Capítulo VI. Pruebas del sistema.

En este capítulo se tocarán los problemas con los que se enfrentó durante el desarrollo del sistema así como las correcciones que se tuvieron que hacer una vez terminada la implementación del mismo. Finalmente, se presentarán algunas comparaciones entre los resultados obtenidos por el sistema antes y después de las correcciones realizadas.

6.1 Problemas y soluciones.

Los problemas que se presentaron durante el desarrollo del sistema en su mayoría fueros fáciles de solucionar, debido a que no correspondían directamente al objetivo de la tesis, es decir, fueron problemas de diseño de las interfaces de usuario.

Uno de los problemas con los que se lidió fue al momento de implementar una herramienta de pruebas para los algoritmos, esta herramienta es una Frame para el tratamiento directo con imágenes y se presentó el problema debido a que se comenzó trabajando con la versión 1.3 del JDK la cual no generaba los resultados esperados y fue necesario cambiar a la versión 1.4. Una vez instalada la nueva versión del JDK, se obtuvieron mejores resultados.

Posteriormente, cuando se implemento la interfase final del sistema, la cual es un servlet en el que el usuario puede subir imágenes al servidor, se presentó otra complicación, esta es que el cliente decida subir una imagen al servidor con un nombre que ya exista. Cuando esto sucedía el servidor perdía la imagen con la que trabajaría el cliente y para solucionarlo se modificó la base de datos donde el campo de nombre se definió como la

llave primaria de la tabla, con esto, la última imagen que se sube al servidor es la que se mantiene y la que tenía el mismo nombre es la que se pierde.

Otra dificultad al momento de trabajar con las interfaces fue cuando el usuario sube al servidor más de una imagen o decide trabajar con imágenes guardadas en el servidor en desorden, es decir, la interfaz para subir más de una imagen, véase la figura 5.3, y la interfaz para trabajar con imágenes previamente guardadas, véase figura 5.4, cuentan con varios campos, cuando se llenaban estos campos dejando uno o más libres entre los seleccionados el sistema era incapaz de determinar el numero de imágenes seleccionadas, para lo cual se utilizó un vector que contiene las imágenes seleccionadas por el usuario en orden.

Cuando se trabajó con las interfaces, se presentó otro problema el cual consistía en ingresar atributos extra para los diferentes operadores, como es el caso de umbrales, y el usuario no los ingresaba. Este problema fue muy sencillo puesto que sólo se dieron valores de inicio a los campos de los atributos, éstos valores son –1, es decir valores no validos para las funciones, cuando el operador recibe el valor –1 utiliza un valor por default.

Otro problema que se detectó fue en el módulo base de todo método, el cual convierte una imagen de color en una matriz de intensidad de escala de grises y producía un resultado demasiado claro y con cierto ruido, por lo que modificó la fórmula de conversión que era $\frac{(r+g+b)}{3}$ por la fórmula de Craig, la cual es la más utilizada para este tipo de operaciones y por ello se le considera como un estándar, la formula de Craig es la siguiente $r*0.3+g*0.59+b*0.11$, donde r corresponde a la componente roja, g a la componente verde y b a la componente azul [Fanning, 2002]. La mejoría en los resultados se puede

observar en las siguientes figuras y proporciona un resultado más claro, lo que mejoró el funcionamiento de algunos métodos.

Figura 6.1 Imagen original

Figura 6.2 Conversión inicial

Figura 6.3 Conversión final.

Finalmente, hubo un problema con el algoritmo de detección de puntos esquina (SUSAN), en el que no se detectaban figuras circulares, este algoritmo consta de una forma de detección de umbrales automático. Mediante pruebas con dichos umbrales de comprobó que depende directamente de ellos el número de puntos esquina detectados por lo que se decidió darle al usuario la oportunidad de modificarlos para obtener un resultado satisfactorio.

6.2 Pruebas y correcciones.

Una vez implementado el sistema se hicieron pruebas con varias imágenes de diferentes tamaños, diferente cantidad de objetos por imagen y diversas circunstancias, entre las que destacan movimientos lineales, movimientos rotacionales en más de dos imágenes, oclusión y presencia de ruido.

Durante todas estas pruebas se obtuvo un tiempo de ejecución de cada método de estimación de movimiento, posteriormente se calculó un promedio de estos tiempos para generalizar su eficiencia. Los tiempos de ejecución generales de cada método se presentan en forma de tabla en la tabla 6.1 y de forma gráfica en las figuras 6.1, 6.2 y 6.3 clasificadas cada una en los tamaños de imágenes procesadas. Cabe mencionar que los tiempos son solamente de ejecución del método en estudio, es decir, los tiempos de carga de la imagen y de desplegado de resultado no están contemplados. Los tiempos fueron medidos en una Computadora personal con procesador Intell Celeron 2.0 Ghz, 256 Mb en RAM y 32 Mb en memoria de video.

