del cuerpo que se buscan clasificar. Como el cuerpo humano posee un rango muy alto de poses que puede realizar, se complica la obtención de imágenes de entrenamiento y etiquetamiento de las mismas. Por esta razón, se automatizó el proceso haciendo uso de ejemplos de la base de datos de captura de movimiento de la Universidad de Carnegie Mellon (CMU) [30] para generar una gran cantidad de imágenes sintéticas para el entrenamiento. No obstante, también se utilizaron imágenes generadas por el sensor Kinect etiquetadas manualmente para el entrenamiento. Se espera, además, que el clasificador pueda generalizar bien ante poses que no haya visto. Se asume sustracción de fondo para cada uno de los tipos de imágenes de entrenamiento. En el apéndice C.1.1 se encuentra una descripción más detallada de los datos utilizados para el entrenamiento y su construcción.

3.2.1. Codificación de las imágenes para entrenamiento

La información de profundidad y de etiquetamiento es almacenada en los formatos **EXR** y **BMP** respectivamente. Ambas imágenes poseen un tamaño de 640×480 píxeles. Sin embargo, como no es necesario almacenar la información del fondo de la imagen, se puede representar en forma de una matriz dispersa.

Debido a que ambas imágenes son generadas desde el mismo punto de vista, existe una correspondencia implícita entre las coordenadas de cada píxel en ambas imágenes. Por esta razón, ambas imágenes pueden codificarse como una matriz dispersa donde los píxeles correspondientes al modelo se codifican en dos vectores: uno correspondiente a la profundidad y otro correspondiente a la intensidad. Los vectores de indexación corresponden a la definición de matriz dispersa en 2.8.

3.2.2. Etiquetamiento de imágenes

La información de las etiquetas se encuentra en imágenes de 6 tonos de escala de grises correspondiente a las 6 partes del cuerpo que se quieren clasificar: Cabeza, Brazo derecho, Brazo izquierdo, Torso, Pierna izquierda, Pierna derecha.

3.3. Implementación de los árboles aleatorios

En esta sección se describirán los detalles respectivos a las estructuras y los procedimientos utilizados en la implementación de los árboles.

3.3.1. Función de atributo

Para la clasificación de los píxeles de la imagen de profundidad es necesario definir una medida numérica que permita diferenciar de alguna manera a qué parte del cuerpo pertenece

un píxel determinado. Se empleó una comparación por los atributos calculados dada la profundidad como en [25]. Dado un píxel $\mathbf{x}$, se calcula su función de atributos como:

$$f_\theta(I, \mathbf{x}) = d_I(\mathbf{x} + \frac{\mathbf{u}}{d_I(\mathbf{x})}) - d_I(\mathbf{x} + \frac{\mathbf{v}}{d_I(\mathbf{x})}) \tag{3.1}$$

Donde $d_I(\mathbf{x})$ es la profundidad del píxel $\mathbf{x}$ en la imagen I y los parámetros $\theta = (\mathbf{u}, \mathbf{v})$ son desplazamientos que se le aplican al píxel $\mathbf{x}$ para reposicionarlo en otra parte de la imagen. En la ecuación anterior se realiza una normalización de los desplazamientos, de esa manera se asegura invariancia con respecto a la profundidad. Si al sumar el desplazamiento al píxel $\mathbf{x}$ se obtiene una posición que yace en el fondo o fuera de los bordes de la imagen se le asigna una profundidad constante muy grande.

3.3.2. Estructura de los nodos

Como se explicó en la sección 2.10, los RF están conformados por nodos internos y nodos hojas, cuya estructura será descrita a continuación.

Estructura de los nodos internos

Cada nodo interno del árbol almacena un par de desplazamientos $\theta = (u, v)$ y un valor de umbral o *Threshold* τ aprendidos después del entrenamiento.

Estructura de los nodos hoja

Cada nodo hoja de un árbol t posee una distribución de probabilidades $P_t(c|I, \mathbf{x})$ aprendida durante el entrenamiento del árbol, donde c es una etiqueta que representa alguna parte del cuerpo e I es la imagen a la que pertenece el píxel $\mathbf{x}$. Cada nodo hoja posee una probabilidad por cada etiqueta c referente a la clasificación del píxel $\mathbf{x}$ en esa hoja.

3.3.3. Clasificación de un píxel

Para clasificar un píxel $\mathbf{x}$, por cada árbol t se empieza desde la raíz, y por cada nodo interno se calcula la función de atributo f_θ y se envía el píxel a la derecha o a la izquierda del árbol dependiendo de la comparación dada con el umbral τ. Al llegar a un nodo hoja se tiene una distribución de probabilidades referente a la parte del cuerpo a la que pueda pertenecer el píxel $\mathbf{x}$ en cada árbol t. Para conocer la clasificación del píxel $\mathbf{x}$ se promedian las distribuciones de probabilidades de la siguiente forma:

$$P(c|I,x) = \frac{1}{T}\sum_{t=1}^{T} P_t(c|I,x) \tag{3.2}$$

Y se clasifica $\mathbf{x}$ con la etiqueta c cuya probabilidad sea la mayor de todas.

3.3.4. Entrenamiento del árbol

Cada árbol entrena utilizando un conjunto de imágenes diferente pero de igual tamaño. De cada imagen se seleccionan de forma aleatoria n píxeles que conformarán el conjunto de entrenamiento. Cada árbol se entrena siguiendo el siguiente algoritmo [25]:

1. Por cada nodo interno:

   a) Se generan de forma aleatoria un conjunto de candidatos de separación $\phi = (\theta, \tau)$ correspondientes a los parámetros $\theta = (\mathbf{u}, \mathbf{v})$ y al umbral τ.

   b) Se particiona el conjunto de entrenamiento $Q = (I, \mathbf{x})$ en subconjuntos izquierdo y derecho según cada ϕ:

$$Q_{izq}(\phi) = \{(I,\mathbf{x}) | f_\theta(I,\mathbf{x}) < \tau\} \tag{3.3}$$

$$Q_{der}(\phi) = Q \setminus Q_{izq}(\phi) \tag{3.4}$$

c) Se toma el ϕ que provea la mayor ganancia de información de acuerdo a lo siguiente:

$$\phi^* = \underset{\phi}{\operatorname{argmax}}\, G(\phi) \tag{3.5}$$

$$G(\phi) = H(Q) - \sum_{s \in \{izq, der\}} \frac{|Q_s(\phi)|}{|Q|} H(Q_s(\phi)) \tag{3.6}$$

Donde $H(Q)$ es la entropía de Shanon para el conjunto de entrenamiento Q.

2. Si se tiene que la ganancia de información $G(\phi^*)$ es suficiente, y la profundidad del árbol no ha llegado a un máximo, se hace recursión por los subconjuntos izquierdo y derecho $Q_{izq}(\phi^*)$ y $Q_{der}(\phi^*)$

3. Al llegar a un nodo hoja se calcula la distribución de probabilidades con los ejemplos de entrenamiento que llegaron a esa hoja.

3.4. Paralelización del entrenamiento con MPI

El entrenamiento de cada árbol de decisión puede ser paralelizado fácilmente en múltiples computadoras. La librería *OpenMPI* [11] para **C++** implementa todas las facilidades del estándar *Message Passing Interface* para el cómputo en paralelo de programas.

En el algoritmo de RF que se utilizó en el presente trabajo (vea la sección 3.3) el mayor tiempo de cómputo se encuentra en la generación de nodos. Esto es debido a la necesidad de generar una cantidad total de candidatos de separación equivalente al número de desplazamientos multiplicado por el número de umbrales. Este conjunto total de candidatos de separación debe ser probado uno a uno sobre el conjunto de píxeles de entrenamiento.

Esta parte es la que se decidió paralelizar en el presente trabajo. Se tomó la cantidad total de candidatos de separación y se dividió entre el número de procesos creados por la librería

OpenMPI. El número de procesos puede variar según la cantidad y el tipo de máquinas que se tengan disponibles. En este caso se decidió tener un proceso por máquina.

Cada proceso se encarga de generar y probar una fracción del número total de candidatos de separación sobre los píxeles de entrenamiento. Una vez probados, cada proceso envía el mejor candidato de separación generado a un proceso especial que se encarga de la construcción del árbol. Este último toma el mejor candidato y crea el nodo. Seguidamente, todos los procesos empiezan de nuevo a generar los candidatos de separación para el siguiente nodo y así continuar la construcción del árbol.

Al principio, un único proceso selecciona aleatoriamente todos los píxeles que conforman el conjunto de entrenamiento y los transmite al resto de los procesos. A partir de ahí, cada proceso puede trabajar independientemente con su propia copia del conjunto de entrenamiento. Una vez que el árbol esté totalmente construido, se puede generar otro árbol o simplemente el proceso encargado de la construcción de los árboles les da una señal al resto de los procesos para que puedan terminar.

3.5. Pre-procesamiento de imágenes en tiempo real

3.5.1. Sustracción de fondo

Para la aplicación del clasificador explicado en la sección 3.3, se aplica sustracción de fondo en tiempo real. Esto con el fin de que el clasificador sea aplicado en los píxeles de interés de la imagen, correspondientes a la persona en la escena. Se asumió que el fondo era estático y la aplicación se desarrollaría en un ambiente cerrado. Para realizar sustracción de fondo se aplicó la técnica explicada en la subsección 2.3.1, que puede manejar cambios de iluminación y se adapta a la presencia nuevos elementos en la escena.

Para sustraer el fondo se utilizó la clase de OpenCV *BackgroundSubtractorMOG2*, la cual

aplica el algoritmo descrito en [34] para realizar dicha tarea y así obtener una máscara que define los elementos pertenecientes al primer plano de la imagen.

Una vez removido el fondo, en la máscara permanece aún un poco de ruido, es decir, elementos que no pertenecen al primer plano de la imagen. Para solventar este problema, se aplica la técnica de erosión (descrita en la sección 2.4) sobre la máscara, de manera que se elimine el ruido de la misma. Fueron necesarias solo dos iteraciones del algoritmo de erosión para eliminar los remanentes de ruido en la máscara.

Ya obtenida una máscara libre de ruido, es necesario definir las componentes conexas que conformarán los elementos pertenecientes al primer plano de la imagen. Para ello, se utilizó la función de OpenCV *findContours* que se basa en el algoritmo descrito en la sección 2.5, específicamente con modo "*CV_RETR_CCOMP*" que indica que retornará las componentes organizándolas en niveles de jerarquías desde la más externa a la más interna; y método de aproximación "*CV_CHAIN_APPROX_SIMPLE*" que indica compresión en los segmentos verticales, horizontales y diagonales. Una vez agrupadas las componentes se toman sólo aquellas que tengan un área mayor a un valor umbral fijado de 6500 píxeles. Este valor fue fijado garantizando que se eliminarán aquellas pequeñas componentes no correspondientes a una persona.

Finalmente, para refinar la solución obtenida con las componentes conexas, se realiza una aproximación poligonal para suavizar los contornos como se describe en la sección 2.6. Para ello, se utilizó la función de OpenCV *approxPolyDP* con valor de precisión de aproximación 2 y aproximación de curva cerrada.

3.5.2. Compresión de imágenes

Para que el algoritmo pueda operar de forma óptima en tiempo real, es necesario reducir la carga de procesamiento correspondiente a aplicar el clasificador sobre todos los píxeles

de la persona en la imagen. Para ello se propone un método de compresión que consiste en clasificar sólo algunos píxeles correspondientes a la persona en vez de clasificarlos todos. Para ello, se fijan saltos de n píxeles en la imagen, es decir, para cada píxel, el próximo vecino con información será aquel que se encuentra a n píxeles de distancia.

A partir de esto se obtiene una imagen de la persona más segmentada, la cual es comprimida uniendo los píxeles que poseen información, obteniendo así una imagen de menor dimensión que es más rápida de procesar. Se fijo como $n = 3$ como valor de salto de píxeles ya que mantiene un equilibrio entre la pérdida de información (por píxeles no clasificados) y disminución en tiempo de procesamiento.

En la figura 3.1 se muestra la imagen sin compresión, y en la figura 3.2 se muestra como se ve la imagen luego de aplicar la clasificación con compresión sin unir los píxeles.

Figura 3.1: Imagen clasificada sin compresión.

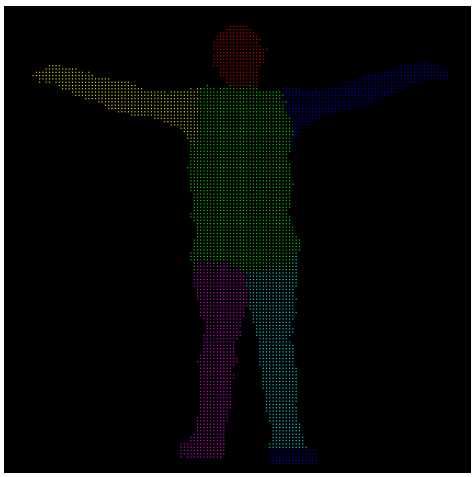

Figura 3.2: Imagen clasificada con compresión

3.5.3. Flood fill

Una vez clasificada la imagen, cada píxel posee un porcentaje de clasificación de acuerdo al color o etiqueta asignada. Es posible que ocurra el caso donde se puedan observar

componentes mal clasificadas ubicadas en diferentes partes del cuerpo. Estas componentes pueden dañar la estimación de las articulaciones del esqueleto y constituyen datos irregulares o *outliers* en la clasificación. Para reparar la imagen de estas componentes mal clasificadas, se emplea una técnica conocida como ***Flood fill***.

Básicamente, *Flood fill* realiza un *BFS* a través de los píxeles de la imagen formando componentes conexas de acuerdo a los colores en la misma, correspondientes a cada parte del cuerpo. Se tiene como objetivo identificar aquellas pequeñas componentes mal clasificadas y colorearlas de acuerdo con la información de los píxeles vecinos. Una vez que se identifica una componente, se compara su tamaño respecto a un valor umbral, si su tamaño no es muy grande, se procede a colorear la componente del color correspondiente a la información provista por sus píxeles vecinos. A los píxeles que conforman la componente coloreada con el nuevo color se les asigna un porcentaje de clasificación igual al 100 % para ese color y un porcentaje de clasificación igual al 0 % para el color anterior.

Como valor de umbral aprendido para el tamaño de las componentes se fijo 10 píxeles. Es importante mencionar que *Flood fill* opera sobre la imagen comprimida la cual posee un tamaño más pequeño que la original.

En la figura 3.3 corresponde a una imagen clasificada, se puede observar claramente que existen componentes mal clasificadas. En la figura 3.4 se observa la misma imagen reparada con *flood fill*.

3.6. Definición de las articulaciones del esqueleto

Un esqueleto en el área de visión por computadora y captura de movimiento puede verse como un conjunto de puntos en el espacio 3D que representan las articulaciones de la persona. A partir de la información inferida sobre el reconocimiento de las partes del cuerpo es necesario entonces generar posibles puntos candidatos para dichas articulaciones.

Figura 3.3: Imagen clasificada sin *flood fill*.

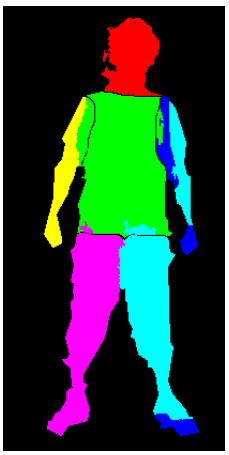

Figura 3.4: Imagen clasificada con *flood fill*.

Para ello, se utiliza el algoritmo *mean-shift* (descrito en la sección 2.7) sobre el conjunto de puntos clasificados de la imagen. El algoritmo es utilizado para estimar los puntos correspondientes a las propuestas finales de las articulaciones de cada parte del cuerpo. *Mean -shift* utiliza las probabilidades de clasificación de los píxeles y desplaza la ventana hacia el punto donde se tengan los mayores porcentajes de clasificación. Con este algoritmo es posible ignorar aquellos puntos fuera de lugar o *outliers* que puedan dañar la estimación final de la articulación.

El algoritmo es aplicado en cada parte del cuerpo. Como posición inicial para la ventana del algoritmo, se utiliza el promedio de los puntos cuyo porcentaje de clasificación sea mayor que un valor umbral aprendido τ, que se fijo a un valor de 0,95. Con ello se garantiza que el algoritmo empiece en una zona donde pueda converger a una buena solución. En la figura 3.5 se observa la solución del *mean-shift* aplicada a una imagen.

