INSTITUTO POLITECNICO NACIONAL
ESCUELA SUPERIOR DE INGENIERIA MECANICA Y ELECTRICA

INGENIERÍA EN COMUNICACIONES Y ELECTRÓNICA

"PROTÓTIPO VIRTUAL PARA UNA DECORADORA DE PASTELES USANDO CONTROL NUMÉRICO POR COMPUTADORA"

T E S I S

QUE PARA OBTENER EL TÍTULO DE
INGENIERO EN COMUNICACIONES Y ELECTRÓNICA

PRESENTA:

HERÓN LEONEL PORTILLA VELÁZQUEZ

ASESOR:

ING. ARMANDO MANCILLA LEÓN

MÉXICO, D.F. 2009

INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA Y ELECTRICA

UNIDAD PROFESIONAL "ADOLFO LÓPEZ MATEOS"

T E M A D E T E S I S

QUE PARA OBTENER EL TITULO DE
POR LA OPCIÓN DE TITULACIÓN
DEBERA(N) DESARROLLAR

INGENIERO EN COMUNICACIONES Y ELECTRÓNICA
TESIS Y EXAMEN ORAL INDIVIDUAL
C.- HERÓN LEONEL PORTILLA VELÁZQUEZ

"PROTOTIPO VIRTUAL PARA UNA DECORADORA DE PASTELES USANDO CONTROL NÚMERIO POR COMPUTADORA"

DESARRROLLAR UNA INTERFAZ GRÁFICA QUE PROCESE IMAGENES, FIGURAS Y SIMULE EN PANTALLA LOS DATOS QUE SERÁN ENVIADOS A UNA MÁQUINA DE CONTROL NUMÉRICO.

- INTRODUCCIÓN
- ANÁLISIS FUNCIONAL DE PROTOTIPO
- INTERFAZ GRÁFICA
- VECTORIZACIÓN
- SIMULADOR VIRTUAL
- PUERTO DE DATOS (PUERTO PARALELO)

MÉXICO D.F., 15 DE ABRIL DE 2009

A S E S O R

ING. ARMANDO MANCILLA LEÓN.

M. EN C. SALVADOR RICARDO MENESES GONZÁLEZ
JEFE DEL DEPARTAMENTO ACADÉMICO
DE INGENIERÍA EN COMUNICACIONES Y ELECTRÓNICA

INSTITUTO POLITECNICO NACIONAL
ESCUELA SUPERIOR DE INGENIERIA MECANICA Y ELECTRICA

INGENIERÍA EN COMUNICACIONES Y ELECTRÓNICA

Trabajo Terminal

"Protótipo virtual para una decoradora de pasteles usando Control Numérico por Computadora"

Que para obtener el Título de
"Ingeniero en Comunicaciones y Electrónica"

Presenta:
Herón Leonel Portilla Velázquez

Asesores:

Ing. Armando Mancilla León.

Presidente del Jurado	**Secretario del Jurado**
Ing. Héctor Piña Canales	**M.C. Samuel Borrego Mora**

México D.F. Abril 2009

INDICE

ÍNDICE DE FIGURAS.

Figuras Capitulo 1.

Figuras Capitulo 2.

Figuras Capitulo 3.

Figuras Capitulo 4.

Figuras Capitulo 5.

Figuras Capitulo 6.

ÍNDICE DE TABLAS.

AGRADECIMIENTOS

En esta oportunidad quiero agradecer a Dios y tantas personas que me han ayudado a culminar una de mis metas de mucha importancia para mi vida.

Definitivamente, Dios, mi guía, mi fin ultimo; sabes cuanto te agradezco por la salud, por la inteligencia y por tantas bendiciones que me han permitido lograr este anhelo en esta etapa de mi vida. Se que después de esta bendición mi responsabilidad con la humanidad a crecido, pero también se que no me abandonas y estarás a mi lado lleno de alegría para seguir construyendo mi camino.

Gracias a mi madre Violeta Velázquez Romero y mi padre Herón E. Portilla Bonilla, por que nunca me ha faltado su amor y confianza, fuerza que me ha impulsado a seguir adelante en los momentos más difíciles. ¡Padre y Madre, gracias por permitirme poder usar la palabra Familia y saber su significado, los amo!

Gracias hermanos, me han apoyado sin darse cuenta, sus consejos, sus palabras, su amor, me acompañaron en muchos procesos de mi vida. Gracias hermanito Pepe, tu sabes cuanto apoyo me has brindado para conseguir esta meta, me has enseñado a compartir frustraciones y triunfos para que no queden en el olvido.

Gracias al Instituto Politécnico Nacional, por permitirme ser parte de la formación de ingenieros, por proporcionarme la educación y una casa para desarrollarme como gente productiva. Gracias por incluir gente que ha influido en mi desarrollo emocional e intelectual, preocupados por el transmitir el conocimiento.

Y a todas aquellas personas que de una u otra forma, colaboraron o participaron en la realización de esta investigación, hago extensivo mi más sincero agradecimiento.

RESUMEN

El prototipo virtual para una decoradora de pasteles usando Control Numérico por Computadora (CNC), consisten en una interfaz gráfica capaz de procesar imágenes digitales, usar herramientas de dibujo básico y simular el efecto que tendrá la información adquirida por el procesamiento en una máquina de control numérico por computadora con un sistema de inyección para depositar fluido (por ejemplo merengue) sobre la superficie de un pastel.

