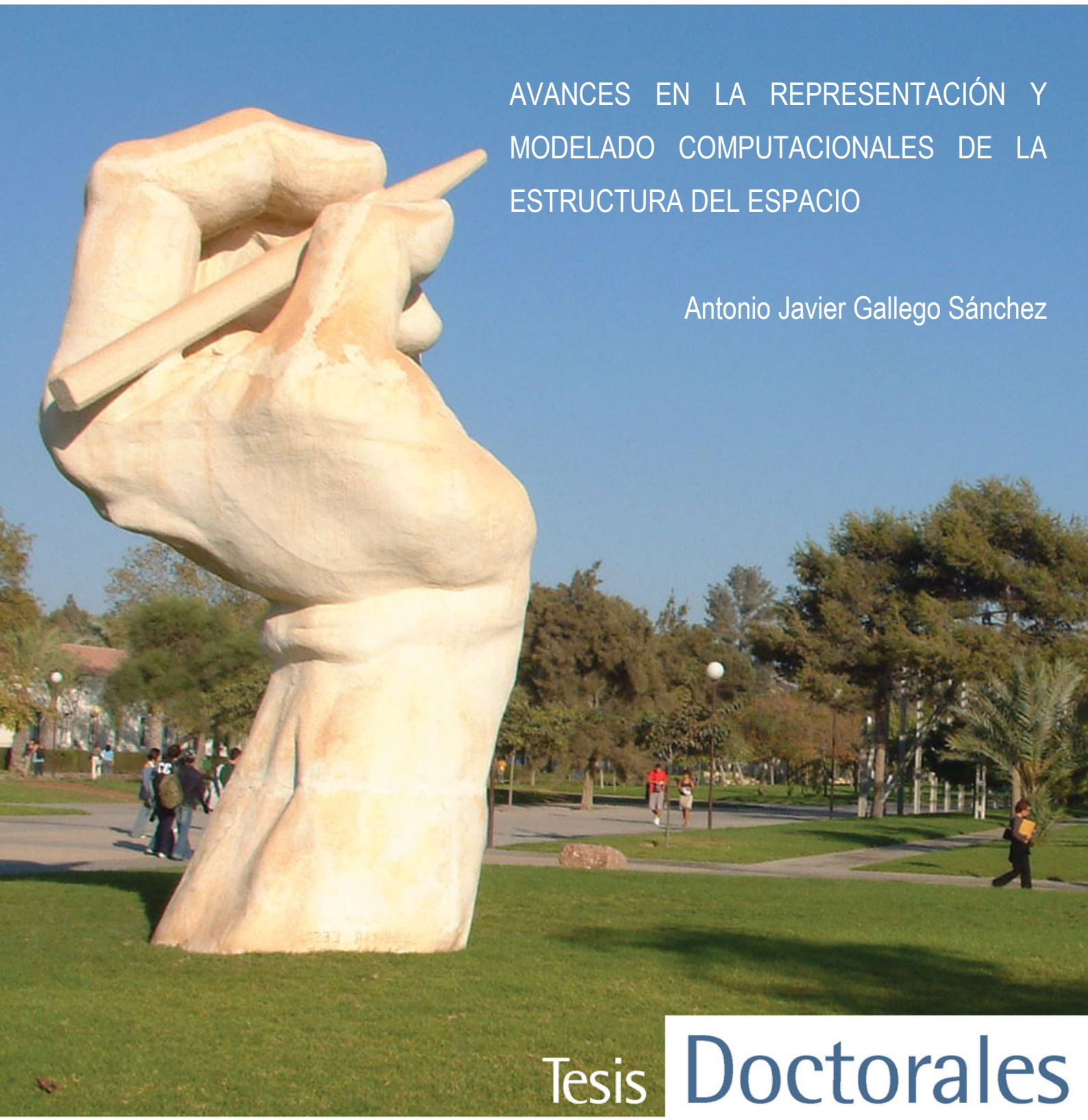
Universitat d'Alacant
Universidad de Alicante

AVANCES EN LA REPRESENTACIÓN Y MODELADO COMPUTACIONALES DE LA ESTRUCTURA DEL ESPACIO

Antonio Javier Gallego Sánchez



Tesis Doctorales

www.eltallerdigital.com

UNIVERSIDAD de ALICANTE

UNIVERSIDAD DE ALICANTE

DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN
E INTELIGENCIA ARTIFICIAL

TESIS DOCTORAL:

AVANCES EN LA REPRESENTACIÓN Y MODELADO COMPUTACIONALES DE LA ESTRUCTURA DEL ESPACIO

Autor:
Antonio Javier Gallego Sánchez

Director:
Dr. Rafael Molina Carmona

Alicante, diciembre de 2012

TESIS DOCTORAL

EN FORMA DE COMPENDIO DE PUBLICACIONES

AVANCES EN LA REPRESENTACIÓN Y MODELADO COMPUTACIONALES DE LA ESTRUCTURA DEL ESPACIO

Este documento contiene la síntesis del trabajo realizado por Antonio Javier Gallego Sánchez, bajo la dirección del doctor Rafael Molina Carmona, para optar al grado de Doctor en Informática. Se presenta en la Universidad de Alicante y se estructura según la normativa establecida para la presentación de tesis doctorales en forma de compendio de publicaciones: una primera parte con un resumen de la labor realizada y una segunda que incluye las divulgaciones científicas que de ella surgieron.

Alicante, diciembre de 2012

Agradecimientos

Me gustaría expresar mi más sincero agradecimiento a todas aquellas personas que de un modo u otro me han ayudado a llevar a cabo este proyecto. En primer lugar quisiera agradecer el apoyo ofrecido por todos mis compañeros del departamento de Ciencia de la Computación e Inteligencia Artificial así como al grupo de investigación de Reconocimiento de Formas e Inteligencia Artificial. En especial a mi director de tesis Rafael Molina Carmona, por haberme ofrecido esta oportunidad y sacar siempre tiempo para ayudarme.

También agradecer a los coautores de los artículos presentados, por el tiempo y aprendizajes compartidos, Gabriel López, Jorge Calera, Damián López, Juan Luis Dalmau, Patricia Compañ, Carlos Villagrá y Pilar Arques.

Y por supuesto estas últimas líneas de agradecimientos están dedicadas a mis padres, mi hermana y mi amiga Nela, que sin duda han estado junto a mí desde el primer momento, ofreciéndome su apoyo incondicional sin el cual jamás habría podido llegar hasta aquí.

Índice general

Parte I

Resumen

Capítulo 1

Introducción

El origen y motivación inicial de este trabajo se encuentra en las nuevas posibilidades que nos ofrecen los dispositivos tecnológicos actuales para ayudarnos en nuestra vida, proporcionando una asistencia eficaz y con bajo coste. Esto es especialmente interesante en determinados contextos, como cuando la ayuda humana no es posible por las condiciones del entorno, cuando existen problemas de discapacidad en los que una ayuda de este tipo puede suponer la mejora de la calidad de vida y de la independencia de la persona o cuando un asistente automático supone una disminución importante de los costes económicos.

Muchos de estos sistemas de asistencia necesitan conocer la estructura del espacio en el que se desenvuelven. Por ejemplo, en sistemas automáticos de navegación, en sistemas de ayuda a la movilidad de personas con discapacidad visual, o en aplicaciones de realidad aumentada, es imprescindible tener un modelo del mundo real y de la estructura del espacio que los rodea. Una de las líneas más activas de investigación en el ámbito de la Inteligencia Artificial es el uso de dispositivos de percepción visual (diferentes cámaras y dispositivos de captación de imágenes) para esta labor.

En este contexto se presenta este trabajo, cuyo eje principal parte de la siguiente hipótesis: es posible utilizar dispositivos de percepción visual como sensores principales para identificar la estructura tridimensional del espacio real, y así poder reconstruirlo, y clasificar y etiquetar los objetos que en él aparecen. A partir de esta hipótesis se plantea la construcción de un sistema en el que confluyen los resultados de investigación de varios campos, principalmente el estudio de los modelos de representación del espacio, junto con todas las técnicas y pro-

cedimientos que dan soporte a esta disciplina, entre ellos, y de forma destacada, la visión artificial.

Las técnicas de visión artificial, también conocida como visión computacional, tratan de emular el sistema de visión humano y sus capacidades mediante un ordenador, de forma que mediante la interpretación de las imágenes adquiridas, por ejemplo mediante una cámara, se puedan reconocer los diversos objetos en el ambiente y su posición en el espacio. La facilidad con la que el ser humano realiza este proceso, llevó a pensar a los primeros investigadores en este campo (hacia 1960) que hacer que un ordenador interpretase imágenes sería relativamente sencillo. Pero no resultó así, muchos años de investigación han demostrado que es un problema esencialmente complejo y que aún en la actualidad sigue sin estar resuelto.

La visión es un actividad que llevamos a cabo de forma inconsciente y automática, sin embargo se estima que empleamos entre el 70% y el 75% del procesamiento del cerebro en el análisis de la información visual. La automaticidad de esta acción dificulta su análisis de forma científica con la intención de averiguar los procesos que intervienen en ella e intentar imitarlos en un ordenador.

Uno de los principales obstáculos con el que se encuentra la visión artificial es que la percepción del entorno se realiza a partir de proyecciones en imágenes de dos dimensiones. La reducción de una dimensión produce la pérdida de gran cantidad de información, lo que eleva notablemente la dificultad del proceso de visión.

Para intentar solventar este tipo de problemas aparecen técnicas como la visión estéreo. Estos sistemas tratan de imitar la capacidad visual que tiene el ser humano, y otros seres vivos, de extraer información del entorno que les rodea y estimar la distancia hasta los objetos a partir de imágenes binoculares. Las cámaras de visión estéreo se componen de dos sensores de características similares montadas a una pequeña distancia horizontal. De esta forma no es necesario realizar el reconocimiento de características ni la interpretación de lo que se está captando en ese momento; sino que se obtiene la información tridimensional a partir de las diferencias existentes entre las imágenes obtenidas.

El análisis de las imágenes adquiridas simultáneamente por los dos sensores debe permitir establecer una correspondencia entre los puntos de ambas imágenes y realizar el cálculo de la profundidad. El proceso de correspondencia se realiza a partir de algoritmos que tratan de localizar las proyecciones de un punto de la escena en las dos imágenes capturadas. La principal limitación en este cálculo

es que un único píxel de una imagen no proporciona información suficiente para identificar el punto correspondiente en la otra. Las soluciones que aplican estos algoritmos son analizar los píxeles de su vecindad por medio de técnicas de correlación, detección y emparejamiento de bordes así como de segmentación de regiones, entre otras.

El primer bloque de las aportaciones presentadas se centra en el área de la visión estéreo. En concreto, con el objetivo de mejorar las imágenes necesarias para la reconstrucción y modelado del espacio, se ha trabajado en un sistema de visión estereoscópica basado en segmentación. La principal aportación presentada es la correspondencia basada en regiones, para las que se define un vector de características que utiliza la posición, el tamaño, el color y la forma de las regiones en la función de emparejamiento. Posteriormente realiza un proceso de extracción de capas para obtener objetos más consistentes. Se complementa con el desarrollo de un factor de corrección empírico que permite reducir las aberraciones producidas por la lente de la cámara. Este sistema permite detectar los objetos de la escena y estimar su profundidad, por lo que los resultados obtenidos sirven como base para el resto de trabajos presentados. Todo esto da lugar a un sistema más robusto y preciso en los resultados, con unos tiempos de cómputo muy razonables.

Dentro de las técnicas de visión artificial, también se presentan un par de métodos para la detección de los objetos que aparecen en la imagen: Una primera solución basada en el color y la forma de los segmentos extraídos durante el proceso de correspondencia; y una segunda propuesta para la detección robusta y precisa de la orientación y la localización de objetos en imágenes de bajo contraste. La principal aportación es la introducción de un proceso de segmentación con auto-umbral, el uso de un nuevo tipo de histograma de forma y de un método de regresión robusta que permiten obtener una mayor precisión y robustez en los resultados. Los resultados obtenidos muestran un coste temporal muy bajo, lo que permite su uso en entornos muy exigentes, como el propuesto en la reconstrucción del espacio, o en entornos industriales.

El bloque central de las aportaciones de esta tesis son los avances en la definición de un modelo computacional adecuado para la representación del espacio. En estos trabajos se presenta un modelo de enumeración espacial que posibilita la reconstrucción de una escena a partir de imágenes, en este caso, estereoscópicas. Este trabajo se enfrenta a un primer reto: la necesidad de rectificar las imágenes para corregir la perspectiva cónica producida por la cámara. El efecto de

perspectiva se produce al realizar la proyección de puntos tridimensionales del espacio (la escena real) en puntos bidimensionales de un plano (la imagen adquirida por la cámara). Este efecto produce que el tamaño de representación de un objeto varíe según su profundidad, y por lo tanto, impide el uso de esta información de forma directa en la reconstrucción. Se presentan varias soluciones para resolver este problema y como resultado se consigue una rectificación de la imagen que permite una reconstrucción muy cercana a la realidad. A partir de la rectificación, se realiza el proceso de reconstrucción mediante etiquetado del espacio (enumeración espacial), además se aplica un filtro cúbico que mejora el resultado utilizando información redundante.

A partir de los resultados obtenidos en la reconstrucción, se propone su aplicación al problema del mapeado del espacio utilizando un robot durante el proceso de adquisición de la secuencia de imágenes.

Una vez obtenida la estructura de una escena, se hace necesario dotar al modelo de representación de mayor potencia, estableciendo un modelo que recoja las relaciones entre los objetos del espacio y que permita, además, una representación más compacta. En este sentido, se propone un lenguaje de modelado basado en gramáticas que tiene diversas ventajas: permite la representación del espacio en forma de cadena perteneciente a un lenguaje y facilita la expresión de relaciones entre los componentes de la cadena. Además, el modelo así definido, también posibilita la obtención de escenas sintéticas (incluyendo un sistema gráfico) y la evolución de las cadenas para dotar a la escena de dinamismo y comportamiento inteligente.

Por otro lado, se da un paso más y se emparenta al modelo con los sistemas multi-agente (SMA), al detectar los puntos en común y constatar las ventajas de utilizar un modelo formal bien definido como el de los SMA. El sistema propuesto se valida con varios ejemplos de utilización, entre los que destaca la construcción de un sistema de navegación para un robot móvil.

En el último bloque se da un primer paso hacia la conexión entre el lenguaje de modelado propuesto y la reconstrucción de escenas mediante visión estéreo. Es decir, a partir de la representación gráfica de una escena (el modelo de datos basado en enumeración) se pretende obtener las posibles cadenas del lenguaje de modelado que lo describan. Para realizar este proceso en primer lugar necesitamos una estructura de representación de los objetos del espacio a partir del cual podamos inferir las cadenas. Una de las estructuras de datos que más potencia de representación proporcionan son los grafos. Prueba de esto son la amplia

variedad de campos en las que se utilizan, como por ejemplo la biología, sociología, numerosos algoritmos matemáticos e informáticos como el problema del viajante, o especialmente las tareas de reconocimiento de patrones.

En esta última parte se aborda la tarea del aprendizaje de grafos mediante un lenguaje *k*-testable. Estos lenguajes son ampliamente conocidos y utilizados en cadenas y árboles, pero todavía no se había dado el paso a su utilización en grafos. En este trabajo se extiende la definición de *k*-testabilidad a grafos acíclicos dirigidos, y se propone un algoritmo gramatical de inferencia con complejidad polinómica para el aprendizaje de esta clase de grafos. Estos avances nos permiten identificar la pertenencia de grafos a un lenguaje *k*-testable o la posterior generación o inferencia de grafos que pertenezcan al lenguaje.

Para el proceso de experimentación de los lenguajes *k*-testables se proponen tres nuevos algoritmos que permiten extraer y representar la estructura de una imagen a partir de grafos acíclicos dirigidos. El primer método propuesto crea caminos de vecindad dentro de la representación q-tree de una imagen, proporcionando nuevos datos y nuevas posibilidades de operación y aplicación. El segundo algoritmo trata la imagen como una rejilla estructural, de forma que tras un postproceso se obtiene un grafo acíclico dirigido. El último método utiliza un esqueleto para construir el grafo. La principal aportación de este trabajo es la extensión del uso de grafos en otras aplicaciones, especialmente en las tareas de reconocimiento de patrones.

Resumiendo, aunque los trabajos presentados se corresponden a ámbitos distintos de investigación, existe un hilo conductor entre todos ellos, con un fin último, que es el avance en la representación y modelado computacionales de la estructura del espacio real.

Los resultados obtenidos han tenido impacto a dos niveles: con publicaciones en congresos internacionales y revistas, y como origen de un grupo de trabajo en este tema. De forma que dos nuevas líneas de investigación se encuentran en desarrollo a partir de algunos de los aspectos aquí señalados: la definición más completa del modelo gramatical de representación del espacio y su aplicación a casos reales; y el estudio del enlace entre los dos modelos, de enumeración espacial y gramatical, a partir de los resultados obtenidos en los lenguajes *k*-testables o mediante otras propuestas como las gramáticas evolutivas.

En los siguientes capítulos se detalla de forma más extensa el contenido de cada uno de estos trabajos. En la segunda parte se incluye el compendio de divulgaciones científicas agrupado por temáticas para facilitar su lectura.

Capítulo 2

Visión artificial

En este primer capítulo se resume el contenido de los trabajos "*Detección de objetos y estimación de su profundidad mediante un algoritmo de estéreo basado en segmentación*" (página 81) y "*Improving edge detection in highly noised sheet-metal images*" (página 97) sobre visión estéreo, cálculo de mapas de disparidad y detección de objectos. En primer lugar se realiza una breve introducción a la visión estéreo y a continuación se resume el contenido de las publicaciones citadas.

2.1 Introducción a la visión estéreo

La visión estéreo modela la capacidad visual que tiene el ser humano, y muchos otros animales, de extraer información del entorno que le rodea. Así, el ser humano, a partir de un proceso binocular transforma dos imágenes que se forman en las retinas de cada ojo en una visión espacial del conjunto. Este sistema no necesita realizar el reconocimiento de características ni la interpretación de lo que está captando en ese momento; sino que obtiene la información tridimensional a partir del desplazamiento lateral entre las dos imágenes retinales correspondientes a una escena del mundo real. Esto es posible gracias a que tenemos dos ojos separados, cuyos campos visuales se solapan en la región central de la visión. Por ello, un mismo punto del entorno se proyecta en la retina en posiciones desplazadas según lo cerca o lejos que esté el objeto visualizado del punto de fijación [1, 2].

Las técnicas de visión estéreo por computador tratan de emular el sistema de

visión humano, en el que se analizan las diferencias de la proyección de la escena en dos imágenes tomadas desde dos posiciones diferentes [2]. A las diferencias entre ambas imágenes se las denomina disparidad, y su análisis permite obtener la dimensión perdida en la proyección de la escena tridimensional en la imagen bidimensional [3].

En el caso de los seres humanos el cerebro es el encargado de combinar ambas imágenes y de calcular su disparidad. En el caso de un sistema de visión por computador, el análisis de las imágenes adquiridas simultáneamente por dos sensores distanciados espacialmente debe permitir establecer una correspondencia entre los puntos de ambas imágenes y realizar el cálculo de la profundidad.

El proceso de correspondencia se realiza a partir de algoritmos que tratan de localizar las proyecciones de un punto de la escena en las dos imágenes capturadas. La principal limitación en este cálculo es que un único píxel de una imagen no proporciona información suficiente para identificar el punto correspondiente en la otra. Las soluciones que aplican estos algoritmos son analizar los píxeles de su vecindad por medio de técnicas de correlación, detección y emparejamiento de bordes así como de segmentación de regiones [4], entre otras.

El cálculo de la profundidad se basa en el conocimiento de la geometría del sistema de visión por computador, especialmente en la posición que ocupa cada una de las cámaras en el espacio. En el caso de que las dos cámaras estén perfectamente alineadas, configuración ideal del sistema, para calcular las coordenadas tridimensionales de un punto de la escena se aplican relaciones entre triángulos semejantes obtenidos de la geometría del sistema. En situaciones en que la geometría del sistema no cumpla la configuración ideal se debe modelar la transformación que liga los sistemas de coordenadas de ambas cámaras, o bien aplicar algún algoritmo de rectificación a las dos imágenes que proporcione las mismas imágenes en la configuración ideal [5].

2.2 Cálculo de la distancia

En nuestro caso nos centraremos en el supuesto básico de dos imágenes pertenecientes a una escena, obtenidas mediante una cámara estéreo de objetivos paralelos. De este modo el área de búsqueda de cada píxel de la imagen izquierda (imagen de *referencia*) se reduce a su misma fila en la imagen derecha. Esta propiedad se denomina “restricción epipolar” y se debe a que los ejes ópticos de las cámaras son paralelos.

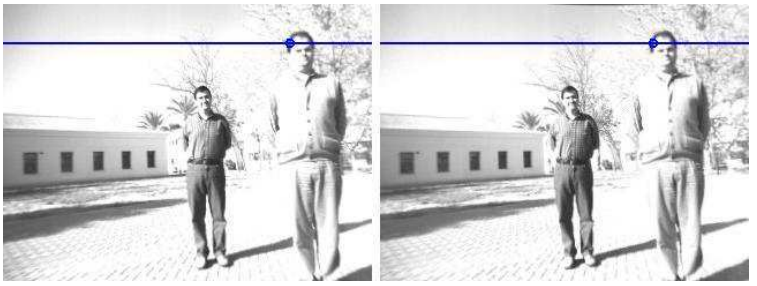


Figura 2.1: *Ejemplo de un par estéreo y una línea epipolar.*

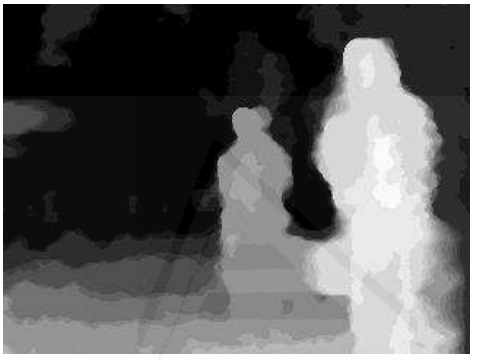


Figura 2.2: *Imagen de disparidad obtenida a partir del par de imágenes de la figura 2.1*

A una fila de la imagen de referencia y su correspondiente fila en la otra imagen se le denomina línea epipolar (figura 2.1). Cada uno de los píxeles de esta línea epipolar tendrá su correspondiente en la otra imagen, pero estará situado en una columna diferente debido a la separación de las cámaras. La diferencia absoluta entre las columnas se denomina disparidad. Cuanto mayor sea la distancia a la que se sitúa el objeto de la cámara menor será la disparidad y viceversa, o dicho de otra forma, la disparidad es inversamente proporcional a la distancia a la que se encuentran los objetos percibidos [3].

Puesto que la disparidad es un valor entero, es posible representarla en forma de imagen, llamada imagen de profundidad o de disparidad (figura 2.2), asociando a cada píxel un valor de gris. Las tonalidades más claras representan disparidades más altas (objetos cercanos) y las más oscuras disparidades más bajas (objetos lejanos).

Existe una relación inversa entre el valor de disparidad y la distancia a la que se encuentra el objeto de la cámara. De esta forma, la disparidad es una medida de profundidad, que en función de la geometría de la cámara, se puede traducir a coordenadas dentro de un espacio Euclídeo donde el centro es el objetivo de la imagen de referencia. La ecuación 2.1 muestra el cálculo de la distancia a partir de la disparidad y de los parámetros de la cámara [11, 6, 26, 27].

$$Z = f\frac{T}{d} \tag{2.1}$$

Donde:

- Z es la profundidad (distancia de la cámara al objeto).
- T es la longitud de la línea base (distancia entre los centros ópticos).
- f es la distancia focal (distancia entre el centro óptico de la lente y el punto focal).
- d es la disparidad del objeto (representado mediante tonos de gris en la imagen).

2.3 Modelo de visión estéreo generalizado

En un modelo de visión estéreo generalizado se suelen realizar los siguientes pasos, desde que se adquieren las imágenes mediante la cámara estéreo, hasta que se genera el mapa de disparidad y se reconstruye:

1. *Captura*: El primer paso es obtener un par de imágenes estéreo.
2. *Rectificación epipolar*: Este proceso transforma las imágenes estéreo para que las líneas epipolares coincidan con las líneas horizontales de rastreo, de esta forma se reduce la complejidad de implementación de los algoritmos de emparejamiento. En el caso de utilizar una cámara estéreo de objetivos paralelos no será necesario realizar esta rectificación.
3. *Preprocesado*: Esta fase consiste en cualquier preprocesado de las imágenes, que no sea la rectificación ya comentada, y que se realice antes del emparejamiento. Algunos ejemplos de preprocesamiento que se emplean

comúnmente en los sistemas de visión estéreo son la extracción de características que serán emparejadas, como bordes, regiones u otro tipo de transformaciones no paramétricas [4].

4. *Emparejamiento, correspondencia o matching*: El proceso de emparejamiento consiste en localizar los puntos (o características de la imagen) que se corresponden entre ambas imágenes del par estéreo. Su cálculo se considera la principal dificultad de la visión estéreo, y el resultado final dependerá de que se haya hecho de forma adecuada las correspondencias entre los píxeles.

5. *Cálculo del mapa de profundidades*: Este cálculo se realiza a partir de los resultados de disparidad obtenidos mediante el proceso de emparejamiento y los parámetros de orientación de la cámara.

6. *Reconstrucción 3D*: El proceso final es la reconstrucción de la escena en 3D utilizando los datos del mapa de profundidades calculado, los parámetros de la cámara, la información de color o texturas de la escena y otras características necesarias como la perspectiva.

En el siguiente apartado nos centraremos en el trabajo realizado sobre el preprocesado y emparejamiento basado en regiones de las imágenes estéreo para la obtención de un mapa de profundidades. En el capítulo siguiente se tratará en detalle la fase final del proceso, la reconstrucción 3D.

2.4 Estéreo basado en regiones

La correspondencia mediante regiones se enmarca dentro de la familia de los métodos basados en características. En general estas técnicas dividen una o, a veces, las dos imágenes estéreo en regiones no solapadas de color homogéneo. En lugar de computar la disparidad para cada píxel individual, asignan un único valor de disparidad a cada una de las regiones obtenidas. A priori este planteamiento presenta varias ventajas: cuanto mayor es el nivel semántico de la primitiva, las correspondencias que se obtienen son más robustas; por otro lado, la cantidad de emparejamientos a realizar entre las regiones es mucho menor que el que se tendría que realizar para el total de píxeles.

Sin embargo, la reducción del número de emparejamientos tiene una contrapartida: debemos realizar un proceso previo de segmentación, cuestión no trivial,

y debemos elegir un conjunto de propiedades que caractericen convenientemente a las regiones para poder realizar la correspondencia.

Hay una considerable cantidad de literatura sobre el problema de la correspondencia en estéreo. En [14] se puede encontrar una extensa revisión de los algoritmos actuales de estéreo que producen un mapa de disparidad denso (con información para todos los puntos de la imagen). Centrándonos en técnicas que emplean segmentación, en [15] los autores proponen un algoritmo estéreo basado en regiones que utiliza deformación de la imagen para medir la calidad del mapa de disparidad. Otras técnicas, como en [12], combinan el emparejamiento basado en regiones con la optimización basada en grafos. En [9] se puede encontrar una amplia documentación sobre todos estos algoritmos, a la vez que se proponen dos métodos con unos buenos resultados para el cálculo del mapa de disparidad utilizando segmentación. En [10] también se tienen en cuenta las ocultaciones que se producen al realizar el estéreo, para lo que se segmentan ambas imágenes con la intención de encontrar las partes de cada región que están parcialmente ocultas (para esto se ayuda de una función de coste). A parte de los métodos comentados, en [13] se hace un interesante estudio sobre técnicas que resuelven el problema de la correspondencia mediante estéreo.

En esta sección se presenta un sistema de visión estereoscópica basada en segmentación que aprovecha la información obtenida y las ventajas de este tipo de sistemas para el cálculo del mapa de profundidades de una escena y la detección de los objetos que aparecen en ella.

2.4.1 Modelo propuesto

Como se ha comentado, la finalidad del sistema propuesto es la detección de objetos y la estimación de su profundidad. El esquema de procesamiento puede dividirse en cinco pasos que se presentan resumidamente a continuación:

1. En primer lugar se segmentan las imágenes de entrada en regiones de color homogéneo.

2. A partir de estas regiones se obtiene un vector de características que nos permite realizar el emparejamiento.

3. A continuación se calcula la profundidad de cada una de las regiones utilizando la información de disparidad.

4. Se analizan las profundidades encontradas y se reajustan mediante la asignación a capas.

5. El último paso es la detección de objetos, para lo cual se utiliza la información obtenida en la segmentación.

Implícitamente este modelo aplica dos asunciones básicas: en primer lugar, los píxeles de un segmento con color homogéneo tendrán valores de disparidad que siguen un modelo de disparidad suavizado (restricción de continuidad). En segundo lugar, se asume que las discontinuidades en profundidad coinciden con las fronteras de dichas regiones. Todo esto se justifica en el hecho de que los píxeles pertenecientes a una región con un mismo color lo más probable es que provengan del mismo objeto de la escena, por lo que se espera que la disparidad varíe suavemente.

En las siguientes secciones se analiza en detalle cada uno de los pasos del algoritmo.

Segmentación

El proceso de segmentación puede definirse como la acción de dividir una imagen en un conjunto de regiones disjuntas cuyos píxeles comparten atributos similares, generalmente de intensidad, color o textura. En principio cualquier algoritmo que divida la imagen en regiones de un color homogéneo puede ser utilizado por el modelo propuesto. En la implementación actual se ha utilizado un algoritmo de segmentación que utiliza una nueva técnica denominada “umbralización adaptativa” y que gracias a su reducido tiempo de cómputo permite trabajar en tiempo real [7]. Lo más importante es que devuelva una serie de características descriptivas para cada región segmentada, que, como se verá en la siguiente sección, serán las que se utilizarán en el proceso de correspondencia.

Emparejamiento de regiones

Para realizar la correspondencia de regiones (establecer para cada región de la imagen izquierda su región correspondiente en la imagen derecha) se ha utilizado un método de comparación basado en la similitud de características. En este trabajo se propone la caracterización de las regiones en base a su posición, tamaño, color y forma. Para ello se utilizan las características que facilita el método de segmentación, concretamente se utilizan como descriptores de los objetos el

centroide (posición), el área (tamaño), el color y el histograma de distancias al centroide (forma). Existen otras muchas posibilidades para describir un objeto, pero éstas se proponen por sus ventajas (en cuanto a su facilidad de obtención) y por ser lo suficientemente significativas para caracterizar una región [8].

En el artículo (página 85) se puede consultar una descripción detallada del proceso de extracción de cada una de estas características, así como las ecuaciones definidas para el cálculo de las distancias que se utilizan en la comparación.

A partir del conjunto de características de una región se forma su *vector de características*, el cual permite realizar el proceso de emparejamiento y además, en una etapa posterior, ayudará al reconocimiento de los objetos de la escena.

Como medida de similitud entre dos regiones se ha tomado el inverso de la media ponderada de las distancias de los descriptores propuestos. En la ecuación 2.2 se puede ver esta medida, donde $w_0, \ldots, w_3$ son los pesos utilizados para ajustar el valor de cada componente en el resultado total de la ecuación. De esta forma se puede dar más peso a una determinada característica que dé mejores resultados en el proceso de emparejamiento (como el color y la posición) y menos a otras (como la forma y el tamaño).

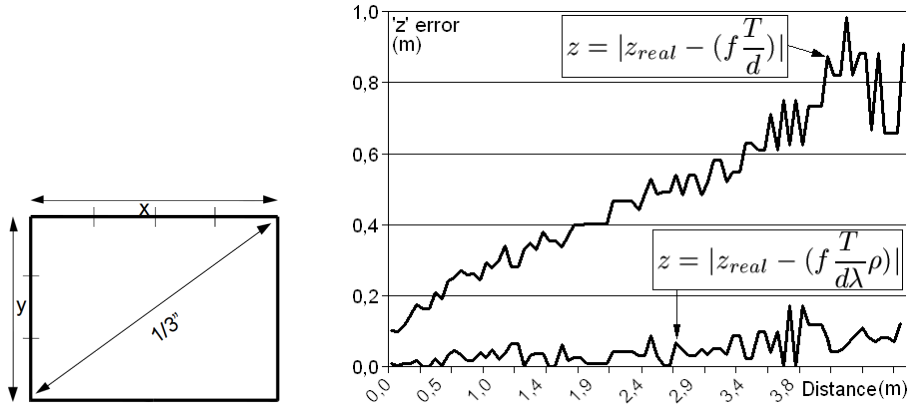
$$\delta = 1/(w_0 DistCentroide + w_1 DistArea + w_2 DistColor + w_3 DistForma) \quad (2.2)$$

En el proceso de correspondencia se van comparando una a una las regiones que estén todavía sin emparejar y que además cumplan las restricciones propuestas. Finalmente se emparejarán aquellas para las que se obtenga el menor valor de δ.

Cálculo de la disparidad

La disparidad se calcula a partir de la diferencia absoluta de la posición en cada imagen de cada una de las regiones emparejadas, esta posición vendrá dada por la componente *x* de su centroide.

Una vez realizado el cálculo de la disparidad, y teniendo en cuenta la relación inversamente proporcional existente entre disparidad y profundidad, estamos en disposición de obtener la profundidad a la que se encuentran los objetos de la escena mediante la ecuación 2.1 ($z = f\frac{T}{d}$), donde f indica la distancia focal, T la línea base y d el valor de disparidad. Para la cámara estéreo utilizada (*Bumblebee HICOL-60*) estos valores son de f = 0,006m y T = 0,12m.



(a) (b)

Figura 2.3: *(a) Dimensiones del CCD de la cámara estéreo. (b) Comparación del error producido en las ecuaciones del cálculo de la profundidad.*

El principal problema es que la disparidad *d* está expresada en píxeles, mientras que el resto de unidades lo están en metros. Para unificar las unidades se ha empleado el factor de conversión λ = *ancho del CCD en metros / ancho de la imagen en píxeles*. Para calcular el ancho del CCD de la cámara necesitamos conocer sus dimensiones y su proporción ancho/alto, que para la cámara utilizada son de $1/3$" y de $x/y = 4/3$ respectivamente (figura 2.3(a)). De forma que substituyendo en la igualdad $x^2 + y^2 = (1/3")^2$ podemos despejar el valor de *x* y obtener así el ancho total en metros.

Además existen un error que aparece cuando la distancia obtenida se compara con la distancia real (figura 2.3(b)). Este error es una pequeña desviación lineal debida a la concavidad de las lentes, el cual se puede corregir añadiendo un factor de corrección ρ a la ecuación (este valor se obtiene empíricamente, en la sección de experimentación de artículo se puede consultar el proceso seguido [página 91]).

En la ecuación 2.3 se muestra la ecuación final con las modificaciones propuestas que se ha utilizado para el cálculo de la profundidad.

$$z = f\frac{T}{d\lambda}\rho \tag{2.3}$$

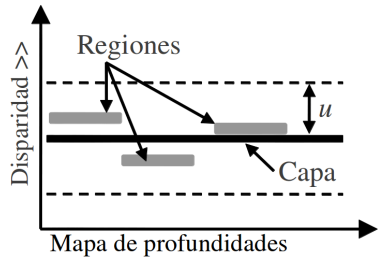


Figura 2.4: *Esquema de la extracción de capas.*

Extracción de capas

Lo que se pretende con este proceso es agrupar en una capa todas aquellas regiones que puedan ser aproximadas por una misma profundidad. De esta forma se eliminan los posibles *outliers* (regiones de un tamaño muy pequeño que además no se pueden asignar a ninguna capa) y se minimizan los posibles errores en el cálculo de la profundidad (debidos a pequeñas diferencias en los valores de los centroides). Suponiendo que las regiones están ordenadas según su profundidad, los pasos que sigue el algoritmo son los siguientes (ver también figura 2.4):

1. Se toma la primera región y se crea una capa a esa profundidad.
2. Se comprueba si la distancia desde la capa hasta la siguiente región es menor que un umbral u dado. En caso afirmativo se asigna la región a la capa y se recalcula la profundidad como la media entre la profundidad actual de la capa y la de la región. En caso negativo se coge la siguiente región, se crea una nueva capa y se vuelve a repetir este paso (hasta que no queden más regiones).

2.5 Detección de objetos

En el mismo artículo también se propone una primera aproximación para la detección y reconocimiento de objetos utilizando información de color y la forma de las regiones (histograma de distancias hasta el centroide). Debido a que estas características ya son extraídas previamente en el cálculo del mapa de disparidad, se consigue que el proceso de reconocimiento sea muy eficiente. En la página 90 se pueden consultar el algoritmo propuesto y los resultados obtenidos.

Por otro lado, y dentro de las técnicas de visión artificial y del reconocimiento de objetos, en el artículo "*Improving edge detection in highly noised sheet-metal images*" (página 97) se presenta un nuevo método para la detección robusta y precisa de la orientación y la localización de objetos en imágenes de bajo contraste. En este trabajo se proponen nuevas técnicas orientadas a aumentar la precisión de los resultados obtenidos.

En primer lugar, y para obtener un mayor precisión, se realiza un proceso de calibración de la cámara utilizando el algoritmo de Zhang [16] mediante patrones de calibración 2D. A continuación, y para cada imagen a procesar, se realizan los siguientes pasos:

1. Eliminar la distorsión de la imagen a partir de los datos de calibración obtenidos previamente.

2. Segmentar la imagen mediante un algoritmo iterativo con auto-umbral. Mediante este proceso se obtienen unos resultados más robustos a cambios en la iluminación o brillos de la imagen.

3. Aplicar la transformada de Hough para la detección de los bordes de los objetos pero aplicando un proceso de clasificación, filtrado y de unión ponderada de las líneas obtenidas.

4. Detectar los bordes con mayor ruido utilizando un nuevo tipo de histograma para la detección de la forma (llamado histograma de tamaño acumulado), que nos permite eliminar los posibles *outliers* en los bordes obtenidos.

5. Por último, mediante un método de regresión robusta, se calculan las líneas representativas de cada lateral del objeto, así como su posición y orientación.

Para el proceso de experimentación se utilizó una base de datos de 200 imágenes de planchas metálicas con diferentes posiciones, orientaciones, y variaciones en las condiciones de iluminación. Los resultados se compararon con medidas muy precisas tomadas sobre la imagen, además se compararon los resultados con los de otros algoritmos de detección. En la tabla 2.1 se puede ver un resumen de estos resultados, donde para el método propuesto se obtuvo un error medio en la orientación de 0.144° y de 1.42 mm en la posición. En la página 107 se puede ver una explicación más completa del proceso de experimentación.

Método	Error medio angular	Error medio posición	Lateral correcto%
Método propuesto	0.144°	1.42 m	95.7%
RANSAC	0.162°	1.46 mm	94.2%
Iterative re-weighting	0.171°	1.49 mm	93.9%
Regresión lineal	0.181°	1.53 mm	93.6%
RR without sampling	0.986°	8.7 mm	89.1%
Model-based technique	2.43°	19.4 mm	-
RR without filtering	3.67°	25.6 mm	78.6%

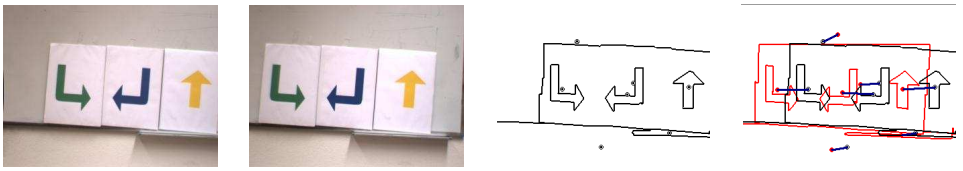
Tabla 2.1: *Resultados de la experimentación realizada*

Como resultado se obtuvo un algoritmo con un error mínimo en la localización y un coste temporal muy bajo, lo que posibilita su uso en entornos muy exigentes, como el propuesto en la reconstrucción del espacio, o en entornos industriales.

2.6 Resultados y conclusiones

Para el proceso de experimentación del cálculo del mapa de disparidad se realizaron pruebas con varias secuencias de imágenes a diferentes distancias, comparando las distancias estimadas con las distancias reales esperadas. Para el proceso de detección se experimentó con el reconocimiento de un conjunto de señales de tráfico, y con una base de datos de planchas metálicas industriales con diferentes orientaciones y condiciones de iluminación para validar la precisión y robustez de los métodos propuestos.

En la sección de experimentación de los artículos resumidos (páginas 91 y 107) se pueden consultar todos los resultados obtenidos sobre el algoritmo de segmentación, emparejamiento, cálculo del mapa de disparidad y detección de objetos. En las figuras 2.5 y 2.6 se muestran algunos de estos resultados. En la primera (2.5) aparece el proceso de segmentación del par estéreo y el emparejamiento en base a sus características. En la segunda figura (2.6) se ha incluido otro ejemplo de la segmentación realizada, del proceso de detección de objetos y del mapa de disparidad obtenido.



(a) (b) (c) (d)

Figura 2.5: *(a,b) Par estéreo a 1.0m. (c) Segmentación. (d) Resultado del emparejamiento.*

(a) (b) (c) (d)

Figura 2.6: *(a) Imagen de referencia. (b) Segmentación. (c) Resultado de la detección. (d) Mapa de disparidad.*

El cálculo de mapas de disparidad mediante la comparación basada en características es una solución más eficiente que otros métodos (como los de correlación por ventana o minimización de energía) ya que manipulan muchos menos datos (sólo se maneja el conjunto de las regiones segmentadas) y facilitan el proceso de detección de objetos en la escena. En contrapartida proporcionan un mapa de disparidad poco denso, pero para algunas aplicaciones pueden ser más adecuados, como la navegación guiada, la detección de marcadores y profundidades de los objetos, la ayuda a personas con alguna discapacidad o la conducción automática, por enumerar sólo unos pocos.

Algunas líneas de investigación abiertas son: mejorar del algoritmo de segmentación aplicando técnicas de sobresegmentación para reducir el número de errores, tratar el problema de las ocultaciones, probar otro tipo de características para mejorar el emparejamiento, e incorporar el algoritmo a un sistema autónomo de navegación guiada que aproveche la información de profundidades obtenida y la detección de los objetos de la escena.

Capítulo 3

Reconstrucción y mapeado

En este capítulo se resumen las aportaciones sobre la definición de un modelo computacional para la representación del entorno mediante enumeración espacial. Concretamente nos centraremos en los procesos de reconstrucción de una escena 3D a partir de imágenes estéreo y de su aplicación a un algoritmo de mapeado. Este trabajo se puede encontrar en las siguientes publicaciones:

1. "*Scene reconstruction and geometrical rectification from stereo images*" (página 111).
2. "*Discrete and continuous reconstruction of 3D scenes from disparity maps*" (página 123).
3. "*Three-Dimensional Mapping from Stereo Images with Geometrical Rectification*" (página 139).
4. "*3D Reconstruction and Mapping from Stereo Pairs with Geometrical Rectification*" (página 165).
5. "*Rectified Reconstruction from Stereo Pairs and Robot Mapping*" (página 153).

3.1 Introducción

La reconstrucción de entornos desconocidos es un requisito indispensable en muchos campos de la investigación. Muchos autores utilizan la visión estéreo para

resolver el problema de la reconstrucción 3D o el mapeado. Por ejemplo, una primera solución a la reconstrucción tridimensional mediante tecnología estéreo fue propuesta por [17]. Esta solución explora la posibilidad de componer varias vistas 3D a partir de las transformaciones de la cámara. Hay otras aproximaciones que infieren rejillas 3D a partir de visión estéreo, debido al hecho de que la información de apariencia no se puede obtener mediante sensores de rango. Para solucionar esto añaden una cámara adicional a los robots móviles [18, 19]. Además, se puede añadir un módulo de reconocimiento 3D para la identificación de objetos. Esta técnica no es exclusiva de la robótica, pues puede ser utilizada en otro tipo de aplicaciones como el guiado automático de máquinas o la detección y estimación del movimiento de un vehículo [20].

La visión estéreo puede mejorar la percepción de las escenas y el modelado del mundo, prueba de ello son los muchos algoritmos existentes que trabajan con imágenes de disparidad. El problema es que estos algoritmos no se pueden aplicar de una forma genérica a todo tipo de estructuras, dado que las imágenes o los objetos capturados mediante una cámara no conservan su tamaño ni perspectiva real. Esto es debido a que se ven deformados por el efecto de la perspectiva cónica. En general, cualquier imagen tomada por una cámara se ve deformada por este efecto, por lo que una reconstrucción directa generaría escenas con un aspecto irreal. Hay muy pocos trabajos que se centren en crear buenos resultados en esta reconstrucción y en obtener una apariencia real. Sin embargo, podemos encontrar algunas publicaciones interesantes como [21, 23, 22], pero en ninguno de ellos realiza ningún tipo de rectificado geométrico. En estos trabajos se reconstruye objectos específicos en lugar de escenas completas, por lo que la estructura real del entorno no es recuperada. En otros campos sí que podemos encontrar la aplicación de soluciones al problema del rectificado, como en [24] para rectificar carreteras y obtener así su apariencia real.

Este trabajo se centra en la reconstrucción de la estructura de una escena que muestre su aspecto real, utilizando solamente la información proporcionada por las imágenes estéreo y los mapas de disparidad (en realidad utilizamos mapas de profundidad). Nuestra propuesta no realiza suposiciones sobre la escena o la estructura de los objetos contenidos, ni realiza ningún proceso de segmentación de objetos intentando identificar formas conocidas. Solo se emplea un par de imágenes estéreo. El rectificado de perspectiva permite eliminar el efecto de perspectiva cónica de la escena además de permitir modificar la orientación de la cámara. De esta forma el algoritmo recupera la estructura de la escena y

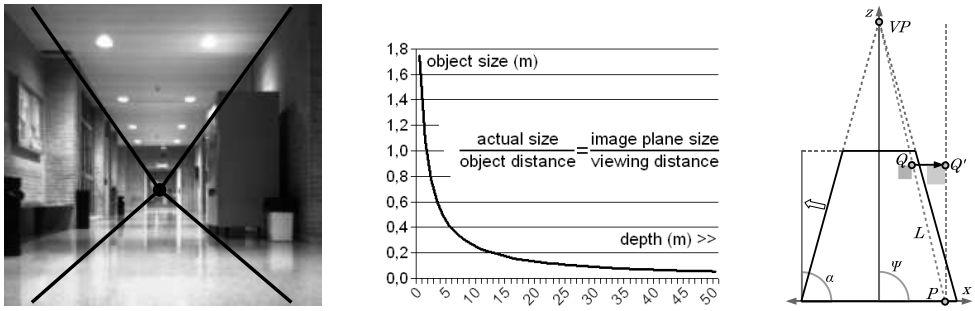
otra información como la geometría de los objetos, su volumen o profundidad. Además, el método de reconstrucción se ha extendido para realizar el mapeado de una secuencia de imágenes mediante el mismo procedimiento.

3.2 Rectificación geométrica

El efecto de perspectiva está presente en la forma en que percibimos el entorno que nos rodea. Este efecto deforma el tamaño y la geometría del espacio y de los objetos que contiene. Debido a esta deformación se crea el efecto de profundidad, por el que vemos más pequeños los objetos según aumente la distancia. En la figura 3.1(b) se muestra como el efecto de perspectiva cónica cambia el tamaño de los objetos según aumente la distancia al punto de visión. La rectificación se utiliza para corregir este efecto y recuperar la geometría real de la escena. Como ejemplo, la figura 3.1(a) muestra una imagen de un pasillo, en la cual, un píxel que esté situado en la parte inferior de la imagen representa un volumen muy pequeño de la escena (debido a que está en primer plano). Sin embargo, un píxel que esté en el centro de la misma imagen representa un volumen mucho mayor (debido a que la parte de la escena representada está en el fondo). Si se reconstruyese la escena directamente, se conservaría ese efecto de perspectiva. Por lo que para realizar una reconstrucción correcta se debería de eliminar este efecto, realizando una corrección sobre la posición de cada uno de los píxeles. De esta forma el resultado mostrará el mismo aspecto que la escena real.

El rectificado se realiza sobre los valores del mapa de profundidad D, el cual se calcula a partir de un par de imágenes estéreo. En principio el mapa de profundidad puede ser calculado por cualquier método, pero para este trabajo se ha utilizado un algoritmo que combina multi-resolución y minimización de energía (EM) [25]. Cada valor $D(x,y)$ del mapa de profundidad contiene la profundidad asociada con el píxel (x,y) de la imagen de referencia (imagen izquierda) [26, 27].

En la figura 3.1(c) se muestra el esquema de este proceso. En la parte izquierda aparece el resultado deseado tras el rectificado de una de las paredes, la cual es rotada α° para recuperar su inclinación real. En el lado derecho de este esquema se muestra el píxel actual Q (obtenido de la imagen de profundidad) con las coordenadas $(x,y,D(x,y))$, el cual se rectifica para obtener Q'. Este píxel forma parte de un supuesto objeto de la imagen, que tras rectificar el conjunto de los píxeles que lo componen recuperará su tamaño real.



(a) (b) (c)

Figura 3.1: *(a) Muestra el efecto de la perspectiva cónica. (b) Muestra como disminuye el tamaño de los objetos según aumenta la profundidad, debido al efecto de la perspectiva. (c) Esquema del rectificado geométrico sobre una escena no rectificada visto desde arriba (plano x-z).*

El primer paso para rectificar Q es obtener la línea L, que conecta los puntos VP (punto de fuga) y Q. A continuación se calcula la intersección de esta línea con el plano *x-y*, obteniendo de esta forma el punto P. A partir de este punto P, la línea L es rotada para que sea perpendicular al plano *x-y*. De esta forma se obtienen las coordenadas (x,y) de Q'.

El calculo de la coordenada z de Q' se obtiene utilizando la versión modificada de la ecuación $z = f\frac{T}{d}$ [11] que se propuso en el capítulo anterior (página 16, ecuación 2.3).

En resumen, la función referida como π en 3.1 muestra como calcular el nuevo punto rectificado $Q' = (x',y',z')$ de la forma $Q' = \pi(Q)$.

$$\pi(Q) = \begin{cases} x_{Q'} &= x_{VP} + z_{VP}\frac{x_{VP}-x_Q}{z_Q-z_{VP}} \\ y_{Q'} &= y_{VP} + z_{VP}\frac{y_{VP}-y_Q}{z_Q-z_{VP}} \\ z_{Q'} &= f\frac{T}{d\lambda}\rho \end{cases} \tag{3.1}$$

Como se puede ver en la ecuación 3.1, es necesario calcular el punto de fuga VP. En general se puede utilizar el punto central de la imagen, con lo que se obtendría una reconstrucción que mantendría el ángulo original de la cámara. En el caso de que el punto de vista sea oblicuo, se puede calcular su posición y corregir en la reconstrucción final obteniendo una perspectiva frontal de la escena final. Cuando no se puede calcular el punto de fuga (imágenes en las que

no aparezcan rectas, conocidas como "non Manhattan Worlds"), se usará por defecto el punto central de la imagen.

Para la profundidad del punto de fuga VP se utiliza el máximo valor de profundidad de todo el mapa de disparidad D. Para calcular la posición (x,y) de VP en la imagen se ha utilizado el método propuesto en [28]. Este método utiliza un modelo Bayesiano que combina la geometría 3D de la escena con información estadística de los bordes encontrados en la imagen. Como solución devuelve un ángulo (llamado Ψ) que define la orientación de la cámara en la dirección $\cos\Psi - \sin\Psi$. Finalmente, las coordenadas cartesianas (x,y,z) de VP se pueden calcular a partir de las componentes del vector $\boldsymbol{u} = (u,v)$ obtenidas de la forma:

$$u = \frac{f \cdot (-x_{VP}\sin\Psi - y_{VP}\cos\Psi)}{x_{VP}\cos\Psi - y_{VP}\sin\Psi}, \quad v = \frac{f \cdot z_{VP}}{x_{VP}\cos\Psi - y_{VP}\sin\Psi} \tag{3.2}$$

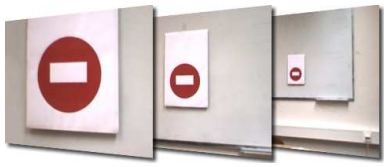
3.2.1 Calculo del error en el rectificado

La imagen 3.2(b) muestra el error en el tamaño de representación de los objetos después del proceso de rectificado. Para obtener este error se ha utilizado una secuencia de imágenes en las que aparece el mismo objeto a diferentes distancias (la señal de la imagen 3.2(a)). En esta gráfica se compara el tamaño de obtenido después del rectificado con el tamaño real esperado del objeto. Este error se puede calcular fácilmente debido a que se conoce el tamaño real del objeto y las coordenadas esperadas de cada punto.

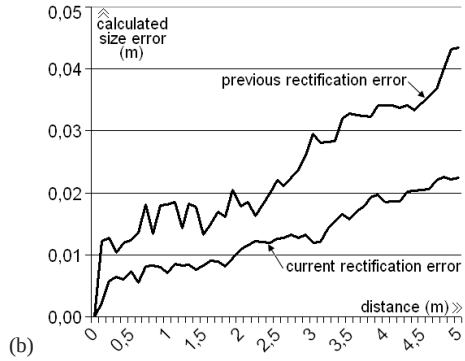
3.3 Aplicaciones del rectificado geométrico

El método propuesto se puede utilizar en un amplio rango de aplicaciones, dado que se recupera información muy valiosa de la escena, como es la geometría, el volumen y la profundidad. Por ejemplo, se podría aplicar en sistemas de Realidad Aumentada (RA) para desarrollar nuevas aplicaciones, mejoras o solucionar problemas relacionados con este campo.

A continuación se presentan dos posibles aplicaciones. La primera es la reconstrucción tridimensional de escenas utilizando la precisión sub-píxel e imágenes estéreo. La segunda es la construcción de un mapa a partir de una secuencia de imágenes estéreo. Estas dos aplicaciones demuestran la utilidad del rectificado geométrico y las ventajas de su aplicación a esta clase de problemas. En la



(a)



(b)

Figura 3.2: *(a) Secuencia de imágenes usadas para calcular el error en el tamaño de representación. (b) Comparación del error cometido en el tamaño según aumenta la profundidad.*

sección de conclusiones se proponen algunas posibles aplicaciones adicionales.

3.4 Reconstrucción utilizando precisión sub-píxel

Para realizar la reconstrucción 3D de una escena en primer lugar se calcula el mapa de profundidad D a partir de un par de imágenes estéreo (imagen izquierda o de referencia LI e imagen derecha RI). Este mapa de profundidad se obtiene utilizando los métodos propuestos en [25].

A partir de D se realiza el proceso de rectificado geométrico. El resultado de esta etapa es una matriz rectificada de vóxeles R que representa la ocupación espacial. Esta matriz se inicializa a cero y a continuación se rellena de la forma $R(x',y',D(x,y)) = 1$ donde (x',y') son las coordenadas rectificadas de (x,y), $\forall x,y \in \mathbb{R}/\{0 \leq x \leq m-1, 0 \leq y \leq n-1\}$ y (m,n) son las dimensiones de la imagen.

El último paso es convertir las coordenadas de los píxeles obtenidos en unidades reales. La ecuación 3.1 muestra como calcular la equivalencia en metros a partir de un valor de disparidad y de las coordenadas rectificadas. En la figura 3.3 se puede ver un esquema de este proceso, y como se utiliza el resultado obtenido en el algoritmo de mapeado (explicado en la siguiente sección).

En la reconstrucción obtenida tras realizar el rectificado aparecen agujeros en la representación. Esto es debido a que los mapas de profundidad son discretos,

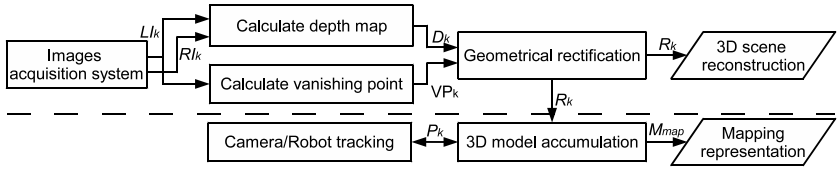


Figura 3.3: *Esquema de los algoritmos de reconstrucción y mapeado.*

y por lo tanto al rectificar (o modificar) la posición de los píxeles, los vóxeles resultantes se separan en la reconstrucción 3D (en la figura 3.4 se puede ver el espacio creado tras la rectificación entre los píxeles representados). Este efecto aumenta de forma proporcional según la profundidad del objecto, puesto que sufrirá una rectificación mayor.

Para minimizar este problema se ha utilizado la *precisión sub-píxel* para introducir píxeles ficticios y así rellenar estos agujeros. El número de píxels ficticios creados también aumenta según la distancia, introduciendo $n = 1-(z/D_{max})$ píxeles entre dos píxeles consecutivos. Esta ecuación devuelve un valor mínimo cuando los píxeles se encuentran en primer plano y máximo cuando pertenecen al fondo.

A continuación se resumen todos los pasos de este algoritmo:

1. $D := CalculateDepthMap(LI, RI)$
2. **while (** $x \leq m-1$ **)**
   (a) **while (** $y \leq n-1$ **)**
      i. $(x', y') := \pi(x, y)$
      ii. $R(x', y', D(\lfloor x \rfloor, \lfloor y \rfloor)) = 1$
      iii. $y = y + 1 - (D(\lfloor x \rfloor, \lfloor y \rfloor)/D_{max})$
   (b) $x = x + 1 - (D(\lfloor x \rfloor, \lfloor y \rfloor)/D_{max})$
3. $Display(R)$

3.4.1 Experimentación sobre la reconstrucción

En la figura 3.4 se pueden ver dos ejemplos de reconstrucción utilizando la precisión sub-píxel. En las imágenes de la izquierda se muestra la malla obtenida durante la reconstrucción del mismo objecto pero en el primero no se ha utilizado la precisión sub-píxel y en el segundo sí. En la parte de la derecha aparece

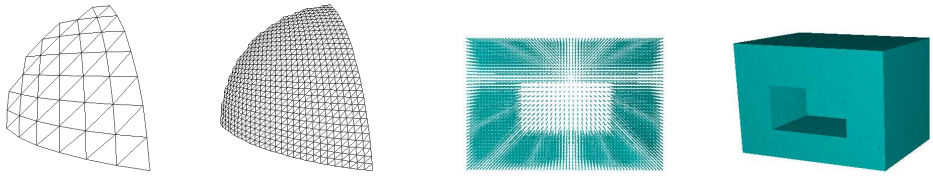


Figura 3.4: *Ejemplos de reconstrucción utilizando la precisión sub-píxel.*

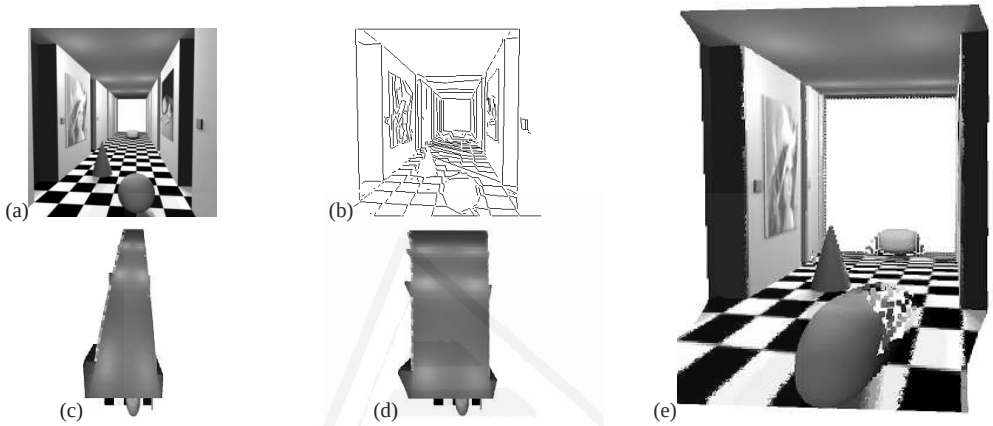


Figura 3.5: *Ejemplos de rectificado geométrico utilizando el mapa de disparidad sintético de un pasillo.*

otro ejemplo de reconstrucción utilizando vóxeles. La primera imagen muestra la reconstrucción sin la precisión sub-píxel. Como se puede ver los vóxeles están separados debido al rectificado geométrico. En la segunda imagen sí que se ha aplicado, obteniendo un resultado mucho más realista al eliminar los agujeros.

La figura 3.5 muestra ejemplos de una reconstrucción 3D utilizando un mapa de profundidad sintético que simula un pasillo (figura (a)). Este ejemplo clarifica el efecto conseguido con el rectificado geométrico. En la figura (b) se puede ver la segmentación realizada para calcular el punto de fuga, el cual se ha estimado que es de -4° mediante el método descrito en la sección 3.2. En las figuras (c), (d) y (e) se compara el efecto del rectificado: (c) muestra una reconstrucción sin aplicar el rectificado (visto desde arriba), y (d) y (e) muestran vistas superiores y laterales del resultado de la reconstrucción después de aplicar el rectificado. Como se puede ver, los muros aparecen rectos y paralelos, como era de esperar.

3.5 Algoritmo de mapeado

Para realizar el mapeado de la escena se ha utilizado una secuencia de N pares de imágenes estéreo de la escena a reconstruir, numeradas de la forma (LI_0, RI_0), (LI_1, RI_1), ..., (LI_{N-1}, RI_{N-1}). Esta secuencia de imágenes se ha tomado a distancias equidistantes entre si. Una vez capturado un par de imágenes estéreo (LI_k, RI_k), se calcula el mapa de disparidad correspondiente D_k y se añade a la lista Σ que contiene todos los mapas de disparidad. A continuación se utiliza el algoritmo de rectificado geométrico para calcular las matrices rectificadas R_k de cada mapa de disparidad. Para cada matriz R_k se calcula su intersección con la matriz anterior $(R_{k-1} \cap R_k)$, y se añade el resultado a la matriz global M_{map}, la cual representa el mapeado final de la escena. En la figura 3.3 se puede ver un resumen de este proceso.

En esta aproximación la posición de los *frames* se obtiene directamente de la odometría del robot. Para reducir el efecto de los errores en la odometría se aplica un *filtro cúbico* en cada una de las intersecciones realizadas entre mapas. Este filtro F (explicado a continuación) discretiza el resultado obtenido, transformándolo en una rejilla de cubos rectangulares. Por último, el resultado obtenido (M_{map}) es representado calculando la equivalencia en unidades reales (metros).

A continuación se resumen los pasos del algoritmo de mapeado:

1. **for each** $D_k \in \Sigma$ **do**
   (a) $D_k := CalculateDepthMap(LI_k, RI_k)$
   (b) $R_k := ApplyRectification(D_k)$
   (c) $M_{map} := M_{map} \cap R_k$
2. $M_{map} := ApplyCubicFilter(M_{map})$
3. $Display(M_{map})$

El filtro cúbico F aplica la ecuación $g(x,y,z) := \Sigma_{(i,j,k)\in S} f(i,j,k)$ a cada cubo de la matriz, donde S representa el conjunto de coordenadas de los puntos que están en la vecindad de $g(x,y,z)$, incluyendo el propio punto. De esta forma, se suma la ocupación espacial de cada cubo en su punto medio, y cada celda contiene el conjunto de lecturas de esa porción del espacio. Denominamos "votos" al número de lecturas que se suman en cada cubo, estos votos representan

la probabilidad de ocupación de esa porción del espacio. Gracias a este procedimiento conseguimos reducir los posibles errores de odometría, pues los valores obtenidos no dependerán únicamente de la lectura de una posición del espacio.

3.5.1 Experimentación sobre el mapeado

Para realizar la experimentación sobre el algoritmo de mapeado se han tomado dos secuencias de 30 imágenes de dos pasillos diferentes (imágenes 3.7 y 3.8). Las figuras 3.7(a) y 3.8(a) muestran las tres primeras capturas de cada secuencia y los mapas de profundidad correspondientes.

El objetivo principal es que los muros, suelo y techo aparezcan sin inclinación en la reconstrucción, y que no aparezca ruido de por medio en el espacio del pasillo. También es importante que las columnas (representadas por círculos en el esquema del pasillo de la figura 3.7(b)) y que la máquina de café (representada por un rectángulo en el esquema de la figura 3.8(b)) sean detectados correctamente.

En las figuras 3.7(c, d) y 3.8(c, d) se muestran los resultados obtenidos del mapeado. En 3.7(c) y 3.8(c) no se ha aplicado el rectificado geométrico, como se puede ver se obtiene una reconstrucción incorrecta: el espacio intermedio del pasillo está completamente lleno de ruido. Esto es debido a que al acumularse las matrices sin rectificar en el mapeado se solapan unas con otras, como se puede ver en el esquema de la figura 3.6.

Figura 3.6: *(a) Ejemplo del error cometido en el mapeado al no aplicar el rectificado geométrico. (b) Resultado correcto esperado tras aplicar el rectificado.*

En 3.7(d) y 3.8(d) sí que se ha aplicado el rectificado, obteniendo unos resultados mucho mejores. El pasillo está correctamente definido, sin ruido, y se puede identificar claramente todos los elementos: las paredes, las columnas y la máquina de café en la parte de la derecha de cada uno de los resultados.

Para estos ejemplos se ha utilizado un filtro cúbico con un tamaño de $3 \times 3 \times 3$ y un número de votos mínimo de 5.