Algoritmo	Tamaño de imagen	Tiempo de procesamiento en milisegundos
Algoritmo 1(susan)	100X100 píxeles	1125
Algoritmo 2(sobel)	100X100 píxeles	625
Algoritmo 2(prewitt)	100X100 píxeles	610
Algoritmo 2(roberts)	100X100 píxeles	593
Algoritmo 2(bloques)	100X100 píxeles	594
Algoritmo 1(susan)	200X200 píxeles	2895
Algoritmo 2(sobel)	200X200 píxeles	906
Algoritmo 2(prewitt)	200X200 píxeles	1281
Algoritmo 2(roberts)	200X200 píxeles	1406
Algoritmo 2(bloques)	200X200 píxeles	1234
Algoritmo 1(susan)	300X300 píxeles	5625
Algoritmo 2(sobel)	300X300 píxeles	1484
Algoritmo 2(prewitt)	300X300 píxeles	1875
Algoritmo 2(roberts)	300X300 píxeles	1687
Algoritmo 2(bloques)	300X300 píxeles	1843

Tabla 6.1 Tiempos de ejecución

El contenido de la tabla 6.1 puede verse de manera más clara y representativa en las siguientes figura, las cuales fueron clasificadas según el tamaño de imágenes que procesan, las figuras se muestran a continuación:

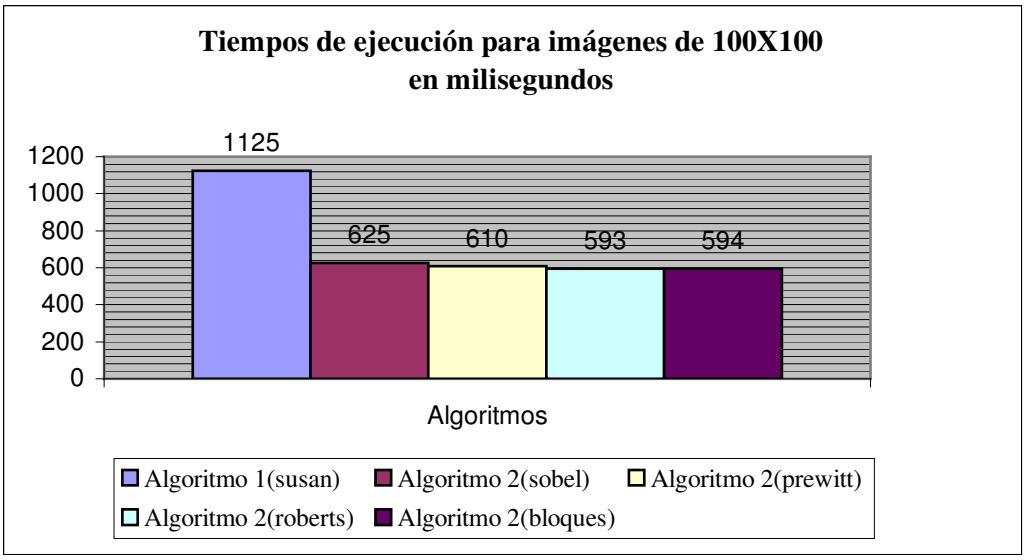

Figura 6.4 Comparación de algoritmos en imágenes de 100X100 píxeles.

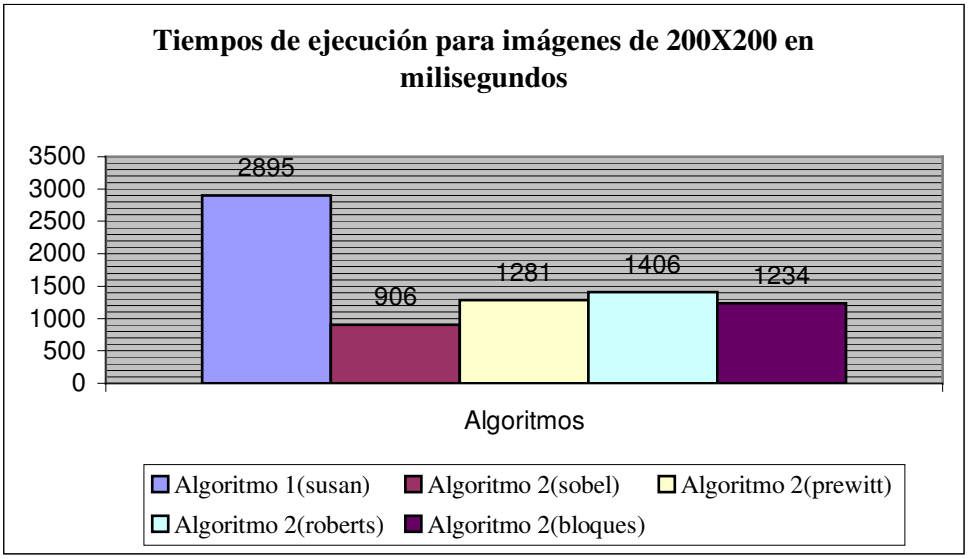

Figura 6.5 Comparación de algoritmos en imágenes de 200X200 píxeles.

