Capítulo IV

Diseño e Implementación del Sistema

IV

4.1 Diseño del Sistema

La principal fuente de inspiración para el diseño preliminar del sistema es basado en la idea de cómo se cree que se realiza el proceso de visión estéreo en el ser humano. Básicamente para el proceso de visión se necesita la percepción del color, la forma y el movimiento; en este trabajo se intenta abordar el proceso de visión especialmente desde el punto de vista de la volumetría estereo, es decir, la forma tridimensional de los objetos de la escena que se analiza.

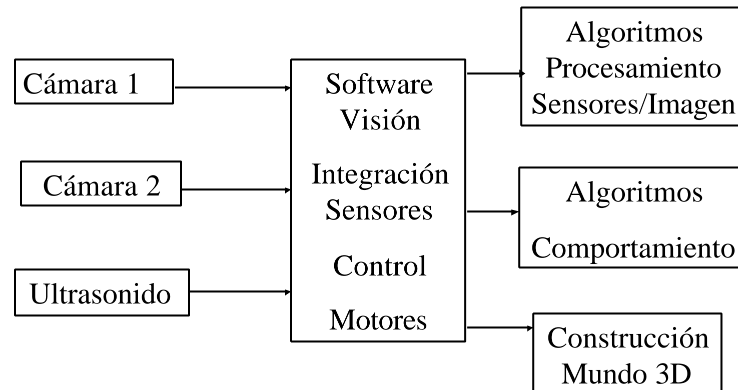

Fig. 4.1 Diagrama General

El siguiente es el diagrama a bloques de nuestro sistema donde se implementa la integración de sensores, el software de visión y los algoritmos de procesamiento, por último se da la construccion del mundo 3D.

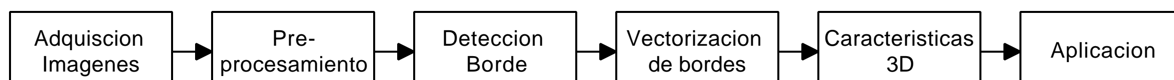

Figura 4.2 Diagrama a bloques general del sistema

Siguiendo los lineamientos de los diagramas UML, a coninuacion se muestran dichos diagramas.

TFormPrincipal
- Video : __published:TVideoCapture* //componente para captura de video
- Image1 : TImage* //canvas de imagen
- Image2 : TImage* //canvas de imagen
- Timer1 : TTimer* //canvas de imagen
- Timer2 : TTimer* //canvas de imagen
- Lienzo1 : TImage* //canvas de imagen
- Lienzo2 : TImage* //canvas de imagen
- Lienzo3 : TImage* //canvas de imagen
- Lienzo4 : TImage* //canvas de imagen
- MainMenu1 : TMainMenu* //barra de menu
- Archivo1 : TMenuItem* //menu de archivo
- Salir1 : TMenuItem* //menu de salida
- Configuracion1 : TMenuItem* //menu de configuracion
- Camara1 : TMenuItem* //menu para ajustes del dispositivo de entrada
- Threshold1 : TMenuItem* //menu de umbrales
- ToolBar1 : TToolBar* //barra de herramientas
- Status : TStatusBar* //barra de estado
- Accion1 : TMenuItem* //menu de accionamiento del sistema
- Detener : TMenuItem* //menu para detener el sistema
- Calibracion1 : TMenuItem* //menu de calibracion
- Errors : TMemo* //indica las caracteristicas encontradas, los pares y coordenadas 3D
- FiltradoPasabajas1 : TMenuItem* //menu activacion filtrado suavizante
- PintaBorde1 : TMenuItem* //menu para seleccionar graficacion de vectores
- BordeCanny1 : TMenuItem* //menu para seleccionar el detector de borde canny
- Panel1 : TPanel* //panel general de la aplicacion
- Caracterist : TMenuItem* //menu para el desplegado de puntos analizados y encontrados
- Extra3D : TMenuItem* //menu de desplegado 3D
- SpeedButton1 : TSpeedButton* //inicio rapido de la adquisicion
- SpeedButton2 : TSpeedButton* //paro rapido de la aplicacion
- Sigmoidal : TMenuItem* //menu de contraste sigmoidal
- config : TGraphConfig* //configuracion graficacion
- arreglo : int[361][601] //lleva cuenta de caracteristicas vectorizadas en transformada hough
- s_arr : bool[2][360][290][40] //contiene informacion de la coordenada y angulo de la caracteristica
- max_img_x : int //largo imagen procesada
- max_img_y : int //ancho imagenes procesadas
- R : float //componente de color R de la imagen
- G : float //componente de color G de la imagen
- B : float //componente de color B de la imagen
- rad : int //valor del radio en coord. polar para el vector de la transformada Hough
- tet : int //valor del angulo en coord. polar para el vector de la transformada Hough
- nombre_foto : AnsiString //nombre de imagen a guardar del dispositivo de entrada
- a : AnsiString //nombre de dispositivo entrada
+ ptr : Byte* //puntero para imagenes
+ ptr0 : Byte* //puntero para imagenes
+ ptr1 : Byte* //puntero para imagenes
+ ptr2 : Byte* //puntero para imagenes
+ ptr3 : Byte* //puntero para imagenes
+ sx : float[8][8] //mascara x
+ sy : float[8][8] //mascara y
+ thres : int //umbral binarizacion
+ thres_h : int //umbral transformada hough
+ thres_z : int //umbral numero zonas hough
+ thres_c : int //umbral color
+ thres_s : int //umbral ajuste contraste
+ ptr_pares : int //variable que guarda el numero de caracteristicas emparejadas
+ Coord3D : int[512][3] //arreglo coordenadas tridimensionales
+ pares : int[512][4] //arreglo de caracteristicas pares
- FormCreate(Sender : TObject*) : void __fastcall //inicializacion
- FormDestroy(Sender : TObject*) : void __fastcall //destruccion y liberacion de memoria
- FormClose(Sender : TObject*, Action : TCloseAction&) : void __fastcall //cierra pantallas
- VideoBitmapGrabbed(CapturedImage : TCapturedBitmap*) : void __fastcall //captura imagen desde disposisitvo activo
- Timer1Timer(Sender : TObject*) : void __fastcall //hace llamado a funciones proc. dig. de imagenes para imagen1
- Timer2Timer(Sender : TObject*) : void __fastcall //hace llamado a funciones proc. dig. de imagenes para imagen2
- Threshold1Click(Sender : TObject*) : void __fastcall //llama a rutinas de ajuste de umbrales
- Salir1Click(Sender : TObject*) : void __fastcall //detiene rutinas y terrmaina programa
- DetenerClick(Sender : TObject*) : void __fastcall //detiene rutinas de captura y procesado
- Accion1Click(Sender : TObject*) : void __fastcall //inicia toma de secuencias de video
- Calibracion1Click(Sender : TObject*) : void __fastcall //calibracion camara
- PintaBorde1Click(Sender : TObject*) : void __fastcall //activa/desactiva graficacion vectores
- FiltradoPasabajas1Click(Sender : TObject*) : void __fastcall //aplica filtrado suavizante
- BordeCanny1Click(Sender : TObject*) : void __fastcall //cambia entre borde canny/sobel
- CaracteristClick(Sender : TObject*) : void __fastcall //activa/desactiva cuadro con coordenadas encontradas
- Extra3DClick(Sender : TObject*) : void __fastcall //lamada a visualizacion en 3D
- SigmoidalClick(Sender : TObject*) : void __fastcall //accion para activar/desactivar contraste
- accion1() : void //inicializa dispositivo 1 y captura
- accion2() : void //inicializa dispositivo 2 y captura
- iniimg(pBitmap : TPicture*, pBitmap2 : TPicture*, numero : int) : void //inicialia y carga imagenes de entrada
- iniimg2(pBitmap : TPicture*, pBitmap2 : TPicture*) : void //inicializa y carga imagenes de bordes
- DrawLinea(x : int, y : int, x1 : int, y1 : int, b : float, g : float, r : float, pBitmap : TPicture*, canv : int) : void //dibuja lineas dadas 2 coordenadas
- polar(r : int, t : int, pBitmap : TPicture*, img : int) : void //grafica vector polar
- get_pol(x : int, y : int, angulo : float, pBitmap : TPicture*) : void //obtiene coordenada polar y guarda en arreglo
- get_rt(x : int, y : int, angulo : float, pBitmap : TPicture*) : void //obtiene r y t dado un punto
- diferencia(x0i : int, x1i : int, x2i : int, x0d : int, x1d : int, x2d : int, xi : int, xd : int, y : int, menor : int) : int //obtiene la diferencia entre 2 caracteristicas dadas
- dosDtotresD(pBitmap : TPicture*) : void //conversion coordenadas 2d a 3D

```
- iniimg(pBitmap : TPicture*, pBitmap2 : TPicture*, numero : int) : void   //inicialia y carga imagenes de entrada
- iniimg2(pBitmap : TPicture*, pBitmap2 : TPicture*) : void   //inicializa y carga imagenes de bordes
- DrawLinea(x : int, y : int, x1 : int, y1 : int, b : float, g : float, r : float, pBitmap : TPicture*, canv : int) : void   //dibuja lineas dadas 2 coordenadas
- polar(r : int, t : int, pBitmap : TPicture*, img : int) : void   //grafica vector polar
- get_pol(x : int, y : int, angulo : float, pBitmap : TPicture*) : void   //obtiene coordenada polar y guarda en arreglo
- get_rt(x : int, y : int, angulo : float, pBitmap : TPicture*) : void   //obtiene r y t dado un punto
- diferencia(x0i : int, x1i : int, x2i : int, x0d : int, x1d : int, x2d : int, xi : int, xd : int, y : int, menor : int) : int   //obtiene la diferencia entre 2 caracteristicas dadas
- dosDtotresD(pBitmap : TPicture*) : void   //conversion coordenadas 2d a 3D
- color(img : int) : void   //procedimiento color
- low_pass(img : int) : void   //filtrado suavizante
- exponential_adj(img : int) : void   //realiza ajuste exponencial-sigmoidal
- gray_borde_bin(img : int) : void   //realiza conversion a escala de gris, deteccion de bordes y binarizaciones
- Hough(img : int) : void   //vectorizacion de bordes
- Matching() : void   //realiza el emparejamiento de las caracteristicas
-------------------------------------------------------------------------------
Realizar operaciones de Procesamiento digital de Imagenes
Realizar Filtros, Realces, Deteccion de Contornos, Vectorizacion, y Matching
Pintar algunas operaciones del procesamiento en pantalla
```