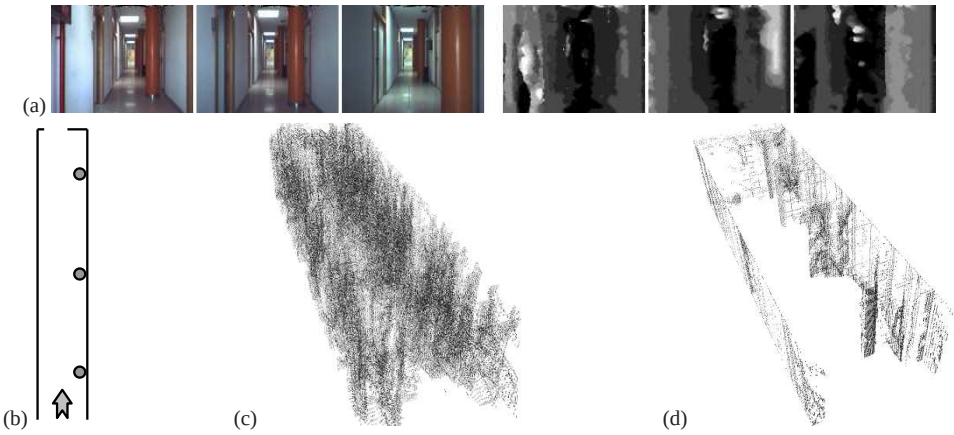


Figura 3.7: *(a) Secuencia de imágenes para el mapeado. (b) Esquema del pasillo. (c, d) Resultados del mapeado del pasillo sin aplicar rectificado y aplicándolo, respectivamente.*

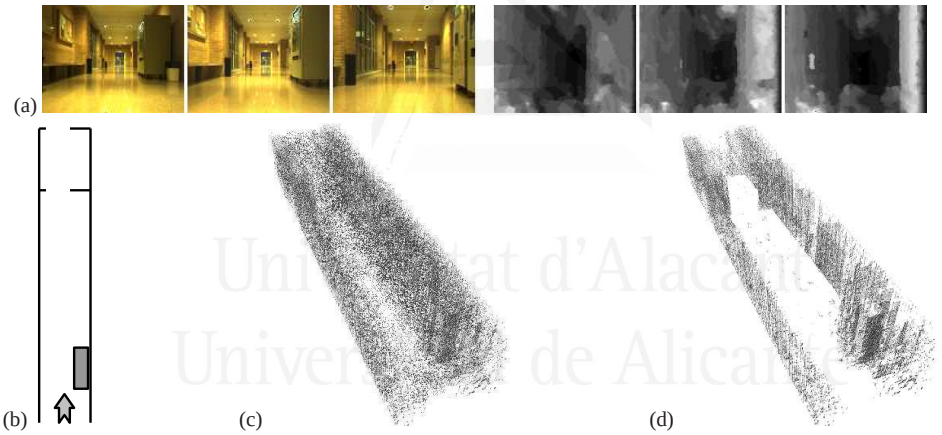


Figura 3.8: *(a) Secuencia de imágenes para el mapeado. (b) Esquema del pasillo. (c, d) Resultados del mapeado del pasillo sin aplicar el rectificado y aplicándolo, respectivamente.*

3.6 Resultados de rendimiento

Para realizar los experimentos se ha utilizado un Pentium IV a 3,20GHz con 2GB de RAM y una tarjeta gráfica de 512MB. La reconstrucción de los mapas se ha

realizado utilizando una matriz de vóxeles con un tamaño de $320 \times 240 \times 256$ y mapas de profundidad con resoluciones de 320×240 píxeles.

Hay que tener en cuenta que únicamente se procesan los píxeles no nulos (con profundidad finita) de los mapas de profundidad. El coste computacional depende linealmente del tamaño de las imágenes de entrada y de la precisión requerida para la reconstrucción.

En los casos propuestos el algoritmo obtiene un buen rendimiento: tarda una media de 9 segundos para procesar toda una secuencia de 30 imágenes (considerando que cada imagen tiene aproximadamente un 70% de píxeles no nulos), y algo menos de 0.3 segundos para realizar la reconstrucción de una imagen de forma individual.

3.7 Trabajo previo

En este capítulo se han resumido principalmente los últimos resultados de los trabajos publicados sobre reconstrucción y mapeado. Pero previamente se publicaron otros métodos o aproximaciones a estos resultados:

- En el artículo "*Scene reconstruction and geometrical rectification from stereo images*" (página 111) se presentó una primera solución a la reconstrucción del entorno a partir de imágenes de disparidad. Para realizar este proceso se proponían y comparaban tres posibles algoritmos de reconstrucción: un método lineal y otros dos que hacían uso de la función logaritmo para aumentar el rectificado según la distancia.

- En "*Discrete and continuous reconstruction of 3D scenes from disparity maps*" (página 123) se amplía el trabajo anterior con un nuevo método de reconstrucción continua para intentar solventar el problema de los huecos producidos tras el rectificado.

- Posteriormente, en "*Three-Dimensional Mapping from Stereo Images with Geometrical Rectification*" (página 139) se propone una nueva ecuación de reconstrucción, que ya utiliza el concepto de punto de fuga pero que únicamente funciona para perspectivas frontales. Además se propone una primera aproximación al algoritmo de mapeado.

Los resultados presentados en este capítulo amplían estos trabajos previos con: una nueva formulación para el proceso de rectificado geométrico (que permite cualquier tipo de perspectiva), un nuevo algoritmo de mapeado, el uso de un filtro cúbico y de la precisión sub-píxel, y una nueva sección de experimentación.

3.8 Conclusiones

Se ha presentado un nuevo método para realizar la reconstrucción de escenas a partir de imágenes estéreo, además de un algoritmo para realizar el mapeado de un entorno a partir de una secuencia de imágenes estéreo. Estos métodos son una mejora sobre los métodos previos debido a que se ha utilizado un nuevo filtro de rectificado, la precisión sub-píxel, y nuevos algoritmos para la reconstrucción y el mapeado. Estos métodos usan el rectificado geométrico para eliminar el efecto de la perspectiva cónica, con la intención de recuperar el aspecto real de la escena en el resultado final. Además el rectificado también permite recuperar información muy importante de la escena, como es la geometría de los objetos, su volumen y profundidad. Sin embargo, la calidad final obtenida en la reconstrucción dependerá de la calidad de los mapas de profundidad con los que se trabaje.

Los resultados muestran como este proceso corrige el efecto de la perspectiva y como ayuda a mejorar el proceso de emparejamiento en el algoritmo de mapeado. Estos algoritmos se podrían utilizar en una amplia variedad de aplicaciones debido al buen rendimiento que obtienen y su baja carga computacional adicional.

Los trabajos actuales están orientados a aplicar los resultados obtenidos en un sistema de Realidad Aumentada. Estos sistemas podrían aprovechar la información de geometría obtenida para desarrollar nuevas aplicaciones (como nuevos tipos de interfaces o juegos) y para solucionar problemas relacionados con esta disciplina (como el *visual tracking*, el alineamiento o el ocultamiento). También se pretende utilizar este trabajo en un sistema robótico autónomo que utilice la información del entorno para identificar ciertas áreas específicas.

Capítulo 4

Lenguaje de modelado

En este capítulo se presenta un resumen del trabajo realizado sobre modelos computacionales para la definición formal de sistemas gráficos. Este trabajo se recoge en las siguientes publicaciones:

- "*Hacia un modelo integral para la generación de Mundos Virtuales*" (página 179)
- "*Formal model to integrate Multi-Agent Systems and Interactive Graphic Systems*" (página 203)
- "*Modeling a Mobile Robot using a Grammatical Model*" (página 215)
- "*A Grammatical Approach to the Modeling of an Autonomous Robot*" (página 227)

4.1 Introducción

Una vez obtenida la estructura del espacio, se hace necesario dotar al modelo de una representación de mayor potencia, estableciendo un método que recoja las relaciones entre los objetos del espacio y que permita, además, una representación más compacta. En este capítulo se propone un lenguaje para el modelado de mundos virtuales basado en gramáticas, mediante el cual se obtienen diversas ventajas: permite la representación del espacio en forma de cadenas pertenecientes a un lenguaje y facilita la expresión de relaciones entre los componentes de la cadena.

Este modelo además afronta uno de los problemas más importantes en los sistemas gráficos: la diversidad de los dispositivos visuales y de interacción que existen en la actualidad. Junto a esto, la heterogeneidad de los motores gráficos, los motores físicos y los módulos de Inteligencia Artificial, propicia que no exista un modelo que aúne todos estos aspectos de una forma integral y coherente. En la página 182 se realiza un estudio del arte sobre la diversidad existente en este tipo de sistemas.

El modelo propuesto se basa en la definición de una gramática que integra toda la actividad necesaria de un sistema gráfico, su visualización y su interacción con el usuario. La descripción de un mundo (ya sea un entorno real o una escena sintética) se presenta como una secuencia ordenada de primitivas, transformaciones y actores. Estos conceptos se definen en un sentido mucho más amplio de lo que es habitual en un lenguaje de modelado clásico: Las primitivas no son simples primitivas de dibujo sino acciones que se deben ejecutar en un determinado sistema de representación, gráfico o no; las transformaciones modifican el comportamiento de las primitivas dependiendo de su naturaleza; y los actores definen la actividad del sistema, los cuales se visualizan mediante primitivas y transformaciones, y usan el concepto de eventos para controlar la activación de su comportamiento.

Este modelo tiene como virtud la separación de la actividad del sistema de los dispositivos visuales y de interacción concretos que lo componen. Gracias a esto los dispositivos pueden ser sustituidos por otros dispositivos o por simulaciones, los elementos del sistema pueden ser reutilizados, y la representación gráfica puede ser diferente dependiendo del dispositivo visual.

En definitiva, se ha pretendido diseñar un sistema integral, adaptativo, reutilizable y genérico. En las próximas secciones se presenta la gramática que define este lenguaje, al cual hemos denominado como “lenguaje Generador de Mundos Virtuales” o GMV. Por último se incluye un caso de estudio para explicar el uso del modelo propuesto. En este ejemplo se formaliza un sistema de navegación autónoma para un robot que utiliza información multimodal y un sistema híbrido de control [30].

4.2 Modelo para la Generación de Mundos Virtuales

En el modelo GMV propuesto, una escena gráfica (o de forma genérica, un mundo virtual) se describe como una secuencia ordenada de primitivas, transforma-

ciones y actores, cuya función se define como:

- Una *primitiva* es la descripción o representación de un objeto en un sistema de representación dado. Normalmente consistirán en primitivas gráficas, pero también podrían ser sonidos u otro tipo de primitiva en un espacio de representación.
- Las *transformaciones* modifican el comportamiento de las primitivas.
- Los *actores* son los componentes que realizarán la actividad del sistema, cuya representación será definida mediante primitivas y transformaciones.

Para modelar las acciones de los actores se introduce además el concepto de *evento*. Estos eventos provocarán la activación de un determinado comportamiento en uno o varios actores.

Cada elemento de la escena se representa mediante un símbolo del *conjunto de símbolos de la escena*. A partir de estos símbolos, y siguiendo unas reglas sintácticas, se podrán construir cadenas para describir mundos virtuales. Estas reglas definen la gramática M del lenguaje [29] que veremos a continuación.

4.2.1 Sintaxis

La gramática M queda definida por una tupla $M = < \Sigma, N, R, s >$, donde Σ es el conjunto de símbolos terminales, N es el conjunto de símbolos no terminales, R es el conjunto de reglas sintácticas (una regla sintáctica es una aplicación de la forma r: $N \rightarrow W^*$, donde $W = \Sigma \cup N$) y $s \in N$ es el símbolo inicial de la gramática. En nuestro caso, M queda definida como:

1. $\Sigma = P \cup T \cup O \cup A^D_{ATTR}$, donde:
   - P: conjunto de símbolos para las primitivas.
   - T: conjunto de símbolos para las transformaciones.
   - $O = \{\cdot\,()\}$: símbolos para la indicación del ámbito "()" y la concatenación de símbolos "·".
   - A^D_{ATTR}: conjunto de símbolos para los actores, donde D es el conjunto de todos los tipos de eventos generados por el sistema y $ATTR$ es el conjunto de todos los atributos de los actores, los cuales definen

todos sus posibles estados. Por ejemplo, el actor a^H_{attr} realizará su actividad cuando reciba el evento e^h, donde $h \in H$, $H \subseteq D$ y $attr \in ATTR$ es su estado actual.

2. N = {MUNDO, OBJETOS, OBJECTO, ACTOR, TRANSFORMACIÓN, FIGURA}.

3. Las reglas gramaticales R se definen como:

   - Regla 1. **MUNDO** $\rightarrow$ OBJETOS
   - Regla 2. **OBJETOS** $\rightarrow$ OBJECTO | OBJECTO · OBJETOS
   - Regla 3. **OBJECTO** $\rightarrow$ FIGURA | TRANSFORM. | ACTOR
   - Regla 4. **ACTOR** $\rightarrow a^H_{attr}$, $a^H_{attr} \in \mathbf{A}^D_{ATTR}, H \subseteq D$
   - Regla 5. **TRANSFORMACIÓN** $\rightarrow t$(OBJETOS), $t \in T$
   - Regla 6. **FIGURA** $\rightarrow p^+$, $p \in P$

4. $s =$ MUNDO, es el símbolo inicial de la gramática.

La gramática M es una gramática independiente del contexto, o de tipo 2, y por lo tanto se asegura que existe un procedimiento que verifica si una escena está bien construida o no, o dicho de otro modo, que la cadena pertenece o no al lenguaje $L(M)$.

4.2.2 Semántica

Además de la sintaxis del lenguaje es necesario definir su semántica, la cual determina la funcionalidad de las cadenas generadas. Existen varias formas de definición: operacional, denotacional y axiomática. En este caso vamos a utilizar el método denotacional para su descripción mediante funciones matemáticas.

Función semántica de las primitivas (Regla 6)

La regla 6 describe una figura como una secuencia de primitivas, cuya semántica queda definida a través de la función α, de la forma:

$$\alpha : P \rightarrow G \tag{4.1}$$

Cada símbolo del conjunto P tendrá su correspondiente representación mediante primitivas en un sistema geométrico G dado. Dependiendo de la definición de α y del sistema geométrico G, la representación del sistema podrá variar. La función α proporciona una capa de abstracción con respecto a diferentes sistemas gráficos o de representación. Debido a esto, solo es necesario una única cadena descriptiva para reproducir la misma escena en diferentes sistemas de representación, tanto visuales como no visuales (como un sistema robótico, por ejemplo).

Función semántica de las transformaciones (Regla 5)

En la regla 5 se utilizan dos funciones para describir la semántica de una transformación, cuyo ámbito estará delimitado por los símbolos "()":

$$\begin{aligned} \beta &: T \rightarrow G \\ \delta &: T \rightarrow G \end{aligned} \tag{4.2}$$

La función β representa el inicio de una transformación, la cual será activada cuando se encuentre el símbolo "(". La función δ define el final de la transformación activada previamente por la función β. Esta función se realizará al encontrar el símbolo ")".

Función semántica de los actores (Regla 4)

La regla 4 define la evolución en el tiempo de los actores, por esta razón hemos denominado esta función como "*función de evolución*" λ. Su expresión general se define como:

$$\lambda : A^D_{ATTR} \times E^D \rightarrow L(M) \tag{4.3}$$

donde E^D es el conjunto de todos los eventos posibles. La función λ tendrá una expresión diferente dependiendo de la evolución del actor y de los eventos recibidos. Sin embargo podemos generalizar esta función como:

$$\lambda(a^H_{ATTR}, e^h) = \left\{ \begin{array}{ll} u_0 \in L(M) & if \;\; h = h_0 \\ \cdots & \\ u_n \in L(M) & if \;\; h = h_n \\ a^H_{ATTR} & if \;\; h \notin H \end{array} \right\} \tag{4.4}$$

donde $u_0, \ldots, u_n$ son cadenas de $L(M)$, y $\{h_0, \ldots, h_n\}$ es el subconjunto de eventos $H \subseteq D$ a los que el actor a^H_{ATTR} está preparado para responder. Esta función nos indica que un actor puede evolucionar y transformarse en otra cadena u_i cuando recibe un determinado evento. Sin embargo, el actor permanecerá inalterado cuando reciba algún evento que no pertenezca al conjunto de eventos para los cuales tiene definida alguna acción.

Para poder representar a los actores en un espacio geométrico G se deben convertir en una cadena de primitivas y transformaciones. Esta operación es llevada a cabo por la función de visualización θ:

$$\theta : A^D_{ATTR} \times E^V \rightarrow L(M') \tag{4.5}$$

donde $V \subseteq D$ es el conjunto de eventos creados en el proceso de visualización, y $L(M')$ es un subconjunto del lenguaje $L(M)$ formado por cadenas sin actores. Los eventos de visualización sirven para definir el tipo de visualización que se desea de entre los diferentes tipos de vista que tenga el sistema. Además nos permitirán filtrar determinados elementos y devolver diferentes representaciones según el sistema de dibujado.

Función semántica de: OBJECTO, OBJECTOS y MUNDO (Reglas 1, 2 y 3)

La función semántica de las reglas 1, 2 y 3 trocea las cadenas y las prepara para poder ser procesadas por el *algoritmo del sistema*. Este proceso es llevado a cabo mediante las funciones η, π y φ que veremos a continuación.

Se define la función η, llamada *función de evolución del sistema*, la cual realizará una serie de llamadas recursivas a la función λ para llevar a cabo la evolución de los actores:

$$\eta(w,S) = \left\{ \begin{array}{ll} w & if \;\; w \in P \\ t(\eta(v,S)) & if \;\; w = t(v) \wedge \; t \in T \\ \prod_{e^i \in S} \lambda(a^H_{attr}, e^i) & if \;\; w = a^H_{attr} \\ \eta(u,S) \cdot \eta(v,S) & if \;\; w = u \cdot v \wedge u, v \in L(M) \end{array} \right\} \tag{4.6}$$

Esta función recibe un conjunto de eventos ordenados $S = e^1 \cdot e^2 \ldots e^n$ y una cadena w de $L(M)$. El operador $\prod_{e^i \in S} \lambda(a^H_{attr}, e^i)$ realiza la concatenación de las cadenas devueltas por λ.

Para representar los actores contenidos en la cadena primero tenemos que transformarlos a primitivas y transformaciones. Para realizar esta acción se define la *función de visualización del sistema* π. La cual recibirá una cadena de $L(M)$ y una secuencia de eventos de visualización ordenada $S' = e^1 \cdot e^2 \dots e^n$, y devolverá una cadena del lenguaje $L(M')$:

$$\pi(w,S') = \left\{ \begin{array}{ll} w & if \;\; w \in P \\ t(\pi(v,S')) & if \;\; w = t(v) \\ \prod_{e^i \in S} \theta(a^H_{ATTR}, e^i) & if \;\; w = a^H_{ATTR} \\ \pi(u,S') \cdot \pi(v,S') & if \;\; w = u \cdot v \end{array} \right\} \tag{4.7}$$

Por último, para representar la cadena en un sistema geométrico G dado, definimos la función φ. La cual recibirá una cadena de $L(M')$ formada únicamente por primitivas y transformaciones:

$$\varphi(w) = \left\{ \begin{array}{ll} \alpha(w) & if \; w \in P \\ \beta(t);\varphi(v);\delta(t) & if \; w = t(v) \wedge v \in L(M) \wedge \; t \in T \\ \varphi(u);\varphi(v) & if \; w = u \cdot v \wedge u,v \in L(M) \end{array} \right\} \tag{4.8}$$

Eventos y generadores

Los eventos son el mecanismo utilizado para modelar la actividad del sistema. Estos eventos pueden ser producidos por otra actividad o por un dispositivo de entrada, y se definen como:

Se define e^d_c *como el evento de tipo* $d \in D$ *que tiene como datos asociados* c.

Cada actor, según su definición, estará preparado para realizar una determinada actividad cuando reciba un evento de un tipo concreto. El concepto de evento dentro de un actor es importante, ya que define cuándo se va a ejecutar la actividad independientemente del origen del evento. Esta independencia es necesaria para la generalización de los dispositivos de entrada, ya que independiza el dispositivo de la actividad que se debe procesar, y además facilita la realización de simulaciones.

Definimos también el concepto de *generador de eventos* $C^d(t)$ como una función que produce una secuencia ordenada de eventos de tipo d en el instante de tiempo t.

Es posible que estos generadores produzcan el mismo tipo de evento, por lo que se hace necesario establecer un orden de prioridad para evitar ambigüedades.

Dados dos generadores C_i y C_j, los cuales pueden crear el mismo tipo de evento, se establece que si $i < j$ entonces el sistema dará una mayor prioridad a los eventos generados por C_i.

Algoritmo del sistema

Una vez definidos todos los elementos implicados en el modelo se puede establecer el algoritmo que ejecutará todo el sistema, tanto la evolución como la visualización del mismo, para cada instante de tiempo t o *frame*:

1) $w = w_0$
2) $t = 0$
3) **mientras** $w \neq \varepsilon$ **hacer**
   - S = Obtener los eventos de los generadores C^* en orden de prioridad.
   - Z = Extraer los eventos visuales de S.
   - $w_{next} = \eta(w, S)$
   - $v = \pi(w, Z)$
   - $g = \varphi(v)$
   - $w = w_{next}$
   - $t = t + 1$
4) **fin mientras**

Resumidamente, los pasos que realiza este algoritmo son:

1. En primer lugar se realiza la inicialización del sistema: se asigna la cadena inicial w_o y se pone a cero el contador de tiempo.

2. A continuación se recogen los eventos del sistema llamando a los generadores de eventos C^* y se separan los eventos de visualización asignándolos a Z.

3. Seguidamente se llama a la *función de evolución del sistema* η con la cadena actual y el conjunto de eventos obtenidos S. Esta función nos devolverá la siguiente cadena w_{next} del sistema.

4. A continuación se llama a la *función de visualización del sistema* π para obtener una cadena v que contenga solamente los elementos a visualizar.

5. Esta cadena v se la pasamos a la función φ para realizar la representación en el sistema geométrico g dado.

6. Para continuar asignamos w_{next} a la cadena actual e incrementamos el contador de tiempo.

7. El algoritmo finalizará cuando *w* contenga una cadena vacía. Esta cadena se puede obtener mediante un evento especial que se genere cuando se desee terminar el algoritmo.

4.3 Caso práctico

En este caso práctico se plantea un sistema de navegación autónoma en el que un robot está programado para transportar objetos en un entorno conocido. Este robot dispone de varios sensores e *inputs* que le proporcionan la siguiente información:

- Un sensor de rango que le permitirá detectar obstáculos y distancias.
- Imagen de una cámara a color para identificar objetos y lugares mediante el uso de *landmarks* o marcadores.
- Una representación interna o mapa del entorno en el cual se va a mover.
- Instrucciones de un supervisor humano, el cual podrá únicamente podrá dar órdenes de alto nivel, como por ejemplo detener la tarea actual o empezar una nueva tarea.

Esta información será combinada utilizando un algoritmo multimodal basado en prioridades [30], para que el robot pueda atender las órdenes del supervisor y a la vez utilizar la información de sus sensores para evitar obstáculos, e indentificar objetos o lugares.

Un sistema de este tipo se puede modelar utilizando un esquema híbrido clásico, como el que se muestra en la figura 4.1, basado en la combinación de comportamientos reactivos y proactivos.

Se pretende adaptar este esquema y sus componentes al lenguaje de modelado GMV para aprovechar algunas de sus ventajas: independencia de los dispositivos de entrada y del sistema de representación. En la figura 4.1 el "Mundo" representa el entorno real y el "modelo del mundo" el mapa interno del que dispone el robot. El sistema reactivo estaría constituido por varios generadores de eventos para los sensores y las instrucciones del supervisor. El sistema proactivo

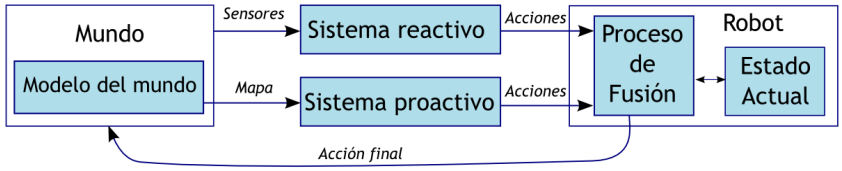


Figura 4.1: *Esquema de un sistema robótico híbrido.*

representaría la inteligencia del robot. El estado actual es el conjunto de atributos del robot y el proceso de fusión sería llevado a cabo por su función de evolución.

Primitivas y transformaciones

En este sistema solo necesitamos una primitiva para representar al robot, la cual puede ser modificada por dos posibles transformaciones: mover y girar (tabla 4.1). Cuando el sistema sea ejecutado en un entorno real la primitiva del robot representará el robot real, y las transformaciones a las acciones reales de movimiento realizadas por el robot. Si se utiliza en un simulador estas primitivas y transformaciones representarán instrucciones de visualización en el sistema gráfico (SG).

	Entorno real	Simulador
PRobot	Ninguna acción	Dibujar el robot en el SG
$TMove_{<dist>}$	Mover una distancia *dist*	Mover una distancia *dist* en el SG
$TRotate_{<angle>}$	Girar un ángulo *angle*	Girar un ángulo *angle* en el SG

Tabla 4.1: *Primitivas y transformaciones del sistema robótico.*

Eventos y generadores

En el sistema robótico propuesto se van a definir los siguientes generadores de eventos:

- *gLaser*: Genera un evento de tipo *eLaser* cuando detecta un obstáculo.

- *gCamera*: Genera un evento *eCamera* cuando se detecta un marcador en la imagen. Estos marcadores se utilizan para identificar las localizaciones en el entorno.

- *gDecide*: Genera un evento de tipo *eDecide* en cada frame para activar en el robot la acción de toma de decisiones.

- *gExecute*: Genera un evento *eExecute* que indica al sistema que tiene que llevar a cabo las acciones del robot en el espacio de representación actual. Si este espacio de representación es el entorno real se realizarán las operaciones reales sobre la plataforma robótica (mover, rotar, etc.), y si es el simulado se utilizarán las operaciones de representación en el sistema gráfico.

- *gObjective*: Genera un evento *eObjective* para establecer un nuevo marcador objetivo. Este generador será activado cuando el supervisor dé la instrucción de cambio de objetivo.

En la tabla 4.2 se muestra un resumen de los generadores de eventos del sistema y los eventos asociados.

Además se tiene que establecer un orden de prioridad entre los generadores. Este orden se ha definido como: *gLaser, gCamera, gObjective, gDecide, gExecute*. Por lo que los eventos relacionados con los sensores tienen una mayor prioridad que los de toma de decisiones y de ejecución.

Actores

En el sistema planteado el único actor es el propio robot, que se define como:

$$ARobot^{eLaser,eCamera,eDecide,eExecute,eObjective}_{<grid,row,column,angle,objective,action>} \tag{4.9}$$

donde el superíndice son los eventos para los que está preparado el robot y el subíndice sus atributos, cuyo significado es:

Generadores y eventos	Descripción	Datos asociados
$gLaser = eLaser_{<dist,angle>}$	Evento generado cuando el láser detecta un obstáculo	*dist*: distancia hasta el obstáculo; *angle*: ángulo con respecto al obstáculo
$gCamera = eCamera_{<marker>}$	Evento producido cuando la cámara detecta un marcador	*marker*: marcador detectado
gDecide = *eDecide*	Evento generado en cada frame para indicar al robot la toma de decisiones	Sin datos
gExecute = *eExecute*	Evento generado cada frame para ejecutar las acciones del robot en el entorno real o en el simulador	Sin datos
$gObjective = eObjective_{<mrkr>}$	Evento producido cuando el supervisor cambia el marcador objetivo	*marker*: marcador objetivo

Tabla 4.2: *Generadores y eventos del sistema robótico*

- El *grid* o rejilla representa el entorno en el que se mueve el robot. En cada celda de la rejilla se almacenan los datos obtenidos por los sensores para esa región del espacio (si por ejemplo se ha detectado un obstáculo o un marcador).
- *Row* y *column* son la posición ocupada por el robot en el *grid*.
- *Angle* es la orientación del robot.
- *Objective* es el marcador objetivo.
- *Action* es la cadena que indica la próxima acción a realizar por el robot.

Para simplificar, en la siguiente ecuación este actor será referido como:

$$ARobot^{E}_{<g,r,c,an,o,ac>}$$

En la función de evolución del robot definiremos su comportamiento, que en este caso comprenderá también su inteligencia artificial ante determinados eventos. Esta función se define como:

$$\lambda(ARobot^{E}_{<g,r,c,an,o,act>}, e) =$$

$$\begin{cases} ARobot^{E}_{<g',r,c,an,o,ac>} & if \;\; e = eLaser_{<dist,angle>} \\ ARobot^{E}_{<g',r,c,an,o,ac>} & if \;\; e = eCamera_{<marker>} \\ ARobot^{E}_{<g,r',c',an',o,ac'>} & if \;\; e = eDecide \\ \alpha(ARobot^{E}_{<g,r,c,an,o,ac>}) & if \;\; e = eExecute \\ ARobot^{E}_{<g,r,c,an,o',ac>} & if \;\; e = eObjective_{<marker>} \\ ARobot^{E}_{<g,r,c,an,o,ac>} & otherwise \end{cases} \tag{4.10}$$

donde e es el evento recibido por el actor y el símbolo de apóstrofe (') en un atributo indica que ese valor se ve modificado como consecuencia del evento recibido. A continuación se describe la forma en que cambian estos atributos:

- Si $e = eLaser_{<dist,angle>}$ se actualiza el *grid* (g) para indicar que se ha detectado un obstáculo. La celda a marcar se encontrará en la posición: $(r + dist\ \cos(ang + angle), c + dist\ \sin(ang + angle))$.
- Si $e = eCamera_{<marker>}$ se actualiza el *grid* (g) para indicar que se ha detectado un marcado. La celda a marcar será: $(r + dist\ \cos(ang), c + dist\ \sin(ang))$.
- Si $e = eDecide$ se actualiza la posición y orientación actual del robot (fila r, columna c y ángulo ang) y las acciones a realizar. En esta función es donde se podrá incluir la inteligencia del robot, para que tome una decisión u otra dependiendo del estado de todos sus atributos.
- Si $e = eExecute$ se representarán las acciones del robot en el espacio de visualización utilizado.
- Si $e = eObjective_{<marker>}$ significa que el supervisor ha seleccionado un nuevo objetivo, por lo que se asignará al objetivo actual "o" el nuevo valor ($marker$).
- En cualquier otro caso el actor permanecerá inalterado.

Cadena inicial

La cadena inicial del sistema se define de la forma:

$$ARobot^{eLaser,eCam.,eDecide,eExec.,eObjct.}_{<grid,row,column,angle,\varepsilon,\varepsilon>} \quad (4.11)$$

Análisis y resultados

Para el análisis del sistema se propusieron un conjunto de pruebas para la validación de diferentes características:

- Test de la función de evolución: se probaron varios algoritmos de inteligencia artificial implementados en esta función.
- Independencia de los dispositivos de entrada: esto se realizó añadiendo nuevos tipos de sensores, como la cámara Kinect.
- Independencia del sistema de visualización: se probaron diferentes sistemas 2D, 3D y entorno real.
- Test de simulación: se substituyeron algunos sensores por simuladores.
- Test de extensibilidad: se probó que con simples modificaciones en la cadena de entrada se podía añadir nuevas instancias de robots y crear de esta forma un sistema multi-robótico.

En la página 237 se puede consultar una descripción más detallada de la experimentación realizada y de los resultados obtenidos.

En la figura 4.2 se muestran un par de capturas del sistema desarrollado utilizando modelos de visualización 2D y 3D.

Otros casos de estudio

Además del ejemplo aquí propuesto se pueden consultar otros casos de estudio en los artículos correspondientes, como la simulación de incendios forestales producidos por tormentas (página 194) o la utilización del modelo GMV para trabajar con sistemas multiagente (página 209).

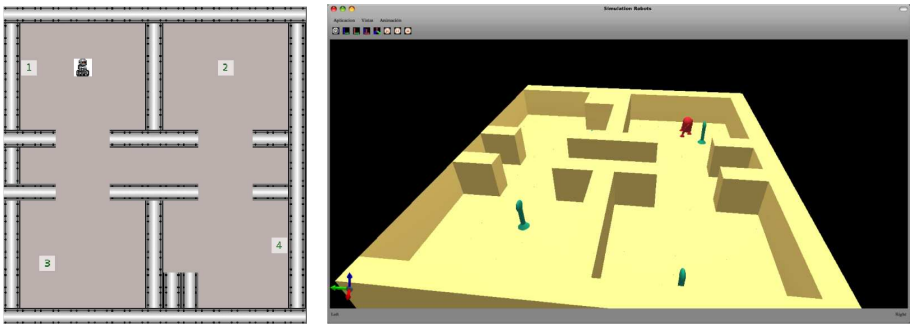


Figura 4.2: *Resultados de la simulación con diferentes sistemas de visualización 2D y 3D.*

4.4 Conclusiones y trabajo futuro

El modelo presentado permite definir formalmente a través de una gramática un sistema gráfico (o no gráfico como hemos visto) y la interacción con los dispositivos físicos de una forma generalizada. De esta forma se consigue la correcta separación de la actividad del sistema de la de los dispositivos visuales y de interacción.

La separación de los dispositivos de entrada de la definición del sistema se ha articulado a través de los generadores de eventos, que levantan una capa entre el hardware de entrada y la representación del sistema. Para enlazar las acciones generadas por los dispositivos de entrada se utiliza el concepto de eventos.

En general, se puede comprobar que todo el modelo intenta separar las partes dependientes del tipo de dispositivo de las de la definición formal de un sistema gráfico. La técnica empleada para dicha separación es utilizar modelos matemáticos que transformen las acciones de los dispositivos, tanto visuales como de entrada, en acciones más generales que puedan ser identificados por el sistema independientemente del origen de la acción.

Esto supone varias ventajas: en primer lugar los dispositivos de entrada pueden ser sustituidos por otros dispositivos o por simulaciones de estos. Por otro lado existe la posibilidad de que los elementos del sistema sean reutilizados. Además, la representación de los elementos puede ser visual o no visual, e incluso ser diferente dependiendo del dispositivo de visualización o de las necesidades del usuario. Así, si el dispositivo tiene unas características concretas, la representación siempre se podrá adaptar al dispositivo.

Como trabajo futuro se pretende estudiar la posibilidad de obtener la función inversa de la ecuación de dibujado φ definida en GMV. Esta función recibe como entrada una cadena w con la descripción de una escena y devuelve su representación gráfica en un sistema G dado, de la forma: $G = \varphi(w)$. Al calcular la inversa de esta función tendríamos que a partir de la representación gráfica de una escena (el modelo de datos propuesto en el capítulo anterior) se obtendría una cadena que lo describa: $w = \varphi^{-1}(G)$.

Como una primera aproximación se plantea el uso de gramáticas evolutivas y de grafos de adyacencias para realizar la estimación de esta función inversa. En el primer caso, siguiendo las reglas gramaticales establecidas por GMV, se generan cadenas mediante el uso de algoritmos evolutivos, y posteriormente se comparan con la estructura del espacio de la escena a obtener. En el segundo caso se aborda el problema analizando la estructura del espacio y generando grafos de adyacencias, sobre los cuales se pretende utilizar el algoritmo de inferencia de lenguajes k-testables para grafos que se propone en el capítulo siguiente. Este algoritmo nos permite la comprobación de la pertenencia a un lenguaje $L(M')$ dado de las estructuras k-testables extraídas de estos grafos.

Capítulo 5

Lenguajes *k*-testables para grafos

En este capítulo se resume el contenido de los artículos "*Inference of k-testable directed acyclic graph languages*" (página 249) y "*Structural Graph Extraction from Images*" (página 271) en los cuales se propone un algoritmo de inferencia para aprendizaje de lenguajes *k*-testables para grafos acíclicos dirigidos. Además también se presentan tres nuevos métodos para la extracción de este tipo de grafos a partir de imágenes.

5.1 Introducción

Los algoritmos de inferencia en lenguajes de cadenas regulares son ampliamente utilizados en muchas tareas, desde la bioinformática [49, 42] hasta el reconocimiento de scripts [45]. En el trabajo [32] se puede encontrar un completo estudio bibliográfico.

Sin embargo en muchos ámbitos es necesario el uso de información estructural, la cual no se puede modelar fácilmente con subclases de un lenguaje regular de cadenas. En trabajos como los de Sakakibara [47, 48] u otros más recientes [41, 31] se presentan algoritmos que trabajan con este tipo de información mediante gramáticas independientes del contexto de varios tipos.

Las estructuras en forma de árbol son en realidad un tipo simple de grafo pero con unas propiedades muy interesantes. En varios trabajos se estudian los lenguajes de inferencia para árboles, como en [33, 39], o su aplicación a tareas

reales [44, 37]. En el campo de la inferencia gramatical, cuando se consideran grafos más generales, aparece el problema de la complejidad computacional, lo cual se suele solucionar reduciendo la representación de los grafos a algún tipo de recorrido por su estructura.

A pesar de esta complejidad existe un gran número de campos que aprovechan la gran potencia de representación que proporcionan los grafos, como por ejemplo la biología, sociología, o una amplia variedad de algoritmos matemáticos e informáticos como el problema del viajante. El poder expresivo de las primitivas de un grafo es especialmente utilizado en tareas de reconocimiento de patrones para la representación de objetos [53] o para la bioinformática [57]. Esto es debido a que los grafos proporcionan una estructura apropiada para la representación de cualquier clase de relación entre los datos o componentes de un problema dado.

En este trabajo se consideran los grafos como elementos de un lenguaje formal. En este marco, en el trabajo de Rozemberg [46] se resumen, entre otros resultados, los dos principales formalismos utilizados para generar lenguajes de grafos (gramáticas de sustitución de nodos e hyper-arcos). Pero a pesar de todos estos trabajos sobre el paradigma generador, no existe ninguna solución sobre el reconocimiento que encaje correctamente todas las clases de lenguajes de grafos.

En algunos trabajos se estudia las gramáticas de inferencia para grafos, pero todas ellas se basan en el isomorfismo de grafos, y, por lo tanto, tienen una alta complejidad computacional [35, 36, 38].

En la solución propuesta se extiende la familia de lenguajes *k*-testables y *k*-testables en el sentido estricto (*k*-TSS) [40] a los lenguajes para grafos acíclicos dirigidos. Seguidamente, y teniendo en cuenta el trabajo de [43], se propone un modelo de autómata para lenguajes de grafos acíclicos dirigidos. Finalmente se presenta un nuevo algoritmo de inferencia gramatical para el aprendizaje de la clase *k*-TSS de grafos acíclicos dirigidos. El algoritmo propuesto tiene una complejidad polinómica e identifica esta clase de lenguajes a partir de datos positivos.

5.2 Notación y definiciones

Se define un grafo etiquetado dirigido (de ahora en adelante referido simplemente como grafo si no se indica otra cosa) como una tupla $g = (V, E, \mu)$, donde V es el conjunto finito de nodos (también llamados vértices), $E \subseteq (V \times V)$ es el conjunto de arcos, y $\mu : V \to \Sigma$ es la función de etiquetado de los nodos. Nos

referiremos a los componentes de un grafo g como V_g, E_g y μ_g solo cuando sea necesario.

Para un nodo dado v, los arcos de entrada serán aquellos en sentido (u,v), respectivamente los de salida lo serán en (v,u). La aridad de entrada de un nodo v se define como $idg(v) = |\{(u,v) \in E\}|$, mientras que su aridad de salida vendrá dado por $odg(v) = |\{(v,u) \in E\}|$. Para un grafo dado $g = (V,E,\mu)$, denotaremos sus vértices de la forma $V_m^n(g) = \{v \in V \ : \ idg(v) = n \wedge odg(v) = m\}$. Además utilizaremos ε_g para referenciar los grafos vacíos (un grafo sin nodos), y $V^0(g)$ y $V_0(g)$ para los nodos con aridad cero de entrada o salida respectivamente.

Definimos el alfabeto tipado Σ_τ^σ como la asociación de un alfabeto Σ con la relación $r \subseteq (\Sigma \times \mathbb{N} \times \mathbb{N})$. Dado que los nodos de un grafo pueden tener diferentes grados de entrada y de salida, es necesario establecer los símbolos que pueden etiquetar esos nodos. Denotaremos como Σ_m^n al conjunto $\{s \in \Sigma \ : \ (s,n,m) \in r\}$, el cual nos indica los símbolos disponibles para etiquetar los nodos con aridad de entrada n y de salida m.

Una vez definido el alfabeto tipado, podemos establecer el conjunto de todos los posibles grafos etiquetados sobre este alfabeto. Formalmente, $\mathscr{G}(\Sigma_\tau^\sigma)$ denota el conjunto de grafos que utiliza el alfabeto Σ_τ^σ. Un *lenguaje de grafos* es cualquier conjunto formado como $L_G \subseteq \mathscr{G}(\Sigma_\tau^\sigma)$.

Dado un alfabeto tipado Σ_τ^σ, el *alfabeto extendido* se define como el conjunto:

$$\widehat{\Sigma} = \{a_m^n \ : \ a \in \Sigma, \ (a,n,m) \in r\}$$

A partir del *alfabeto extendido*, y dado un grafo $g = (V,E,\mu)$, se define su *grafo extendido* como $\hat{g} = (V,E,\hat{\mu})$, donde $\hat{\mu} : V \to \widehat{\Sigma}$, de tal manera que para cada nodo v en $V_m^n(g)$, si $\mu(v) = a$, entonces $\hat{\mu}(v) = a_m^n$. Intuitivamente, las etiquetas de los nodos del grafo extendido $\hat{g}$ incluyen explícitamente la aridad de entrada y de salida de cada nodo del grafo. En la figura 5.1 se muestra un ejemplo.

En adelante se considerará por simplicidad únicamente el esqueleto de los grafos y lenguajes para estos. Por esqueleto de un grafo nos referimos a grafos cuyos nodos internos v (con $odg(v) \neq 0$) están etiquetados con el mismo símbolo. Sin embargo, los resultados obtenidos se pueden utilizar de la misma forma con grafos de tipo general. Además, para clarificar la representación de los grafos se utilizarán letras griegas para los nodos internos y letras del alfabeto latino para los nodos frontera (aquellos con aridad de salida cero).

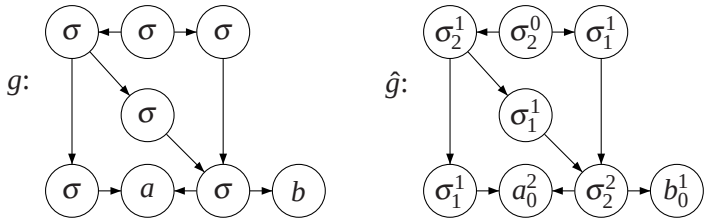


Figura 5.1: *Ejemplo de un grafo acíclico dirigido g y su correspondiente grafo extendido ĝ. Por claridad se han etiquetado los nodos internos (aquellos con aridad de salida mayor que cero) con letras griegas.*

Para una secuencia de nodos $w_1, w_2, \ldots, w_k$, si $(w_i, w_{i+1}) \in E$ para $1 \leq i < k$, entonces decimos que existe un camino desde w_1 hasta w_k. Se define la longitud de este camino como el número de nodos en la secuencia. En un grafo es posible que existan varios caminos entre un par de nodos u y v dados. Se denota el conjunto de los caminos más cortos desde u hasta v por $((u,v))$, y la longitud del camino (o caminos) más corto como $|((u,v))|$. Si un camino no existe entonces su longitud será infinita, además la longitud de un nodo a si mismo es $|((u,u))| = 1$. Se define el *diámetro* de un grafo $g = (V,E,\mu)$ como la máxima distancia entre dos nodos del grafo:

$$diameter(g) = \max_{u,v \in V} \{|((u,v))| \ : \ |((u,v))| < \infty\}$$

Dado un grafo $g = (V,E,\mu)$, el *subgrafo* de g con *raíz* en el nodo v y radio k se define como $R_g(v,k) = (W,E',\mu')$ tal que $W = \{u \in V \ : \ |((v,u))| \leq k\}$ y donde $E' = E \cap (W \times W)$, esto es, el conjunto de arcos restringido a los nodos en W. De la misma forma, μ' es la restricción de la función μ a los nodos en W. Además extendemos esta definición para considerar, para un grafo $g = (V,E,\mu)$, el *subgrafo* de g con *raíz* en el nodo v denotado por $R_g(v) = (W,E',\mu')$ donde $W = \{u \in V \ : \ |((v,u))| < \infty\}$ con el conjunto E' y la función de etiquetado μ' definidas de la misma forma.

5.2.1 Multiconjuntos

Recordamos algunas definiciones de la teoría de multiconjuntos que utilizaremos en la función de transición del nuevo modelo de autómata para grafos. En el texto siguiente denotaremos al conjunto de los números naturales como $\mathbb{N}$.

Para un conjunto dado D, un *multiconjunto* de D es un par $\langle D, f\rangle$ donde f es una función de enumeración de la forma $f : D \to \mathbb{N}$. Esto es, para un $a \in D$, la función $f(a)$ denota el número de elementos de a en el multiconjunto. Además decimos que a se encuentra en A, y escribimos $a \in A$, si y solo si $f(a) \neq 0$. El tamaño de un multiconjunto se define como el número de elementos que contiene. Este número puede ser finito, en este caso el multiconjunto será finito. El tamaño de un multiconjunto M se denota por $|M|$. En particular nos interesan los tipos de multiconjuntos cuyo tamaño es igual a una constante n. Esto es, la clase de todos los multiconjuntos $\langle D, f\rangle$ tal que $\sum_{a\in D} f(a) = n$. En lo que sigue denotaremos esta clase por $\mathscr{M}_n(D)$.

Decimos que un multiconjunto $A\langle D, f\rangle$ está *vacío* si y solo si para todo $a \in D$, $f(a) = 0$. De esta forma, para cualquier par de multiconjuntos $A = \langle D, f\rangle$ y $B = \langle D, g\rangle$, decimos que $A = B$ si y solo si para todo $a \in D$, $f(a) = g(a)$, y, de la misma forma, A es un *subconjunto* de B ($A \subseteq B$) si y solo si para todo $a \in D$, $f(a) \leq g(a)$. Además definimos la suma de dos multiconjuntos $A = \langle D, f\rangle$ y $B = \langle D, g\rangle$ (denotado $A \oplus B$) como el multiconjunto $C = \langle D, h\rangle$ donde para todo $a \in D$, $h(a) = f(a) + g(a)$. Por último, se extiende la definición de *conjunto potencia* a multiconjuntos, de la forma: dado un multiconjunto C, su conjunto potencia es el conjunto de todos los posibles subconjuntos de C, y se denotará como 2^C.

Un concepto muy útil para trabajar con multiconjuntos es el *mapeado de Parikh*. Formalmente, el *mapeado de Parikh* puede verse como la aplicación $\Psi : D^* \to \mathbb{N}^n$ donde $D = \{d_1, d_2, \ldots, d_n\}$ y D^* es el conjunto de cadenas sobre D. Para cualquier $x \in D^*$, este mapeado se define como $\Psi(x) = (\#_{d_1}, \#_{d_2}, \ldots, \#_{d_n})$ donde $\#_{d_i}$ denota el número de ocurrencias de d_j en x. Esto nos permite representar un multiconjunto utilizando cualquier cadena con el *mapeado de Parikh* correcto (es decir, no importa la ordenación). Este concepto lo utilizaremos a continuación.

5.3 Lenguaje *k*-testable para grafos

Los lenguajes testables y testables en el sentido estricto [40] se definen mediante un vector (I, S, F) que representa las estructuras permitidas en los miembros del lenguaje. Estos conceptos están definidos para lenguajes de cadenas y árboles, como podemos ver en [34, 33]. En esta sección extendemos su definición para considerar los lenguajes de grafos.

Dado un alfabeto tipado Σ_τ^σ y el correspondiente conjunto de grafos sobre este alfabeto $\mathscr{G}(\Sigma_\tau^\sigma)$, para cualquier $g = (V, E, \mu) \in \mathscr{G}(\Sigma_\tau^\sigma)$, definimos el *vector*

de k-testabilidad $T_k(g) = (I_{k-1}(g), P_k(g), F_{k-1}(g))$ donde:

$$
\begin{aligned}
I_{k-1}(g) &= \left\{ R_{\hat{g}}(v, k-1) \ : \ v \in V^0(g) \right\} \\
P_k(g) &= \left\{ R_{\hat{g}}(v, k) \ : \ v \in V, \ diameter(R_g(v)) \geq k \right\} \\
F_{k-1}(g) &= \left\{ R_{\hat{g}}(v, k-1) \ : \ v \in V, \ diameter(R_g(v)) \leq k-1 \right\}
\end{aligned}
$$

El conjunto $P_k(g) = \emptyset$ si $diameter(g) < k$. A continuación se incluyen algunos ejemplos.

Ejemplo. Dado el grafo de la figura 5.1, en la figura 5.2 se muestran los componentes del vector 3-testable extraídos. Los nodos de los subgrafos de cada componente de un vector k-testable se etiquetan mediante la función μ extendida $\hat{\mu}$. Para cada nodo, la función de etiquetado extendida respecta la aridad de entrada y salida que tenía en el grafo original. Por ejemplo, los nodos etiquetados con a_0^2 en la figura 5.2 no tienen siempre dos arcos de entrada, sin embargo se mantiene este etiquetado porque es el que tenía en el grafo original. Esta función extendida permite tratar de forma sencilla información estructural muy importante, la cual utilizaremos posteriormente en el algoritmo de inferencia propuesto.

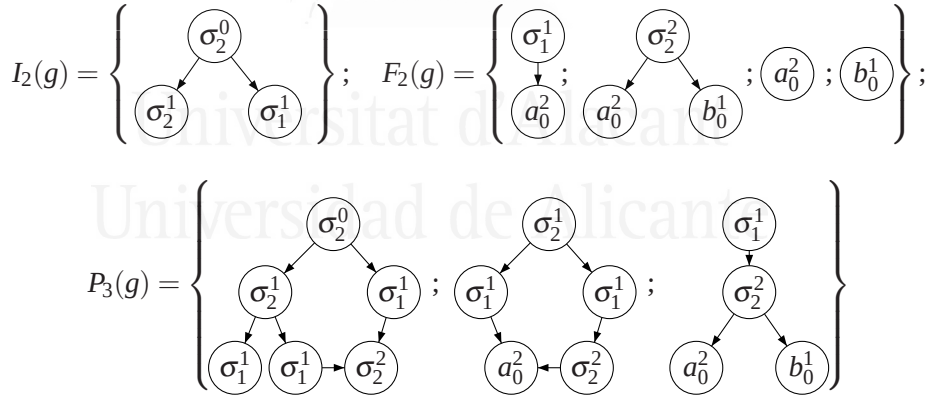
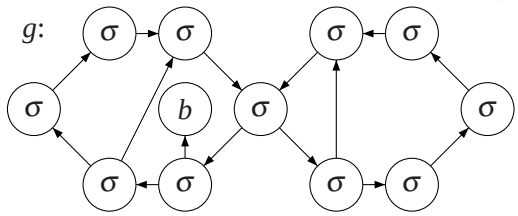


Figura 5.2: *Componentes del vector* 3*-testable obtenidos a partir del grafo de la figura 5.1.*

Para obtener el vector k-testable no es necesario imponer la condición de grafo acíclico. Como ejemplo, en la figura 5.3 se muestra los componentes del vector

2-testable a partir de un grafo cíclico. En este caso el conjunto $I_1(g)$ está vacío dado que no hay ningún nodo inicial, es decir, el conjunto $V^0(g)$ está vacío.



$$I_1(g) = \emptyset; \quad F_1(g) = \left\{ \left(b_0^1\right) \right\};$$

$$P_2(g) = \left\{ \begin{pmatrix} \sigma_1^1 \\ \sigma_1^2 \end{pmatrix}; \begin{pmatrix} \sigma_1^2 \\ \sigma_2^2 \end{pmatrix}; \begin{pmatrix} \sigma_2^2 \\ \sigma_2^1 \quad \sigma_2^1 \end{pmatrix}; \begin{pmatrix} \sigma_2^1 \\ b_0^1 \quad \sigma_2^1 \end{pmatrix}; \begin{pmatrix} \sigma_2^1 \\ \sigma_1^2 \quad \sigma_1^1 \end{pmatrix}; \begin{pmatrix} \sigma_1^1 \\ \sigma_1^1 \end{pmatrix} \right\}$$

Figura 5.3: *Grafo dirigido y su correspondiente vector* 2*-testable.*

Las funciones I_k, F_k y P_k se pueden extender de forma natural a un conjunto de grafos G de la forma:

$$I_k(G) = \{I_k(g) : g \in G\}; \quad P_k(G) = \{P_k(g) : g \in G\}; \quad F_k(G) = \{F_k(g) : g \in G\}$$

Para cualquier grafo G con $k \geq 2$, se cumple la propiedad $L_{k+1}(G) \subseteq L_k(G)$. Esta propiedad está demostrada para lenguajes de cadenas [34] y árboles [33], en este caso se necesita demostrar que, para cualquier $g \in L_{k+1}(G)$, g se encuentra también en $L_k(G)$, o en otras palabras, se tiene que demostrar que para cualquier $g \in L_{k+1}(G)$, $I_{k-1}(g) \subseteq I_{k-1}(G)$, $F_{k-1}(g) \subseteq F_{k-1}(G)$ y $P_k(g) \subseteq P_k(G)$.

Los conjuntos $I_{k-1}(G)$ y $F_{k-1}(G)$ se pueden obtener a partir de $I_k(G)$ y $F_k(G)$ de la forma:

$$I_{k-1}(G) = I_{k-1}(I_k(G))$$
$$F_{k-1}(G) = F_{k-1}(F_k(G))$$

En el caso de $P_k(G)$ y $P_{k+1}(G)$, para cada grafo $g \in L_{k+1}(G)$, se distinguen dos casos:

- Si $diameter(g) \leq k$, entonces $P_{k+1}(g) = \emptyset$, el cual es un subconjunto de $P_k(G)$.
- Si $diameter(g) > k$, entonces $P_k(g) = P_k(P_{k+1}(g)) \subseteq P_k(P_{k+1}(G)) = P_k(G)$.

Por lo tanto, cualquier grafo que cumpla las condiciones de un lenguaje $(k+1)$-testable también cumplirá las del k-testable, o expresado formalmente: $L_{k+1}(G) \subseteq L_k(G)$

En la página 254 se puede consultar una definición más extensa del lenguaje k-testable y k-testable en el sentido estricto [40] para grafos que hemos propuesto, así como la demostración de otras propiedades de los lenguajes k-testables.

En la siguiente sección se propone un nuevo modelo de autómata de grafos para grafos acíclicos diridos y se utiliza este modelo para proponer un algoritmo de inferencia gramatical.

5.4 Autómata para grafos

El modelo de autómata propuesto se basa en el trabajo de Potthoff [43]. Este autómata no puede ser utilizado para procesar toda clase de grafos dirigidos, sino solamente aquellos sin ciclos. La función de transición del autómata utiliza el concepto de multiconjunto explicado antes, el cual nos permite procesar los grafos de una forma natural sin tener en cuenta el orden entre los nodos, excepto el orden parcial impuesto por los arcos dirigidos.

El análisis de grafos es, en esencia, similar al análisis realizado para los lenguajes de árboles, pero considerando que no existe orden entre los nodos hermanos. Este proceso se puede realizar en tiempo polinómico utilizando un esquema de programación dinámica similar al propuesto por Zhang en [50] para obtener una distancia de edición entre dos árboles sin ordenación.

Definición. Sea Σ_τ^σ un alfabeto tipado y m la aridad máxima de salida en $\widehat{\Sigma}$. Un autómata para grafos que use el lenguaje Σ_τ^σ se define como una tupla $GA = (Q, \widehat{\Sigma}, \delta, F)$, donde Q es un conjunto finito de estados, $\widehat{\Sigma}$ es el alfabeto extendido de Σ_τ^σ, el conjunto $F \subseteq Q$ contiene los estados finales y δ es un conjunto de funciones de transición definidas como:

$$\delta = \bigcup_{\substack{0 \leq j \leq m \\ j:\ \exists n > 0, \Sigma_j^n \neq \emptyset}} \delta_j$$

donde cada δ_j se define como:

$$\delta_j : \widehat{\Sigma}_j^n \times \mathscr{M}_j(Q) \to 2^{\mathscr{M}_1(Q)}, \qquad 0 \leq j \leq m$$

y donde $\mathscr{M}_j$ representa la clase de multiconjunto de tamaño j según se definió en la sección 5.2.1.

Estas funciones de transición utilizan el alfabeto extendido en lugar del original. Esto permite procesar los grafos teniendo en cuenta tanto la aridad de salida (que sería el tamaño del multiconjunto) como la aridad de entrada (almacenada en el símbolo correspondiente del alfabeto extendido). Al extender la función de transición para que funcione en grafos, se tiene que realizar un análisis recursivo a partir de cada nodo cuya aridad de entrada sea cero. Formalmente, para un grafo g dado la función δ se extiende de la forma:

$$\delta : \mathscr{G}(\widehat{\Sigma}) \to 2^{\mathscr{M}(Q)}$$
$$\delta(g) = \delta(\hat{g}) = \bigoplus_{v_i \in V^0(\hat{g})} \delta(R_{\hat{g}}(v_i))$$

donde:

$$\delta(R_g(v_i)) = \delta_m(\mu_{\hat{g}}(v_i), M_{i1} \oplus \ldots \oplus M_{im}) : \; M_{ij} \in \delta(R_g(w_j)),\; (v_i, w_j) \in E$$

El lenguaje aceptado por este autómata se define como:

$$L(GA) = \{g \in \mathscr{G}(\Sigma_\tau^\sigma) \; : \; \forall q \in \delta(\hat{g}),\; q \in F\}$$

Por lo que un grafo g es aceptado por el autómata GA si y solo si la función de transición devuelve un multiconjunto para el que todo estado q, cuya función de enumeración sea distinta de cero, dicho estado sea final. Estos conjuntos serán denominados multiconjuntos finales.

Ejemplo. En la figura 5.4 se puede ver un esquema del análisis realizado por el autómata propuesto sobre del grafo de la figura 5.1. En la siguiente tabla se puede ver las funciones de transición extraídas:

$$\begin{aligned}
\delta(a_0^2,\lambda) &= q_1\\
\delta(b_0^1,\lambda) &= q_2\\
\delta(\sigma_2^2,q_1q_2) &= q_1\\
\delta(\sigma_1^1,q_1) &= q_2\\
\delta(\sigma_2^1,q_2q_2) &= q_1\\
\delta(\sigma_2^0,q_1q_2) &= q_1, \text{ where } q_1 \in F
\end{aligned}$$

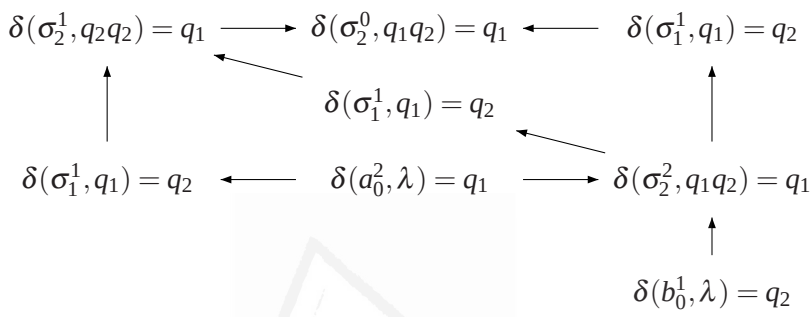


Figura 5.4: *Ejemplo del parsing realizado por el autómata para el grafo de la figura 5.1. El grafo se recorre recursivamente hasta alcanzar los nodos cuya aridad de salida sea cero. Las flechas muestran el orden del proceso de parsing.*

5.5 Algoritmo de inferencia

En esta sección proponemos el algoritmo de inferencia de lenguajes *k*-testables para grafos (ver algoritmo 1). Este algoritmo sigue el mismo esquema que se ha utilizado previamente para inferir lenguajes *k*-testables para cadenas y árboles [34, 33]. En primer lugar el algoritmo obtiene el conjunto de estados a partir de las estructuras del grafo de diámetro $k-1$ en el vector *k*-testable de la muestra de entrada. Seguidamente define el conjunto de estados finales (F o estados de aceptación) y extrae las funciones de transición usando los subgrafos de $F_{k-1}(G)$ y $P_k(G)$. A continuación se incluye una traza de ejemplo de este algoritmo.

Ejemplo. A partir del conjunto G de grafos mostrados en la figura 5.5 y con $k = 2$ se extrae los elementos del vector 2-testable que se muestran en la figura 5.6.

En primer lugar, el algoritmo construye el conjunto de estados a partir de $I_1(G)$, $F_1(G)$ y $I_1(P_2(G))$, de la forma:

Algoritmo 1. Algoritmo de inferencia gramatical a partir de muestras positivas para lenguajes k-testables de grafos.

Requiere: Un conjunto G de grafos. Un valor $k \geq 2$
Método:
Calcular $(I_{k-1}(G), P_k(G), F_{k-1}(G))$
Sea Σ_τ^σ el alfabeto tipado de G y $\widehat{\Sigma}_\tau^\sigma$ su alfabeto extendido.
for $\{g \in \{I_{k-1}(G) \cup F_{k-1}(G) \cup I_{k-1}(P_k(G))\}\}$ **do**
  Sea $Q[g]$ el nuevo estado obtenido de g
end for
$F = \{Q[g] \ : \ g \in I_{k-1}(G)\}$ // Estados de aceptación
for all $\{g \in F_{k-1}(G)$ con $(v, w_i) \in E_g$, $v \in V^0(g)$, $1 \leq i \leq m\}$ **do**
  $\delta_m(\mu(v), Q[R_g(w_1)] \ldots Q[R_g(w_m)]) = Q[g]$
end for
for all $\{g \in P_k(G)$ con $v \in V^0(g)$, $(v, w_i) \in E_g$, $1 \leq i \leq m\}$ **do**
  $\delta_m(\mu(v), Q[R_g(w_1, k-1)] \ldots Q[R_g(w_m, k-1)]) = Q[R_g(v, k-1)]$
end for
return $(Q, \widehat{\Sigma}_\tau^\sigma, F, \delta)$
Fin Método

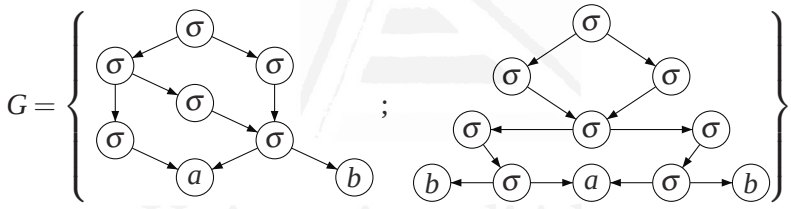
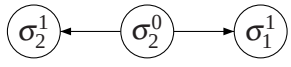


Figura 5.5: *Conjunto de grafos de ejemplo.*

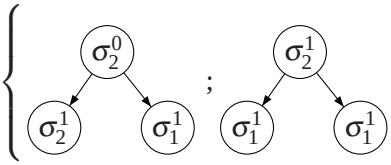
$$Q[(a_0^2)] = q_1;\ Q[(b_0^1)] = q_2;\ Q[(\sigma_2^2)] = q_3;\ Q[(\sigma_1^1)] = q_4;\ Q[(\sigma_2^1)] = q_5;\ Q[(\sigma_2^0)] = q_6$$

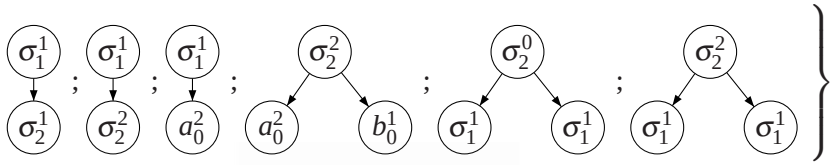
El algoritmo obtiene el conjunto de estados finales (o de aceptación), que en este caso es $F = \{q_6\}$. A continuación procesa los grafos en $F_{k-1}(G)$ (cuyo diámetro en este caso es de 1), obteniendo las transiciones: $\delta(a_0^2, \lambda) = q_1$ y $\delta(b_0^1, \lambda) = q_2$ (en este caso λ denota la cadena vacía). Seguidamente el algoritmo procesa los grafos en $P_k(G)$. Por ejemplo, para el siguiente grafo en $P_2(G)$:



$$I_1(G) = \left\{ \sigma_2^0 \right\}; \quad F_1(G) = \left\{ a_0^2 \; ; \; b_0^1 \right\};$$

$$P_2(G) = \left\{ \right.$$





$\left. \right\}$

Figura 5.6: *Elementos del vector* 2*-testable para los grafos de ejemplo de la figura 5.5.*

El algoritmo extrae los subgrafos de tamaño $k-1$ con raíz en los nodos hijos de σ_2^0 y el grafo de diámetro $k-1$ con raíz en el nodo σ_2^0. En este caso el algoritmo añade la transición: $\delta(\sigma_2^0, q_5 q_4) = q_6$.

Una vez procesadas todas las estructuras en el vector k-testable se obtiene el siguiente autómata:

$\delta(a_0^2, \lambda) = q_1$	$\delta(\sigma_1^1, q_5) = q_4$
$\delta(b_0^1, \lambda) = q_2$	$\delta(\sigma_1^1, q_1) = q_4$
$\delta(\sigma_2^0, q_5 q_4) = q_6$, donde $q_6 \in F$	$\delta(\sigma_2^2, q_1 q_2) = q_3$
$\delta(\sigma_2^1, q_4 q_4) = q_5$	$\delta(\sigma_2^0, q_4 q_4) = q_6$, donde $q_6 \in F$
$\delta(\sigma_1^1, q_3) = q_4$	$\delta(\sigma_2^2, q_4 q_4) = q_3$

En la página 262 se puede consultar una explicación más completa del algoritmo de inferencia propuesto, así como otras trazas y ejemplos. Además también se demuestran en estas páginas dos propiedades muy importantes del algoritmo, las cuales son:

- Identifica en el límite a partir de muestras positivas.
- Se ejecuta en tiempo polinómico.

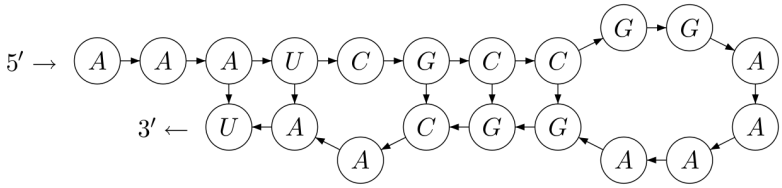


Figura 5.7: *Ejemplo de representación mediante un grafo de una molécula ARN con hairpin loops con la secuencia AAAUCGCCGGAAAAGGCAAU y dos bucles. Se ha seguido el orden* $5' \rightarrow 3'$ *para definir la orientación de los arcos del grafo.*

5.6 Experimentación

A continuación se presentan los resultados experimentales obtenidos de la aplicación del método propuesto al reconocimiento de grafos acíclicos dirigidos. Para realizar esta tarea se han generado una base de datos sintética de grafos que representan moléculas ARN. Concretamente se han utilizado las moléculas conocidas como "*hairpin RNA molecules*". Esta estructura biológica se forma a partir de subsecuencias complementarias de ARN las cuales forman bucles o *hairpin loops* en sus límites.