Una vez obtenidos los puntos de la imagen correspondientes a cada articulación, con sus respectivos valores de profundidad, es necesario obtener la posición en el espacio 3D de los mismos para obtener los puntos finales del esqueleto. Para ello se realiza la transformación explicada en la sección 2.11.2, a partir de la cual se obtienen las coordenadas 3D en espacio

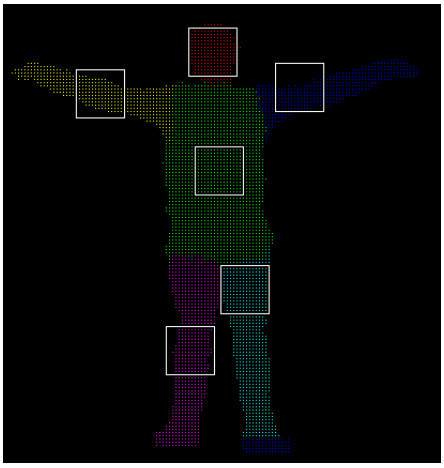

Figura 3.5: *Mean - shift* sobre imagen clasificada.

euclidiano de las articulaciones del esqueleto. Los valores estimados de la distancia focal (f) y el centro de la imagen (o) fueron tomados de los resultados de la calibración del *kinect* como se describe en [7]. Donde se estimaron los valores de f y o para la cámara de profundidad del *kinect* como se encuentran descritos en la sección A.2 del apéndice A.

En la figura 3.6 se muestra el esqueleto de la persona sobre la imagen clasificada antes de aplicar la transformación, mientras que en la figura 3.7 se muestra el mismo esqueleto transformado al mundo real en un espacio 3D.

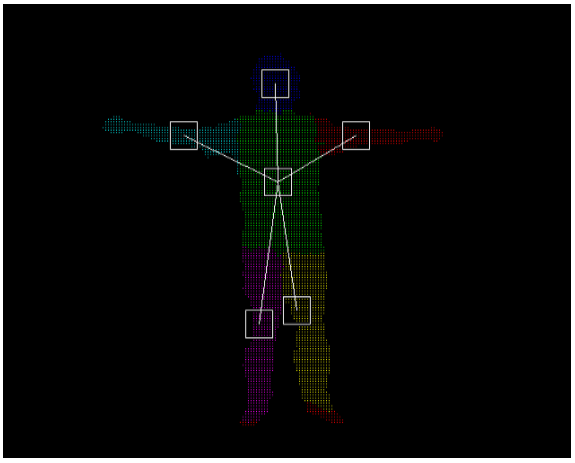

Figura 3.6: Esqueleto sin transformación sobre imagen clasificada.

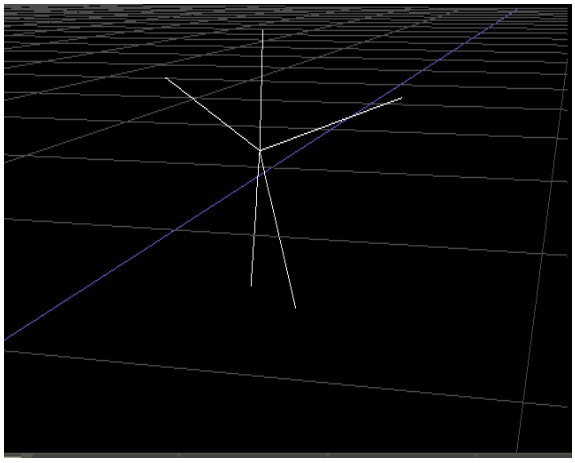

Figura 3.7: Esqueleto transformado sobre espacio 3D.

Capítulo 4

Diseño e implementación de integración de múltiples vistas

En el capítulo anterior, se explicó el método utilizado para detectar un individuo dada la información generada por un sensor Kinect. En este capítulo, se describirá el método utilizado para relacionar y complementar esa información utilizando varios sensores simultáneamente. Teniendo múltiples puntos de vista, se puede mejorar la estimación de un punto de interés descartando valores incorrectos o completarlos si se presenta una oclusión en una de las vistas.

La posición de los puntos de interés viene dada por un conjunto de vectores que componen un esqueleto estimado de la pose del individuo. Existe un conjunto de estos vectores por cada sensor. La única suposición que se tomó es que los conjuntos de puntos obtenidos corresponden al mismo individuo en la escena.

En un principio, se describe el procedimiento utilizado para unir las perspectivas de varios sensores, se explican las estrategias empleadas, se describen los métodos utilizados para refinar la solución dada por la calibración y finalmente como la información de cada sensor es utilizada para formar el esqueleto resultante.

4.1. Cálculo de la transformación rígida

El sistema de coordenadas que genera el sensor viene dado por la figura 4.1. Las coordenadas de los esqueletos están representadas en el sistema de coordenadas correspondiente al sensor que las generó. El origen de coordenadas viene dado por el centro de la cámara del

kinect.

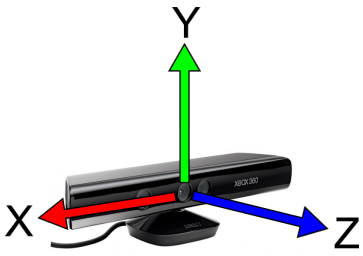

Figura 4.1: Sistema de coordenadas generado por el sensor Kinect.

Para llevar todos los puntos obtenidos por un Kinect a un mismo sistema de coordenadas, se debe aplicar una transformación rígida (ver apéndice A.1.2). De esta manera se logra unir la perspectiva de cada sensor en una. Para hacer esto, se debe escoger el sistema base al cual será destinado todas las transformaciones de los puntos. En la presente implementación, se eligió arbitrariamente el sistema dado por alguno de los sensores, que será considerado como sistema base. Posteriormente se debe hallar una matriz de transformación entre el sistema base y cada sistema adicional.

A continuación se presenta el formato utilizado para representar una transformación rígida. Posteriormente se describen los dos métodos utilizados.

4.1.1. Representación de la transformación rígida

Por fines prácticos, se decidió integrar la matriz de rotación R con el vector de translación T dentro de una misma matriz perteneciente a $\mathbb{R}^{4\times4}$, de la siguiente forma:

$$\begin{pmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.1}$$

Donde los elemento $r_{i,j}$ corresponden a los componentes de la matriz de rotación R y los elementos t_i corresponden a los componentes del vector T.

Se utilizó esta representación por compatibilidad con las librerías *Eigen* [12] y *Point Cloud Library* [24] utilizadas en el presente proyecto.

4.1.2. Método de calibración estéreo

Para este método utilizamos el software ya implementado por Nicolas Burrus, Kinect RGBDemo [6]. Para calcular la matriz de transformación entre dos sensores, se deben colocar de forma tal que haya una intersección del campo de visión de ambos. Esto es a razón de que deben ser tomadas al rededor de 30 imágenes diferentes desde ambas cámaras de un patrón de calibración como se muestra en las figuras 4.2 y 4.3.

Figura 4.2: Ejemplo de imagen tomada desde el primer Kinect para el RGBDemo

Figura 4.3: Ejemplo de imagen tomada desde el segundo Kinect para el RGBDemo

Después de haber tomado las fotos, la matriz de transformación es calculada en base a los puntos del patrón de calibración detectados como se mostró en la sección 2.12.

La principal ventaja de este método es que la matriz obtenida es bastante precisa, además de ofrecer también una corrección de los parámetros intrínsecos de la cámara (descritos en la sección 2.11.2). Sin embargo, este método debe repetirse siempre que los sensores hayan cambiado de posición. Con esto, viene su principal desventaja, que es la cantidad de tiempo

necesaria para tomar todas las fotografías diferentes para volver a calibrar.

4.1.3. Método de descomposición aplicado sobre ambos esqueletos

Para hallar la matriz se utilizaron los puntos estimados de los esqueletos por cada sensor. Como se conoce la correspondencia de los puntos, se puede aplicar directamente sobre ellos el método descrito en la sección 2.13.

A diferencia del método anterior, la ventaja de éste es que no requiere tomar fotos con un patrón de calibración, sino utilizar los puntos estimados del esqueleto de la persona para encontrar la correspondencia. Sin embargo, como son puntos estimados, la matriz obtenida no es tan precisa y necesita ser refinada. Al igual que el método anterior, este debe repetirse cada vez que los los sensores se hayan movido de posición.

Para este método se utilizó la librería *Point Cloud Library* [24] para el cálculo de la matriz. Esta librería ofrece una gran variedad de algoritmos para trabajar con nubes de puntos, entre los cuales se encuentra implementado el método de descomposición **SVD**.

La función que utiliza SVD se encuentra como *estimateRigidTransformation* y pertenece a la clase *pcl::registration::TransformationEstimationSVD*. Para utilizar esta función una instancia de la clase debe ser creada proporcionando la información del par de nubes de puntos de las cuales se desea extraer la transformación. Las nubes llevan el nombre de "fuente" y "destino", lo que significa que según el orden en que se especifiquen, la transformación resultante solo podrá ser aplicada a la nube de puntos que se especificó como "fuente".

De igual manera, para aplicar la matriz de transformación a una nube de puntos, se empleó la función *pcl::transformPointCloud*. Esta función recibe una nube de puntos y una matriz de transformación para obtener la nube de puntos transformada.

La matrices obtenidas por este método son denominadas soluciones y son refinadas por los algoritmos que serán explicados en las siguientes secciones.

4.1.4. Justificación del método empleado

Para este caso particular se escogió el método descrito en la sección 4.1.3. La principal razón es la búsqueda de flexibilidad en cuanto al posicionamiento de los sensores. Si se hiciera uso del primer método, al cambiar la posición de alguno de los sensores, es necesario repetir todo el proceso de calibración estéreo que implica tomar un nuevo conjunto de fotos del patrón de calibración. El segundo método ofrece una alternativa simple que no requiere de mucho tiempo ni el uso del patrón de calibración.

4.2. Estrategias para mejorar el resultado final

En esta sección se describen los métodos empleados para mejorar la calibración y la propuesta final del esqueleto de la persona. A continuación se detallan las estrategias utilizadas para refinar la solución al problema de múltiples Kinects. Primero se define el error utilizado para estimar la bondad de una solución y luego se describen cada uno de los algoritmos empleados. Finalmente se mencionan las diferentes acciones que se consideraron producir un esqueleto resultante en base a la información obtenida por cada sensor.

4.2.1. Error asociado a una solución

Para evaluar la bondad de una solución representada por una matriz de transformación, se hará uso de un conjunto de prueba conformado por n cuadros tomados por los sensores. Sobre este conjunto se aplicará la matriz para superponer los puntos dados por cada uno de los sistemas y calcular el promedio de las diferencias de las distancias entre ambos esqueletos. La medida de bondad viene dada por el promedio de los errores calculados por cada cuadro.

Específicamente, sea n la cantidad de cuadros del conjunto de prueba y m la cantidad de puntos del esqueleto, se define $a_j^{(i)}$ como el j-ésimo punto del primer sistema en el i-ésimo

cuadro y $a_j^{\prime(i)}$ el j-ésimo punto del segundo sistema el i-ésimo cuadro, la función objetiva viene dada por:

$$\frac{1}{n}\sum_{i=1}^{n}(\frac{1}{m}\sum_{j=1}^{m}||a_j^{(i)} - a_j^{\prime(i)}||) \tag{4.2}$$

El resultado de la ecuación 4.2 se utiliza como término de error asociado a una matriz. En todos los algoritmos utilizados, el objetivo es producir una matriz con el menor error asociado posible.

4.2.2. Transformación rígida entre puntos acumulados

Cada cuadro representa un instante de tiempo en el cual se genera un esqueleto de la persona por cada sensor. El objetivo es obtener la matriz de transformación rígida mediante la aplicación del método de descomposición de valor singular (como se describe en la sección 2.13) sobre un par de nubes de puntos acumuladas a partir de la información del par de esqueletos por un número fijo de cuadros.

4.2.3. Promedio de matrices de transformación rígida

Este método consiste en obtener la matriz promedio de las matrices de transformación rígida generadas a partir de la aplicación del método de descomposición de valor singular por un número fijo de cuadros. Este método posee la desventaja de perder la ortogonalidad de la matriz de rotación asociada a la transformación. Como se muestra en la sección A.1.1 del apéndice, las matrices de rotación tienen la propiedad de ser ortogonales y la suma de matrices ortogonales no garantiza la ortogonalidad. Sin embargo, se trata de un método de estimación muy rápido y la pérdida de ortogonalidad es casi imperceptible a simple vista.

4.2.4. Mejoramiento iterativo (Búsqueda local)

Se implementó el algoritmo de mejoramiento iterativo *Best-first* sobre un conjunto de matrices de transformación correspondientes a posibles soluciones al problema obtenidas a partir de un número fijo de cuadros calibrados. A continuación se explican los parámetros y detalles del algoritmo.

Función objetivo

Como función objetivo para este algoritmo se buscó minimizar la ecuación 4.2 que representa el error asociado a una matriz.

Generación de vecindad

La función de generación de vecindades consiste en la aplicación de dos operadores, una traslación y una rotación a la matriz de transformación rígida. La función de vecindad genera 12 matrices de transformación de las cuales 6 de ellas corresponden a la aplicación del operador de rotación y las otras 6 a la aplicación del operador de traslación. A continuación se explican con detalle:

- **Rotación:** La rotación de una matriz de transformación M puede realizarse en los ejes **X**, **Y** y **Z**. Dado un ángulo muy pequeño α, para cada eje se construye una matriz de rotación R representando una rotación de α grados y una R' representando una rotación de $-\alpha$ grados, lo que da un total de 6 matrices de rotación. Cada matriz de rotación se utiliza para generar una nueva solución a partir de la matriz de transformación mediante la multiplicación de matrices MR y MR'.

- **Traslación:** El operador de traslación consiste en sumar un valor fijo c a alguno de los ejes representados por el vector de traslación asociado a la matriz de transformación

descrita en la sección 4.1.1. Para este operador se generan 6 nuevas matrices, 2 matrices por cada eje de la siguiente forma:

$$R_x = \begin{pmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \pm c \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.3}$$

$$R_y = \begin{pmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \pm c \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.4}$$

$$R_z = \begin{pmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \pm c \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.5}$$

El conjunto de 12 matrices resultante es lo que se define como la vecindad de la solución origina. Entre las nuevas matrices puede haber alguna que produzca una mejor transformación debido al pequeño ajuste realizado. A medida que se vayan produciendo nuevas vecindades, el ángulo α y el valor c pueden ser ajustados para producir cambios más ligeros que produzcan matrices más precisas.

4.2.5. Algoritmo genético

Se implementó el algoritmo genético con reemplazo de estado estable descrito en la sección B.4.5 del apéndice B, para ello se utilizó la librería GAlib [33] implementada en **C++**. A

continuación se explican los detalles asociados a la implementación del algoritmo genético.

Representación de soluciones

Cada cromosoma del algoritmo genético consiste en una matriz de transformación rígida que corresponde a una posible solución del problema. La matriz es codificada en forma de vector concatenando las filas de la misma de la siguiente manera:

Matriz de transformación

$$\begin{pmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Cromosoma

$r_{1,1}$	$r_{1,2}$	$r_{1,3}$	t_1	$r_{2,1}$	$r_{2,2}$	$r_{2,3}$	t_2	$r_{3,1}$	$r_{3,2}$	$r_{3,3}$	t_3

Inicialización de la población

La población inicial la constituye un conjunto de matrices de transformación obtenidas a partir de un número fijo de cuadros con esqueleto calibrado.

Función objetivo

Como función objetivo para este algoritmo se buscó minimizar la ecuación 4.2 que representa el error asociado a una matriz.

Función de comparación

La función de comparación especifica que tan diferente es una solución de otra. Para comparar dos cromosomas se tomó el valor absoluto de la diferencia de los elementos que conforman ambos cromosomas. Sea $A = a_0, a_1, \ldots, a_n$ y $B = b_0, b_1, \cdots, b_n$ dos cromosomas, su función de comparación se define como:

$$f_{AB} = \sum_{i=0}^{n} |a_i - b_i| \tag{4.6}$$

Operador de cruce

El operador de cruce utilizado fue el promedio de los elementos contenidos en los cromosomas que representan dos soluciones dadas. Como se explicó anteriormente, una solución al problema corresponde a una transformación rígida A.1.2. El promedio no conserva la ortogonalidad de la matriz de rotación asociada a la solución, sin embargo, la pérdida de ortogonalidad es casi imperceptible y el método de algoritmo genético a diferencia de 4.2.3 penaliza soluciones de muy bajo nivel de adaptabilidad, por lo que se consideró utilizar el promedio a pesar de la pérdida de ortogonalidad.