Figura 4.3

El diseño consta de 4 módulos, estos son: un programa principal encargado de llevar a cabo las tareas de control de la entrada de las secuencias de video y procesamiento digital de dichas secuencias de entrada. El diagrama UML anterior nos muestra la sección del programa principal (ver fig. 4.3). Una segunda sección consta de el desplegado y manipulación de las coordenadas de salida 3D y el diagrama UML es mostrado a continuación (ver fig. 4.4).

```
TFormTresD
-------------------------------------------------------------------------------
- OpenGLPanel1 : __published:TOpenGLPanel*   //canvas para visualizacion OpenGL
- TrackBar1 : TTrackBar*   //barra para cambiar angulo de rotacion X
- TrackBar2 : TTrackBar*   //barra para cambiar angulo de rotacion Y
- Button1 : TButton*   //boton de activacion salvar escena
- Button2 : TButton*   //boton activacion impresion escena
- TrackBar3 : TTrackBar*   //barra para cambiar angulo de rotacion Z
- TrackBar4 : TTrackBar*   //barra para cambiar distancia camara
- luz : TCheckBox*   //opcion activacion de la luz en la escena
- smooth : TCheckBox*   //opcion activacion suavizamiento de objetos
- fill : TCheckBox*   //opcion activacion llenado de objetos
- segunda : TCheckBox*   //opcion activacion vista 2D
- SaveDialog1 : TSaveDialog*   //dialogo para salvar imagen
- PrintDialog1 : TPrintDialog*   //dialogo impresion imagen
+ RotX : float   //angulo rotacion eje X
+ RotY : float   //angulo rotacion eje Y
+ RotZ : float   //angulo rotacion eje Z
+ cam : float   //distancia localizacion camara
+ no_coord : int   //contador numero de coordenadas 3D
-------------------------------------------------------------------------------
- OpenGLPanel1Init(Sender : TObject*) : void __fastcall   //inicializacion canvas open gl
- OpenGLPanel1Resize(Sender : TObject*) : void __fastcall   //Reacomodo pantalla en cambio de tamaño
- OpenGLPanel1Paint(Sender : TObject*) : void __fastcall   //Pintado canvas open GL
- Button1Click(Sender : TObject*) : void __fastcall   //accion para salvar a imagen la escena
- Button2Click(Sender : TObject*) : void __fastcall   //accion para impresion de escena
- TrackBar1Change(Sender : TObject*) : void __fastcall   //cambio orientacion X
- TrackBar2Change(Sender : TObject*) : void __fastcall   //cambio en orientacion Y
- TrackBar3Change(Sender : TObject*) : void __fastcall   //cambio en orientacion Z
- TrackBar4Change(Sender : TObject*) : void __fastcall   //cambio en perspectiva de la camara
- luzClick(Sender : TObject*) : void __fastcall   //activa/desactiva luz
- fillClick(Sender : TObject*) : void __fastcall   //accion para activar/desactivar el relleno de los objetos
- smoothClick(Sender : TObject*) : void __fastcall   //accion para activar/desactivar suavizameinto objetos
- segundaClick(Sender : TObject*) : void __fastcall   //accion para camiar a vista 2D
+ TFormTresD(Owner : TComponent*) : __fastcall
-------------------------------------------------------------------------------
Realizar operaciones de Construccion 3D
Representacion 3D en open GL
```