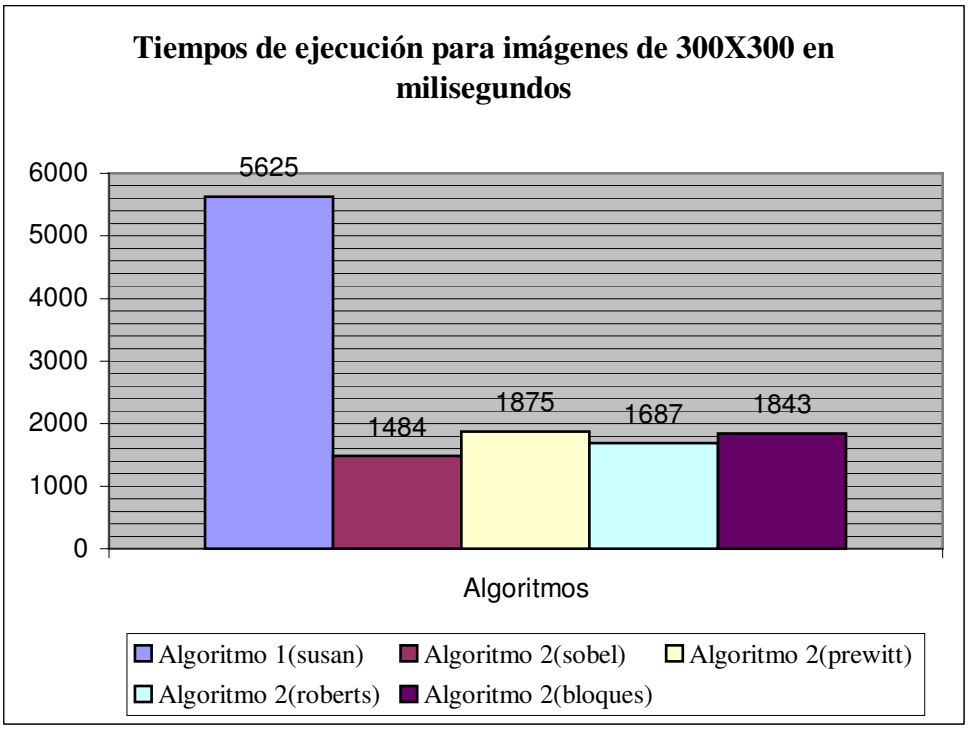

Figura 6.6 Comparación de algoritmos en imágenes de 300X300 píxeles.

Por medio de las pruebas mencionadas con anterioridad pude ver algunas fallas del sistema que tenían que ser mejoradas. También por medio de las pruebas y de las figuras de los tiempos de ejecución se tomó la decisión de que método de extracción de bordes se utilizaría para el segundo método de detección de movimiento.

Como se mencionó anteriormente en este capítulo, surgió un problema con el algoritmo de detección de puntos esquina SUSAN el cual no detectaba objetos circulares, mediante las pruebas se detectó que la cantidad de puntos detectados por el algoritmo dependía directamente de los umbrales *g* y *t* de dicho algoritmo. La corrección realizada para solucionar el problema fue hacer que dichos umbrales no tuvieran valores fijos, a cambio se permitió al usuario proporcionar dichos valores. Las pruebas resaltaron el hecho

que mientras más grande fuera el valor del umbral *g* el algoritmo detectaba más puntos hasta detectar los bordes del objeto. Las siguientes figuras muestran algunos resultados obtenidos al variar los valores del umbral *g*.

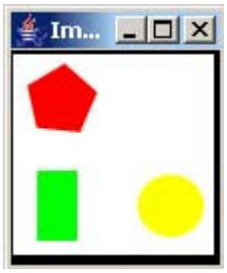

Figura 6.7 Imagen original

Figura 6.8 SUSAN original

Figura 6.9 SUSAN t=18

Figura 6.10 SUSAN t=19

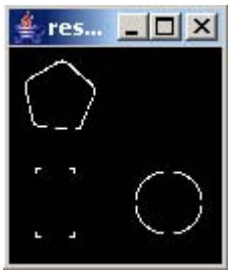

Figura 6.11 SUSAN t=22

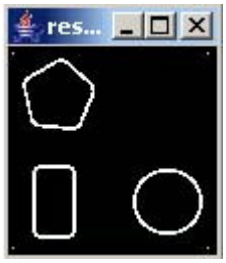

Figura 6.12 SUSAN t=25

Con el problema de detección de objetos redondos solucionado, se utilizó el algoritmo SUSAN como la base para el primer algoritmo de detección de un objeto en movimiento en una secuencia limitada de imágenes. El segundo algoritmo de detección de movimiento utiliza como base uno de los distintos algoritmos de extracción de bordes (Roberts, Sobel y Prewitt). Para este último algoritmo las pruebas determinaron que con el uso del algoritmo Roberts la detección era un poco más rápida que con las otras opciones, sin embargo, era más sensible a la presencia de ruido, por lo que convino utilizar el algoritmo de Sobel para este método. Las siguientes imágenes muestran los resultados

obtenidos por el algoritmo de movimiento 1 así como los resultados del algoritmo de movimiento 2 cuando un objeto se mueve libre en las escenas. Ambos algoritmos arrojaban el mismo resultado antes y después de las correcciones.