A continuación, se ejemplifica la forma en que se aplica este operador de cruce a dos cromosomas.

$$A = \boxed{a_0}\boxed{a_1}\boxed{\cdots}\boxed{a_n} \qquad B = \boxed{b_0}\boxed{b_1}\boxed{\cdots}\boxed{b_n}$$

$$C = \boxed{\tfrac{a_0+b_0}{2}}\boxed{\tfrac{a_1+b_1}{2}}\boxed{\cdots}\boxed{\tfrac{a_n+b_n}{2}}$$

4.2.6. Construcción del esqueleto final

A continuación se explican las acciones que se tomaron para estimar un esqueleto final de la persona en base a la información obtenida por cada sensor Kinect. Para esto se aprovechó el hecho de tener múltiples sensores para descartar posibles errores en la estimación de cada articulación y completar la información en caso de oclusión.

Cada sensor Kinect puede proponer un punto para cada tipo de articulación, lo que significa que por cada articulación se tendrá un número de puntos propuestos menor o igual al número de sensores que se este utilizando. Si la transformación obtenida por los métodos explicados en las secciones anteriores es buena, se puede esperar que los puntos estimados para cada articulación se encuentren relativamente cercanos.

Es por esta razón que la primera acción que se tomó fue eliminar aquellos puntos propuestos para cada articulación cuya distancia con los demás puntos del mismo tipo supera un cierto umbral. Estos puntos descartados suponen errores en la estimación por parte de alguno de los sensores. Hay que señalar que no se puede estar completamente seguro de que un punto propuesto sea erróneo aparte del hecho de que éste difiera bastante en comparación con el resto de los puntos. Mientras más propuestas haya por articulación, con mayor probabilidad se descartaran verdaderos errores al comparar con la mayoría.

En caso de que el número de puntos por articulación no sea suficiente para obtener una mayoría y descartar errores, se confía en estos puntos y son propuestos como la posición final para la articulación. Nótese que este es el caso de las oclusiones, donde uno o varios puntos pueden estar ocultos para un sensor pero visibles para otro.

Finalmente, cada conjunto de puntos propuestos correspondiente a cada tipo de articulación es promediado y el resultado será la posición final de dicha articulación. Todos los puntos resultantes conforman el esqueleto final para la persona.

Capítulo 5

Experimentos y resultados

Este capítulo se divide en dos secciones principales: La primera referente a los experimentos y resultados obtenidos con el algoritmo RF (detallado en el capítulo 3) utilizando datos sintéticos y datos reales, y los experimentos y resultados obtenidos a partir de unir las vistas de múltiples sensores *Kinect* como se describe en el capítulo 4.

Para el entrenamiento del algoritmo se utilizaron 13 computadoras cuyas características aparecen descritas en el cuadro 5.1.

CPU	Intel Core2 Quad 2.83GHz
RAM	3 GB
Distribución	Debian 5.0 x86
Kernel Linux	2.6.30
Versión gcc	4.3.2

Cuadro 5.1: Computador usado para el entrenamiento.

La ejecución del algoritmo RF y las pruebas referentes a la integración de múltiples vistas, se llevaron a cabo en el computador cuyas características se encuentran en el cuadro 5.2.

CPU	Intel Core2 Duo 1.83GHz
RAM	3 GB
Distribución	Ubuntu 10.04 x86
Kernel Linux	2.6.32.33
Versión gcc	4.4.3

Cuadro 5.2: Computador usado para el procesamiento de las imágenes de los sensores.

5.1. Algoritmo RF

Para el entrenamiento del algoritmo se utilizó un conjunto de imágenes sintéticas y otro de imágenes reales descritas con mayor detalle en la sección C.1.1 del apéndice C. Los objetivos principales del conjunto de pruebas realizado en esta sección son: Medir el porcentaje de clasificación del algoritmo con imágenes sintéticas y reales, comparar los resultados obtenidos y determinar qué tipo de imagen aporta mayor información en el entrenamiento. Para ello, se realizan varias corridas del algoritmo variando algunos parámetros de interés explicados más adelante, con el fin de obtener el conjunto de ellos que influya de mejor manera en el desempeño de dicho algoritmo. Para comparar los resultados se utilizó el porcentaje de clasificación sobre conjuntos de entrenamiento y de prueba de ambos tipos de imágenes.

5.1.1. Descripción general de los experimentos

Para realizar los experimentos se tomó un conjunto de entrenamiento y uno de prueba conformado por imágenes de cada tipo (reales y sintéticas) para probar la clasificación del algoritmo. A partir del conjunto de árboles resultantes para cada caso, se realizó la clasificación de imágenes de entrenamiento e imágenes del conjunto de prueba. La clasificación consiste en comparar si un píxel de la imagen, clasificado con el algoritmo, corresponde a la clasificación dada de la imagen de entrenamiento o del conjunto de prueba.

Se realizaron cuatro entrenamientos, tanto para imágenes reales como sintéticas. Para las imágenes sintéticas se utilizó un modelo de una persona caminando y para las reales se utilizaron dos individuos de prueba como se describe en la sección C.1.1 del apéndice C. Los tres primeros, fueron entrenamientos que consistían en lo siguiente: El primero se realizó utilizando un conjunto de entrenamiento conformado únicamente por imágenes de los individuos vistos de frente. El segundo se realizó con un conjunto formado por imágenes de

los individuos vistos de lado izquierdo y el tercero con imágenes de los individuos vistos de lado derecho. El cuarto entrenamiento se realizó utilizando imágenes vistas a 360°.

Como conjunto de imágenes sintéticas de entrenamiento se utilizó aquel descrito en C.1.1. Para el experimento de 360° se utilizó el conjunto completo. Para los conjuntos derecho, izquierdo y de frente, se toman de las 128 imágenes, aquellas que correspondan con la perspectiva que denomina el conjunto, correspondientes a 18 imágenes para cada uno. Como conjunto de prueba se utilizó un conjunto 35 imágenes reales de un individuo caminando a 360 grados.

Para cada uno de los experimentos se especificaron tres conjuntos de parámetros los cuales representan tres niveles de dificultad: fácil, medio y difícil respectivamente. Se variaron los valores de los parámetros **Número de árboles**, **Número de píxeles** y **Profundidad de árboles**, de acuerdo a los resultados obtenidos en C.1.2, los valores escogidos se muestran en la tabla 5.3. Se mantuvieron fijos los valores de los demás parámetros como se muestra en la tabla 5.4.

Parámetros	Valores sencillos	Valores medios	Valores difíciles
Número de árboles	4	7	10
Profundidad del árbol	9	15	21
Número de píxeles tomados para entrenar	5000	15000	25000

Cuadro 5.3: Valores de parámetros utilizados para el entrenamiento.

Rango para valores de desplazamientos	150
Rango para valores de umbrales	3
Número de desplazamientos a generar	1500
Número de umbrales a generar	25

Cuadro 5.4: Parámetros por fijos.

Variables de interés

Los criterios de evaluación utilizados para medir el desempeño del algoritmo están representados en las siguientes variables: **Porcentaje de clasificación**, **Tiempo de clasificación**, mediante los cuales se evalúa el comportamiento del algoritmo.

Parámetros estudiados

Los parámetros seleccionados a variar del algoritmo corresponden a aquellos descritos en [25], además de aquellos inherentes al algoritmo de bosques aleatorios. Entre éstos parámetros, los estudiados serán variaciones de: Rango de desplazamientos u *Offset*, Rango de valores de umbral, Número de árboles, Profundidad máxima, Número de píxeles por imagen, Cantidad de desplazamientos, Cantidad de valores umbral.

Se realizaron un conjunto de pruebas preliminares sobre un conjunto sencillo de imágenes reales vistas de frente. Esto para determinar qué parámetros influyen de mayor forma sobre el rendimiento del algoritmo. Una descripción detallada de estas pruebas, junto con los resultados obtenidos, se encuentran en la sección C.1.2 del apéndice C.

A partir de estos resultados, se escogieron los parámetros **Número de árboles**, **Número de píxeles** y **Profundidad de árboles** para realizar un análisis de la varianza **ANOVA** con el fin de determinar estadísticamente el nivel de influencia de ellos en el comportamiento del algoritmo.

Como resultado del análisis de la varianza detallado en el apéndice C.1.3 se demostró que la **profundidad de los árboles** tiene una mayor influencia ante el **número de árboles** y el **número de píxeles por imagen** en el desempeño del algoritmo con un nivel de significación bastante pequeño de 0,001.

5.1.2. Entrenamiento para imágenes sintéticas

Este experimento se realizó con el fin de comprobar el aprendizaje del clasificador entrenado con imágenes sintéticas sobre la clasificación de imágenes reales. Se realizaron cuatro entrenamientos sobre diferentes conjuntos de imágenes sintéticas como se describió en la subsección 5.1.1. Los árboles obtenidos a partir de cada uno de los entrenamientos fueron utilizados para clasificar un conjunto de prueba y el conjunto de entrenamiento.

El conjunto de prueba consistía en un conjunto de imágenes reales que presentaban la misma configuración que el aquellas que conforman el conjunto de entrenamiento, es decir, si el conjunto de entrenamiento son imágenes sintéticas de un individuo caminando visto de frente, el conjunto de prueba serán imágenes reales de un individuo caminando visto de frente.

En el cuadro 5.5 se presentan los resultados de clasificación del conjunto de entrenamiento. Dadas las características de aleatoriedad del algoritmo, muchos de los píxeles de las imágenes utilizadas para entrenamiento han quedado por fuera, por lo que el riesgo de *overfitting* es muy pequeño.

Conjuntos de entrenamiento	Parámetros sencillos	Parámetros medios	Parámetros difíciles
Imágenes de frente	86.748213 %	87.359708 %	88.247305 %
Imágenes de lado izquierdo	71.837998 %	70.042384 %	69.475078 %
Imágenes de lado derecho	72.961321 %	68.821737 %	71.173931 %
Imágenes a 360°	65.138193 %	68.297977 %	68.658200 %

Cuadro 5.5: Porcentaje de clasificación promedio sobre conjunto de entrenamiento a partir del entrenamiento usando datos sintéticos. Se muestra el porcentaje asociado a la cantidad de píxeles correctamente clasificados.

Se puede observar en el cuadro 5.6 que los datos sintéticos, en general, proveen una buena clasificación de los datos reales. Como se esperaba, la clasificación es mayor para el conjunto de imágenes de frente, ya que no se tiene que desambiguar los lados del cuerpo. Para los conjuntos

rotados, incluyendo el que contiene imágenes de 360°, la clasificación es mucho menor dado la dificultad del mismo. Para obtener una mejor clasificación es necesario entrenar con un conjunto de imágenes rotadas más grande, con el fin de que el clasificador pueda aprender las pequeñas diferencias de profundidad que permitan desambiguar la rotación del individuo respecto a la cámara.

Conjuntos de entrenamiento	Parámetros sencillos	Parámetros medios	Parámetros difíciles
Imágenes de frente	68.048791 %	69.282290 %	69.303241 %
Imágenes de lado izquierdo	51.070878 %	51.570630 %	53.849571 %
Imágenes de lado derecho	53.597047 %	58.149163 %	47.595016 %
Imágenes a 360°	51.742078 %	51.481345 %	52.633425 %

Cuadro 5.6: Porcentaje de clasificación promedio sobre conjunto de prueba a partir del entrenamiento usando datos sintéticos. Se muestra el porcentaje asociado a la cantidad de píxeles correctamente clasificados.

Se observa que los porcentajes de clasificación para el conjunto de entrenamiento son mayores que para el conjunto de prueba conformado por imágenes reales. Esto es debido a que el modelo sintético usado para entrenar corresponde con las imágenes clasificadas en el conjunto de entrenamiento. Se observa que la diferencia de clasificación entre ambos conjuntos es en promedio de 12 % a favor del conjunto de entrenamiento. A pesar de ello, las imágenes sintéticas demuestran ser buenos candidatos para el entrenamiento del algoritmo, ya que obtienen altos porcentajes de clasificación sobre el conjunto de imágenes reales semejantes a los registrados en [25].

5.1.3. Entrenamiento para imágenes reales

Para este experimento se utilizó el conjunto de imágenes reales descrito en la sección C.1.1 del apéndice C. De cada conjunto se tomo el 80 % de las imágenes para conjunto de entrenamiento y el 20 % restante como conjunto de prueba. El objetivo de este experimento consiste en evaluar el aporte en la clasificación dado por el entrenamiento con imágenes reales,

además de evaluar la capacidad de aprendizaje del algoritmo para imágenes no aprendidas.

En el cuadro 5.7 se presenta la clasificación en promedio de las imágenes pertenecientes al conjunto de entrenamiento usando cada conjunto de parámetros como se describe en la sección 5.1.1.

Conjuntos de entrenamiento	Parámetros sencillos	Parámetros medios	Parámetros difíciles
Imágenes de frente	89.332366 %	89.382840 %	89.375066 %
Imágenes de lado izquierdo	79.389082 %	78.739580 %	77.626177 %
Imágenes de lado derecho	82.869292 %	82.006661 %	80.517646 %
Imágenes a 360°	65.851656 %	69.740912 %	69.844408 %

Cuadro 5.7: Porcentaje de clasificación promedio sobre el conjunto de entrenamiento a partir del entrenamiento usando datos reales. Se muestra el porcentaje asociado a la cantidad de píxeles correctamente clasificados.

Conjuntos de entrenamiento	Parámetros sencillos	Parámetros medios	Parámetros difíciles
Imágenes de frente	79.105813 %	78.574360 %	79.474805 %
Imágenes de lado izquierdo	79.389082 %	78.739580 %	77.626177 %
Imágenes de lado derecho	82.957987 %	83.271013 %	80.741331 %
Imágenes a 360°	60.145458 %	63.055175 %	63.005762 %

Cuadro 5.8: Porcentaje de clasificación promedio sobre el conjunto de prueba a partir del entrenamiento usando datos reales. Se muestra el porcentaje asociado a la cantidad de píxeles correctamente clasificados.

En el cuadro 5.8 se expone la clasificación promedio de las imágenes pertenecientes al conjunto de prueba por cada conjunto de parámetros. La clasificación disminuye en el conjunto de entrenamiento con imágenes de 360° por el aumento de la complejidad en el aprendizaje de las diferencias de profundidad, que permiten discernir entre el lado derecho e izquierdo del cuerpo (como se observó igualmente en el entrenamiento con imágenes sintéticas). En relación al conjunto de prueba la clasificación disminuye un 4 % en promedio. Como se esperaba, la clasificación para este conjunto fue mejor, sin embargo, se puede observar que no fue tan significativa a pesar de tratarse de imágenes con las que el algoritmo entrenó. Se puede

decir entonces que el algoritmo es capaz de generalizar bien para datos no observados.

Al haber entrenado con datos reales se comprueba que el algoritmo logra clasificar imágenes de manera satisfactoria. Sin embargo, el entrenar únicamente con imágenes reales es una tarea complicada en cuanto a generación de datos reales, principalmente debido a su dificultad en cuanto a etiquetamiento que debe hacerse a mano.

5.1.4. Comparación de imágenes sintéticas contra imágenes reales

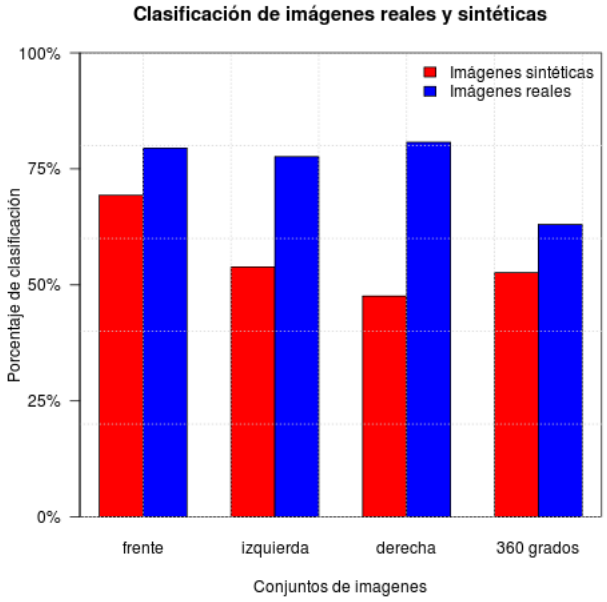

Figura 5.1: Comparación de clasificación al entrenar con imágenes reales y con imágenes sintéticas.

El entrenamiento con imágenes sintéticas para clasificación de imágenes reales, en general, presenta un desempeño menor que el entrenamiento con imágenes reales, como se muestra en la figura 5.1. Esto se debe al hecho de que el modelo no se ajusta de gran manera a los individuos presentes en las imágenes reales, es por ello que es necesario tener un conjunto sintético de gran variedad y mayor tamaño, de esa manera se pueden obtener mejores resultados. Los datos reales logran un 19 % de clasificación más que los datos sintéticos, sin

embargo, con una mayor cantidad de imágenes se espera mejorar la clasificación y disminuir el porcentaje de diferencia entre ambos tipos de imágenes.