Figura 4.4 Diagrama UML desplegado

Por último se tiene la sección de ajuste y calibración donde se realizan los cambios en los parámetros, así como ajustes y configuraciones. El diagrama UML de la implantación de dicha sección es el siguiente (ver fig. 5.5).

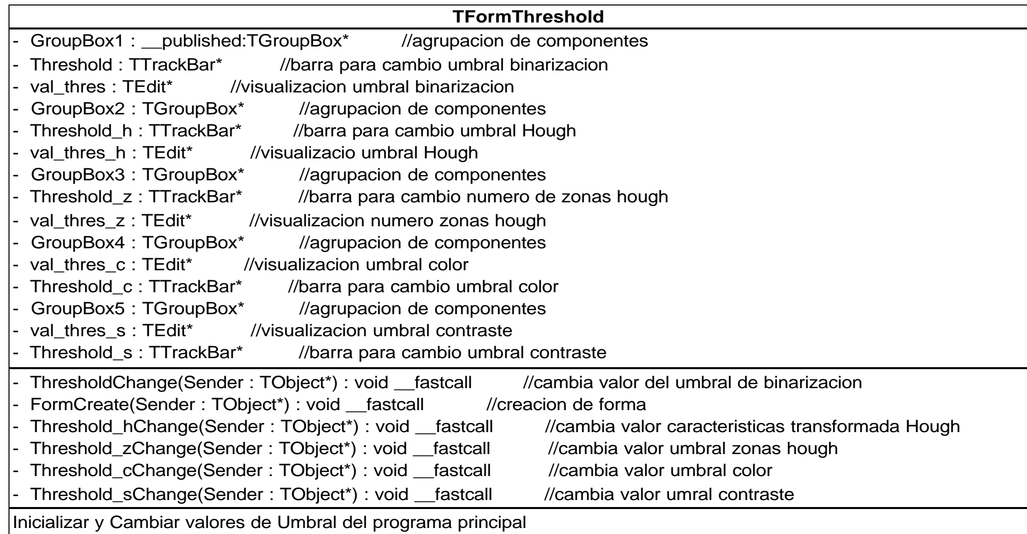

Figura 4.5 Diagrama UML configuración

Las reglas básicas que han sido tomadas para el diseño de el sistema son las siguientes: se prohíbe diferencia en profundidad en regiones constantes; en una línea que se analiza, un objeto que es parcialmente visible a una cámara y que también lo es a otra cámara, su distancia puede ser determinada mediante triangulación; el objetivo primordial consiste en detectar las discontinuidades de profundidad mas que computar un mapa exacto de profundidad.

4.2 Hardware/Software a utilizar

Para el desarrollo de este proyecto se utiliza C++ Builder Versión 5.0 de la Compañía Borland.

Como hardware, se usa una computadora PC con un procesador marca AMD modelo Athlon trabajando a una frecuencia de 1.33 Ghz. (1.5 Ghz+), con 256Mb de memoria RAM y sistema operativo Windows XP.

La entrada del sistema es un par de cámaras digitales marca Alaska, con un sensor de 100,000 pixeles CMOS y una resolución máxima de 352x288 pixeles, con una profundidad de color de 24 bits RGB, con un rango de refresco de hasta 30 fps., la interfaz con la computadora es una conexión tipo USB, cuentan con lentes de 10 cm. al infinito, lo que resulta óptimo para los propósitos de esta aplicación; la

apertura de la lente es de 40?. Y como dispositivo auxiliar se utiliza un sistema ultrasónico de detección de obstáculos a 40 Khz, con un alcance de 3 metros y una precisión de 2 centímetros, con una interfaz serial con la computadora para medir la distancia entre las cámaras y el objeto que se analiza.

4.3 Implantación

Para lograr la implantación del sistema que construya un ambiente virtual tridimensional de modelos geométricos básicos de un ambiente que se explora, se lleva a cabo mediante la implementación de los bloques mostrados en la figura 4.6.

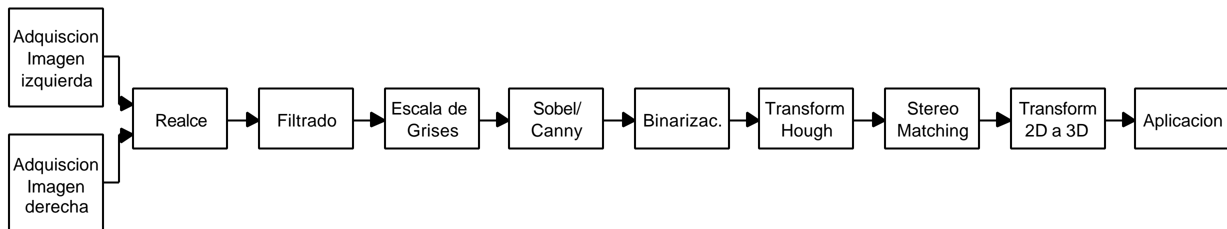

Fig. 4.6 Diagrama a bloques del sistema

La explicación de cada uno de los bloques y su forma de implantación se encuentra a continuación.

4.3.1 Entrada del Sistema

Como sistema de adquisición de imágenes se utiliza un par de cámaras digitales de video, con las características mencionadas anteriormente. Hay que tener en cuenta que para que el sistema tenga un mejor desempeño, las cámaras deben de ser iguales, es decir: de la misma marca y modelo; aun con esta restricción se tienen errores en el muestreo debido a la diferencia en los sensores.