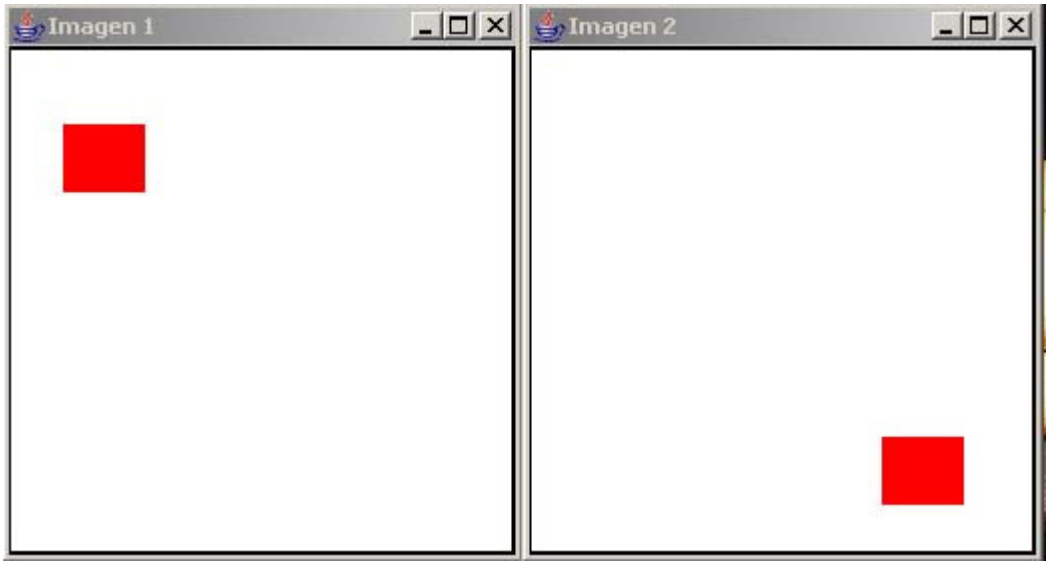

Figura 6.13 Imágenes originales del movimiento libre de un objeto

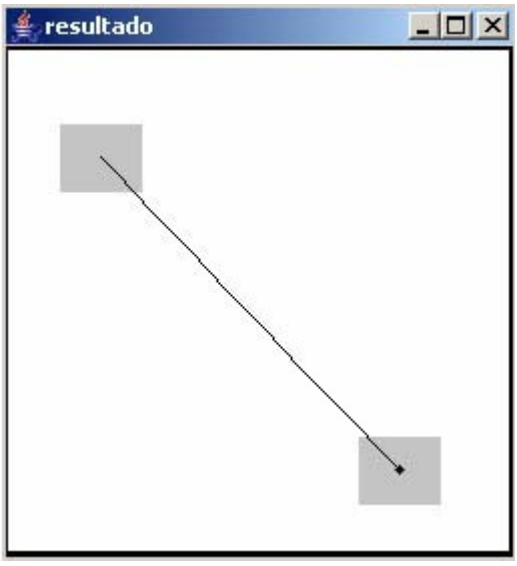

Figura 6.14 Resultado del algoritmo 1

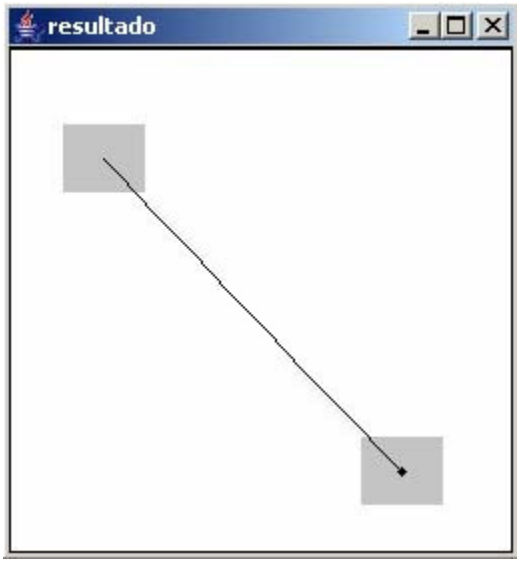

Figura 6.15 Resultado del algoritmo 2

Otra modificación que se hizo al sistema una vez terminado fue la manera de detectar al objeto en movimiento de ambos algoritmos, ya que cuando el objeto se movía libre en una escena, ambos algoritmos detectaban su desplazamiento con gran eficiencia, pero tenían problemas cuando se presentaban oclusiones, esto es, que el objeto en movimiento fuera obstruido por algún otro objeto estático en la escena. Esto se solucionó obteniendo todos los objetos estáticos de las imágenes a estudiar y de esta forma se pudo obtener solamente el objeto en movimiento y se pudo calcular su desplazamiento de forma más exacta. Las figuras 6.17 y 6.19 muestran los resultados obtenidos por los algoritmos antes de solucionar el problema de oclusión, mientras las figuras 6.18 y 6.20 muestran los resultados una vez resuelto el problema.