5.2. Compresión de imágenes

Como se indicó en la sección 3.5.2 la compresión de imágenes permite que el clasificador opere de una forma más rápida en tiempo real. Se estudió la reducción en tiempo de procesamiento y número de cuadros empleando la técnica de compresión de imágenes para un salto de tres píxeles. En un principio, el Kinect procesa 30 cuadros cada segundo, esta medida es la que se usará como base para el análisis.

Para una corrida sin compresión de imagen, el tiempo promedio empleado en procesar 30 cuadros es de $3,530$ segundos, que corresponde a procesar aproximadamente 8 cuadros por segundo. Mientras que aplicando la compresión de imagen el tiempo promedio para procesar 30 cuadros es de $1,048$ segundos, que corresponde a procesar aproximadamente 29 cuadros por segundo. Se puede observar que la mejora empleando la compresión de imágenes es significativa ya que tiende a comportarse de una forma muy cercana al óptimo correspondiente a 30 cuadros por segundo. Se concluye que la compresión de imágenes es un paso importante de pre - procesamiento para la posterior clasificación y aplicación de *Mean - shift* en la imagen. La pérdida de información introducida por el salto de píxeles es despreciable respecto a la velocidad de procesamiento que se obtiene al aplicar la compresión.

5.2.1. Aplicación de *Flood Fill* sobre una imagen clasificada

Una vez obtenida una imagen clasificada, se utiliza *Flood Fill* como un paso de post - procesamiento para mejorar la calidad de la solución obtenida. Con la utilización de esta técnica es posible reparar componentes mal clasificadas de la imagen y así estabilizar el proceso de estimación de articulaciones por cada cuadro.

El algoritmo posee dos parámetros que definen su comportamiento, estos son: **Cantidad de iteraciones** y **Umbral de tamaño de componente**. El primer parámetro define cuántas iteraciones de *Flood Fill* serán realizadas, mientras que el segundo define a partir de que tamaño las componentes serán coloreadas. En la figura 5.2 se muestra el comportamiento del algoritmo para una a tres iteraciones con valores de umbral de 10, 50, 100 y 200 píxeles. Se observa que la ganancia en cuanto a porcentaje de clasificación no es tan significativa, teniendo una mejora máxima del 1,4 %. En la figura 5.3 se observa que el crecimiento en tiempo es aproximadamente lineal respecto al número de iteraciones.

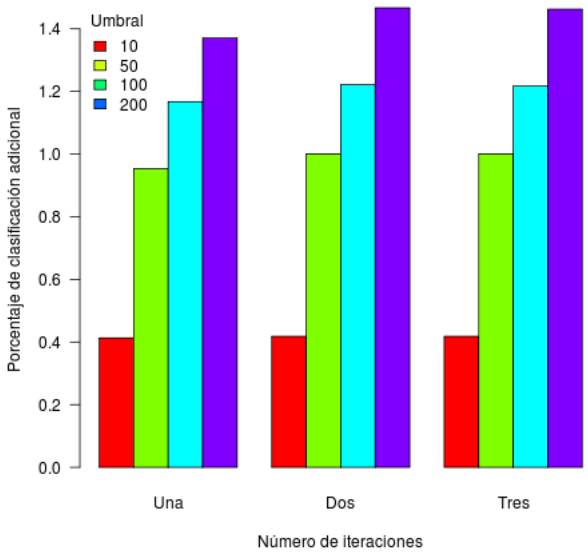

Figura 5.2: Comparación de clasificación de *Flood Fill* para varios parámetros estudiados.

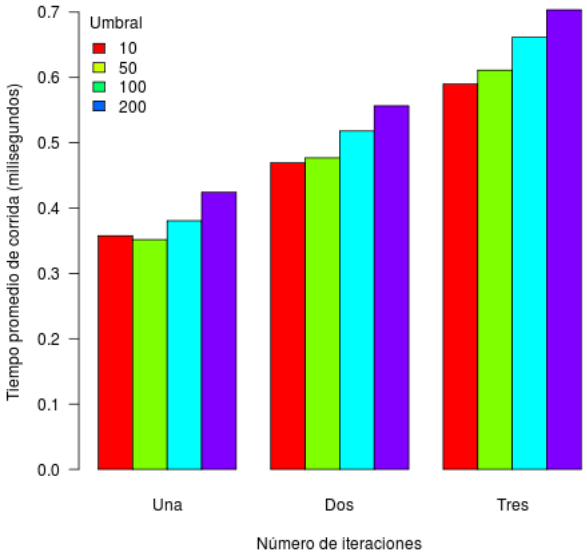

Figura 5.3: Tiempo de corrida empleado para cada uno de los experimentos de *Flood Fill*.

A partir de los resultados obtenidos se observa que el mejor comportamiento para *Flood Fill* es aquel correspondiente a dos iteraciones con tamaño de componente igual a 200 píxeles. Para más de dos iteraciones la mejora es poco significativa y sólo acarrea un mayor tiempo de cómputo. A pesar de presentar una pequeña mejora en la clasificación, el fuerte de *Flood Fill* es la estabilización del comportamiento del *Mean - shift* utilizado para estimar las articula-

ciones del esqueleto, ya que, al eliminar pequeñas componentes mal clasificadas, se garantiza una convergencia más rápida y precisa.

5.3. Algoritmos utilizados para refinamiento de la matriz de transformación

En esta sección se probaron los algoritmos descritos en la sección 4.2 para refinar las matrices de transformación para obtener una solución que integre mejor las vistas de cada par de sensores.

5.3.1. Descripción general de los experimentos

Los experimentos se realizaron a partir de datos tomados con dos sensores *Kinect*. Debido a que los algoritmos operan por cada par de sensores, los resultados son igual de consistentes para el uso de más de dos de ellos. Para cada uno de los métodos se utilizó un conjunto de prueba de 1000 pares de cuadros con puntos del esqueleto estimados con ambos sensores respectivamente. El conjunto total de cuadros fue dividido a la mitad obteniendo dos conjuntos de 500 cuadros. Uno de los conjuntos se utilizó como entrada para la corrida de cada método y el otro conjunto se utilizo para evaluar el error de la solución producida.

Los esqueletos fueron estimados utilizando la librería **NITE** de *PrimeSense* debido a que cuentan con un mayor número de puntos de articulaciones y una mayor precisión para medir la efectividad de la calibración.

El error de una solución se calcula aplicándola a cada par de cuadros del conjunto y calculando la distancia promedio entre pares de puntos correspondientes como se explica en la sección 4.2.1.

Parámetros estudiados

Para cada método se probó variando el **número de cuadros iniciales** con los que se calcula la matriz solución. De cada par de cuadros del conjunto de entrada se extrae una matriz solución como se explicó en la sección 4.1.3, estas matrices son utilizadas por cada método para producir una mejor. Para cada algoritmo se tomo como parámetro variable el **tamaño del conjunto de cuadros inicial** sobre el cual trabajar. Se hicieron pruebas para cada algoritmo tomando desde un cuadro hasta el total de los 500 cuadros de entrada.

5.3.2. Comparación entre el desempeño de cada algoritmo

Los resultados expuestos en el cuadro 5.9 representan la máxima reducción del error lograda al producir una solución por cada método diferente. También se muestra el tiempo de corrida que tomó por cada algoritmo para calcular a la solución.

Algoritmo	Error promedio (milímetros)	Cuadros	Tiempo (segundos)
Transformación entre dos nubes	29.8226	158	0.01123095
Promedio entre matrices	30.3137	376	0.05436206
Local search	29.6246	150	7.49534
Algoritmo genético	29.5964	168	10.80967

Cuadro 5.9: Menor error promedio de adaptación logrado por cada algoritmo con el número de cuadros usado y el tiempo de corrida correspondiente.

De todas las matrices obtenidas del conjunto de entrada para los algoritmos, la que posee un menor error al aplicarla en el conjunto de prueba, tiene un error promedio de **32.6632**. Como se puede observar en el cuadro 5.9, al aplicar los algoritmos sobre esta matriz se logra una reducción del error de al menos 3 milímetros.

En la figura 5.4 se muestra el promedio del error de la solución obtenida por cada método variando la cantidad de matrices iniciales utilizadas para el cálculo de la misma. En la figura 5.5 se expone el consumo de tiempo por cada algoritmo en el cálculo de la solución.

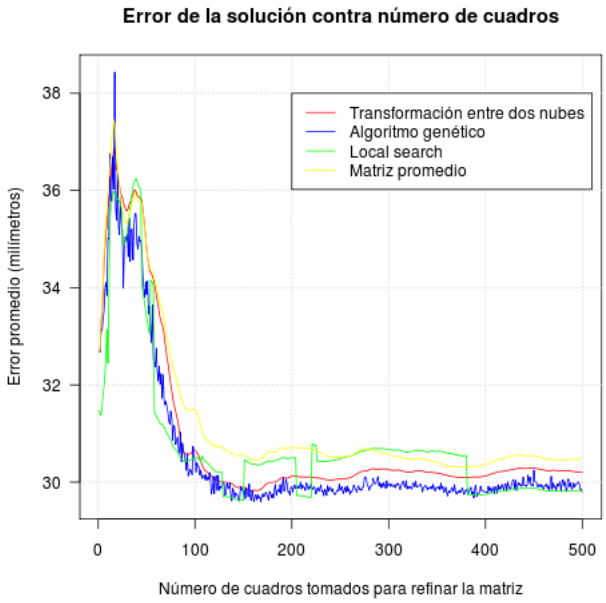

Figura 5.4: Error asociado a una solución contra número de cuadros inicial para los métodos de refinación probados

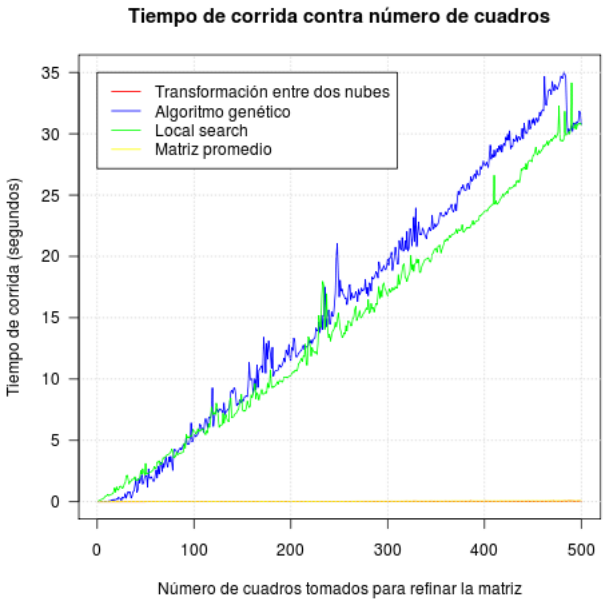

Figura 5.5: Tiempo de corrida contra número de cuadros inicial para los métodos de refinación probados

Se puede apreciar por la figura 5.4 que a partir de los 100 cuadros, las soluciones de los algoritmos dejan de mejorar. Este resultado es importante para determinar el número de cuadros que es necesarios capturar para que se logre obtener una transformación que permita una captura más precisa del movimiento. En la misma imagen se muestra que todos los algoritmos siguen un comportamiento similar y casi paralelo con respecto a los demás. En general, el algoritmo genético posee el mejor desempeño con respecto a la reducción del error que los demás métodos. Sin embargo, en la figura 5.5 el algoritmo genético supera al resto en tiempo de corrida.

Capítulo 6

Conclusiones y recomendaciones

En el presente proyecto se logró implementar un sistema que realiza captura de movimiento utilizando la cámara de profundidad del sensor *Kinect*. Para esto se utilizó el algoritmo *Random Forest* adaptado para el aprendizaje de las diferencias en los píxeles en las imágenes de profundidad propuesto por [25] y una aproximación basada en *Mean - shift* para estimar la propuesta final de articulaciones del esqueleto. Además se diseñó e implementó un sistema que es capaz de tomar los puntos de los esqueletos propuestos por cada sensor e integrarlos en un mismo sistema de coordenadas para promediar las predicciones, descartar errores y completar la información en caso de oclusión.

Las conclusiones específicas obtenidas tras la realización de este trabajo son las siguientes:

1. El clasificador, como se observó en los resultados, es capaz de generalizar efectivamente para datos no vistos. Sin embargo, para obtener el mejor desempeño, es necesario contar con una gran cantidad y variedad de datos de entrenamiento, además de mucho poder de cómputo y suficientes recursos para poder procesarlos. Esto significó un problema en el desarrollo, ya que era complicado generar los datos y procesarlos en un tiempo factible. Fue necesario trabajar un modelo 3D sobre un conjunto de datos de movimiento previamente generados para obtener los datos de entrenamiento sintéticos necesarios para el entrenamiento e implementar un cluster para lograr optimizar el tiempo de entrenamiento del algoritmo. A pesar de ello, el clasificador logró porcentajes de clasificación mayores al 60 %, similares a los resultados observados en [25].

2. El aprendizaje con datos sintéticos es una opción adecuada para generar datos de

entrenamiento para el algoritmo. Dichas imágenes son capaces de generalizar bien y aportar suficiente información para el caso real. La utilización de datos sintéticos ayuda a automatizar el proceso de generación y etiquetamiento de imágenes, ya que este último en imágenes reales es un trabajo que debe realizarse manualmente. A partir de los datos sintéticos, además, es posible variar fácilmente la forma del modelo utilizado, por ende, es posible aumentar el espectro de individuos que pueden presentarse en una escena, suministrando así más información al clasificador.

3. Como el clasificador ha de trabajar sobre imágenes reales generadas por el sensor *kinect* en ambientes diversos, fue necesario restringir el entrenamiento a imágenes de personas, asumiendo sustracción de fondo. Por ende, ante diversos ambientes, se necesita realizar sustracción de fondo en tiempo real y conservar únicamente la forma de la persona sobre la cual se aplicará el clasificador. De no ser así, por la estrategia de entrenamiento empleada, el clasificador puede confundirse con elementos presentes en el fondo.

4. Para obtener las propuestas de posición de las articulaciones, no es suficiente con sólo estimar los centros de masa promediando los puntos pertenecientes a las regiones clasificadas. El algoritmo *Mean-shift* aplicado sobre dichas áreas permite eliminar aquellos píxeles mal clasificados o *outliers* que puedan degradar la calidad de las soluciones.

5. Utilizando una aproximación basada en SVD para unir las propuestas de articulaciones, es posible relacionar los esqueletos vistos desde distintas perspectivas. Además, dado el porcentaje de confiabilidad de la clasificación de una articulación, es posible determinar que sensor está fallando en la estimación de una de las articulaciones del esqueleto y rescatarla a partir de la información inferida por los demás sensores. Métodos de refinación de la solución como algoritmo genético o búsqueda local, a pesar de demostrar una mayor calidad de solución, no son realmente necesarios y consumen mucho tiempo.

Aportes realizados

Se implementó una librería de código abierto basada en la aplicación del clasificador *Random forest* para el problema de detección de elementos en imágenes de profundidad. Actualmente este clasificador se encuentra implementado en librerías como *OpenCV*, sin embargo, su enfoque radica directamente en problemas de regresión y clasificación siendo complicada su adaptación a problemas más específicos como el presente en este proyecto. Es importante señalar que el clasificador implementado no sólo se aplica al problema de detección personas en imágenes, sino que puede ser utilizado para detectar cualquier otro tipo de objeto en imágenes de profundidad, correspondiente con los datos de entrenamiento que se le haya suministrado.

Debido a la naturaleza del clasificador, no es necesario realizar alguna pose inicial para generar el esqueleto, como se observa en otros sistemas como el desarrollado por *NITE* para *OpenNI*. El esqueleto es generado inmediatamente al entrar a la escena, y se actualiza por cada cuadro.

Se implementó un sistema de captura de movimiento que maneja el problema de oclusión introducida por la falta de información provista por un sólo sensor. Con más sensores posicionados alrededor de la escena es posible completar la información de captura de movimiento que un sólo sensor puede obtener. El sistema es capaz de aumentar el espectro de movimientos gracias a la información introducida por los demás sensores.

Direcciones futuras

El principal impedimento en el desarrollo del presente trabajo fue la falta de recursos en cuanto a hardware y datos de entrenamiento. En este sentido se recomienda escalar este sistema de tal forma que sea capaz de detectar una mayor variedad de poses y partes del

cuerpo, de forma que se pueda aportar mucha más información del movimiento.

A continuación, se nombran una serie de recomendaciones que pueden mejorar la clasificación de píxeles y la captura de movimiento:

- Generar un conjunto de imágenes de entrenamiento de gran tamaño con diferentes modelos y poses variadas, con el fin de obtener una clasificación más completa que dé mayor flexibilidad en la detección de diferentes poses; así como la detección de más partes del cuerpo que ayuden a construir un esqueleto más completo que pueda aportar más información.