Las cámaras obtienen una imagen cada 1250ms, alternando entre la imagen izquierda y la derecha; la razón de esto es que que el ancho del bus USB no soporta tener 2 dispositivos con una tasa de transferencia de este tipo al mismo tiempo, por lo que se optó por encender una sola cámara a la vez, apagarla y en el siguiente instante encender la segunda cámara; de esta forma se toman las secuencias alternativamente. Acceder a estas capacidades anteriormente no era posible en un ambiente Windows, si se quería tener acceso a dos cámaras iguales desde una plataforma PC era necesario trabajar dentro de un ambiente Linux por ejemplo y una alternativa a esto era el uso de tarjetas digitalizadoras de video que necesitaban una secuencia de video de tipo NTSC; sin embargo, al incorporar las capacidades DirectShow a partir de la versión 8.0 de DirectX, es posible acceder a dos cámaras por ejemplo del tipo

webcam, el único problema encontrado fue el que las cámaras al tener el mismo controlador son llamadas con el mismo nombre por lo que el sistema no puede diferenciar entre una u otra, por lo que se procedió a editar el controlador para que pudieran ser reconocidas individualmente, identificándose con los nombres ICM532A y ICM532B respectivamente. Siendo que originalmente las dos cámaras eran reconocidas como ICM532A e ICM532A..

Estas cámaras dan como salida imágenes de tipo BMP con una resolución de 352 x 288 pixeles y con una profundidad de color de 24 bits; las imágenes producidas por las cámaras son la entrada a nuestro sistema.

Las cámaras se montaron a una distancia de 7 centímetros entre sí, y es con esta medida con la que se realizan la mayoría de los experimentos, aunque cabe aclarar que también se realizan experimentos y mediciones localizando las cámaras a una distancia de 10 y 14 cm. Se analiza de igual manera el uso de un dispositivo ultrasonico para medir la distancia real y asi poder hacer comparaciones con la profundidad obenida mediante la triangulación de dos imágenes; los resultados dados por la incorporación de este sistema se explican en el capítulo 5.

4.3.2 Procesamiento

Una vez que se tienen las imágenes a color, se realiza el procesamiento digital de la imagen. Al usar dos cámaras es necesaria la calibración, debido a que las diferencias en la posición de la imagen causada por pequeños desajustes en la localización de las cámaras o también por alinealidad de los sensores, da lugar a errores, por lo que el primer paso a realizar es la calibración de las cámaras.

El preprocesamiento aplicado a las imágenes consiste en aplicar un contraste de tipo sigmoide a las imágenes, es decir; se realiza una operación de contraste que hace que los colores claros se vuelvan más claros y los obscuros más obscuros y con ésto lograr más fácilmente la detección de bordes y de color. La ecuación del contraste usado se muestra en la ecuación 4.1.

$$pixel_out \ ? \ \frac{1}{1 \ ? \ e^{?x}}$$

Donde x corresponde al color del pixel y puede ser afectado por un parámetro del programa modificable

Ec. 4.1 Ecuación del filtro Sigmoide usado

Después del contraste se puede realizar un filtrado de tipo pasabajas o Gaussiano; la mascara del filtro pasabajas usado se muestra en la figura 4.7; los parámetros del contraste y la aplicación o no tanto del contraste como del filtrado son parámetros que pueden ser modificados mediante controles en el programa.

1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

Fig. 4.7 Máscara pasabajas usada

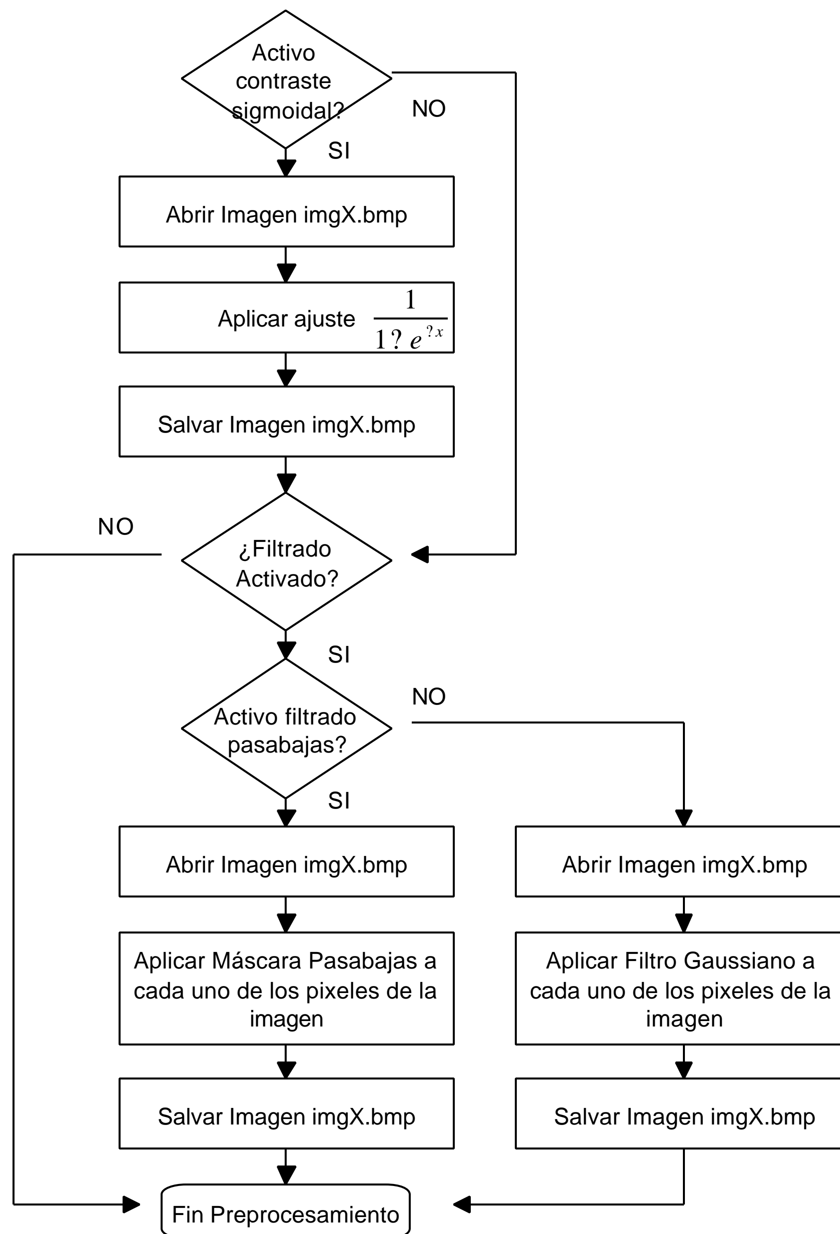

Fig. 4.8 Diagrama de flujo Preprocesamiento

El diagrama de flujo del bloque del preprocesamiento, es mostrado en la figura 4.8. donde X puede tomar valores de 1 o 2 para indicar si se procesa en ese momento la imagen izquierda o derecha (un 1 indica que se maneja la imagen proveniente de la cámara izquierda y 2 para indicar que se maneja la derecha).