Para construir una estructura de este tipo definimos el alfabeto Σ con las cuatro posibles bases del ARN: adenina (A), guanina (G), timina (U) y citosina (C), donde la adenina y la timina al igual que la citosina y la guanina son complementarias. Estas relaciones complementarias se utilizan para la formación de la cadena y para obtener la posición de los bucles. Como se puede observar en la figura 5.7 cuando en el plegado de la cadena aparezcan bases no complementarias se formará un bucle. Además también se considera el orden de la secuencia proporcionado por la estructura molecular para definir la orientación de los arcos del grafo (desde la molécula situada en el extremo $5'$ hasta el otro extremo $3'$).

La base de datos de grafos de este tipo se ha generado siguiendo los siguientes criterios:

- Los valores de las bases se obtuvo de forma aleatoria, pero siguiendo los criterios explicados para el pareado.
- La longitud de los bucles se genera de forma aleatoria siguiendo una distribución normal con una media de 6 bases y una desviación estándar de 1.5 bases.

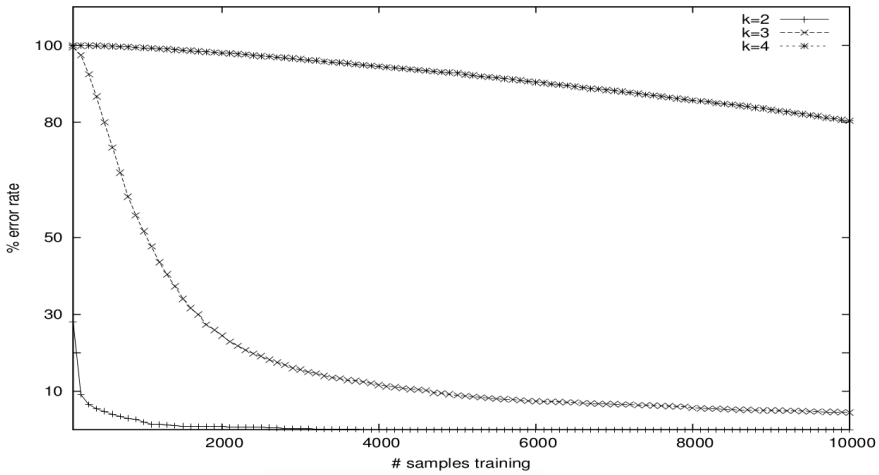


Figura 5.8: *Resultados de la experimentación utilizando la base de datos sintética de moléculas ARN de tipo hairpin.*

- La longitud de las moléculas se obtiene de forma aleatoria en el rango [10, 60].
- La posición del bucle se genera también de forma aleatoria, permitiendo secuencias sin bucles (las cuales suceden con una probabilidad de 0.2%).

En el proceso de inferencia se utilizaron conjuntos de entrenamiento incrementales y considerando valores de k desde 2 hasta 4. En el test se utilizaron tres conjuntos de 10000 muestras sin solapamiento para las cuales se comprobó su pertenencia al lenguaje. En la figura 5.8 se muestra la media de los resultados obtenidos.

Como se esperaba, los lenguajes 2-testables necesitan menos información para converger al lenguaje objetivo. Esto es debido a que las estructuras de tamaño menor generalizan más, y por lo tanto se necesitan menos muestras para representar el lenguaje.

En el siguiente experimento se tuvo en cuenta únicamente la información estructural de los grafos (sin etiquetas). La experimentación se realizó de la misma forma, obteniendo los resultados mostrados en la figura 5.9.

Estos resultados muestran que los lenguajes k-testables para grafos propuestos son adecuados para modelar estructuras biológicas. Sin embargo, no dispone-

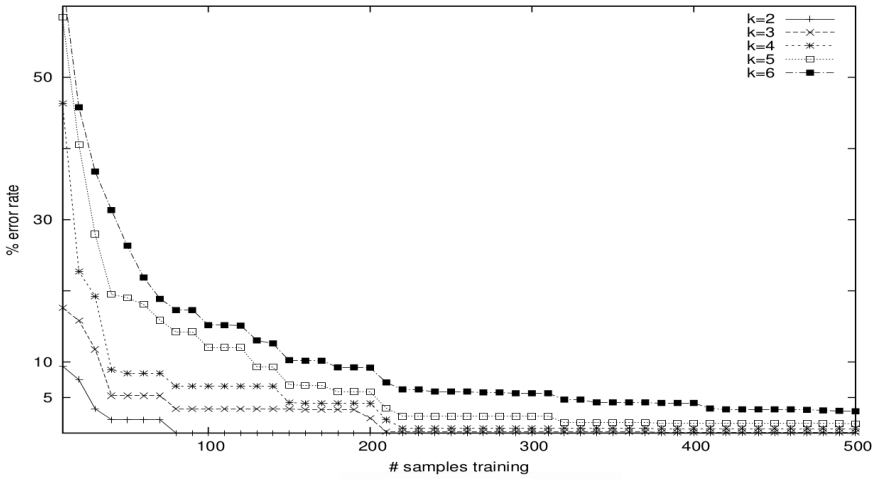


Figura 5.9: *Resultados de la experimentación considerando únicamente la estructura de los grafos.*

mos de bases de datos reales para aplicar nuestro algoritmo en otro tipo de tareas como la clasificación. Por este motivo se realizó el trabajo que se propone en la siguiente sección, en la cual se proponen nuevos algoritmos para la extracción de grafos acíclicos dirigidos que representen estructuras reales.

5.7 Grafos acíclicos dirigidos a partir de imágenes

En el artículo "*Structural Graph Extraction from Images*" (página 271) se presentan tres nuevos algoritmos que permiten extraer y representar la estructura de una imagen a partir de grafos acíclicos dirigidos. Este trabajo fue motivado por la necesidad de generar bases de datos reales de este tipo de grafos para realizar la experimentación de los lenguajes *k*-testables para grafos.

En las siguientes secciones se describen los métodos propuestos: (1) grafos obtenidos de la vecindad del árbol q-tree de una imagen, (2) mediante el uso de una rejilla estructural y (3) a partir de la extracción del esqueleto que representa una imagen.

5.7.1 Grafos de vecindad

El primer método extrae el grafo acíclico dirigido a partir de los caminos de vecindad obtenidos en la representación en forma de q-tree de una imagen.

El grafo se crea siguiendo un recorrido de arriba hacia abajo y de izquierda a derecha del árbol q-tree y uniendo los nodos no vacíos encontrados. Para cada nodo no vacío se calculan sus 8-vecinos, teniendo en cuenta que: no se añaden arcos hacia nodos que ya hayan sido procesados, y que cuando se tenga que crear un arco desde un nodo no apuntado (que no existan arcos que lleguen a él) hasta uno que sí que lo esté, se deberá cambiar la dirección de dicho arco. Estos criterios se utilizan para evitar la formación de ciclos.

El cálculo de los 8-vecinos de un nodo en un árbol q-tree no es una tarea trivial debido a que los vecinos se pueden encontrar en ramas diferentes y a distintos niveles de profundidad en el árbol (ver figura 5.10).

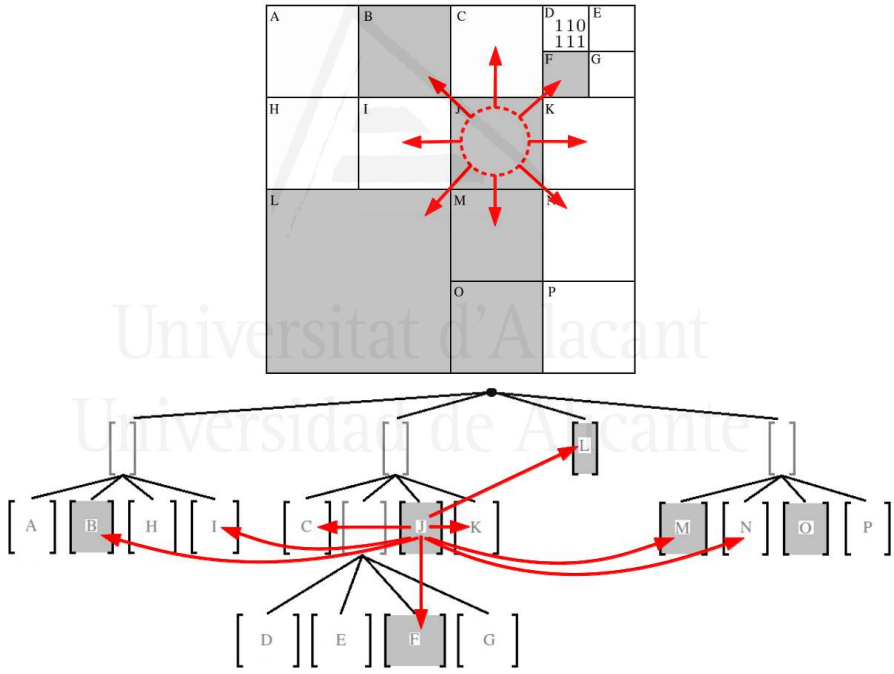


Figura 5.10: *Ejemplo de q-tree y de los 8-vecinos calculados a partir del nodo J.*

Debido a esto es necesario utilizar un sistema que nos permita calcular dicha

vecindad. En el algoritmo propuesto se ha extendido los trabajos para el cálculo de vecindad en q-trees de [52, 55], los cuales únicamente calculan los 4-vecinos de un nodo, o realizan dos pasos para calcular los 8-vecinos.

Nuestra solución calcula en una sola etapa y con el mismo nivel de complejidad los 8-vecinos de un nodo y además con cualquier nivel de profundidad en el árbol. A continuación se describe el proceso que realiza este cálculo.

Cálculo de la vecindad

Se define un array de localización para codificar la posición de las particiones en la imagen y su correspondiente recorrido en el q-tree. Cada dimensión de este array representa una coordenada (vertical u horizontal) que se codifica utilizando 1 bit: las direcciones positivas (arriba y derecha) se codifican con 1 y las negativas (abajo e izquierda) con 0 (ver figura 5.11).

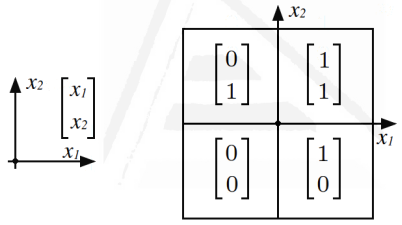


Figura 5.11: *Ejemplos del array de condificación binaria.*

El algoritmo propuesto calcula la vecindad para un nodo dado a partir de su array de localización (*M*) y la dirección en la cual se quiere calcular su vecino. Dependiendo de esta dirección el algoritmo distingue entre dos casos: los vecinos esquina (dos regiones que únicamente comparten un punto, por ejemplo en la figura 5.10 serían *A* e *I*), y los vecinos que comparten más de un punto (llamados "*face neighbours*", *A* y *B* en la figura 5.10).

Para el cálculo de un vecino, el algoritmo recorre el array de localización (*M*) de derecha a izquierda en la dimensión indicada (o dimensiones en el caso de los vecinos esquina), y va negando el valor de cada bit hasta que el resultado obtenido sea igual a la dirección indicada (1: positiva, 0: negativa).

Por ejemplo, para calcular el vecino derecho (dirección positiva: 1) de *I* (figura 5.10), el algoritmo niega el valor de cada bit de la columna superior (dirección

horizontal) empezando por la derecha y hacia la izquierda, hasta que alcance la condición de finalización: el resultado sea igual a 1:

$$I = \begin{bmatrix} 0 & \overline{\mathbf{1}} \\ 1 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} \overline{\mathbf{0}} & 0 \\ 1 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} = J$$

Cuando el algoritmo termina la exploración de una dimensión del array sin alcanzar la condición de finalización indica que el nodo no tiene vecino en esa dirección (por ejemplo si intentamos calcular el vecino izquierdo del nodo *A*).

El proceso para el cálculo de los vecinos esquina es el mismo, pero realizando la misma operación en ambas direcciones hasta alcanzar la condición de finalización en ambas. En el siguiente ejemplo se puede ver el cálculo del vecino superior izquierda del nodo *N*:

$$N = \begin{bmatrix} 1 & \overline{\mathbf{1}} \\ 0 & \mathbf{1} \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 \\ \overline{\mathbf{0}} & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} = J$$

Una vez calculado el vecino el algoritmo comprueba el tipo de nodo obtenido, distinguiendo tres posibles casos:

1. *Nodo hoja*: el algoritmo termina devolviendo dicho nodo.
2. *Nodo interno*: en este caso se tienen que calcular los vecinos de mayor nivel hasta obtener un nodo hoja.
3. *Ningún nodo*: esto es debido a que el vecino es de menor nivel que el nodo actual, por lo que habrá que calcular los nodos padre hasta obtener un nodo hoja.

En el artículo se puede encontrar una descripción más detallada de cada uno de estos procesos y los algoritmos completos para su cálculo (ver página 274).

5.7.2 Grafos mediante una rejilla estructural

El segundo algoritmo trata la imagen como una rejilla estructural, de forma que tras un postproceso obtiene un grafo acíclico dirigido. Los grafos obtenidos mantienen las mismas ventajas que el método anterior: permiten la reconstrucción de la muestra y la extracción mediante una resolución parametrizable.

Empezando en la esquina superior derecha el algoritmo divide la imagen en regiones cuadradas de un tamaño dado *k*. Las regiones cuyo valor supere un determinado umbral son consideradas los nodos del grafo. Los arcos se definen desde un nodo dado hasta sus nodos adyacentes en las direcciones: este, sur y sureste. En la figura 5.12 se puede ver un ejemplo.

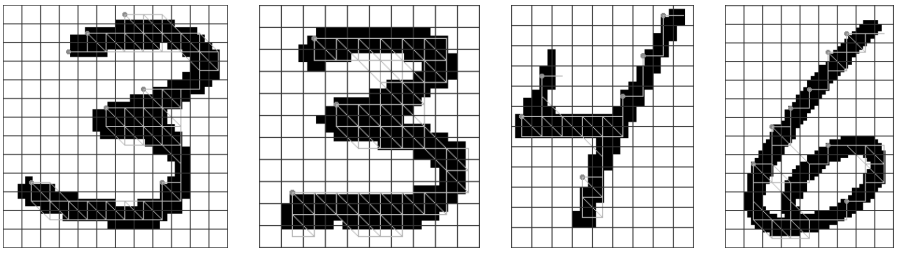


Figura 5.12: *Ejemplo del cálculo de un grafo mediante una rejilla estructural. Los nodos raíz se han marcado en gris.*

5.7.3 Grafos a partir de la extracción del esqueleto

El tercer método propuesto es una extensión natural a grafos del trabajo realizado en [56, 54] para árboles. El algoritmo primero procesa la imagen para obtener el esqueleto que la representa [51]. A continuación genera el grafo recorriendo dicho esqueleto, teniendo en cuenta que cuando dos caminos se crucen significará una conjunción en el grafo (el algoritmo completo se puede consultar en la página 278). En la figura 5.13 se puede ver un ejemplo de los resultados obtenidos.

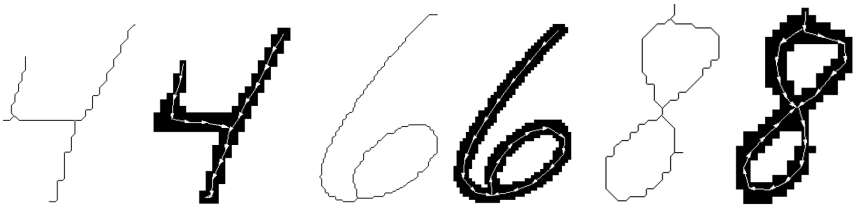


Figura 5.13: *Resultados de la extracción de grafos a partir de su esqueleto.*

5.8 Conclusiones y resultados

En este trabajo se extiende la definición de lenguaje *k*-testable a grafos acíclicos dirigidos. Este es el primer resultado que utiliza las características de los grafos en vez de la estructura de las reglas gramaticales. Además se propone un modelo de autómata para grafos que nos permite la introducción de un algoritmo de inferencia que identifica la clase de los lenguajes *k*-testables de grafos acíclicos dirigidos. Este algoritmo se ejecuta en tiempo polinómico e identifica este tipo de lenguajes a partir de muestras positivas.

La definición propuesta para los lenguajes *k*-testables permite el uso de grafos dirigidos que contengan ciclos. Sin embargo, el autómata y el algoritmo de inferencia propuesto no lo permiten, y se enfocan únicamente para la clase de grafos acíclicos dirigidos. El principal problema para extender estos resultados a grafos que contenga ciclos es que se necesita establecer un orden de procesamiento y un criterio de aceptación, y en el caso general, al ser cíclicos, es posible que los grafos no tuvieran nodos de inicio o final.

Debido a la no disposición de bases de datos reales para el proceso de experimentación, se realizó un primer estudio utilizando una base de datos de grafos generados de forma sintética. Posteriormente se propusieron nuevos algoritmos para la extracción de grafos a partir de imágenes.

Los resultados obtenidos de la extracción de grafos se validaron mediante métodos de clasificación simples para identificar la correcta separación de las clases (para más información sobre la experimentación ver página 279). En este sentido los resultados obtenidos demostraron la validez de los métodos, siendo un punto de partida para trabajos futuros y para su utilización en los modelos *k*-testables.

Estos avances nos permiten identificar la pertenencia de grafos a un lenguaje *k*-testable o la posterior generación o inferencia de grafos que pertenezcan al lenguaje. También abren nuevas posibilidades de operación y aplicación, extendiendo el uso de los grafos a otro tipo de aplicaciones. Muchos campos pueden obtener ventaja de los resultados obtenidos, como la bioinformática, la compresión de datos, o especialmente las tareas de reconocimiento de patrones.

Como futuras líneas de investigación se pretende obtener la extensión probabilística por interpolación y *backing-off* de los lenguajes *k*-testables propuestos, así como el estudio de un algoritmo generador a partir de un lenguaje *k*-testable dado. Se estudiará también su uso para el aprendizaje de las estructuras pro-

puestas para la reconstrucción y modelado del espacio. De forma que a partir de grafos que representen la forma de los objetos de una escena se realice un proceso de clasificación en posibles primitivas del lenguaje de modelado.

Bibliografía

[1] Amador González, J. A. *Adquisición y procesamiento de imágenes estereoscópicas y modelado de mundos 3D para su implementación en exploración de ambientes.* PhD Thesis. Universidad de las Américas-Puebla. 2004.

[2] J. M. López-Vallés, A. Fernández-Caballero y M. A. Fernández. *Conceptos y técnicas de estereovisión por computador.* Inteligencia artificial, 9(27):págs. 35–62. 2006.

[3] S. Gutiérrez and J. L. Marroquín. *Robust approach for disparity estimation in stereo vision.* Image and Vision Computing, 22(3):págs. 183–195. 2004.

[4] M. Sonka, V. Hlavac and R. Boyle. *Image processing, analysis and machine vision.* Chapman & Hall, Londres, Reino Unido. 1993.

[5] J. G. Jiménez. *Visión por computador.* Paraninfo, Madrid, España. 1999.

[6] Satorre Cuerda, Rosana. *Visión estéreo, multirresolución y modelo integrado.* Tesis doctoral. Universidad de Alicante. 2002

[7] P. Arques, F. Aznar, M. Pujol, R. Rizo. *Segmentación de Imágenes en Tiempo Real Mediante Umbralización Adaptativa.* Conferencia de la Asociación Española para la Inteligencia Artificial. 2005.

[8] P. Arques, R. Molina, M. Pujol, R. Rizo. *Distance Histogram to Centroid as a Unique Feature to Recognize Objects.* First International Conference on Computer Vision Theory and Applications. Portugal, 2006.

[9] Michael Bleyer. PhD Thesis. *Segmentation-based Stereo and Motion with Occlusions.* Institute for Software Technology and Interactive Systems - Interactive Media Systems Group, 2006.

[10] Y. Deng, Q. Yang, X. Lin, and X. Tang. *A symmetric patch-based correspondence model for occlusion handling.* In International Conference on Computer Vision, pages 542-567, 2005.

[11] Oliver Faugeras. *Three-dimensional computer vision: a geometric viewpoint.* The MIT Press. Cambridge, Massachussets, 1993.

[12] L. Hong and G. Chen. *Segment-based stereo matching using graph cuts.* In conference on Computer Vision and Pattern Recognition, volume 1, pages 74-81, 2004.

[13] A. López. *Visión estereoscópica basada en regiones: estado del arte y perspectivas de futuro.* Conferencia de la Asociación Española para la Inteligencia Artificial. 2001.

[14] D. Scharsteing and R. Szeliski. *A taxonomy and evaluation of dense two-frame stereo correspondence algorithms.* International Journal of Computer Vision, 47(1/2/3):7-42, 2002.

[15] H. Tao and H. Sawhney. *Global matching criterion and color segmentation based stereo.* In Workshop on the Application of Computer Vision, pages 246-253, 2000.

[16] Z. Zhang. *A flexible new technique for camera calibration.* IEEE Trans. on Pattern Analysis and Machine Intelligence, 22(11):1330–1334, 2000.

[17] Hans P. Moravec. *Robot spatial perception by stereoscopic vision and 3D evidence grids.* The Robotics Institute Carnegie Mellon University. Pittsburgh, PA (1996)

[18] Stephen Se, D. Lowe, J. Little. *Vision-based mobile robot localization and mapping using scale-invariant features.* ICRA (2001)

[19] C. Martin and S. Thrun. *Real-time acquisition of compact volumetric maps with mobile robots.* ICRA (2002)

[20] Gonzalo Pajares Martinsanz, Jess M. de la Cruz Garca. *Visión por computador: imágenes digitales y aplicaciones.* Ed. Ra-Ma, D.L. Madrid (2001)

[21] G. Vogiatzis, P.H.S. Torr and R. Cipolla. *Multi-view stereo via Volumetric Graph-cuts.* CVPR (2005) 391–398.

[22] L. Zhang, Steven M. Seitz. *Parameter estimation for MRF stereo.* In IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), San Diego, CA, June (2005).

[23] S. Sinha, M. Pollefeys. *Multi-view Reconstruction using Photo-consistency and Exact Silhouette Constraints: A Maximum-Flow Formulation.* ICCV (2005).

[24] Alberto Broggi. *Robust Real-Time Lane and Road Detection in Critical Shadow Conditions.* In Proceedings IEEE International Symposium on Computer Vision, Coral Gables, Florida. IEEE Computer Society (1995)

[25] Compañ, P.; Satorre, R.; Rizo, R. *Disparity estimation in stereoscopic vision by simulated annealing.* Artificial Intelligence research and development. IOS Press. (2003) 160–167

[26] E. Trucco and A. Verri. *Introductory techniques for 3-D Computer Vision.* Prentice Hall (1998)

[27] I. Cox, S. Ignoran, and S. Rao. *A maximum lilelihood stereo algorithm.* Computer Vision and Image Understanding, 63 (1996)

[28] J. Coughlan and A. L. Yuille. *Manhattan World: Orientation and Outlier Detection by Bayesian Inference.* Neural Computation. Vol. 15, No. 5 (2003) 1063–88

[29] Davis, Martin; Sigal, Ron and Weyuker, Elaine J. *Computability, Complexity, and Languages.* Fundamentals of Theoretical Computer Science, 2nd ed. San Diego: Elsevier Science, 1994.

[30] Singhal, A.; Brown, C. *Dynamic bayes net approach to multimodal sensor fusion.* SPIE, 1997.

[31] A. Clark, R. Eyraud, and A. Habrard. *A polynomial algorithm for the inference of context free languages.* LNAI, 5278:29–42, 2008. Proceedings of ICGI-08.

[32] C. de la Higuera. *Grammatical inference. Learning automata and grammars.* Cambridge University Press, 2010.

[33] P. García. *Learning k-testable tree sets from positive data.* Technical Report DSIC/II/46/1993, Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia, 1993. Available on: `http://www.dsic.upv.es/users/tlcc/tlcc.html`.

[34] P. García and E. Vidal. *Inference of k-testable languages in the strict sense and application to syntactic pattern recognition.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 12:920–925, 1990.

[35] E. Jeltsch and H. J. Kreowski. *Grammatical inference based on hyperedge replacement.* LNCS, 532:461–474, 1991. 4th International workshop on graph grammars and their application to computer science.

[36] I. Jonyer, L. B. Holder, and D. J. Cook. *Concept formation using graph grammars.* In Proceedings of the KDD Workshop on Multi-Relational Data Mining, pages 71–792, 2002.

[37] R. Kosala, H. Blockeel, M. Bruynooghe, and J. Van den Bussche. *Information extraction from documents using k-testable tree automaton inference.* Data & Knowledge Engineering, 58:129–158, 2006.

[38] J. P. Kukluk, L. B. Holder, and D. J. Cook. *Inference of node replacement recursive graph grammars.* In Proceedings of the Sixth SIAM International Conference on Data Mining, 2006.

[39] D. López, J. M. Sempere, and P. García. *Inference of reversible tree languages*. IEEE Transactions on System Man. and Cybernetics, Part B: Cybernetics, 34(4):1658–1665, 2004.

[40] R. McNaughton. *Algebraic decision procedures for local testability.* Math. Sysr. Theory, 8(1):60–76, 1974.

[41] K. Nakamura. *Incremental learning of context free grammars by bridging rule generation and search for semi-optimum rule sets.* LNAI, 4201:72–83, 2006. Proceedings of ICGI-06.

[42] P. Peris, D. López, and M. Campos. *IgTM: An algorithm to predict transmembrane domains and topology in proteins.* BMC - Bioinformatics, 9:367, 2008.

[43] A. Potthoff, S. Seibert, and W. Thomas. *Nondeterminism versus determinism on finite automata over directed acyclic graphs.* Bull. Belg. Math. Soc., 1:285–298, 1994.

[44] J. R. Rico-Juan, J. Calera-Rubio, and R. C. Carrasco. *Smoothing and compression with stochastic k-testable tree languages.* Pattern Recognition, 38: 1420–1430, 2005.

[45] D. Ron, Y. Singer, and N. Tishby. *On the learnability and usage of acyclic probabilistic finite automata.* Journal of Computer and System Sciences, 56: 133–152, 1998.

[46] G. Rozemberg, editor. *Handbook of graph grammars and computing by graph transformation*, volume 1. Word Scientific, 1997.

[47] Y. Sakakibara. *Learning Context-Free Grammars from Structural Data in Polynomial Time.* Theoretical Computer Science, 76:223–242, 1990.

[48] Y. Sakakibara. *Efficient Learning of Context-Free Grammars from Positive Structural Examples.* Information and Computation, 97:23–60, 1992.

[49] Y. Sakakibara. *Grammatical inference in bioinformatics.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 27(7):1051–1062, 2005.

[50] K. Zhang. *A constrained edit distance between unordered labelled trees.* Algorithmica, 15:205–222, 1996.

[51] J. M. Cychosz. *Thinning algorithm from the article: Efficient binary image thinning using neighbourhood maps.* Graphics Gems IV. Academic Press, p. 465–473, 1994.

[52] M. Goodchild. *Quadtree algorithms and spatial indexes.* Technical Issues in GIS, NCGIA, Core Curriculum, (37):5–6, 1990.

[53] J. Liu, M. Li, Q. Liu, H. Lu, and S. Ma. *Image annotation via graph learning.* Pattern Recognition, 42:218–228, 2009.

[54] D. López and I. Piñaga. *Syntactic pattern recognition by error correcting analysis on tree automata.* Advances in Pattern Recognition, LNCS 1876, p. 133–142, 2000.

[55] J. Poveda and M. Gould. *Multidimensional binary indexing for neighbourhood calculations in spatial partition trees.* Comput. Geosci., 31(1):87–97, 2005.

[56] J. R. Rico-Juan, L. Micó. *Comparison of AESA and LAESA search algorithms using string and tree edit distances.* Pattern Recognition Letters, 24:1427–1436, 2003.

[57] H. Shin, K. Tsuda, and B. Schölkopf. *Protein functional class prediction with a combined graph.* Expert Systems with Applications, 36:3284–3292, 2009.

Parte II

Publicaciones

En esta segunda parte se incluye el compendio de publicaciones siguiendo un orden cronológico pero agrupado por temáticas para facilitar su lectura y comprensión.

Publicación 1

Detección de objetos y estimación de su profundidad mediante un algoritmo de estéreo basado en segmentación

Antonio Javier Gallego, Patricia Compañ, Pilar Arques,
Carlos Villagrá, Rafael Molina

Grupo de Informática Industrial e Inteligencia Artificial
Universidad de Alicante, Ap.99, E-03080, Alicante, Spain

{*ajgallego, company, arques, villagra, rmolina*}*@dccia.ua.es*

VIII Workshop en Agentes Físicos WAF (2007) pp. 65-72.
ISBN: 978-84-9732-597-4

Detección de objetos y estimación de su profundidad mediante un algoritmo de estéreo basado en segmentación

Antonio Javier Gallego, Patricia Compañ, Pilar Arques,
Carlos Villagrá, Rafael Molina

Grupo de Informática Industrial e Inteligencia Artificial
Universidad de Alicante, Ap.99, E-03080, Alicante, Spain
{*ajgallego, company, arques, villagra, rmolina*}*@dccia.ua.es*

Resumen

Se presenta un sistema de visión estereoscópica basada en segmentación que aprovecha la información obtenida y las ventajas de este tipo de sistemas para la detección de objetos en la escena y la estimación de su profundidad. El proceso de segmentación elegido, umbralización adaptativa, permite obtener buenos resultados con un tiempo de cómputo muy bajo. Cada región extraída debe ser caracterizada mediante un vector de propiedades que permita la identificación eficiente y unívoca del objeto, proponemos una serie de características basadas en la posición, el tamaño, el color y la forma. El proceso de correspondencia utiliza este vector para emparejar las regiones en base a la similitud que presentan. Esta medida se obtiene mediante la ponderación de las características que forman el vector. Posteriormente se realiza el cálculo de la disparidad y de la profundidad, incorporando un factor de corrección empírico. Además se ha añadido un postproceso de extracción de capas que consigue eliminar *outliers* y mejorar las profundidades obtenidas. Por último, en base a la segmentación inicial y a las profundidades calculadas, se detectan los objetos buscados. El modelo propuesto presenta ventajas de tiempo de cómputo y de precisión en la estimación de la profundidad y en la detección de objetos.

1.1 Introduction

La detección de objetos y el cálculo de la profundidad a la que se encuentran son dos procesos de gran utilidad en multitud de ámbitos, como la robótica, la supervisión y el control de calidad, la ayuda a personas con alguna discapacidad o la conducción automática, por enumerar sólo unos pocos. La visión estereoscópica puede ser una solución, aunque hoy en día es todavía un campo abierto de investigación. Se han conseguido resultados adecuados en entornos simples, sin embargo el cálculo de la profundidad en determinados casos es un problema muy complejo, especialmente cuando hay poca textura, existen oclusiones, etc. Por otro lado, los procesos asociados a la visión estereoscópica (emparejamiento, cálculo de disparidades, etc.) son procedimientos intrínsecamente complejos, con costes temporales muy altos.

Una posible aproximación al problema es la segmentación previa de las imágenes estereoscópicas. Estas técnicas dividen una o, a veces, las dos imágenes estéreo en regiones no solapadas de color homogéneo. En lugar de computar la disparidad para cada píxel individual, estas técnicas asignan un único valor de disparidad a cada uno de las regiones obtenidas. A priori este planteamiento presenta dos ventajas: por un lado, el hecho de utilizar regiones en vez de píxeles hace el proceso más robusto frente a la presencia o no de texturas; por otro lado, la cantidad de emparejamientos a realizar entre las regiones es mucho menor que el que se tendría que realizar para el total de píxeles.

Sin embargo, la reducción del número de emparejamientos tiene una contrapartida: debemos realizar un proceso previo de segmentación, cuestión no trivial, y debemos elegir un conjunto de propiedades que caractericen convenientemente a las regiones para poder realizar la correspondencia.

Hay una considerable cantidad de literatura sobre el problema de la correspondencia en estéreo. En [11] se puede encontrar una extensa revisión de los algoritmos actuales de estéreo que producen un mapa de disparidad denso (con información para todos los puntos de la imagen). Centrándonos en técnicas que emplean segmentación, en [12] los autores proponen un algoritmo estéreo basado en regiones que utiliza deformación de la imagen para medir la calidad del mapa de disparidad. Otras técnicas, como en [9], combinan el emparejamiento basado en regiones con la optimización basada en grafos. En [4] se puede encontrar una amplia documentación sobre todos estos algoritmos, a la vez que se proponen dos métodos con unos buenos resultados para el cálculo del mapa de disparidad

utilizando segmentación. En [7] también se tienen en cuenta las ocultaciones que se producen al realizar el estéreo, para lo que se segmentan ambas imágenes con la intención de encontrar las partes de cada región que están parcialmente ocultas (para esto se ayuda de una función de coste). Aparte de los métodos comentados, en [10] se hace un interesante estudio sobre técnicas que resuelven el problema de la correspondencia mediante estéreo.

En este trabajo presentamos un esquema de visión estereoscópica basado en segmentación cuya finalidad es la detección de objetos y el cálculo de la profundidad de los mismos. El uso de la segmentación previa aporta varias ventajas frente a otros algoritmos. En primer lugar la detección de los objetos ya viene dada de antemano por dicha segmentación (en un sistema basado en correspondencia entre píxeles sería necesaria una segmentación posterior) y, además, los objetos pueden ser diferenciados a través del vector de características (aprovechado para el proceso de correspondencia del estéreo). Otra ventaja del método propuesto es su velocidad. La baja complejidad del algoritmo permite utilizarlo en tiempo real en multitud de aplicaciones, como por ejemplo la navegación, la detección de objetos y su profundidad o el guiado mediante *landmarks*.

1.2 Modelo propuesto

Como se ha comentado, la finalidad del sistema propuesto es la detección de objetos y la estimación de su profundidad. El punto de partida es un par de imágenes estereoscópicas en color, que han sido rectificadas epipolarmente [8]. Dichas imágenes rectificadas se obtienen directamente de la cámara estéreo sin ningún coste adicional para el algoritmo. El esquema de procesamiento puede dividirse en cinco pasos que se presentan resumidamente a continuación:

1. En primer lugar se segmentan las imágenes de entrada en regiones de color homogéneo.

2. A partir de estas regiones se obtiene un vector de características que nos permite realizar el emparejamiento.

3. A continuación se calcula la profundidad de cada una de las regiones utilizando la información de disparidad.

4. Se analizan las profundidades encontradas y se reajustan mediante la asignación a capas.

5. El último paso es la detección de objetos, para lo cual se utiliza la información obtenida en la segmentación.

Implícitamente este modelo aplica dos asunciones básicas: en primer lugar, los píxeles de un segmento con color homogéneo tendrán valores de disparidad que siguen un modelo de disparidad suavizado (restricción de continuidad). En segundo lugar, se asume que las discontinuidades en profundidad coinciden con las fronteras de dichas regiones. Todo esto se justifica en el hecho de que los píxeles pertenecientes a una región con un mismo color lo más probable es que provengan del mismo objeto de la escena, por lo que se espera que la disparidad varíe suavemente.

En las siguientes secciones se analiza en detalle cada uno de los pasos del algoritmo.

1.2.1 Segmentación

En un sistema automatizado de visión, la segmentación de imágenes es uno de los procesos más importantes, ya que permite extraer los objetos de la imagen para su posterior descripción y reconocimiento. Puede definirse como el proceso de dividir una imagen en un conjunto de regiones disjuntas cuyos píxeles comparten atributos similares, generalmente de intensidad, color o textura. En principio cualquier algoritmo que divida la imagen en regiones de un color homogéneo puede ser utilizado por el modelo propuesto. En la implementación actual se está trabajando con un algoritmo de segmentación que utiliza una nueva técnica denominada “umbralización adaptativa” y que gracias a su reducido tiempo de cómputo permite trabajar en tiempo real [1]. Lo más importante es que devuelva una serie de características descriptivas para cada región segmentada, que, como se verá en la siguiente sección, serán las que se utilizarán en el proceso de correspondencia.

1.2.2 Emparejamiento de regiones

Para realizar la correspondencia de regiones (establecer para cada región de la imagen izquierda cuál es su correspondiente en la imagen derecha) se ha utilizado un método de comparación basado en la similitud de características. En este trabajo proponemos la caracterización de las regiones en base a su posición, tamaño, color y forma. Para ello se utilizan las características que facilita el método

de segmentación, concretamente se utilizan como descriptores de los objetos el centroide (posición), el área (tamaño), el color (color) y el histograma de distancias al centroide (forma). Existen otras muchas posibilidades para describir un objeto, pero éstas se proponen por sus ventajas (en cuanto a su facilidad de obtención) y por ser lo suficientemente significativas para caracterizar una región [2]. A continuación se verá cada una de estas características en detalle.

Centroide: Es el punto que define el centro geométrico de una región y, por consiguiente, su posición en la imagen. Un método sencillo para hallarlo es calcular el promedio de las coordenadas de todos los puntos que forman parte del objeto. Para realizar la comparación (*DistC*) se calcula la distancia euclídea entre los centroides de las regiones a emparejar. En el proceso de emparejamiento hay que tener en cuenta que, dado que las imágenes están rectificadas, las líneas epipolares de búsqueda coinciden. Por lo que el valor de la componente y del centroide de una región prácticamente coincidirá con el de su región correspondiente en la otra imagen. De esta forma podemos restringir el área de búsqueda de dicha comparación a un ε dado.

Área: Indica el tamaño de la superficie comprendida dentro del perímetro de una región. El método más simple para estimar el área es contar el número de píxeles que representan dicha forma. Para comparar la similitud entre dos áreas (*DistA*) se utiliza el valor absoluto de su diferencia.

Color: Una medida que se utiliza habitualmente como métrica en el espacio RGB es la distancia euclídea y variaciones de la misma. En [5] se estudian dos medidas de similitud de colores: la distancia euclídea (ecuación 1) y la distancia angular (ecuación 2). La distancia euclídea presenta una pobre discriminación entre valores distintos que tienen una intensidad similar. Por otra parte, la distancia angular presenta un comportamiento inestable cuando los colores están cerca del origen de coordenadas. Debido a esto, para realizar la comparación (*DistRGB*) se ha optado por utilizar una distancia combinada sobre la media RGB de cada región obtenida en la segmentación, de modo que utiliza la distancia euclídea cuando la saturación es baja y la distancia angular cuando es alta [6]. Para calcular la saturación de un píxel se utiliza la ecuación 3.

$$D_{EUC}(p,q) = \| \overrightarrow{p} - \overrightarrow{q} \| \quad (1)$$

$$D_{ANG}(p,q) = \sqrt{1 - \left(\frac{\overrightarrow{p}^T \cdot \overrightarrow{q}}{\| \overrightarrow{p} \| \cdot \| \overrightarrow{q} \|} \right)^2} \quad (2)$$

$$S(p) = 1 - \frac{3 \cdot min(R_p, G_p, B_p)}{R_p + G_p + B_p} \quad (3)$$

Histograma de distancias al centroide: Es el histograma de las distancias existentes desde el centroide hasta cada uno de los puntos que forman el perímetro de la región. Esta métrica es muy importante para la posterior detección de objetos, pues es un rasgo que caracteriza unívocamente la forma de los objetos y además es invariante a traslación, rotación y escalado [2].

En el siguiente algoritmo muestra el procedimiento para calcular el histograma de distancias al centroide H para la región R y para N_{CAT} categorías, donde *Per* es su perímetro, $F_1,\ldots,F_N$ son sus puntos frontera, $Dist(F_i)$ es la distancia euclídea entre un punto frontera y el centroide, D_{MAX} es la distancia máxima al centroide, $Cat(Dist(F_i))$ es la categoría de la distancia calculada (ecuación 4) y $H[1]..H[N_{CAT}]$ son los valores obtenidos para cada categoría.

```
Para todo F_j de R hacer
   Calcular Dist(F_j)
   Calcular dc = Cat( Dist(F_j) )
   H[dc]++
Desde i = 1 hasta N_CAT hacer
   H[i] = H[i] / Per
```

En este algoritmo, una vez calculada la distancia euclídea de los puntos frontera al centroide, se asigna dicha distancia a la categoría correspondiente, teniendo en cuenta el valor de la distancia máxima de la frontera al centroide. Por último normalizamos el histograma dividiendo por el perímetro, de esta forma obtenemos la invarianza ante el escalado (importante para el reconocimiento de objetos).

Como medida de comparación se ha utilizado la raíz cuadrada de la suma de los cuadrados de las diferencias entre dichas series de histogramas. En la ecuación 5 se muestra esta medida.

$$Cat(Dist(F-i)) = Dist(F_i)\cdot\frac{N_{CAT}}{D_{MAX}} \quad (4)$$

$$DistH_{lr} = \sqrt{\sum_{i=1}^{N_{CAT}}(H_l[i]-H_r[i])^2} \quad (5)$$

Todas estas características definen el denominado *vector de características* de cada región, el cual permite realizar el proceso de emparejamiento y además, en una etapa posterior, ayudará al reconocimiento de los objetos.

Como medida de similitud entre dos regiones se ha tomado el inverso de la media ponderada de las distancias de los descriptores propuestos. En la ecuación

6 se puede ver esta medida, donde $w_0, \ldots, w_3$ son los pesos utilizados para ajustar el valor de cada componente en el resultado total de la ecuación. De esta forma se puede dar más peso a una determinada característica que de mejores resultados en el proceso de emparejamiento (como el color y la posición) y menos a otras (como la forma y el tamaño).

En el proceso de correspondencia se van comparando una a una las regiones que estén todavía sin emparejar y que además cumplan las restricciones propuestas (explicadas en el párrafo correspondiente de cada característica). Finalmente se emparejarán aquellas para las que se obtenga el menor valor de δ.

$$\delta = 1/(w_0 DistC + w_1 DistA + w_2 DistRGB + w_3 DistH) \quad (6)$$

1.2.3 Cálculo de la disparidad y profundidad

Por disparidad nos referimos al desplazamiento lateral que se produce en la representación de los objetos de la escena en cada una de las imágenes del par estéreo, esto es debido a la separación horizontal de los objetivos de la cámara. En el caso de cámaras paralelas y asumiendo que las imágenes se encuentran convenientemente rectificadas, se puede calcular la disparidad a partir de la diferencia absoluta de la posición en cada imagen de cada una de las regiones emparejadas (esta posición vendrá dada por la componente x de su centroide).

Una vez realizado el cálculo de la disparidad, y teniendo en cuenta la relación inversamente proporcional existente entre disparidad y profundidad, estamos en disposición de obtener la profundidad a la que se encuentran los objetos de la escena mediante la ecuación $z = f\frac{T}{d}$ [8], donde f es la distancia focal, T es la línea base (distancia entre los centros ópticos) y d es el valor de disparidad. Para la cámara estéreo utilizada (*Bumblebee HICOL-60*) estos valores son de f = 0,006m y T = 0,12m.

El problema es que la disparidad d está expresada en píxeles, mientras que el resto de unidades lo están en metros. Para unificar las unidades se ha empleado el factor de conversión λ = *ancho del CCD en metros / ancho de la imagen en píxeles*. Para calcular el ancho del CCD de la cámara necesitamos conocer sus dimensiones y su proporción ancho/alto, que para la cámara utilizada (ver apartado 1.3) son de $1/3$" y de $x/y = 4/3$ respectivamente (figura 1.1). De forma que substituyendo en la igualdad $x^2 + y^2 = (1/3")^2$ podemos despejar el valor de x y obtener así el ancho total en metros.

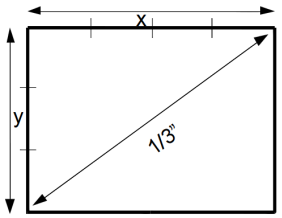


Figura 1.1: *Dimensiones del CCD de la cámara estéreo.*

Además se ha añadido el factor de corrección ρ para solucionar un pequeño error debido a la concavidad de la lente de la cámara. Este factor ρ ha sido obtenido empíricamente (ver sección de experimentación). En la ecuación 7 se puede ver la ecuación final que se ha utilizado para el cálculo de la profundidad.

$$z = f\frac{T}{d\lambda}\rho \quad (7)$$

1.2.4 Extracción de capas

Lo que se pretende con este proceso es agrupar en una capa todas aquellas regiones que puedan ser aproximadas por una misma profundidad. De esta forma se eliminan los posibles *outliers* (regiones de un tamaño muy pequeño que además no se pueden asignar a ninguna capa) y se minimizan los posibles errores en el cálculo de la profundidad (debidos a pequeñas diferencias en los valores de los centroides). Suponiendo que las regiones están ordenadas según su profundidad, los pasos que sigue el algoritmo son los siguientes (ver también figura 1.2):

1. Se toma la primera región y se crea una capa a esa profundidad.

2. Se comprueba si la distancia desde la capa hasta la siguiente región es menor que un umbral u dado. En caso afirmativo se asigna la región a la capa y se recalcula la profundidad como la media entre la profundidad actual de la capa y la de la región. En caso negativo se coge la siguiente región, se crea una nueva capa y se vuelve a repetir este paso (hasta que no queden más regiones).

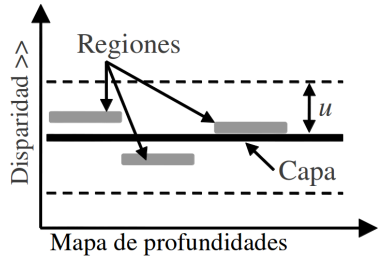


Figura 1.2: *Esquema de la extracción de capas.*

1.2.5 Detección de objetos

La detección de objetos se basa en una modificación del modelo propuesto en [3], el cual trabaja en el espacio RGB y realiza segmentación en color. Se ha elegido este modelo para realizar una primera aproximación a la detección y reconocimiento de objetos, en concreto de señales de tráfico. En nuestro caso se utilizan las regiones obtenidas en la segmentación, a partir de las cuales se buscarán las regiones con unos colores y formas determinadas (empleando la media de color y el histograma de distancias al centroide). La comparación de las regiones con el objeto que se desea encontrar se realiza fácilmente a partir de las distancias definidas en la sección anterior.

La detección se basa en que las señales de tráfico son de colores muy llamativos (rojo o azul), los cuales son fácilmente diferenciables. De esta forma se pueden desechar el resto de píxeles de la imagen y obtener una localización rápida y sencilla de los objetos que deseamos reconocer. Pero tenemos el inconveniente de que también se localizan otras regiones que cumplen estas condiciones, incluyendo ruido. Para minimizar este problema se desecharán las zonas que no tengan una mínima agrupación de las tonalidades seleccionadas.

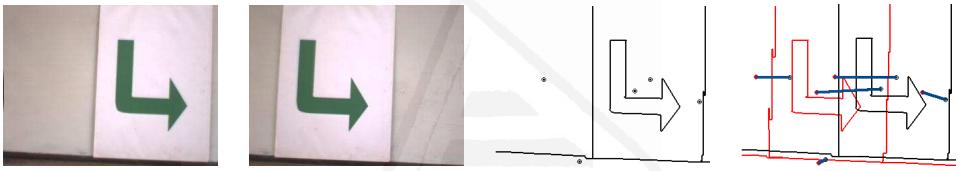
Para la detección se han empleado los umbrales $R_U = 255 - 70$, $G_U = 255/2$, $B_U = 255/2$ y la condición definida en la ecuación 8. Estos valores se han obtenido empíricamente mediante un estudio de los colores de las señales en diferentes momentos del día.

$$\left(\frac{G_p}{R_p - 70} < \frac{G_U}{R_U}\right) \wedge \left(\frac{B_p}{R_p - 70} < \frac{B_U}{R_U}\right) \quad (8)$$

1.3 Resultados de la experimentación

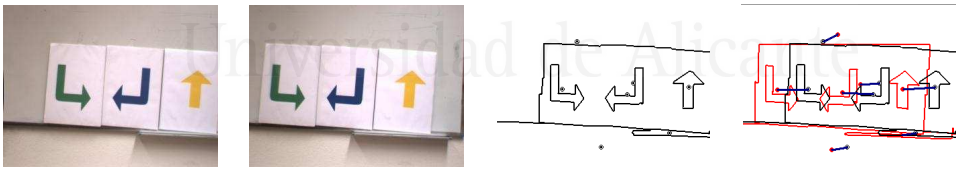
En este apartado se muestran los resultados obtenidos del proceso de experimentación sobre el algoritmo de segmentación, el emparejamiento, el cálculo del mapa de disparidad y el proceso de reconocimiento.

En las figuras 1.3, 1.4 y 1.5 se pueden ver los resultados de cómo se realiza la segmentación de un par estéreo y del emparejamiento en base a sus características. En cada uno de estos ejemplos se muestra el par estéreo utilizado (a,b), la segmentación de la imagen de referencia junto con el centroide de cada región (c) y por último el resultado del emparejamiento (d), donde además se puede ver como se unen las regiones. En la figura 1.5, debido a errores en la segmentación (para la imagen derecha se obtienen más regiones que para la izquierda), se puede ver en la parte inferior como hay alguna regiones para las que no se ha podido realizar la correspondencia.



(a) (b) (c) (d)

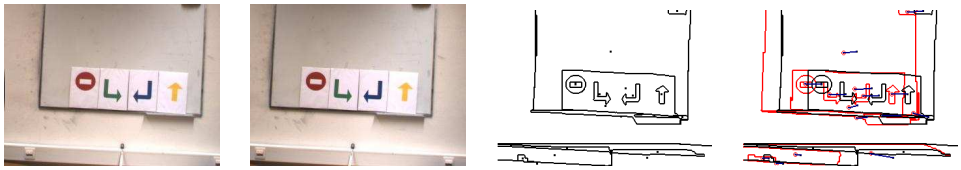
Figura 1.3: *(a,b) Par estéreo de una señal a 0.5m. (c) Segmentación. (d) Emparejamiento.*



(a) (b) (c) (d)

Figura 1.4: *(a,b) Par estéreo a 1.0m. (c) Segmentación. (d) Resultado del emparejamiento.*

En las figuras 1.6 y 1.7 se muestran dos ejemplos de los resultados obtenidos para la segmentación y la detección de objetos. En las imágenes se observan otras tonalidades azules que en algún caso pueden llegar a confundirse con las



(a) (b) (c) (d)

Figura 1.5: *(a,b) Par estéreo a 2.0m. (c) Segmentación. (d) Resultado del emparejamiento.*

(a) (b) (c) (d)

Figura 1.6: *(a) Imagen de referencia. (b) Segmentación. (c) Resultado de la detección. (d) Mapa de disparidad.*

tonalidades de una señal, pero que son eliminadas por las restricciones impuestas al algoritmo de detección (color e histograma de distancias, ver sección correspondiente).

Además, en la figura 1.6 se ha incluido la segmentación y el resultado del cálculo del mapa de disparidad. Se puede ver como se han agrupado todas las regiones pequeñas que aparecen en la segmentación (gracias al proceso de extracción de capas) y como de esta forma se han obtenido valores de profundidad unificados para las capas.

(a) (b)

Figura 1.7: *(a) Imagen de referencia. (b) Resultado del proceso de detección de objetos.*

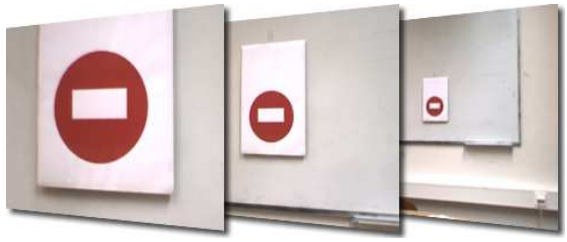


Figura 1.8: *Secuencia de imágenes a diferentes distancias.*

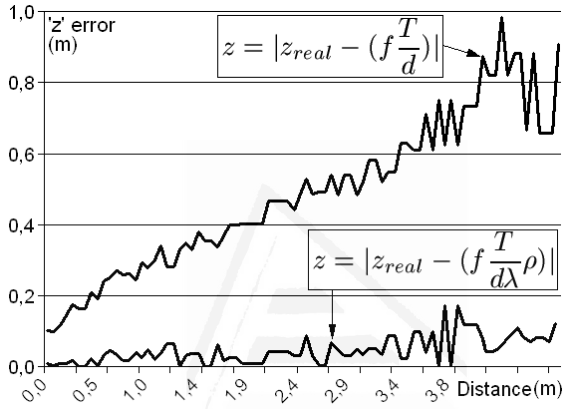


Figura 1.9: *Comparación del error producido en las ecuaciones del cálculo de la profundidad.*

Para obtener el error cometido al calcular la profundidad se tomó una secuencia de imágenes del mismo objeto (la señal de la figura 1.8) a diferentes distancias. De esta forma se pudo calcular el error cometido por la ecuación original frente a la distancia real a la que se encontraba la señal. Se llegó a la conclusión de que este error lineal se debía a la concavidad de la lente de la cámara. Para solucionarlo se calculó la media del error de todas las distancias calculadas, y se obtuvo así el factor ρ (ver ecuación 7) que permite corregir dicho error. En la figura 1.9 se muestra una gráfica con los resultados obtenidos, en la que se compara la distancia real a la que está la señal con la profundidad calculada mediante la ecuación original y la ecuación 7.

En lo referente a la complejidad temporal, en realidad el algoritmo no realiza tantas iteraciones como las que se muestran en el esquema inicial del apartado

1.2. En primer lugar se realiza la segmentación para la que se tienen que recorrer los m*n píxeles de la imagen (complejidad lineal). A continuación se calcula el emparejamiento, este proceso sólo se realiza para el conjunto de las regiones obtenidas, además a la vez se calcula la profundidad de la región y se compara con el objeto a detectar. Por último se vuelve a recorrer el conjunto de las regiones para realizar el reajuste de las profundidades. En resumen, sólo se recorre una vez toda la imagen y dos veces el conjunto de regiones obtenidas. Como se ve la complejidad del algoritmo es lineal, lo que ha permitido obtener unos resultados temporales muy buenos (incluso para procesamiento en tiempo real).

En [1] se pueden ver los resultados temporales del algoritmo de segmentación utilizado. El proceso completo para una imagen de 320x240 píxeles tarda menos de 0,1 seg. Es importante tener en cuenta que todos los experimentos se han realizado en un Pentium IV a 3,20GHz con 2GB de RAM y una tarjeta gráfica de 512MB, y que el tamaño utilizado para todas las imágenes es de 320x240 píxeles.

1.3.1 Conclusiones

Se ha presentado un modelo para realizar la detección de objetos y la estimación de su profundidad a partir de información estéreo y técnicas de segmentación.

Para el cálculo del mapa de disparidad se ha utilizado una comparación basada en características, la cual suele ser más rápida que otros métodos (como los de correlación por ventana o minimización de energía) ya que manipulan muchos menos datos (sólo se maneja el conjunto de las regiones segmentadas, en otros métodos el proceso de correspondencia se realiza píxel a píxel, por lo que se tiene que recorrer toda la imagen). En contrapartida proporcionan un mapa de disparidad poco denso, pero para algunas aplicaciones pueden ser más adecuados. Como la finalidad del proyecto es la detección de un marcador y de su profundidad (y en futuras versiones la navegación visual) se concluye que es el método más adecuado por las siguientes razones: se puede aprovechar la información de las características para el proceso de detección, la información de profundidad es suficiente para nuestro propósito y además los resultados temporales son más rápidos que con otros métodos.

Al utilizar segmentación nos hemos encontrado con algunos problemas: el más severo es que la segmentación obtenida no siempre tiene que ser la correcta. Por lo que la precisión de estos métodos depende directamente del algoritmo de segmentación. Además, es probable obtener segmentaciones diferentes para

cada imagen, debido principalmente a objetos que aparecen o se ocultan por los lados de la imagen, diferentes iluminaciones, etc.

Para intentar minimizar estos problemas se concluye que en general es preferible realizar una sobresegmentación de la imagen, pues tiene muchas ventajas: el algoritmo es más sencillo y la complejidad temporal mucho menor, además se reduce el número de errores (al intentar agrupar regiones más grandes se suelen obtener resultados diferentes entre ambas imágenes), y otra ventaja muy importante es que ante un error de emparejamiento el error es mucho menor (por ejemplo, para una imagen de 320x240 (76800 píxeles en total), un error en el emparejamiento de un región de 100 píxeles sólo supondría un error del 0.1%. mientras que un error en una región de 10000 píxeles supondría un error del 13%).

Otro problema son las ocultaciones, pues resolverlas únicamente a nivel de segmentos es muy complicado. De hecho, este problema es doble: primero está el problema del emparejamiento incorrecto de puntos medio ocultos a puntos mutuamente visibles y, segundo, incluso si un punto puede ser identificado como parcialmente oculto, ¿qué profundidad se le debería de asignar?

En el método propuesto no se ha realizado ningún tratamiento específico de las ocultaciones, por lo que como trabajo futuro se pretende estudiar posibles soluciones a este problema. También se planea analizar si mediante el uso de otro tipo de métricas RGB o HLS se obtienen mejores resultados en el emparejamiento o en la detección de objetos. Por último se pretende experimentar con otro tipo de características en el emparejamiento y reconocimiento de objetos.

Agradecimientos

Este trabajo se ha realizado gracias a la ayuda del proyecto “Utilización de la información de color para reconocer un marcador y la distancia a la que se encuentra mediante visión estéreo y técnicas de segmentación (GV06/158)” financiado por la Generalitat Valenciana.

Bibliografía

[1] P. Arques, F. Aznar, M. Pujol, R. Rizo. Segmentación de Imágenes en Tiempo Real Mediante Umbralización Adaptativa. Conferencia de la Asociación Española para la Inteligencia Artificial. 2005.

[2] P. Arques, R. Molina, M. Pujol, R. Rizo. Distance Histogram to Centroid as a Unique Feature to Recognize Objects. First International Conference on Computer Vision Theory and Applications. Portugal, 2006.

[3] Mohamed Bénallal and Jean Meunier. Real-time color segmentation of road signs. Proceedings of The IEEE Canadian Conference of Electrical and Computer Engineering, 2003.

[4] Michael Bleyer. PhD Thesis. "Segmentation-based Stereo and Motion with Occlusions". Institute for Software Technology and Interactive Systems - Interactive Media Systems Group, 2006.

[5] S. Bogumil. Color image edge detection and segmentation: a comparison of the vector angle and the euclidean distance color similarity measures. PhD thesis, Waterloo, 1999.

[6] P. Compañ. Estimación de la disparidad en visión estereoscópica. Tratamiento del color. PhD thesis, Alicante, 2004.

[7] Y. Deng, Q. Yang, X. Lin, and X. Tang. A symmetric patch-based correspondence model for occlusion handling. In International Conference on Computer Vision, pages 542-567, 2005.

[8] Oliver Faugeras. Three-dimensional computer vision: a geometric viewpoint. The MIT Press. Cambridge, Massachussets, 1993.

[9] L. Hong and G. Chen. Segment-based stereo matching using graph cuts. In conference on Computer Vision and Pattern Recognition, volume 1, pages 74-81, 2004.

[10] A. López. Visión estereoscópica basada en regiones: estado del arte y perspectivas de futuro. Conferencia de la Asociación Española para la Inteligencia Artificial. 2001.

[11] D. Scharsteing and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. International Journal of Computer Vision, 47(1/2/3):7-42, 2002.

[12] H. Tao and H. Sawhney. Global matching criterion and color segmentation based stereo. In Workshop on the Application of Computer Vision, pages 246-253, 2000.

Publicación 2

Improving Edge Detection In Highly Noised Sheet-Metal Images

Antonio-Javier Gallego-Sánchez, Jorge Calera-Rubio

Departamento de Lenguajes y Sistemas Informáticos, University of Alicante, Spain

{jgallego, calera}@dlsi.ua.es

IEEE Workshop on Applications of Computer Vision (WACV), Snowbird, Utah (2009). ISBN: 978-1-4244-5498-3, ISSN: 1550-5790. Calificado como CORE A

Improving Edge Detection In Highly Noised Sheet-Metal Images

Antonio-Javier Gallego-Sánchez, Jorge Calera-Rubio

Departamento de Lenguajes y Sistemas Informáticos,
University of Alicante, Spain,
{*jgallego, calera*}*@dlsi.ua.es*

Abstract

This article proposes a new method for robust and accurate detection of the orientation and the location of an object on low-contrast surfaces in an industrial context. To be more efficient and effective, our method employs only artificial vision. Therefore, productivity is increased since it avoids the use of additional mechanical devices to ensure the accuracy of the system.

The technical core is the treatment of straight line contours that occur in close neighbourhood to each other and with similar orientations. It is a particular problem in stacks of objects but can also occur in other applications. New techniques are introduced to ensure the robustness of the system and to tackle the problem of noise, such as an auto-threshold segmentation process, a new type of histogram and a robust regression method used to compute the result with a higher precision.

2.1 Introduction

Our aim is to accurately detect the location and the orientation of objects so as to transmit this data to industrial robots. The objects that we are working with are rectangular, but the algorithm allows other types of straight edged shapes. These objects are stacked up and always are placed in roughly the same location. The main problem is that although their location is known, as they are stacked up, they could be gently moved or rotated. It causes a lot of noise just around the areas of interest, due to faint edges: overlapping edges, little projections or parallel laterals (better explained in section 2.2).

Production lines use an intermediate system, called *squaring machine*, to obtain the precise location of these objects. This process is very slow, because the robotic arm, after taking a sheet from the stack, has to deposite it in this machine to obtain the correct orientation, and then to pick it up again. To be more efficient and effective, our method only employs artificial vision to solve this problem. Therefore, it accelerates the whole process (as shown in the experimentation), since it avoids the necessity of this additional mechanical device.

Accuracy and robustness are fundamental in the area of industrial applications, because it directly influence the results of a subsequent processing. There are also several problems and requirements inherent in these types of environments, such as the multiplicity of experimental situations, the lighting, the shadows or the possible defects of the objects. Moreover, the system has associated another major difficulty: The lens of the camera produce a radial distortion which change the appearance of the image, so that straight lines appear to be curve. In addition, it is a generic problem in which the objects can have different sizes, shapes and colors.

For all these reasons, we can not soley rely on the results of basic techniques. The proposed algorithm is designed specifically for this type of problem, but it can have many other applications as proposed in the conclusions. The following section describes the proposed method in details. Section 2.3 shows the results of the experimentation and a comparison with other methods. The last section presents the conclusions and the lines for future work.

2.2 Description of the Algorithm

The main problem in detecting the position and the orientation of stacked objects is that they are usually slightly moved. It may seem that the topmost sheet occludes all others. But in general, a minimal variation creates a bundle of such lines just around the areas of interest. In addition, there may be dirt or burrs caused by the production of the sheets close to the margin or highlights from the cutting or punching deformations. From now on, all these lines will be referred as lines of noise (see Figure 2.1).

The following algorithm is proposed to detect the location and the orientation of the topmost object of a stack of objects:

1. *Capture* an image and *correct* the distortion. All the images are taken from the top using controlled lighting in order to improve imaging conditions.

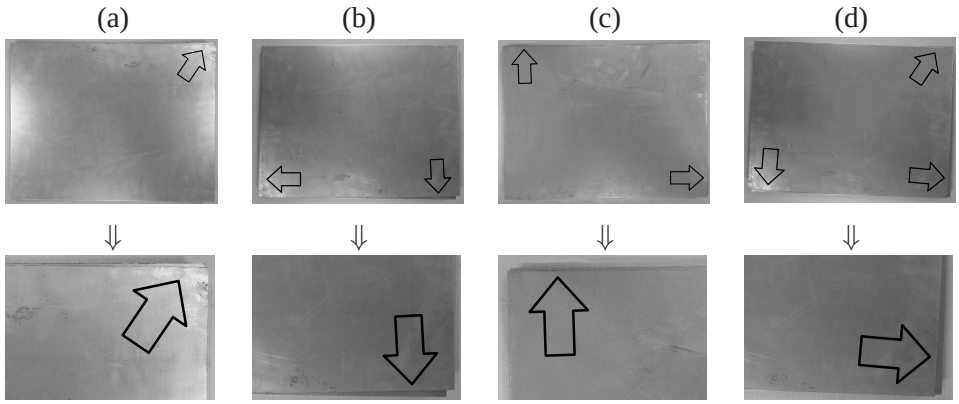


Figure 2.1: *First row: Different images of the used dataset (see section 2.3). Second row: Detail of the above image in which appears more lines of noise (indicated by an arrow).*

2. *Iterative edge-based segmentation with auto-threshold.*

3. *Hough transform*: It is applied to find the straight lines of the object. The extracted lines are also *filtered* and *classified*.

4. *Weighted union of nearly-coincident lines*: It joins the discontinued lines.

5. *Accumulated size histogram*: It adjusts the lines that belong to each lateral.

6. *Calculate the representative line of each lateral*: To perform this task a robust regression estimator is used.

Steps 1 and 3 are known techniques, but properly tuned to match the specific problem requirements. In steps 2, 4, 5 and 6 new algorithms are introduced. Each step is described in details in next sections.

2.2.1 Calibration and Undistortion

Lots of research has been done in this area (in [10] can be seen a review). We have used the calibration algorithm proposed by Zhang [10]. It uses a two-dimensional calibration pattern, and it does not need to know the movement of the camera. Moreover, the modifications proposed in [7] have been applied.

They improve the results using different pre and post-processes, and separating in two parts the calculation of the parameters. If all the parameters are calculated at the same time, the results would be worse. It is due to the fact that the error in a variable could be offset by other errors. The steps that this algorithm follows are:

1. Take n images of the calibration pattern from different positions.
2. Detect the reference marks of the calibration pattern with subpixel precision.
3. Preprocess the distortions and normalize the coordinates of the points [1, 3].
4. Estimate the matrix of the intrinsic camera parameters [10].
5. Compute the extrinsic parameters given the intrinsic parameters [10].
6. Calculate the undistortied image using the homografy matrix H.