- Mejorar el hardware disponible, aumentando el número de máquinas y procesadores para reducir el tiempo de entrenamiento. Con esta recomendación también se incluye mejorar el método de paralelización para el entrenamiento del algoritmo utilizando **MPI** o simplemente utilizando hilos de ejecución sobre hardware y/o software que realicen la paralelización en múltiples máquinas implícitamente.

- Estudiar posibles alternativas para adaptar el algoritmo *Random Forest* para el aprendizaje de las imágenes de profundidad producidas por el sensor *Kinect*. En el presente proyecto se utilizó la función de atributo descrita en la sección 3.3.1 para separar los conjuntos de píxeles. Se podrían estudiar otras funciones alternativas y probar su desempeño.

- Estudiar la posibilidad de utilizar la información de la cámara **RGB** del sensor *Kinect* para desambiguar y/o completar la clasificación. Por ejemplo, detectando el rostro del individuo, se podría saber si se encuentra de frente o no, lo que ayudaría a desambiguar la clasificación entre las partes derecha o izquierda del individuo.

- Probar un algoritmo de seguimiento de los puntos del esqueleto que funcione con los valores de profundidad y pueda ayudar a refinar los datos obtenidos descartando valores

erróneos o completándolos en caso que el algoritmo de clasificación falle. Este método mejoraría la clasificación para poses no vistas en el entrenamiento.

- Implementar la clasificación de los píxeles en la tarjeta gráfica con el fin de acelerar el procesamiento de cuadros y píxeles en tiempo real. Esto le daría espacio a otros métodos que puedan mejorar la clasificación como los mencionados anteriormente.

- Diseñar una estrategia que consista en proponer un valor umbral para cada parte del cuerpo, que permita filtrar en tiempo real aquellos píxeles cuya probabilidad no sea adecuada. Actualmente se tiene un valor umbral único para todas las partes del cuerpo.

Con respecto al uso de múltiples sensores se describen las siguientes recomendaciones:

- Desarrollar un método alternativo para hallar la matriz de transformación entre los sistemas de coordenadas dados por cada sensor. Se puede probar un algoritmo de detección de bordes sobre la imagen de profundidad y obtener puntos en tres dimensiones de un objeto con esquinas como un cuadrado y conociendo la correlación de los puntos en dos imágenes se puede hallar la matriz con el método descrito en la sección 2.13 de forma más precisa.

- Se puede utilizar un algoritmo de aprendizaje sobre cada punto del esqueleto que ajuste una distribución de probabilidades a la distancia que separa cada punto propuesto por cada sensor en particular con el fin de descartar anomalías.

Bibliografía

[1] *Handbook of Approximation Algorithms and Metaheuristics (Chapman & Hall/CRC Computer & Information Science Series)*. Chapman and Hall/CRC, 1 edition, May 2007.

[2] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-d point sets. *IEEE Trans. Pattern Anal. Mach. Intell*, 9:698–700, September 1987.

[3] M Bertozzi, A Broggi, C Caraffi, M Delrose, M Felisa, and G Vezzoni. Pedestrian detection by means of far-infrared stereo vision. *Computer Vision and Image Understanding*, 106(2-3):194–204, 2007.

[4] Dr. Gary Rost Bradski and Adrian Kaehler. *Learning opencv*. O'Reilly Media, Inc., first edition, 2008.

[5] Leo Breiman. Random forests. *Machine Learning*, 45:5–32, 2001. 10.1023/A:1010933404324.

[6] Nicolas Burrus. Rgbdemo. `http://nicolas.burrus.name/index.php/Research/KinectRgbDemoV6`, 2011.

[7] Nicolas Burrus. Kinectcalibration. `http://nicolas.burrus.name/index.php/Research/KinectCalibration`, 2012.

[8] David H. Douglas and Thomas K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The*

International Journal for Geographic Information and Geovisualization, 10(2):112–122, October 1973.

[9] S. C. Eisenstat, M. C. Gursky, M. H. Schultz, and A. H. Sherman. Yale sparse matrix package i: The symmetric codes. *International Journal for Numerical Methods in Engineering*, 18(8):1145–1151, 1982.

[10] S.C. Eisenstat, M.C. Gursky, M.H. Schultz, A.H. Sherman, and Yale university new haven conn dept. of computer science. *Yale Sparse Matrix Package. II. The Nonsymmetric Codes*. Defense Technical Information Center, 1977.

[11] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users'Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.

[12] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[13] Bohyung Han, D. Comaniciu, Ying Zhu, and L. S. Davis. Sequential Kernel Density Approximation and Its Application to Real-Time Visual Tracking. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(7):1186–1197, 2008.

[14] T. S. Huang, S. D. Blostein, and E. A. Margerum. Least-squares estimation of motion parameters from 3-d point correspondences. *Proc. IEEE Conf. Computer Vision and Pattern Recognition.*, June 24-26 1986.

[15] B. Jacob. *Linear algebra*. Series of books in the mathematical sciences. Freeman, 1990.

[16] V. Lepetit and P. Fua. Keypoint recognition using randomized trees. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(9):1465–1479, September 2006.

[17] E. J. Marey. *Animal mechanism: a treatise on terrestrial and aerial locomotion*. London, 1906.

[18] Microsoft. Introducing kinect for xbox 360. `http://www.xbox.com/kinect`, 2012.

[19] Tom M. Mitchell. *Machine learning*. McGraw Hill series in computer science. McGraw-Hill, 1997.

[20] Thomas B Moeslund and Erik Granum. A survey of computer vision-based human motion capture. *Academic Press*, (81):231–268, 2001.

[21] Eadweard Muybridge. *Muybridge's Complete human and animal locomotion: all 781 plates from the 1887 Animal locomotion / by Eadweard Muybridge ; introd. to the Dover edition by Anita Ventura Mozley*. Dover Publications, New York, 1979.

[22] Yuri Pekelny and Craig Gotsman. Articulated Object Reconstruction and Markerless Motion Capture from Depth Video. *Computer Graphics Forum*, 27(2):399–408, April 2008.

[23] David A. D. Gould Markus Gross Chris Kazmier Richard Keiser Charles J Lumsden Alberto Menache Matthias MÜller-Fischer F. Kenton Musgrave Mark Pauly Darwyn Peachey Ken Perlin Hanspeter Pfister Jason Sharpe Martin Wicke Mark R. Wilkins Nicholas Woolridge Steven Worley Rick Parent, David S. Ebert. *Computer Animation Complete. All-in-One: Learn Motion Capture, Characteristic, Point-Based, and Maya Winning Techniques*. 2010.

[24] R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). pages 1–4, May 2011.

[25] Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake. Real-Time Human Pose Recognition in Parts from Single Depth Images. June 2011.

[26] C. Stauffer and W. E. L. Grimson. Adaptive background mixture models for real-time tracking. volume 2, pages 246–252, Los Alamitos, CA, USA, August 1999. IEEE Computer Society.

[27] Satoshi Suzuki and KeiichiA be. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32 – 46, 1985.

[28] Richard Szeliski. *Computer Vision: Algorithms and Applications (Texts in Computer Science)*. Springer, 1st edition, November 2010.

[29] Trucco and Alessandro Verri. *Introductory Techniques for 3-D Computer Vision*. Prentice Hall, March 1998.

[30] Carnegie Mellon University. Cmu graphics lab motion capture database. `http://mocap.cs.cmu.edu/`, 2012.

[31] Urs and Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(3):244 – 256, 1972.

[32] Paul Viola, Michael J. Jones, and Daniel Snow. Detecting pedestrians using patterns of motion and appearance. *International Journal of Computer Vision*, 63:153–161, 2005. 10.1007/s11263-005-6644-8.

[33] M. Wall. GAlib: A C++ library of genetic algorithm components. *Mechanical Engineering Department, Massachusetts Institute of Technolog y*, 1996.

[34] Z. Zivkovic. Improved adaptive Gaussian mixture model for background subtraction. *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, 2:28–31 Vol.2, 2004.

Apéndice A

Conceptos relacionados

A.1. Definiciones matemáticas

A.1.1. Matriz de rotación

Según [15] una **matriz de rotación** es una matriz que es empleada para realizar una rotación en un espacio vectorial. Esta rotación puede ser representada con un eje de rotación y un ángulo. En un espacio de tres dimensiones $\mathbb{R}^3$, para los ejes x, y y z, tenemos las siguientes matrices de rotación en cada uno de ellos para un determinado ángulo α.

$$R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & -sen(\alpha) \\ 0 & sen(\alpha) & cos(\alpha) \end{pmatrix} \tag{A.1}$$

$$R_y(\alpha) = \begin{pmatrix} cos(\alpha) & 0 & sen(\alpha) \\ 0 & 1 & 0 \\ -sen(\alpha) & 0 & cos(\alpha) \end{pmatrix} \tag{A.2}$$

$$R_z(\alpha) = \begin{pmatrix} cos(\alpha) & -sen(\alpha) & 0 \\ sen(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{A.3}$$

La rotación es obtenida utilizando la multiplicación de matrices. Dado un vector v y una matriz de rotación R, podemos obtener el vector v' rotado mediante la multiplicación Rv.

De esta misma forma, se pueden obtener otras matrices de rotación utilizando la multiplicación de matrices. Por ejemplo, la siguiente multiplicación:

$$R_x(\alpha)R_y(\beta)R_z(\gamma) \tag{A.4}$$

Las matrices de rotación cumplen las siguientes propiedades:

- $R^t = R^{-1}$ (R es ortogonal)
- $det(R) = 1$
- $R(\alpha + \beta) = R(\alpha).R(\beta)$
- $R(0) = I$

A.1.2. Transformación rígida

De acuerdo con [15], cuando hablamos de **transformación rígida** nos referimos a una traslación y una rotación aplicadas a la posición de un objeto. Estas operaciones se representan a través de un vector y una matriz respectivamente. Teniendo entonces:

$$T = \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix} \qquad R = \begin{pmatrix} r_{1,1} & r_{1,2} & r_{1,3} \\ r_{2,1} & r_{2,2} & r_{2,3} \\ r_{3,1} & r_{3,2} & r_{3,3} \end{pmatrix}$$

Donde T es el vector de translación y R es la matriz de rotación. Se puede obtener un punto x' después de aplicar la transformación rígida a un punto tridimensional x con la siguiente fórmula:

$$x' = (R * x) + T$$

A.1.3. Sistema de ecuaciones inconsistente

Se dice que un **sistema de ecuaciones** es **inconsistente** si no existe una solución que cumpla con todas las ecuaciones que lo componen.

Solución de mínimos cuadrados

Sea A una matriz real $m \times n$. Un vector $\vec{c} \in R^n$ es llamado una **solución de mínimos cuadrados** a un posible sistema inconsistente $A\vec{X} = \vec{b}$ si la distancia

$$||A\vec{c} - \vec{b}|| = \sqrt{< A\vec{c} - \vec{b}, A\vec{c} - \vec{b} >}$$

es un mínimo (entre todos los posibles valores de $\vec{c}$)

A.1.4. Centroide

Un **centroide** de un conjunto finito de k puntos $x_1, x_2, .., x_k \in \mathbb{R}^n$ viene dado por

$$C = \frac{x_1 + x_2 + ... + x_k}{k}$$

A.2. Modelo de cámara estenopeica

El proceso de visión comienza con la detección de la luz del entorno [4]. La luz viaja a través del espacio hasta que choca con un objeto, que puede absorberla o reflejarla. La luz reflejada es captada por la cámara y es la que permite formar la imagen. El modelo de **cámara estenopeica** (*pinhole camera model*) es el modelo más simple utilizado para explicar la geometría asociada al proceso descrito anteriormente.

El modelo consiste en una pared imaginaria que posee un pequeño orificio en el centro

por el cual debe pasar la luz. Este modelo es puramente teórico y es ideal para explicar la geometría de proyección. En la práctica, es necesario contar con lentes en la cámara para recoger más luz y así poder formar imágenes. Los lentes, sin embargo, complican el modelo ya que introducen factores de distorsión inherentes a ellos. Los problemas de estimación de la geometría del modelo y la distorsión de los lentes son resueltos mediante el proceso de **calibración de cámara**.

En este modelo solamente entra un rayo de luz por cada punto de la escena. Cada punto se proyecta entonces sobre un plano llamado "plano de la imagen", que siempre se encuentra en foco y cuyo tamaño de imagen respecto a lo visto en la escena, viene dado por un parámetro de la cámara llamado **distancia focal**. Para este modelo, la distancia focal representa la distancia entre el orificio y el plano de la imagen. El punto u orificio descrito en el modelo se interpreta como el "centro de proyección" de la imagen, sobre el cual se construye un sistema de coordenadas, en el cual se define un eje que intercepta el plano de la imagen, llamado "eje óptico", que indica la dirección de la cámara.

Por lo general, el chip de la cámara no se encuentra alineado con el eje óptico, por lo que se introducen los parámetros c_x y c_y, correspondientes a un posible desplazamiento del centro de coordenadas en el plano de la imagen respecto al eje óptico. Si se tiene que f es la distancia focal, se puede estimar la posición en píxeles (x_{plano}, y_{plano}) en el plano de la imagen de algún punto $Q = (X, Y, Z)$ en el mundo real mediante las siguientes ecuaciones:

$$x_{plano} = f_x \frac{X}{Z} + c_x \tag{A.5}$$

$$y_{plano} = f_y \frac{Y}{Z} + c_y \tag{A.6}$$

Donde se introducen distintas distancias focales f_x y f_y, ya que, por lo general, los píxeles en cámaras de bajo costo son rectangulares en vez de ser cuadrados.

En particular, para el caso del sensor Kinect, los valores de f y c se muestran a continuación:

- $f_x = 5{,}9421434211923247e+02$
- $f_y = 5{,}9104053696870778e+02$
- $c_x = 3{,}3930780975300314e+02$
- $c_y = 2{,}4273913761751615e+02$

A.3. Distorsión de lentes

Normalmente, las imágenes tomadas desde cámaras contienen una ligera distorsión producida por sus lentes. Esta distorsión se debe a la forma y a la posición de los lentes dentro de la cámara. Existen dos tipos principales de distorsión, la distorsión **radial** y la **tangencial** [4].

A.3.1. Distorsión radial

Este tipo de distorsión es producida por la forma circular de los lentes. El efecto que produce se conoce como "ojo de pescado" y se observa en los píxeles más alejados del centro de la imagen.

La distorsión generada en el centro de la imagen es nula y se incrementa a medida en que se aleja de éste. El crecimiento de la intensidad en la distorsión puede ser modelada utilizando la *serie de Taylor* expandida alrededor de cero. Sólo son necesarios los primeros tres términos de ésta para tener una buena aproximación. Por convención, estos términos se conocen como k_1, k_2 y k_3 y la corrección de los píxeles en el plano de proyección viene dado por las fórmulas:

$$x_{corregida} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{A.7}$$

$$y_{corregida} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{A.8}$$

Donde el par $(x_{corregida}, y_{corregida})$ representa a la coordenada del píxel (x, y) una vez corregida la distorsión.

A.3.2. Distorsión tangencial

La distorsión tangencial tiene relación con la alineación de los lentes con el plano de la imagen. En una cámara digital, esto significa que el sensor **CMOS**, que recibe la luz para transformarla en una imagen, no se encuentra perfectamente alineado de forma paralela a los lentes de la cámara.

Para representar este tipo de distorsión de la forma más simple, se necesitan los parámetros p_1 y p_2. Estos parámetros son usados para corregir la distorsión dada la siguiente fórmula:

$$x_{corregida} = x + [2p_1 y + p_2(r^2 + 2x^2)] \tag{A.9}$$

$$y_{corregida} = y + [p_1(r^2 + 2y^2) + 2p_2 x] \tag{A.10}$$

Donde el par $(x_{corregida}, y_{corregida})$ representa a la coordenada del píxel (x, y) una vez corregida la distorsión.

A.4. Descomposición de valor singular

Una **descomposición de valor singular** es una factorización de una matriz real o compleja. Sea A una matriz $m \times n$, ésta puede ser expresada como un producto $A = USV^t$

denominado descomposición de valor singular, donde U es una matriz ortogonal $m \times m$, S es una matriz pseudo diagonal $m \times n$ y V es una matriz ortogonal $n \times n$. Los elementos de la diagonal de S, σ_i, son llamados valores singulares y son tales que $\sigma_1 \leq \sigma_2 \leq ... \leq \sigma_n$.

Entre los usos de la descomposición de valor singular encontramos tres importantes:

- Resolver sistemas de ecuaciones no homogéneos.
- Resolver sistemas de ecuaciones lineales homogéneos de rango deficiente.
- Garantizar que una matriz cumple con ciertas propiedades (Por ejemplo ortogonalidad).