En el capitulo 5 se dan los resultados de la modificación de los parámetros disponibles del preprocesamiento.
Una vez realizado el preprocesamiento de las imágenes, se realiza la detección de contornos; esto se hace en tres pasos: el primero es la conversión de las imágenes a escala de grises, esto se hace con dos propósitos; primero para utilizar una imagen en escala de grises en algún algoritmo de detección de borde, (aunque también se analiza el uso de la imagen a colores para este propósito) y segundo, la imagen en gris es usada para el emparejamiento estéreo; es decir: para dar el nivel de compaginación entre las características de las 2 imágenes.
La ecuación para dar el nivel de gris correspondiente es la siguiente:

$$f=0.299*R+0.587*G+0.114*B$$

Ecuación 4.2 Conversión nivel de gris

La ecuación 4.2 para la conversión de color a un nivel de gris, es usada debido a que el ojo humano tiene diferente sensibilidad para percibir los distintos colores como intensidad en la iluminación, por lo tanto lo anterior transforma el color al nivel de intensidad equivalente para el ojo humano.

Dos son los algoritmos empleados para la detección de bordes y el análisis de la aplicación de estos algoritmos es explicado en el siguiente capítulo. Estos algoritmos que se emplearon son el Algoritmo Sobel y el Canny, estos han sido explicados en capítulos anteriores.

Si es seleccionado el detector de bordes Sobel se aplica una máscara y una vez realizado el algoritmo de detección de bordes se aplica una binarización con un threshold determinado de manera experimental, pero que puede ser ajustado mediante software.
Si el detector de bordes es el Canny lo primero que se realiza es la aplicación de una mascara canny mostrada en la figura 4.9; con esto se obtienen valores proporcionales a la magnitud de la gradiente, con

estos valores se obtiene la dirección de la gradiente y es aplicado el adelgazamiento de contorno o supresión de los no máximos.

$$P[i,j] = \begin{array}{|c|c|} \hline -1 & 1 \\ \hline -1 & 1 \\ \hline \end{array} \qquad Q[i,j] = \begin{array}{|c|c|} \hline 1 & 1 \\ \hline -1 & -1 \\ \hline \end{array}$$

Fig. 4.9 Máscara Usada en el detector de bordes Canny

Para el adelgazamiento de contorno se hace como a continuación se indica: primero se obtiene el ángulo del borde, basándose en esto se compara cada pixel de borde con los adyacentes perpendiculares a la línea de borde y se selecciona únicamente el pixel cuyo valor sea mayor que los otros dos, es decir, el pixel cuyo valor es pico correspondiendo éste al punto preciso de borde, por lo que son suprimidos los valores que no son máximos; lo anterior conduce a tener bordes de un pixel de ancho. En nuestro programa al mismo tiempo que se adelgaza el contorno se aplica la binarización por umbral, ahorrando con esto tiempo de ejecución.