This algorithm reduces the average error of the estimation to 0.053 ± 0.012 (in points of the image, using the same resolution as in [10]). It improves the results of the basic algorithm of Zhang, which obtains an error of 0.345 ± 0.059. This error is calculated using the average of geometric distances between the set of measured points and the marks of the pattern projected with the estimated parameters [7].

To reduce the error of calibration, the following considerations are proposed: Take about 10 pictures in which the pattern covers the whole image. According to [9], there is no improvement in the results with more than 8 images. The same camera parameters have to be maintained for the calibration and for all the images that are going to be undistortied.

2.2.2 Iterative Segmentation with Auto-Threshold

A Canny segmentation algorithm is applied iteratively over the undistortied image until the gray level reaches the value of δ. This value is an empirically calculated percentage of the level of border pixels that the segmented image should have (In our case, it is set to $\delta = 2.5\%$ and $\xi = 0.5\%$).

```
th := lastThreshold;
do {
   img := Canny(img, th);
   δc := CalculateGrayLevel(img);
   if( δc > δ ) th++;
   else         th--;
} while( | δc - δ | > ξ );
lastThreshold := th;
```

Other edge-based segmentation algorithms have been tried, but due to variability (lighting, shadows, defects, colours) better results are obtained by applying a simple method iteratively until the desired threshold is reached. In this way, it obtains more robustness to possible changes in brightness and lighting.

2.2.3 Hough Transform

Hough transform [8] is applied to extract the straight lines that appear after the segmentation. We use the classical algorithm which is based on a probabilistic approach to find straight lines. We have also tried some variations of the classical algorithm, such as multi-scale or fuzzy Hough, obtaining similar results. Therefore, it is preferred to use a simple and fast method to extract the segments. The threshold values are set reaching a compromise, because, if it removes a lot of noise, it may also remove real edges (In our case, they are set to: $threshold = 30$, $minimum - length = 50$, and $maximum - gap = 20$).

The extracted segments are filtered and classified using the following criteria:

1. *Orientation*: A range of possible orientations $[\theta_1, \theta_2]$ is established ($\theta_R \pm 10^o$, where θ_R is the reference orientation of the object). The lines with a different orientation are filtered. Overlapping lines are also filtered, because Hough algorithm sometimes obtains more than one result for the same line.

2. *Location*: It is used to:

   (a) *Classify*: Each line is classified in the set $\mathscr{M}_k$ for its corresponding lateral k (For this type of objects: $0 \leq k < 4$).

   (b) *Filter*: All the lines whose location is outside of the limits are removed. These limits are obtained from the known dimensions of the

object (see section 2.3) and the range of allowed error in the displacement ($\mathscr{R}_k \pm 0.05$m, where $\mathscr{R}_k$ is the expected position for the lateral $\mathscr{M}_k$).

2.2.4 Weighted Union of Nearly-Coincident Lines

This process is applied to join discontinued lines, and form a new line l_j with a new orientation θ_j. The new angle is calculated by weighting the angle of each line by its size, thereby, it gives more weight to the orientation of long lines and forms more solid lines. θ_j is obtained using $\theta_j = \sum \frac{\theta_i \cdot d(l_i)}{d_t} \; \forall l_i \in \mathscr{N}$, where $d(l_i) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$, $d_t = \sum d(l_i) \; \forall l_i \in \mathscr{N}$ and $\mathscr{N}$ is the set of lines which fulfills all the following criteria:

- Lines with the same classification (Section 2.2.3).
- Coincident lines with the same slope and *y*-interception. However, a small angle error $|\theta_1 - \theta_2| \leq \phi$ is permited (In our case, it is set to $\phi = 0.05$). This way, it also allows the union of lines that are nearly coincident.
- Lines within a certain range of distance ($d_m \leq max(d(l_i))/30 \; \forall l_i \in \mathscr{M}_k$). It uses the *point-line distance* equation $d_m = \frac{\vec{AB} \times \vec{AC}}{|\vec{AB}|}$, from the line AB to the midpoint C of the line to join.

All the lines which keep having a size smaller than a given threshold $\mathscr{U}$ are filtered (It is set to $\mathscr{U} = max(d(l_i))/20 \; \forall l_i \in \mathscr{M}_k$). These lines are mainly created by points of noise, so it often has a wrong orientation.

It would be possible to adjust the Hough parameters to be less restrictive, but this post-processing obtains better results because: First, with a more restrictive parameters, it ensures that the extracted lines actually appear in the image. Second, this process takes into account the possible angle errors, allowing a small error in the union, and also adjusting the angle of the resulting line.

2.2.5 Accumulated Size Histogram

An histogram is calculated for each of the main orientations of the object (vertical and horizontal). It establishes a relation between the lines' location and their size. To calculate the vertical histogram (Fig. 2.2(a)) the equation 2.1 is used, where d_k is the accumulated size of all the lines in the set L_k, $L_k = \{l_i : k \leq b_i <$

$k+1,\ 0 \leq k < img_{Width}\}$ (Fig. 2.2(c)) and $l_i \equiv x = a_i y + b_i$. Thus, the histogram is divided into ranges of $[k, k+1]$ which accumulate the size of all the lines that appear in them. For the horizontal histogram (Fig. 2.2(b)) the same equation is used, but $l_i \equiv y = a_i x + b_i$ and $0 \leq k < img_{Height}$.

$$d_k = \sum d(l_i) \ \ \forall\, l_i \in L_k \tag{2.1}$$

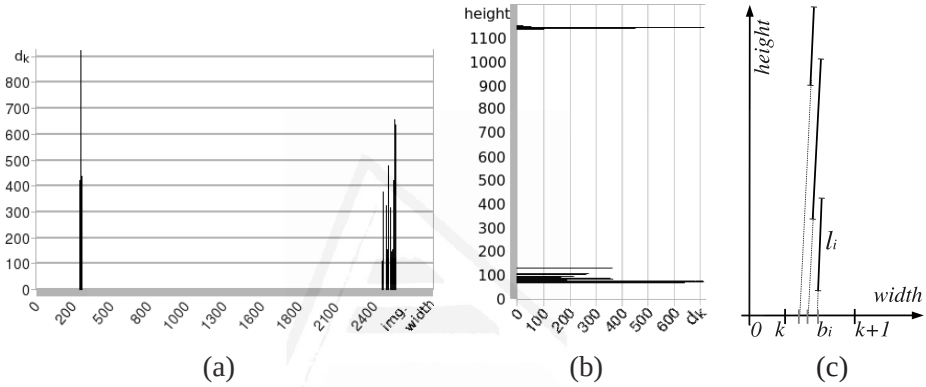


Figure 2.2: *(a, b) Accumulated-size histogram for the vertical and the horizontal lines. (c) Size-accumulation scheme for the range* $[k, k+1]$.

We have also tried other types of histogram, such us accumulating the size at the midpoint of each segment. But the current filter obtains better results because it takes into account the main orientation of each lateral and the orientation and the position of each individual line.

These histograms are used to remove the outliers (lines of noise) in the distribution. First, the standard deviation σ is calculated for each mode. Then, the interquartile range (IQR) is defined as $IQR = 1,35\sigma$ [5] and its limits are set as lower quartile $Q_1 = \bar{x} - IQR/2$ and upper quartile $Q_3 = \bar{x} + IQR/2$. All the lines which are outside the range $[f_1, f_2] = [Q_1 - 1,5IQR,\ Q_3 + 1,5IQR]$ are considered outliers, and therefore are removed. In other words, the lateral's frequency distribution is compared with an ideal distribution (a perfect edge represented by a single line) which identifies the atypical values in the distribution.

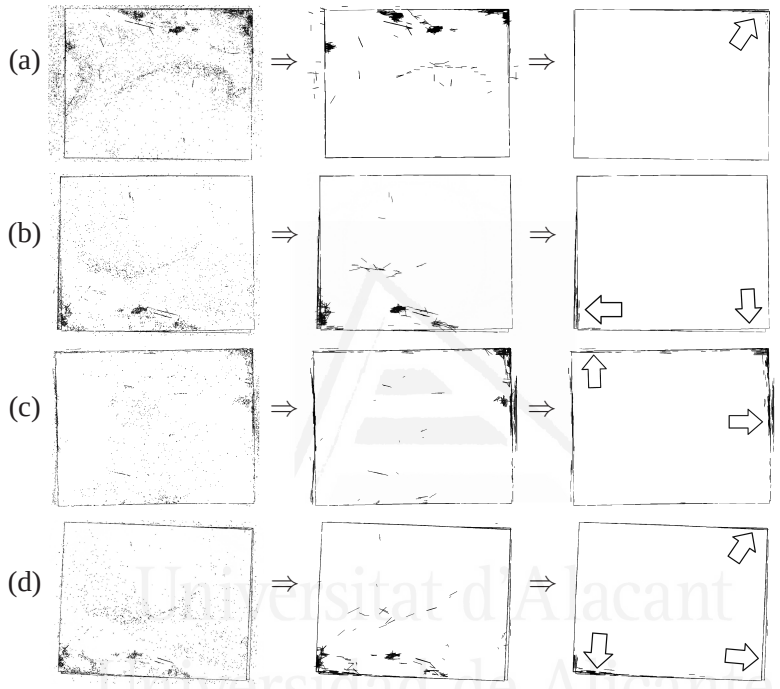


Figure 2.3: *Steps of the process for the objects in Figure 2.1: Iterative segmentation, Hough transform and filtering. The parts with more lines of noise are indicated by arrows.*

2.2.6 Calculate the Representative Line of each Lateral

To calculate the representative line of each side, instead of using a couple of points of each line, a sampling of the filtered lines at a constant distance τ is used. Thus, more weight is applied to long lines. τ is calculated as $\tau = gcd(d(l_i)) \ \forall \ l_i \in \mathscr{M}_k$. The sampling equation is derived from the formula of the Euclidean distance $\tau^2 = (x_2 - x_1)^2 + (y_2 - y_1)^2$ and the straight line's equation $y_2 = ax_2 + b$. The combination of these two formulas returns two solutions that correspond to the previous and the next point.

This sampling process returns a two-dimensional sample (x_1, y_1), (x_2, y_2),..., (x_n, y_n) for each lateral. The intention is to approximate this distribution by the line which minimizes the sum of quadratic differences between the observed values and the adjusted by the line [6]. The problem is that the *linear model* provides an estimator which is very sensitive to the possible presence of abnormal data and homoscedasticity. *Robust regression* is a powerful alternative which obtains better results with a slightly higher computational cost. It is insensitive to small departures from the idealized assumptions for which a statistical estimator is optimized [4]. The basic idea is to calculate the estimator α which minimizes the function:

$$\Psi(\alpha) = \sum_{i=1}^{n} \omega_\alpha(e_i) e^2 \tag{2.2}$$

where ω is a weighted function that is introduced to reduce (or even remove) the effect of the outliers. Thus, the weights $\omega(e_i)$ are defined to take small values for the high remainders e_i. There are many estimators that can be used for this purpose. In this case, as the error is not normally distributed, it is better to use a maximum-likelihood formula for the parameter estimation [6].

The independent variable is changed depending on the orientation of the lateral. The vertical and the horizontal laterals will be adjusted to the line which minimizes the error $e_i^2 = (x_i - \hat{x}_i)^2$ and $e_i^2 = (y_i - \hat{y}_i)^2$ respectively.

Therefore, it obtains good results without the need of using a *multiple regression model*, much more complex and time-comsuming. It has also been compared with other robust methods, such as iterative re-weighting [6] and RANSAC [3] (see section 2.3), obtaining similar results but with a higher computational cost. Moreover, the solutions obtained with these alternative methods may not be the optimal one and the convergence is not guaranteed.

2.2.7 Calculating the Final Location and Orientation

The last step is to calculate the correlation coefficient R of each representative line. It is a measure of the linear relationship among variables. An absolute value of $|R|$ close to 1 indicates a strong linear relationship. Therefore, it allows to identify the most appropriate lateral to calculate the final orientation of the object (primary lateral). To calculate the final location and orientation, it is only necessary to know with precision one lateral and one corner. The location of the primary corner is calculated from the intersection between the primary lateral and the perpendicular lateral with the highest correlation coefficient. In the example of Figure 2.3, the chosen laterals are: (a) bottom-right, (b) right-top, (c) bottom-left and (d) top-left.

The location of the rest of laterals can be estimated geometrically. First, the equivalence in pixels of the object's size has to be calculated. The conversion factor $\lambda = \{$ *CCD width in meters* / *Image width in pixels* $\}$ is used to convert pixels into metres. To calculate the CCD width, its dimensions $(1/3'')$ and its proportions $(\frac{x}{y} = \frac{4}{3})$ are needed. Therefore, the value of x (total width in metres) can be isolated from the equation $x^2 + y^2 = (\frac{1}{3}'')^2$. Next, the object's size is calculated using the general equation of lens $\frac{1}{s} + \frac{1}{s'} = \frac{1}{f}$, where s is the object distance, s' is the image distance and f is the focal length [2].

2.3 Results

In order to test the algorithm, a database of 200 images with a resolution of 3072×2304 pixels has been used[1]. These images are obtained from a top view of a stack of 50 sheets (similar to Fig. 2.1). In each image, slight variations in lighting and in the position and the orientation of the sheets are introduced. The dimensions of the sheet metal are $50 \times 40 \times 0.1$ cm. The obtained results have been compared with very precise measures taken manually over the image.

Table 1 shows the results of the proposed method. It is also compared with previous results obtained using: RANSAC, iterative re-weighting (explained in section 2.2.6), Linear Regression, Robust Regression (RR) without sampling, RR without filtering and a simple Model-Based Technique (First, the edges closest to the center are detected, then they are used in this order to find the object that matches the model). In addition, other methods have been tried getting worse

[1] If you want to obtain this database, please contact the first author.

results, this is due to the high variability of the problem (see explanation in the Introduction section).

Table 1 shows the average error of the primary lateral orientation and of the primary corner location. It also shows the percentage of hits obtained during the selection of the most appropriate lateral (see section 2.2.7). The proposed method obtains better results with a similar computational cost. Its average angle error is 0.144°, with a minimum error of 0.02°and a maximum error of 0.25°.

The complete algorithm has an average computational cost of 0.37 sec. It is linear with respect to the size of the image. The squaring machine takes ∼15 sec. in performing the same task (as described in the introduction). So, if the total manipulation time by the robot is ∼60 sec., by incorporating the proposed system, productivity would be increased in ∼20 items per hour.

Method	Average angle error	Average corner error	Correct lateral %
Proposed method	0.144°	0.00142 m	95.7%
RANSAC	0.162°	0.00146 m	94.2%
Iterative re-weighting	0.171°	0.00149 m	93.9%
Linear regression	0.181°	0.00153 m	93.6%
RR without sampling	0.986°	0.0087 m	89.1%
Model-based technique	2.43°	0.0194 m	-
RR without filtering	3.67°	0.0256 m	78.6%

Table 2.1: *Benchmarking*

2.4 Conclusions and Future Work

A new method for accurate and robust detection of the location and the orientation of an object on low-contrast surfaces has been presented. It has been designed specifically for this type of problem, in which each step is focused to ensure the reliability and the accuracy of the system. It is a particular problem in stacks of objects but can also occur in other applications, such as industrial automation, control of measures, quality control, positioning of objects, and packaging. Moreover, the algorithm allows other types of straight edged shapes, sizes and colors.

According to the results, the proposed method could be incorporated into an operating environment to replace the squaring machine, thereby substantially increasing the productivity of the entire production line.

Acknowledgments

The authors thank the reviewers for their thoughtful comments which helped to improve the initial manuscript. This work is partially supported by Spanish MICINN (contract TIN2009-14205-C04-01, contract DPI2006-15542-C04-01) and CONSOLIDER-INGENIO 2010 (contract CSD2007-00018).

Bibliography

[1] M. Ahmed and A. Farag. Nonmetric calibration of camera lens distortion: differential methods and robust estimation. *Image Processing, IEEE Transactions on*, 14(8):1215–1230, Aug. 2005.

[2] O. Faugeras. *Three-dimensional computer vision: a geometric viewpoint*. The MIT Press. Cambridge, Massachusetts, 1993.

[3] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004.

[4] P. J. Huber. *Robust Statistics*. Wiley Series in Probability and Statistics, 1981.

[5] I. Michael Sullivan. *Fundamentals of Statistics*. Pearson Prentice Hall, 2006.

[6] W. H. Press, B. P. Flanner, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 2007.

[7] C. Ricolfe-Viala and A.-J. Sanchez-Salmeron. Improved camera calibration method based on a two-dimensional template. In *Pattern Recognition and Image Analysis*, pages 420–427, 2007.

[8] L. Shapiro and G. Stockman. *Computer Vision*. Prentice-Hall, Inc, 2001.

[9] W. Sun and J. R. Cooperstock. An empirical evaluation of factors influencing camera calibration accuracy using three publicly available techniques. *Machine Vision and Applications*, 17:51–67, 2006.

[10] Z. Zhang. A flexible new technique for camera calibration. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.

Publicación 3

Scene reconstruction and geometrical rectification from stereo images

A. Javier Gallego Sánchez, Rafael Molina Carmona, Carlos Villagrá Arnedo

Grupo de Informática Industrial e Inteligencia Artificial (I3A)
Universidad de Alicante, Alicante (Spain)

{*ajgallego, rmolina, villagra*}*@dccia.ua.es*

The 9th World Multi-conference on Systemics Cybernetics and Informatics. Orlando, Florida, USA, July 2005. Vol. 2. pp. 354 – 358. ISBN: 980-6560-52-3.
Calificado como CORE C

Scene reconstruction and geometrical rectification from stereo images

A. Javier Gallego Sánchez, Rafael Molina Carmona,
Carlos Villagrá Arnedo

Grupo de Informática Industrial e Inteligencia Artificial (I3A)
Universidad de Alicante, Alicante (Spain)
{*ajgallego, rmolina, villagra*}*@dccia.ua.es*

Abstract

A system to reconstruct three-dimensional scenes from stereo images is presented. The reconstruction is based on a dense disparity image obtained by a process of window correlation, applying a geometrical rectification before generating a three-dimensional matrix which stores the spatial occupation. The geometrical rectification is essential to correct the conical perspective of the camera and to obtain real scenes. For the geometrical rectification, three approaches are proposed, based on one linear and two logarithmic functions. As a result, the rectification allows the system to adequately correct the reconstruction, showing that the best choice depends on the features of the original images.
Keywords*: stereoscopic vision, disparity images, geometrical rectification and three-dimensional scene reconstruction.*

3.1 Introduction

A central aspect nowadays in artificial intelligence research is the perception of the environment by artificial systems. Specifically, stereoscopic vision opens new paths that in future will allow these systems to capture the three-dimensional structure of the environments without any physical contact. For instance, a first solution to three-dimensional reconstruction with stereo technology is developed in Carnegie Mellon University. The possibility of composing several three-dimensional views from the camera transforms is set out, to build the so-called “3D evidence grid”. [1]

Most proposals in this field are based on disparity maps obtained by the extraction of scene characteristics, such as corners, borders, and so on, or this extraction can be done once the disparity map is obtained [2, 3]. There are some solutions which use a background image to obtain objects and silhouettes, making a difference operation with the image to reconstruct the scene [4, 5]. It is also habitual to use a general model or *a priori* knowledge to compare the result with, so that the reconstruction is built using this knowledge as a model. This is the case with the reconstruction of faces or known objects [6]. For instance, in [2] the author reconstructs some basic objects from a stereo image using primitives, such as cubes or boxes, and the objects are displayed in 3D. Moreover, several views of a scene or a sequence of them can be used to make the reconstruction [7]; or it can be based on other type of sensors, such as laser range sensors.

All these algorithms cannot be applied in a general manner and their field of application is limited. Nevertheless, our proposal has some advantages, due to the fact that it does not make assumptions about the scene nor the object structure, no characteristics are extracted, it does not segment objects trying to find a known shape and a stereo pair only is used, not a sequence.

The method that we propose reconstructs a three-dimensional scene from a dense disparity map (the map contains depth information for every pixel in the image) obtained from a binocular camera. It makes a geometrical rectification to show the same aspect as the real scene, removing the conical perspective (see section 3.3) that the images from a camera show. For instance, if the image of a corridor is reconstructed with no rectification, the walls, the floor and the ceiling would appear with a slope, seeming to converge at a point.

In the second section a more detailed description of the problem is given, with special focus on stereo vision and disparity maps. The proposed model is explained in the third section, including several solutions. Experiments are shown in the fourth section and, finally, some conclusions and future works to be developed are presented.

3.2 Problem description

Stereo vision techniques are based on the possibility of extracting three-dimensional information from a scene, using two or more images taken from different view points. We will focus on the basic case of two images belonging to a scene. The function of corresponding pixels from each image must be obtained. Let

us consider that the camera objectives are parallel, so that the search area for each pixel in the image from the left camera (reference image) is reduced to the same row as the image from the right camera. Every pixel in this row, named epipolar line (figure 3.1), has a corresponding pixel in the other image, placed in a different column, due to the separation of cameras. The difference between the columns in absolute value is called disparity. The further the object is from the camera, the smaller the disparity, and vice versa.

Figure 3.1: *A stereo par showing an epipolar line (left). Image of disparity (right)*

Due to the fact that the disparity is a numerical value, it can be displayed as an image, named depth or disparity image (figure 3.1), assigning a grey level to every pixel. Bright colours represent high disparities (near objects) and dark ones low disparities (distant objects). There is an inverse relationship between the disparity value and the distance to the object. Depending on the camera geometry, the distance can be transformed to coordinates in an Euclidean space, where the centre is placed in the camera position. [2, 8, 9, 10, 11]

The quality of the three-dimensional reconstruction depends on the quality of the disparity map. Strange objects (or false objects) can cause mistaken shapes and depth values, so errors will be translated to the reconstruction. Another important consideration is the characteristics of the environment where the images are taken. Most techniques are developed to work in indoor environments, such as office spaces or similar. This is due to the fact that the capture systems usually have a limited reach and modelling of the environment is conducted using geometrical primitives (walls, ceilings, floors and, even, guiding marks). All these constraints avoid the problems to be solved in open spaces. Therefore, some unlimited and general solutions to non-structured environments are set out in this paper.

3.3 Proposed Model

3.3.1 Three-dimensional Reconstruction

The reconstruction is based on a dense disparity image obtained through a process of window correlation (the correspondence between pixels from both images is done using a window correlation criterion, in order to identify similar areas in both images) [10, 11]. This depth image V_{depth} contains the disparity which is associated to each pixel in the reference image (left image). Therefore, for every pixel [i,j] in the original image, we can find the disparity value in $V_{depth}[i][j]$. Horizontal and vertical components for each point are directly obtained from the row and the column in which the point is located in the image. [12, 13]

To perform a three-dimensional reconstruction process four basic steps are taken: firstly the disparity image is stored in a two-dimensional matrix, then some smooth filters are applied (average and/or median filters), then the geometrical rectification is done (see next section) and, finally, a three-dimensional matrix is generated to store the spatial occupation.

$$\begin{aligned} &M2D[x][y] = V_{depth}[x][y] \quad \forall x,y \in V_{depth} \\ &M2D = ApplySmoothFilters(M2D) \\ &M3D = ApplyRectification(M2D) \\ &Display(ObtainRealUnits(M3D[x][y])) \quad \forall x,y \in M3D \end{aligned} \tag{3.1}$$

where:

- V_{depth}: disparity matrix
- M2D: two-dimensional matrix
- M3D: three-dimensional matrix
- ApplySmoothFilters(): applies smooth filters on a 2D matrix, and leaves the result in a new 2D matrix.
- ApplyRectification(): applies geometrical rectification on a 2D matrix, and returns a 3D matrix containing the result.
- Display(): displays a given point on the screen.

- ObtainRealUnits(): returns a value in real units (metres) given a value in pixels.

3.3.2 Geometrical Rectification

In order to correct the perspective in the images a rectification is needed. An image taken with a camera is in conical perspective, such that all parallel lines converge at a point (see figure 3.2). So, to perform the reconstruction correctly the perspective must be rectified.

Figure 3.2: *Conical perspective*

A rectification on each point coordinates x and y is performed, maintaining the same value for z. The rectification depends on depth and on coordinates x and y. The higher the value of z, the higher the rectification; the more central the point appears in the image, the higher the rectification. Figure 3.3 shows the scheme for the rectification process: figure 3.3(a) shows a non-rectified scene in 3D, in figure 3.3(b) the scene is seen from the top (only x and z coordinates are shown), and figure 3.3(c) shows the scene after being rectified.

After the analysis of variables implied in geometrical rectification, we can conclude that the process of rectification depends on depth and difference between x, y and centre coordinates. As a result, the so called **Lineal Rectification** is obtained. The coordinates are lineally corrected, so that the rectification directly depends on grey level (that is, z coordinate) and position (x and y coordinates). In an ideal situation, the Linear Rectification rectifys the scene to

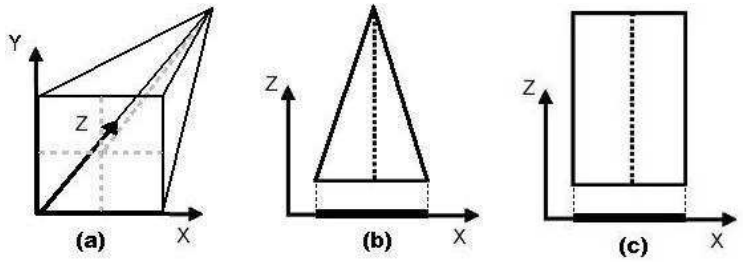


Figure 3.3: *Rectification scheme*

obtain the result given in figure 3.3(c), but in real images some problems arise. The most important drawback is the fact that this method does not distinguish whether the figure is very close, and so some errors occur with certain images, especially if the main object is too far from the camera. In fact, pixels corresponding to a distant object are split, leaving a hole whose dimensions increase as the distance to the object increases. So, important far non-centred objects can have holes and be dispersed. The rectification equation is:

$$\begin{aligned} newX &= x + (V_{depth}[x][y]/kMax) * factorX \\ newY &= y + (V_{depth}[x][y]/kMax) * factorY \\ V_{depth}[newX][newY] &= V_{depth}[x][y] \end{aligned} \tag{3.2}$$

Where:

- $V_{depth}[x][y]$ contains the grey level corresponding to the pixel of coordinates 'x' and 'y'.
- kMax is the maximum value of depth.
- $factorX = \begin{cases} -x & if\ x < width/2 \\ width - x & if\ x \geq width/2 \end{cases}$
- $factorY = \begin{cases} -y & if\ y < height/2 \\ height - y & if\ y \geq height/2 \end{cases}$

Observe that in the previous equation the new coordinates of a pixel are obtained from the former value of the coordinates, the depth value and the x, y

position of the pixel. Variables *factorX* and *factorY* contain the highest displacement that can be performed to place one pixel at the borders of the scene, that is, if the pixel is placed at the centre, the highest displacement is half the image.

The value of $V_{depth}[x][y]/kMax$ is in the range [0, 1], and it depends on the depth: it is 0 if the pixel is in the foreground, and it is 1 if it is in the background (in this case, if the pixel is also in the centre of the image, the rectification is maximum, so the pixel is moved to the border of the image).

In order to improve the rectification and avoid the holes in the main objects, we propose a **Natural Logarithmic Rectification**. In this case, a logarithmic function is applied to the depth value. The logarithm has the property of reducing the rectification when the object is close to the camera, and of magnifying the rectification when the object is far away. So, objects in the background suffer a higher correction than those in the foreground.

$$\begin{aligned} newX &= x + ln(V_{depth}[x][y]/kMax) * factorX \\ newY &= y + ln(V_{depth}[x][y]/kMax) * factorY \\ V_{depth}[newX][newY] &= V_{depth}[x][y] \end{aligned} \tag{3.3}$$

Finally, we also propose to use the **Base-10 Logarithmic Rectification**, whose function has a smoother slope than the natural logarithm. In some images, it is a better option due to the fact that small details do not disappear and holes are smaller.

$$\begin{aligned} newX &= x + log_{10}(V_{depth}[x][y]/kMax) * factorX \\ newY &= y + log_{10}(V_{depth}[x][y]/kMax) * factorY \\ V_{depth}[newX][newY] &= V_{depth}[x][y] \end{aligned} \tag{3.4}$$

3.4 Experiments

Some experiments have been done to prove the validity of every kind of proposed geometrical rectification. Figure 3.4 shows four cases: the first image corresponds to the reconstruction of the disparity image of figure 3.1, without any kind of rectification. The result of linear and both types of logarithmic rectifications are shown in images (b), (c) and (d) of figure 3.4. In the non-rectified image, the floor shows several steps, corresponding to the conical perspective effect.

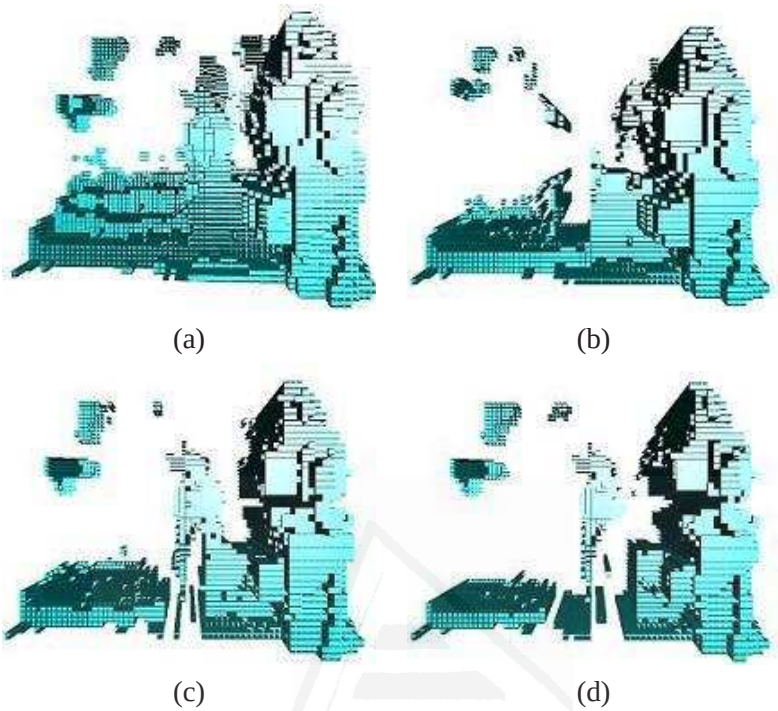


Figure 3.4: *Reconstruction and rectification: (a) no rectification, (b) lineal rectification, (c) natural logarithmic rectification and (d) base-10 logarithmic rectification*

The images in figure 3.4 show that the rectification reduces the slope of the floor, especially in logarithmic rectifications. However, as the rectification increases, some details tend to disappear. Figure 3.5 show the effect of rectification on the floor.

3.5 Conclusions and future work

In this paper we have presented a method of reconstructing three-dimensional scenes from stereo images. The reconstruction is corrected using three kinds of rectification. Although the results show that rectification improves the reconstruction, the final quality of the reconstructed image depends on the quality of the disparity map. In future, we will try to obtain better disparity images that

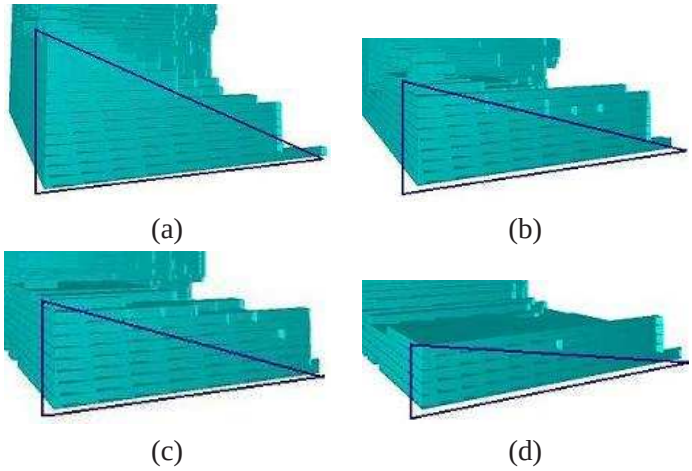


Figure 3.5: *Effect of rectification on the floor: (a) no rectification, (b) lineal rectification, (c) natural logarithmic rectification and (d) base-10 logarithmic rectification*

will improve the final result.

It can also be concluded that each type of rectification is suitable for a different kind of images. If the main object in the image is central and close to the observer, the best choice is the logarithmic rectification, to avoid holes in the shapes. However, is the scene is made up of several objects, distributed throughout the image, the best choice is linear rectification, to conserve all details.

We are very interested in obtaining a method to rectify all kinds of images. So, a new line of research, the Continuous Geometrical Rectification, is set out. Due to the discreteness of disparity maps, rectifying each pixel produces holes, as we have already shown. To solve this drawback, we propose treating the disparity maps in a continuous way, so that given two points in the space, the number of points (or pixels, in the map) between them is not finite but infinite. Nevertheless, although from a theoretical point of view the real space is continuous, the representation we use (the disparity map) is a discrete sample of the real space. So a way is needed to fill the "gaps" between given points (the pixels of the map). In order to fill these gaps it is first necessary to detect which the objects in the scene are, so some kind of segmentation is needed. As a first approach, the objects are detected using segmentation by colour and then, starting with the first pixel detected, the object is explored to obtain its silhouette. Only the points in

the silhouette are rectified and the holes are interpolated to obtain a filled object. In figure 3.6, the proposed scheme is shown.

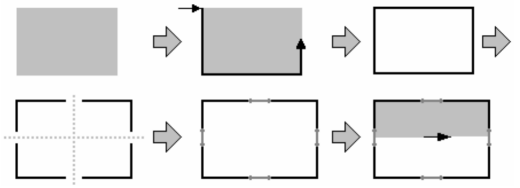


Figure 3.6: *Scheme for continuous rectification*

Acknowledgments

This work has been done with the support of the project "Scene reconstruction and integration of 3D visual information in an augmented reality system (GV04A-730)" given by the "Generalitat Valenciana" (regional government of Valencia, Spain).

Bibliography

[1] Hans P. Moravec. "Robot spatial perception by stereoscopic vision and 3D evidence grids". The Robotics Institute Carnegie Mellon University. Pittsburgh, Pennsylvania, 1996.

[2] Amador González, J.A. "Adquisición y procesamiento de imágenes estereoscópicas y modelado de mundos 3D para su implementación en exploración de ambientes". Tesis. Universidad de las Américas-Puebla. 2004.

[3] Camillo J. Taylor. "Surface Reconstruction from Feature Based Stereo". Proceedings of the Ninth IEEE International Conference on Computer Vision (ICCV 2003), pp. 184-190.

[4] Bastian Goldlücke, Marcus A. Magnor. "Joint 3D-Reconstruction and Background Separation in Multiple Views using Graph Cuts". Proceedings of CVPR 2003, pp. 683-694, IEEE Computer Society, Madison, USA, June 2003.

[5] K.M. Cheung, T. Kanade, J. Bouguet, and M. Holler. “A real time system for robust 3D voxel reconstruction of human motions”. Proceedings of the 2000 IEEE Conference on Computer Vision and Pattern Recognition (CVPR ’00), Vol. 2, June, 2000, pp. 714 - 720.

[6] Richard Lengagne, Pascal Fua and Olivier Monga. “3D Stereo Reconstruction of Human Faces driven by Differential Constraints”. Image and Vision Computing 18, pp.337-343, 2000.

[7] M. Li, H. Schirmacher, M. Magnor, and H.P. Seidel. “Combining stereo and visual hull information for on-line reconstruction and rendering of dynamic scenes”. In IEEE Workshop on MMSP, pages 9–12, 2002.

[8] E. Trucco and A. Verri. “Introductory techniques for 3-D Computer Vision”. Prentice Hall, 1998

[9] I. Cox, S. Ignoran, and S. Rao. “A maximum lilelihood stereo algorithm”. Computer Vision and Image Understanding, 63, 1996.

[10] Compañ Rosique, Patricia. “Estimación de la disparidad en visión estereoscópica. Tratamiento del color”. Tesis doctoral. Universidad de Alicante. 2004

[11] Satorre Cuerda, Rosana. “Visión estéreo, multirresolución y modelo integrado”: tesis doctoral. Universidad de Alicante. 2002

[12] M. Pollefeys, R. Koch and L. Van Gool. “A simple and efficient rectification method for general motion”. Proc. International Conference on Computer Vision, pp.496-501, Corfu (Greece), 1999.

[13] M. Pollefeys. “3D modelling from images”. Tutorial on conjunction with ECCV 2000, 2000.

Publicación 4

Discrete and Continuous Reconstruction of 3D Scenes from Disparity Maps

A. Javier Gallego Sánchez, Rafael Molina Carmona,
Carlos Villagrá Arnedo

Grupo de Informática Industrial e Inteligencia Artificial (I3A)
Universidad de Alicante, Alicante (Spain)

{*ajgallego, rmolina, villagra*}*@dccia.ua.es*

Proceedings of the Fifth IASTED International Conference on Visualization, Imaging, and Image Processing. September 7-9, 2005, Benidorm, Spain. pp. 366–371. ISBN 0-88986-528-0.
Calificado como CORE C

Discrete and Continuous Reconstruction of 3D Scenes from Disparity Maps

A. Javier Gallego Sánchez, Rafael Molina Carmona, Carlos Villagrá Arnedo

Grupo de Informática Industrial e Inteligencia Artificial (I3A)
Universidad de Alicante, Alicante (Spain)
{*ajgallego, rmolina, villagra*}*@dccia.ua.es*

Abstract

A system to reconstruct three-dimensional scenes from stereo images is presented. It is based on a dense disparity image obtained by a process of window correlation. Starting from these images and after the application of a **geometrical rectification**, the 3D reconstruction of the scene is obtained. The geometrical rectification is essential to correct the conical perspective of the camera and to obtain real scenes. Two 3D reconstruction types will be compared, one made in a discrete way and the other in a continuous way to try to solve the problems of the discrete reconstruction. In the case of the discrete reconstruction, two variants of the geometrical rectification equation are also proposed.
Keywords*: Disparity images, geometrical rectification, scene reconstruction.*

4.1 Introduction

Nowadays a central aspect in artificial intelligence research is the perception of the environment by artificial systems. Specifically, stereoscopic vision opens new paths that in the future will allow these systems to capture the three-dimensional structure of their environment without any physical contact. For instance, an initial solution to three-dimensional reconstruction with stereo technology was developed at Carnegie Mellon University. The possibility of composing several three-dimensional views from the camera transforms is set out, to build the so-called "3D evidence grid". [1]

Most proposals in this field are based on disparity maps obtained by the extraction of scene characteristics, such as corners, borders, and so on. Or this extraction can be done once the disparity map is obtained [2, 3]. Some solutions propose the use of a background image to obtain objects and silhouettes, performing a different operation with the image to reconstruct the scene [4]. It is also habitual to use a general model or *a priori* knowledge with which the result is compared, so that the reconstruction is built using this knowledge as a model. This is the case with the reconstruction of faces or known objects [5]. For instance, in [2] the author reconstructs some basic objects from a stereo image using primitives, such as cubes or boxes, and the objects are displayed in 3D. Moreover, several views of a scene, or a sequence of them can be used to make the reconstruction [6]; or it can be based on other types of sensors, such as laser range sensors.

All of these algorithms cannot be applied in a general manner and their field of application is limited. Nevertheless, our proposal has some advantages. Due to the fact that it does not make assumptions about the scene nor the object structure, it does not segment objects trying to find a known shape and only a stereo pair is used, not a sequence. Moreover, there are very few works which focus on creating a good reconstruction and on obtaining a real appearance of the scene.

The two methods proposed here reconstruct a three- dimensional scene from a dense disparity map (the map contains depth information for every pixel in the image) obtained from a binocular camera. Both make a geometrical rectification to show the same aspect as the real scene, removing the conical perspective (see section 4.3) that the images from a camera show. For instance, if the image of a corridor is reconstructed with no rectification, the walls, the floor and the ceiling would appear with a slope, seeming to converge at a point.

In the next section a more detailed description of the problem is given, with special focus on stereo vision and disparity maps. The geometrical rectification and the two types of reconstruction are explained in the third section. The experiments are shown in the fourth section, where we compare two reconstruction methods which use the geometrical rectification equation. One of them is carried out in a discrete way and the other in a continuous way. Finally, experiments on both methods and some conclusions and future works to be undertaken are presented.

4.2 Problem description

Stereo vision techniques are based on the possibility of extracting three-dimensional information from a scene using two or more images taken from different view points. We will focus on the basic case of two images belonging to a scene, obtained with a stereo camera with parallel objectives. In order to gather this 3D information, a function that computes the correspondence between the pixels from the left camera (reference image) and those from the right camera must be defined. The difference between each of these pairs of pixels is a value called disparity. This information can be displayed as an image, and is known as the depth or disparity image (figure 4.1). Depending on the camera geometry, the distance can be transformed to coordinates in a Euclidean space, where the centre is placed in the camera position. [2, 7, 8, 9]

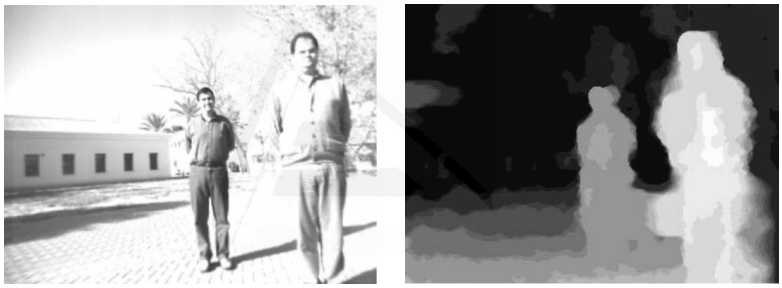


Figure 4.1: *Reference and disparity images.*

The methods of making the geometrical rectification and the discrete 3D reconstruction are presented in [10]. However, as these algorithms presented some problems in the visual aspect of the final result, further research has been carried out.

4.3 Proposed Model

In this section the proposed method is be presented. Firstly, the geometrical rectification equation, which is used by the two reconstruction methods, is briefly explained. Then a comparison between discrete and continuous reconstruction is shown, in order to be able to see the problems that they have and how they are solved.

4.3.1 Geometrical Rectification

In order to correct the perspective in the images a rectification is needed. An image taken with a camera is in conical perspective, such that all parallel lines converge at a point (see figure 4.2.a). So, to perform the reconstruction correctly the perspective must be rectified, thus the obtained result shows the same aspect as the real scene.

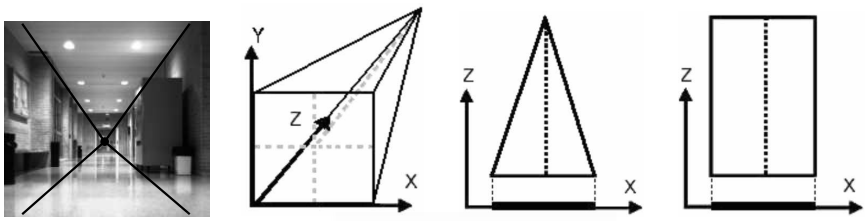


Figure 4.2: *(a) Conical perspective and (b, c, d) rectification scheme*

A rectification of each point for coordinates x and y is performed, whilst maintaining the same value for z. The rectification depends on depth and on coordinates x and y. The higher the value of z, the higher the rectification; the more central the point appears in the image, the higher the rectification. Figure 4.2 shows the scheme for the rectification process: figure 4.2(b) shows a non-rectified scene in 3D, in figure 4.2(c) the scene is seen from above (with only x and z coordinates shown), and figure 4.2(d) shows the scene after being rectified.

After the analysis of variables implied in geometrical rectification, we can conclude that the process of rectification depends on the depth and difference between x, y and centre coordinates. As a result, the so called **Lineal Rectification** is obtained. The coordinates are lineally corrected, so that the rectification directly depends on grey level (that is, z coordinate) and position (x and y coordinates). In an ideal situation, the Linear Rectification would rectify the scene to obtain the result given in figure 4.2(d), but with real images some problems arise. The rectification equation is:

$$\begin{aligned} newX &= x + (V_{depth}[x][y]/kMax) * factorX \\ newY &= y + (V_{depth}[x][y]/kMax) * factorY \\ V_{depth}[newX][newY] &= V_{depth}[x][y] \end{aligned} \tag{4.1}$$

Where:

- $V_{depth}[x][y]$ contains the grey level corresponding to the pixel of coordinates 'x' and 'y'.
- kMax is the maximum value of depth.
- $factorX = \begin{cases} -x & if\ x < width/2 \\ width - x & if\ x \geq width/2 \end{cases}$
- $factorY = \begin{cases} -y & if\ y < height/2 \\ height - y & if\ y \geq height/2 \end{cases}$

Observe that in the previous equation the new coordinates of a pixel are obtained from the former value of the coordinates, the depth value and the x, y position of the pixel. Variables *factorX* and *factorY* contain the highest displacement that can be performed to place one pixel at the borders of the scene, that is, if the pixel is placed at the centre, the highest displacement is half the image.

The value of $V_{depth}[x][y]/kMax$ is in the range [0, 1], and depends on the depth: it is 0 if the pixel is in the foreground, and it is 1 if it is in the background (in this case, if the pixel is also in the centre of the image, the rectification is maximum, so the pixel is moved to the border of the image).

4.3.2 Discrete three-dimensional reconstruction

The reconstruction is based on a dense disparity image obtained through a process of window correlation (the correspondence between pixels from both images is done using a window correlation criterion, in order to identify similar areas in both images) [9]. This depth image V_{depth} contains the disparity which is associated to each pixel in the reference image (left image). Therefore, for every pixel [i,j] in the original image, we can find the disparity value in $V_{depth}[i][j]$. Horizontal and vertical components for each point are directly obtained from the row and the column in which the point is located in the image. [11, 12]

To perform a three-dimensional reconstruction process four basic steps are taken: firstly the disparity image is stored in a two-dimensional matrix, then some smooth filters are applied (average and/or median filters), then the geometrical rectification is done (see previous section) and, finally, a three-dimensional matrix is generated to store the spatial occupation.

$$\begin{aligned} &M2D[x][y] = V_{depth}[x][y] \quad \forall x,y \in V_{depth} \\ &M2D = ApplySmoothFilters(M2D) \\ &M3D = ApplyRectification(M2D) \\ &Display(ObtainRealUnits(M3D[x][y])) \quad \forall x,y \in M3D \end{aligned} \tag{4.2}$$

where:

- V_{depth}: disparity matrix
- M2D: two-dimensional matrix
- M3D: three-dimensional matrix
- ApplySmoothFilters(): applies smooth filters on a 2D matrix, and leaves the result in a new 2D matrix.
- ApplyRectification(): applies geometrical rectification on a 2D matrix, and returns a 3D matrix containing the result.
- Display(): displays a given point on the screen.
- ObtainRealUnits(): returns a value in real units (metres) given a value in pixels.

The most important drawback is the fact that when the geometrical rectification equation (4.1) is applied, holes are produced in the 3D representation. This is due to the discreteness of disparity maps. In fact, pixels corresponding to a distant object are split, leaving a hole whose dimensions increase as the distance to the object increases. In order to solve these problems, a **Natural Logarithmic Rectification** is proposed. In this case, a logarithmic function is applied to the depth value. The logarithm has the property of reducing the rectification when the object is close to the camera, and of magnifying the rectification when the object is far away. So, objects in the background suffer a higher correction than those in the foreground.

$$\begin{aligned} &newX = x + ln(V_{depth}[x][y]/kMax) * factorX \\ &newY = y + ln(V_{depth}[x][y]/kMax) * factorY \\ &V_{depth}[newX][newY] = V_{depth}[x][y] \end{aligned} \tag{4.3}$$

We also propose to use the **Base-10 Logarithmic Rectification**, whose function has a smoother slope than the natural logarithm. In some images, it is a better option due to the fact that small details do not disappear and holes are smaller.

$$\begin{aligned} newX &= x + log_{10}(V_{depth}[x][y]/kMax) * factorX \\ newY &= y + log_{10}(V_{depth}[x][y]/kMax) * factorY \\ V_{depth}[newX][newY] &= V_{depth}[x][y] \end{aligned} \tag{4.4}$$

4.3.3 Continuous three-dimensional reconstruction

The following method is a 3D reconstruction system which works with the coordinates in a continuous way. It tries to solve the inconveniences that the original reconstruction algorithm has (see previous section) [10]. To solve these drawbacks, it is proposed to deal with the disparity maps in a continuous way, so that given two points in the space, the number of points (or pixels, in the map) between them is not finite but infinite. As a result of this, holes will not be formed when the geometrical rectification is made. This process basically takes the following steps:

1. The image is scanned by rows and columns, until the first pixel of an object's border is detected:
   (a) The object's border is scanned until it arrives at the beginning.
   (b) The interior of the object is emptied, so leaving only the border.
   (c) Each pixel of the border is geometrically rectified.
   (d) Any holes formed by the rectification of the border pixels are joined using lines.
   (e) The interior of the rectified object is then refilled.
2. The result is displayed

The proposed reconstruction algorithm only needs a single scan of the disparity image to complete all the steps of the process. In other words, firstly it scans the image looking for a new object. When one is detected the scan of its border begins. For each new border pixel, the following steps are taken: The interior of the object is emptied, the position of the pixel is rectified and joined

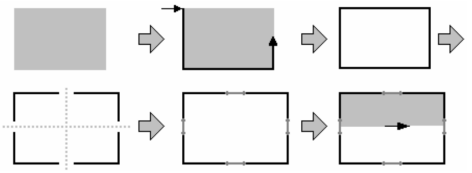


Figure 4.3: *Continuous rectification scheme*

with the previous pixel using a line, and finally, the interior of the rectified area is filled, leaving the result in a three-dimensional matrix.

In order to detect the objects, the method is based on the difference of tonality among the pixels, which greatly facilitates the task because the disparity images are in grey scale. Therefore, in order to detect a new object, the disparity image will be scanned until an intensity difference is found between two adjacent pixels. This intensity difference means that a new object has been located.

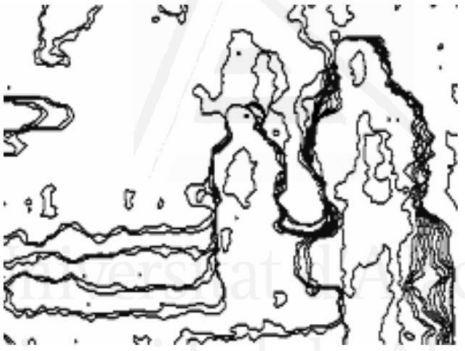


Figure 4.4: *Grey level segmentation of figure 4.1*

In the following sections, a more detailed description of each of the steps comprising the continuous reconstruction algorithm is given.

Scan of the Border (1.a)

The border of each object is scanned following its contour until arriving at the beginning (first detected pixel) of the same border. This kind of ordered scan is used because when each pixel of the object's border is rectified the previous pixel must be known (they must be joined in a later step). It would also be possible to

use a scan line algorithm, but in such a case a later process of ordering would be necessary, which would increase the cost.

The scan method is based on the fact that each pixel of the border has a neighbour which is external to the object (a different colour's pixel). This way, given a pixel of an object: another pixel also belonging to the object is searched for among its eight-neighbours, and subsequently it is checked to see if these two points share an external pixel (that has distance one to each of them). In the following algorithm these steps are detailed:

```
For each border's pixel scan their 8-neighbours
If it belongs to the object
   Scan the 8-neighbours of the new pixel
      If it shares an external pixel with the initial pixel
         Finish
```

Figure 4.5 shows the scheme for the scan process, where the grey boxes represent the interior pixels of the object and the white boxes, the exterior pixels. Figure 4.5(a) shows the current pixel marked with an X. In (b) the 8-neighbours of the current pixel are shown. The candidate pixel has been marked with a circle. In (c) the new 8-neighbours used to find an external pixel can be seen, and the shared external pixel has been marked with a circle. Finally, in (d) the new border's pixel is represented, and the direction of border growth represented by an arrow.

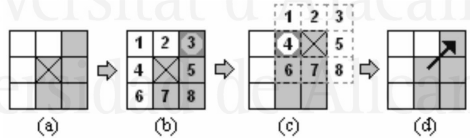


Figure 4.5: *Scan of the border scheme*

Empty the interior of the object (1.b)

As the border of an object is scanned, the interior of the object is emptied. This step is necessary to detect that the object has already been processed when the scan arrives back at the beginning (Figure 4.6).

Figure 4.6: *Emptying of the object*

Rectification of border pixels and joining using lines (1.c and 1.d)

The geometrical rectification (equation 4.1) is only carried out on the border's pixels (figure 4.7). When each pixel is moved to a rectified position, holes will appear in the silhouette. Hence, it will be necessary to join these holes using lines. The process is as follows: As the border of the object is scanned and its interior is emptied, so each pixel is rectified. Once the pixel has been moved to the rectified position it is joined with the previous pixel using a line. In this way, when the algorithm finishes, a complete contour of the object, without any holes, will remain.

Figure 4.7: *Rectification and joining using lines*

Fill the interior of the object (1.e)

Once the border has been detected, rectified and joined using lines, then the final step of the algorithm is to fill the object. As the information provided by the disparity image is available, the interior of an object can be easily identified. In this way, the borders are classified as either *incoming* (if its right area is interior to the figure) or *outcoming* (if its right area is external).

Scan lines will be followed to fill the interior of objects: when an incoming area is detected the current scan line is filled until arriving to the next border, and when an outcoming area is detected the scan line will be emptied. This process basically makes the following steps:

```
Scan the line of the image toward the right
If it is not a border of the object
   If it is incoming area → Fill
   Else if it is outcoming area → Empty
Else
   Finish
```

4.4 Experiments

Some experiments on the continuous reconstruction algorithm have been done in order to make comparisons with previous results. Firstly, a synthetic image is used to be able to check the steps made by the algorithm.

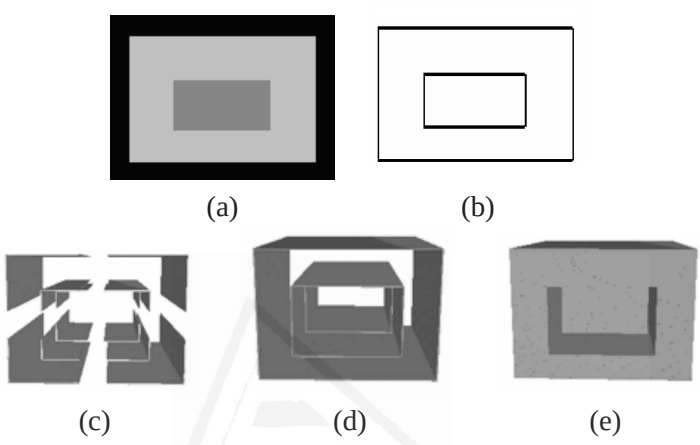


Figure 4.8: *Continuous reconstruction results*

Figure 4.8 shows the steps that the continuous reconstruction algorithm follows until the obtainment of the final 3D reconstruction (e). In (a) the disparity image used to make the reconstruction is shown. Figure (b) shows the objects obtained after segmentation. In (c) the scan of the border and the geometric rectification of each border pixel have been made. In (d) the holes which appeared in the borders during (c) have been linked using lines. Finally, in (e) the interior of the object has been filled.

In the following example (figure 4.9) a comparison of the different reconstruction types can be seen, based on the reference image (a) and on the disparity image (b). The results of discrete reconstruction and different types of rectification (lineal, natural logarithmic and base-10 logarithmic respectively) are shown in images (c, d, e) of figure 4.9. The geometrical rectification can result in some gaps appearing in the reconstruction result, even though the number of removed details from the shape will depend on the kind of geometrical rectification used. The lineal rectification has almost eliminated all the details, the other rectifica-

tion types work quite better. Finally, in (f) the result of a continuous 3D reconstruction can be seen, where no holes appear, so giving a more realistic aspect.

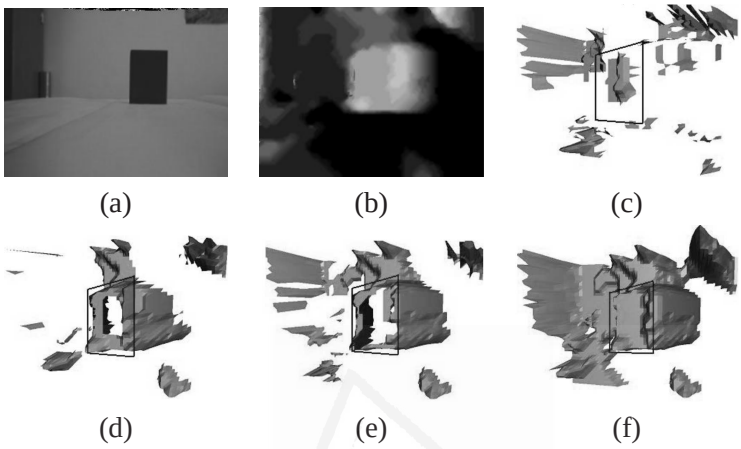


Figure 4.9: *Results of the reconstruction*

Using the disparity image of figure 4.1 other comparisons are shown below in order to test the effectiveness of the proposed methods with an image taken of an external scene. The results of the discrete reconstruction and the different types of rectification (lineal, natural logarithmic and base-10 logarithmic respectively) are shown in figure 4.10, images (c, d, e). The result has some gaps, as shown in the previous example, thereby creating an unreal appearance. In (d) the result of the continuous 3D reconstruction is shown, where the holes have been joined together, so giving a better result with a more realist aspect. Finally, a detail of the floor reconstruction is included, (images (e, f), to appreciate the reduction of the slope of the floor. In (e) the floor without any rectification is shown, and (f) shows how rectification has really reduced the slope.

It is important to note that the process time is less than 0.3 sec., therefore, it could be used for real-time applications. The time for the continuous reconstruction process takes only 0.09 sec. more than the discrete algorithm. The visualization time is only dependent on the precision required.

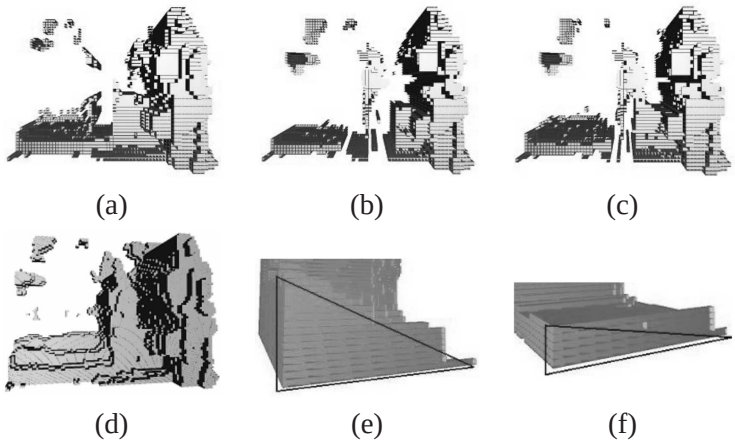


Figure 4.10: *Results of the reconstruction*

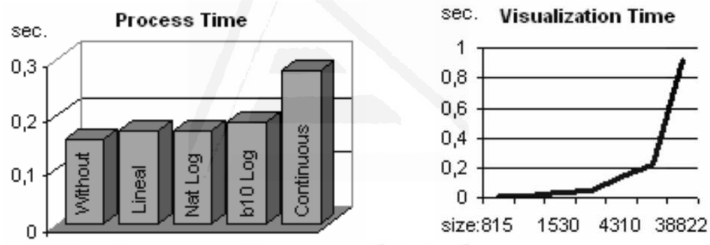


Figure 4.11: *Comparison of times*

4.5 Conclusions and future work

This paper has presented a new method of continuous reconstruction of 3D scenes from stereo images, as an improvement to a previous method based on discrete reconstruction [10]. The holes which appeared with the original algorithm have been eliminated, giving a realer aspect to the final reconstruction. Nevertheless, the final quality of the reconstructed image depends on the quality of the disparity map. In future experiments, better disparity images will try to be obtained that will improve the final result.

The proposals given here do have certain aspects that need to be improved,

however. When the objects to reconstruct are distant from the camera, the reconstruction suffers from aliasing. Moreover, the segmentation process makes this method a little slower than the previous ones.

Current work is focused on another method of continuous reconstruction, based on ”sub-pixel precision”. This method tries to solve the problems of the proposed algorithms. It is estimated that this method would be quicker than those proposed here, because it would remove the need to make segmentation or detection of borders. In addition, the obtained reconstruction would be of a better quality and would present a more realistic aspect, because it would eliminate the aliased borders.

Acknowledgments

This work has been done with the support of the project “Scene reconstruction and integration of 3D visual information in an augmented reality system (GV04A- 730)” given by the “Generalitat Valenciana” (Regional Government of Valencia, Spain).

Bibliography

[1] Hans P. Moravec. “Robot spatial perception by stereoscopic vision and 3D evidence grids”. The Robotics Institute Carnegie Mellon University. Pittsburgh, Pennsylvania, 1996.

[2] Amador González, J.A. “Adquisición y procesamiento de imágenes estereoscópicas y modelado de mundos 3D para su implementación en exploración de ambientes”. Tesis. Universidad de las Américas-Puebla. 2004.

[3] Camillo J. Taylor. “Surface Reconstruction from Feature Based Stereo”. Proceedings of the Ninth IEEE International Conference on Computer Vision (ICCV 2003), pp. 184-190.

[4] Bastian Goldlücke, Marcus A. Magnor. “Joint 3D- Reconstruction and Background Separation in Multiple Views using Graph Cuts”. Proceedings of CVPR 2003, pp. 683-694, IEEE Computer Society, Madison, USA, June 2003.

[5] Richard Lengagne, Pascal Fua and Olivier Monga. “3D Stereo Reconstruction of Human Faces driven by Differential Constraints”. Image and Vision Computing 18, pp.337-343, 2000.

[6] M. Li, H. Schirmacher, M. Magnor, and H.P. Seidel. “Combining stereo and visual hull information for on-line reconstruction and rendering of dynamic scenes”. In IEEE Workshop on MMSP, pages 9–12, 2002.

[7] E. Trucco and A. Verri. “Introductory techniques for 3-D Computer Vision”. Prentice Hall, 1998

[8] I. Cox, S. Ignoran, and S. Rao. “A maximum lilelihood stereo algorithm”. Computer Vision and Image Understanding, 63, 1996.

[9] Satorre Cuerda, Rosana. “Visión estéreo, multirresolución y modelo integrado”. Tesis doctoral. Universidad de Alicante. 2002

[10] Antonio Javier Gallego Sánchez, Rafael Molina Carmona, Carlos Villagrá Arnedo, “Scene reconstruction and geometrical rectification from stereo images”, World Multi- Conference on Systemics, Cybernetics and Informatics (WMSCI), Orlando, 2005

[11] M. Pollefeys, R. Koch and L. Van Gool. “A simple and efficient rectification method for general motion”. Proc. Internat. Conference on Computer Vision, pp.496-501, Corfu (Greece), 1999.

[12] M. Pollefeys. “3D modelling from images”. Tutorial on conjunction with ECCV 2000, 2000.

Publicación 5

Three-Dimensional Mapping from Stereo Images with Geometrical Rectification

A. J. Gallego Sánchez, R. Molina Carmona and C. Villagrá Arnedo

Grupo de Informática Industrial e Inteligencia Artificial
Universidad de Alicante, Ap.99, E-03080, Alicante, Spain

{*ajgallego, rmolina, villagra*}*@dccia.ua.es*

Proceedings of the 4th international conference on Articulated Motion and Deformable Objects (AMDO'06). Springer-Verlag - LNCS 4069 (2006) pp. 213 – 222. ISSN: 0302-9743. ISBN: 978-3-540-36031-5

Three-Dimensional Mapping from Stereo Images with Geometrical Rectification

A. J. Gallego Sánchez, R. Molina Carmona and C. Villagrá Arnedo

Grupo de Informática Industrial e Inteligencia Artificial
Universidad de Alicante, Ap.99, E-03080, Alicante, Spain
{*ajgallego, rmolina, villagra*}*@dccia.ua.es*

Abstract

In this paper we present a method for mapping 3D unknown environments from stereo images. It is based on a dense disparity image obtained by a process of window correlation. To each image in the sequence a geometrical rectification process is applied, which is essential to remove the conical perspective of the images obtained with a photographic camera. This process corrects the errors in coordinates *x* and *y* to obtain a better matching for the map information. The mapping method is an application of the geometrical rectification and the 3D reconstruction, whose main purpose is to obtain a realistic appearance of the scene.
Keywords*: Disparity images, Geometrical rectification, 3D mapping.*

5.1 Introduction

Nowadays, a central aspect in artificial intelligence research is the perception of the environment by artificial systems. It is a critical element in robot navigation tasks like map building (mapping) and self-location. Specifically, stereoscopic vision opens new paths that in the future will allow these systems to capture the three-dimensional structure of their environment without any physical contact. Moreover, range sensors can also acquire very detailed models [1], but these types of sensors are more expensive and they cannot provide information of both range and appearance, which is useful for navigation algorithms and texture mapping. For these reasons we will focus on stereo vision.

Several authors use stereo vision and disparity images to solve the 3D mapping problem. For instance, a first solution to three-dimensional reconstruction

with stereo technology was developed at Carnegie Mellon University. The possibility of composing several three-dimensional views from the camera transforms is set out, to build the so-called "3D evidence grid" [2]. There are other approaches which infer 3D grids from stereo vision, due to the fact that appearance information is not provided by range finders. Hence, they add an additional camera to their mobile robots [3, 4]. Moreover, a module of 3D recognition could be added to identify some objects. This technique is not exclusive of robotics, but it could be used in other applications such as automatic machine guidance or also for detection and estimation of vehicle movement [5].

Stereo vision can improve the perception of scenes and world modelling, so there are some methods which work with disparity images due to their advantages. The problem is that these algorithms cannot be applied in a widespread manner with all types of structures; because the images (or the objects) obtained from a camera have no real size, since they are deformed by the conical perspective effect.