A.5. Método para hallar la transformación rígida usando la descomposición de valores singulares

De acuerdo al método descrito en [2], dos series de puntos tridimensionales p_i y $p_i{}'$ con $i = 1, 2, ..., n$ se relacionan mediante una transformación rígida (ver sección A.1.2) y se representan como un sistema inconsistente de ecuaciones (ver sección A.1.3) de la siguiente forma:

$$p'_i = Rp_i + T \tag{A.11}$$

donde p_i y p'_i son considerados vectores columna en $\mathbb{R}^3$, R es una matriz de rotación en $\mathbb{R}^{3\times3}$ y T es un vector de traslación en $\mathbb{R}^3$. Debido a que es un sistema inconsistente, se debe hallar una solución de mínimos cuadrados (ver sección A.1.3). El objetivo entonces es hallar R y T tales que minimicen la ecuación:

$$\Sigma^2 = \sum_{i=1}^{N} ||p'_i - (Rp_i + T)||^2 \tag{A.12}$$

Sea $\widehat{R}$ y $\widehat{T}$ una solución de mínimos cuadrado para A.12, entonces $\{p'_i\}$ y $\{p''_i = \widehat{R}p_i + \widehat{T}\}$ tienen los mismos centroides (ver sección A.1.4) como se muestra en [14]. Por lo tanto, se tiene que:

$$p' = p'' \tag{A.13}$$

donde,

$$p' = \sum_{i=1}^{N} p'_i \tag{A.14}$$

$$p'' = \sum_{i=1}^{N} p''_i = \widehat{R}p + \widehat{T} \tag{A.15}$$

$$p = \sum_{i=1}^{N} p_i \tag{A.16}$$

Sea,

$$q_i = p_i - p \tag{A.17}$$

$$q'_i = p'_i - p' \tag{A.18}$$

Tenemos entonces:

$$\Sigma^2 = \sum_{i=1}^{N} ||q'_i - Rq_i||^2 \tag{A.19}$$

Por lo tanto el problema original de mínimos cuadrados es reducido a dos partes:

1. Encontrar $\widehat{R}$ para minimizar Σ^2 en A.19.

2. Encontrar $\widehat{T}$ dada por $\widehat{T} = p' - \widehat{R}p$.

A.5.1. Algoritmo para hallar $\widehat{R}$ usando una descomposición SVD

1. Se calculan los centroides de $\{p_i\}$ y $\{p'_i\}$. Después se calculan $\{q_i\}$ y $\{q'_i\}$.

2. Se calcula la matriz 3×3

$$H = \sum_{i=1}^{N} q_i (q'_i)^t \tag{A.20}$$

3. Encontrar la descomposición de valores singulares (descrito en la sección A.4)

$$H = USV^t \tag{A.21}$$

4. Se calcula

$$X = VU^t \tag{A.22}$$

5. Se calcula el determinante de X:

   - Si $det(X) = +1$, entonces $\widehat{R} = X$.
   - Si $det(X) = -1$, entonces el algoritmo falla.

Apéndice B

Algoritmos de aprendizaje y metaheurísticas

B.1. Árboles de decisión

A continuación se dará una breve descripción de los algoritmos utilizados para el entrenamiento de los árboles, así como los detalles sobre el cálculo de la entropía y ganancia de información asociados a dicho proceso.

B.1.1. Algoritmo ID3

El algoritmo ID3 "Aprende" construyendo el árbol de decisión de arriba hacia abajo, recibe como entrada un conjunto T de ejemplos de entrenamiento t_i con N variables ya clasificados. Cada ejemplo t_i es un vector $t_i = x_1, x_2, ..., x_n$ donde cada x_i corresponde a las variables del ejemplo. Se tiene además un vector $y = y_1, y_2, ...y_n$ que indica la clasificación o valor de predicción asociados a cada ejemplo del conjunto.

Se empieza por el nodo raíz donde se decide cuál es la variable que mejor separa el conjunto de ejemplos de entrenamiento, para ello se emplea una prueba estadística que por lo general consiste en la medida de entropía y ganancia de información. Se selecciona la mejor variable para ser utilizada como prueba en el nodo raíz. Se crean tantos nodos hijos como valores pueda tener la variable escogida. El proceso se repite para cada nodo hijo usando los ejemplos de entrenamientos asociados a ese nodo con el fin de escoger la variable que mejor

clasifica los ejemplos de entrenamiento en ese punto del árbol. El proceso termina cuando el subconjunto de ejemplos tienen el mismo valor de predicción o cuando seguir dividiendo no aporta información que afecte el valor de las predicciones.

B.1.2. Algoritmo C4.5

El algoritmo C4.5 opera de la misma manera que ID3, posee un conjunto de mejoras entre las que se pueden nombrar:

- Manejo de variables con valores continuos, en este caso, en cada nodo se realiza una comparación con un valor que define hacia que rama del árbol se clasificará el ejemplo.
- Manejo de datos con valores de variables faltantes.
- Poda de ramas del árbol que no aportan valor a la predicción de ejemplos.
- Posibilidad de asociar costos a las variables. Se busca un árbol de decisión que sea capaz de minimizar los costos asociados a los valores de las variables de los ejemplos de entrenamiento.

B.1.3. Medida de entropía

En teoría de la información, la entropía es una medida que permite caracterizar la impureza de un conjunto de ejemplos T cuya clasificación puede tomar N valores distintos. En el caso de clasificación binaria, se define la entropía de T de la siguiente manera [19]:

$$Entrop(T) = -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus} \tag{B.1}$$

Donde $p_{\oplus}$ representa la proporción de ejemplos positivos en T y $p_{\ominus}$ es la proporción de ejemplos negativos en T. Para el caso de cálculo de entropía se define $0 \log(0) = 0$.

En el caso más general, la entropía de T se define de la siguiente manera:

$$Entrop(T) = \sum_{i=1}^{n} -p_i \log_2 p_i \tag{B.2}$$

B.1.4. Ganancia de información

La ganancia de información se define como la reducción esperada de la entropía causada por el particionamiento del conjunto de entrenamiento respecto a una variable en particular. La ganancia de información mide el grado de efectividad con que un atributo X clasifica el conjunto de ejemplos de entrenamiento. La ganancia de información de un atributo X respecto a un conjunto de ejemplos T se define como:

$$Ganancia(T, X) = Entrop(T) - \sum_{v \in Valores(X)} \frac{|T_v|}{|T|} Entrop(T_v) \tag{B.3}$$

Donde $Valores(X)$ es el conjunto de todos los posibles valores que puede tomar el atributo X, y T_v, es el subconjunto de T para el cual el atributo X tiene el valor v, específicamente, $T_v = \{t \in T | X(t) = v\}$.

B.2. Bosques aleatorios

El entrenamiento de los árboles se realiza de una forma parecida a los árboles de decisión, la diferencia yace en los factores de aleatorización que se incluyen durante la construcción de cada árbol. Al igual que en el caso de los árboles de decisión, el árbol se construye de arriba hacia abajo, se empieza desde el nodo raíz, y en cada nodo se realiza una prueba que separa el espacio de ejemplos de entrenamiento particionando a medida en que se desciende en el árbol. Cada árbol se construye hasta su mayor extensión posible, no se podan ramas. Al llegar a un nodo hoja, se almacena la distribución de probabilidades de las clases de los

ejemplos que han llegado a dicho nodo.

Los árboles se entrenan con los mismos parámetros pero en conjuntos de entrenamientos diferentes, que son generados a partir del conjunto de entrenamiento original, tomando ejemplos de forma aleatoria y con remplazo. El tamaño de cada conjunto de entrenamiento es igual al del conjunto original.

Específicamente, en cada nodo se realiza una prueba basada en la ganancia de información (según la ecuación B.3) para estimar cual es el atributo que mejor separa al conjunto de ejemplos de entrenamiento. En los bosques aleatorios la prueba de ganancia de información no se realiza sobre todos los atributos que definen un ejemplo, sino sobre un subconjunto de ellos tomado de forma aleatoria.

Sea N la cantidad de atributos que poseen los ejemplos de entrenamiento, se define un número $m \ll N$ que especifica la cantidad de atributos que serán tomados de forma aleatoria en cada nodo. El valor de m se mantiene constante durante todo el crecimiento del bosque.

Por lo general, como criterios de parada para el entrenamiento del árbol se pueden especificar los siguientes:

- Alcanzar una profundidad máxima previamente especificada.
- Todos los elementos del conjunto de entrenamiento poseen la misma clasificación.
- Los elementos del conjunto de entrenamiento son insuficientes y no aportan información para una nueva subdivisión.

De acuerdo con [16], cada nodo hoja del árbol contendrá una distribución de probabilidades de las posibles clasificaciones de los ejemplos. Estas probabilidades son de la forma $P_{\eta(l,\mathbf{p})}(Y(\mathbf{p}) = c)$, donde c es una de las posibles clasificaciones que puede tener el ejemplo $\mathbf{p}$ y $\eta(l, \mathbf{p})$ es el nodo hoja del árbol T_l alcanzado por el ejemplo $\mathbf{p}$.

Las probabilidades se calculan durante el entrenamiento como la relación entre el número de ejemplos de clase c que llegan al nodo hoja η y el número total de ejemplos que llegan al mismo.

La clasificación de un ejemplo $\mathbf{p}$ viene dada por el promedio las probabilidades $P_{\eta(l,\mathbf{p})}(Y(\mathbf{p}) = c)$ de cada uno de los árboles del bosque. Específicamente:

$$Y'(\mathbf{p}) = \underset{c}{\operatorname{argmax}}\ p_c(\mathbf{p}) = \underset{c}{\operatorname{argmax}} \frac{1}{L} \sum_{t=1..L} P_{\eta(l,\mathbf{p})}(Y(\mathbf{p}) = c) \tag{B.4}$$

Cada árbol del bosque realiza una partición distinta del conjunto de entrenamiento original, al combinar las respuestas de cada una de los árboles se obtiene una partición más refinada y por ende una mejor clasificación que al usar un solo árbol.

$p_c(\mathbf{p})$ es el promedio de las probabilidades de ejemplos clasificados como c, y es utilizada para estimar la clasificación del ejemplo $\mathbf{p}$. Durante el entrenamiento se estima un parámetro T_c para decidir si la clasificación es correcta o no dada una medida de bondad o confianza s:

$$P(Y(\mathbf{p}) = c | Y'(\mathbf{p}) = c, p_c(\mathbf{p}) > T_c) > s \tag{B.5}$$

Por lo general se toma el parámetro s entre 60 % y 90 %. Si se tiene $p_c(\mathbf{p})$ más pequeño que T_c no se toma en cuenta la clasificación obtenida.

B.2.1. Error de clasificación

Los bosques aleatorios no necesitan procedimientos separados para estimar el error de clasificación, tales como: un conjunto de validación o un conjunto de prueba separado. Durante el proceso de entrenamiento se puede estimar internamente dicho error. Cuando se forma el conjunto de entrenamiento para cada árbol, algunos ejemplos quedan fuera (*out of bag*), por lo general, la cantidad de ejemplos que quedan fuera en cada árbol son aproximada-

mente un tercio de la cantidad total de ejemplos de entrenamiento.

Para estimar el error de clasificación primero se clasifica cada ejemplo dejado por fuera utilizando el árbol correspondiente al conjunto de entrenamiento donde el ejemplo no fue escogido. Como un ejemplo puede no ser escogido por varios árboles, este mismo puede terminar con más de una clasificación asociada. Después de haber entrenado los árboles, tomamos para cada ejemplo la clase que mayor cantidad de votos obtuvo del conjunto asociado a cada uno de ellos.

El error de clasificación se calcula tomando la relación entre la cantidad de ejemplos dejados por fuera mal clasificados y la cantidad total de ejemplos del conjunto de entrenamiento. En el caso de regresión, el error se calcula como el error cuadrático de la diferencia de los valores de los ejemplos dejados por fuera con sus valores de predicción, dividido entre el número total de ejemplos.

Existen dos conceptos fundamentales en los bosques aleatorios que influyen en el cálculo del error de clasificación, estos son: la **correlación** entre dos árboles cualesquiera del bosque y la **fuerza** de cada árbol individual.

La **correlación** determina la similitud de clasificación entre dos árboles, se quiere que los árboles tengan una correlación baja, por lo que incrementarla conlleva a un mayor error de clasificación del bosque.

La **fuerza** del árbol determina que tan bien un árbol clasifica los ejemplos, por lo que un árbol con un bajo error de clasificación es un clasificador "fuerte". Incrementar la fuerza del árbol decrementa el error del bosque.

La correlación y la fuerza se pueden controlar alterando el parámetro m que determina la cantidad de variables que serán escogidas de manera aleatoria en cada nodo del árbol. Reducir m reduce tanto la correlación como la fuerza. Por ello es importante estimar el valor del parámetro m que menor error de clasificación obtenga mediante el método de los ejemplos

que quedan por fuera. La cantidad de variables que se utilizan por nodo es el único parámetro para el cual los bosques aleatorios son sensibles.

B.3. Mejoramiento iterativo

Mejoramiento iterativo (*iterative improvement*) es un algoritmo de BLE, también llamado meta-heurística que consiste en mejorar una solución candidata de algún problema de optimización respecto a una función objetivo. El algoritmo empieza generando una solución inicial. A raíz de cada solución se generan iterativamente nuevas soluciones vecinas en el espacio de búsqueda, si se tiene que alguna solución nueva es mejor que la actual respecto a la función objetivo, ésta es reemplazada por la nueva y se repite el proceso hasta que no se encuentren mejores soluciones o hasta que el algoritmo converja respecto a alguna condición de parada.

Para generar el conjunto de soluciones vecinas, se define un operador de vecindad $V(s)$ que se aplica sobre alguna solución candidata s. Por lo general, el operador de vecindad consiste en realizar algún cambio sobre la estructura de s tal que genere un espacio de soluciones nuevas. Se define, además, una función de costos $c(s)$ que es utilizada para evaluar la bondad de una solución en particular. Una solución s se considera mejor que otra s' si se tiene que $c(s) \leq c(s')$ cuando se busca minimizar el costo. El algoritmo de mejoramiento iterativo también puede ser aplicado en problemas de optimización en el que se busque maximizar costos, sin embargo, nos enfocaremos en el uso de búsqueda local para problemas de minimización.

Este algoritmo tiene la característica de solamente intensificar en el espacio de búsqueda, no busca salirse a distintas vecindades por lo que es susceptible a converger en un mínimo local. Sin embargo, tiene la ventaja de ser muy simple y rápido en tiempo de corrida.

Existen dos variantes del algoritmo que se diferencian por la forma en que se escoge la mejor solución:

- **Primera mejor:** La primera solución que se encuentre que mejore la actual es escogida.
- **Mejor mejor:** Se explora toda la vecindad de soluciones y la mejor de todas las soluciones de la vecindad es escogida

B.4. Algoritmo Genético

Se tiene una población P, donde cada individuo es representado por una cadena o vector que codifica un conjunto de atributos. Esta cadena recibe el nombre de **cromosoma** o **genotipo** y representa una posible solución al problema, una solución parcial u otro objeto que pueda ser llevado a representar una solución. El conjunto de todos los posibles individuos es llamado espacio de genotipos G.

Al inicio del algoritmo se genera una población de forma aleatoria o a partir de alguna función heurística respecto a G. Se selecciona un subconjunto de individuos P' de P para reproducirse. La selección se realiza mediante el criterio dado por una función de adaptación o *fitness* que se aplica a cada individuo de la población y evalúa que tan buena es la solución representada por el mismo. Se crea una nueva población P'' de hijos a partir de P' mediante la aplicación de operadores de cruce y mutación.

Seguidamente se reemplaza un número de individuos de la población P por los de P''. Todo el proceso se repite un número determinado de veces o hasta que se cumpla la condición de parada.

B.4.1. Representación de las soluciones

Por lo general las soluciones (cromosomas) se representan como cadenas de bits o permutaciones de números, sin embargo, también son válidas representaciones más complejas como estructuras de árboles.

B.4.2. Métodos de selección

La selección es el paso del algoritmo genético en donde se escogen los individuos que van a reproducirse para formar nuevas generaciones. Existen distintos métodos de selección, de los cuales los más comunes son la selección por ruleta y la selección por torneo.

Selección por ruleta

Consiste en asignar una probabilidad de selección a cada individuo respecto a la adaptación de cada uno. Sea f_i el valor de adaptación del i-ésimo individuo y n la cantidad de individuos en la población, la probabilidad de selección del individuo i se define como:

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j} \tag{B.6}$$

Mediante este método se asigna a cada individuo una probabilidad proporcional a su valor de adaptación, por lo tanto individuos más fuertes tienen mayor probabilidad de ser escogidos que los individuos más débiles. Sin embargo, esto no quiere decir que siempre se escogerán los individuos más fuertes, sino que, debido a su valor de adaptación, tienen mayor ventaja de selección.