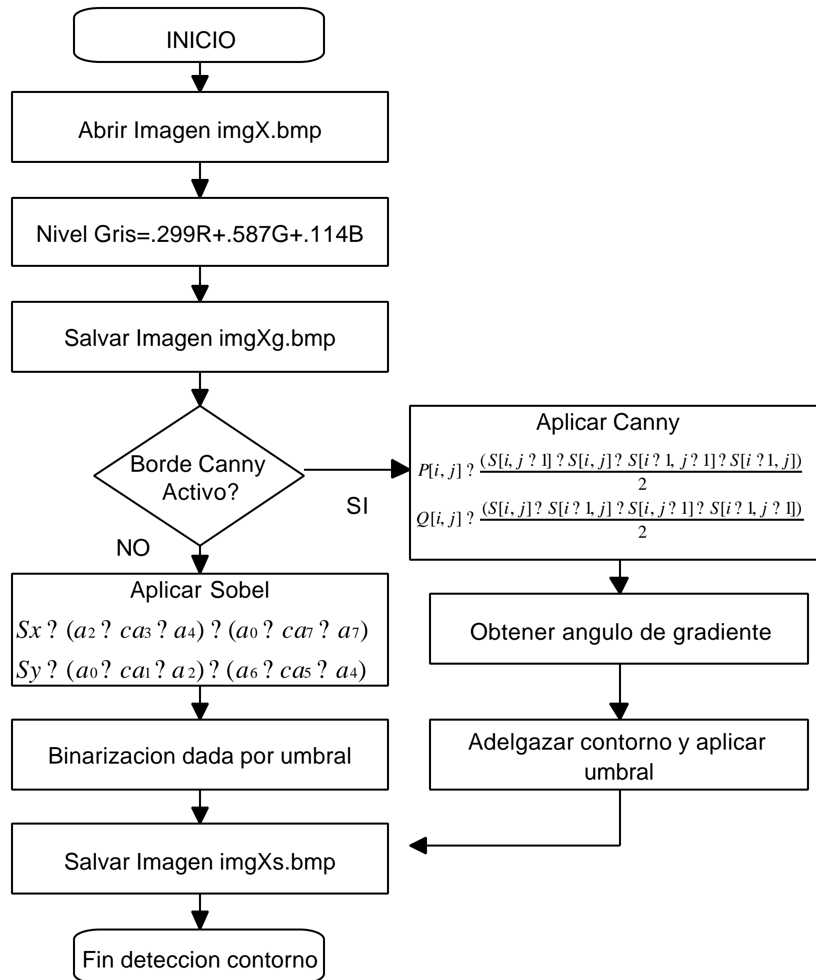

Fig. 4.10 Diagrama de Flujo Detección de Borde

El algoritmo Canny, sin embargo; fue implantado utilizando un solo nivel de umbral y como trabajo a futuro se encuentra el implementar el doble threshold y su encadenación entre las dos imágenes; los resultados obtenidos con este tipo de detector de borde son muy buenos y son expuestos en el capitulo 5. El diagrama de flujo de la sección correspondiente a la detección de borde es mostrado en la Fig. 4.10.

```
for (int x = 1; x < (Bitmap->Bitmap->Width); x++)
   {
     a1=gradiente x de la imagen
     b1=gradiente y de la imagen
    obtiene magnitud de la gradiente
     f=atan2(b1,a1);
     //sectoriza
```

```
    if((f<=M_PI)&(f>7*M_PI/8))|((f<=M_PI/8)&(f>-M_PI/8))|((f<=-7*M_PI/8)&(f>M_PI))sector=0;
    if((f<=7*M_PI/8)&&(f>5*M_PI/8))||((f<=-1*M_PI/8)&&(f>-3*M_PI/8))sector=3;
    if((f<=5*M_PI/8)&&(f>3*M_PI/8))||((f<=-3*M_PI/8)&&(f>-5*M_PI/8))sector=2;
    if((f<=2*M_PI/8)&&(f>M_PI/8))||((f<=-5*M_PI/8)&&(f>-7*M_PI/8)) sector=1;
    x=x+2;
   }//fin x
  for (int y = 1; y < Bitmap->Bitmap->Height-1; y++)
  {
   for (int x = 1; x < (Bitmap->Bitmap->Width); x++)
   {
    if(sector==0)
    {
      if((pixel[x][y]>pixel[x-1][y])&&(pixel[x][y]>pixel[x+1][y])&&(pixel[x][y]>thresh))
      pixel_salida[x][y]=blanco
    }
    if(sector==1)
    {
     if((pixel[x][y]>pixel[x-1][y+1])&&(pixel[x][y]>pixel[x+1][y-1])&&(pixel[x][y]>thresh))
     pixel_salida[x][y]=blanco
    }
    if(sector==2)
    {
      if(((pixel[x][y]>pixel[x][y-1])&&(ptr1[x]>pixel[x][y+1])&&(pixel[x][y]=thresh))
      pixel_salida[x][y]=blanco
    }
    if(sector==3)
    {
     if((pixel[x][y]>pixel[x-1][y-1])&&(ptr1[x]>pixel[x+1][y+1])&&( pixel[x][y]>thresh))
     pixel_salida[x][y]=blanco
    }
   }
```

Como puede apreciarse en la sección de código anterior, para obtener los bordes de un objeto por medio del algoritmo de Canny, primero se obtiene la detección de contornos por medio de una máscara que obtiene los bordes en la dirección *x(a1)* y en la dirección *y(b1)*; teniendo estas componentes, es posible obtener la magnitud y la dirección, esto es lo siguiente que se realiza, una vez obtenida la dirección del borde se determina a que sector corresponde (ver cap. 3), una vez obtenido el sector es posible el adelgazamiento del contorno, para hacerlo se comparan los pixeles vecinos que se encuentran sobre la línea de contorno de acuerdo al sector correspondiente y si el pixel central es mayor que sus vecinos y mayor a cierto umbral el pixel es reconocido como borde; ejemplo: si el sector asociado es el 0 se compara el pixel central contra el izquierdo y el derecho, si el sector asociado es el 1, se compara el pixel central contra el superior derecho y el inferior izquierdo y así sucesivamente.

Una vez extraídos los bordes de la imagen, se aplica la transformada Hough para determinar la dirección de los bordes más representativos, es decir, los bordes son vectorizados.

En realidad no se aplicó el algoritmo Hough original sino una modificación de éste; la modificación fue hecha por dos motivos; primero, debido a que la transformada hough es un algoritmo registrado, segundo porque la modificación realizada fue para que el tiempo de corrida fuera más rápido, ya que el tiempo de corrida de nuestros algoritmos resulta un factor crucial si se intenta trabajar en modelos en tiempo real; esta modificación consistió básicamente en que en lugar de hacer un barrido de todos los ángulos en los puntos con característica, únicamente se obtienen los ángulos comprendidos entre 0 y 180 grados, ahorrando con esto la mitad del tiempo de corrida en este algoritmo. Además de lo anterior, no se hace un barrido en cada grado entre 0 y 180 grados, sino que se puede realizar un barrido amplio cada 90? y hasta un barrido mínimo de 4.5 grados, al no realizar un cálculo de grado en grado se pierde precisión; obviamente el hacer un barrido con ángulos pequeños tiene como consecuencia un pago en el tiempo de corrida.

Otra modificación realizada al algoritmo es la siguiente: debido al ruido y a objetos pequeños, el algoritmo Hough los puede tomar como vectores representativo de objetos mayores que se desean estudiar, por lo que existe un parámetro configurable en el programa que nos ayuda a eliminar este tipo de pequeños vectores y de esta forma únicamente tomar en cuenta los vectores de características más representativas.

El diagrama de flujo de la aplicación del algoritmo de la transformada Hough modificada es mostrado a continuación (ver fig. 4.11).