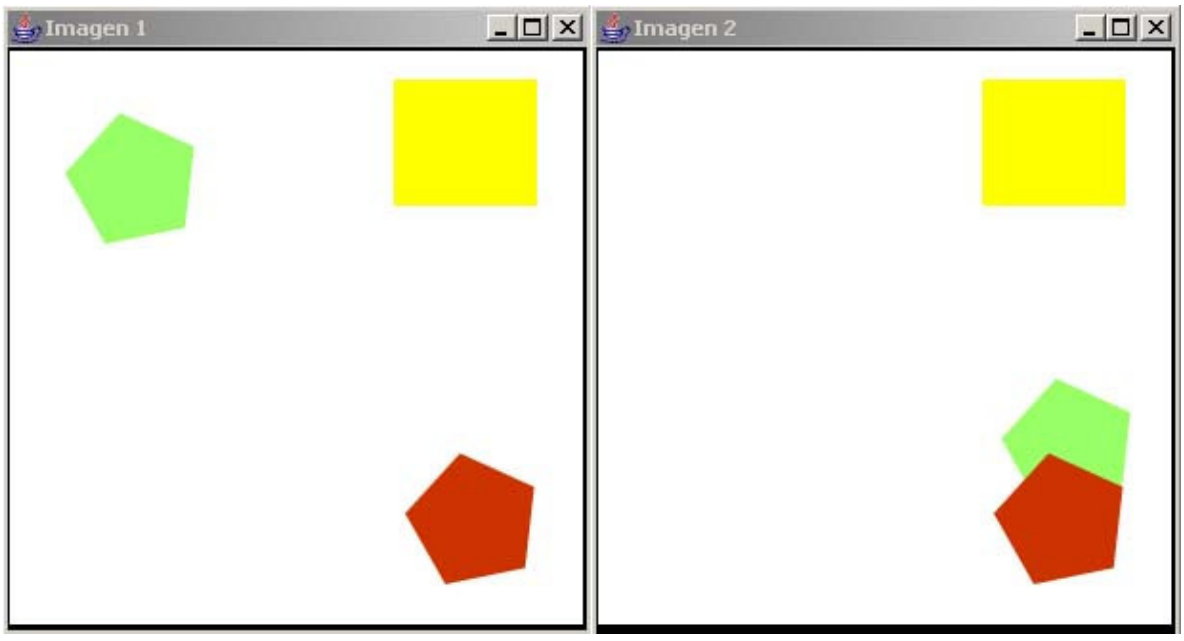

Figura 6.16 Imágenes originales con oclusión.