Selección por torneo

En este método de selección se toman de manera aleatoria subconjuntos (torneos) de poblaciones de un tamaño determinado y de ellos se escoge el mejor individuo de acuerdo al valor de adaptación.

Este método posee un parámetro ajustable correspondiente al tamaño del torneo, mientras más grande sea el tamaño del torneo, los individuos más débiles tienen menor probabilidad de ser escogidos para reproducirse.

B.4.3. Operador de cruce

La forma de cruce más común es el **cruce de dos padres** de donde resultan dos hijos. Para ello, por lo general, se utiliza el **cruce de un punto**, donde, se fija o se toma de forma aleatoria un punto que se utilizará de pivote en los cromosomas padres. Este punto divide dichos cromosomas de tal forma que se combina una mitad del cromosoma "padre" con una mitad del cromosoma "madre" para producir uno de los hijos, el segundo hijo se produce de la combinación de las mitades restantes. De una forma parecida existe otro tipo de cruce llamado **cruce de dos puntos**, donde se fijan dos puntos en los cromosomas padres que determinarán los puntos donde estos cromosomas se combinarán.

Si se tiene que la información codificada por los cromosomas corresponde a valores reales, es válido utilizar como operador de cruce un promedio entre los cromosomas. Sea el cromosoma $A = a_0, a_1, ..., a_n$ y $B = b_0, b_1, ..., b_n$ con $a_i^{(j)}$, se puede definir el operador de **cruce por promedio de cromosomas** como un nuevo cromosoma hijo dado por:

$$C = \frac{a_0 + b_0}{2}, \frac{a_1 + b_1}{2}, ..., \frac{a_n + b_n}{2}$$

B.4.4. Operador de mutación

El operador de mutación se utiliza para introducir una especie de ruido en el cromosoma, la idea detrás de este operador es **diversificar** en el espacio de búsqueda del algoritmo. La forma más simple de hacerlo es reemplazar aleatoriamente el valor de uno de los **genotipos** del cromosoma. De igual forma, se pueden definir operadores de mutación que alteren varios elementos o realicen distintos cambios en el cromosoma.

B.4.5. Estrategias de reemplazo

Una vez generada la nueva población, es necesario decidir que individuos serán reemplazados en la población anterior por individuos de la nueva. Entre las formas más comunes de estrategias de reemplazo tenemos:

- **Reemplazo generacional o simple:** La población de padres es reemplazada por completo por la nueva población de hijos.
- **Elitismo:** Se escogen los mejores individuos de entre la población de padres e hijos, por lo que solo los individuos con mayor valor de adaptación son los que se mantienen durante generaciones.
- **Reemplazo de estado - estable (*steady - state replacement*):** Sólo algunos individuos de la población de padres son reemplazados por los hijos, por lo general, se reemplazan los padres más débiles por hijos más fuertes de la nueva población.

Apéndice C

Experimentos y resultados auxiliares

C.1. Experimentos preliminares

En la sección C.1.1 se explican las características de cada conjunto de entrenamiento utilizado, y en la sección C.1.2 se muestra el comportamiento del algoritmo al variar sus parámetros independientemente. Este experimento fue realizado con la finalidad de escoger el conjunto de parámetros que inciden de mayor forma sobre el algoritmo para realizar la prueba de análisis de varianza.

C.1.1. Conjuntos de entrenamiento

En esta sección se describen los distintos conjuntos de entrenamiento utilizados. Los conjuntos de entrenamiento se dividen en dos grupos: Los de imágenes reales y los de imágenes sintéticas.

Imágenes reales

Este conjunto de entrenamiento está conformado por 144 imágenes obtenidas del sensor Kinect con 2 sujetos de prueba diferentes. Las imágenes se dividen en las 3 poses que se muestran en las figuras C.1,C.2 y C.3 respectivamente. Las poses fueron rotadas 360° en intervalos de 45° cada una. Cada pose fue realizada 3 veces por cada sujeto tratando de variar un poco la pose original con el objetivo de que el algoritmo pueda generalizar mejor. Las imágenes fueron etiquetadas a mano en 6 partes diferentes (cabeza, torso, brazo derecho, brazo izquierdo, pierna derecha y pierna izquierda).

Figura C.1: Primera pose.

Figura C.2: Segunda pose.

Figura C.3: Tercera pose.

Este conjunto se subdividió a su vez en otros tres conjuntos separados, correspondientes a poses vistas desde cierto ángulo en específico, con el fin de medir el desempeño del clasificador con un conjunto de imágenes más reducido y de una misma perspectiva. La subdivisión consiste en lo siguiente:

- Imágenes de frente (35 imágenes)
- Imágenes de lado izquierdo (18 imágenes)
- Imágenes de lado derecho (18 imágenes)

Cada uno está conformado por 18 imágenes a excepción del conjunto de imágenes de frente, que contiene 17 imágenes extra de las mismas características utilizadas en pruebas anteriores.

Imágenes sintéticas

Para generar los datos sintéticos se utilizó un modelo 3D de una persona que fue cargado en ***Autodesk Maya*** y posteriormente etiquetado con el mismo programa. El modelo fue alterado eliminando y suavizando detalles del mismo para convertirlo en uno lo más general posible. Finalmente, para el etiquetamiento se asignó un color distinto a cada parte del cuerpo que se desea detectar.

Para generar las imágenes sintéticas se cargaron los ejemplos provistos en la base de datos de captura de movimiento de CMU en ***Adobe Motion Builder***. Posteriormente, se le asoció esta información al modelo 3D descrito anteriormente.

Se utilizó la captura número 2 del sujeto número 7 en la acción "caminando" de la base de datos de CMU. Se escogió esta prueba ya que la acción de caminar representa un conjunto de poses sencillas y comunes de observar por la cámara. Se sintetizaron 128 imágenes en intervalos de 5 cuadros para no obtener imágenes muy similares. Todo el movimiento se repitió rotando la cámara en intervalos de 45°.

La rotación se realiza con el fin de que el algoritmo aprenda distintos puntos de vista de una persona, y así, poder clasificar correctamente haciendo uso de varios sensores *Kinect*, colocados en distintos ángulos de la escena. La escena del personaje realizando los movimientos se exportan en dos formatos distintos con la información de profundidad y la información del etiquetamiento. En la figura C.4 se muestra el modelo con las partes del cuerpo etiquetadas (representadas por colores) y en la figura C.5 se muestra una imagen de profundidad donde los píxeles más cercanos a la cámara son más claros que los que se encuentran más lejos.

Figura C.4: Imagen con las partes del cuerpo etiquetadas.

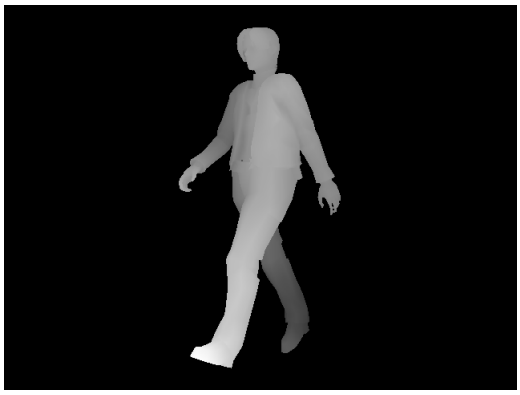

Figura C.5: Imagen con la profundidad de la escena.

Las imágenes sintéticas deben reflejar no solo variedad en pose sino en forma, tamaño de la

persona y posicionamiento de la cámara ya que éstas son variaciones que pueden presentarse al momento de prueba dependiendo de la persona y condiciones de la escena. Las variaciones de la distancia entre la persona y la cámara se encuentran codificadas dentro del algoritmo, por lo que no es necesario generar imágenes de entrenamiento que manejen esos casos.

C.1.2. Comportamiento al variar los parámetros

Las siguientes gráficas muestran el comportamiento del algoritmo al variar cada uno de sus parámetros individualmente. Para estos resultados se entrenó el algoritmo utilizando el conjunto de entrenamiento de imágenes reales vistas de frente descrito en la sección C.1.1. Se apartó un grupo equivalente al 20 % del total de las imágenes como conjunto de **prueba** para evaluar la generalización del algoritmo una vez entrenado.

Cada parámetro fue alterado dentro de un rango de valores para observar su influencia en el desempeño final del algoritmo. La correlación de los parámetros no es lineal, sin embargo, se realizó esta prueba como base para seleccionar aquellos parámetros que afecten más la clasificación del algoritmo. El objetivo de este entrenamiento es realizar un análisis de la varianza sobre los parámetros de mayor interés. Para todas las pruebas, se fijaron los parámetros en los valores por defecto mostrados en el cuadro C.1.

Rango para valores de desplazamientos	150
Rango para valores de umbrales	3
Número de árboles	3
Profundidad del árbol	10
Número de píxeles tomados para entrenar	2000
Número de desplazamientos a generar	1500
Número de umbrales a generar	25

Cuadro C.1: Parámetros por defecto.

Las figuras están agrupadas en pares. La primera imagen muestra el porcentaje de clasificación al variar uno de los parámetros en el conjunto de prueba y en el conjunto de entre-

namiento y la segunda muestra el consumo de tiempo en el entrenamiento.

En las figuras C.6 y C.7 se alteran los rangos para los desplazamientos, en las figuras C.8 y C.9 se alteran los rangos de valores para los umbrales, en las figuras C.10 y C.11 se varia la cantidad de árboles entrenados, en las figuras C.12 y C.13 se altera la profundidad de los árboles, en las figuras C.14 y C.15 se altera el número de píxeles seleccionados en el entrenamiento, en las figuras C.16 y C.17 se altera la cantidad de desplazamientos generados y en las figuras C.18 y C.19 se altera la cantidad de umbrales generados.

Se puede observar que parámetros como: **número de umbrales**, **número de desplazamientos** y **rango de umbrales**, no afectan de gran manera el desempeño del algoritmo, es por ello que se deciden fijar esos parámetros y estudiar aquellos de mayor incidencia como lo son: **rango de desplazamientos**, **Número de árboles** y **profundidad de los árboles**.

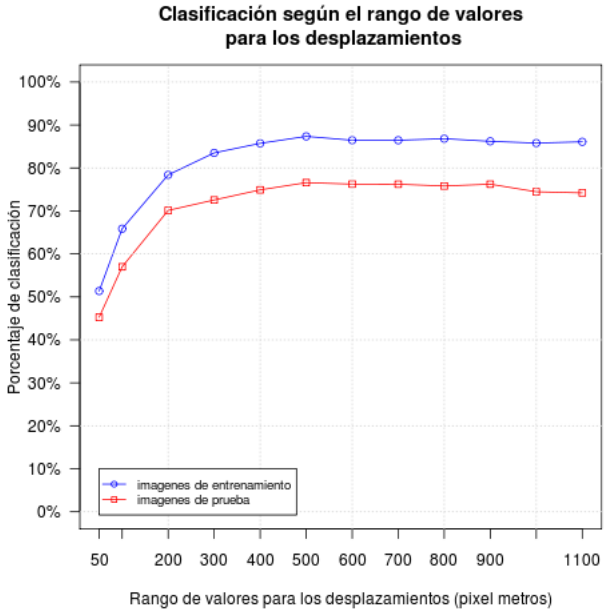

Figura C.6: Porcentaje de clasificación del conjunto de prueba y entrenamiento al alterar los rangos de valores para los desplazamientos

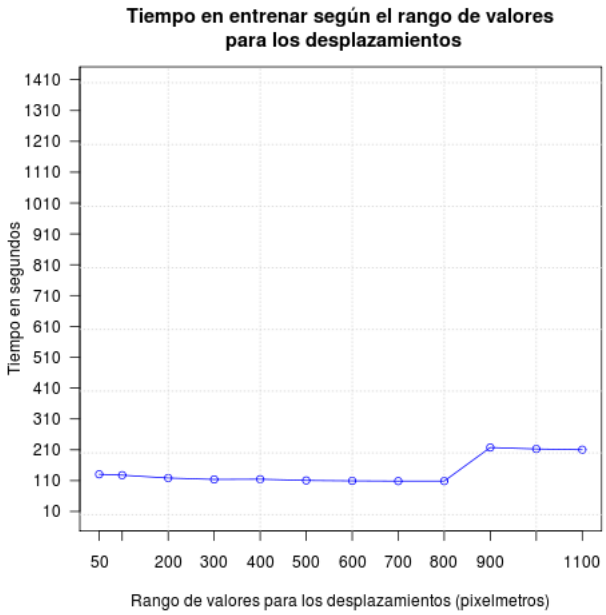

Figura C.7: Tiempo consumido en el entrenamiento al alterar los rangos de valores para los desplazamientos

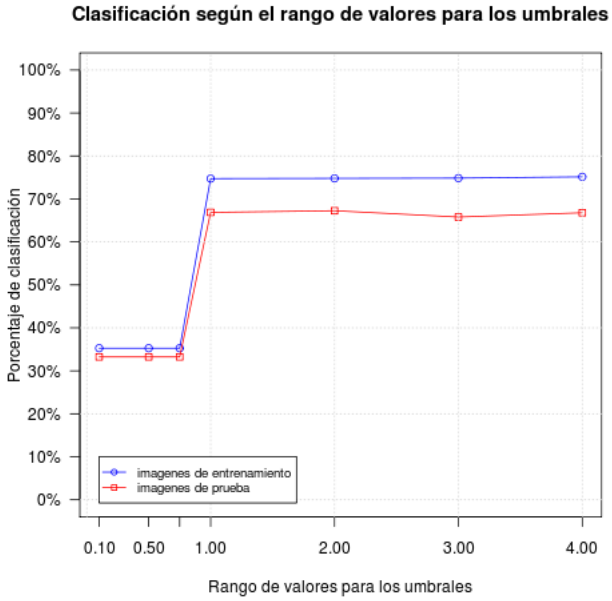

Figura C.8: Porcentaje de clasificación del conjunto de prueba y entrenamiento al alterar los rangos de valores para los umbrales

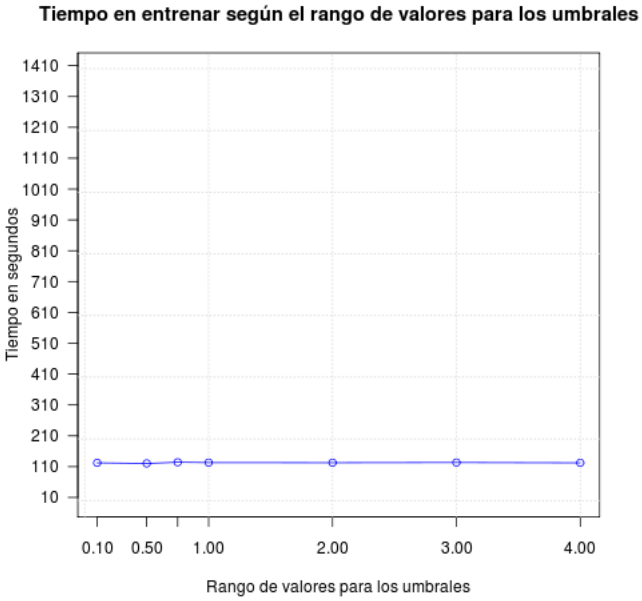

Figura C.9: Tiempo consumido en el entrenamiento al alterar los rangos de valores para los umbrales

C.1.3. Análisis de varianza

En esta sección se muestran los resultados de los experimentos realizados para el análisis de la varianza en los parámetros de interés. En la sección C.1.2 se realizaron diferentes corridas variando cada parámetro individualmente para seleccionar los parámetros que tengan mayor influencia sobre el desempeño del algoritmo. Como resultado de las corridas se seleccionaron los parámetros que se mencionan a continuación:

- Número de árboles
- Profundidad de los árboles
- Número de píxeles por imágenes

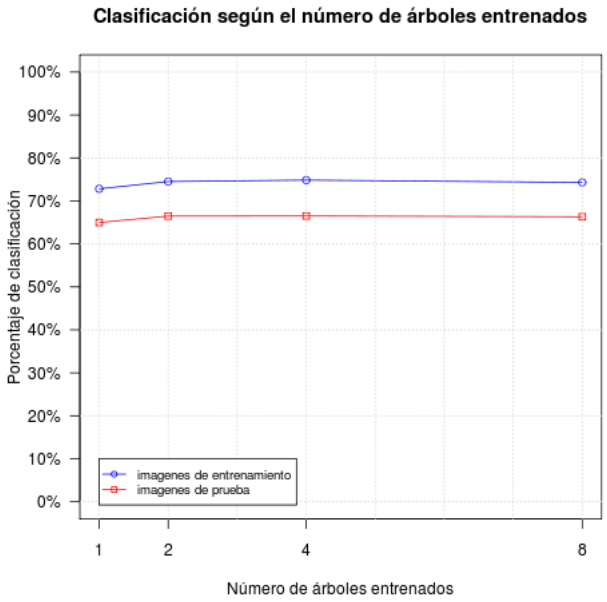

Figura C.10: Porcentaje de clasificación del conjunto de prueba y entrenamiento al alterar la cantidad de árboles

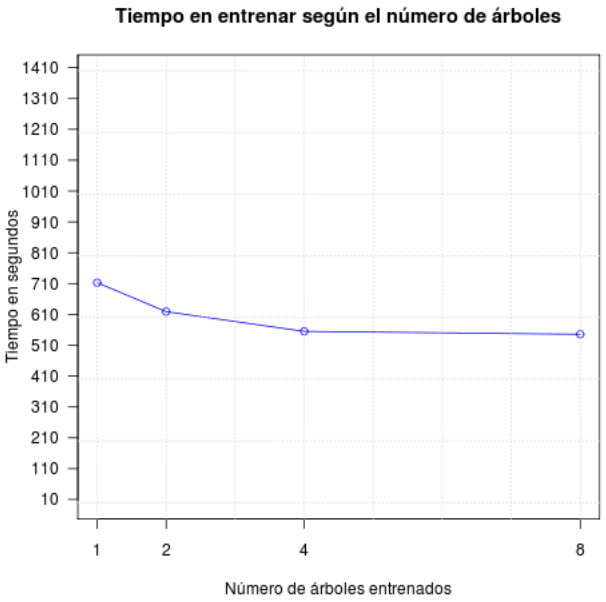

Figura C.11: Tiempo consumido en el entrenamiento al alterar la cantidad de árboles

Se realizaron corridas para todas las combinaciones de un conjunto de posibles valores para cada uno de los tres parámetros. Los valores probados son: **2**, **4**, **6**, **8** y **10** árboles,**9**, **12**, **15**, **18** y **21** niveles de profundidad para los árboles y **5000**, **10000**, **15000**, **20000** y **25000** píxeles por imagen. El resto de los parámetros fueron fijados con los valores que se muestran en el cuadro C.2.