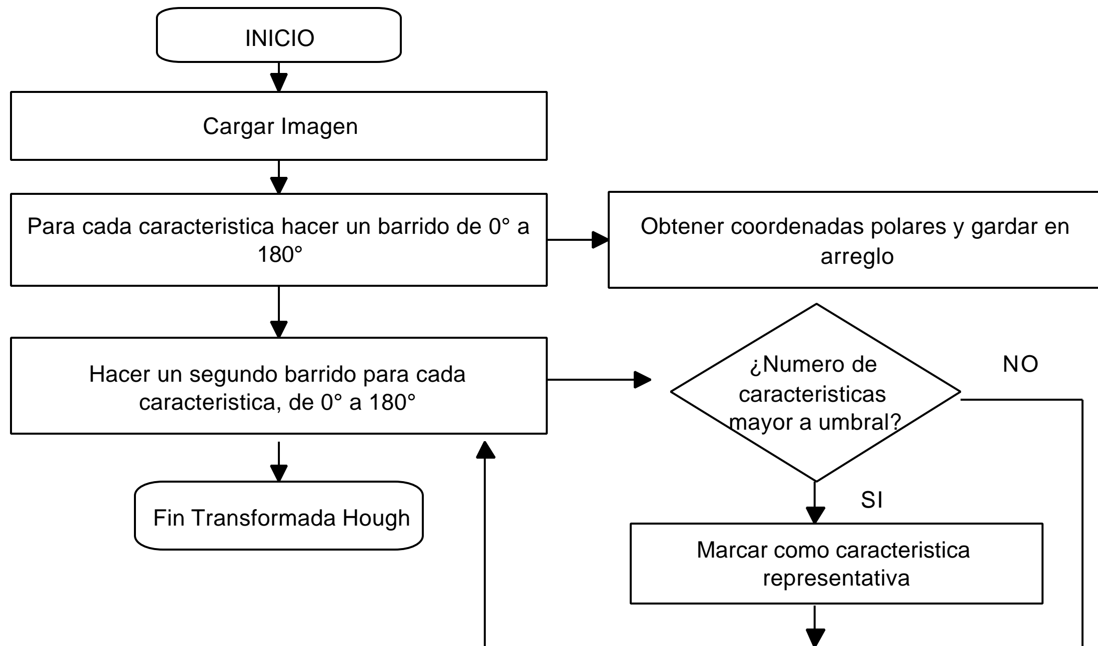

Fig. 4.11 Diagrama de Flujo Transformada Hough

A continuación se muestra la sección de código donde es implementado el algoritmo Hough; en primer lugar se hace un barrido para toda la imagen y en el momento de encontrarse un pixel blanco, es decir un pixel de borde, se procede a realizar la vectorizacion de ese punto en todas las direcciones que se han establecido; la coordenada polar obtenida del punto resultante de realizar el barrido de la característica en varios ángulos es obtenida mediante la funcion *get_pol*;

```
for (int j = 0; j < Bitmap->Bitmap->Height-1; j++)
 {
   x=-cx;
   y=cy-j;
   for (int i = 0; i < ((Bitmap->Bitmap->Width); i++)
   {
     if(pixel[i][j]==255)   //si es uno(característica)
     {
       for(int ciclo=1;ciclo<=val_div/2;ciclo++)  //ciclo angulo de .1 a 180
       {
         angulo=((2.0*M_PI)/val_div)*ciclo;  //angulo del vector en rad
         get_pol(x,y,angulo,pBitmap);    //obtiene polares con x,y cartesianas
       }
     } //si es borde
   }  // end i
 }  //end j
```

Una vez obtenidos los bordes de las imágenes izquierda y derecha y sus correspondientes vectores, es posible aplicar un algoritmo para el emparejamiento de las características, para esto se estudió el algoritmo tradicional matching y el algoritmo pixel by pixel de Birchfield. Una característica de este sistema es que muchos algoritmos usan únicamente el nivel de gris para el emparejamiento estéreo, en este trabajo, además de trabajar con la escala de gris, se hace uso de la orientación de borde. Para evitar diferencias positivas y negativas, en nuestro algoritmo obtenemos el valor absoluto de la diferencia antes de realizar la adición de cada una de las diferencias.

En lugar de usar el valor en el borde mismo, se usó la comparación del pixel izquierdo y derecho; es decir, se comparó el pixel izquierdo de la imagen izquierda con el pixel izquierdo de la imagen derecha y lo mismo pero con los pixeles derechos, con esto se obtuvo un emparejamiento más cercano y se redujo el error por la alinealidad del sensor.

Para realizar el matching típico el algoritmo es el siguiente:

1.1 Obtener el arreglo de bordes del renglón de la imagen Izquierda.

1.2 Obtener el arreglo de los bordes del renglón de la imagen Derecha.

1.3 Tratar de encontrar pares similares (todos vs todos) y se asocian con el más similar (menor diferencia) y menor a cierto umbral. Tomando en cuenta el pixel central.

1.4 Quitar las características no compatibles o sobrantes (sin par)

1.5 Cada par (match) se almacena para mas adelante obtener las coordenadas X, Y, Z tridimensionales correspondientes.

1.6 Todo lo anterior se realiza para los diferentes renglones.

Para realizar la implantación modificada del algoritmo matching pixel by pixel de Birchfield se uso lo siguiente:

2.1 Obtener el arreglo de bordes del renglón de la imagen Izquierda.

2.2 Obtener el arreglo de los bordes del renglón de la imagen Derecha.

2.3 Tratar de encontrar pares similares (todos vs todos) y se asocian con el más similar (menor diferencia) y menor a cierto umbral.

2.4 Probar los 2 lados, es decir, se comparan los pixeles izquierdos de las dos imágenes y los pixeles derechos en las dos imágenes y el resultado es la menor diferencia entre los izquierdos y los derechos.

2.5 En nuestro caso se hace una mejora respecto al algoritmo tradicional donde únicamente se prueban los niveles de gris y en este caso se realiza un chequeo de la inclinación de nuestra característica, los resultados de esta mejora se pueden observar en el capítulo 5, resultados

2.6 Quitar las características no compatibles o sobrantes (sin par).

2.7 Cada par (match) se almacena para mas adelante obtener las coordenadas *X*, *Y*, y *Z* tridimensionales correspondientes.

2.8 Todo lo anterior se realiza para los diferentes renglones.

Para realizar el matching en lo que respecta a la implantación del sistema del algoritmo de emparejamiento estéreo utilizado, fue básicamente el de búsqueda hacia delante. Que como se mencionó anteriormente consiste en ir almacenando la menor diferencia entre 2 puntos y comparar ésta contra otra comparación, al final se obtiene el matching de los puntos que tienen una menor diferencia entre sí. La razón para haber utilizado este tipo de matching fué que aunque es más intuitivo el de búsqueda hacia atrás, el de hacia delante resulta en un tiempo de corrida menor. [Birchfield '96]

El segmento de código de dicha implantación es mostrado a continuación; lo que realiza es lo siguiente: como primer paso localiza todos los puntos considerados como bordes y los guarda en vectores, uno para la imagen izquierda y otro para la imagen derecha llamados *car1* y *car2*; una vez obtenidos estos arreglos se procede a realizar el emparejamiento, se tienen 2 subrutinas y se usan dependiendo si el arreglo mas pequeño es el de la imagen izquierda o el de la derecha respectivamente y es comparado el arreglo mas pequeño contra todas las características del otro arreglo, una vez realizadas todas las comparaciones queda asimilado el par con una menor diferencia; esta diferencia es obtenida con la función diferencia y es explicada mas adelante.

```
for (int y = 1; y < Bitmap->Bitmap->Height; y=y+25) //recorrido cada y líneas
 {
  for (int x = 1; x < (Bitmap->Bitmap->Width); x++)
  {
    if(pixel[x][y]==255) car1[c1++]=x;//arreglo imagenes izq
    if(pixel[x][y]==255) car2[c2++]=x;//arreglo imagenes der
  }
    if(c1<c2)  //si la izquierda tiene menos
    {
     for(int ciclo1=0;ciclo1<c1;ciclo1++)
     {
       for(int ciclo2=0;ciclo2<c2;ciclo2++)
```

```
        {
          result=diferencia(pixel_izq[car1[ciclo1]-1][y],pixel_izq[car1[ciclo1]-1][y],pixel_izq[car1[ciclo1]-
1][y],pixel_der[car2[ciclo2]-1][y],pixel_der[car2[ciclo2]-1][y],pixel_der[car2[ciclo2]-1][y],car1[ciclo1],car2[ciclo2],y,1);
          if(result<menor_carac)
          {
            guarda pares en arreglo pares
            menor_carac=result;
          }
        }
        ptr_pares++;              //incrementa para siguiente match
      }
    }
    else   //(c2<c1)  //si la derecha tiene menos
    {
      for(int ciclo2=0;ciclo2<c2;ciclo2++)
      {
        for(int ciclo1=0;ciclo1<c1;ciclo1++)
        {
result=diferencia(pixel_izq[car1[ciclo1]-1][y],pixel_izq[car1[ciclo1]-1][y],pixel_izq[car1[ciclo1]-1][y],pixel_der[car2[ciclo2]-
1][y],pixel_der[car2[ciclo2]-1][y],pixel_der[car2[ciclo2]-1][y],car1[ciclo1],car2[ciclo2],y,1);
          if(result<menor_carac)
          {
            guarda pares en arreglo pares
            menor_carac=result;
          }
        }
        ptr_pares++;              //incrementa para siguiente match
      }
    }
  } // si ya esta normalizado (cada uno con un par)
 }   //fin y
```