We present an original mapping method which reconstructs the environment from a sequence of geometrically rectified images. For each pair of stereo images in the sequence a dense disparity map is calculated (the map contains depth information for every pixel in the image) and next it is geometrically rectified in order to show the same aspect as the real scene. This process is essential to remove the conical perspective of the images obtained with a binocular camera. Other simpler geometrical rectifications have already been used in other fields, like in [6] to rectify roads and to obtain their real appearance.

5.2 Proposed Model

5.2.1 Process Scheme

Stereo vision techniques are based on the possibility of extracting 3D information from a scene using two or more images taken from different view points. We will focus on the basic case of two images of a scene, obtained with a stereo camera with parallel objectives. In order to gather this 3D information, a function that computes the correspondence between the pixels from the left camera (reference image) and those from the right camera must be defined. The positional difference between each of these pairs of pixels is a value called disparity.

This information can be displayed as an image, and is known as depth or disparity image. Depending on the camera geometry, the distance can be transformed to coordinates in an Euclidean space, where the centre is placed in the camera position. [7, 8, 9]

In this work a 3D reconstruction method and a scene mapping algorithm are presented. For the 3D reconstruction several steps are followed: First, the disparity map is calculated starting from the images captured by the stereo camera. Then, a geometrical rectification process is applied in order to remove the effect of the conical perspective (see section 5.2.2). And finally, the 3D reconstruction is obtained (see section 5.2.3). The mapping algorithm is an application of this reconstruction method. It is based on a sequence of stereo images. For each image of this sequence its disparity map and its geometrical rectification are calculated. Once their space occupation matrix has been obtained, the mapping process is applied (see section 5.2.4).

Camera calibration and disparity algorithm are not the purpose of this paper. The disparity image is computed using multi-resolution and energy function [10, 11]. Moreover, it is important to note that the quality of the three-dimensional reconstruction depends on the quality of the disparity map. Errors in the disparity map can cause mistaken shapes and incorrect depth values, so errors will be transferred to the reconstruction.

5.2.2 Geometrical Rectification

In order to correct the perspective in the images a rectification is needed. An image taken with a camera is in conical perspective, such that all parallel lines converge at a point. As an example, figure 5.1(a) shows an image of a corridor, in which, due to the perspective effect induced by the acquisition conditions, the size of all the elements changes according to their distance from the camera. In this example, a pixel in the lower part of the image represents a small volume of the scene (it represents a part of the scene in the foreground); while a pixel in the centre of the same image represents a larger volume (because the part of the scene represented by the pixel is in the background). So, to correctly perform the 3D reconstruction the perspective must be rectified, thus the obtained result shows the same aspect as the real scene. [12, 13]

Figure 5.1 shows the scheme for the rectification process: figure 5.1(b) shows a non-rectified scene in 3D, in figure 5.1(c) the scene is seen from above (with

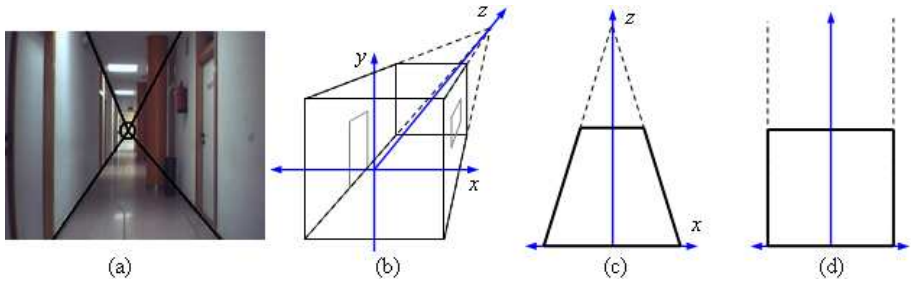


Figure 5.1: *Rectification scheme*

only x and z coordinates shown), and figure 5.1(d) shows the result which are desired after rectification.

After the analysis of variables implied in geometrical rectification, it can be concluded that the process of rectification depends on the depth and difference between x, y and centre coordinates. As a result, the so called **Linear Rectification** is obtained. The coordinates are linearly corrected, so that the rectification directly depends on grey level (that is, z coordinate) and position (x and y coordinates). In an ideal situation, the Linear Rectification would rectify the scene to obtain the result given in figure 5.1(d), but with real images some problems arise. The rectification equation is:

$$\begin{cases} x' := x + f\left(\frac{D(x,y)}{D_{max}}\right) \times \alpha \times w \\ y' := y + f\left(\frac{D(x,y)}{D_{max}}\right) \times \alpha \times w \\ z' := z \end{cases} \tag{5.1}$$

Where:

- f is a depth modifier. In the original equation it is a linear function but other functions can also be applied (explained below).
- $D(x,y)$ contains the grey level or the depth corresponding to the pixel of coordinates x and y.
- D_{max} is the maximum value of depth.
- α is a value in the range $[0,1]$ which measures the filter proportion.

- $w := \begin{cases} -x & \dots \quad \text{if } x < width/2 \\ width - x & \dots \quad \text{if } x \geq width/2 \end{cases}$

- $h := \begin{cases} -y & \dots \quad \text{if } y < height/2 \\ height - y & \dots \quad \text{if } y \geq height/2 \end{cases}$

It is important to note that in the previous equation the new coordinates of a pixel are obtained from the former value of the coordinates, the depth value and the x, y position of the pixel. Variables w and h contain the highest displacement that can be performed to place one pixel at the borders of the scene, that is, if the pixel is placed at the centre, the highest displacement is half the image. The value of $D(x,y)/D_{max}$ is in the range $[0,1]$, and depends on the depth: it is 0 if the pixel is in the foreground, and 1 if it is in the background (in this case, if the pixel is in the centre of the image, the rectification is maximum, so the pixel is moved to the border of the image).

The most important drawback is the fact that this method does not distinguish whether the figure is very close, and therefore, errors occur with some images, especially if the main object is too far from the camera. In fact, pixels corresponding to a distant object are split, leaving a hole whose dimensions increase as the distance to the object increases. So, important far non-centred objects can have holes and be dispersed.

In order to minimize these problems, a **Logarithmic Rectification** is proposed. In this case, a logarithmic function is applied to the depth value, substituting the function f. The logarithm has the property of reducing the rectification when the object is close to the camera, and of magnifying the rectification when the object is far away. So, objects in the background suffer a higher correction than those in the foreground.

5.2.3 Discrete Three-Dimensional Reconstruction

The reconstruction is based on a dense disparity image obtained through a process of window correlation (the correspondence between pixels from both images is carried out using a window correlation criterion, in order to identify similar areas in both images). This depth image contains the disparity which is associated to each pixel in the reference image (left image). Therefore, for every pixel in the original image, we can find the disparity value in $D(x,y)$. Horizontal and vertical components for each point are directly obtained from the row and the column in which the point is located in the image [9, 12, 13, 14]. In this

way, a three-dimensional matrix M_{3D} which represents the space occupation of the scene can be filled.

To perform a simple three-dimensional reconstruction process four basic steps are taken: firstly the disparity map (D) is stored in a two-dimensional matrix (M_{2D}) which has the same size as the disparity map ($m \times n$), and then some smooth filters can be applied if needed (average and/or median filters). Next, a geometrical rectification process (see section 5.2.2) is applied, which takes the 2D matrix (M_{2D}) and returns a 3D matrix (M_{3D}) containing the result. In this way, the matrix M_{3D} (which is initialized to zero) is filled, making $M_{3D}(x', y', D(x,y)) = 1$ where $x = 0, 1, ..., m-1$ and $y = 0, 1, ..., n-1$, which will indicate the space occupation of the final result. Starting from each of the M_{3D} depth values, their equivalence in real units (metres) is calculated and, finally, the result is shown.

1. $M_{2D} := ObtainDisparityData(D)$
2. $M_{2D} := ApplySmoothFilters(M_{2D})$
3. $M_{3D} := ApplyRectification(M_{2D})$
4. $Display(ObtainRealUnits(M_{3D}))$

The most important drawback is the fact that when the geometrical rectification equation (5.1) is applied, holes are produced in the 3D representation. This is due to the discreteness of disparity maps. In fact, pixels corresponding to a distant object are split, leaving a hole whose dimensions increase as the distance to the object increases. To minimize these problems the geometrical rectification filters which use a logarithmic function were introduced.

5.2.4 Mapping Algorithm

In this section a novel mapping algorithm is presented. It demonstrates the utility of the geometrical rectification and the advantages of its application to this kind of problem.

In order to do the 3D mapping of the scene, N stereo images $I_0, I_1, ..., I_{N-1}$ of the environment are taken. Each of these images is captured at a fixed distance. Once a stereo pair ($I_i, i = 0, 1, ..., N-1$) is obtained, its corresponding disparity map D_i is calculated and added to the Σ list which stores all the disparity maps. Next, the algorithm of geometrical rectification (explained in the

previous section) is used in order to compute the rectified matrix $M3D_i$ of each disparity map. For each matrix $M3D_i$ its intersection with the previous matrix is calculated ($M3D_{i-1} \cap M3D_i$), and its result is added to the main matrix M_{map} which represents the mapping of the scene. A cubic filter F (explained below) is applied to the whole matrix M_{map}, which discretizes the three-dimensional matrix and transforms it into a grid of rectangular cubes. Lastly, the result (M_{map}) is represented according to the space occupation of this matrix and calculating its equivalence in real units (metres). All these steps could be summarized as follows:

1. **for each** $D_k \in \Sigma$ **do**
   (a) $M_{2D} := ObtainDisparityData(D_k)$
   (b) $M_{2D} := ApplySmoothFilters(M_{2D})$
   (c) $M_{3D} := ApplyRectification(M_{2D})$
   (d) $M_{map} := M_{map} \cap M_{3D}$
2. $M_{map} := ApplyCubicFilter(M_{map})$
3. $Display(ObtainRealUnits(M_{map}))$

Cubic filter F applies the equation $g(x,y,z) := \Sigma_{(i,j,k)\in S} f(i,j,k)$ to each cube of the matrix, where S represents the set of point coordinates which are located in the neighbourhood of $g(x,y,z)$, including the point in question. In this way the space occupation of each cube is in the centre, and each cell contains the set of readings of that portion of the space. The number of readings is referred to as "votes", and represents the probability of space occupation.

Figure 5.2 shows the scheme for the mapping process: figure 5.2(a) represents the first image of the sequence, in figure 5.2(b) the scene is seen from above (with only x and z coordinates shown), and figure 5.2(c) shows the union of this image with the following image in the sequence; also, the intersection area of both can be seen.

5.3 Experiment Results

In this section the experiment results are shown. Figure 5.3 shows a reconstruction comparison using a synthetic disparity map (a) which simulates a corridor.

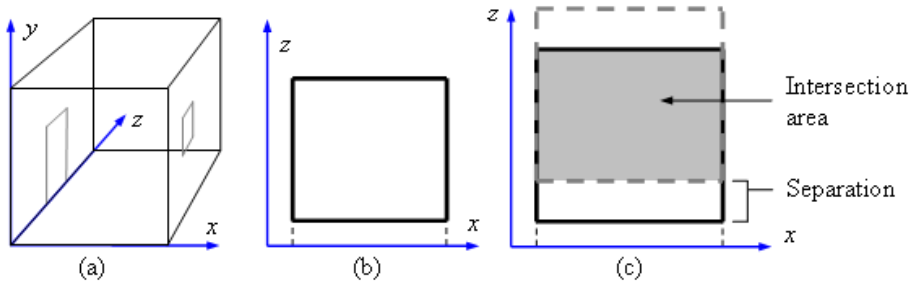


Figure 5.2: *Mapping scheme*

It clearly shows the effect of the geometrical rectification. In figure 5.3(b) no rectification is applied and in (c) the result of the rectification is shown. As can be seen in (c), the walls are perfectly rectified, becoming parallel as expected.

To do the mapping experimentation we took a sequence of 25 images of a corridor with a resolution of 320x240 pixels. Figure 5.4 shows three images of the sequence as well as their disparity maps, and the corridor plan is shown below. The main objective is that the walls, floor and roof appear without slope in the reconstruction; it is also important, that columns (represented by circles in the plan) are detected correctly and that there should not be any obstacle in the corridor.

In figure 5.5 the results of the corridor mapping are shown; a comparison between the different types of rectification can be seen. For all of them a cubic filter size of 3x3x3, and a number of votes of 5 have been used.

In figure 5.5(a) there is no geometrical rectification, and a wrong result is obtained: the in-between space of the corridor is not clear. In figure 5.5(b) the Logarithmic Rectification has been applied, but only with a factor of 50%. This result is better than the previous one, because the walls are limited and the in-between area of the corridor can be seen. In (c) the same type of rectification has been used, but increasing the factor to 100%. The result is similar to the previous one, although the corridor appears clearer. In figures 5.5(d) and (e) the Linear Rectification (without applying the logarithm function to the depth value) has been made. These results show a better definition of the corridor and a clearer in-between space, moreover, the columns can be distinguished on the right hand side (they are marked with an arrow). Figure 5.6 shows a pair of lateral views of the result 5.5(e).

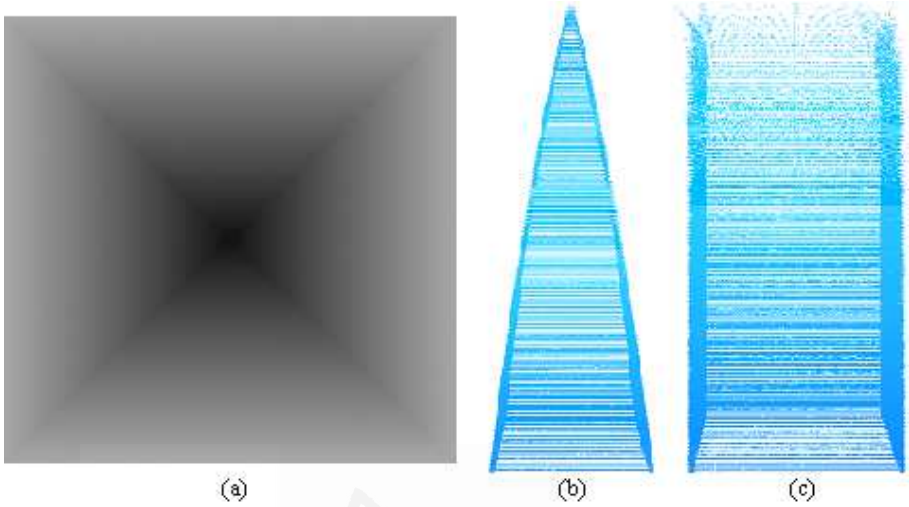


Figure 5.3: *Effect of the geometrical rectification on a corridor*

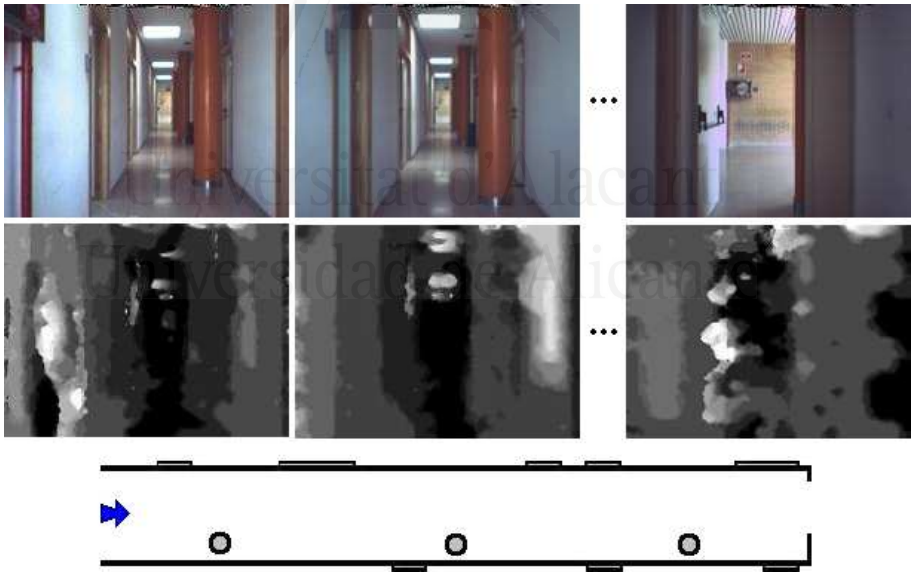


Figure 5.4: *Sequence of images for the mapping*

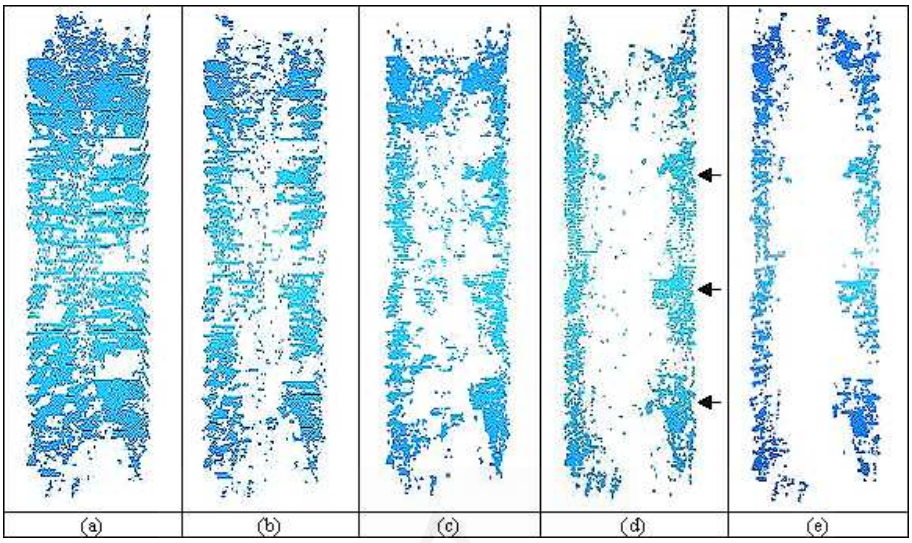


Figure 5.5: *Mapping results of the corridor*

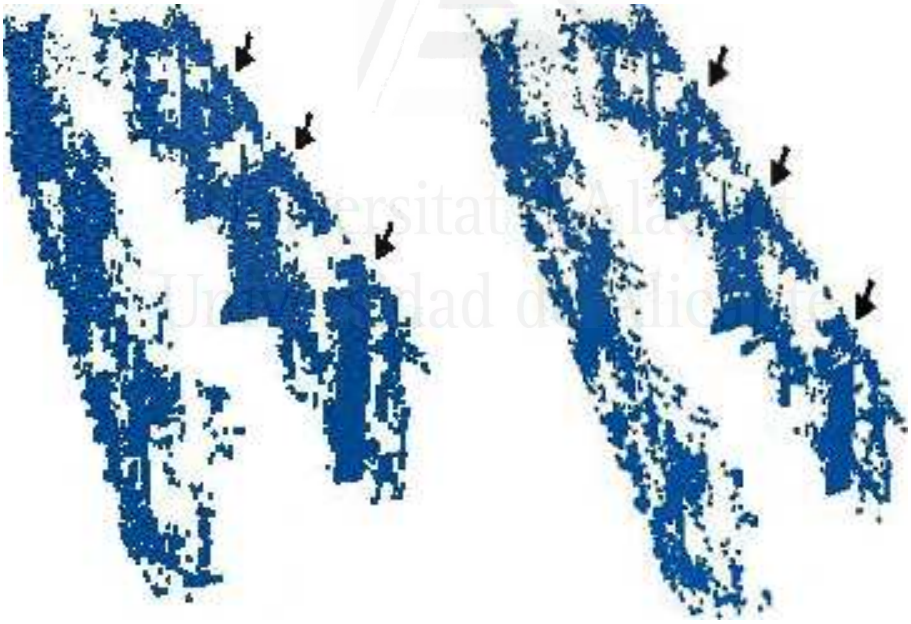


Figure 5.6: *Lateral views of the corridor*

To conduct the experiments, a Pentium IV 3,20GHz with 2GB of RAM and a 512MB graphic card has been used. The reconstruction of the map has been made using a 320x240x256 voxels matrix and disparity maps with a size of 320x240 pixels. Moreover, it is important to note that only the pixels which have some value in the disparity map are processed. In other words, the black pixels whose distance is considered infinite will not be processed. To process the sequence of 25 images (each image has a level of 70% of processed data) the algorithm takes approximately 8 seconds. This time depends on the precision of the final 3D reconstruction. So, the process time of an individual reconstruction is less than 0.3 seconds.

5.4 Conclusions and Future Work

This paper has presented a novel mapping algorithm which works with disparity maps in order to reconstruct unknown environments. It is an application based on a previous 3D reconstruction work. This method uses geometrical rectification to eliminate the effect of conical perspective, with the intention of obtaining a real aspect in the final result. The cubic filter is very useful to solve odometry problems; if it is not applied the number of coincidences would be too small. Nevertheless, the final quality of the reconstructed image depends on the quality of the disparity map. In future experiments, better disparity images will improve the final result.

These methods have several advantages. Firstly, due to the fact that in the reconstructed scene each position represents the same size, the matching of the mapping algorithm is improved, and also it would be possible to process the information in parallel, allowing a homogeneous distribution of the information among all the image pixels. For instance, as geometrical rectification is applied to all the pixels in the image, a SIMD massively parallel system could be used. Moreover, it can take advantage of the parallel processing to carry out more elaborate operations.

Current work is focused on applying textures to the reconstruction in order to give a more realistic aspect to the final result. Furthermore, we will try to improve the geometrical rectification, calculating the point of view and considering the used camera characteristics [1]. As future work, the obtained results will be used in an Augmented Reality system and in an autonomous robot system. It could solve the occlusion problem using the depth information from the disparity

map, and it could take advantage of the scene reconstruction to recognize their geometry, objects, and so on.

Bibliography

[1] Jose M. Sanchiz, Robert Burns Fisher: Viewpoint Estimation in Three-Dimensional Images Taken with Perspective Range Sensors. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, no. 11 (2000) 1324–1329

[2] Hans P. Moravec: Robot spatial perception by stereoscopic vision and 3D evidence grids. The Robotics Institute Carnegie Mellon University. Pittsburgh, Pennsylvania (1996)

[3] Stephen Se, D. Lowe, J. Little: Vision-based mobile robot localization and mapping using scale-invariant features. Proc. of IEEE International Conference on Robotics and Automation. Seoul, Korea (2001)

[4] C. Martin and S. Thrun: Real-time acquisition of compact volumetric maps with mobile robots. In Proceedings of ICRA'02: IEEE International Conference on Robotics and Automation (2002)

[5] Gonzalo Pajares Martinsanz, Jesús M. de la Cruz García: Visión por computador: imágenes digitales y aplicaciones. Ed. Ra-Ma, D.L. Madrid (2001)

[6] Alberto Broggi: Robust Real-Time Lane and Road Detection in Critical Shadow Conditions. In Proceedings IEEE International Symposium on Computer Vision, Coral Gables, Florida. IEEE Computer Society (1995)

[7] E. Trucco and A. Verri: Introductory techniques for 3-D Computer Vision. Prentice Hall (1998)

[8] I. Cox, S. Ignoran, and S. Rao: A maximum lilelihood stereo algorithm. Computer Vision and Image Understanding, 63 (1996)

[9] Oliver Faugeras: Three-dimensional computer vision: a geometric viewpoint. The MIT Press. Cambridge, Massachusetts (1993)

[10] Compañ, P.; Satorre, R.; Rizo, R.: Disparity estimation in stereoscopic vision by simulated annealing. Artificial Intelligence research and development. IOS Press. (2003) 160–167

[11] Patricia Compañ, Rosana Satorre, Ramón Rizo, Rafael Molina: Inproving depth estimation using colour information in stereo vision. IASTED International Conference on Visualization, Imaging and Image Processing (VIIP 2005), Benidorm (Spain) (2005) 377–389

[12] Antonio Javier Gallego Sánchez, Rafael Molina Carmona, Carlos Villagrá Arnedo: Scene reconstruction and geometrical rectification from stereo images. World Multi-Conference on Systemics, Cybernetics and Informatics (WMSCI), Orlando (2005)

[13] Antonio Javier Gallego Sánchez, Rafael Molina Carmona, Carlos Villagrá Arnedo: Discrete and Continuous Reconstruction of 3D Scenes from Disparity Maps. IASTED International Conference on Visualization, Imaging, and Image Processing, Benidorm (Spain) (2005) 366–371

[14] M. Pollefeys, R. Koch and L. Van Gool: A simple and efficient rectification method for general motion. Proc. International Conference on Computer Vision, Corfu (Greece) (1999) 496–501

Publicación 6

Rectified Reconstruction from Stereo Pairs and Robot Mapping

Antonio Javier Gallego, Rafael Molina, Patricia Compañ and Carlos Villagrá

Grupo de Informática Industrial e Inteligencia Artificial
Universidad de Alicante, Ap.99, E-03080, Alicante, Spain

{*ajgallego, rmolina, company, villagra*}*@dccia.ua.es*

Lecture Notes in Computer Science, Springer-Verlag, LNCS 4673 (2007), pp. 141-148. ISSN: 0302-9743.
Calificado como CORE A

Rectified Reconstruction from Stereo Pairs and Robot Mapping

Antonio Javier Gallego, Rafael Molina, Patricia Compañ and Carlos Villagrá

Grupo de Informática Industrial e Inteligencia Artificial
Universidad de Alicante, Ap.99, E-03080, Alicante, Spain
{*ajgallego, rmolina, company, villagra*}*@dccia.ua.es*

Abstract

The reconstruction and mapping of real scenes is a crucial element in several fields such as robot navigation. Stereo vision can be a powerful solution. However the perspective effect arises, as well as other problems, when the reconstruction is tackled using depth maps obtained from stereo images. A new approach is proposed to avoid the perspective effect, based on a geometrical rectification using the vanishing point of the image. It also uses sub-pixel precision to solve the lack of information for distant objects. Finally, the method is applied to map a whole scene, introducing a cubic filter.

6.1 Introduction

Nowadays, a central aspect in artificial intelligence research is the perception of the environment by artificial systems. It has been considered as one of the most important problems for effective autonomous robotic navigation and reconstruction [1]. Most research on robot mapping makes use of powerful computers and expensive sensors, such as scanning laser rangefinders. Nevertheless, other sensors, such as stereoscopic sensors, can be used. Stereo cameras are cheap and provide information of both range and appearance. Several authors use stereo vision and disparity images to solve the 3D reconstruction or mapping problems. For instance, a first solution to three-dimensional reconstruction with stereo technology explores the possibility of composing several 3D views from the camera

transforms, to build the so-called ”3D evidence grid” [2]. There are other approaches which infer 3D grids from stereo vision, due to the fact that appearance information is not provided by range finders. Hence, they add an additional camera to their mobile robots [3, 4]. Moreover, a module of 3D recognition could be added to identify some objects. This technique is not exclusive of robotics, but it could be used in other applications such as automatic machine guidance or also for detection and estimation of vehicle movement [5].

Any image taken by a camera is deformed by the conical perspective effect, so direct reconstruction generates scenes with unreal aspect. There are very few works which focus on creating a good reconstruction and on obtaining a real appearance of the scene. However, some interesting works can be found [6, 7], but none of them makes any type of perspective rectification. Specific objects are reconstructed instead of the whole scene, so the real structure of the environment is not recovered. Some naïve perspective rectifications have already been used in other fields, to rectify roads and to obtain their real appearance [8].

This work is centred in the reconstruction of the structure of the scene showing its real aspect, using the information provided by the stereo images and the disparity maps (in fact depth maps, their duals, are used). Our proposal does not make assumptions about the scene nor the object structure, it does not segment objects trying to identify known shapes, only a stereo pair is needed and it is nor correspondence dependent. Perspective rectification method allows to eliminate the conical perspective of the scene and to remove the camera orientation. This way the algorithm recovers the structure of the scene and some crucial information such us object geometry, volume and depth.

6.2 Process Scheme

The reconstruction and mapping methods follow the process scheme shown in figure 6.1. In the upper part of the figure, the reconstruction method is presented (section 6.3). Given a pair of stereo images (LI_k and RI_k, where k is an index indicating the current image within a sequence) the depth map and the vanishing point position are calculated. A depth map D_k and the vanishing point position VP_k are obtained as a result. Then, a rectification process is applied in order to remove the effect of the conical perspective. The result of this stage is a rectified 3D matrix of voxels R_k representing the reconstructed scene. In the lower part of

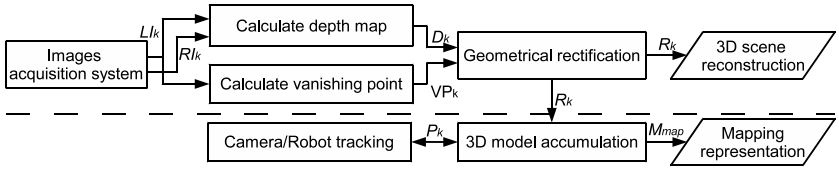


Figure 6.1: *Structure of both 3D reconstruction and mapping.*

the figure, the mapping method is represented (section 6.4). Every reconstruction R_k is accumulated to finally obtain a whole mapping representation of the environment M. The camera position is needed to carry out this final step.

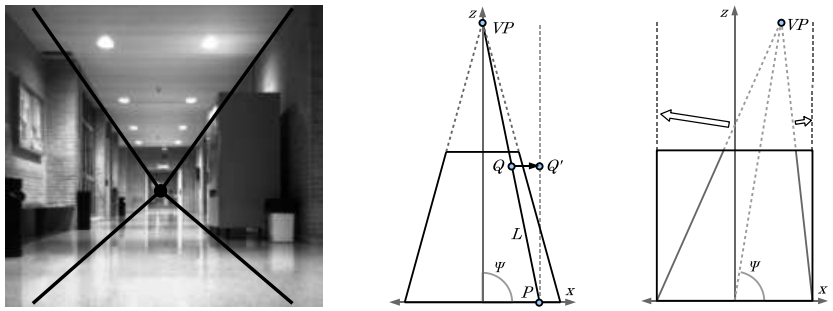
6.3 3D Reconstruction Method

6.3.1 Perspective Rectification using a Vanishing Point

Let LI and RI (the k subindex is removed for simplicity) be a pair of stereo images (left and right image) of $m \times n$ pixels, and D a dense depth map obtained by any method (in this approach, the depth image is computed using multi-resolution and energy function [9]). Each value $D(x,y)$ of the depth map contains the depth associated to the pixel (x,y) of the reference image (left image) [10, 11, 12, 13]. The reconstruction of the scene can be done by labelling a 3D matrix of voxels R representing the spatial occupation. However, due to the conical perspective, it would produce an unwanted effect in the reconstruction (figure 6.2(a) and 6.2(b)). So, to correctly perform the 3D reconstruction the perspective must be rectified making a correction to the pixel's coordinates. This way the obtained result will show the same aspect as the real scene (figure 6.2(c)).

Figure 6.2(b) shows the scheme of the process, in which the point Q (current pixel being processed obtained from the input depth map) with coordinates $(x,y,D(x,y))$ is rectified to obtain Q'.

The position of the scene vanishing point VP is calculated using the method proposed in [14]. It uses a Bayesian model which combines knowledge of the 3D geometry of world with statistical knowledge of edges in images. The method returns an angle (called as Ψ) which defines the orientation of the camera. This angle is transformed to Cartesian coordinates to obtain the position (x,y) of scene vanishing point VP. For the depth of point VP the maximum depth value (D_{max}) of the whole depth map is used.



(a) (b) (c)

Figure 6.2: *Rectification scheme: (a) shows the effect of the conical perspective, (b) shows a non-rectified scene seen from above (with only x and z coordinates shown), and (c) shows the result which is desired after rectification (with parallel walls). This image also illustrates the compass angle of the camera in the case of a lateral view.*

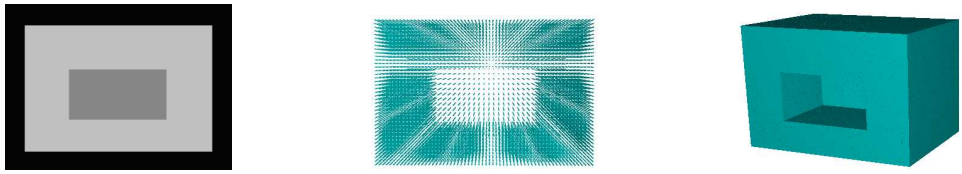
Once the VP is obtained, a line L is traced through VP and Q. Next, the intersection of the line L with the x-y plane is calculated, obtaining in this way the point P. Starting from P, the line L is rotated to be perpendicular to the x-y plane. After this process, the new point $Q' = (x', y', z')$ is calculated as follows:

$$\begin{cases} x_{Q'} = x_{VP} + z_{VP} \frac{x_{VP} - x_Q}{z_Q - z_{VP}} \\ y_{Q'} = y_{VP} + z_{VP} \frac{y_{VP} - y_Q}{z_Q - z_{VP}} \\ z_{Q'} = z_Q \end{cases} \tag{6.1}$$

6.3.2 3D Reconstruction using Sub-pixel Precision

Once the depth map (D) and the vanishing point (VP) are calculated (they can be obtained in parallel), the reconstruction process is performed, including the perspective rectification (section 6.3.1). The reconstruction method returns a 3D matrix (R) containing the space occupation of the final result. R is initialized to zero and, then, it is filled as follows: $R(x', y', D(x,y)) = 1$ where (x', y') are the rectified coordinates of (x,y), $\forall x, y \in \mathbb{R} / \{0 \leq x \leq m-1, 0 \leq y \leq n-1\}$. The final step is obtaining the real units. This is a direct calculation if the camera parameters are known.

The most important drawback is the fact that when the perspective rectification corrects the pixels' coordinates, the voxels are separated in the 3D rep-



(a) (b) (c)

Figure 6.3: *Example of reconstruction using the depth map of figure (a). Figure (b) shows the reconstruction without using sub-pixel precision, in which the voxels are separated due to the perspective rectification. In figure (c) the sub-pixel precision has been used. The result has a more realistic appearance because holes have been filled.*

resentation (figure 6.3(b)). This is due to the discreteness of depth maps. In fact, pixels corresponding to a distant object are split, leaving a hole whose dimensions increase as the distance to the object increases. To minimize these problems a sub-pixel precision technique is proposed to calculate the position of n fictitious pixels between two consecutive pixels. The precision used for the reconstruction is calculated using the equation $1-(z/D_{max})$, which returns the minimum value when the pixel is in the foreground and the maximum one when it is in the background. All the steps of the sub-pixel reconstruction method can be summarized as follows:

1. $D := CalculateDepthMap(LI, RI)$
2. $VP := CalculateVanishingPoint(LI)$
3. **while (** $x \leq m-1$ **)**
   (a) **while (** $y \leq n-1$ **)**
      i. $(x', y') := Rectify(x, y, VP)$
      ii. $R(x', y', D(\lfloor x \rfloor, \lfloor y \rfloor)) = 1$
      iii. $y = y + 1 - (D(\lfloor x \rfloor, \lfloor y \rfloor)/D_{max})$
   (b) $x = x + 1 - (D(\lfloor x \rfloor, \lfloor y \rfloor)/D_{max})$
4. $Display(ObtainRealUnits(R))$

(a) (b)

Figure 6.4: *Mapping comparison with and without rectification.*

6.4 Mapping Algorithm

The mapping algorithm is an application of the reconstruction method. It demonstrates the utility of the perspective rectification and the advantages of its application to this kind of problem. A significant example is shown in figure 6.4, where (a) shows two consecutive images of a corridor (there is also a column) and the result of their intersection. Since the scene is not rectified, the intersection of the columns and walls are not coincident. The rectification of the images in (a) is presented in (b). In this case, the intersection is perfectly coincident, so the mapping algorithm is significantly improved.

In order to do the 3D mapping of the scene, N stereo pairs (LI_0, RI_0), (LI_1, RI_1), ..., (LI_{N-1}, RI_{N-1}) of the environment are taken. Each of these images is captured at a fixed distance. Once a stereo pair (LI_k, RI_k) is obtained, its corresponding depth map D_k is calculated and added to the Σ list which stores all the depth maps. Next, the algorithm of perspective rectification is used in order to compute the rectified matrix R_k of each depth map. For each matrix R_k its intersection with the previous matrix is calculated ($R_{k-1} \cap R_k$), and its result is added to the main matrix M_{map} which represents the mapping of the scene. In this approach the position of the frames is obtained from robot odometry. The system only needs the relative position of the next frame to do the reconstruction from the sequence of images. In order to reduce the effect of possible odometry errors the algorithm uses a cubic filter. This filter F (explained below) is applied to the whole matrix M_{map}, which discretizes the three-dimensional matrix and transforms it into a grid of rectangular cubes. Lastly, the result (M_{map}) is represented according to the space occupation of this matrix and calculating its equivalence in real units (metres).

Cubic filter F applies the equation $g(x,y,z) := \Sigma_{(i,j,k)\in S} f(i,j,k)$ to each cube of the matrix, where S represents the set of point coordinates which are located in the neighbourhood of $g(x,y,z)$, including the point in question. In this way the

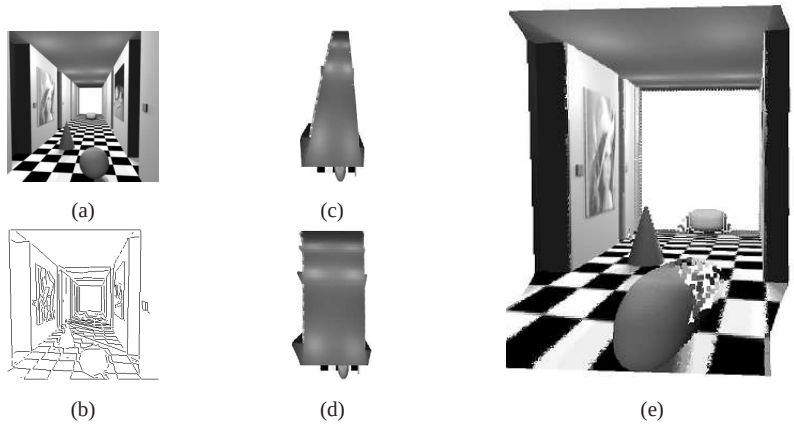


Figure 6.5: *Perspective rectification comparison using a corridor depth map.*

space occupation of each cube is in the centre, and each cell contains the set of readings of that portion of the space. The use of these cells instead of a unique sample let the system avoid possible odometry errors. The number of readings is referred to as "votes", and represents the probability of space occupation.

6.5 Experimentation and results

In this section the experiment results are shown. Figure 6.5 shows a reconstruction comparison using a synthetic disparity map (a) which simulates a corridor. This example clearly shows the effect of the perspective rectification. Figure (b) shows the segmentation used to calculate the vanishing point, which is estimated to be -4º. In figures (c), (d) and (e) the rectification effect is compared: (c) shows a non-rectified reconstruction (seen from above), and (d) and (e) show a top and an oblique view of the correct result after the rectification. As it can be seen, the walls are perfectly rectified, becoming parallel as expected.

To do the mapping experimentation two sequences of 30 images obtained from two different corridors have been used. All the images were taken using the Pioneer P3-AT robot and the Bumblebee HICOL-60 stereo camera (figure 6.6(a)). Figure 6.6(b) shows four images of one of these sequences as well as their depth maps. The main objective is that the walls, floor and roof appear without slope in the reconstruction, and that there should not be any obstacle

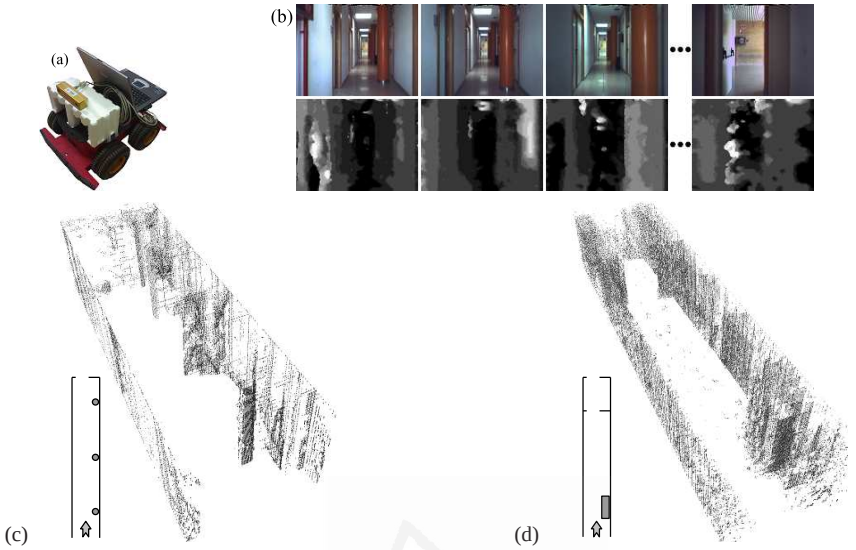


Figure 6.6: *(a) Pioneer Robot P3-AT with stereo camera. (b) Sequence of images for the mapping. (c,d) Mapping results of the corridors.*

(noise) in the corridor. It is also important that the columns (represented by circles in the plan (c)) and the coffee machine (represented by a rectangle in the plan (d)) are detected correctly. Figures (c) and (d) show the mapping results of the two sequences and their respective corridor maps. For these examples a cubic filter size of $3 \times 3 \times 3$ and a number of votes of 5 have been used. These results show a good definition of the corridors because the walls are limited and the in-between area can be seen. Moreover, the columns and the coffee machine can be distinguished on the right hand side of each one of the results.

To conduct the experiments, a Pentium IV 3,20GHz with 2GB of RAM and a 512MB graphic card have been used. The reconstruction of the maps have been made using a $320 \times 240 \times 256$ voxels matrix and depth maps with a size of 320×240 pixels. Moreover, it is important to note that only non-null pixels (finite depth) in the depth map are processed. The computational cost linearly depends on the size of the input images and on the precision of the reconstruction. So the algorithm obtains a good performance: To process just one sequence of 30 images (each image has a level of 70% of processed data) the algorithm takes approximately 9 seconds. This time depends on the precision of the final 3D

reconstruction. So, the process time of an individual reconstruction is less than 0.3 seconds.

6.6 Conclusions

A new method to reconstruct 3D scenes from stereo images has been presented, as well as an algorithm for environment mapping. These methods use geometrical rectification to eliminate the effect of conical perspective. Due to the fact that the vanishing point position is calculated, rectification can remove the camera orientation and obtain a front view of the scene. It is important to notice that the final quality of the reconstructed image depends on the quality of the depth map. In future experiments, better depth images will probably improve the final result. Moreover, other reconstruction algorithms with different geometric primitives will be implemented to be able to compare the results.

The results show how this process corrects the perspective effect and how it helps to improve the matching in the mapping algorithm. An advantage of this method is that it is nor correspondence dependent. In addition, it could probably be used for real-time applications due to the low computational burden and to the good performance. The cubic filter is very useful to solve odometry problems in the mapping. This is a statistical approach used to compute the matrix of the occupancy evidence. The mapping algorithm will be improved in future experiments in order to consider moving objects, robot drift and other kind of problems.

The main interest of the method is that some crucial information from the scene, such as object geometry, volume and depth, is retrieved. For these reasons the proposed methods are useful in a wide range of applications, such as Augmented Reality (AR) and autonomous robot navigation. As future work we want to use the obtained results in this type of applications.

6.7 Acknowledgments

This work has been done with the support of the Spanish Generalitat Valenciana, Project GV06/158.

Bibliography

[1] Pérez, J., Castellanos, J., Montiel, J., Neira, J., Tardós, J.: Continuous mobile robot localization: Vision vs. laser. ICRA (1999)

[2] Hans P. Moravec: Robot spatial perception by stereoscopic vision and 3D evidence grids. The Robotics Institute Carnegie Mellon University. Pittsburgh, PA (1996)

[3] Stephen Se, D. Lowe, J. Little: Vision-based mobile robot localization and mapping using scale-invariant features. ICRA (2001)

[4] C. Martin and S. Thrun: Real-time acquisition of compact volumetric maps with mobile robots. ICRA (2002)

[5] Gonzalo Pajares Martinsanz, Jesús M. de la Cruz García: Visión por computador: imágenes digitales y aplicaciones. Ed. Ra-Ma, D.L. Madrid (2001)

[6] G. Vogiatzis, P.H.S. Torr and R. Cipolla: Multi-view stereo via Volumetric Graph-cuts. CVPR (2005) 391–398.

[7] S. Sinha, M. Pollefeys: Multi-view Reconstruction using Photo-consistency and Exact Silhouette Constraints: A Maximum-Flow Formulation. ICCV (2005).

[8] Alberto Broggi: Robust Real-Time Lane and Road Detection in Critical Shadow Conditions. In Proceedings IEEE International Symposium on Computer Vision, Coral Gables, Florida. IEEE Computer Society (1995)

[9] Compañ, P.; Satorre, R.; Rizo, R.: Disparity estimation in stereoscopic vision by simulated annealing. Artificial Intelligence research and development. IOS Press. (2003) 160–167

[10] E. Trucco and A. Verri: Introductory techniques for 3-D Computer Vision. Prentice Hall (1998)

[11] I. Cox, S. Ignoran, and S. Rao: A maximum lilelihood stereo algorithm. Computer Vision and Image Understanding, 63 (1996)

[12] Oliver Faugeras: Three-dimensional computer vision: a geometric viewpoint. The MIT Press. Cambridge, Massachusetts (1993)

[13] A.J. Gallego Sánchez, R. Molina Carmona, C. Villagrá Arnedo: Three-Dimensional Mapping from Stereo Images with Geometrical Rectification. AMDO, LNCS 4069, Mallorca, Spain (2006) 213–222

[14] J. Coughlan and A.L. Yuille: Manhattan World: Orientation and Outlier Detection by Bayesian Inference. Neural Computation. Vol. 15, No. 5 (2003) 1063–88

Publicación 7

3D Reconstruction and Mapping from Stereo Pairs with Geometrical Rectification

Antonio Javier Gallego, Rafael Molina, Patricia Compañ
and Carlos Villagrá

Grupo de Informática Industrial e Inteligencia Artificial
Universidad de Alicante, Ap.99, E-03080, Alicante, Spain

{*ajgallego, rmolina, company, villagra*}*@dccia.ua.es*

Lecture Notes in Computer Science, Springer-Verlag, LNCS 4729 (2007), pp. 318–327. ISSN: 0302-9743. ISBN:978-3-540-75554-8.

3D Reconstruction and Mapping from Stereo Pairs with Geometrical Rectification

Antonio Javier Gallego, Rafael Molina, Patricia Compañ
and Carlos Villagrá

Grupo de Informática Industrial e Inteligencia Artificial
Universidad de Alicante, Ap.99, E-03080, Alicante, Spain
{*ajgallego, rmolina, company, villagra*}*@dccia.ua.es*

Abstract

In this paper a new method for reconstructing 3D scenes from stereo images is presented, as well as an algorithm for environment mapping, as an application of the previous method. In the reconstruction process a geometrical rectification filter is used to remove the conical perspective of the images. It is essential to recover the geometry of the scene (with real data of depth and volume) and to achieve a realistic appearance in 3D reconstructions. It also uses sub-pixel precision to solve the lack of information for distant objects. Finally, the method is applied to a mapping algorithm in order to show its usefulness.

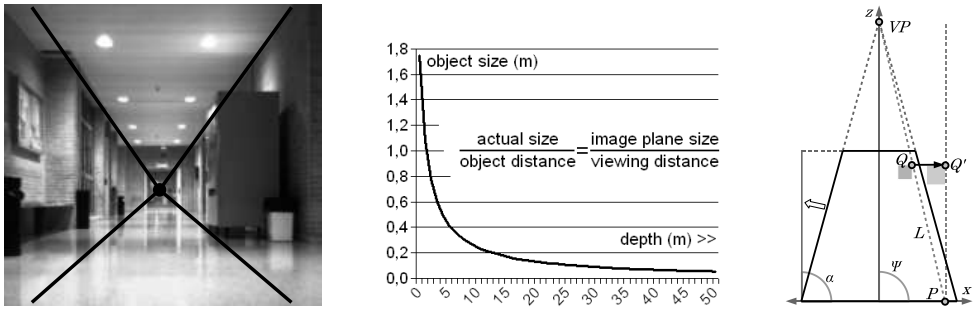
7.1 Introduction

Unknown environments reconstruction is a fundamental requirement in several fields of research. Stereoscopic vision opens new paths that in the future will allow to capture the three-dimensional structure of the environment, and take advantage of this to calculate the geometry, volume and depth of the objects in the scene. Range sensors can also acquire very detailed models [1], but these types of sensors are more expensive and they cannot provide information of both range and appearance, which is useful for reconstruction and texture mapping. For these reasons we will focus on stereo vision.

Several authors use stereo vision and disparity images to solve the 3D reconstruction or mapping problems. For instance, a first solution to three-dimensional reconstruction with stereo technology explores the possibility of composing several 3D views from the camera transforms [2]. There are other approaches which infer 3D grids from stereo vision, due to the fact that appearance information is not provided by range finders. Hence, they add an additional camera to their mobile robots [3, 4]. Moreover, a module of 3D recognition could be added to identify some objects. This technique is not exclusive of robotics, but it could be used in other applications such as automatic machine guidance or also for detection and estimation of vehicle movement [5].

Stereo vision can improve the perception of scenes and world modelling, so there are some methods which work with disparity images due to their advantages. The problem is that these algorithms cannot be applied in a widespread manner with all types of structures; because the images (or the objects) obtained from a camera have no real size, since they are deformed by the conical perspective effect. In general, any image taken by a camera is deformed by this effect, so direct reconstruction generates scenes with unreal aspect. There are very few works which focus on creating a good reconstruction and on obtaining a real appearance of the scene. However, some interesting works can be found [6, 7, 8], but none of them makes any type of geometrical rectification. Specific objects are reconstructed instead of the whole scene, so the real structure of the environment is not recovered. Some naïve rectifications have already been used in other fields, to rectify roads and to obtain their real appearance [9].

This work is centred in the reconstruction of the structure of the scene showing its real aspect, using the information provided by the stereo images and the disparity maps (in fact depth maps, their duals, are used). Our proposal does not make assumptions about the scene nor the object structure, it does not segment objects trying to identify known shapes, only a stereo pair is needed and it is not correspondence dependent. Perspective rectification allows the method to eliminate the conical perspective of the scene and to remove the camera orientation. This way the algorithm recovers the structure of the scene and some crucial information such us object geometry, volume and depth. Moreover, the reconstruction method is also extended to manage a sequence of stereo images to map a whole environment.



(a) (b) (c)

Figure 7.1: *(a) shows the effect of the conical perspective. (b) shows how the object size decreases as the depth is increased, due to the perspective's effect. (c) shows the rectification scheme on a non-rectified scene seen from above (x-z plane).*

7.2 Geometrical Rectification: Recovery of the Real Perspective

Perspective effect arises from the common appearance of the real world which surrounds us. This effect deforms the size and geometry of the space and the objects contained in it in order to create the depth effect. Figure 7.1(b) shows how the conical perspective effect changes the size with which the objects are represented according to their distance from the view point. Rectification is used for correcting this effect and recovering the real scene geometry. As an example, figure 7.1(a) shows an image of a corridor, in which, a pixel in the lower part of the image represents a small volume of the scene (it represents a part of the scene in the foreground); while a pixel in the centre of the same image represents a larger volume (because the part of the scene represented by the pixel is in the background). If the scene is directly reconstructed, the perspective is preserved. So, to correctly perform the 3D reconstruction the perspective must be rectified making a correction to the pixel's coordinates. This way the obtained result will show the same aspect as the real scene.

Rectification is performed on values of a depth map D calculated from a pair of stereo images. In principle the depth map can be obtained by any method, but in this approach the depth image is computed using multi-resolution and energy function [10]. Each value $D(x,y)$ of the depth map contains the depth associated to the pixel (x,y) of the reference image (left image) [11, 12].

Figure 7.1(c) shows the scheme of the process. On the left hand side it illustrates the desired result of the left wall rectification, which is rotated α° to recover its real inclination. And on the right hand side, it shows the point Q (current pixel being processed obtained from the input depth map) with coordinates $(x,y,D(x,y))$, which is rectified to obtain Q'. This pixel is a part of an object which is rectified to recover its real size. The first step to rectify Q is to obtain the line L, which links the points VP and Q. Next, the intersection of the line L with the *x*-*y* plane is calculated, obtaining in this way the point P. Starting from P, the line L is rotated to be perpendicular to the *x*-*y* plane. This way the coordinates (x,y) of Q' can be obtained. The calculation of the coordinate $z_{Q'}$ is shown in the section 7.2.1. In short, the equation 7.1 shows the calculation of the new rectified point $Q' = (x',y',z')$. Rectification is referred as π, where the new coordinates of Q are obtained as $Q' = \pi(Q)$.

$$\pi(Q) = \begin{cases} x_{Q'} &= x_{VP} + z_{VP}\frac{x_{VP}-x_Q}{z_Q - z_{VP}} \\ y_{Q'} &= y_{VP} + z_{VP}\frac{y_{VP}-y_Q}{z_Q - z_{VP}} \\ z_{Q'} &= f\frac{T}{d\lambda}\rho \end{cases} \tag{7.1}$$

As can be seen in the equation 7.1, the vanishing point position has to be obtained. In general, the central point can be used as the vanishing point of the scene, obtaining a reconstruction that maintains the original angle of the camera. If the camera view is oblique, the real position of the vanishing point can be calculated. This way, the camera orientation is corrected and a frontal view is obtained after the reconstruction. When the perspective cannot be calculated (in non Manhattan Worlds), the central point is taken by defect.

For the depth of the real vanishing point VP the maximum depth value (D_{max}) of the whole depth map is used. The coordinates x and y of VP are calculated using the method proposed in [13]. It uses a Bayesian model which combines knowledge of the 3D geometry of world with statistical knowledge of edges in images. The method returns an angle (called as Ψ) which defines de orientation of the camera in direction $\cos\Psi\boldsymbol{i} - \sin\Psi\boldsymbol{j}$. Cartesian coordinates (x,y,z) of VP can be obtained from the following camera coordinates $\boldsymbol{u} = (u,v)$:

$$u = \frac{f\cdot(-x_{VP}\sin\Psi - y_{VP}\cos\Psi)}{x_{VP}\cos\Psi - y_{VP}\sin\Psi}, \quad v = \frac{f\cdot z_{VP}}{x_{VP}\cos\Psi - y_{VP}\sin\Psi} \tag{7.2}$$

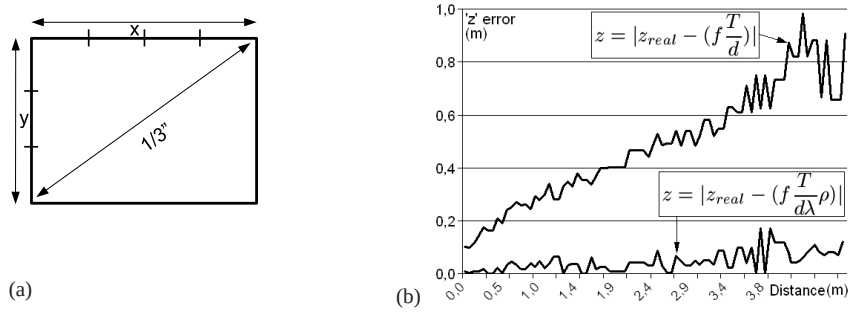
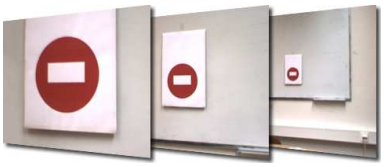


Figure 7.2: *(a) Camera lens proportions. (b) Comparison of the error made in the equations for the depth calculation.*

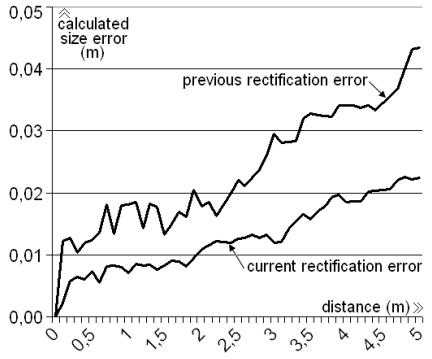
7.2.1 Calculating the Error in the Rectification

Coordinate $z_{Q'}$ is obtained using a modified version of the equation $z = f\frac{T}{d}$ [14]. Where d is the disparity ($D(x,y)$) for this pixel, T is the length of the base line, f is the focal distance. The main problem is that disparity d is expressed in pixels, while the other parameters are expressed in metres. So, a conversion factor is used $\lambda = CCD\ width\ in\ meters\ /\ Image\ width\ in\ pixels$ to convert pixels into metres. To calculate the CCD width, its dimensions (1/3") and its proportions ($\frac{x}{y} = \frac{4}{3}$) are needed (figure 7.2 (a)). Therefore, the value of x (total width in metres) can be isolated from the equation $x^2 + y^2 = (\frac{1}{3}")^2$. There is also an error that appears when the obtained distance is compared with the real one (figure 7.2(b)). This error is a small linear deviation due to the lens concavity and it is corrected adding a correction factor ρ to the equation (obtained empirically).

Figure 7.3(b) shows the error in the objects size made when they are represented after de geometrical rectification. To obtain the error, a sequence of images of the same object (the sign in image 7.3(a)) were taken at different distances. A comparison between the representation size and the real size were done. The representation size can be easily calculated due to the fact that the coordinates of each point are known. In figure 7.3(b) the error made by the previous approach to the geometrical rectification is also shown [15].



(a)



(b)

Figure 7.3: *(a) Sequence of images taken to calculate the error in the representation size. (b) Comparison of the error made in the representation as the depth increases.*

7.3 Applications of the Geometrical Rectification

The proposed method can be useful in a wide range of applications, because some crucial information from the scene is retrieved, such as object geometry, volume and depth. For example, it could be applied in Augmented Reality (AR) systems to solve some problems related to this discipline, as well as to set out new applications and improvements.

Next, two possible applications are presented. The first one is the 3D reconstruction of scenes using sub-pixel precision and stereo images. The second one is the map building from a sequence of stereo images. They both demonstrate the utility of the geometrical rectification and the advantages of its application to this kind of problems. For this reason simple methods are used. In the conclusions section more applications of these methods will be proposed.

7.4 Reconstruction using Sub-Pixel Precision

In the first step to do the 3D reconstruction of a scene, the depth map D is calculated from a pair of stereo images (LI and RI). The depth map D is obtained using the methods proposed in [10]. Based on D, the geometrical rectification process is applied in order to remove the effect of the conical perspective and recover the real structure of the scene (section 7.2). The result of this stage

is a rectified matrix of voxels R, which is used in the reconstruction to represent the space occupation. R is initialized to zero and, then, it is filled as follows: $R(x',y',D(x,y)) = 1$ where (x',y') are the rectified coordinates of (x,y), $\forall x,y \in \mathbb{R}/\{0 \leq x \leq m-1, 0 \leq y \leq n-1\}$. The final step is obtaining the real units. Equation 7.1 shows how to obtain the equivalence in metres from a disparity value and the rectified coordinates.

The most important drawback is the fact that when the geometrical rectification corrects the pixels' coordinates, the voxels are separated in the 3D representation (see figure 7.4). This is due to the discreteness of depth maps. In fact, pixels corresponding to a distant object are split, leaving a hole whose dimensions increase as the distance to the object increases. To minimize these problems a sub-pixel precision technique is proposed to calculate the position of n fictitious pixels between two consecutive pixels. The precision used for the reconstruction is calculated using the equation $1-(z/D_{max})$, which returns the minimum value when the pixel is in the foreground and the maximum one when it is in the background. All the steps of the sub-pixel reconstruction method can be summarized as follows:

1. $D := CalculateDepthMap(LI, RI)$
2. **while (** $x \leq m-1$ **)**
   (a) **while (** $y \leq n-1$ **)**
      i. $(x',y') := \pi(x,y)$
      ii. $R(x',y',D(\lfloor x \rfloor, \lfloor y \rfloor)) = 1$
      iii. $y = y + 1 - (D(\lfloor x \rfloor, \lfloor y \rfloor)/D_{max})$
   (b) $x = x + 1 - (D(\lfloor x \rfloor, \lfloor y \rfloor)/D_{max})$
3. $Display(R)$

7.4.1 Reconstruction Experimentation

In figure 7.4 two examples of reconstruction using sub-pixel precision are shown. The images on the left show the mesh used during the reconstruction, these images are obtained without and with sub-pixel precision respectively. On the right side, there is another example of reconstruction using voxels. The first image shows the reconstruction without using sub-pixel precision, in which the voxels

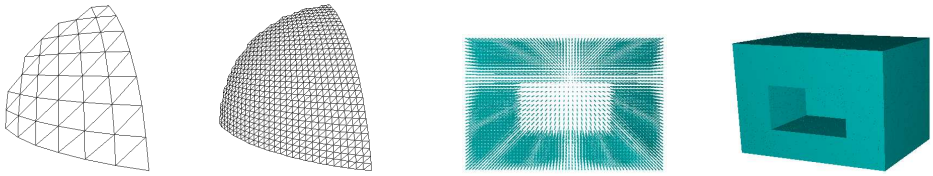


Figure 7.4: *Examples of reconstruction using sub-pixel precision.*

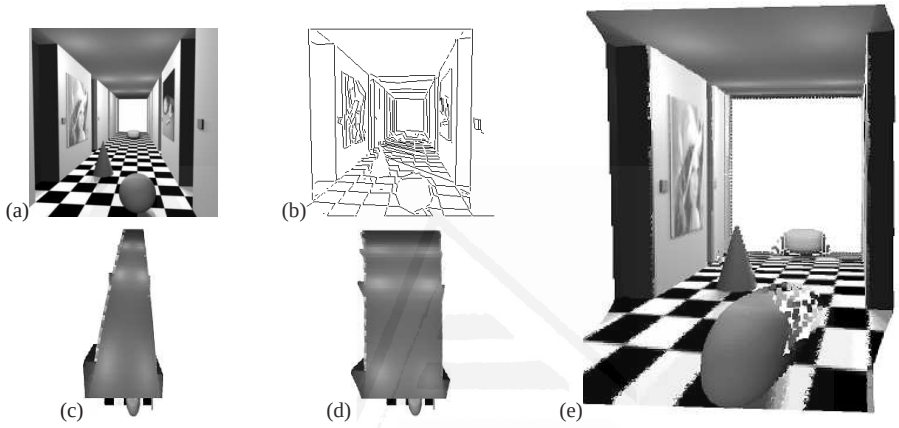


Figure 7.5: *Geometrical rectification comparison using a corridor depth map.*

are separated due to the geometrical rectification. In the second image the sub-pixel precision has been used. The result has a more realistic appearance because holes have been filled.

Figure 7.5 shows a 3D reconstruction comparison using a synthetic depth map (a) which simulates a corridor. This example clearly shows the effect of the geometrical rectification. Figure (b) shows the segmentation used to calculate the vanishing point, which is estimated to be $-4°$. The camera deviation is very small as it can be observed, so the real orientation could have been used avoiding so the VP calculation. In figures (c), (d) and (e) the rectification effect is compared: (c) shows a non-rectified reconstruction (seen from above), and (d) and (e) show a top and an oblique view of the correct result after the rectification. As it can be seen, the walls are perfectly rectified, becoming parallel as expected.

7.5 Mapping Algorithm

In order to do the 3D mapping of the scene, N stereo pairs (LI_0, RI_0), (LI_1, RI_1), ..., (LI_{N-1}, RI_{N-1}) of the environment are taken. Each of these images is captured at a fixed distance. Once a stereo pair (LI_k, RI_k) is obtained, its corresponding depth map D_k is calculated and added to the Σ list which stores all the depth maps. Next, the algorithm of geometrical rectification is used in order to compute the rectified matrix R_k of each depth map. For each matrix R_k its intersection with the previous matrix is calculated ($R_{k-1} \cap R_k$), and its result is added to the main matrix M_{map} which represents the mapping of the scene. In this approach the position of the frames is obtained from robot odometry. The system only needs the relative position of the next frame to do the reconstruction from the sequence of images. In order to reduce the effect of possible odometry errors the algorithm uses a cubic filter. This filter F (explained below) is applied to the whole matrix M_{map}, which discretizes the three-dimensional matrix and transforms it into a grid of rectangular cubes. Lastly, the result (M_{map}) is represented according to the space occupation of this matrix and calculating its equivalence in real units (metres).

1. **for each** $D_k \in \Sigma$ **do**
   (a) $D_k := CalculateDepthMap(LI_k, RI_k)$
   (b) $R_k := ApplyRectification(D_k)$
   (c) $M_{map} := M_{map} \cap R_k$
2. $M_{map} := ApplyCubicFilter(M_{map})$
3. $Display(M_{map})$

Cubic filter F applies the equation $g(x,y,z) := \Sigma_{(i,j,k)\in S} f(i,j,k)$ to each cube of the matrix, where S represents the set of point coordinates which are located in the neighbourhood of $g(x,y,z)$, including the point in question. In this way the space occupation of each cube is in the centre, and each cell contains the set of readings of that portion of the space. The use of these cells instead of a unique sample let the system reduce the effect of possible odometry errors. The number of readings is referred to as "votes", and represents the probability of space occupation.