Rango para valores de desplazamientos	500
Rango para valores de umbrales	2
Número de desplazamientos a generar	1500
Número de umbrales a generar	25

Cuadro C.2: Valores de parámetros fijos para el análisis de varianza.

Como resultado del análisis de la varianza se obtuvo la tabla **ANOVA** mostrada en el cuadro C.3. Como se puede observar, la variable con menor nivel de significación es la **profundidad de los árboles** seguida por el **número de píxeles por imagen**. En la

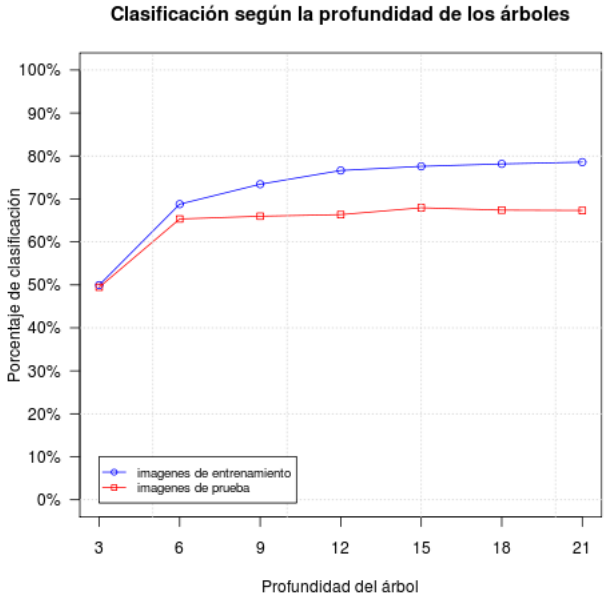

Figura C.12: Porcentaje de clasificación del conjunto de prueba y entrenamiento al alterar la profundidad de los árboles entrenados

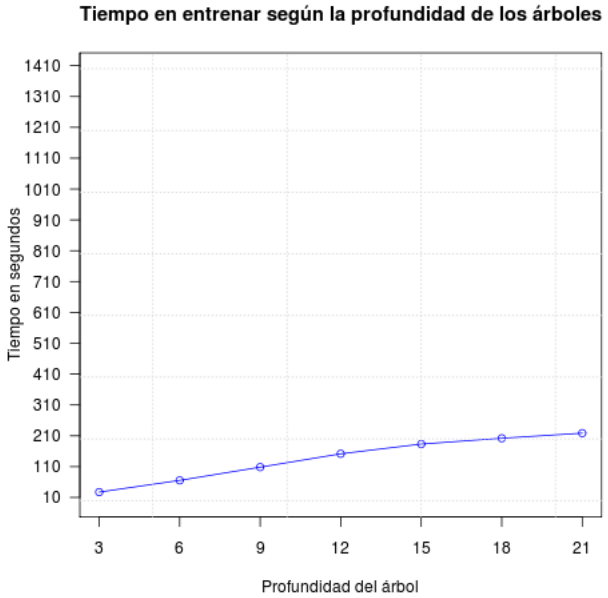

Figura C.13: Tiempo consumido en el entrenamiento alterando la profundidad de los árboles entrenados

figura C.20 se muestra el porcentaje de clasificación para diferente número de árboles con diferente profundidad y en la figura C.21 se muestra el porcentaje de clasificación para árboles con diferente número de píxeles por imagen y diferente profundidad. En ambas imágenes se puede observar claramente como la profundidad influye principalmente en el porcentaje de clasificación.

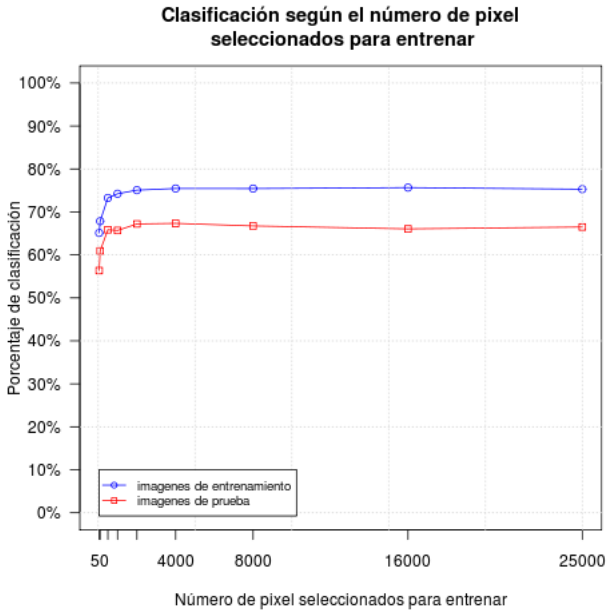

Figura C.14: Porcentaje de clasificación del conjunto de prueba con el algortimo entrenado con diferente número de píxeles.

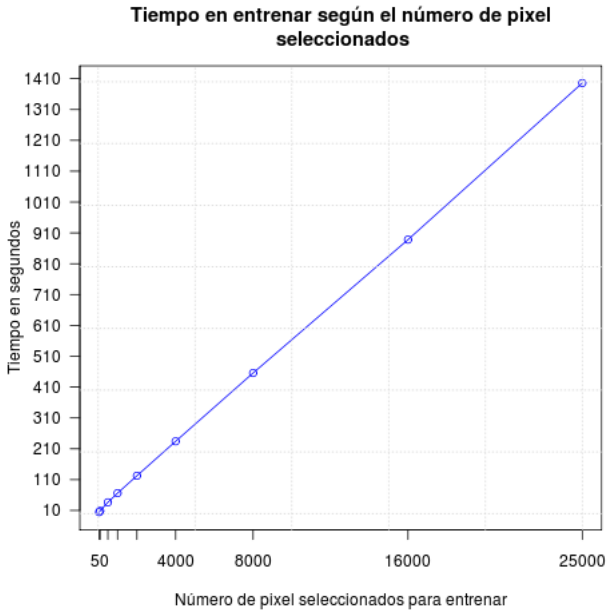

Figura C.15: Tiempo consumido al entrenar tomando diferentes números de píxeles.

Grupos	G.L.	SS	MS	F	p-valor
NA	1	0.003	0.003	0.0050	0.94369
PA	1	117.576	117.576	187.0816	< 2e-16 ***
NP	1	2.871	2.871	4.5688	0.03464 *
NA : PA	1	0.475	0.475	0.7563	0.38627
NA : NP	1	0.488	0.488	0.7772	0.37981
PA : NP	1	0.009	0.009	0.0147	0.90362
NA : PA : NP	1	0.071	0.071	0.1134	0.73686
Niveles de significación: 0 ‘***’0.001 ‘**’0.01 ‘*’0.05 ‘.’0.1 ‘ ’1					

Cuadro C.3: Análisis de la varianza para el número de árboles (NA), la profundidad de los árboles (PA) y el número de píxeles por imagen (NP)

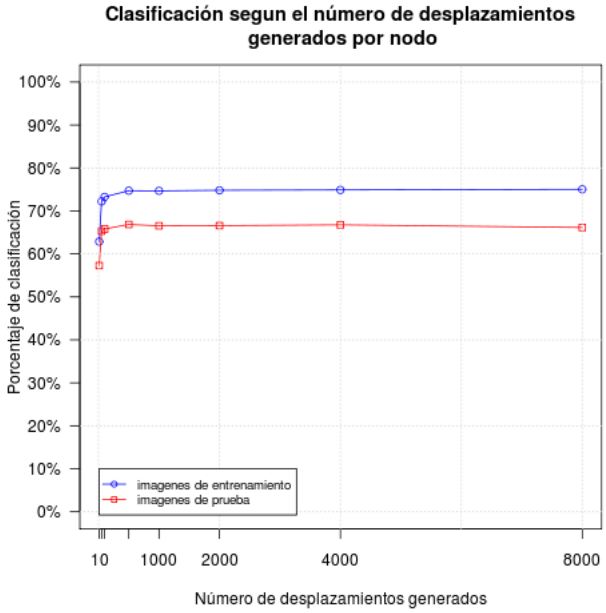

Figura C.16: Porcentaje de clasificación del conjunto de prueba y entrenamiento por árboles entrenados con distinto número de desplazamientos

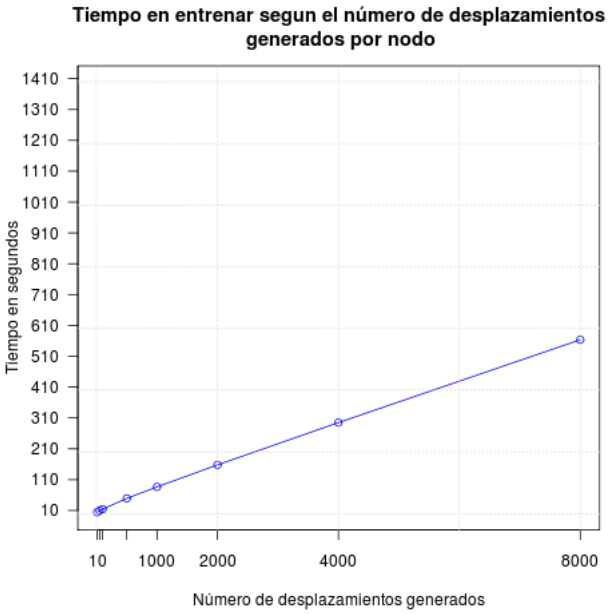

Figura C.17: Tiempo consumido en el entrenamiento por árboles entrenados con distinto número de desplazamientos

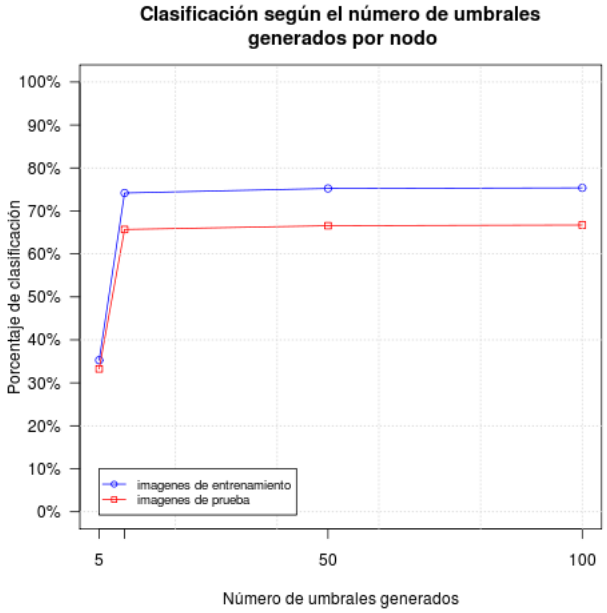

Figura C.18: Porcentaje de clasificación del conjunto de prueba y entrenamiento por árboles con distinto número de umbrales

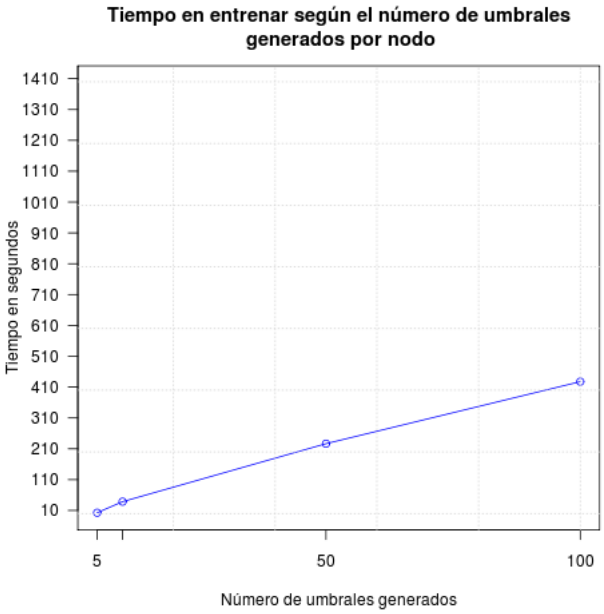

Figura C.19: Tiempo consumido en el entrenamiento por árboles con distinto número de umbrales

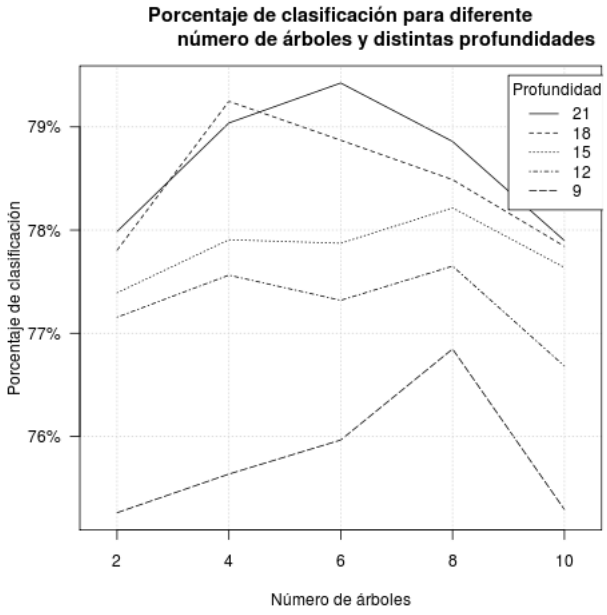

Figura C.20: Porcentaje de clasificación para diferentes números de árboles con diferentes profundidades.

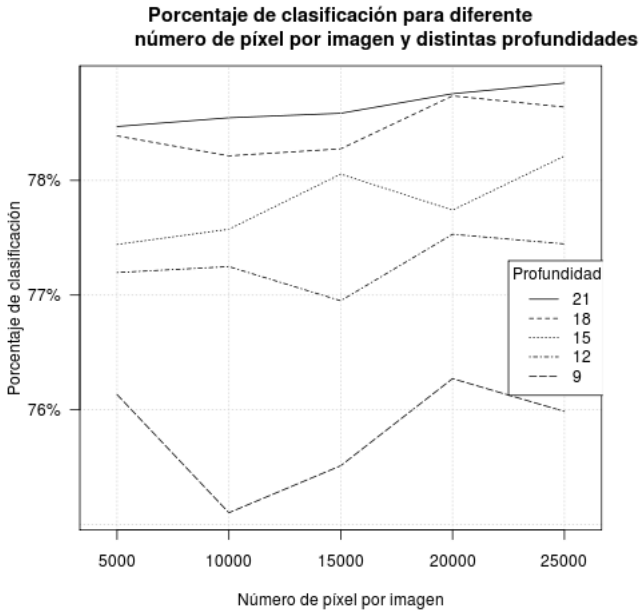

Figura C.21: Porcentaje de clasificación para diferente cantidad de píxeles por imagen y diferentes profundidades.