Para obtener la diferencia se probaron varias formas, una es comparar incluyendo el pixel de en medio, otra únicamente inlcuyendo los bordes y otra bordes e inclinación; así la función para obtener la diferencia entre las características se implantó como a continuación se muestra, donde en primer lugar se obtienen las inclinaciones del par de características que se analiza; también se muestra las distintas formas implantadas para obtener la diferencia en las características; la primera, tomando en cuenta los pixeles izquierdo, derecho y central, segunda sin tomar en cuenta el pixel central y sin tomar en cuenta la inclinación y por último, sin tomar en cuenta el pixel central, pero considerando la inclinación; esta última es la que finalmente resultó tener mejores respuestas y por lo tanto la que utiliza el sistema.

```
if(s_arr[0][xi][y][ciclo-1]==true)
    inci=((2.0*M_PI)/(thres_z*4))*ciclo;
```

```
    if(s_arr[1][xd][y][ciclo-1]==true)
      incd=((2.0*M_PI)/(thres_z*4))*ciclo;
  }
  //result=abs(x1i-x1d)+abs(x0i-x0d)+abs(x2i-x2d);
  //result=abs(x0i-x0d)+abs(x2i-x2d);
  result=abs(x0i-x0d)+abs(x2i-x2d)+abs(inci-incd)
```

Para obtener las coordenadas 3D, el algoritmo matching regresa un arreglo con todas las coordenadas 2D emparejadas, el algoritmo 2D a 3D se vale de este arreglo para convertirlo a 3D, lo que realiza es recorrer el arreglo y a cada par de coordenadas le aplica la ecuación descrita en el capítulo 3, con esto se obtiene las coordenadas 3D.

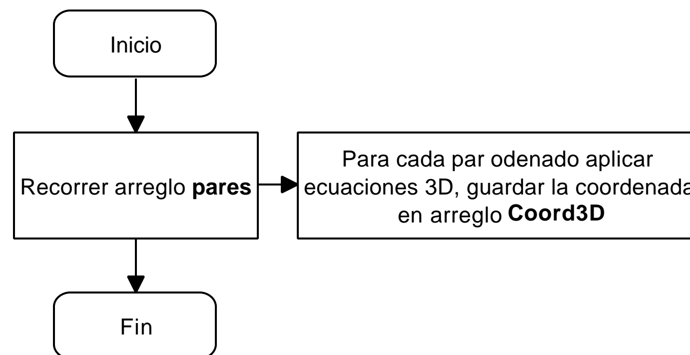

Fig 4.12 Diagrama de flujo Transformación puntos 2D a coordenadas 3D

```
for(int ciclo1=0;ciclo1<ptr_pares;ciclo1++)//recorre arreglo pares
  {
    xl=pares[ciclo1][0];
    xr=pares[ciclo1][2];
    y=pares[ciclo1][1];
    xl=xl-cx;
    xr=xr-cx;
    sd=stereo_disp=xl-xr;
    y=cy-y;
    X=floor(((D*(xr+xl))/(2.0*(xl-xr)))+.5); //formula X
    Y=floor(((D*y)/(xl-xr))+0.5); //formula Y
    if(thres_cam==7)
      Z=floor(((7*575.0)/(xl-xr))+0.5);
     if(thres_cam==10)
        Z=floor(((10*700.0)/(xl-xr))+0.5);
      if(thres_cam==14)
        Z=floor(((14*620.0)/(xl-xr))+0.5);

    Coord3D[ciclo1][0]=X;
    Coord3D[ciclo1][1]=Y;
```

```
    Coord3D[ciclo1][2]=Z;
}
```

Como puede observarse en el código anterior, en primer lugar las coordenadas *xl*, *xr* y *y* son obtenidas del arreglo pares que contiene las características emparejadas, a continuación éstas están con referencia a coordenadas en imagen y tienen que ser convertidas a su equivalente en coordenada cartesiana en la imagen, por lo que se les resta *cx* y suma *cy* respectivamente. Una vez obtenidos estos datos, se aplican las fórmulas deducidas en el cap. 3 y con ésto se obtienen las coordenadas *X*, *Y* respectivamente; también puede observarse que para obtener la coordenada Z, se tienen 3 fórmulas que son seleccionadas de acuerdo a la distancia entre las cámaras. Z se obtiene por medio de una ecuación y su uso es explicado en el capítulo 5 y la coordenada 3D obtenida finalmente se envía al arreglo *Coord 3D*.

4.3.3 Salida del sistema

Como salida de nuestro sistema obtenemos un arreglo con puntos tridimensionales que pueden ser graficados por medio de Open Gl para modelar el objeto u objetos detectados en 3D

Open GL significa Open Graphics Library, es decir: que son librerías de acceso abierto para realizar graficación.

Para realizar la representación de un polígono, por ejemplo únicamente es necesario proporcionar 3 puntos y con el menos 4 polígonos es posible construir objetos tridimensionales.

Los puntos obtenidos anteriormente pueden ser utilizados con varios propósitos, el implantado es la representación burda de objetos y obstáculos de el medio que se explora, mas allá de eso y al ser usado junto con autómatas, pueden ayudarles para la predicción de sus movimientos.