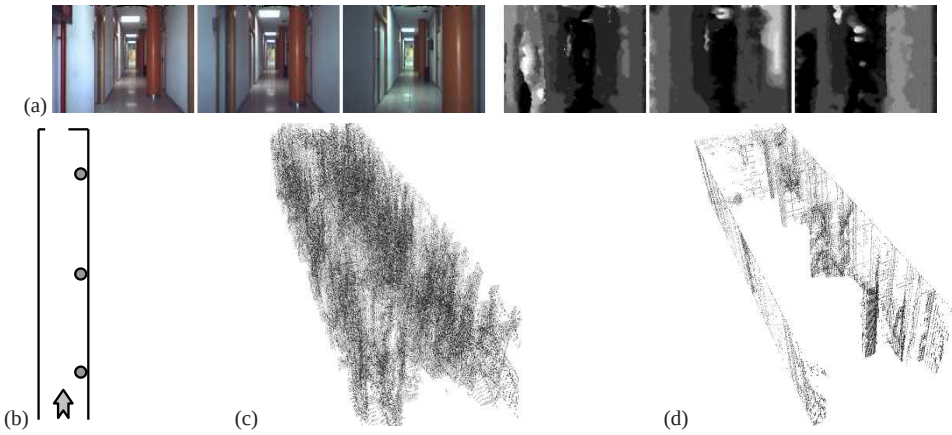


Figure 7.6: *(a) Sequence of images for the mapping. (b) Corridor sketch. (c, d) Mapping results of the corridor.*

7.5.1 Mapping Experimentation

To do the mapping experimentation two sequences of 30 images obtained from two different corridors have been used (Figures 7.6 and 7.7). Figures 7.6(a) and 7.7(a) show the first three images of both sequences as well as their depth maps. The main objective is that the walls, floor and roof appear without slope in the reconstruction, and that there should not be any obstacle (noise) in the corridor. It is also important that the columns (represented by circles in the plan (Fig. 7.6(b))) and the coffee machine (represented by a rectangle in the plan (Fig. 7.7(b))) are detected correctly. In figures 7.6(c, d) and 7.7(c, d) the results are shown. In 7.6(c) and 7.7(c) there is no perspective rectification, consequently a wrong result is obtained: the in-between space of the corridors is not clear. In 7.6(d) and 7.7(d) the rectification has been applied. These results show a good definition of the corridors because the walls are limited and the in-between area can be seen. Moreover, the columns and the coffee machine can be distinguished on the right hand side of each one of the results. For these examples a cubic filter size of $3 \times 3 \times 3$ and a number of votes of 5 have been used.

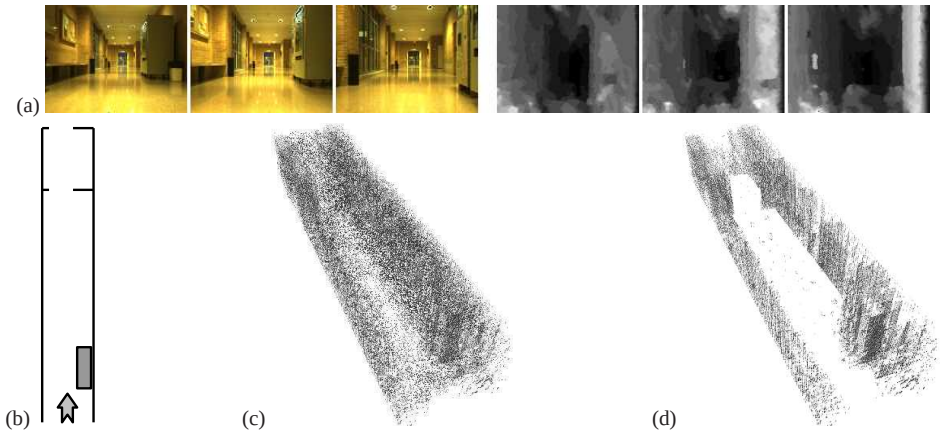


Figure 7.7: *(a) Sequence of images for the mapping. (b) Corridor sketch. (c, d) Mapping results of the corridor.*

7.6 Performance Results

To conduct the experiments, a Pentium IV 3,20GHz with 2GB of RAM and a 512MB graphic card have been used. The reconstruction of the maps have been made using a $320 \times 240 \times 256$ voxels matrix and depth maps with a size of 320×240 pixels. Moreover, it is important to note that only non-null pixels (finite depth) in the depth map are processed. The computational cost linearly depends on the size of the input images and on the precision of the reconstruction. So the algorithm obtains a good performance: To process just one sequence of 30 images (each image has a level of 70% of processed data) the algorithm takes approximately 9 seconds. The process time of an individual reconstruction is less than 0.3 seconds.

7.7 Conclusions

A new method to reconstruct 3D scenes from stereo images has been presented, as well as an algorithm for environment mapping. This is an improvement of a previous reconstruction method for which a new perspective rectification method, and news algorithms for reconstruction and mapping with sub-pixel precision have been incorporated. These methods use the geometrical rectification

to eliminate the effect of conical perspective, with the intention of getting a real aspect in the final result. It also allows the retrieval of some crucial information from the scene, such as object geometry, volume and depth. Nevertheless, the final quality of the reconstructed image depends on the quality of the disparity map. In future experiments, better disparity images will improve the final result.

The results show how this process corrects the perspective effect and how it helps to improve the matching in the mapping algorithm. An advantage of this method is that it is not correspondence dependent. In addition, it could probably be used for real-time applications due to the low computational burden and to the good performance.

Current work is focused on applying the obtained results to an Augmented Reality system. So that it can take advantage from the geometry information in order to develop new applications (as new interfaces or games) and solve problems related to this discipline (visual tracking, hidden objects and alignment). This work will also be related with an autonomous robot system which uses the environment information to be able to identify specifical areas.

Acknowledgments

This work has been done with the support of the Spanish Generalitat Valenciana, Project GV06/158.

Bibliography

[1] Jose M. Sanchiz, Robert Burns Fisher: Viewpoint Estimation in Three-Dimensional Images Taken with Perspective Range Sensors. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, no. 11 (2000) 1324–1329

[2] Hans P. Moravec: Robot spatial perception by stereoscopic vision and 3D evidence grids. The Robotics Institute Carnegie Mellon University. Pittsburgh, PA (1996)

[3] Stephen Se, D. Lowe, J. Little: Vision-based mobile robot localization and mapping using scale-invariant features. ICRA (2001)

[4] C. Martin and S. Thrun: Real-time acquisition of compact volumetric maps with mobile robots. ICRA (2002)

[5] Gonzalo Pajares Martinsanz, Jess M. de la Cruz Garca: Visión por computador: imágenes digitales y aplicaciones. Ed. Ra-Ma, D.L. Madrid (2001)

[6] G. Vogiatzis, P.H.S. Torr and R. Cipolla: Multi-view stereo via Volumetric Graph-cuts. CVPR (2005) 391–398.

[7] L. Zhang, Steven M. Seitz: Parameter estimation for MRF stereo. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), San Diego, CA, June (2005).

[8] S. Sinha, M. Pollefeys: Multi-view Reconstruction using Photo-consistency and Exact Silhouette Constraints: A Maximum-Flow Formulation. ICCV (2005).

[9] Alberto Broggi: Robust Real-Time Lane and Road Detection in Critical Shadow Conditions. In Proceedings IEEE International Symposium on Computer Vision, Coral Gables, Florida. IEEE Computer Society (1995)

[10] Compa, P.; Satorre, R.; Rizo, R.: Disparity estimation in stereoscopic vision by simulated annealing. Artificial Intelligence research and development. IOS Press. (2003) 160–167

[11] E. Trucco and A. Verri: Introductory techniques for 3-D Computer Vision. Prentice Hall (1998)

[12] I. Cox, S. Ignoran, and S. Rao: A maximum lilelihood stereo algorithm. Computer Vision and Image Understanding, 63 (1996)

[13] J. Coughlan and A.L. Yuille: Manhattan World: Orientation and Outlier Detection by Bayesian Inference. Neural Computation. Vol. 15, No. 5 (2003) 1063–88

[14] Oliver Faugeras: Three-dimensional computer vision: a geometric viewpoint. The MIT Press. Cambridge, Massachusetts (1993)

[15] A.J. Gallego Sánchez, R. Molina Carmona, C. Villagrá Arnedo: Three-Dimensional Mapping from Stereo Images with Geometrical Rectification. AMDO. LNCS 4069, Mallorca, Spain (2006) 213–222

Publicación 8

Hacia un modelo integral para la generación de Mundos Virtuales

Gabriel López García, Rafael Molina Carmona y
Antonio Javier Gallego Sánchez

Grupo de Informática Industrial e Inteligencia Artificial
Universidad de Alicante, Ap.99, E-03080, Alicante, Spain

{*glopez, rmolina, ajgallego*}*@dccia.ua.es*

CEIG'08, Barcelona, Spain. Sept. 3–5 (2008). ISBN: 978–3–905673–69–2

Hacia un modelo integral para la generación de Mundos Virtuales

Gabriel López García, Rafael Molina Carmona y
Antonio Javier Gallego Sánchez

Grupo de Informática Industrial e Inteligencia Artificial
Universidad de Alicante, Ap.99, E-03080, Alicante, Spain
{*glopez, rmolina, ajgallego*}*@dccia.ua.es*

Resumen

Uno de los problemas más importantes en los sistemas de Realidad Virtual es la diversidad de los dispositivos visuales y de interacción que existen en la actualidad. Junto a esto, la heterogeneidad de los motores gráficos, los motores físicos y los módulos de Inteligencia Artificial, propicia que no exista un modelo que aúne todos estos aspectos de una forma integral y coherente. Con el objetivo de unificar toda esta diversidad, presentamos un modelo formal que afronta de forma integral el problema de la diversidad en los sistemas de RV, así como la definición de los módulos principales que los constituyen.

El modelo propuesto se basa en la definición de una gramática, que integra la actividad necesaria en un sistema de RV, su visualización y su interacción con el usuario. La descripción de un mundo se presenta como una secuencia ordenada de primitivas, transformaciones que modifican el comportamiento de las primitivas y actores que definen la actividad del sistema. Los conceptos de primitiva, transformación y actor son mucho más amplios de lo que es habitual en estos sistemas: Las primitivas no son simples primitivas de dibujo sino acciones que se deben ejecutar en un determinado sistema de presentación, gráfico o no; las transformaciones modifican las primitivas, dependiendo de su naturaleza; los actores desarrollan una o varias actividades en el mundo virtual, se visualizan mediante primitivas y transformaciones, y usan eventos que también se definen en sentido amplio.

El modelo presentado tiene como virtud la separación de la actividad del sistema de los dispositivos visuales y de interacción concretos que lo componen. Esto supone varias ventajas: los dispositivos pueden ser sustituidos por otros dispositivos o por simulaciones de estos, los elementos del

sistema pueden ser reutilizados, y la representación gráfica puede ser diferente dependiendo del dispositivo visual. En definitiva, se ha pretendido diseñar un sistema integral, adaptativo, reutilizable y genérico.

Por último se presenta un caso práctico que permite concretar cómo se utiliza el modelo propuesto.

8.1 Introducción

El desarrollo en las últimas décadas de los sistemas de Realidad Virtual (RV) y de los sistemas gráficos en general ha sido espectacular. Este progreso ha contribuido a que la información de nuestro alrededor se presente mediante gráficas, animaciones o simulaciones, dando a conocer nuevas formas de análisis. Los videojuegos constituyen hoy en día una de las industrias más pujantes y cada día nuevos desarrollos de Realidad Virtual aparecen a nuestro alrededor. Sin embargo, esto no significa que todo esté hecho ni mucho menos, sino que la situación actual hace que los retos que se presentan sean aún más estimulantes.

Uno de los problemas más acuciantes es la diversidad de los dispositivos visuales. Se tiene toda una variedad de dispositivos hardware que generan imágenes (monitores, teléfonos móviles, PDAs, gafas de realidad virtual) y la visualización debe adaptarse a cada uno de ellos según sus características particulares (resolución, tamaño, etc.).

Mientras que todo lo referente a la visualización ha alcanzado una cierta madurez en las pasadas décadas, en el campo de la interacción no se ha producido la evolución deseada [10, 2]. Sigue dominando el teclado y el ratón en los interfaces de usuario, los mandos con libertad de movimiento son complejos de manejar, y otras posibilidades de interacción se quedan en el mundo de la investigación y el arte gráfico [10]. Si que es cierto, que en los últimos años ha habido tímidos avances. Entre ellos se puede destacar el mando de la Wii.

Es necesario el desarrollo de algún tipo de modelo que unifique de alguna forma toda la diversidad de dispositivos para la interacción y la visualización. Por ejemplo, si se desea mover un objeto, sería interesante que la acción no se corresponda con una pulsación de ratón o un determinado botón de un mando, sino que toda esta diversidad de eventos se unifique en una sola acción, mover el objeto, y filtre todos los detalles concretos de los dispositivos de entrada.

En el desarrollo de un sistema de RV existen tres módulos importantes que han tenido desigual evolución. Estos tres módulos son: Motor Gráfico, Motor

Físico e Inteligencia Artificial. El primero se hace cargo de todo lo referente a la visualización de los datos en dispositivos visuales, el segundo tiene como objetivo simular todos los aspectos físicos que hacen que la acción que se desarrolla en la escena sea coherente y el tercero busca que los actores de la escena se comporten de una forma independiente mostrando en el mundo virtual capacidades inteligentes.

Las diferentes herramientas aportadas en la actualidad para el desarrollo de los tres módulos anteriores son muy variadas. Existen múltiples soluciones a diferentes problemas. Sin embargo, generalmente no se observa ninguna unificación de criterios a la hora de abordar los diferentes retos que tiene su desarrollo. Si que es cierto, por otra parte, que se vislumbran ciertos puntos en común a la hora de definir determinados diseños estructurales, aunque no se aprecia un estudio de cómo los tres módulos se relacionan entre sí.

En los próximos capítulos se presentará un modelo integral que unifica de forma sencilla y flexible la diversidad que existe en los sistemas de RV y en los tres módulos anteriormente descritos. Para ello se utilizará la definición de un lenguaje que integra la actividad necesaria, su visualización y su interacción con el usuario. En el apartado 8.2 se presentan algunos antecedentes de sistemas de RV. Los objetivos que pretendemos alcanzar centran el contenido del apartado 8.3. En el apartado 8.4 se define el modelo propuesto, mientras que el 8.5 presenta un caso práctico. Por último, las conclusiones y los trabajos futuros se exponen en el apartado 8.6.

8.2 Antecedentes

Un sistema de RV está formado por tres grandes módulos: un motor gráfico, un motor físico y un módulo de inteligencia artificial. A continuación se hace un breve repaso de los antecedentes referidos a estos módulos.

En el desarrollo de los componentes que se utilizan para los motores gráficos existen dos librerías: Direct3D [12] y OpenGL [13]. Ambas librerías se definen como una capa entre la aplicación y la tarjeta gráfica. Han aparecido sistemas que unifican las dos API (*Aplication Program Interfaces*) en un único interfaz. Este es el caso de OGRE [15], o de VTK [16].

Existen librerías que se utilizan para gestionar el hardware interactivo. Dos ejemplos son SDL [14] y DirectX [12]. En general, tienen las mismas características salvo que SDL es software libre y está implementada tanto para Win-

dows, como para Linux, mientras que la segunda no.

Hay una gran variedad de herramientas que implementan motores físicos Como ejemplos se tiene Working Model [3], que realizan simulaciones que se utilizan principalmente en el área educativa. Después existe Newton Game Dynamics [5] Además existe Physics Abstraction Layer (PAL) [11] que es una capa abstracta para el desarrollo de MTR siendo Open Dynamics Engine (ODE) [6] una implementación de parte de las especificaciones de PAL. Dentro del software propietario existen otras tantas API, destancando por un lado PhysX [1], cuyo propietario es NVidia y es utilizado en la PlayStation 3, y Havok [8], perteneciente a la compañía Havok y que está implementado para Windows, Xbox 360, Wii, PlayStation3 y PlayStation Portable, Mac OS X y Linux.

Los últimos tipos de herramientas son los que desarrollan el módulo de IA. No existen librerías que puedan ser utilizadas como soluciones genéricas, salvo en algún caso. Su número es escaso y cada sistema de IA está diseñado específicamente para una aplicación concreta. Uno de los motivos por los que los sistemas de IA se han desarrollado con mayor lentitud que el resto de sistemas podría ser el esfuerzo que se realiza por mejorar el aspecto gráfico de los juegos, y no tanto por la inteligencia de sus personajes [20]. Aún así, podemos especificar varios tipos de sistemas: Sistemas evolutivos y Sistemas basados en agentes.

Dentro de los sistemas evolutivos hay alguna herramienta que facilita el uso de dichos algoritmos. Podemos destacar EO Evolutionary Computation Framework [17] desarrollado en C++ y que implementa lo necesario para el desarrollo de algoritmos evolutivos. También existe CILib [18], entorno de desarrollo implementado en Java. Por el contrario, sí que se puede encontrar una gran variedad de artículos que describen el desarrollo de algoritmos genéticos y artículos que definen sistemas evolutivos para casos concretos [21, 22, 23].

En cuanto a los sistemas basados en agentes, se puede obtener una extensa literatura que detalla los aspectos de este tipo de elementos, como por ejemplo en [24, 25]. En algunos casos, estos agentes se denominan *bots*, sobre todo en el desarrollo de juegos en primera persona. Actualmente, este tipo de sistemas están muy extendidos en grupos de investigación para el desarrollo de IA, tanto para juegos como para simulaciones sociales y desarrollo de robots móviles [9, 26]. Existen varios entornos que nos simplifican la tarea para desarrollar *bots*, aunque estos son sistemas para juegos concretos: por ejemplo se tiene QuakeBot para el juego Quake y FlexBot para el juego Half-Life [20, 27]. Existen librerías más genéricas para el desarrollo de Agentes, por ejemplo Jade [19], que modela

sistemas multiagentes basados en Java.

8.3 Objetivos

Los objetivos se orientan principalmente a conseguir un modelo que integre los diferentes módulos que forman parte de un sistema de RV y que tenga una total independencia de los dispositivos hardware, tanto visuales como de interacción, pudiendo, si fuera necesario, sustituir un dispositivo por otro, o por una simulación, sin afectar a los mecanismos internos del sistema. Para lograr esto se pretende:

1. Definir un motor gráfico que elimine la diversidad de los diferentes dispositivos visuales. Es decir, se desea que con una única descripción de la escena se procese la misma, de tal manera que se muestre en cualquier dispositivo gráfico y con un nivel de detalle acorde con las características gráficas del dispositivo.

2. Definir un motor físico que modele toda la actividad del sistema, adaptándose a los diferentes componentes hardware donde se va a ejecutar. Si se dispone de componentes hardware que implementan algoritmos físicos, el sistema debe aprovecharlos, pero si por el contrario no existen, los debe implementar mediante software.

3. El motor de IA se debe integrar con el motor físico considerando las limitaciones impuestas por este. Esto supone que se debe realizar un trabajo importante en la integración de un motor sobre el otro.

4. Se pretende que la interacción con el sistema no dependa del hardware de entrada, si no que se abstraiga del origen de la interacción, y procese directamente las órdenes del usuario.

5. La reutilización de los diferentes elementos de forma casi inmediata. Si, por ejemplo, se diseña un elemento para un determinado mundo virtual, utilizando los mecanismos proporcionados por el sistema, se podrá reutilizar dicho componente en cualquier otro mundo virtual o aplicación que gestione un sistema de RV.

Para la realización de todos estos objetivos se utilizarán modelos matemáticos que formalizarán los diferentes componentes del sistema, abstrayendo las características que definen los tres módulos importantes de un sistema de RV.

8.4 Modelo para la generación de Mundos Virtuales

Un mundo virtual se caracteriza por un conjunto de actores o elementos, con una descripción geométrica y que realizan una actividad en un escenario. La actividad puede o no modificar objetos que componen el escenario, así como el aspecto del actor o de otros actores.

La descripción de un mundo se puede considerar como una secuencia ordenada de primitivas, transformaciones que modifican el comportamiento de las primitivas y actores que definen la actividad dentro del sistema. El concepto de primitiva se debe considerar, no como una primitiva de dibujo tal como una esfera, un cubo, etc. sino más bien como una acción que se debe ejecutar en un determinado sistema de visualización. Por ejemplo, dentro del concepto de primitiva cabe considerar la ejecución de un sonido. Los actores, por su parte, serán los componentes que desarrollen una o varias actividades en el mundo virtual y que se visualizarán mediante primitivas y transformaciones. Para modelar las diferentes actividades de los actores se va a usar el concepto de evento, eso sí, considerando un evento, no como una acción generada por un dispositivo de entrada, sino más bien como una acción que representa la activación de una determinada actividad que puede ser ejecutada por uno o varios actores.

A cada elemento que compone una escena se le puede asignar un símbolo, formando un conjunto de símbolos. Con este conjunto se puede construir diferentes cadenas que describen una escena. Estas cadenas deben construirse con una sintaxis definiendo un lenguaje. Habitualmente una sintaxis se presenta como una gramática [4], en adelante denotada por M, que queda definida por la tupla $M = < \Sigma, N, P, S >$. Con la gramática M se determinará el lenguaje $L(M)$. La definición de la tupla gramatical se expresa como:

Sea P el conjunto de símbolos que define un conjunto de primitivas.

Sea T el conjunto de símbolos que define las transformaciones.

Sea O = $\{ \cdot ( ) \}$ *el conjunto de símbolos de separadores y operaciones.*

Sea D el conjunto de tipos de eventos generados para el sistema.

Sea A^D *el conjunto de símbolos que representan actores que van a activarse*

para los eventos enumerados en el superíndice. Por ejemplo, el actor a^d será ejecutado cuando se produzca un evento e^d.

Sea $\Sigma = P \cup T \cup O \cup A^D$ el conjunto de símbolos terminales.

Sea $N = \{$mundo, variosObjetos, objeto, actor, transformación, figura$\}$ el conjunto de símbolos no terminales.

Sea $S = \{$mundo$\}$ el símbolo inicial de la gramática.

Definimos las reglas gramaticales P como:

mundo :- variosObjetos
variosObjetos :- objeto | objeto · variosObjetos
objeto :- figura | transformación | actor
actor :- a^d(variosObjetos), $a^d \in M^D, d \in D$
transformacion :- t(variosObjetos), $t \in T$
figura :- p, $p \in P$

La gramática M es una gramática independiente del contexto, o de tipo 2, y por tanto se asegura que existe un procedimiento que verifica si una escena está bien descrita o no, o dicho de otro modo, que la cadena pertenece o no al lenguaje $L(M)$. Es decir, se podrá determinar que, por ejemplo, la cadena $a_1^d(p_1 \cdot p_2) \cdot p_3 \cdot t_1(p_1 \cdot p_2) \in L(M)$ donde $p_i \in P$, $t_i \in T$, $a_i^d \in A^D, d \in D$, pero que la cadena $a_1^d \cdot p_1 \notin L(M)$.

Sin embargo, se necesita saber cuál va a ser la funcionalidad de las diferentes cadenas. Se puede considerar que la semántica de una lenguaje explica esta funcionalidad. La semántica tiene varias formas de definición: operacional, denotacional y axiomática. En el caso presente se va a utilizar el método denotacional que describe funciones matemáticas para cada una de las cadenas.

La semántica del actor se describe como la ejecución del mismo cuando recibe o no un evento que debe manejar. Esta ejecución se representará como una función que define la evolución del actor a través del tiempo. Esto significa que va a cambiar la cadena que describe su actual estado. La función que define el comportamiento de un actor para un evento producido se denomina *función evolutiva del actor* y se denotará con λ. Su expresión viene dada por:

$$\lambda(a^d(v), e^f) = \left\{ \begin{array}{ll} u \in L(M) & Si \quad f = d \\ a^d(v) & Si \quad f \neq d \end{array} \right\} \tag{8.1}$$

El resultado de la función λ puede contener o no al propio actor o puede generar otro evento para la siguiente etapa. Si se desea acumular un evento para la

siguiente etapa se utilizará la notación Δe^f. Esto permitirá que los actores puedan generar eventos, y pasen a la siguiente etapa acciones que puedan recoger otros actores implementando un proceso de retroalimentación del sistema. En el caso de que el evento no coincida con el evento activador entonces la función devuelve el mismo actor. Más adelante se verá que las funciones λ pueden clasificarse según las cadenas que se definan en su evolución. Sobre todo, van a ser esenciales para la visualización del sistemas determinadas funciones λ, que solo devuelven cadenas con primitivas y transformaciones.

Con la función λ se puede definir un algoritmo que, dado un conjunto de eventos y una cadena, describe como el sistema evoluciona. A esta función se la llamará función de evolución α del sistema. Para su definición se necesita un conjunto de eventos $e^i, e^j, e^k, ..., e^n$ que abreviando se denotará como e^v siendo $v \in D^+$. La función se define como:

$$\alpha(w, e^v) = \left\{ \begin{array}{cc} w & Si \;\; w \in P \\ t(\alpha(v, e^v)) & Si \;\; w = t(v) \\ \prod\limits_{\forall f \in v} (\lambda(a^d(\alpha(y, e^v)), e^f)) & Si \;\; w = a^d(y) \\ \alpha(s, e^v) \cdot \alpha(t, e^v) & Si \;\; w = s \cdot t \end{array} \right\} \tag{8.2}$$

El operador $\prod\limits_{\forall f \in v} (\lambda(a^d(x), e^v))$ realiza la concatenación de la cadenas generadas por λ. Se debe observar que para las cadenas que solo son transformaciones y primitivas, el sistema se queda como está y no forman parte de su evolución. Este tipo de cadenas van a ser importantes a la hora de optimizar el sistema, ya que solo ejecutaremos α para aquellas cadenas que sean actores.

Por otro lado, están las primitivas cuya semántica será definida mediante la función γ. Dado un símbolo del alfabeto P ejecuta una primitiva sobre lo que se denominará una geometría aplicada G. Por tanto, la función será una aplicación definida como:

$$\gamma : P \rightarrow G \tag{8.3}$$

Esto significa que dependiendo de la función γ, el resultado variará según la definición que se haga de G. Esta G se puede definir como las acciones que se ejecutan en una librería gráfica concreta, como OpenGL o Direct3D. Se relacionarán los símbolos de P con la ejecución de funciones de la librería y si, por ejemplo, $esfera \in P$ se podrá ejecutar la función que dibuja una esfera en OpenGL o Direct3D:

$$\gamma_{opengl}(esfera) = glutSolidSphere$$
$$\gamma_{direc3d}(esfera) = drawSphereDirect3D$$

Sin embargo, esta definición no solo se puede utilizar para librerías gráficas, sino que se puede ampliar a otras definiciones. Por ejemplo, se puede definir para el mismo alfabeto P otra función γ_{bounds} que calcule los límites de la figura que define una primitiva. Es decir, si '*boundSphere*' es una función que implementa el cálculo de los límites de una esfera, entonces:

$$\gamma_{bounds}(esfera) = boundSphere$$

Los ejemplos anteriores demuestran que la función γ aporta toda la abstracción necesaria para homogeneizar las diferentes implementaciones que existen para realizar la definición de un mundo concreto.

Sin embargo, solo se han tratado elementos geométricos o de dibujo. No hay ninguna restricción al respecto siempre y cuando exista la definición para esta primitiva en la función γ. Así, también se pueden considerar como primitivas la ejecución de un sonido, el reflejo de un material, el color de un personaje, el movimiento de un robot, la manipulación de una máquina, etc. Además, la definición de la función γ también describe sistemas que escriban diferentes formatos de ficheros (VRML, DWG, DXF, XML, etc.), transporten cadenas por la red, ejecuten instrucciones de una máquina que realiza esculturas 3D, etc.

Por último, la gramática define lo que son las transformaciones que modifican el comportamiento de las primitivas. Estas transformaciones tienen un ámbito de aplicación y deberán ser aplicadas a determinado conjunto de primitivas. Es decir, se desea que una transformación $t \in T$ se aplique sobre una subcadena w que se simboliza en el lenguaje como $t(w)$. Para definir el ámbito de aplicación de una trasformación se definen las siguientes funciones semánticas:

I: T $\rightarrow$ G (Inicia transformación)
F: T $\rightarrow$ G (Finaliza transformación)

Estas dos funciones tienen las mismas características que la función γ pero aplicadas al conjunto de transformaciones anteriormente descritas.

Existe una restricción sobre las tres últimas funciones, y es que no pueden ser aplicadas sobre actores. Es decir, no se puede ejecutar $\gamma(a_1^d(w))$, ni $I(a_1^d(w))$, ni $F(a_1^d(w))$. Se debe primero transformar el actor en una cadena de primitivas

y transformaciones, que será su representación gráfica. Por ejemplo, si se define un actor que describe la actividad de una nave espacial, se debe transformar el actor en una secuencia de primitivas y transformaciones que van a representar la imagen de la nave espacial.

Como se ha dicho con anterioridad, las funciones λ se pueden clasificar según la cadena de resultados que devuelvan. Uno de los tipos de funciones λ son aquellas que devuelven solo cadenas que sean primitivas y transformaciones. A estas funciones se les va a denominar función de aspecto τ y su expresión es:

$$\tau(a^d(v), e^f) = \begin{Bmatrix} z \in L(E) & Si \;\; f = d \\ \varepsilon & Si \;\; f \neq d \end{Bmatrix} \tag{8.4}$$

Se puede observar que existen pequeñas diferencias entre λ y τ. La primera es que τ devuelve cadenas que pertenecen a $L(E)$, siendo este el lenguaje de la gramática E que es igual que M pero eliminado los actores. Además si no coincide con el evento devuelve una cadena vacía. Esto quiere decir que para ese evento el actor no tiene representación en la vista. El tipo de evento es importante y no se corresponde con ningún dispositivo de entrada en concreto, si no más bien es un evento que se encargará de establecer los diferentes tipos de vista que tiene el sistema. Así, si se desea un sistema de dibujo que filtre determinados elementos para que queden invisibles, solo tienen que no reaccionar al evento. Estos eventos también sirven para decidir qué tipo de visualización se desea, dependiendo de la ventana o dispositivo de salida.

Al igual que con λ, se define un algoritmo para τ que dado una cadena $w \in L(M)$ y un conjunto de eventos e^v con $v \in V^+$ y $V \subset D$, siendo V los tipos de eventos para visualizar el sistema, se devuelva una cadena $v \in L(E)$. A esta función le denominamos función de visualización Ω y se define como:

$$\Omega(w, e^v) = \begin{Bmatrix} w & Si \;\; w \in P \\ t(\Omega(y, e^v)) & Si \;\; w = t(y) \\ \prod\limits_{\forall f \in v} \tau(a^d(\Omega(y, e^v)), e^f) & Si \;\; w = a^d(y) \\ \Omega(s, e^v) \cdot \Omega(t, e^v) & Si \;\; w = s \cdot t \end{Bmatrix} \tag{8.5}$$

Se puede comprobar que es igual que la función α salvo que sustituimos λ por τ y que los eventos que pueden ser pasados a la función solo pueden pertenecer a V. En el caso de que a la función Ω se le pase un evento que no pertenece a V lo único que producirá será una cadena vacía.

8.4.1 Actividad y Eventos

La actividad en un sistema de RV la representan los actores. Esta actividad suele realizarse entre actores del propio sistema y entre dispositivos de entrada y cadenas de la escena.

Toda actividad consiste en un proceso que se produce en un sistema como reacción a un determinado evento. Este evento puede ser producido por otra actividad o por un dispositivo de entrada. Como resultado de una actividad se puede producir una modificación de los estados de los actores que forman el sistema. Así, se puede ver que lo que define a una actividad es, por un lado, el proceso de ejecución de la actividad que realiza el actor, y por otro, el evento o el conjunto de eventos que van a activar dicha actividad.

El concepto de evento dentro de un actor es importante, ya que define cuándo se va a ejecutar la actividad independientemente del origen del evento. Esta independencia es necesaria para la generalización de los dispositivos de entrada, ya que independiza el dispositivo de la actividad que se debe procesar.

La generalización de los eventos facilita la realización de simulaciones. Por ejemplo, si se quiere hacer una simulación en la que un usuario mueve un determinado elemento, solo se deben generar tantos eventos de 'movimiento' como se desee, simulando la presencia del usuario. También, se puede simular la existencia de dispositivos. Por ejemplo, simular la existencia de un guante de realidad virtual con un ratón. Para ello, se debe generar aquellos eventos producidos por el guante de realidad virtual.

Ya se ha visto que la identificación del evento es uno de los atributos dentro de la actividad de un sistema de RV. Pero además, en un evento se pueden definir unos datos que caracterizan a una instancia del evento. Por ejemplo, un evento de movimiento puede contener los datos del movimiento. Por supuesto, pueden existir eventos que no tengan datos asociados.

No toda actividad tiene porqué ejecutarse cuando se envía un evento. Pueden existir determinados eventos que ejecutan la actividad si se cumplen ciertas condiciones, dependiendo del estado del actor. Esta comprobación va a estar definida en el evento y no en el objeto que tiene definida la actividad. Por tanto, se va a establecer una definición de evento:

Se define e_c^d como el evento de tipo 'd', que tiene como datos 'e' y que se ejecuta de forma opcional bajo la condición 'c'.

Se puede observar que el origen de la creación de dichos eventos puede ser cualquiera, y que no es importante el origen, sino más bien el tipo de evento que

se quiere enviar y su definición.

8.4.2 Dispositivos de entrada

Se desea establecer una independencia entre el sistema y los dispositivos de entrada. Por tanto, se necesita definir, para un conjunto de dispositivos de entrada, los eventos necesarios para hacer reaccionar al sistema. Por ejemplo, si tenemos un ratón y se quiere mover un personaje de un juego, no se tiene que definir para el personaje el evento de ratón, sino que el ratón va a definir un evento que genera el movimiento del personaje.

Para ello, se va a especificar una función que, dependiendo del dispositivo, va a generar los eventos que se necesitan para hacer reaccionar al sistema. A este elemento se va denominar *generador de eventos* y su definición es:

Se define como $G^D(t)$ al generador que crea los eventos del tipo d en el instante t, siendo $d \in D$ donde D es el conjunto de tipos de eventos que pueden ser generados.

En la definición anterior se debe destacar que los eventos se generan en un instante t. Esto se debe a motivos de sincronización. Existe además la posibilidad de que G no cree ningún evento.

Con esta definición se independiza los dispositivos de entrada del sistema de RV. Así, se puede crear un generador para un ratón, para un guante, para un teclado o para cualquier dispositivo de entrada y que puedan generar los mismos tipos de eventos. Lo interesante de la implementación del generador es que solo se tendrá en un sitio la parte dependiente del dispositivo de entrada.

La actual definición de generador de eventos no tiene porqué estar asociada a un dispositivo de entrada hardware, sino que también se pueden asociar a procesos software. Estos procesos software pueden ser de cualquier índole: de colisión entre elementos, de contabilización, de acumulación de información para que en los instantes siguientes se produzcan otros eventos, etc. En definitiva, cualquier proceso que interactúe con el sistema.

Existe el problema de que varios generadores pueden crear eventos del mismo tipo y con distintos datos. Para que esto no cree conflictos se va a establecer un orden de prioridad entre los distintos generadores, de forma que, dados dos generadores G_i y G_j, se creen dos eventos del mismo tipo, por lo que prevalecerá el evento creado por G_i sobre el de G_j. Por ejemplo, si existe un teclado, un ratón y un *joystick*, se puede establecer el siguiente orden de prioridad: 1º

Joystick, 2º Ratón, 3º Teclado. Estos procesos de redundancia se producen porque el sistema debe ser implementado para todos estos dispositivos. Si existen varios, solo prevalecerá uno de ellos. El orden de prioridad se puede establecer, por ejemplo, por orden de inmersión en el sistema o facilidad de uso. En el caso de que no exista uno de los dispositivos, actuaría el de siguiente prioridad.

El proceso para la obtención de los eventos producidos por todos los dispositivos de entrada y sus generadores asociados se define como:

Se define G^* como el conjunto de todos los generadores de eventos del sistema asociados a los dispositivos de entrada.

Se define $e(z,e^i)$ siendo $G_i(t) = e^i$, como:

$$e(z,e^i) = \begin{Bmatrix} z \cdot e^i & si \;\; e^i \notin z \\ z & si \;\; e^i \in z \end{Bmatrix}$$

La función $E(G^*,t)$ que recolecta todos los eventos de todos los generadores se define como:

$$E(G^*,t) = \begin{Bmatrix} e(z,G_i(t)) & z = E(G^* - G_i,t) \\ \varepsilon & si \;\; G^* = \emptyset \end{Bmatrix}$$

8.4.3 Algoritmo del sistema

Una vez definidos todos los elementos implicados en el modelo que gestionará un sistema de RV se puede establecer el algoritmo que ejecutará todo el sistema, tanto la evolución como la visualización del mismo, para cada instante *t* o frame. El algoritmo es:

Sea D = { Conjunto de todos los tipos de eventos posibles en el sistema}.

Sea V = { Conjunto de todos los tipos de eventos de dibujo} con $V \subset D$.

Sea G^* = { Todos los generadores de eventos que genera eventos de tipo D}

Sea g el dispostivivo de salida.

Sea e^* todos los eventos generados por el sistema.

Sea e^v todos los eventos de los dispositivos visuales. Estos eventos será una de las entradas de la función de visualización del sistema Φ.

Sea e^u todos los demás eventos que no son visuales. Estos eventos será la entrada de la función de evolución del sistema α.

Se define el algoritmo Generador de Mundos Virtuales como:

Paso 1. Sea $w = w_o$ donde w_o es la cadena inicial del sistema.

Paso 2. Sea $t = 0$.

Paso 3. $e^* = E(G^*, t)$
Paso 4. $e^v =$ *Evento de* e^* *donde* $v \in V^+$
Paso 5. $e^u = e^* - e^v$
Paso 6. $w_{sig} = \alpha(w, e^u)$
Paso 7. $v = \Omega(w, e^v)$
Paso 8.$g = \Phi(v)$
Paso 9. $w = w_{sig}$; $t = t + 1$
Paso 10. Si $w = \varepsilon$ *entonces ir a Paso 12.*
Paso 11. Ir a paso 3.
Paso 12. Fin.

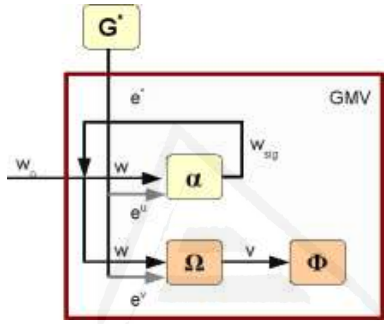


Figura 8.1: *Diagrama del Algoritmo Generador de Mundos Virtuales*

Los dos primeros pasos del algoritmo conforman la parte de la inicialización del sistema. Esto es, al sistema se le introduce el estado inicial de la escena w_o y el primer frame va a ser el 0.

Los pasos 3, 4 y 5 gestionan los eventos del sistema. Primero se llama a los generadores y se asigna a una lista e^* todos los eventos generados. En los pasos 4 y 5 se dividen los eventos en eventos para la visualización e^v y los demás eventos e^u. La primera será una de las entradas a la función Ω y la segunda a la función α.

En el paso 6, con la cadena actual del sistema w y los eventos que no son de visualización (e^u), se llama a la función de evolución α para calcular la cadena del siguiente frame o instante t.

En el paso 7 y 8 se realiza la visualización del sistema. En primer lugar se transforman los actores en primitivas y transformaciones llamando a la función Ω y con los eventos de visualización e^v. Una vez se tiene la cadena que representa

la visualización del sistema en este instante t se llama a la función $\Phi(v)$ para visualizar el sistema en el motor gráfico.

En el paso 9 se prepara para la siguiente iteración y se asigna w a w_{sig} y se aumenta en 1 el número de frames o instantes.

En el paso 10 se mira si se ha llegado a la condición de finalización, esto es, si la cadena siguiente es vacía y si es cierto se termina el algoritmo, paso 12. En caso contrario, se vuelve al paso 3 para el siguiente frame.

Se puede destacar que el paso 6 es intercambiable con los pasos 7 y 8 porque no comparten datos que tengan que modificar. Esta característica es importante para la implementación en paralelo del algoritmo. Efectivamente, si este algoritmo se desea paralelizar se puede poner en tareas diferentes el paso 6 por un lado, y los pasos 7 y 8 por otro.

También se puede observar que para que termine el algoritmo solo se debe devolver en α una cadena vacía. Esta cadena vacía se puede obtener mediante un evento especial que se genere cuando se desea terminar el sistema.

8.5 Caso práctico

Para exponer con mayor claridad cómo funciona y se define un sistema de RV con el modelo anteriormente definido, se va a describir un caso práctico consistente en la implementación de un modelo para la simulación de incendios forestales producidos por tormentas [9]. El modelo consiste en la representación de un mundo, donde cada cierto tiempo crece un árbol con una cierta probabilidad g. El crecimiento de un árbol hace que este se sitúe en un determinado lugar (i, j) de un tablero 2D. Por otro lado, con una probabilidad f puede caer un rayo en una determinada casilla (i, j). Si un relámpago cae en una casilla sin árbol, entonces no sucederá nada. Si por el contrario, hay un árbol, entonces esté arderá y quemará todos los árboles que estén a su alrededor, y estos a su vez quemarán los que estén al suyo, produciendo una reacción en cadena. Se puede ver un ejemplo de la implementación de este modelo en la figura 8.2.

Se debe resaltar que no se está interesado en el resultado propio del modelo anterior, sino más bien en definir un sistema de RV para este modelo con el sistema propuesto en el capítulo anterior.

Son cuatro los elementos principales que se deben definir. En primer lugar se definirán los eventos necesarios para que el sistema pueda ejecutar las actividades. En segundo lugar, se diseñarán los generadores de eventos. En tercer lugar,

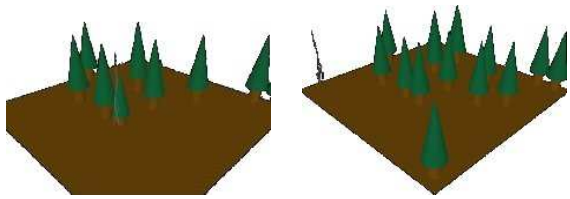


Figura 8.2: *Ejemplos de diferentes estados del tablero de juego*

se definirán los actores que forman parte de la escena. Por último, se introducirán las primitivas que podrán hacer visible las diferentes partes que se deseen mostrar.

Los eventos que se pueden producir en el sistema se muestran en la tabla 8.1.

Tipo de evento	*Significado*	*Datos asociados*
t	Evento que se genera cada cierto tiempo *t*	Incremento del tiempo con respecto al anterior evento
c	Crear un árbol en una determinada posición	Posición (i,j) donde se crea el árbol
r	Creación de un relámpago en una posición	Posición (i,j) donde se crea el relámpago
v	Elimina el árbol creado en una posición	Posición (i,j) donde se elimina el árbol
q	Quemar un árbol en una posición	Posición (i,j) donde se quema el árbol
d	Dibuja en OpenGL	Nada

Tabla 8.1: *Definición de eventos*

Los siguientes elementos que se deben definir son los generadores de eventos. Estos son tres:

$$G_{tiempo} = \{e^t \ \textit{cada cierto tiempo } t\}$$
$$G_{bosque} = \begin{Bmatrix} e^c & \textit{con probabilidad } g \\ e^r & \textit{con probabilidad } f \end{Bmatrix}$$
$$G_{opengl} = \{e^d \ \textit{cada redibujado}\}$$

Por supuesto, podríamos definir más generadores. Por ejemplo, cuando pulsamos un botón se podría generar un evento e^r y lanzar un relámpago en la posicion (i,j). Sin embargo, para este caso práctico no es necesario.

En la tabla 8.2 se definen cada cada una de las primitivas y las transformaciones.

Primitiva	*Dibujo*
A	Árbol.
A_q	Árbol quemado
R	Relámpago.
B_{NxN}	Tablero de NxN que representa el bosque
Transformaciones	*Transformación*
$T_{i,j}$	Traslación (i, j)
S_s	Escalado (s)

Tabla 8.2: *Definición de primitivas y transformaciones*

Las funciones γ, I, F son implementadas para Opengl y dibujadas en una ventana.

Por último, se especifican los actores que componen nuestro sistema. Para la definición de un actor se debe definir la función de evolución λ. La tabla 8.3 muestra los diferentes actores y su función de evolución.

Aclarar que en el caso del actor B^{crv} y el del evento e^v, solo cambiará el estado interno del actor al asignar la posición (i, j) del evento como vacía.

Ahora todos los actores deben tener una representación que se verá en el dispositivo visual. Para ello, se define la función τ que se muestra en la tabla 8.4.

Se debe aclarar que en el dibujado de AC, el escalado S va a corresponder a un escalado creciente dependiendo del estado actual del actor, que será modificado dependiendo del evento e^t. El efecto es el de un árbol que crece de menos a más. La escala para cada uno de los t vendrá definida por la expresión $s = \frac{t}{N}$, donde N es el número de frames totales de la animación.

En el caso de AQ, el escalado se va a aplicar en sentido inverso, representado por s^{-1}. Esto significa que el efecto es el de un árbol quemándose y menguando.

Por último decir, que el dibujado de un relámpago dependerá del instante t en que se encuentre R, y que poco a poco se dibujará un relámpago con un algoritmo dependiente de t.

Actor	*Descripción*	*Función* λ
B^{crv}	Representa el bosque	$\lambda(B^{crv}, e^f) = \begin{Bmatrix} AC^t \cdot B^{crv} & f = c \\ R^t \cdot B^{crv} & f = r \\ B^{crv} & f = v \\ B^{crv} & f \neq c, r, v \end{Bmatrix}$
AC^t	Representa un árbol en crecimiento.	$\lambda(AC^t, e^f) = \begin{Bmatrix} AC^{t+1} & f = t \\ A^q & f = t \wedge t+1 > N_{frames} \\ AC^t & f \neq t \end{Bmatrix}$
AR^q	Árbol	$\lambda(A^q, e^f) = \begin{Bmatrix} AQ^t & f = q \\ A^q & f \neq q \end{Bmatrix}$
R^t	Representa la animación de un relámpago	$\lambda(R^t, e^f) = \begin{Bmatrix} R^{t+1} & f = t \\ \Delta e^q & f = t \wedge t+1 > N_{frames} \\ R^t & f \neq t \end{Bmatrix}$
AQ^t	Árbol quemándose	$\lambda(AQ^t, e^f) = \begin{Bmatrix} AQ^{t+1} & f = t \\ \Delta e^v & f = t \wedge t+1 > N_{frames} \\ AQ^t & f \neq t \end{Bmatrix}$

Tabla 8.3: *Descripción de los actores*

Actor	*Función* τ
B^d	$\tau(B^d(v), e^f) = \begin{Bmatrix} B_{NxN} & f = d \\ \varepsilon & f \neq d \end{Bmatrix}$
AC^d	$\tau(AC^d(v), e^f) = \begin{Bmatrix} D_{(i,j)}(S_s(A)) & f = d \\ \varepsilon & f \neq d \end{Bmatrix}$
A^d	$\tau(AR^d(v), e^f) = \begin{Bmatrix} T_{(i,j)}(A) & f = d \\ \varepsilon & f \neq d \end{Bmatrix}$
R^d	$\tau(R^d(v), e^f) = \begin{Bmatrix} T_{(i,j)}(R) & f = d \\ \varepsilon & f \neq d \end{Bmatrix}$
AQ^d	$\tau(AQ^d(v), e^f) = \begin{Bmatrix} T_{(i,j)}(S_{s^{-1}}(A_q)) & f = d \\ \varepsilon & f \neq d \end{Bmatrix}$

Tabla 8.4: *Definición de las funciones de dibujo*

Como se puede comprobar, en las animaciones el dibujo de los actores siempre va a depender de t, que modificará el estado del actor para cambiar su representación según vaya desarrollándose la animación.

Para terminar se debe definir la cadena inicial: $w_0 = B^{crv}$

Con esta última expresión queda terminada la definición de todos los elementos que componen el sistema de RV para modelar una simulación de incendios forestales producido por relámpagos. Para ver como funciona el algoritmo propuesto en el capítulo anterior, se va a suponer que los generadores definidos anteriormente construyen varios eventos. Dada la cadena inicial, el sistema evolucionaría de la siguiente manera:

Paso 3. $e^* = E(G^*, t) = \{e^t, e^c, c^d\} = \{e^{tcd}\}$
Paso 4. $e^v = e^d$
Paso 5. $e^u = e^* - e^v = e^{ct}$
Paso 6. $w_{sig} = \alpha(w, e^{ct}) = \lambda(B^{crv}, e^{ct}) = AC^0 \cdot B^{crv}$
Paso 7. $v = \Omega(w, e^v) = \Omega(B^{crv}, e^d) = \tau(B^{cr}, e^d) = B_{NxN}$
Paso 8. $\Phi(v) = \Phi(B_{NxN}) = \gamma(B_{NxN})$
Paso 9. $w = w_{sig}$ y $t = t + 1$
Paso 11. Ir a paso 3
Paso 3. $e^* = E(G^*, t) = \{e^t, e^r, c^d\} = \{e^{trd}\}$
Paso 4. $e^v = e^d$
Paso 5. $e^u = e^* - e^v = e^{rt}$
Paso 6. $w_{sig} = \alpha(AC^0 \cdot B^{crv}, e^{rt}) = \lambda(AC^0, e^t) \cdot \lambda(B^{crv}, e^r) = AC^1 \cdot R^0 \cdot B^{crv}$
Paso 7. $v = \Omega(AC^0 \cdot B^{crv}, e^v) = \tau(AC^0, e^d) \cdot \tau(B^{cr}, e^d) = T_{(i,j)}(S_0(A)) \cdot B_{NxN}$
Paso 8. $\Phi(D_{(i,j)}(S_0(A)) \cdot T) = I(T_{(i,j)}) \cdot I(S_0) \cdot \gamma(A) \cdot F(S_0) \cdot F(T_{(i,j)}) \cdot \gamma(B_{NxN})$
Paso 9. $w = w_{sig}$ y $t = t + 1$
Paso 11. Ir a paso 3
.....

8.6 Conclusiones y trabajos futuros

Se ha presentado un modelo con un objetivo prioritario, la separación de la actividad de un sistema gráfico de la de los dispositivos visuales y de interacción que componen dicho sistema.

Por uno lado, se tiene un lenguaje definido mediante gramáticas independientes del contexto que detalla los elementos que componen el sistema. Se establecen determinadas funciones que representan la evolución de cada uno de los elementos a través del tiempo. Se definen funciones que extraen la representación gráfica de los elementos. Esta representación se envía a los dispositivos visuales mediante funciones que separan la implementación concreta sobre un dispositivo visual y la descripción gráfica del elemento, eliminando la dependencia del

sistema de los dispositivos gráficos.

La separación de los dispositivos de entrada de la definición del sistema se ha articulado a través de los generadores de eventos, que levantan una capa entre el hardware de entrada y la representación del sistema. Para enlazar las acciones generadas por los dispositivos de entrada se utiliza el modelo de eventos.

En general, se puede comprobar que todo el modelo intenta separar las partes dependientes del tipo de dispositivo de las de la definición formal de un sistema de RV. La técnica empleada para dicha separación es utilizar modelos matemáticos que transformen las acciones de los dispositivos, tanto visuales como de entrada, en acciones más generales que puedan ser identificados por el sistema independientemente del origen de la acción, mediante abstracciones.

Esto supone varias ventajas: en primer lugar los dispositivos de entrada pueden ser sustituidos por otros dispositivos o por simulaciones de estos. Por otro lado existe la posibilidad de que los elementos del sistema sean reutilizados. Además, la representación de los elementos puede ser visual o no visual, e incluso ser diferente dependiendo del dispositivo de visualización o de las necesidades del usuario. Así, si el dispositivo tiene unas características concretas, la representación siempre se podrá adaptar al dispositivo. Por último, varios actores pueden relacionarse entre sí enviándose mutuamente eventos que ambos reconozcan.

Con el sistema planteado se puede diseñar fácilmente un motor físico utilizando los elementos de la escena. Esto se realiza diseñando generadores de eventos de diferentes tipos, dependiendo de la característica física y activando la actividad del actor necesaria para que este reaccione ante el proceso físico. Por ejemplo, si se quiere que un actor reaccione ante una colisión, el generador de eventos de este tipo calculará las colisiones de los elementos extrayendo la geometría de la escena gracias al motor gráfico (utilizando la implementación de las funciones γ, I, F para calcular los cubos envolventes de los elementos) y generando los eventos necesarios para las colisiones. Si el sistema lo permite, se podría implementar mediante hardware este generador de eventos.

El motor de IA puede ser diseñado en las funciones de evolución de los actores. En estas funciones es donde cada actor va a tomar sus decisiones dependiendo de su estado actual. Mediante eventos se pueden ejecutar actividades que cambien el estado del actor y que modifiquen su comportamiento. Además, como el propio actor puede generar eventos es posible diseñar procesos de retroalimentación utilizados en IA. Además la integración entre motor físico y motor de IA está garantizada ya estos dos motores se relacionan entre sí mediante los

eventos.

El uso de un lenguaje para la descripción de los elementos de la escena y la independencia del sistema gráfico hace que estas cadenas descriptivas puedan ser utilizadas en cualquier otro sistema, siempre y cuando están implementados los elementos básicos (todas las funciones semánticas explicadas).

El modelo presentado en este trabajo está actualmente en desarrollo y se desea seguir desarrollando los diferentes aspectos tratados aquí. Uno de los puntos a investigar y que se apuntan en el artículo es la optimización del algoritmo y su paralelización. La definición del sistema mediante cadenas facilita la posibilidad de algoritmos paralelos. Por otro lado, las cadenas representan estados del sistema y su evolución. Esta evolución puede variar mediante mutaciones y se puede investigar posibles soluciones evolutivas para el diseño del sistema. Por otro lado, se puede estudiar la posibilidad de activar eventos con una cierta probabilidad. Es decir, un actor activaría una actividad, ya no solo cuando suceda un determinado evento, sino que este se activase con una cierta probabilidad. Así un actor podría definirse como $a^{p(e^1),p(e^2)}(w)$, siendo $p(e^i)$ la probabilidad de reaccionar al evento e^i.

En definitiva, se ha pretendido diseñar un sistema reutilizable, genérico y que se pueda incrementar y adaptar fácilmente, en el que la parte fundamental, que es la evolución en el tiempo, sea independiente de cómo se represente y con quién interactúe.

Bibliografía

[1] PHYSX BY AGEIA: `http://physx.ageia.com/`

[2] DAVID J. KASIK WILLIAM BUXTON D. R. F.: Ten cad challenges. *IEEE Computer Graphics and Applications 25* (2005), 81–90.

[3] WORKING MODEL: SIMULACIÓN DE SISTEMAS: `http://www.design-simulation.com`

[4] DAVIS MARTIN D.; SIGAL R., WEYUKER E. J.: *Computability, Complexity, and Languages, Fundamentals of Theoretical Computer Science*, 2nd ed. San Diego: Elsevier Science, 1994.

[5] NEWTON GAME DYNAMICS: `http://www.newtondynamics.com/`

[6] OPEN DYNAMICS ENGINE: http:www.ode.org

[7] WIKIPEDIA - PHYSICS ENGINE: http://en.wikipedia.org/wiki/physics_engine

[8] HAVOK: http://www.havok.com/

[9] JOHN H. MILLER S. E. P.: *Complex Adaptative Systems*. Princeton University Press, 2007.

[10] JOSHUA STRICKON J. A. P.: Emerging technologies. *Siggraph* (2004).

[11] PAL: PHYSICS ABSTRACTION LAYER: http://www.adrianboeing.com/pal/

[12] PÁGINA OFICIAL DE DIRECTX: http://www.microsoft.com/windows/directx/default.mspx

[13] PÁGINA OFICIAL DE OPENGL: http://www.opengl.org/

[14] SIMPLE DIRECTMEDIA LAYER (SDL): http://www.libsdl.org/

[15] OGRE 3D : OPEN SOURCE GRAPHICS ENGINE: http://www.ogre3d.org/

[16] THE VISUALIZATION TOOLKIT (VTK): http://public.kitware.com/vtk/

[17] EO EVOLUTIONARY COMPUTATION FRAMEWORK: http://eodev.sourceforge.net/

[18] CILIB (COMPUTATIONAL INTELLIGENCE LIBRARY): http://cilib.sourceforge.net/

[19] JADE - JAVA AGENT DEVELOPMENT FRAMEWORK: http://jade.tilab.com/

[20] LAIRD J. E.: Using a computer game to develop advanced ai. *Computer 34 (7)* (2001), 70–75.

[21] GEORGIOS N. YANNAKAKIS JOHN LEVINE J. H.: An evolutionary approach for interactive computer games. *In Proceedings of the Congress on Evolutionary Computation* (2004), 986–993.

[22] CHRIS MILES JUAN QUIROZ R. L. S. J. L.: Co-evolving influence map tree based strategy game players. *IEEE Symposium on Computational Intelligence and Games* (2007), 88–95.

[23] ROBERT G. REYNOLDS ZIAD KOBTI T. A. K. L. Y. L. Y.: Unraveling ancient mysteries: Reimagining the past using evolutionary computation in a complex gaming environment. *IEEE transactions on evolutionary computation 9* (2005), 707–720.

[24] WOOLDRIDGE M.: Agent-based software engineering. *IEEE Proceedings Software Engineering 144* (1997), 26–37.

[25] WOOD M. F., DELOACH S.: An overview of the multiagent systems engineering methodology. *AOSE* (2000), 207–222.

[26] KENYON S. H.: Behavioral software agents for real-time games. *IEEE Potentials 25* (2006), 19–25.

[27] AARON KHOO R. Z.: Applying inexpensive ai techniques to computer games. *IEEE Intelligent Systems 17(4)* (2002), 48–53.

Publicación 9

Formal model to integrate Multi-Agent Systems and Interactive Graphic Systems

Gabriel López-García, Rafael Molina-Carmona and Antonio-Javier Gallego-Sánchez

Grupo de Informática Industrial e Inteligencia Artificial, University of Alicante, Spain

{*glopez, rmolina, ajgallego*}*@dccia.ua.es*

International Conference on Agents and Artificial Intelligence, Valencia, Spain (2010). pp. 264-267. ISBN: 978-989-674-022-1. Calificado como CORE C

Formal model to integrate Multi-Agent Systems and Interactive Graphic Systems

Gabriel López-García, Rafael Molina-Carmona and Antonio-Javier Gallego-Sánchez

Grupo de Informática Industrial e Inteligencia Artificial,
University of Alicante, Spain,
{*glopez, rmolina, ajgallego*}*@dccia.ua.es*

Abstract

A formal grammar-based model is presented to integrate the essential characteristics of a Multi-Agent System with the visualization given by an Interactive Graphic Systems. This model adds several advantages, such as the separation between the implementation of the system activity and the hardware devices, or the easy reusability of components. To illustrate the model, a practical case is presented.
Keywords*: Virtual Reality, Virtual Worlds generation, Formal model, Interaction.*

9.1 Introduction

The growing influence of Multi-Agent Systems (MAS) in several fields of research has led to a significant evolution in its development. On the other hand, the spectacular progress of the Interactive Graphic Systems (IGS) has contributed to present the information in a more friendly way using new forms of analysis. The entertainment industry have had a decisive influence on this progress [10].

Agents are usually considered to have some generic characteristics that make them difficult to model [3]. Although there are some common strategies, there is a lack of a unified design pattern. Hence, some problems arise, such as the difficult reproduction of results [1], the lack of a suitable visual representation or the limited capabilities of interaction.

There are many work environments to develop MASs, but they seldom include advanced visual features. For instance, in Sociology [1, 3], very specific

solutions are used [11, 9]. Netlogo [12] is a useful tool with some basic graphics to study complex systems.

Some applications of the MASs to Virtual Reality (VR) can also be found. [7] proposes an implementation of a virtual world where objects react to gestures of the user. The work of [13] uses the concept of MASs perception applied to VR. In entertainment industry, agents are usually called bots [5] and they are programmed as characters of the game. These systems are so successful that they are being used in research [10]. An growing variety of generic development environments are emerging: they implement the most important features of agents [3]. Repast [8] and MASON [6] are examples.

As a conclusion, there are a variety of environments, but a model that unifies the definition of MASs and IGSs has not been proposed. This paper describes our proposal for an integral model based on grammars to develop complex environments that take advantage of the MASs and the IGSs features. This system uses a descriptive language and discrete events to specify agents. It will the interaction with the user and it is independent from the display and the interaction devices. It could also incorporate a physics engine and the agents could be easily reused.

9.2 Proposed Model

In our model, a scene is a set of dynamic and static elements. They are all represented by a sequence of primitives an transformations of a geometric system G. A primitive is not just a draw primitive but an action on the geometric system that can be visual or not. A transformation is a modification on the primitives inside its scope of application.

Agents are the dynamic part and they are made of activities and a set of internal states. Each activity is executed as a reaction to an event. The agents can have a geometrical representation, defined using primitives and transformations. They also provide different ways of communication.

Formally, each element in the scene is represented by a symbol, which are used to build strings to describe the scene. The strings follow a language syntax, which is presented as a grammar [14] defined by the tuple $M = < \Sigma, N, R, s >$, where $\Sigma = P \cup T \cup O \cup A_{ST}^{D}$ is the set of terminal symbols (P: set of symbols for primitives, T: set of symbols for transformations, $O = \{\cdot\,()\}$: set of symbols of separation and operation, A_{ST}^{D}: set of symbols for agents with D the set of

all possible types of events and ST the set of possible states), N = {WORLD, OBJECTS, OBJECT, AGENT, TRANSFORMATION, FIGURE } is the set of non-terminal symbols, s = WORLD is the initial symbol and R is the set of grammar rules defined in the following table.

1.**WORLD** $\rightarrow$ OBJECTS
2.**OBJECTS** $\rightarrow$ OBJECT | OBJECT $\cdot$ OBJECTS
3.**OBJECT** $\rightarrow$ FIGURE | TRANSF. | AGENT
4.**AGENT** $\rightarrow a_{st}^{d}$ (OBJECTS), $a_{st}^{d} \in A_{ST}^{D}, d \in D, st \in ST$
5.**TRANSFORMATION** $\rightarrow t$ (OBJECTS), $t \in T$
6.**FIGURE** $\rightarrow p^{+}$, $p \in P$

A string $w \in \Sigma^{*}$ is generated by M, if it can be obtained from the initial symbol using the given rules. The language $L(M)$ is the set of all the strings which can be generated $L(M) = \{w \in \Sigma^{*} \mid \text{WORLD} \overset{*}{\rightarrow} w\}$. M is a context-free grammar, so there is a procedure to verify if a scene is correctly described.

Apart from the language syntax, it is necessary to define the semantics. It will be defined with a denotational method, through mathematical functions.

Rule 6 defines the syntax of a figure as a sequence of primitives. Primitive semantics is defined by function $\alpha : P \rightarrow G$. Each symbol in P runs a primitive on a geometric system G. So, depending on α and on the geometric system G, the result may be different. G represents actions on a specific geometric system (e.g. a graphical library such as OpenGL).

The scope of a transformation is limited by the symbols "()". Two functions are used to describe the semantics of a transformation: $\beta : T \rightarrow G$ (run when the symbol "(" is processed), and $\delta : T \rightarrow G$ (run when the symbol ")" is found). These two functions have the same features as α, but they are applied to transformations T, on the same geometric system G.

Given a string $w \in L(M)$, a new function φ is defined to run a sequence of primitives P and transformations T in a geometric system G:

$$\varphi(w) = \left\{ \begin{array}{ll} \alpha(w) & \textit{if} \;\; w \in P \\ \beta(t); \varphi(v); \delta(t) & \textit{if} \;\; w = t(v) \wedge v \in L(M) \wedge \\ & \wedge \, t \in T \\ \varphi(s); \varphi(v) & \textit{if} \;\; w = s \cdot v \wedge s, v \in L(M) \end{array} \right\} \tag{9.1}$$

One of the most important features of this system is the independence from a specific graphics system. The definition of α, β and δ provides the differences

in behaviour, encapsulating the implementation details. Therefore, the strings to define a scene may be reused in other systems.

The semantics of agents is a function which defines its evolution in time. It is called *evolution function* λ and is defined as: $\lambda : L(M) \times E^D \to L(M)$, where E^D is the set of events for the device D. By applying $\lambda(w, e^f)$, $w \in L(M)$ is transformed into another string u, which allows the system to evolve. It has a different expression depending on its evolution, but the general expression is:

$$\lambda(a_{st}^d(v), e^f) = \left\{ \begin{array}{ll} u \in L(M) & if \quad f = d \\ a_{st}^d(v) & if \quad f \neq d \end{array} \right\} \tag{9.2}$$

The result u may contain or not the own agent, it can generate other event for the next frame or change the agent state '*st*'.

The function λ defines the *function of the system evolution* η. Given a set of events $e^i, e^j, e^k, \ldots, e^n$ (denoted as e^v, where $v \in D^+$) and a string w, it describes the evolution at a frame. This algorithm uses the operator $\prod_{\forall f \in v}$ to concatenate strings.

$$\eta(w, e^v) = \left\{ \begin{array}{ll} w & if \quad w \in P \\ t(\eta(v, e^v)) & if \quad w = t(v) \\ \prod\limits_{\forall f \in v} (\lambda(a_{st}^d(\eta(y, e^v)), e^f)) & if \quad w = a_{st}^d(y) \\ \eta(s, e^v) \cdot \eta(t, e^v) & if \quad w = s \cdot t \end{array} \right\} \tag{9.3}$$

For the visualization of an agent, it must be first converted into a string of primitives and transformations. This conversion is done by the *visualization function* $\theta : L(M) \times E^V \to L(E)$, where $V \subseteq D$ are events used to create different views, E^V are events created in the visualization process, and $L(E)$ is the language $L(M)$ without agents. It is defined as:

$$\theta(a_{st}^d(v), e^f) = \left\{ \begin{array}{ll} w \in L(E) & if \quad f = d \wedge d \in V \\ \varepsilon & if \quad f \neq d \end{array} \right\} \tag{9.4}$$

As with the function λ, an algorithm is defined for θ. It returns a string $z \in L(E)$, given a string $w \in L(M)$ and a set of events e^v, where $v \in V^+$ and $V \subseteq D$. This function is called *function of system visualization* π and it is defined as: $\pi : L(M) \times E^V \to L(E)$

$$\pi(w,e^v)=\left\{\begin{array}{ll} w & if\ \ w\in P^+ \\ t(\pi(y,e^v)) & if\ \ w=t(y) \\ \prod\limits_{\forall f\in v}(\theta(a^v_{st}(\pi(y,e^v)),e^f)) & if\ \ w=a^v_{st}(y) \\ \pi(s,e^v)\cdot\pi(t,e^v) & if\ \ w=s\cdot t \end{array}\right\} \tag{9.5}$$

The activities are run by agents and activated by events under certain conditions. An event is defined as: e^d_c *is an event of type* $d\in D$ *with data* e*, which is carried out when the condition* c *is fulfilled.* The condition is omitted if $c=true$. Events may include information identifying who sent the message. So, it provides a generic communication system that can implement FIPA or KMQL [2].