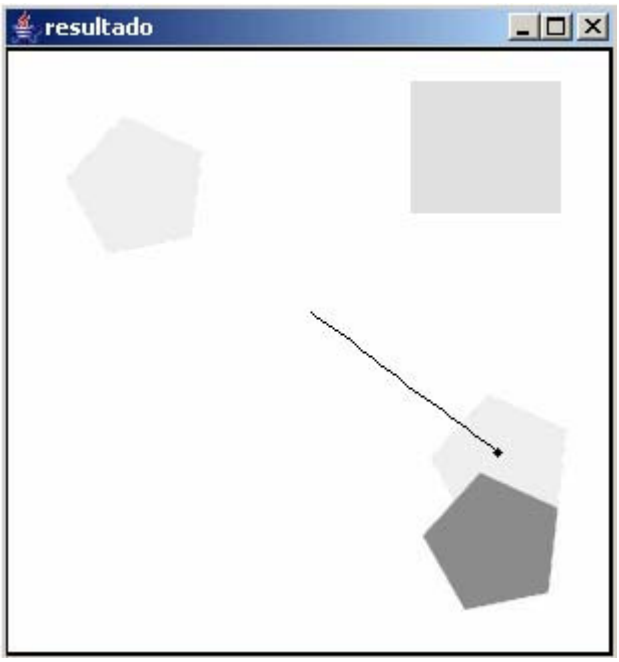

Figura 6.17 Resultado obtenido por algoritmo 1 antes de la corrección

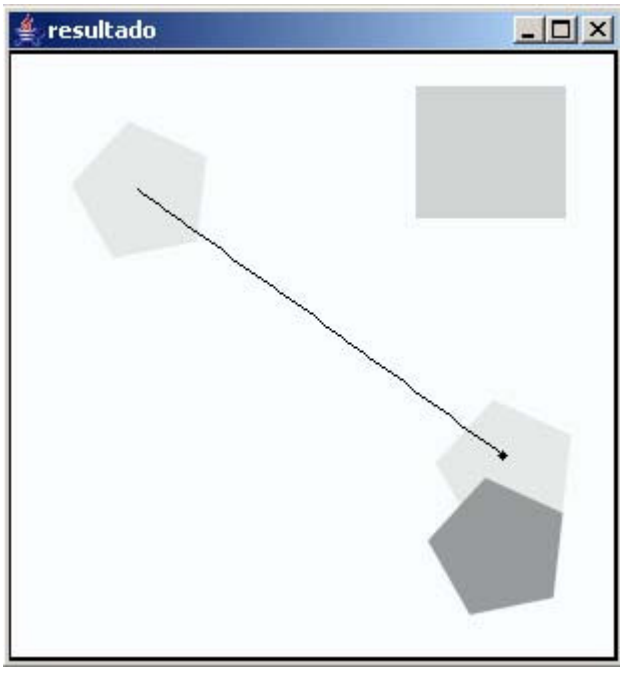

Figura 6.18 Resultado obtenido por algoritmo 1 después de la corrección

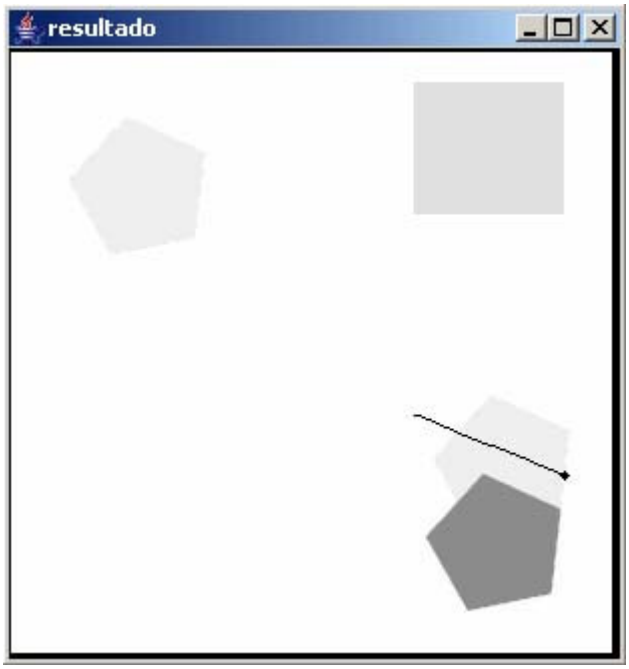

Figura 6.19 Resultado obtenido por algoritmo 2 antes de la corrección

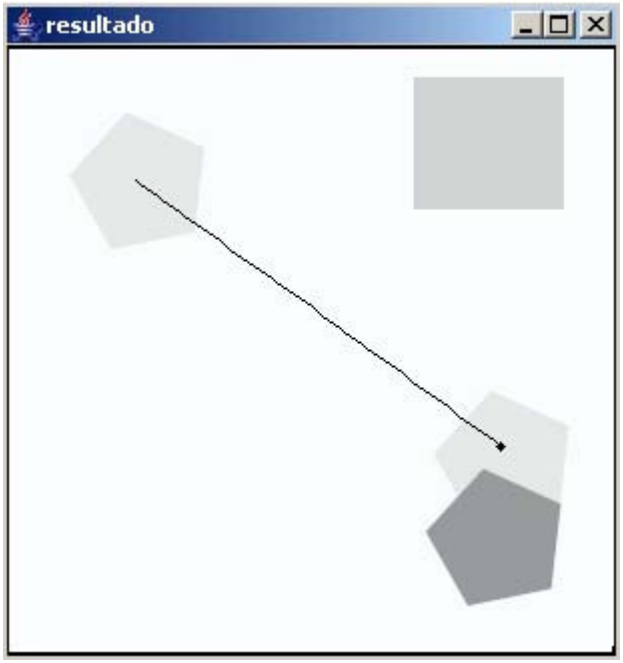

Figura 6.20 Resultado obtenido por algoritmo 2 después de la corrección

La exactitud de los métodos para detectar el campo de movimiento depende de la exactitud de las características detectadas del objeto en movimiento, de la cantidad del objeto que se pierde por una oclusión y de la forma de detectar los centros de los objetos. Esta última se puede realizar de dos maneras, la primera fue determinar los centros geométricos de los objetos por medio de las fórmulas siguientes:

$$Xc = \frac{\sum_{i}^{t} x_i}{t} \ , \ Yc = \frac{\sum_{i}^{t} y_i}{t}$$

donde t es el número de puntos detectados del objeto en movimiento, x_i corresponde a la coordenada x del punto i y y_i corresponde a la coordenada y del punto i. El resultado de esta forma de detectar el centro no fue exacta en la presencia de oclusiones, por lo que se generó la segunda forma de detectar centros geométricos, la cual construye un rectángulo con los puntos x mayor y menor y los puntos y menor y mayor, de tal manera que todo el objeto en movimiento queda dentro del cuadrado. La figura 6.21 muestra la generación del cuadro y la forma de determinar su centro.

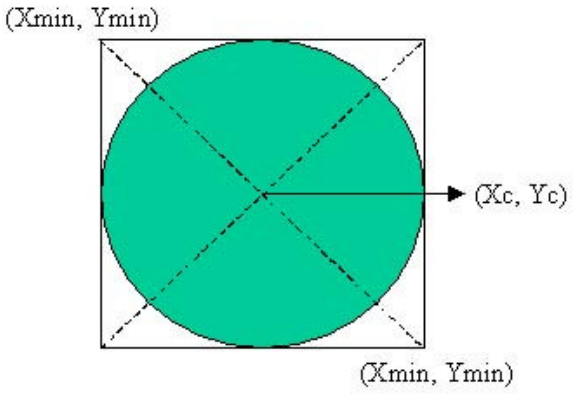

Figura 6.21 Obtención de centro geométrico.

$$Xc = X_{\min} + \frac{(X_{\max} - X_{\min})}{2},\ Yc = Y_{\min} + \frac{(Y_{\max} - Y_{\min})}{2}$$

Esta última forma de detectar centros dio mejores resultados que la primera en presencia de oclusiones, además de ser más rápida. Para ilustrar los resultados se presentan las siguientes figuras:

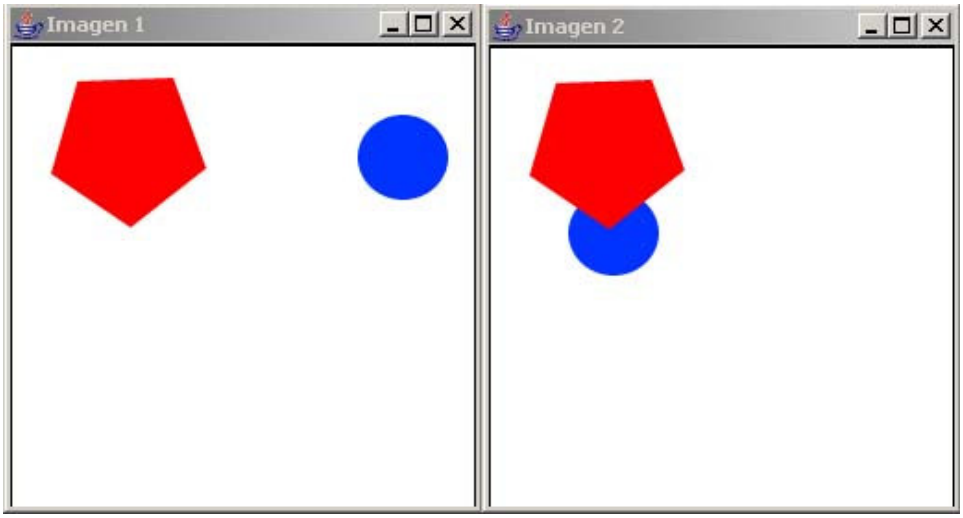

Figura 6.22 Imágenes originales para detección de centros geométricos.

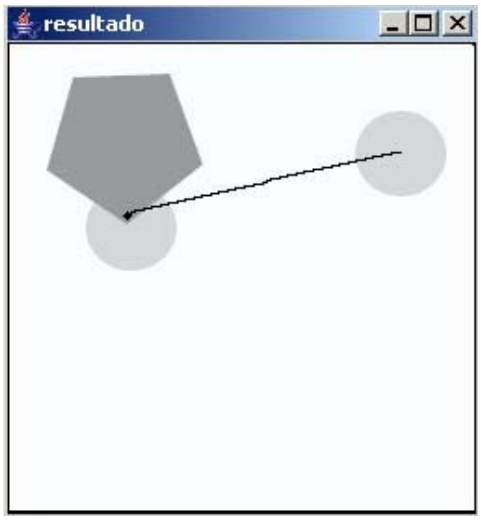

Figura 6.23 Resultado inicial con algoritmo 1

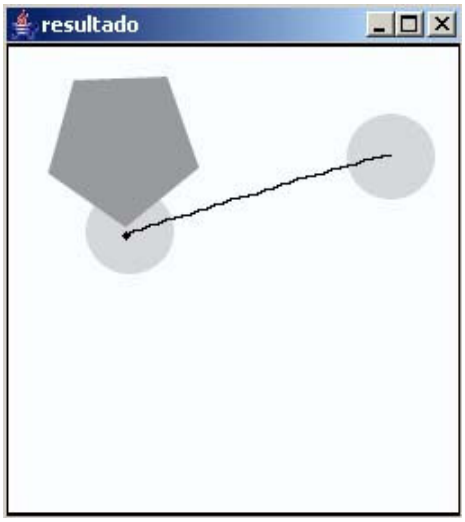

Figura 6.24 Resultado final con algoritmo 2.

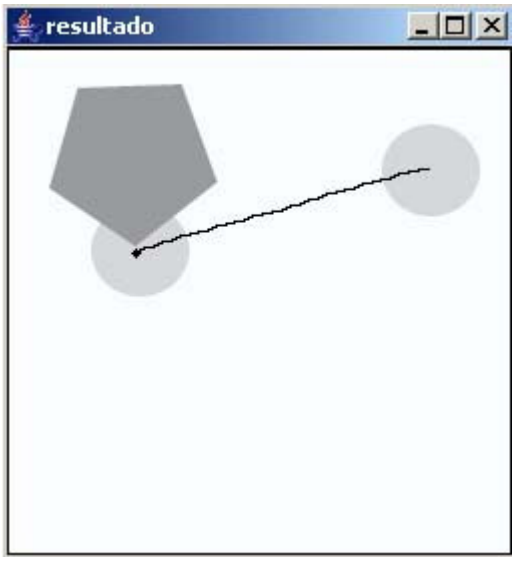

Figura 6.25 Resultado inicial con algoritmo 1.

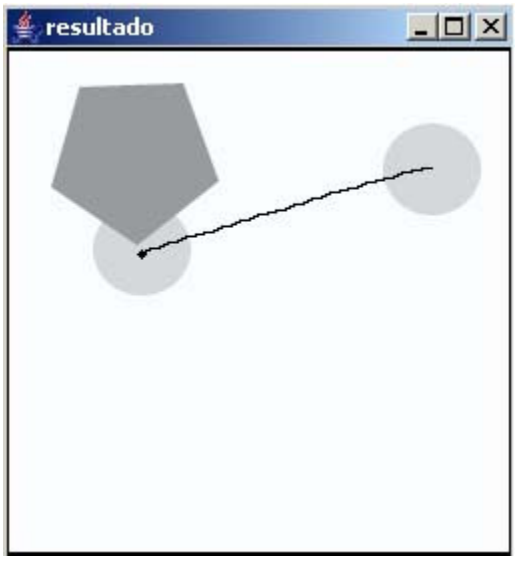

Figura 6.26 Resultado final con algoritmo 2.