It is necessary to establish the independence between the system and the input devices that generate events (hardware or software). So, the events needed to make the system respond to the input devices must be defined. A new function called *event generator* is defined: *Let* $C^d(t)$ *be a function which creates events of type* d *at the time instant* t*, where* $d\in D$ *and* D *is the set of event types which can be generated.*

It is important to note that event generators encapsulate the device-dependent code. They also can model the communication processes that exist in a MAS (agent-agent and agent-environment).

The process which obtains the events produced by the generators is defined as: *Let* C^* *be the set of all the event generators which are associated with input devices and* $E(C^*,t)$ *the function that collects all the events from all the generators, then:*

$$\begin{array}{l} E(C^*,t)=\left\{\begin{array}{ll} e(z,C_i(t)) & if\ z=E(C^*-C_i,t) \\ \varepsilon & if\ C^*=\emptyset \end{array}\right\} \\ e(z,e^i)=\left\{\begin{array}{ll} z\cdot e^i & if\ e^i\notin z \\ z & if\ e^i\in z \end{array}\right\} \end{array} \tag{9.6}$$

Once all the elements involved in the model have been defined, the algorithm which carries out the entire system can be established:

1.	$w = w_o; t = 0$
2.	$e^* = E(G^*, t)$
3.	$e^v =$ events of e^* where $v \in V^+$
4.	$e^u = e^* - e^v$
5.	$w_{next} = \eta(w, e^u)$
6.	$v = \pi(w, e^v)$
7.	$g = \varphi(v)$
8.	$w = w_{next}; \ t = t + 1$
9.	If $w = \varepsilon$ then go to 11
10.	Go to 2
11.	End

where w_o is the initial string, e^* are the events generated at a frame t, G^* = {All the event generators}, D = {All the possible events}, V = {All the visual events}; $V \subseteq D$, e^v all the visual events, e^u all the non-visual events and g the output device.

Steps 2, 3 and 4 manage the system events. In step 5, the evolution algorithm is called to obtain the string for the next frame. In steps 6 and 7, the visualization of the system is performed. In step 8, the next iteration is prepared. The algorithm finishes if the following string is empty.

9.3 Case of Study

This example is an application to simulate fires in forests caused by lightning [4]. The system consists of an agent to define the forest that can create other agents: trees (with a given probability g) and lightning (with a probability f). If lightning are created in the same position as a tree, it will burn as well as the trees around it (Figure 9.1).

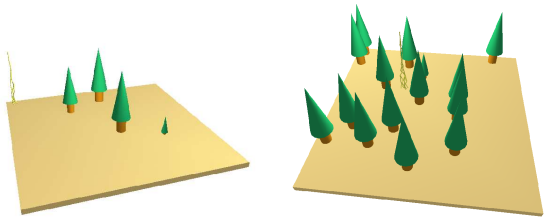


Figure 9.1: *Examples of different simulation states*

To model this example, four elements are defined: events, event generators, agents and primitives. Events are used to produce the necessary activity of system. The events defined for this example are shown in table 9.1. The next step is to define the event generators (table 9.2). Table 9.3 shows the primitives and the transformations that make up the scene. The functions α, β and δ define them.

t	Event generated to increase the time.
c	Creates a tree at the position (i, j) of the forest.
f	Creates a bolt of lightning at position (i, j).
e	Eliminates the tree of the position (i, j).
b	Burns the tree at position (i, j).
v	Draws using a graphics library (e.g. OpenGL).

Table 9.1: *Table of Events.*

(C_{time})	Generate an event e^t at instant t
(C_{forest})	Create tree (e^c) or lightning (e^f)
(C_{draw})	Generate draw event

Table 9.2: *Table of Event Generators.*

Primitive	*Description*
TR	Draw a tree
TR_b	Draw a burning tree
FA	Draw a bolt of lightning
BO	Draw a grid of NxN
Transformations	*Description*
$D_{(i,j)}$	Translate (i,j)
$S_{(s)}$	Scale (n) units

Table 9.3: *Primitives and Transformations.*

The evolution function λ and the graphical representation π are presented in table 9.4. The agent defined for trees (TR) has three internal states $st = \{s1, s2, s3\}$: $s1$ growth, $s2$ adult tree and $s3$ burning tree. This is an example of

different representation for an agent depending on its internal state. The function Nbo sends burn events e^b to all the neighbour trees, creating a chain reaction (agent-agent communication). An example of agent-environment communication is made between the forest and the generator C_{forest}.

Agent	*Function λ and π*
BO Forest	$\lambda(BO^{cfe}, e^i) = \left\{ \begin{array}{ll} TR_{s1}^{t=1} \cdot BO^{cfv} & i = c \\ FA^{t=1} \cdot BO^{cfv} & i = f \\ BO^{cfe} & i = e \\ BO^{cfe} & i \neq c, f, e \end{array} \right\}$ $\pi(BO^v, e^i) = \left\{ \begin{array}{ll} BO & i = v \\ \varepsilon & i \neq v \end{array} \right\}$
TR Tree	$\lambda(TR_{st}^x, e^i) = \left\{ \begin{array}{ll} TR_{s1}^{t+1} & i = t \wedge t+1 \leqslant N \wedge s = s1 \\ TR_{s2}^{b} & i = t \wedge t+1 > N \wedge s = s1 \\ TR_{s3}^{t=1} \wedge & \\ \quad \wedge \Delta Nbo^b & i = b > N \wedge s = s2 \\ TR_{s3}^{t+1} & i = t \wedge s = s3 \\ \Delta e^e & i = t \wedge t+1 > N \wedge s = s3 \\ TR_{st}^{t} & i \neq t \wedge i \neq b \end{array} \right\}$ $\pi(TR_{st}^v, e^i) = \left\{ \begin{array}{ll} D_{(i,j)}(S_{(n)}(TR)) & i = v \wedge st = s1 \\ D_{(i,j)}(TR) & i = v \wedge st = s2 \\ D_{(i,j)}(S_{(-n)}(TR)) & i = v \wedge st = s3 \\ \varepsilon & i \neq v \end{array} \right\}$
FA Lightning	$\lambda(FA^t, e^i) = \left\{ \begin{array}{ll} FA^{t+1} & i = t \wedge t+1 \leqslant N \\ \Delta e^b & i = t \wedge t+1 > N \\ FA^t & i \neq t \end{array} \right\}$ $\pi(FA^v, e^i) = \left\{ \begin{array}{ll} D_{(i,j)}(FA) & i = v \\ \varepsilon & i \neq v \end{array} \right\}$

Table 9.4: *Evolution function λ and the graphical representation π.*

9.4 Conclusions

In this paper a proposal to unify the most relevant features of MASs and IGSs has been presented. The proposed model uses a context-free language to define

the elements. Although further work needs to be done, the use of a descriptive language seems to have several advantages. Firstly, the definition of the scene is reusable and independent from the platform. The use of event generators also makes the interaction with the user independent from the hardware. Event generators are also used to implement the communication, both agent-agent and agent-environment.

As future work the model will be applied to other problems to validate its features. New possibilities such as probabilistic learning strategies or genetic algorithms will be considered.

Bibliography

[1] R. Axelrod (1997). Advancing the Art of Simulation in the Social Sciences. *Simulating Social Phenomena*, Springer.

[2] Michael R. Genesereth, Steven P. Ketchpel (1995). Software Agents. *Communications of the ACM*.

[3] N. Gilbert (2008). Agent-Based Models. *SAGE Publications*.

[4] John H. Miller (2007). Complex Adaptative Systems. *Princeton University Press*.

[5] Aaron Khoo, Robert Zubek (2002). Applying Inexpensive AI Techniques to Computer Games. *IEEE Intelligent Systems* July-August.

[6] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, and Keith Sullivan (2005). MASON: A New Multi-Agent Simulation Toolkit. *Vol. 81, SAGE Journals*.

[7] P. Maes, T. Darrell, B. Blumberg, A. Pentland (1997). The ALIVE System:Wireless, Full-body Interaction with Autonomous Agents. *ACM Multimedia Systems, Vol.5, No.2, pp.105-112*.

[8] M.J. North, T.R. Howe, N.T. Collier, J.R. Vos (2005). The Repast Simphony Runtime System. *Generative Social Processes, Models, and Mechanisms*.

[9] Craig Reynolds (2000). Interaction with Groups of Autonomous Characters. *Game Developers Conference*.

[10] Theresa-Marie Rhyne (2000). Computer Games'Influence on Scientific and Information Visualization. *Entertainment Computing*.

[11] B. Ulicny, D. Thalmann (2001). Crowd simulation for interactive virtual environments and VR training systems. *Computer Animation and Simulation, Springer*.

[12] Wilensky, U. (1999). NetLogo. User Manual.

[13] Ipke Wachsmuth, Yong Cao (1995). Interactive Graphics Design with Situated Agents. *Graphics and Robotics.*

[14] D.Martin, R.Sigal, E.J.Weyuker (1994). *Computability, Complexity, and Languages, Fundamentals of Theoretical Computer Science*, 2nd ed, Elsevier Science.

Publicación 10

Modeling a Mobile Robot using a Grammatical Model

Gabriel López-García, Javier Gallego-Sánchez, J. Luis Dalmau-Espert, Rafael Molina-Carmona and Patricia Compañ-Rosique

Grupo de Informática Industrial e Inteligencia Artificial, Universidad de Alicante, Alicante, Spain

{*glopez, ajgallego, jldalmau, rmolina, patricia*}*@dccia.ua.es*

Int. Symp. on Distributed Computing and Artificial Intelligence (DCAI 2012), Salamanca, Spain (2012). pp. 445-452. ISBN: 978-3-642-28764-0. ISSN: 1867-5662

Modeling a Mobile Robot using a Grammatical Model

Gabriel López-García, Javier Gallego-Sánchez, J. Luis Dalmau-Espert, Rafael Molina-Carmona and Patricia Compañ-Rosique

Grupo de Informática Industrial e Inteligencia Artificial,
Universidad de Alicante, Alicante, Spain
{*glopez, ajgallego, jldalmau, rmolina, patricia*}*@dccia.ua.es*

Abstract

Virtual Worlds Generator is a grammatical model that is proposed to define virtual worlds. It integrates the diversity of sensors and interaction devices, multimodality and a virtual simulation system. Its grammar allows the definition and abstraction in symbols strings of the scenes of the virtual world, independently of the hardware that is used to represent the world or to interact with it.

A case study is presented to explain how to use the proposed model to formalize a robot navigation system with multimodal perception and a hybrid control scheme of the robot.

10.1 Introduction

Autonomous robots are physical agents that perform tasks by navigating in an environment and by manipulating objects in it. To perform these tasks, they are equipped with effectors to act on the environment (wheels, joints...) and with sensors that can perceive it (cameras, sonars...) [6].

The growing disparity of available sensors adds complexity to systems, but it also allows the control of robots to be more accurate. For example, humans and other animals integrate multiple senses [7]. There are other reasons of mathematical nature: combining multiple observations from the same source provides statistical advantages because some redundant observations are obtained for the same estimation.

In this paper we deal with the integration of multimodal inputs in the sense stated by Signhal and Brown [8], that is, the use of data of different nature for decision-making in high-level tasks performed by a robot. However, the proposed system can also deal with the concept of fusion.

The Virtual Worlds Generator (VWG), our proposal, is a grammatical model, which integrates the diversity of interaction and sensing devices and the modules that make up a Graphics System (Graphics, Physics and AI engines). The scene definition is separated from the hardware-dependent characteristics of the system devices. It uses a grammar definition which integrates activities, visualization and interaction with users. The hypothesis is that it can be used as a formal framework to model a robot navigation system, including several multimodal inputs, sensor fusion and integration, and behaviour strategies.

10.2 Model for Virtual Worlds Generation

In the VWG model, a virtual world is described as an ordered sequence of primitives, transformations and actors. A primitive is the description of an object in a given representation system (typically, they are graphical primitives but they could also be sounds or any other primitive in a representation space). Transformations modify the behaviour of primitives, and actors are the components which define the activities of the system in the virtual world. The actors may be finally displayed through primitives and transformations. To model the different actor's activities, the concept of an event is used. Events cause the activation of a certain activity that can be processed by one or more actors.

Each element in the scene is represented by a symbol from the *set of symbols of the scene*. The symbols make up strings that describe the scenes, in accordance with a language syntax, which is presented as a grammar [1].

A grammar M is a tuple $M = < \Sigma, N, R, s >$, where Σ is the finite set of terminal symbols, N is the finite set of non-terminal symbols, R is the finite set of syntactic rules (a syntactic rule is an application r: $N \rightarrow W^*$, where $W = \Sigma \cup N$) and $s \in N$ is the initial symbol of the grammar. In our case, M is defined as:

1. $\Sigma = P \cup T \cup O \cup A^D_{ATTR}$, where:

    - P: set of symbols for primitives.
    - T: set of symbols for transformations.

- $O = \{\cdot\,()\}$: symbols for indicating the scope () and the concatenation $\cdot$.
- A^D_{ATTR}: set of symbols for actors, where D is the set of all the types of events generated by the system and $ATTR$ is the set of all the attributes of actors which define all the possible states. For example, the actor a^H_{attr} will carry out its activity when it receives an event e^h, where $h \in H$, $H \subseteq D$ and $attr \in ATTR$ is its current state.

2. $N = \{\text{WORLD, OBJECTS, OBJECT, ACTOR, TRANSFORM., FIGURE}\}$.
3. Grammar rules R are defined as:
   - Rule 1. **WORLD** $\rightarrow$ OBJECTS
   - Rule 2. **OBJECTS** $\rightarrow$ OBJECT | OBJECT $\cdot$ OBJECTS
   - Rule 3. **OBJECT** $\rightarrow$ FIGURE | TRANSFORM. | ACTOR
   - Rule 4. **ACTOR** $\rightarrow a^H_{attr}$, $a^H_{attr} \in \mathbf{A}^D_{ATTR}, H \subseteq D$
   - Rule 5. **TRANSFORMATION** $\rightarrow t$(OBJECTS), $t \in T$
   - Rule 6. **FIGURE** $\rightarrow p^+$, $p \in P$
4. $s =$ WORLD is the initial symbol of the grammar.

M is a context-free grammar. $L(M)$ is the language generated by the grammar M: $L(M) = \{w \in \Sigma^* \mid \text{WORLD} \overset{*}{\rightarrow} w\}$.

Apart from the language syntax, it is necessary to define the semantics of $L(M)$. It will be defined with a denotational method, that is, through mathematical functions.

Rule 6 defines a figure as a sequence of primitives. Primitive's semantics is defined as a function $\alpha : P \rightarrow G$. Each symbol in the set P carries out a primitive on a given geometric system G. So, depending on the definition of the function α and on the geometry of G, the result of the system may be different. G represents the actions to be run on a specific visual or non-visual geometric system (e.g. the actions on OpenGL or on the system of a robot). The function α provides the abstraction needed to homogenize the different implementations of a rendering system. Therefore, only a descriptive string is needed to run the same scene on different systems.

In **Rule 5**, two functions are used to describe the semantics of a transformation, whose scope is limited by the symbols "()": $\beta : T \rightarrow G$ (carried out when the symbol "(" is processed) and $\delta : T \rightarrow G$ (run when the symbol ")" is found). These two functions have the same features that the function α, but they are applied to the set of transformations T, using the same geometric system G.

Rule 4 refers to actors, which are the dynamic part of the system. The semantics of the actor is a function which defines its evolution in time. For this reason, the semantic function is called *evolution function* λ and it is defined as: $\lambda : A^D_{ATTR} \times E^D \rightarrow L(M)$, where E^D is the set of events for the set of all event types D. The function λ has a different expression depending on its evolution. However, a general expression can be defined. Let $H = \{h_0, \ldots, h_n\} \subseteq D$ be the subset of event types which the actor a^H_{ATTR} is prepared to respond to. The general expression for λ can be seen at (e1), where $u_0, \ldots, u_n$ are strings of $L(M)$. This equation means that an actor a^H_{ATTR} can evolve, that is, it is transformed into another string u_i when it responds to an event e^h which the actor is prepared to respond to. However, the actor remains unchanged when it is not prepared to respond.

As well as dynamic elements, actors can also have a representation in the geometric space G. To be displayed, an actor must be converted to a string of primitives and transformations. This visualization function is defined as: $\theta : A^D_{ATTR} \times E^V \rightarrow L(M')$, where $V \subseteq D$, $E^V \subseteq E^D$ are events created in the visualization process, and $L(M')$ is a subset of the language $L(M)$, made up of the strings with no actors. Let $H \cap V = \{v_0, \ldots, v_n\} \subseteq D$ be the subset of visual event types which the actor a^H_{ATTR} is prepared to respond to. The expression of θ can be seen at (e2).

$$\lambda(a^H_{ATTR}, e^h) = \left\{ \begin{array}{ll} u_0 \in L(M) & if \;\; h = h_0 \\ \ldots & \\ u_n \in L(M) & if \;\; h = h_n \\ a^H_{ATTR} & if \;\; h \notin H \end{array} \right\} \qquad \text{(e1)}$$

$$\theta(a^H_{ATTR}, e^v) = \left\{ \begin{array}{ll} z_0 \in L(M') & if \;\; v = v_0 \\ \ldots & \\ z_n \in L(M') & if \;\; v = v_n \\ \varepsilon & if \;\; v \notin H \cap V \end{array} \right\} \qquad \text{(e2)}$$

The semantic function of these **Rules 1, 2, and 3** breaks down the strings and converts them into substrings, executing the so called *algorithm of the system*, which performs the complete evolution of the system and displays it in the cur-

rent geometric system. It performs several actions, which are described in the following paragraphs.

To display the scene on the geometric system G, the function φ is defined, for the set of symbols that can directly be displayed: primitives and transformations. Given a string $w \in L(M)$ and using only symbols of P and T, φ is defined as:

$$\varphi(w) = \left\{ \begin{array}{ll} \alpha(w) & if\ w \in P \\ \beta(t);\varphi(v);\delta(t) & if\ w = t(v) \wedge v \in L(M) \wedge\ t \in T \\ \varphi(u);\varphi(v) & if\ w = u \cdot v \wedge u, v \in L(M) \end{array} \right\}$$

In the case of strings including both displayable elements, and actors, two functions must be defined. The first one is the so called *function of the system evolution* η, which requires a sequence of sorted events $S = e^1 \cdot e^2 \ldots e^n$, where every $e^i \in E^D$ and a string of $L(M)$ including actors, and implements a set of recursive calls to the function λ to perform the evolution of all the actors in the system at a given frame:

$$\eta(w,S) = \left\{ \begin{array}{ll} w & if\ \ w \in P \\ t(\eta(v,S)) & if\ \ w = t(v) \\ \prod_{e^i \in S} \lambda(a^H_{attr}, e^i) & if\ \ w = a^H_{attr} \\ \eta(u,S) \cdot \eta(v,S) & if\ \ w = u \cdot v \end{array} \right\}$$

The operator $\prod_{e^i \in S} \lambda(a^H_{attr}, e^i)$ concatenates the strings of the function λ.

For the actors to be displayed in the system, they must be converted to displayable elements, that is, primitives and transformations. The second function, returns a string of the language $L(M')$ given a string $w \in L(M)$ and a sequence of ordered visualization events $S' = e^1 \cdot e^2 \ldots e^n$, where every $e^i \in E^V$ and $S' \subseteq S$. This function is called *function of system visualization* π and it is defined as:

$$\pi(w,S') = \left\{ \begin{array}{ll} w & if\ \ w \in P \\ t(\pi(v,S')) & if\ \ w = t(v) \\ \prod_{e^i \in S} \theta(a^H_{ATTR}, e^i) & if\ \ w = a^H_{ATTR} \\ \pi(u,S') \cdot \pi(v,S') & if\ \ w = u \cdot v \end{array} \right\}$$

The **events** are the mechanism to model the activity in the system. The actors activity is carried out when a certain type of event is produced. The following event definition is established: e^d_c *is defined as an event of type* $d \in D$ *with data* c.

A new function called ***event generator*** is defined as: Let $C^d(t)$ be a function which creates a sequence of ordered events of type d at the time instant t, where $d \in D$ and D is the set of event types which can be generated by the system. This function is: $C^d : Time \rightarrow (E^D)^*$

Different event generators can create the same type of events. So, a priority order among event generators must be established to avoid ambiguities.

Once all the elements involved in the model have been defined, the **System Algorithm** can be established. It defines the system evolution and its visualization at every time instant 't' or frame:

1. $w = w_0$; $t = 0$
2. **while** $w \neq \varepsilon$ **do**
   - S = collect events from generators C^* in order of priority.
   - Z = extract visual events from S.
   - $w_{next} = \eta(w, S)$
   - $v = \pi(w, Z)$; $g = \varphi(v)$
   - $w = w_{next}$; $t = t + 1$
3. **end while**

Where w_0 is the initial string, C^* = { All the event generators which generate events of type D }, D = { Set of all the types of possible events in the system }, g is the output device, S is a sequence of all the events generated by the system at instant t, Z is a subsequence of S, and it includes all the events from visual devices. These events are the input of the visual algorithm π.

10.3 Case study

Let us consider a robot with several sensors that provide information about the environment. It is programmed to autonomously navigate in a known environment, and to transport objects from one place to another. The input data are the data from a range sensor, the image from a camera to identify objects and places using markers and a humen supervisor that he is controlling the robot. The information is combined using a multimoldal algorith based on priorities.

A system like this can be modeled using a classical hybrid scheme (figure 10.1). This hybrid scheme can be adapted using the VWG introduced in the previous section.

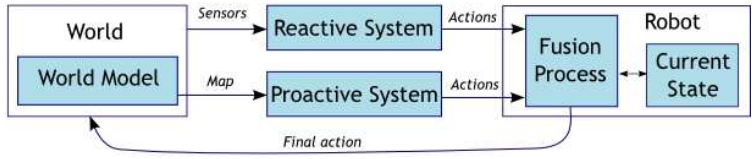


Figure 10.1: *Hybrid scheme for a robotic system.*

In this picture the world is the real environment. The world model is a map containing the static elements of the environment. The reactive system is made of several generators, for the sensors and for the user's orders. The proactive system is the AI of the robot. The robot is the only actor in the system. The current state is the set of robot attributes. The multisensorial integration process is the evolution function of the robot. The final action is the result of the process of sensor integration and the final action carried out by the robot.

Only one primitive is needed, the robot, and it is modified by two possible transformations: move and rotate (table 10.1). When the system is executed in a real environment, the transformations correspond to the actual operations performed by the robot. If it is executed in a simulator, the primitive and the transformations will represent the operations carried out in the graphics system (GS).

	Real environment	Simulator
PRobot	No action	Draw the robot in the GS
$TMove_{<dist>}$	Move a distance *dist*	Move a distance *dist* in the GS
$TRotate_{<angle>}$	Rotate an angle *angle*	Rotate an angle *angle* in the GS

Table 10.1: *Primitives and transformations of the robotic system.*

Events are used to define the activity in the system. Each event is defined by its identifier and some attributes. They produce changes on the actors through their evolution functions. These events are produced by generators. There is a generator for each event type. In the robotic system, five generators are needed:

- *gLaser*: It generates an *eLaser* event when the laser detects an obstacle, by obtaining the laser data and processing them to find the possible obstacles.

It is defined as: $gLaser = eLaser_{<dist,angle>}$ if obstacle, where *dist* is the distance to the obstacle and *angle* is the angle to the obstacle.

- *gCamera*: It generates an *eCamera* event when a marker is detected in the camera image. Markers are used to identify the rooms in the environment. It is defined as: $gCamera = eCamera_{<marker>}$ if a marker is detected.

- *gDecide*: It generates an *eDecide* event each frame to indicate to the robot to make a decision. Is is defined as: *gDecide = eDecide* each frame.

- *gExecute*: It generates an *eExecute* event to indicate the system to execute the robot actions in the current representation space. If the representation space is the real environment, the real operations will take place (move the robot, rotate the robot...). If the current space is the simulator, the operations will take place in the graphics system. It is defined as: *gExecute = eExecute* each frame.

- *gObjective*: It generates an *eObjective* event to set a new objective marker. This generator is connected to the user orders. Users can specify a new target room simply by selecting its associated marker. It is defined as: $gObjective = eObjective_{<marker>}$ if user order.

An order relation must be defined to establish an execution priority among generators. In the robotic system, the order relation is: *gLaser, gCamera, gObjective, gDecide, gExecute*. Therefore, events related with the acquisition of data have the highest priority, compared with the events of decision and execution.

The only actor in our robotic system is the robot which is defined as:

$$ARobot^{eLaser,eCamera,eDecide,eExecute,eObjective}_{<grid,row,column,angle,objective,action>}$$

where the superscript are the events which it is prepared to respond to, and the subscript are the attributes, whose meanings are: the *grid* represents the environment where the robot moves in. Each cell stores the registered data obtained from the sensors (the detected obstacles and markers). *Row and column* are position occupied by the robot in the grid. *Angle* is the robot orientation. *Objective* is the objective room, represented by its marker. And *action* is the string of primitives and transformations which indicates the next command to be executed. To simplify, in the following equations this actor will be referred as $ARobot^{E}_{<g,r,c,an,o,ac>}$.

The evolution function defines the way the robot behaves in the environment. Let e be an event that is received by the actor, the evolution function is defined as:

$$\lambda(ARobot^{E}_{<g,r,c,an,o,act>}, e) = \begin{cases} ARobot^{E}_{<g',r,c,an,o,ac>} & if \;\; e = eLaser_{<dist,angle>} \\ ARobot^{E}_{<g',r,c,an,o,ac>} & if \;\; e = eCamera_{<marker>} \\ ARobot^{E}_{<g,r',c',an',o,ac'>} & if \;\; e = eDecide \\ \alpha(ARobot^{E}_{<g,r,c,an,o,ac>}) & if \;\; e = eExecute \\ ARobot^{E}_{<g,r,c,an,o',ac>} & if \;\; e = eObjective_{<marker>} \\ ARobot^{E}_{<g,r,c,an,o,ac>} & otherwise \end{cases} \tag{10.1}$$

where the symbol apostrophe (') indicates that it has changed as follows:

- If $e = eLaser_{<dist,angle>}$, the grid ($g$) must be updated to indicate that an obstacle has been detected. The cell to mark is the one in position $(r + dist\ \cos(ang + angle), c + dist\ \sin(ang + angle))$.
- If $e = eCamera_{<marker>}$, the grid must be updated to indicate that a marker has been detected. The cell to mark is $(r + dist\ \cos(ang), c + dist\ \sin(ang))$.
- If $e = eDecide$, the current position and orientation of the robot (r, c, ang), must be updated, as well as the actions to be executed.
- If $e = eExecute$, the actions of the robot must be executed in the representation space, through the use of the α function.
- If $e = eObjective_{<marker>}$, a new objective has been set by the user, so the objective (o) must be changed to the new one ($marker$).
- In any other case, the actor must remain unchanged.

The initial string in our system is defined as:

$$ARobot^{eLaser,eCam.,eDecide,eExec.,eObjct.}_{<grid,row,column,angle,\varepsilon,\varepsilon>}$$

where the attribute grid is initialized to a set of empty cells, the attributes row, column and angle are the initial position, and the objective and the actions are empty.

10.3.1 Analysis

A set of tests has been designed to prove the features of our model: The aim of the first test is to prove the suitability of the evolution function to introduce new AI algorithms. This test is not to obtain the best AI algorithm to achieve the goal. Two simple decision algorithms have been used to decide how the robot should move. The first algorithm makes decisions randomly to find the target position. The second algorithm is A* [4]. These two AI algorithms were introduced without making any changes in other parts of the system, just by changing the evolution function.

The aim of the second test is to prove that the input devices can be replaced without changing the system. We can change the laser to a Kinect to detect obstacles. To change this device, we have just designed a new event generator (*gKinect*) that creates events of the same type that the ones generated by the laser generator.

In the third test we want to test the extensibility of the system. New instances of the actor symbols (representing robots) have been added to the definition string to extend the system and create a multi-robot system in an almost immediate way. The updating of the definition string supposes the extension of the model and the addition of new features. Moreover, most elements can be reused in new definition strings to obtain new behaviours with little effort.

In the last experiment we tested the flexibility to work under different conditions. To prove this feature, the system has been tested with different maps, in the case of the simulated robot, and in different real environments, in the case of the real robot.

10.4 Conclusions

A new model to formally define virtual worlds, independently from the underlying physical layer, has been presented. Is has been used to model the control of a mobile robot, navigating in a given environment, and using a set of multimodal inputs from different types of sensors.

Taking into account the diversity of virtual worlds available nowadays and the wide variety of devices, this model seems to be able to provide interesting features. Firstly, it is a formal model that allows to abstract and represent the states of the system in a general way by avoiding specific features. It is a device-independent model, therefore, is not linked to the implementation. It allows the

replacement of physical devices by simulated ones, and the easy addition of new ones.

In conclusion, it has been achieved the main objective of defining a new formal and generic model that is able to model general virtual worlds systems by avoiding the specific peculiarities of other models existing today.

Bibliography

[1] Davis, Martin; Sigal, Ron and Weyuker, Elaine J.: Computability, Complexity, and Languages, Fundamentals of Theoretical Computer Science, 2nd ed. San Diego: Elsevier Science (1994)

[2] Ernst, Marc O. and Bülthoff, Heinrich H.: Merging the senses into a robust percept. TRENDS in Cognitive Sciences, vol.8, no.4, (2004)

[3] Ingrand, F.; Chatila, R. and Alami, R.: An Architecture for Dependable Autonomous Robots. IARP-IEEE RAS Workshop on Dependable Robotics (2001)

[4] Luo, Ren; Lin, Yu-Chih; Kao, Ching-Chung: Automous mobile robot navigation and localization based on floor paln map information and sensory fusion approach. IEEE MFI (2010)

[5] Posadas, J.L.; Poza, J.L., Simó, J.E.; Benet, G.; Blanes, F.: Agent-based distributed architecture for mobile robot control. Engineering Applications of Artificial Intelligence, 805-823 (2008)

[6] Russell, Stuart Jonathan and Norvig, Peter: Artificial intelligence: a modern approach. Prentice Hall. ISBN: 0136042597. (2010)

[7] Sharma, R.; Pavlovic, V. I.; Huang, T. S.: Toward Multimodal Humar-Computer Interface. Proceedings of the IEEE, vol. 86(5), pp. 853-869 (1998)

[8] Singhal, A.; Brown, C.: Dynamic bayes net approach to multimodal sensor fusion. SPIE 1997

Publicación 11

A Grammatical Approach to the Modeling of an Autonomous Robot

Gabriel López-García, A. Javier Gallego-Sánchez, J. Luis Dalmau-Espert, Rafael Molina-Carmona and Patricia Compañ-Rosique

Group of Industrial Computing and Artificial Intelligence
University of Alicante, Alicante, Spain

{*glopez, ajgallego, jldalmau, rmolina, patricia*}*@dccia.ua.es*

International Journal of Interactive Multimedia and Artificial Intelligence, ISSN: 1989-1660, pp. 30-37, Vol. 1, 2012.

A Grammatical Approach to the Modeling of an Autonomous Robot

Gabriel López-García, A. Javier Gallego-Sánchez, J. Luis Dalmau-Espert, Rafael Molina-Carmona and Patricia Compañ-Rosique

Group of Industrial Computing and Artificial Intelligence
University of Alicante, Alicante, Spain
{*glopez, ajgallego, jldalmau, rmolina, patricia*}*@dccia.ua.es*

Abstract

Virtual Worlds Generator is a grammatical model that is proposed to define virtual worlds. It integrates the diversity of sensors and interaction devices, multimodality and a virtual simulation system. Its grammar allows the definition and abstraction in symbols strings of the scenes of the virtual world, independently of the hardware that is used to represent the world or to interact with it. A case study is presented to explain how to use the proposed model to formalize a robot navigation system with multimodal perception and a hybrid control scheme of the robot. The result is an instance of the model grammar that implements the robotic system and is independent of the sensing devices used for perception and interaction. As a conclusion the Virtual Worlds Generator adds value in the simulation of virtual worlds since the definition can be done formally and independently of the peculiarities of the supporting devices.
Keywords*: Autonomous robots, virtual worlds, grammatical models, multimodal perception.*

11.1 Introduction

Autonomous robots are physical agents that perform tasks by navigating in an environment and by manipulating objects in it. To perform these tasks, they are equipped with effectors to act on the environment (wheels, joints, grippers...) and with sensors that can perceive it (cameras, sonars, lasers, gyroscopes...). It should be notice that, in general, the environment in which a robot operates may be inaccessible (it is not always possible to obtain all the information necessary

for decision-making in every moment) non-deterministic (the effect of the action taken by the robot in the environment cannot be guaranteed), non-episodic (the action to be performed by the robot depends on the current perceptions and on the previous decisions), dynamic (the robot and the other elements in the environment may be constantly changing) and continuous (the location of the robot and the moving obstacles change in a continuous range of time and space) [8].

The growing disparity of available sensors adds complexity to systems, but it also allows the control of robots to be more accurate. There are several reasons that support the use of a combination of different sensors to make a decision. For example, humans and other animals integrate multiple senses. Various biological studies have shown that when the signals reach the superior colliculus converge to the same target area [9], which also receives signals from the cerebral cortex and causes the resulting behavior. A large majority of superior colliculus neurons are multisensory. There are other reasons of mathematical nature: combining multiple observations from the same source provides statistical advantages because some redundant observations are obtained for the same estimation.

The concepts from biology can be extrapolated to the field of robotics. In fact, one of the current research fields that arouses most interest is the management of several inputs from different types, the so called multimodal data.

Combining data from different sensors is an open field of research. In this sense, there are several concepts related to this subject that deals with the concept of multimodality from different points of view. Signhal and Brown [10] consider that two main processes may be performed from several multimodal inputs: multisensor fusion and multisensor integration. Multisensor integration refers to the synergistic use of the information provided by multiple sensory devices to assist in the accomplishment of a task by a system. Multisensor fusion refers to any stage in the integration process where there is actual combination (fusion) of different sources of sensory information into one representation format. Other authors describe the evidence that humans combine information following two general strategies: The first one is to maximize information delivered from the different sensory modalities (sensory combination). The second strategy is to reduce the variance in the sensory estimate to increase its reliability (sensory integration) [3]. Another example is set in [11]. They consider that, in general, multimodal integration is done for two reasons: sensory combination and sensory integration. Sensory combination describes interactions between sensory signals that are not redundant. That means crossmodal integration leads to

increased information compared to single modalities. By contrast, sensory integration describes interactions between redundant signals. This leads to enhanced robustness and reliability of the derived information.

In this paper we deal with the integration of multimodal inputs in the sense stated by Signhal and Brown [10], that is, the use of data of different nature for decision-making in high-level tasks performed by a robot. However, the proposed system can also deal with the concept of fusion, defined as the combination of low-level redundant inputs for the cooperative construction of the complete information of the environment, reducing, as a consequence, the levels of uncertainty.

Different architectures have been described for defining the behavior of a robot and the combination of sensory information. A robotic control architecture should have the following properties: programmability, autonomy and adaptability, reactivity, consistent behavior, robustness and extensibility [4].

To achieve those requirements, most robot architectures try to combine reactive control and deliberative control. The reactive control is guided by sensors and it is suitable for low-level decisions in real time. The deliberative control belongs to a higher level, so that global solutions can be obtained from the data collected by the sensors but also from information from an a priori model. They are, therefore, hybrid architectures.

Hybrid architectures arise due to the problems and inconveniences of pure reactive approaches, such as the lack of planning, and of pure deliberative approaches, such as the slow reactions. An example of hybrid architecture is the PRS (Procedural Reasoning System). When the hybrid architectures face a problem, the deliberative mechanisms are used to design a plan to achieve an objective, while the reactive mechanisms are used to carry out the plan. The communications framework is the base that enables the necessary interaction between reactive and deliberative levels, by sending distributed sensory information to tasks at both levels and sending actions to actuators. Deliberative and reactive tasks can be structured in a natural way by means of independent software components [6].

An example of implementation is the model SWE (Sensor Web Enablement), which is applied to systems that are based on the use of sensors to obtain the information that is processed later [1]. In [7] an architecture based on models SWE and DDS (Data Distribution Service) is proposed. DDS is a general-purpose middleware standard designed specifically to satisfy the performance and Qual-

ity of Service (QoS) requirements of real-time systems.

The Virtual Worlds Generator (VWG), our proposal, is a grammatical model, which integrates the diversity of interaction and sensing devices and the modules that make up a Graphics System (Graphics, Physics and AI engines). The scene definition is separated from the hardware-dependent characteristics of the system devices. It uses a grammar definition, which integrates activities, visualization and interaction with users. The hypothesis is that it can be used as a formal framework to model a robot navigation system, including several multimodal inputs, sensor fusion and integration, and behavior strategies.

In section 11.2, the formal model for the VWG is presented. In section 11.3, the formal model is applied to construct a robotic system. Finally, some conclusions are presented in the last section.

11.2 Model for Virtual Worlds Generation

In the VWG model, a virtual world is described as an ordered sequence of primitives, transformations and actors. A primitive is the description of an object in a given representation system (typically, they are graphical primitives but they could also be sounds or any other primitive in a representation space). Transformations modify the behavior of primitives, and actors are the components that define the activities of the system in the virtual world. The actors may be finally displayed through primitives and transformations. To model the different actor's activities, the concept of an event is used. Events cause the activation of a certain activity that can be processed by one or more actors.

Each element in the scene is represented by a symbol from the *set of symbols of the scene*. The symbols make up strings that describe the scenes, in accordance with a language syntax, which is presented as a grammar [2].

11.2.1 Syntax

A grammar M is a tuple $M = <\Sigma, N, R, s>$, where Σ is the finite set of terminal symbols, N is the finite set of non-terminal symbols, R is the finite set of syntactic rules (a syntactic rule is an application r: $N \rightarrow W^*$, where $W = \Sigma \cup N$) and $s \in N$ is the initial symbol of the grammar. In our case, M is defined as:

1. $\Sigma = P \cup T \cup O \cup A^D_{ATTR}$, where:

- P: set of symbols for primitives.
- T: set of symbols for transformations.
- $O = \{\cdot\,()\}$: symbols for indicating the scope () and the concatenation $\cdot$.
- A^D_{ATTR}: set of symbols for actors, where D is the set of all the types of events generated by the system and $ATTR$ is the set of all the attributes of actors which define all the possible states. For example, the actor a^H_{attr} will carry out its activity when it receives an event e^h, where $h \in H$, $H \subseteq D$ and $attr \in ATTR$ is its current state.

2. $N = \{\text{WORLD, OBJECTS, OBJECT, ACTOR, TRANSFORM., FIGURE}\}$.
3. Grammar rules R are defined as:
   - Rule 1. **WORLD** $\rightarrow$ OBJECTS
   - Rule 2. **OBJECTS** $\rightarrow$ OBJECT | OBJECT $\cdot$ OBJECTS
   - Rule 3. **OBJECT** $\rightarrow$ FIGURE | TRANSFORM. | ACTOR
   - Rule 4. **ACTOR** $\rightarrow a^H_{attr}$, $a^H_{attr} \in \mathbf{A}^D_{ATTR}, H \subseteq D$
   - Rule 5. **TRANSFORMATION** $\rightarrow t$(OBJECTS), $t \in T$
   - Rule 6. **FIGURE** $\rightarrow p^+$, $p \in P$
4. $s =$ WORLD is the initial symbol of the grammar.

M is a context-free grammar. $L(M)$ is the language generated by the grammar M: $L(M) = \{w \in \Sigma^* \mid \text{WORLD} \xrightarrow{*} w\}$.

11.2.2 Semantics

Apart from the language syntax, it is necessary to define the semantics of $L(M)$. It will be defined with a denotational method, that is, through mathematical functions.

Semantic Function of Primitives (Rule 6)

Rule 6 defines a figure as a sequence of primitives. Primitive's semantics is defined as a function α, as follows:

$$\alpha : P \rightarrow G \tag{11.1}$$

Each symbol in the set P carries out a primitive on a given geometric system G. So, depending on the definition of the function α and on the geometry of G, the result of the system may be different. G represents the actions to be run on a specific visual or non-visual geometric system (e.g. the actions on OpenGL or on the system of a robot). The function α provides the abstraction needed to homogenize the different implementations of a rendering system. Therefore, only a descriptive string is needed to run the same scene on different systems.

Semantic Functions of Transformations (Rule 5)

In Rule 5, two functions are used to describe the semantics of a transformation, whose scope is limited by the symbols "()":

$$\begin{aligned} \beta &: T \rightarrow G \\ \delta &: T \rightarrow G \end{aligned} \tag{11.2}$$

β represents the beginning of the transformation. It is carried out when the symbol "(" is processed. Function δ defines the end of the transformation which has previously been activated by the function β. It is run when the symbol ")" is found. These two functions have the same features that the function α, but they are applied to the set of transformations T, using the same geometric system G.

Semantic Functions of Actors (Rule 4)

Rule 4 refers to actors, which are the dynamic part of the system. The semantics of the actor is a function that defines its evolution in time. For this reason, the semantic function is called *evolution function* λ and it is defined as:

$$\lambda : A^D_{ATTR} \times E^D \rightarrow L(M) \tag{11.3}$$

where E^D is the set of events for the set of all event types D. Some deeper aspects about events will be discussed later.

The function λ has a different expression depending on its evolution. However, a general expression can be defined. Let $H = \{h_0, \ldots, h_n\} \subseteq D$ be the subset of event types which the actor a^H_{ATTR} is prepared to respond to. The general expression for λ is:

$$\lambda(a^H_{ATTR}, e^h) = \left\{ \begin{array}{ll} u_0 \in L(M) & if \;\; h = h_0 \\ \cdots & \\ u_n \in L(M) & if \;\; h = h_n \\ a^H_{ATTR} & if \;\; h \notin H \end{array} \right\} \tag{11.4}$$

where $u_0, \ldots, u_n$ are strings of $L(M)$. This equation means that an actor a^H_{ATTR} can evolve, that is, it is transformed into another string u_i when it responds to an event e^h which the actor is prepared to respond to. However, the actor remains unchanged when it is not prepared to respond.

As well as dynamic elements, actors can also have a representation in the geometric space G. To be displayed, an actor must be converted to a string of primitives and transformations. This visualization function is defined as:

$$\theta : A^D_{ATTR} \times E^V \rightarrow L(M') \tag{11.5}$$

where $V \subseteq D$, $E^V \subseteq E^D$ are events created in the visualization process, and $L(M')$ is a subset of the language $L(M)$, made up of the strings with no actors. Let $H \cap V = \{v_0, \ldots, v_n\} \subseteq D$ be the subset of visual event types which the actor a^H_{ATTR} is prepared to respond to. The expression of θ is defined as:

$$\theta(a^H_{ATTR}, e^v) = \left\{ \begin{array}{ll} z_0 \in L(M') & if \;\; v = v_0 \\ \cdots & \\ z_n \in L(M') & if \;\; v = v_n \\ \varepsilon & if \;\; v \notin H \cap V \end{array} \right\} \tag{11.6}$$

Semantic Functions of OBJECT, OBJECTS and WORLD (Rules 1, 2 and 3)

The semantic function of Rules 1, 2, and 3 breaks down the strings and converts them into substrings, executing the so called *algorithm of the system*, which performs the complete evolution of the system and displays it in the current geometric system. It performs several actions, which are described in the following paragraphs.

To display the scene on the geometric system G, the function φ is defined, for the set of symbols that can directly be displayed: primitives and transformations. Given a string $w \in L(M)$ and using only symbols of P and T, φ is defined as:

$$\varphi(w) = \left\{ \begin{array}{ll} \alpha(w) & if\ w \in P \\ \beta(t);\varphi(v);\delta(t) & if\ w = t(v) \wedge v \in L(M) \wedge\ t \in T \\ \varphi(u);\varphi(v) & if\ w = u \cdot v \wedge u, v \in L(M) \end{array} \right\} \quad (11.7)$$

In the case of strings including both displayable elements, and actors, two functions must be defined. The first one is the so called *function of the system evolution* η, which requires a sequence of sorted events $S = e^1 \cdot e^2 \ldots e^n$, where every $e^i \in E^D$ and a string of $L(M)$ including actors, and implements a set of recursive calls to the function λ to perform the evolution of all the actors in the system at a given frame:

$$\eta(w,S) = \left\{ \begin{array}{ll} w & if\ \ w \in P \\ t(\eta(v,S)) & if\ \ w = t(v) \\ \prod_{e^i \in S} \lambda(a^H_{attr}, e^i) & if\ \ w = a^H_{attr} \\ \eta(u,S) \cdot \eta(v,S) & if\ \ w = u \cdot v \end{array} \right\} \quad (11.8)$$

The operator $\prod_{e^i \in S} \lambda(a^H_{attr}, e^i)$ concatenates the strings of the function λ.

The actors to be displayed in the system must be converted to displayable elements, that is, primitives and transformations. The second function, returns a string of the language $L(M')$ given a string $w \in L(M)$ and a sequence of ordered visualization events $S' = e^1 \cdot e^2 \ldots e^n$, where every $e^i \in E^V$ and $S' \subseteq S$. This function is called *function of system visualization* π and it is defined as:

$$\pi(w,S') = \left\{ \begin{array}{ll} w & if\ \ w \in P \\ t(\pi(v,S')) & if\ \ w = t(v) \\ \prod_{e^i \in S} \theta(a^H_{ATTR}, e^i) & if\ \ w = a^H_{ATTR} \\ \pi(u,S') \cdot \pi(v,S') & if\ \ w = u \cdot v \end{array} \right\} \quad (11.9)$$

11.2.3 Events and Generators

The events are the mechanism to model the activity in the system. The actors activity is carried out when a certain type of event is produced. The following event definition is established: *e^d_c is defined as an event of type $d \in D$ with data c.*

A new function called *event generator* is defined as: Let $C^d(t)$ be a function which creates a sequence of ordered events of type d at the time instant t, where $d \in D$ and D is the set of event types which can be generated by the system. This function is:

$$C^d : Time \rightarrow (E^D)^* \tag{11.10}$$

In the previous definition, it should be noticed that events are generated in the time instant t. It is due to synchronization purpose. The event generator can generate several or no events at a given moment.

Different event generators can create the same type of events. So, a priority order among event generators must be established to avoid ambiguities. Given two generators C_i and C_j which create the same event, if $i < j$, then the events generated by C_i will have a higher priority.

11.2.4 System Algorithm

Once all the elements involved in the model have been defined, the *System Algorithm* can be established. It defines the system evolution and its visualization at every time instant 't' or frame:

1. $w = w_0$; $t = 0$
2. **while** $w \neq \varepsilon$ **do**
   - S = collect events from generators C^* in order of priority.
   - Z = extract visual events from S.
   - $w_{next} = \eta(w, S)$
   - $v = \pi(w, Z)$; $g = \varphi(v)$
   - $w = w_{next}$; $t = t + 1$
3. **end while**

where w_0 is the initial string, C^* = { All the event generators which generate events of type D }, D = { Set of all the types of possible events in the system }, g is the output device, S is a sequence of all the events generated by the system at instant t, Z is a subsequence of S, and it includes all the events from visual devices. These events are the input of the visual algorithm π.

A diagram of the virtual world generation algorithm is shown in Fig. 11.1.

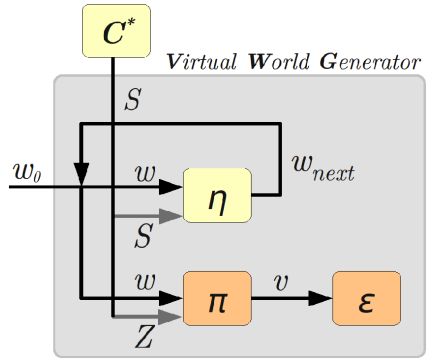


Figure 11.1: *Virtual world generator algorithm.*

This formalization of the system has two main consequences. First, the scene definition is separated from the hardware- dependent characteristics of components. The functions α, β and δ provide the independence from the visualization system, and the event generators provide the independence from the hardware input devices. Secondly, due to the fact that there is a specific scheme to define the features of a system, the different system elements can be reused easily in other areas of application.

11.3 Case Study

11.3.1 Description

Let us consider a robot with several sensors that provide information about the environment. It is programmed to autonomously navigate in a known environment, and to transport objects from one place to another. The input data are: the data from a range sensor (e.g. a laser to detect obstacles and distances), the image from a camera to identify objects and places using markers, an internal representation of the environment (a map) and a human supervisor who is controlling the robot (he can give some high level instructions, such as interrupt the current task or begin a new task). The information is combined using a multimodal algorithm based on priorities, so that the robot can attend to the users' request, select the best way to follow to the destination and use the sensors to detect and avoid obstacles, as well as to identify the objects and the places.

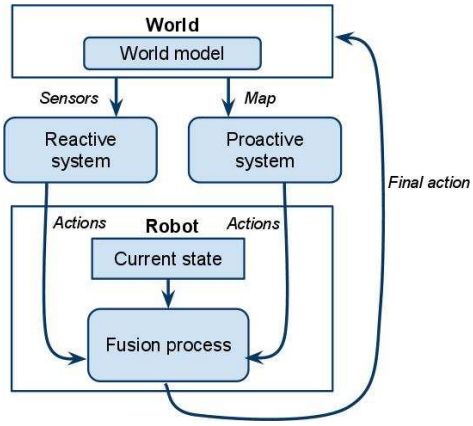


Figure 11.2: *Hybrid scheme for a robotic system.*

A system like this can be modeled using a classical hybrid scheme (Fig. 11.2), based on the combination of a reactive system and a proactive system. This hybrid scheme can be adapted using the VWG introduced in the previous section.

In this picture the world is the real environment. The world model is a map containing the static elements of the environment. The reactive system is made of several generators, for the sensors and for the user's orders. The proactive system is the AI of the robot. The robot is the only actor in the system. The current state is the set of robot attributes. The multisensorial integration process is the evolution function of the robot. The final action is the result of the process of sensor integration and the final action carried out by the robot.

11.3.2 Primitives and Transformations

As it was stated in section 11.2, primitives are the description of objects in the space of representation, and transformations are used to modify primitives. In our robotic system, only one primitive is needed, the robot, and it is modified by two possible transformations: move and rotate (table 11.1). When the system is executed in a real environment, the robot primitive represents the real robot and the transformations correspond to the actual operations performed by the robot. If it is executed in a simulator, the primitive and the transformations will represent the operations carried out in the simulated robot, that is, the operations in the

graphics system (GS). The operations are performed by the semantic functions α for the primitives and β and δ for the transformations.

	Real Environment	Simulator
PRobot	No action	Draw the robot in the GS
$TMove_{<dist>}$	Move a distance *dist*	Move a distance *dist* in the GS
$TRotate_{<angle>}$	Rotate an angle *angle*	Rotate an angle *angle* in the GS

Table 11.1: *Primitives and transformations of the robotic system.*

11.3.3 Events and Generators

Events are used to define the activity in the system. Each event is defined by its identifier and some attributes. They produce changes on the actors through their evolution functions. These events are produced by generators. There is a generator for each event type. In the robotic system, five generators are needed:

- *gLaser*: It generates an *eLaser* event when the laser detects an obstacle, by obtaining the laser data and processing them to find the possible obstacles.
- *gCamera*: It generates an *eCamera* event when a marker is detected in the camera image. Markers are used to identify the rooms in the environment.
- *gDecide*: It generates an *eDecide* event each frame to indicate to the robot to make a decision.
- *gExecute*: It generates an *eExecute* event to indicate the system to execute the robot actions in the current representation space. If the representation space is the real environment, the real operations will take place (move the robot, rotate the robot...). If the current space is the simulator, the operations will take place in the graphics system.
- *gObjective*: It generates an *eObjective* event to set a new objective marker. This generator is connected to the user orders. Users can specify a new target room simply by selecting its associated marker.

The generators in our system and their associated events are shown in table 11.2.

Generator and Events	Description	Associated data
$gLaser = eLaser_{<dist,angle>}$ if obstacle	Event produced when the laser detects an obstacle	*dist*: distance to the obstacle *angle*: angle to the obstacle
$gCamera = eCamera_{<marker>}$ if marker	Event produced when the camera detects a marker	*marker*: detected marker
gDecide = *eDecide* each frame	Event generated each frame to indicate to the robot to make a decision	No data
gExecute = *eExecute* each frame	It runs the robot action in the real environment or in the simulator	No data
$gObjective = eObjective_{<mrkr>}$ if user order	Event produced by the user to set the objective marker	*marker*: objective marker

Table 11.2: *Generators and events of the robotic system.*

An order relation must be defined to establish an execution priority among generators. In the robotic system, the order relation is: *gLaser, gCamera, gObjective, gDecide, gExecute*. Therefore, events related with the acquisition of data have the highest priority, compared with the events of decision and execution.

11.3.4 Actors

The only actor in our robotic system is the robot, which is defined as:

$$ARobot^{eLaser,eCamera,eDecide,eExecute,eObjective}_{<grid,row,column,angle,objective,action>} \tag{11.11}$$

where the superscript are the events which it is prepared to respond to, and the subscript are the attributes, whose meanings are: the *grid* represents the environment where the robot moves in. Each cell stores the registered data obtained

from the sensors (the detected obstacles and markers). *Row* and *column* are the position occupied by the robot in the grid. *Angle* is the robot orientation. *Objective* is the objective room, represented by its marker. And *action* is the string of primitives and transformations that indicates the next command to be executed by the robot. To simplify, in the following equations this actor will be referred as $ARobot^{E}_{<g,r,c,an,o,ac>}$.

The evolution function is, probably, the most important element in the system, as it defines the way the robot behaves in the environment, that is, it defines the artificial intelligence of the robotic system. Let e be an event that is received by the actor, the evolution function is defined as:

$$\lambda(ARobot^{E}_{<g,r,c,an,o,act>}, e) =$$

$$\begin{cases} ARobot^{E}_{<g',r,c,an,o,ac>} & if \;\; e = eLaser_{<dist,angle>} \\ ARobot^{E}_{<g',r,c,an,o,ac>} & if \;\; e = eCamera_{<marker>} \\ ARobot^{E}_{<g,r',c',an',o,ac'>} & if \;\; e = eDecide \\ \alpha(ARobot^{E}_{<g,r,c,an,o,ac>}) & if \;\; e = eExecute \\ ARobot^{E}_{<g,r,c,an,o',ac>} & if \;\; e = eObjective_{<marker>} \\ ARobot^{E}_{<g,r,c,an,o,ac>} & otherwise \end{cases} \tag{11.12}$$

where the symbol apostrophe (') on an attribute indicates that it has changed as a consequence of the received event. The way the attributes change is the following:

- If $e = eLaser_{<dist,angle>}$, the grid ($g$) must be updated to indicate that an obstacle has been detected. The cell to mark is the one in position $(r + dist\ \cos(ang + angle), c + dist\ \sin(ang + angle))$.

- If $e = eCamera_{<marker>}$, the grid (g) must be updated to indicate that a marker has been detected. The cell to mark is $(r + dist\ \cos(ang), c + dist\ \sin(ang))$.

- If $e = eDecide$, the current position and orientation of the robot (row r, column c and angle ang), must be updated, as well as the actions to be executed. This function is very important, as it provides the behavior of the robot. In the following section, the way to introduce intelligent behaviors will be shown.

- If $e = eExecute$, the actions of the robot must be executed in the representation space, through the use of the α function.
- If $e = eObjective_{<marker>}$, a new objective has been set by the user, so the objective (o) must be changed to the new one (*marker*).
- In any other case, the actor must remain unchanged.

11.3.5 Initial string

The initial string in our systems defined as:

$$ARobot^{eLaser,eCam.,eDecide,eExec.,eObjct.}_{<grid,row,column,angle,\varepsilon,\varepsilon>} \tag{11.13}$$

where the attribute *grid* is initialized to a set of empty cells, the attributes *row*, *column* and *angle* are the initial position and orientation, and the *objective* and the *action* are empty.

11.3.6 Analysis

A set of tests has been designed to prove the features of our model. Specifically, five tests have been carried out.

Test of the evolution function

As it was stated before, the evolution function is the way of introducing intelligent behaviors in an actor. Therefore, the aim of this test is to prove the suitability of the evolution function to introduce new AI algorithms. This test is not to obtain the best AI algorithm to achieve the goal, but to prove that a new intelligent behavior can be introduced by just changing the evolution function. An important question is guaranteeing the same conditions for all the experiments, so the AI algorithms are introduced with no other modification in other parts of the system.

Two simple decision algorithms have been used to decide how the robot should move in the world. The first algorithm makes decisions randomly to find the target position. The second one is the A* algorithm [5], considering the Euclidean distance to the goal as the weights. If there is an obstacle the distance is defined as infinite.

Test of device independence

One of the main features of our model is that the system definition is independent from the input devices. The aim of this test is to prove that the input devices can be replaced without changing the definition of the string representing the system.

In our original system, a laser range sensor was used to detect obstacles. In this test, a Kinect device is introduced. To add this new device, we have just designed a new event generator (*gKinect*) that creates events of the same type that the ones generated by the laser generator. That is, it provides the same information: the angle and the distance to the obstacle. The new device is then introduced with no other modification in the system. The Kinect is then used to replace the laser device or to obtain redundant information for the detection of obstacles.

Test to validate the simulation

The most important achievement in the proposed model is the fact that the description for the simulation and for the real robot is exactly the same. That is, the command execution for the simulated robot can be directly used for the real robot with no change in the string that represents the system.

To achieve this goal, two generators for the execution of the robot commands have been implemented: one for the real robot and one for the robot simulation. This way, the commands are transparently executed no matter whether the robot is real or simulated, just using the appropriate generator. As a result, the navigation would be exactly the same for the simulated robot and for the real one, if there were not odometry errors. A good way to improve the simulation is introducing some odometry errors in the motors and in the sensor signals, accordingly with the features of the real robot.

Test of the system extensibility

The proposed model is, by definition, easily extensible. The updating of the definition string supposes the extension of the model and the addition of new features. Moreover, most elements can be reused in new definition strings to obtain new behaviors with little effort.

In our case, new instances of the actor symbols (representing robots) have been added to the definition string to extend the system in an almost immediate way and to create a multi-robot system.

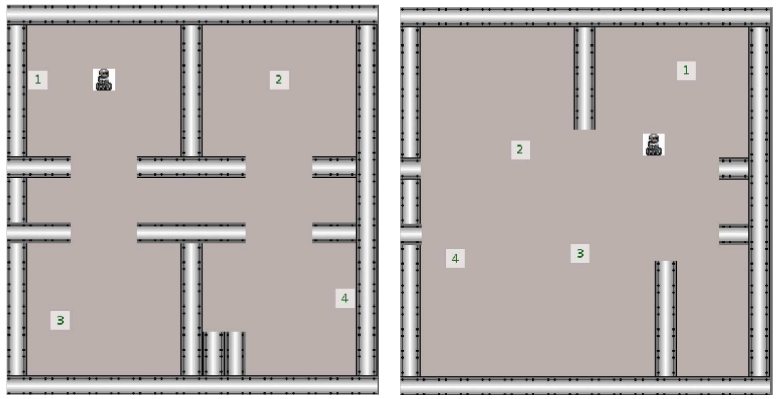


Figure 11.3: *Example map in 2D.*

Figure 11.4: *Example map in 3D.*

Test of changes in the environment

A desired capability in a robot navigation system is, obviously, to be flexible enough to work under very different conditions. To prove this feature, the system has been tested with different maps (Fig. 11.3 and 11.4), in the case of the simulated robot, and in different real environments, in the case of the real robot.

11.4 Conclusions

A new model to formally define virtual worlds, independently from the underlying physical layer, has been presented. Is has been used to model the control of a mobile robot, navigating in a given environment, and using a set of multimodal

inputs from different types of sensors.

The model is based on a grammar which consists, on the one hand, of symbols to abstract and represent the elements of the system (primitives, actors, and so on) and, on the other hand, of a set of evolution functions so that all these elements can be combined in different ways leading to an infinite set of possible strings belonging to the grammar. By definition, each string has the ability to represent the interaction between the elements (symbols) of the system and their state at any given instant. By extension, these strings can also synthesize and formally define the system state.

As in other systems for modeling virtual worlds, the event and, in particular, the occurrence thereof, can bring about a change in the state of a particular element and, in general, a change in the state of the system. Within the model, the event generators are responsible for managing all the possible events associated with the elements of the system.

The result of the events, namely the transition between states, involves an evolution of the original string of the system to another evolved string, which is obtained from the application of certain rules on the first string. These rules are defined within the actors, which contain the logic of how to act and deal with an event if it is activated. The main restriction to design the rules is that they should be able to translate the consequence of the events into grammar rules. The grammar rules must be applicable to the symbols of the state string and the outcome of the rules application must return a consistent string, syntactically and semantically possible.

The evolution function of the actors can be as complex as needed. In fact, this function is the vehicle to introduce intelligent behaviors in the system. This way, artificial intelligence algorithms can be introduced into the evolution function of the actor to provide it with the needed behavior.

Taking into account the diversity of virtual worlds systems available nowadays and the wide variety of devices, this model seems to be able to provide interesting features. Firstly, it is a formal model based on a grammar that allows abstracting and representing the states of the system in a general way by avoiding the specific features of other existing systems. The use of strings facilitates the parallelization and optimization of the system processes. It is also a device-independent model, therefore, is not linked to the implementation of the system with a given set of devices. It also allows the replacement of physical devices by simulated ones, and the easy addition of new ones. For instance, in

the case of our robotic system, the definition string of the system is exactly the same for the simulator and for the real robot. Finally, it is a flexible model since it contemplates the possibility of reinterpreting the outputs of the actions.

In conclusion, it has been achieved the main objective of defining a new formal and generic model that is able to model general virtual worlds systems by avoiding the specific peculiarities of other models existing today.

Bibliography

[1] Botts, M.; Percivall, G.; Reed, C. and Davidson, J.: OGC Sensor Web Enablement: Overview And High Level Architecture. OGC White Paper. Open Geospatial Consortium Inc., 2006.

[2] Davis, Martin; Sigal, Ron and Weyuker, Elaine J.: Computability, Complexity, and Languages, Fundamentals of Theoretical Computer Science, 2nd ed. San Diego: Elsevier Science, 1994.

[3] Ernst, Marc O. and Bülthoff, Heinrich H.: Merging the senses into a robust percept. TRENDS in Cognitive Sciences, vol.8, no.4, 2004.

[4] Ingrand, F.; Chatila, R. and Alami, R.: An Architecture for Dependable Autonomous Robots. IARP-IEEE RAS Workshop on Dependable Robotics, 2001.

[5] Luo, Ren; Lin, Yu-Chih; Kao, Ching-Chung: Autonomous mobile robot navigation and localization based on floor paln map information and sensory fusion approach. IEEE MFI, 2010.

[6] Posadas, J.L.; Poza, J.L., Simó, J.E.; Benet, G.; Blanes, F.: Agent-based distributed architecture for mobile robot control. Engineering Applications of Artificial Intelligence, pp. 805-823, 2008.

[7] Poza, L.; Posadas, J.; Simó, J.; Benet, G.: Arquitecturas de control jeraárquico inteligente con soporte a la calidad de servicio. XXIX Jornadas de Automática, 2008.

[8] Russell, Stuart Jonathan and Norvig, Peter: Artificial intelligence: a modern approach. Prentice Hall. ISBN: 0136042597, 2010.

[9] Sharma, R.; Pavlovic, V. I.; Huang, T. S.: Toward Multimodal Humar-Computer Interface. Proceedings of the IEEE, vol. 86(5), pp. 853-869, 1998.

[10] Singhal, A.; Brown, C.: Dynamic bayes net approach to multimodal sensor fusion. SPIE, 1997.

[11] Weser, Martin; Jockel, Sascha and Zhang, Jianwei: Fuzzy Multisensor Fusion for Autonomous Proactive Robot Perception IEEE International Conference on Fuzzy Systems (FUZZ), 2263-2267, 2008.

Publicación 12

Inference of *k*-testable directed acyclic graph languages

Damián López*, J. Calera-Rubio** and A. Javier Gallego-Sánchez**

** Departamento de Sistemas Informáticos y Computación, Technical University of Valencia, Spain, dlopez@dsic.upv.es*

*** Departamento de Lenguajes y Sistemas Informáticos, University of Alicante, Spain, {calera, jgallego}@dlsi.ua.es*

Journal of Machine Learning Research: Workshop and Conference Proceedings, Vol. 21: ICGI 2012, pp 149-163, Washington, D.C., USA (2012). ISSN: 1938-7228.
Proceedings de la revista JMLR. JCR Impact Factor 2.561

Inference of *k*-testable directed acyclic graph languages

Damián López*, J. Calera-Rubio** and A. Javier Gallego-Sánchez**

* Departamento de Sistemas Informáticos y Computación,
Technical University of Valencia, Spain,
dlopez@dsic.upv.es
** Departamento de Lenguajes y Sistemas Informáticos,
University of Alicante, Spain,
{*calera, jgallego*}*@dlsi.ua.es*

Abstract

In this paper, we tackle the task of graph language learning. We first extend the well-known classes of *k-testability* and *k-testability in the strict sense* languages to directed graph languages. Second, we propose a graph automata model for directed acyclic graph languages. This graph automata model is used to propose a grammatical inference algorithm to learn the class of directed acyclic *k*-testable in the strict sense graph languages. The algorithm runs in polynomial time and identifies this class of languages from positive data.
Keywords*: Graph languages; graph automata; k-testable languages.*

12.1 Introduction

Regular string language inference has been widely applied to many tasks, from bioinformatics [23, 16] to script recognition [19]. See [5] for a bibliographic study.

It is important here to note that in many contexts structural information is of great importance. This kind of information is not easy to model with subclasses of the regular language class. Nevertheless, structural information can be naturally modeled by context-free grammars of various types. In this line of work, Sakakibara [21, 22] presented the first algorithms for learning context-free languages with polynomial time complexity. Some other recent results for non-regular language inference study the inference of context-free languages [15, 4].

Trees are a simple class of graphs with very interesting properties. Thus, several works study the task of tree language inference [7, 13], as well as its application to real tasks [18, 11]. In the grammatical inference framework, when more general graphs are considered, the main problem that arises is computational complexity, and, usually, graphs are reduced to less complex representations (usually some kind of graph traversal).

Our paper considers graphs as elements of some formal language. In this framework, the handbook edited by [20] summarizes, among other results, the two main formalisms used to generate graph languages (node and hyperedge replacement grammars) as well as many theoretical results that relate graph grammars with logic. But, despite many works study the generating paradigm, there does not exist a recognizing device that properly fits all the different characterized classes of graph languages. Despite this, there have been proposed some graph automata models, for instance [17, 3, 1].

Some works have studied the inference of graph grammars, but all of them based on the search of general subgraph isomorphism, and therefore, with high time complexity [9, 10, 12]. Recently, some works take profit from results on mining in graphs in order to propose graph grammar inference methods [2, 6].

In this paper, we first extend the well-known families of *k-testable* and *k-testable in the strict sense* (k-TSS) languages [14] to directed-graph languages. We also prove some lemmas that show that the main features of the k-TSS class of languages are still applicable to graph languages. Second, we take into account the paper by [17] to propose a graph automata model for directed acyclic graph languages, and finally, we propose a polynomial grammatical inference algorithm to learn the same class of directed acyclic k-TSS graph languages from positive data. Let us note that the k-testable structures our approach takes into account help to bound the above mentioned high complexity of graph comparison. Besides, the consideration of directed acyclic graphs also help to further ease the general complexity. We study the time complexity of this algorithm and prove its polynomial behavior.

12.2 Notation and Definitions

A node-labeled directed graph (from now on referred to as graph if not stated otherwise) can be defined by a tuple $g = (V, E, \mu)$, where V is a finite set of nodes (also called vertexes), $E \subseteq (V \times V) - id_V$ is the set of edges (where id_V denotes

the smallest reflexive relation), and $\mu : V \to \Sigma$ is the node labeling function. We will refer to the components of a graph g as V_g, E_g and μ_g only when necessary. An acyclic graph is such that the reflexive-transitive closure of E is a partial order. Two graphs $g = (V,E,\mu)$ and $g' = (V',E',\mu')$ are *isomorphic* if there is a bijection $f : V \to V'$ such that, for any nodes $u,v \in V$, $\mu(v) = \mu'(f(v))$ and $(u,v) \in E$ if and only if $(f(u), f(v)) \in E'$.

For any given node v, an edge (u,v) is called *incoming* (resp. *outgoing* for edges of the form (v,u)). The *incoming degree* of a node v (resp. *outgoing degree*) will be denoted by $idg(v)$ and is defined as $idg(v) = |\{(u,v) \in E\}|$ (resp. the outgoing degree is defined as $odg(v) = |\{(v,w) \in E\}|$). For any graph $g = (V,E,\mu)$, let $V_m^n(g)$ be defined as $V_m^n(g) = \{v \in V \ : \ idg(v) = n \wedge odg(v) = m\}$, and let ε_g denote the empty graph (the graph with no nodes). In the following, two sets of nodes will be of special interest: the set of nodes with zero incoming degree and the set of nodes with zero outgoing degree of a graph g, which we will denote respectively with $V^0(g)$ and $V_0(g)$.

Let a typed alphabet Σ_τ^σ be the association of an alphabet Σ with a relation $r \subseteq (\Sigma \times \mathbb{N} \times \mathbb{N})$. This alphabet plays the same role as the plain alphabet in string languages or the ranked alphabet in tree languages. Since the nodes of the graph may have different incoming and outgoing degrees, it is necessary to establish which symbols can label those nodes. We will denote with Σ_m^n the set: $\{s \in \Sigma \ : \ (s,n,m) \in r\}$, that is, the set of symbols that are able to label the nodes with a given incoming degree n and outgoing degree m.

Once the typed alphabet is defined, the set of all possible consistently labeled graphs can be defined. Formally, let $\mathscr{G}(\Sigma_\tau^\sigma)$ denote the set of graphs over Σ_τ^σ. A *graph language* is any set $L_G \subseteq \mathscr{G}(\Sigma_\tau^\sigma)$.

Given a typed alphabet Σ_τ^σ, let the *extended alphabet* $\widehat{\Sigma}$ be the alphabet defined as the set:

$$\widehat{\Sigma} = \{a_m^n \ : \ a \in \Sigma, \ (a,n,m) \in r\}$$

Taking into account the extended alphabet, given a graph $g = (V,E,\mu)$, we define its *extended graph* as $\hat{g} = (V,E,\hat{\mu})$, where $\hat{\mu} : V \to \widehat{\Sigma}$, and such that for each node v in $V_m^n(g)$, if $\mu(v) = a$, then $\hat{\mu}(v) = a_m^n$. Intuitively, the labels of the nodes of the extended subgraph $\hat{g}$ explicitly include the incoming and outgoing degrees. Figure 12.1 shows an example.

From now on we will consider skeletal graphs and skeletal graph languages (graphs where the nodes v such that $odg(v) \neq 0$ (internal nodes) are labeled with

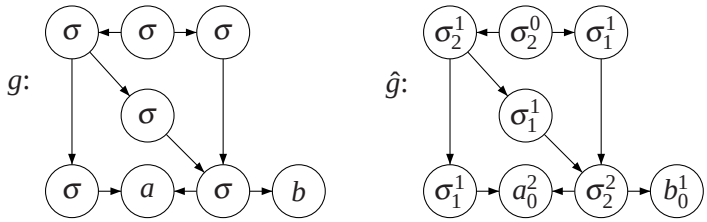


Figure 12.1: *Example of a directed acyclic graph g and its corresponding extended graph ĝ. In order to give an explicit representation, we label non-frontier nodes (those with outgoing degree greater than 0) with Greek letters.*

the same symbol). Nevertheless, our results either support, or can be easily extended, to consider general graphs. In order to give the clearer the better (two-dimensional) graph representations, in the following, we will use Greek symbols to label internal nodes and Latin symbols to label frontier nodes (those with outgoing degree zero).

For any given sequence of nodes $w_1, w_2, \ldots, w_k$ such that $(w_i, w_{i+1}) \in E$ for $1 \leq i < k$, we say that there exists a path from w_1 to w_k. We define the length of the path as the number of nodes in the sequence. A graph is possible to have more than one path between a pair of nodes u and v. Thus, let us denote the set of shortest paths from u to v by $((u,v))$. Also, let $|((u,v))|$ denote the length of the (possibly multiple) shortest path. Thus, the length of a non-existent path is infinite and $|((u,u))| = 1$. We define the *diameter* of a graph $g = (V,E,\mu)$ as the maximum distance among two connected nodes of the graph. More formally:

$$diameter(g) = \max_{u,v \in V} \{|((u,v))| \ : \ |((u,v))| < \infty\}$$

Given a graph $g = (V,E,\mu)$, the *subgraph of g rooted* in the node v with radius k is defined as $R_g(v,k) = (W,E',\mu')$ such that $W = \{u \in V \ : \ |((v,u))| \leq k\}$ and where $E' = E \cap (W \times W)$, that is, the set of edges restricted to the nodes in W. In the same way, μ' is the restriction of μ to the nodes in W. We extend this definition to consider, for any graph $g = (V,E,\mu)$, the *subgraph of g rooted* in the node v, denoted by $R_g(v) = (W,E',\mu')$ where $W = \{u \in V \ : \ |((v,u))| < \infty\}$ with the set E' and the labeling function μ' defined as above.

We now recall some definitions from multiset theory which will be used in the transition function of a new graph automata model. In the following, we will denote the set of naturals with $\mathbb{N}$.

For any given set D, a *multiset* over D is a pair $\langle D, f\rangle$ where $f : D \to \mathbb{N}$ is an enumeration function. That is, for any $a \in D$, the function $f(a)$ denotes the number of elements a in the multiset, and we say that a is in A, and write it $a \in A$, if and only if $f(a) \neq 0$. The size of a multiset is defined as the number of elements that a multiset contains. This number can be finite, in which case the multiset is finite. The size of a multiset M will be denoted by $|M|$. We are interested in the class of multisets whose size is equal to a constant n. That is, the class of all multisets $\langle D, f\rangle$ such that $\sum_{a\in D} f(a) = n$. In the sequel, we will denote this class by $\mathscr{M}_n(D)$.

We say that a multiset $A\langle D, f\rangle$ is *empty* if and only if, for all $a \in D$, $f(a) = 0$. In this way, for any pair of multisets $A = \langle D, f\rangle$ and $B = \langle D, g\rangle$, we say that $A = B$ if and only if, for all $a \in D$, $f(a) = g(a)$, and in the same way, A is a *subset* of B ($A \subseteq B$) if and only if, for all $a \in D$, $f(a) \leq g(a)$. Furthermore, let the sum of two multisets $A = \langle D, f\rangle$ and $B = \langle D, g\rangle$ (denoted by $A \oplus B$) be defined as the multiset $C = \langle D, h\rangle$ where for all $a \in D$, $h(a) = f(a) + g(a)$. Finally, we extend in a natural way the definition of *power set* to multisets, thus, for any multiset C, its power set is the set of all possible subsets of C and will be denoted by 2^C.

A very useful concept for dealing with multisets is the *Parikh mapping*. Formally, a Parikh mapping can be viewed as the application $\Psi : D^* \to \mathbb{N}^n$ where $D = \{d_1, d_2, \ldots, d_n\}$ and D^* is the set of strings over D. For any $x \in D^*$, this mapping is defined as $\Psi(x) = (\#_{d_1}, \#_{d_2}, \ldots, \#_{d_n})$ where $\#_{d_i}$ denotes the number of occurrences of d_j in x. Note that this allows to represent a multiset using whichever string with the correct Parikh mapping. We will do so in the following.

12.3 k-testable graph languages

Testable and testable in the strict sense languages [14] are defined by a vector (I, S, F) which represents those structures that are allowed to appear in the members of the language. These families have been defined over string and tree languages [8, 7]. In this section, we extend the definition to consider graph languages.

Given a typed alphabet Σ_τ^σ and the corresponding set of graphs over it $\mathscr{G}(\Sigma_\tau^\sigma)$, for any $g = (V, E, \mu) \in \mathscr{G}(\Sigma_\tau^\sigma)$, let us define the *k-testability vector* $T_k(g) =$

$(I_{k-1}(g), P_k(g), F_{k-1}(g))$ where:

$$I_{k-1}(g) = \left\{R_{\hat{g}}(v, k-1) \ : \ v \in V^0(g)\right\}$$
$$P_k(g) = \{R_{\hat{g}}(v,k) \ : \ v \in V, \ diameter(R_g(v)) \geq k\}$$
$$F_{k-1}(g) = \{R_{\hat{g}}(v,k-1) \ : \ v \in V, \ diameter(R_g(v)) \leq k-1\}$$

Note that $P_k(g) = \emptyset$ if $diameter(g) < k$. Some examples are given below.

Example 1. *Given the graph in Figure 12.1, Figure 12.2 shows the components of the* 3*-testability vector. It is worth noting that the nodes of the graphs in each component of the k-testability vector are labeled with the extended function* $\hat{\mu}$*. For each node, this extended labeling function depicts what the neighborhood was in the* mother *graph. For instance, note that the nodes labeled* a_0^2 *in Figure 12.2 do not always have two incoming edges. The extended labeling allows relevant structural information to be dealt with in a straightforward way. This labeling will play an important role in our grammatical inference algorithm.*

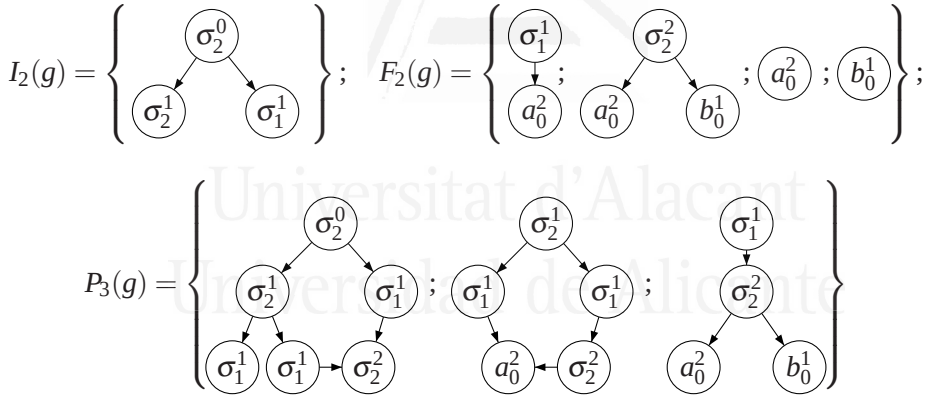
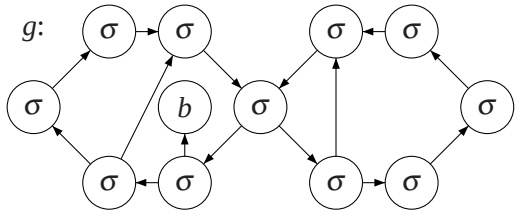


Figure 12.2: *The components of the* 3*-testability vector for the graph in Figure 12.1 are shown.*

It is not necessary for the graph to be acyclic in order to obtain the k-testability vector. An example is shown in Figure 12.3. The 2*-testability vector is also shown in the same figure. Note that, in this case, the set* $I_1(g)$ *is empty because the set* $V^0(g)$ *is empty.*



$$I_1(g) = \emptyset; \quad F_1(g) = \left\{ \left(b_0^1\right) \right\};$$

$$P_2(g) = \left\{ \begin{pmatrix} \sigma_1^1 \\ \sigma_1^2 \end{pmatrix}; \begin{pmatrix} \sigma_1^2 \\ \sigma_2^2 \end{pmatrix}; \begin{pmatrix} \sigma_2^2 \\ \sigma_2^1 \end{pmatrix}; \begin{pmatrix} \sigma_2^1 \\ \sigma_2^1 \end{pmatrix}; \begin{pmatrix} \sigma_2^1 \\ \sigma_1^2 \end{pmatrix}; \begin{pmatrix} \sigma_2^1 \\ \sigma_1^1 \end{pmatrix}; \begin{pmatrix} \sigma_1^1 \\ \sigma_1^1 \end{pmatrix}; \begin{pmatrix} b_0^1 \\ \sigma_2^1 \end{pmatrix}; \begin{pmatrix} \sigma_1^1 \\ \sigma_1^1 \end{pmatrix} \right\}$$

Figure 12.3: *Directed graph and its corresponding* 2*-testability vector.*

The functions I_k, F_k and P_k can be extended in a natural way to a set G of graphs:

$$I_k(G) = \{I_k(g) : g \in G\}; \quad P_k(G) = \{P_k(g) : g \in G\}; \quad F_k(G) = \{F_k(g) : g \in G\}$$

For any pair of graphs g and g', it is possible to define an equivalence relation $\equiv_k$ over $\mathscr{G}(\Sigma_\tau^\sigma)$ taking into account the k-testability vector, where $g \equiv_k g'$ if and only if $T_k(g) = T_k(g')$. This equivalence relation is key in defining the classes of k-testable and k-testable in the strict sense graph languages.

Definition 1. *A graph language G is* k-testable *($k \geq 2$) if it results from the union of a finite number of equivalence classes of the relation $\equiv_k$.*

Intuitively, and in the same way it happens with string or tree languages, if g is a graph in a k-testable graph language, then, every graph g' such that it has the same k-testability vector than g is also in G.

Definition 2. *For any $k \geq 2$, a graph language G is* k-testable in the strict sense *(k-TSS) if there exist three finite sets of graphs (B,S,E) such that, $g \in G$ if and only if $I_{k-1}(g) \subseteq B$, $P_k(g) \subseteq S$ and $F_{k-1}(g) \subseteq E$.*

According the definition, and a shared feature of string and tree k-TSS languages, the membership to a k-TSS graph language of graphs with diameter

smaller than k depends on $F_{k-1}(g)$. This is because for those graphs $P_k(g) = \emptyset$ and $I_{k-1}(g) \subseteq F_{k-1}(g)$.

Note that, for a given set of graphs G, the k-testability vector defines a k-TSS language when the sets of graphs B, S and E are set to $I_{k-1}(G)$, $P_k(G)$ and $F_{k-1}(G)$, respectively. Let this language be denoted by $L_k(G)$. We now prove some results related to this class of languages.

Lemma 1. *Let G be a finite set of graphs and $k \geq 2$, then $G \subset L_k(G)$.*

Proof. Let $g \in G$, trivially $I_{k-1}(g) \subseteq I_{k-1}(G)$, $P_k(g) \subseteq P_k(G)$ and $F_{k-1}(g) \subseteq F_{k-1}(G)$.

The language $L_k(G)$ is defined by the vector $T_k(G) = (I_{k-1}(G),\ P_k(G),\ F_{k-1}(G))$; therefore, $g \in L_k(G)$.

To prove that the inclusion is strict, note that any disconnected graph obtained by joining graphs in the set G will belong to the k-TSS language $L_k(G)$. This implies that every k-TSS graph language, except the empty one, are infinite. □

In string [8] and tree languages [7], it is proved that, when the value of k is greater than the maximum length (depth in trees) of the elements in a set S, then the k-TSS language obtained from that set (of strings or trees) equals S. This is not generally true when graph languages are taken into account. To enlighten this, please note first that, in this context, it is always possible to obtain new graphs belonging to the k-TSS language by *disconnected joint* of different graphs in the set S. Furthermore, it is possible in some cases to build new graphs taking into account the graphs rooted in the set of nodes with zero incoming arity. As an example, let consider a set of graphs containing only the graph g shown in Figure 12.4 (with $diameter(g) = 3$) and its corresponding 4-testable vector. Note that the graph g' shown in the same figure belongs to the language $L_4(\{g\})$.

Lemma 2. *For any set of graphs G and a given $k \geq 2$, the language $L_k(G)$ is the smallest k-TSS language that contains G.*

Proof. We will prove that, for any given k-TSS language T, if $G \subset T$, then $T \not\subset L_k(G)$.

Let (B_T, S_T, E_T) be the sets that define the language T. Let us suppose that $T \subset L_k(G)$; then there is a graph $g \in L_k(G) - T$. On the one hand, $g \in L_k(G)$, then $I_{k-1}(g) \subseteq I_{k-1}(G)$, $P_k(g) \subseteq P_k(G)$ and $F_{k-1}(g) \subseteq F_{k-1}(G)$. On the other hand, $g \notin T$, therefore, $I_{k-1}(g) \not\subseteq B_T$ or $P_k(g) \not\subseteq S_T$ or $F_{k-1}(g) \not\subseteq E_T$.

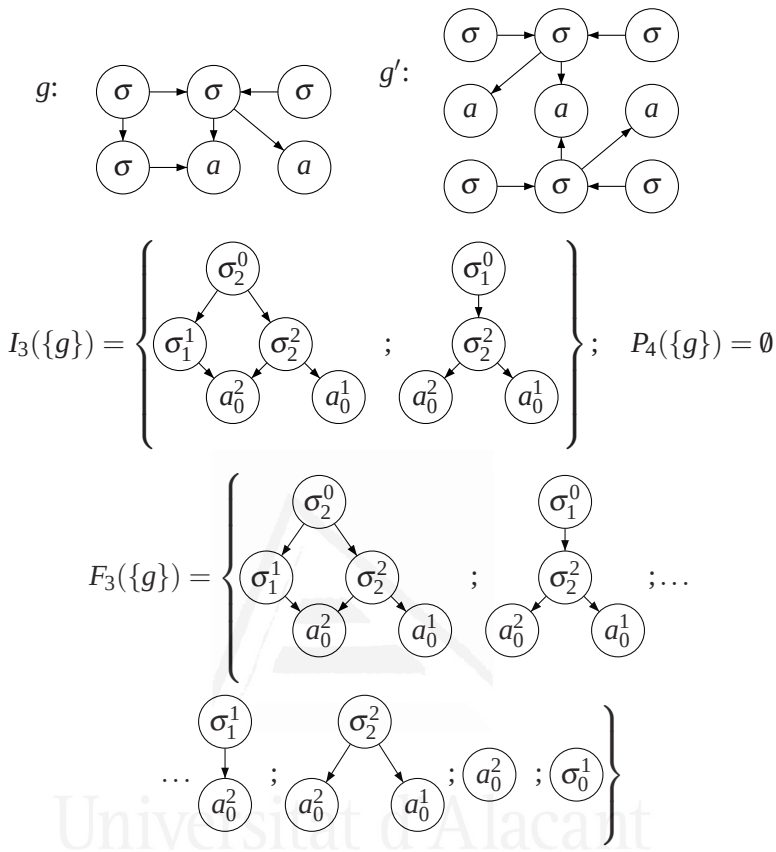


Figure 12.4: *Given the graphs g and g′ note that the graph g′ belongs to the language $L_4(\{g\})$. The components of the 4-testability vector are also shown.*

In other words, there are some structures in the k-testability vector of $L_k(G)$ that are not present in the one of T. From this, it follows that there exists a graph g' such that $g' \in G$ and $g' \notin T$; therefore, $G \not\subseteq T$, which contradicts the previous assumption. □

Lemma 3. *Let G and G' be two sets of graphs and $k \geq 2$. If $G \subseteq G'$, then $L_k(G) \subseteq L_k(G')$.*

Proof. It is easy to see that, if $G \subseteq G'$, then $I_{k-1}(G) \subseteq I_{k-1}(G')$, $P_k(G) \subseteq P_k(G')$

and $F_{k-1}(G) \subseteq F_{k-1}(G')$. Therefore, $L_k(G) \subseteq L_k(G')$. □

Lemma 4. *For any set of graphs G and $k \geq 2$, $L_{k+1}(G) \subseteq L_k(G)$.*

Proof. We need to prove that, for every $g \in L_{k+1}(G)$, g is also in $L_k(G)$, in other words, we need to prove that for any $g \in L_{k+1}(G)$, $I_{k-1}(g) \subseteq I_{k-1}(G)$, $F_{k-1}(g) \subseteq F_{k-1}(G)$ and $P_k(g) \subseteq P_k(G)$ hold.

Note that the sets $I_{k-1}(G)$ and $F_{k-1}(G)$ can be obtained from the sets $I_k(G)$ and $F_k(G)$ as follows:

$$I_{k-1}(G) = I_{k-1}(I_k(G))$$
$$F_{k-1}(G) = F_{k-1}(F_k(G))$$

Concerning $P_k(G)$ and $P_{k+1}(G)$, for every graph $g \in L_{k+1}(G)$, we distinguish two cases:

- if $diameter(g) \leq k$, then $P_{k+1}(g) = \emptyset$, which is a subset of $P_k(G)$
- if $diameter(g) > k$, then $P_k(g) = P_k(P_{k+1}(g)) \subseteq P_k(P_{k+1}(G)) = P_k(G)$

Thus, as mentioned above, any graph fulfilling the conditions fixed by the $(k+1)$-testability vector also fulfills those fixed by the k-testability vector. Therefore, we conclude that $L_{k+1}(G) \subseteq L_k(G)$ □

In the next section, we propose a new model of graph automata for directed acyclic graphs and use this model to propose a grammatical inference algorithm.

12.4 Graph automata

The automata model we propose takes into account the work by [17]. We note here that our proposal is not able to process whole class of directed graphs but those without cycles. The transition function of the automata we propose takes into account a multiset which permits, in a natural way, the graphs to be processed without taking into account any order among the nodes except for the partial one induced by the directed edges.

Let us first note that the analysis of any graph is, in essence, similar to the analysis carried out in the context of tree languages, but taking into account that there does not exist any order among the siblings. Second, we also note that

this process can be carried out with polynomial complexity using a dynamic programming scheme similar to the proposed in the paper by [24]. In his paper, [24] proposes a polynomial algorithm to obtain an edit measure among two unordered trees.

Due to the few graph automata models proposed in the literature, we think it is interesting to provide the more general definition the better, thus, we define our model as non-deterministic.

Definition 3. *Let Σ_τ^σ be a typed alphabet where m denotes the maximum outgoing degree in $\widehat{\Sigma}$. A (non-deterministic) graph automaton for a language over Σ_τ^σ is defined as the tuple $GA = (Q, \widehat{\Sigma}, \delta, F)$, where Q is a finite set of states, $\widehat{\Sigma}$ is the extended alphabet of Σ_τ^σ, the set $F \subseteq Q$ contains the final states and δ is a set of transition functions defined as follows:*

$$\delta = \bigcup_{\substack{0 \le j \le m \\ j:\ \exists n > 0, \Sigma_j^n \neq \emptyset}} \delta_j$$

where each δ_j is defined as:

$$\delta_j : \widehat{\Sigma}_j^n \times \mathscr{M}_j(Q) \to 2^{\mathscr{M}_1(Q)}, \qquad 0 \le j \le m$$

and where $\mathscr{M}_j$ represents the class of multisets of size j as defined in Section 12.2.

We note that the definition of the domain of the transition function considers the extended alphabet instead of the original one. This allows the graph to be processed taking into account both the outgoing degree (the size of the multiset) and the incoming degree of the node (captured in the symbol of the extended alphabet). In order to extend the transition function to operate on graphs, the intuitive idea consist of a recursive analysis over each zero-incoming degree node. The multisets returned by these analysis are then summed. Formally, for any given graph g the function δ is extended as follows:

$$\delta : \mathscr{G}(\widehat{\Sigma}) \to 2^{\mathscr{M}(Q)}$$
$$\delta(g) = \delta(\hat{g}) = \bigoplus_{v_i \in V^0(\hat{g})} \delta(R_{\hat{g}}(v_i))$$

where:

$$\delta(R_g(v_i)) = \delta_m(\mu_{\hat{g}}(v_i), M_{i1} \oplus \ldots \oplus M_{im}) : \; M_{ij} \in \delta(R_g(w_j)),\; (v_i, w_j) \in E$$

For any graph g the extended version $\hat{g}$ is isomorphic, thus, there is no problem in reducing the parsing of a graph to its extended version.

Let us now define the language accepted by the automaton as follows:

$$L(GA) = \{g \in \mathscr{G}(\Sigma_\tau^\sigma) \; : \; \forall q \in \delta(\hat{g}),\; q \in F\}$$

Thus, any graph g is accepted by the automaton GA if and only if the extended transition function returns a multiset such that for every state q with non zero enumeration function, the state q is final. When necessary, we will refer to these multisets as final multisets.

Example 2. *Taking into account the graph shown in Figure 12.1 and the following automaton, a representation of the analysis is depicted in Figure 12.5. We recall that the strings in the transition function denote multisets according the Parikh function. Thus, the string q_1q_2 represents the multiset with one element q_1 and one element q_2.*

$$\begin{array}{r}
\delta(a_0^2, \lambda) = q_1 \\
\delta(b_0^1, \lambda) = q_2 \\
\delta(\sigma_2^2, q_1q_2) = q_1 \\
\delta(\sigma_1^1, q_1) = q_2 \\
\delta(\sigma_2^1, q_2q_2) = q_1 \\
\delta(\sigma_2^0, q_1q_2) = q_1, \textit{ where } q_1 \in F
\end{array}$$

12.5 Inference algorithm

We now propose Algorithm 12.1 to infer the class of k-TSS graph languages from positive presentation. The algorithm follows the same scheme used previously to infer k-TSS string or tree languages [8, 7]. The algorithm first establishes the set of states taking into account the graph structures of diameter $k-1$ in the k-testability vector of the input sample. The set of final states is also established. Then, the algorithm creates the transitions using the graphs in $F_{k-1}(G)$ and $P_k(G)$. Please note that the automaton output by the algorithm is deterministic. An example of run is given below.

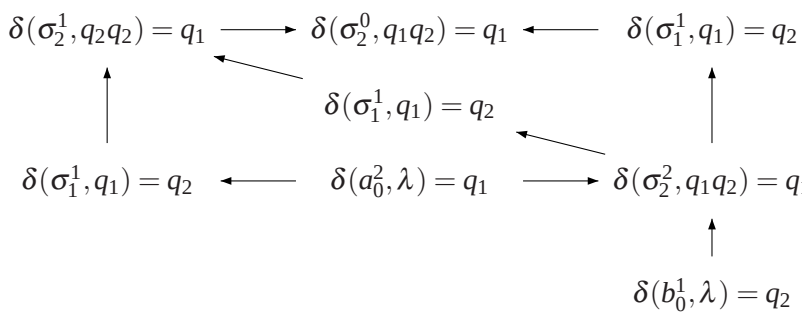


Figure 12.5: *Example of the parsing of the graph in Figure 12.1. The graph is recursively traversed to reach those nodes with zero outgoing degree. The arrows show the order of the parsing process once those nodes are reached.*

Example 3. *Let us consider $k = 2$, and the set G of graphs shown in Figure 12.6. The elements of the 2-testability vector are shown in Figure 12.7.*

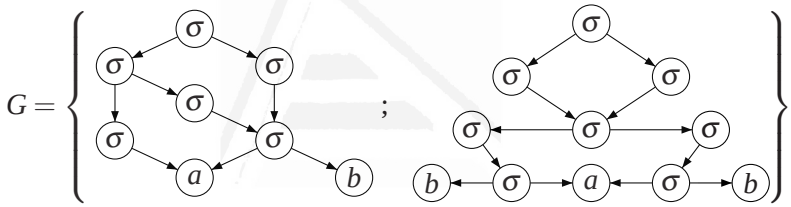
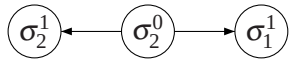


Figure 12.6: *Set of graphs example.*

First, the algorithm constructs the set of states taking into account $I_1(G)$, $F_1(G)$ and $I_1(P_2(G))$:

$$Q[(a_0^2)] = q_1;\ Q[(b_0^1)] = q_2;\ Q[(\sigma_2^2)] = q_3;\ Q[(\sigma_1^1)] = q_4;\ Q[(\sigma_2^1)] = q_5;\ Q[(\sigma_2^0)] = q_6$$

The algorithm obtains the set of final states, which is $F = \{q_6\}$. Then, the algorithm considers the graphs in $F_{k-1}(G)$. Note that the diameter of the graphs is 1. Therefore, the transitions $\delta(a_0^2, \lambda) = q_1$ and $\delta(b_0^1, \lambda) = q_2$ are added to the automaton. Note that λ denotes the empty string. The algorithm now processes the graphs in $P_k(G)$. As an example, let us consider the following graph in $P_2(G)$:



Algorithm 12.1 Grammatical inference algorithm from positive sample for the class of k-TSS graph languages.

Require: A set G of graphs. A value $k \geq 2$
Ensure: A graph automaton that recognizes the language $L_k(G)$
Method:
Compute $(I_{k-1}(G), P_k(G), F_{k-1}(G))$
Let Σ_τ^σ be the typed alphabet from G and $\widehat{\Sigma}_\tau^\sigma$ be the extended one
for $g \in \{I_{k-1}(G) \cup F_{k-1}(G) \cup I_{k-1}(P_k(G))\}$ **do**
Let $Q[g]$ be a new state related to g
end for
$F = \{Q[g] \ : \ g \in I_{k-1}(G)\}$
for all $g \in F_{k-1}(G)$ with $(v, w_i) \in E_g$, $v \in V^0(g)$, $1 \leq i \leq m$ **do**
$\delta_m(\mu(v), Q[R_g(w_1)] \ldots Q[R_g(w_m)]) = Q[g]$
end for
for all $g \in P_k(G)$ with $v \in V^0(g)$, $(v, w_i) \in E_g$, $1 \leq i \leq m$ **do**
$\delta_m(\mu(v), Q[R_g(w_1, k-1)] \ldots Q[R_g(w_m, k-1)]) = Q[R_g(v, k-1)]$
end for
return $(Q, \widehat{\Sigma}_\tau^\sigma, F, \delta)$
EndMethod:

$$I_1(G) = \left\{ \sigma_2^0 \right\}; \quad F_1(G) = \left\{ a_0^2 \ ; \ b_0^1 \right\};$$

$$P_2(G) = \left\{ \begin{array}{c} \sigma_2^0 \\ \sigma_2^1 \quad \sigma_1^1 \end{array} \ ; \ \begin{array}{c} \sigma_2^1 \\ \sigma_1^1 \quad \sigma_1^1 \end{array} \right.$$

$$\left. \begin{array}{c} \sigma_1^1 \\ \sigma_2^1 \end{array} \ ; \ \begin{array}{c} \sigma_1^1 \\ \sigma_2^2 \end{array} \ ; \ \begin{array}{c} \sigma_1^1 \\ a_0^2 \end{array} \ ; \ \begin{array}{c} \sigma_2^2 \\ a_0^2 \quad b_0^1 \end{array} \ ; \ \begin{array}{c} \sigma_2^0 \\ \sigma_1^1 \quad \sigma_1^1 \end{array} \ ; \ \begin{array}{c} \sigma_2^2 \\ \sigma_1^1 \quad \sigma_1^1 \end{array} \right\}$$

Figure 12.7: *Elements of the* 2*-testability vector for the graphs example.*

The algorithm takes into account the subgraphs of diameter $k-1$ *rooted at the nodes* below *the node* σ_2^0 *and the graph of diameter* $k-1$ *rooted at the node* σ_2^0. *Thus, the algorithm adds the transition* $\delta(\sigma_2^0, q_5q_4) = q_6$.

Once all the structures in the k-testability vector have been processed, the following automaton is obtained:

$\delta(a_0^2, \lambda) = q_1$	$\delta(\sigma_1^1, q_5) = q_4$
$\delta(b_0^1, \lambda) = q_2$	$\delta(\sigma_1^1, q_1) = q_4$
$\delta(\sigma_2^0, q_5q_4) = q_6$, *where* $q_6 \in F$	$\delta(\sigma_2^2, q_1q_2) = q_3$
$\delta(\sigma_2^1, q_4q_4) = q_5$	$\delta(\sigma_2^0, q_4q_4) = q_6$, *where* $q_6 \in F$
$\delta(\sigma_1^1, q_3) = q_4$	$\delta(\sigma_2^2, q_4q_4) = q_3$

As an example, Figure 12.8 shows two graphs that were not in the input set and that belong to the 2*-TSS graph language.*

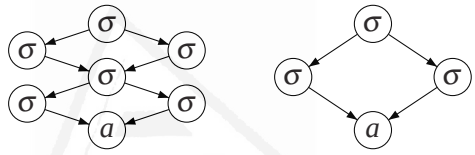


Figure 12.8: *Two graphs not provided in the input set that belong to the example* 2*-TSS language.*

We now prove that the algorithm identifies in the limit the class of *k*-TSS directed acyclic graph languages from positive presentation.

Theorem 1. *Algorithm 12.1 identifies the class of k-TSS graph languages from positive sample.*

Proof. We first prove that, given a set of graphs G, the algorithm returns a graph automaton GA that accepts the language $L_k(G)$. Note that, on the one hand, for any graph g, its membership to the language of the automaton output by Algorithm 12.1 implies the analysis of each of the p nodes with zero incoming arity. In order to accept the graph g, all these analysis should return a final multiset. On the other hand, the membership of any graph g to $L_k(G)$ implies that, among other criteria, for every node v in $V^0(g)$, $R_{\hat{g}}(v, k-1) \in I_{k-1}(G)$. Thus, for the sake of clarity, and without loss of generality, we will consider graphs with just one node with zero incoming degree.

- $L_k(G) \subseteq L(GA)$:

  We will prove by induction on the diameter of the graphs, that, if $g \in L_k(G)$, then $\delta(g)$ returns a final state.

  First, if $diameter(g) < k$ and $v \in V^0(g)$, then g is isomorphic to $\hat{g} = R_{\hat{g}}(v) \in I_{k-1} \cap F_{k-1}$, and the algorithm sets $\delta(R_{\hat{g}}(v)) = Q[R_{\hat{g}}(v, k-1)] = Q[R_{\hat{g}}(v)]$, which is a final state.

  Let us suppose that, for any graph g such that $diameter(g) = n$, it is fulfilled that $\delta(g) = Q[R_{\hat{g}}(v, k-1)]$ where v is in $V^0(g)$.

  Now let g be a graph with $v \in V^0(g)$, such that, for $1 \leq i \leq m$, there exists $(v, w_i) \in E$ and $diameter(R_g(w_i)) \leq n$, and where at least one of the graphs $R_g(w_i)$ has diameter n. Then, $\delta(R_{\hat{g}}(w_i)) = Q[R_{\hat{g}}(w_i, k-1)]$ for each i, and therefore:

$$\begin{aligned}\delta(g) = \delta(\hat{g}) =& \delta_m(\mu_{\hat{g}}(v), \delta(R_{\hat{g}}(w_1)) \oplus \delta(R_{\hat{g}}(w_2)) \oplus \ldots \oplus \delta(R_{\hat{g}}(w_m))) \\ =& \delta_m(\mu_{\hat{g}}(v), Q[R_{\hat{g}}(w_1, k-1)] Q[R_{\hat{g}}(w_2, k-1)] \ldots \\ & \ldots Q[R_{\hat{g}}(w_m, k-1)]\end{aligned}$$

  Note that there is an edge in g from v to each w_i. Thus, the resulting joint graph with all the $R_{\hat{g}}(w_i, k-1)$, where v is in $P_k(g)$, is such that $\delta(g) = Q[R_{\hat{g}}(v, k-1)]$. Note also that the state is final, because $R_{\hat{g}}(v, k-1)$ is in $I_{k-1}(g)$.

- $L(GA) \subseteq L_k(G)$:

  We will prove that, for any graph $g \in L(GA)$, it is fulfilled that $F_{k-1}(g) \subseteq F_{k-1}(G)$, $P_k(g) \subseteq P_k(G)$ and there is a final state q ($q \in I_{k-1}(g)$) such that: $\delta(g) = Q[q]$. We will prove the result by induction on the diameter of the graph.

  First, if $diameter(g) < k$ with $v \in V^0(g)$, then $\hat{g} \in F_{k-1}(g) \subseteq F_{k-1}(G)$, $P_k(g) = \emptyset$ and $R_{\hat{g}}(v, k-1)$ is in $I_{k-1}(G)$.

  Let us suppose by induction hypothesis that, for any graph $g \in L(GA)$ such that $diameter(R_g(w_i)) = n \geq k$, it is fulfilled that $F_{k-1}(g) \subseteq F_{k-1}(G)$, $P_k(g) \subseteq P_k(G)$ and $\delta(g) = Q[R_{\hat{g}}(v, k-1)]$.

  Now let g be a graph such that $v \in V^0(g)$, with $(v, w_i) \in E$ where $diameter(R_g(w_i)) \leq n)$ for all $1 \leq i \leq m$, with at least one of the graphs

$R_g(w_i)$ of diameter n. Therefore:

$$\begin{aligned}\delta(g) = \delta(\hat{g}) =& \delta_m(\mu_{\hat{g}}(v), \delta(R_{\hat{g}}(w_1)) \oplus \delta(R_{\hat{g}}(w_2)) \oplus \ldots \oplus \delta(R_{\hat{g}}(w_m))) = \\ =& \delta_m(\mu_{\hat{g}}(v), Q[R_{\hat{g}}(w_1, k-1)]Q[R_{\hat{g}}(w_2, k-1)] \ldots \\ & \ldots Q[R_{\hat{g}}(w_m, k-1)] = Q[R_{\hat{g}}(v, k-1)]\end{aligned}$$

where $R_{\hat{g}}(v,k) \in P_k(g)$ because $(v,w_i) \in E$ for all $1 \leq i \leq m$. Besides, $Q[R_{\hat{g}}(v,k-1)] = q$. Moreover:

$$F_{k-1}(g) = \bigcup_{1 \leq i \leq m} F_{k-1}(R_{\hat{g}}(w_i)) \subseteq F_{k-1}(G)$$

$$P_k(g) = \left(\{R_{\hat{g}}(v,k)\} \cup \bigcup_{1 \leq i \leq m} P_k(R_{\hat{g}}(w_i)) \right) \subseteq P_k(G)$$

Also, if $g \in L(GA)$, then $Q[R_{\hat{g}}(v,k-1)]$ is a final state. Therefore $R_{\hat{g}}(v,k-1)$ is in $I_{k-1}(G)$ and $g \in L_k(G)$.

Given the fact that, for any k given, the elements in the components of the k-testable vector are finite, we conclude that the proposed algorithm identifies the class of k-TSS directed acyclic graph languages. □

Finally, we note that our algorithm runs in polynomial time. Let us consider any input set of graphs G over Σ_τ^σ such that m denotes the greatest outgoing degree of the graphs nodes in G. Let also be $k \geq 2$. The time complexity to obtain each transition is bounded by $\mathcal{O}(m \cdot |\widehat{\Sigma_\tau^\sigma}| \cdot m^{k-1})$, that is, the biggest outgoing degree times the size of the alphabet times the size of the greatest subgraph that can be reduced to a state. The whole inference step implies, at most, the creation of as many transitions as the number of nodes of the graphs in G. Let n denote that number. Thus, the inference process is bounded by $\mathcal{O}(n \cdot |\widehat{\Sigma_\tau^\sigma}| \cdot m^k)$.

12.6 Conclusions and Future work

In this paper, we extend the well-known families of k-testable and k-TSS languages to directed-graph languages. To our knowledge, this is the first result that characterizes a class of graph languages taking into account the features of the

graphs instead of the structure of graph grammar rules. We also propose a model of graph automata that allows us to propose a polynomial time algorithm which identifies the subclass of directed acyclic *k*-TSS graph languages.

The definition of *k*-testable and *k*-TSS languages support general directed graph languages (those that may contain cycles), nevertheless, the automata model proposed, as well as the inference algorithm do not so, and are focused to directed acyclic graphs. The main problems to extend the results to general directed graphs are the need to establish a processing order and the accepting criterion (because both the zero-incoming and zero-outgoing sets of nodes may be empty).

Of course, both the definition of *k*-TSS graph grammars, and the algorithm to obtain, from any given (*k*-TSS or not) graph automaton, an equivalent graph grammar, are very interesting problems and should be addressed as future work.

Acknowledgments

Damián López is partially supported by the Spanish Ministerio de Economía y Competitividad under research project TIN2011-28260-C03-01. Jorge Calera-Rubio and Antonio-Javier Gallego-Sánchez thank the Spanish CICyT for partial support of this work through project TIN2009-14205-C04-01, the IST Programme of the European Community, under the PASCAL Network of Excellence, IST-2002-506778, and the program CONSOLIDER INGENIO 2010 - (CSD2007-00018).

Bibliography

[1] D. Berwanger and D. Janin. Automata on directed graphs: edge versus vertex marking. *LNCS*, 4178:46–60, 2006. Proceedings of 3rd International Workshop on Software Evolution Through Transformations.

[2] H. Blockeel and S. Nijssen. Induction of node label controlled graph grammar rules. In *Proceedings of 6th International Workshop on Mining and Learning with Graphs*, 2008.

[3] F. J. Branderburg and K. Skodinis. Finite graph automata for linear and boundary graph languages. *Theoretical Computer Science*, 332:199–232, 2005.

[4] A. Clark, R. Eyraud, and A. Habrard. A polynomial algorithm for the inference of context free languages. *LNAI*, 5278:29–42, 2008. Proceedings of ICGI-08.

[5] C. de la Higuera. *Grammatical inference. Learning automata and grammars*. Cambridge University Press, 2010.

[6] C. Costa Florêncio. Identification of hyperedge-replacement graph grammars. In *Proceedings of 7th International Workshop on Mining and Learning with Graphs*, 2009.

[7] P. García. Learning *k*-testable tree sets from positive data. Technical Report DSIC/II/46/1993, Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia, 1993. Available on: `http://www.dsic.upv.es/users/tlcc/tlcc.html`.

[8] P. García and E. Vidal. Inference of k-testable languages in the strict sense and application to syntactic pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:920–925, 1990.

[9] E. Jeltsch and H. J. Kreowski. Grammatical inference based on hyperedge replacement. *LNCS*, 532:461–474, 1991. 4th International workshop on graph grammars and their application to computer science.

[10] I. Jonyer, L. B. Holder, and D. J. Cook. Concept formation using graph grammars. In *Proceedings of the KDD Workshop on Multi-Relational Data Mining*, pages 71–792, 2002.

[11] R. Kosala, H. Blockeel, M. Bruynooghe, and J. Van den Bussche. Information extraction from documents using *k*-testable tree automaton inference. *Data & Knowledge Engineering*, 58:129–158, 2006.

[12] J. P. Kukluk, L. B. Holder, and D. J. Cook. Inference of node replacement recursive graph grammars. In *Proceedings of the Sixth SIAM International Conference on Data Mining*, 2006.

[13] D. López, J. M. Sempere, and P. García. Inference of reversible tree languages. *IEEE Transactions on System Man. and Cybernetics, Part B: Cybernetics*, 34(4):1658–1665, 2004.

[14] R. McNaughton. Algebraic decision procedures for local testability. *Math. Sysr. Theory*, 8(1):60–76, 1974.

[15] K. Nakamura. Incremental learning of context free grammars by bridging rule generation and search for semi-optimum rule sets. *LNAI*, 4201:72–83, 2006. Proceedings of ICGI-06.

[16] P. Peris, D. López, and M. Campos. IgTM: An algorithm to predict transmembrane domains and topology in proteins. *BMC - Bioinformatics*, 9: 367, 2008.

[17] A. Potthoff, S. Seibert, and W. Thomas. Nondeterminism versus determinism on finite automata over directed acyclic graphs. *Bull. Belg. Math. Soc.*, 1:285–298, 1994.

[18] J. R. Rico-Juan, J. Calera-Rubio, and R. C. Carrasco. Smoothing and compression with stochastic *k*-testable tree languages. *Pattern Recognition*, 38: 1420–1430, 2005.

[19] D. Ron, Y. Singer, and N. Tishby. On the learnability and usage of acyclic probabilistic finite automata. *Journal of Computer and System Sciences*, 56:133–152, 1998.

[20] G. Rozemberg, editor. *Handbook of graph grammars and computing by graph transformation*, volume 1. Word Scientific, 1997.

[21] Y. Sakakibara. Learning Context-Free Grammars from Structural Data in Polynomial Time. *Theoretical Computer Science*, 76:223–242, 1990.

[22] Y. Sakakibara. Efficient Learning of Context-Free Grammars from Positive Structural Examples. *Information and Computation*, 97:23–60, 1992.

[23] Y. Sakakibara. Grammatical inference in bioinformatics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1051–1062, 2005.

[24] K. Zhang. A constrained edit distance between unordered labelled trees. *Algorithmica*, 15:205–222, 1996.

Publicación 13

Structural Graph Extraction from Images

Antonio-Javier Gallego-Sánchez*, Jorge Calera-Rubio* and Damián López**

** Departamento de Lenguajes y Sistemas Informáticos, University of Alicante, Spain, {jgallego, calera}@dlsi.ua.es*

*** Departamento de Sistemas Informáticos y Computación, Technical University of Valencia, Spain, dlopez@dsic.upv.es*

Proceedings of the 9th International Conference on Distributed Computing and Artificial Intelligence (DCAI 2012), Series of Advances in Intelligent and Soft Computing, Springer-Verlag, vol. 151, pp 717-724, Salamanca, Spain (2012), ISSN: 1867-5662, ISBN: 978-3-642-28764-0

Structural Graph Extraction from Images

Antonio-Javier Gallego-Sánchez*, Jorge Calera-Rubio*
and Damián López**

* Departamento de Lenguajes y Sistemas Informáticos,
University of Alicante, Spain,
{*jgallego, calera*}*@dlsi.ua.es*
** Departamento de Sistemas Informáticos y Computación,
Technical University of Valencia, Spain,
dlopez@dsic.upv.es

Abstract

We present three new algorithms to model images with graph primitives. Our main goal is to propose algorithms that could lead to a broader use of graphs, especially in pattern recognition tasks. The first method considers the q-tree representation and the neighbourhood of regions. We also propose a method which, given any region of a q-tree, finds its neighbour regions. The second algorithm reduces the image to a structural grid. This grid is postprocessed in order to obtain a directed acyclic graph. The last method takes into account the skeleton of an image to build the graph. It is a natural generalization of similar works on trees [8, 12]. Experiments show encouraging results and prove the usefulness of the proposed models in more advanced tasks, such as syntactic pattern recognition tasks.

13.1 Introduction

Many fields take advantage of graph representations, these fields include but are not limited to: biology, sociology, design of computer chips, and travel related problems. This is due to the fact that graph are suitable to represent any kind of relationships among data or their components. The expressive power of graph primitives has been specially considered in pattern recognition tasks, in order to represent objects [7] or bioinformatics [13]. Therefore, the development of algorithms to handle graphs is of major interest in computer science.

In this paper we present three new methods to model images using graph primitives. The main goal we aim to achieve is to propose algorithms that could lead to a broader use of graphs. In that way, the work of [4] shows that this kind of contribution is suitable and interesting.

The first method we propose (Section 2) takes into account the neighbour paths within a q-tree [2]. We present algorithms that extend other works for neighbourhood calculations in spatial partition trees [5, 11]. These works generally calculate only the 4-neighbours of each region, or use two steps to calculate the corner neighbours. The proposed method completes these works with an estimation of equal complexity for the 8-neighbours at any level of resolution (equal, higher or lower level). Moreover, neighbour calculation can help to improve other applications, speeding up the process to locate the next element, such as in: image representation, spatial indexing, or efficient collision detection.

The second method (Section 3) considers a structural grid of the image which is then postprocessed to obtain the graph model. This method maintains those desiderable features of q-tree representation, that is: the resolution is parametrizable and allows to reconstruct the image.

The third method (Section 4) extends a similar method used on trees [12, 8]. The initial image is preprocessed to obtain the skeleton. This skeleton is then traversed to build a graph that summarizes the core structure of the image.

13.2 Neighbourhood Graphs

Using the tree defined by a q-tree (see Fig. 13.1) and the proposed algorithm for neighbourhood calculation (see section 13.2.1), a directed acyclic graph (dag) can be extracted to represent the sample and to provide information such as: neighbourhood, structural traversal, resolution or thickness, in addition to allow its reconstruction.

The graph is created walking top-down the tree, and from left to right. For each nonempty node, its 8-neighbours are calculated. It is important that: 1) Do not add arcs to processed nodes. 2) If it has to create an arc from a non-pointed node to one that is already pointed: then the direction of the arc is changed. These criteria ensures that the graph is acyclic.

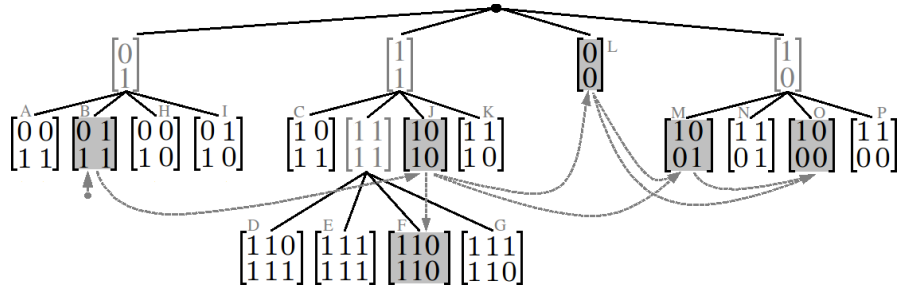


Figure 13.1: *Tree obtained from the q-tree of the Fig. 13.2(c), and its corresponding neighbour graph.*

13.2.1 Calculating Neighbours

Binary encoding is used to encode the position of partitions whithin the image and their corresponding traversal path down the q-tree [5]. Each dimension is encoded with 1 bit coordinate, describing the direction as positive (1) or negative (0). A location array is defined using this encoding (Fig. 13.2(a)). In q-trees, this array has two rows: horizontal and vertical coordinates. Positive directions (up and right) are represented by a 1, and the negative directions (down and left) by 0 (Fig. 13.2(b)). In general, the location array associated with a given node is defined as the location array of his father, but adding on the right a new column with its own encoding. For each new partition, the origin of the reference system is placed in the center of the area where the subdivision is done. In Fig. 13.2(c), the second level node *K* is represented by the $\left[\begin{smallmatrix}1\\1\end{smallmatrix}\right]$ of his father, and then adding the $\left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]$ for its own level.

Algorithm

To calculate the neighbours of a given node, the location array described above is used. The proposed algorithm distinguishes two cases: face neighbours (two regions share more than one point, such as *A* with *B* in Fig. 13.2(c)), and corner neighbours (they share one point, *A* with *I* in Fig. 13.2(c)).

Face Neighbours

To calculate the face neighbour of a given node, the algorithm needs to know its location array (*M*), and the dimension *dim* and the direction *dir* in which the

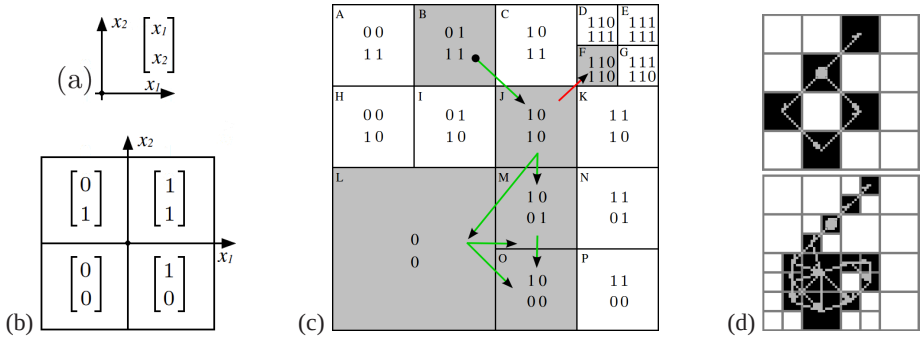


Figure 13.2: *(a) Reference system and location array, (b,c) Q-trees with the corresponding location arrays, (d) Graphs extracted using different truncate levels.*

neighbour has to be calculated. The algorithm (see Sec. 13.2.1) visits each bit of the location array (M) in the given dimension from right to left, and negates the bits until the result is equal to the explored direction (1: positive, 0: negative).

As an example, the horizontal neighbour of I (Fig. 13.2(c)) in the positive direction (1) is calculated. The algorithm begins with the location array and negates each bit in the top row (dim 1) from right to left (indicated with a bar) until reaching the halt condition: the result is equal to the explored direction.

$$I = \begin{bmatrix} 0 & \overline{\mathbf{1}} \\ 1 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} \overline{\mathbf{0}} & 0 \\ 1 & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} = J$$

Corner Neighbours

In the case of corner neighbours, the algorithm has to modify the two dimensions of the location array. As for the face neighbours, it begins with the location array M, but in this case it receives as input an array dir with both horizontal and vertical directions. Below is the complete algorithm to compute face and corner neighbours:

```
function GetNeighbour( M, dim, dir )
  level := |M[0]|   // Level is equal to the number of columns
  if( |dir| == 1 ) {      // Face neighbours
    do {
      M[dim][level] := ! M[dim][level]
      level := level − 1
    } while( level ≥ 0 ∧ M[dim][level] != dir )
  } else {      // |dir| == 2 → Corner neighbours
    flag1 := false ; flag2 := false
    do {
      if( !flag1 )
        M[0][level] := ! M[0][level]
        if( M[0][level] == dir[0] )   flag1 := true
      if( !flag2 )
        M[1][level] := ! M[1][level]
        if( M[1][level] == dir[1] )   flag2 := true
      level := level − 1
    } while( level ≥ 0 ∧ ( !flag1 ∨ !flag2))
  }
  return M
```

As example, the corner neighbours of N (Fig. 13.2(c)) are calculated:

$$N = \begin{bmatrix} 1 & \overline{\mathbf{1}} \\ 0 & \mathbf{1} \end{bmatrix} \Rightarrow \begin{bmatrix} \underline{1} & 0 \\ \mathbf{0} & 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} = J \qquad \| \qquad N = \begin{bmatrix} 1 & \overline{\mathbf{1}} \\ 0 & \mathbf{1} \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} = O$$

Type of Nodes

It is important to differentiate the type of a node within the structure (Fig. 13.1). The function *TypeOfNode(M)* returns this type, it can be: *leaf-node* (the node exists and has no children), *inner-node* (the node exists and has children) or *no-node* (the node does not exist). If the location array is a *leaf-node* the algorithm ends. If it is an *inner-node*, then the algorithm has to calculate the higher-level neighbours (Sec. 13.2.1). If the node does not exist, the algorithm has to calculate the lower-level neighbours (Sec. 13.2.1).

Lower Resolution Neighbours

In this case, the algorithm first calculates the same resolution neighbour using *M := GetNeighbour(M, dim, dir)*; and then eliminates the final column/s (on the right side) until it finds a *leaf-node*:

```
function GetLowerLevel( M )
  do {
    M_{2×n} ← M_{2×n−1}
  } while( TypeOfNode( M ) == no-node )
  return M
```

Example: region D in Fig. 13.2(c).

$$D = \begin{bmatrix} 1 & 1 & \overline{\mathbf{0}} \\ 1 & 1 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & \overline{\mathbf{1}} & 1 \\ 1 & 1 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \Rightarrow \textit{no-node} \Rightarrow \begin{bmatrix} 1 & 0 & \not{1} \\ 1 & 1 & \not{1} \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} = C$$

Higher Resolution Neighbours

The algorithm first calculates the neighbour at the same level of resolution using $M := GetNeighbour(M, dim, dir)$. If M corresponds to an *inner-node*, then the higher-level neighbour/s is/are calculated. It is worth to note that, face neighbours may have two or more neighbours (e.g. L with H and I in Fig. 13.2(c)), whereas corner neighbours only have one neighbour. In this process, columns are added on the right side of the matrix M until reaching a *leaf-node*. These columns are created in the opposite direction of the dimension/s of interest. See the pseudocode below:

```
function GetHigherLevel( M, dim, dir )
  L := ⟨∅⟩      // Result set of higher-level neighbours
  if( |dir| == 1 ) {      // Face neighbours
    for( i = 1; i ≤ 2, ++i) {
      N_i[dim][0] :=!dir  ;  N_i[ ! dim][0] := i − 1
      M_i := M ⊕ N_i
      if( TypeOfNode( M_i ) == leaf-node )  L ← M_i
      else  L ← GetHigherLevel( M_i, dim, dir )
    }
  } else {      // |dir| == 2 → Corner neighbours
    do {
      N[0][0] :=!dir[0]  ;  N[1][0] :=!dir[1]
      M := M ⊕ N
    } while( TypeOfNode( M ) != leaf-node )
    L ← M
  }
  return L
```

The symbol $\oplus$ denotes the concatenation of a matrix M with a vector N:

$$M \oplus N = \begin{bmatrix} x_0^1 & x_1^1 \\ x_0^2 & x_1^2 \end{bmatrix} \oplus \begin{bmatrix} y^1 \\ y^2 \end{bmatrix} = \begin{bmatrix} x_0^1 & x_1^1 & y^1 \\ x_0^2 & x_1^2 & y^2 \end{bmatrix}$$

Below is an example of higher-level neighbours calculation (see Fig. 13.2(c)):

$$C = \begin{bmatrix} 1 & \overline{\mathbf{0}} \\ 1 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \Rightarrow \textit{inner-node} \Rightarrow \left\langle \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} = D; \quad \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix} = F \right\rangle$$

Neighbourhood of Border Regions

When a region is on the border of the structure, do not present neighbours on that border side. In this case, the algorithm ends without reaching the halt condition. E.g. node A (horizontal-left):

$$A = \begin{bmatrix} 0 & \overline{\mathbf{0}} \\ 1 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} \overline{\mathbf{0}} & 1 \\ 1 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \Rightarrow \textit{border}$$

13.3 Structural Grid Graph

We here propose an algorithm to model images using graph primitives. The main goal of this new proposal is to maintain the better properties of q-trees but introducing higher variability in the incoming and outgoing degrees. Thus, the properties to fulfill are: 1) The model should consider the resolution as a parameter, allowing higher and lower resolution representations. 2) The representation should contain enough information to reconstruct the original image.

Without loss of generality, we will consider directed acyclic graphs and two colour images: *background* and *foreground* colours. Starting from the top-left corner, the algorithm divides the image into squared regions of a given size k. Those regions with foreground colour will be considered the nodes of the graph. The edges are defined from a foreground region (node) to those adjacent-foreground regions to the east, south and southeast. Figure 13.3 shows an example.

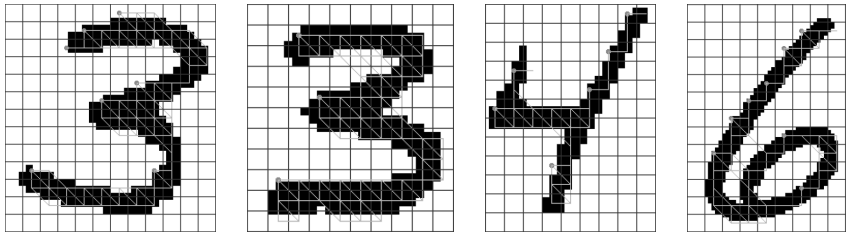


Figure 13.3: *Some examples of the grid graph modeling are shown. Those regions with at least 20% black pixels are considered foreground. Root nodes are marked in gray.*

13.4 Skeleton Graphs

The last method models the graph using the skeleton that defines the shape of the figure. It is a natural extension of the algorithms proposed for trees in [8, 12]. First, a thinning process is applied to the image [1], and then the graph is generated as:

1. The top left pixel of the skeleton is chosen as the root of the graph.

2. Every node in the graph will have as many children as unmarked neighbour pixels has the current pixel. For each child, an arc is created following that direction until one of the following criteria is true:

   (a) The arc has the maximum fixed parameter size (window parameter).

   (b) The pixel has no unmarked neighbours (terminal node).

   (c) The pixel has more than one unmarked neighbour (intersection).

   (d) The pixel is marked (existing node).

3. A new node is assigned to every end of arc pixel obtained by the previous step (a,b,c), or it is joined with the corresponding node (d). If the node has unmarked neighbours, then go to step 2, otherwise it terminates.

Figure 13.4: *Skeleton and graph results.*

13.5 Experimentation

This section aims to demonstrate the usefulness of the proposed models. For this reason, it uses simple classification methods, without stochastic layer and using only the structure of graphs (without labels). Results prove the correct separation of classes, in addition to be a baseline for future research.

To perform the experiments, 5 thousand images of digits (10 classes), with resolution of 64x64 pixels, from the NIST dataset are used. Graphs have been extracted using the three proposed methods, but varying the parameters (truncate level, size of cells, and window size) (see Fig. 13.5). Six methods are used to perform the classification: First, a feature vector is calculated using the graph degree distribution [10], and then classified using Naive Bayes, Random Forest, SVM and k-NN [6]. The 75% of the dataset is used for training and the rest for test. A preliminar error correcting distance is also defined to test the proposed methods. It is a simple algorithm with linear complexity. For this distance, the classification is performed using leaving-one-out. A semi-structural method of Flow Complexity is also used [3]. It analyzes the structure of the graph based on feature selection via spectral analysis and complexity.

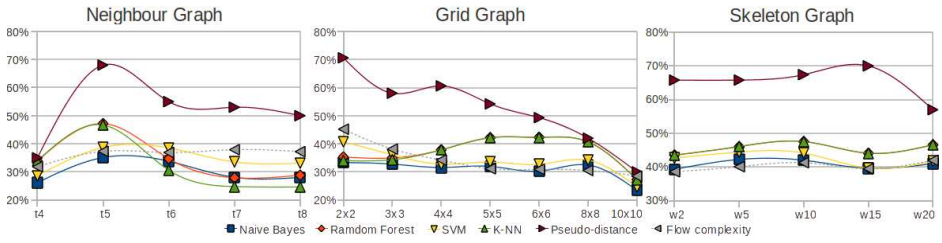


Figure 13.5: *Results using different classification methods.*

13.6 Conclusions

This paper presents three methods for extracting graphs from images. The obtained graphs introduce a variability that should create new characteristic features in the modelling of the classes of a pattern recognition task, and thus, it should ease a syntactic approach. Experiments show encouraging results which prove the usefulness of the proposed models, the correct separation of classes, in addition to be a baseline for future work.

The first method creates neighbourhood paths or segmentations within the q-tree, providing more data and creating new possibilities of operations and applications. The second algorithm maintains the features of q-tree representation, that is: the resolution is parametrizable and allows to reconstruct the original image. The third method takes into account the skeleton of the shape to build the graph, summarizing the core structure of the image.

Acknowledgments

This work is partially supported by Spanish MICINN (contract TIN2011-28260-C03-01, contract TIN2009-14205-C04-C1) and CONSOLIDER-INGENIO 2010 (contract CSD2007-00018).

Bibliography

[1] J. M. Cychosz. Thinning algorithm from the article: Efficient binary image thinning using neighbourhood maps. *Graphics Gems IV. Academic Press*, p. 465–473, 1994.

[2] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational geometry*, pages 291–306. Springer-Verlag, 2000.

[3] F.Escolano, D.Giorgi, E.R.Hancock, M.A.Lozano, and B.Falcidieno. Flow complexity: Fast polytopal graph complexity and 3d object clustering. *GbRPR*, 2009.

[4] M.Flasinski, S.Myslinski. On the use of graph parsing for recognition of isolated hand postures of Polish Sign Language. *Pattern Recognition*, 43:2249–2264, 2010.

[5] M. Goodchild. Quadtree algorithms and spatial indexes. *Technical Issues in GIS, NCGIA, Core Curriculum*, (37):5–6, 1990.

[6] M.Hall, E.Frank, G.Holmes, B.Pfahringer, P.Reutemann, I.H.Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[7] J. Liu, M. Li, Q. Liu, H. Lu, and S. Ma. Image annotation via graph learning. *Pattern Recognition*, 42:218–228, 2009.

[8] D. López and I. Piñaga. Syntactic pattern recognition by error correcting analysis on tree automata. *Advances in Pattern Recognition*, LNCS 1876, p. 133–142, 2000.

[9] R. G. Luque, J. ao L.D. Comba, and C. Freitas. Broad-phase collision detection using semi-adjusting bsp-trees. In *ACM i3D*, pages 179–186, 2005.

[10] M.E.J.Newman. The structure and function of complex networks. *SIAM*, 45. 2003.

[11] J. Poveda and M. Gould. Multidimensional binary indexing for neighbourhood calculations in spatial partition trees. *Comput. Geosci.*, 31(1):87–97, 2005.

[12] J.R.Rico-Juan, L.Micó. Comparison of AESA and LAESA search algorithms using string and tree edit distances. *Pattern Recognition Letters*, 24:1427–1436, 2003.

[13] H. Shin, K. Tsuda, and B. Schölkopf. Protein functional class prediction with a combined graph. *Expert Systems with Applications*, 36:3284–3292, 2009.