Una vez corregidos ambos algoritmos de detección de movimiento se hicieron pruebas con ellos para detectar el campo de movimiento cuando el objeto esta rotando entre escenas como se muestra en la figura 6.27. Debido a que ambos algoritmos determinan el desplazamiento mediante la obtención de centros geométricos, la presencia de rotación no afecta el funcionamiento de los mismos. El resultado a la estimación en presencia de rotación se presenta en la figura 6.28.

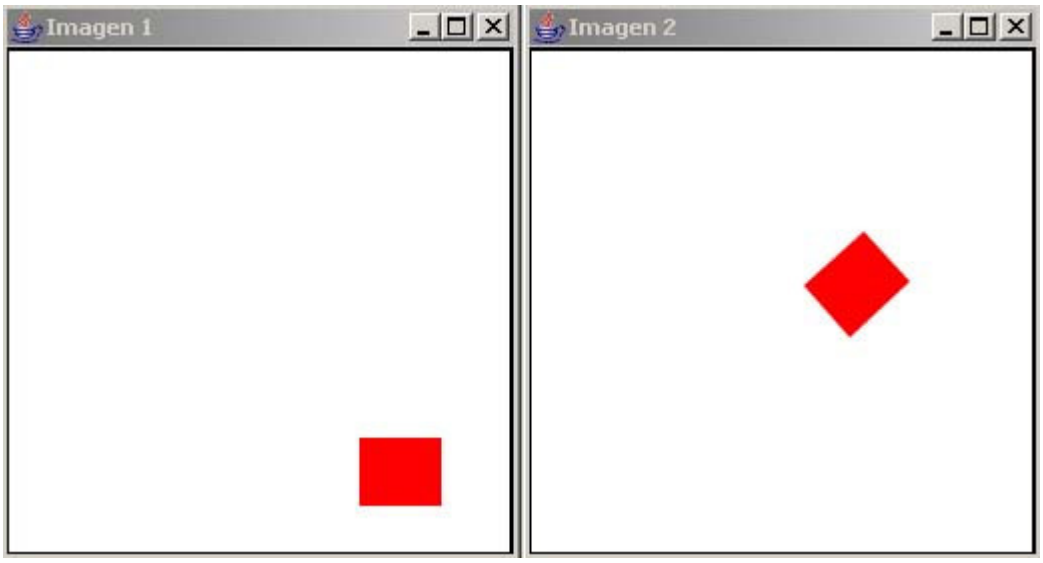

Figura 6.27 Imágenes originales para detección de movimiento invariante a rotación

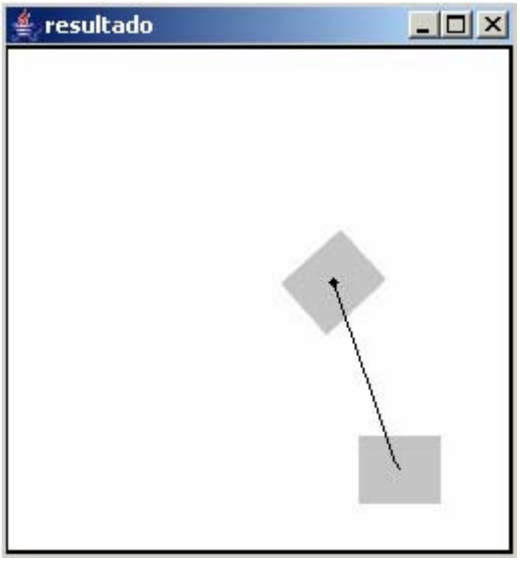

Figura 6.28 Resultado de algoritmos ante rotación.

Por último, la detección de la velocidad del objeto en movimiento se calcula mediante la fórmula $v = \frac{d}{t}$ donde d corresponde a la distancia en píxeles que se desplazó el centro del objeto y t corresponde al tiempo que transcurrió entre una escena y otra. Mediante este calculo se obtiene la velocidad en píxeles/segundo, en caso de querer

determinar la velocidad en metros/segundo, se necesitaría conocer la escala de la imagen. En la figura 6.27 se sabe que el tiempo que transcurrió entre la imagen 1 y la imagen 2 es de 0.020 segundos y el desplazamiento es de 63 píxeles, se obtiene la velocidad del desplazamiento la cual es de 3150 píxeles/segundo. El sistema despliega el valor de las velocidades en la interfaz de resultados debajo de la imagen obtenida por el procesamiento.

6.3 Conclusiones.

Este capítulo muestra los problemas con los que se tuvo que lidiar durante la implementación del sistema así como los errores que generaban los métodos una vez terminada la aplicación. Aclara la forma en que se solucionó cada dificultad y muestra gráficamente la diferencia entre los resultados generados por el sistema antes y después de las correcciones.

También por medio de este capítulo se explican las razones de la selección de cada método para detectar el movimiento de un objeto, los tiempos de ejecución y la exactitud de cada método en las diferentes circunstancias que se pueden presentar como es el caso de movimiento libre de un objeto o la presencia de oclusiones.

En este capítulo se muestran algunos resultados generados por el sistema, pero para revisar más imágenes procesadas en su interfaz final, véase Apéndice D correspondiente a pruebas adicionales.