El procesamiento de las imágenes se enfoca en la detección de contornos de la imagen y esta condicionado a las imágenes que se usan frecuentemente en la decoración de pasteles, como son las caricaturas, por lo que el procesamiento resulta óptimo cuando se trata de imágenes con pocos efectos de luz, sombra y degradación de color.

Las herramientas de dibujo básico para este prototipo, son herramientas para dibujar líneas, rectángulos, círculos, óvalos, texto y forma libre, por medio del cursor.

La simulación consiste en una presentación en pantalla de una mesa de decoración que cuenta con un eje de desplazamiento en “X”, un eje en “Y” y un sistema de inyección. Estos desplazamientos tendrán efecto siguiendo la información obtenida en el proceso de vectorización, el resultado obtenido durante la simulación es el resultado que se espera obtener en un dispositivo de control numérico real.

OBJETIVO GENERAL.

Desarrollar una interfaz gráfica que procese imágenes digitales, dibuje figuras geométricas y simule en pantalla a través de una decoradora virtual, los datos adquiridos en el procesamiento de imagen o trazado de figuras. De tal manera que la información simulada pueda ser enviada a través del puerto paralelo a una máquina de control numérico por computadora, para decorar un pastel.

Objetivos Particulares.

- Realizar una rutina para detectar los bordes de una imagen, considerando que debe tener un píxel de grosor el resultado de la detección y la menor cantidad de pixeles no pertenecientes a la imagen.

- Realizar una rutina para convertir los contornos en vectores de desplazamiento y que puedan ser datos que pueda interpretar una unidad de control de una máquina de control numérico (CNC).

- Realizar una rutina para crear figuras en pantalla y convertirlas en datos para una máquina de CNC.

- Diseñar una rutina para simular una máquina de 3 ejes con un sistema de inyección utilizando en la simulación los datos adquiridos en el proceso de vectorización.

- Programar el puerto paralelo para enviar datos a una máquina de CNC.

JUSTIFICACIÓN

Con la diversidad de máquinas de control numérico que existen en la actualidad es complicado encontrar una aplicación de software que se acople a todos los diseños industriales y, las aplicaciones que logran acoplarse a casi cualquier diseño, generalmente son complejas de usar y muchas veces costosas.

En este trabajo, se desarrollará una aplicación específica para poder manipular una máquina con una aplicación específica, sin dejar la posibilidad en un futuro de ser adaptable a otro dispositivo diferente con la menor complejidad posible.

El desarrollo de este prototipo está enfocado en el desarrollo de una aplicación para la decoración de pasteles, la intensión esta dedicada principalmente a los empleados de una repostería, que generalmente no cuentan con los conocimientos de computación para entender un paquete de control numérico como los que se ofrecen en la actualidad, pero si poseen los conocimientos para copiar, pegar, abrir un archivo y mandar a imprimirlo. Esta flexibilidad es la que sustenta el desarrollo de esta aplicación.

Logrando el desarrollo de este objetivo se pueden obtener dos cosas:

- Una aplicación lista para ser adaptada en algún dispositivo de control numérico.
- Una aplicación pensada en el desarrollo de un prototipo de control numérico diseñado para la decoración de un pastel.

La primera opción es inmediata cuando se ha logrado el objetivo principal, pero no todas las empresas podrían solventar el precio de una adaptación tan costosa, debido a que la mayoría de los aparatos de control numérico se producen en el extranjero y en México se venden a un precio excesivo.

La segunda opción viene a ser la idea principal, del desarrollo de este prototipo, tener una aplicación de control numérico para después desarrollar una máquina de control numérico especializada en la decoración de tortas de pastel. Esto al final traería múltiples beneficios:

- Posible reducción de costos.
- Se logra un producto autóctono, con posibilidades de cubrir una necesidad mexicana, como lo es la exportación.
- Reduciendo su precio para la venta en México se apoya a las PyMes dándoles la capacidad de adquisición.
- La sustitución del esfuerzo físico que emplea el decorador por la acción de activar y monitorear el trabajo de la maquina.
- La velocidad y calidad del proceso se mejora.

Para entender los puntos mencionados, se detallan las siguientes justificaciones al proyecto.

Apoyo a las PyMes y sociedad trabajadora.

En la industria de los pasteles, las pequeñas empresas cuentan solo con los utensilios necesarios para la elaboración de pasteles, utensilios que adquieren un alcance limitado por la habilidad del hombre, pero que son la única opción cuando se cuenta con un capital ajustado al precio de estas herramientas. El grabado de una imagen sobre una torta de pastel para las pequeñas empresas, resulta un proceso artesanal, pues solo requiere de las manos del decorador y el beneficio que aporta la herramienta para decorar, de igual forma este proceso es limitado por la habilidad del decorador.

El problema de la decoración manual, empieza cuando el proceso se vuelve monótono y desgastante para la persona. La habilidad no se pierde, pero si las fuerzas y la capacidades físicas, como son la vista y articulaciones (muy utilizadas en el proceso de decoración), pues no existe una postura o posición física que no cause fatiga física.

Esto, en algún momento, exige un cambio de personal, lo cual es algo que no se puede evaluar fácilmente para una pequeña empresa que cuenta con pocos recursos de administración y gente capacitada, pero cuando ocurre, la estructura o funcionamiento interno puede sufrir caídas, pues se tiene que adaptar el nuevo empleado a las necesidades y habilidades que poseía el anterior, y una vez logrado el objetivo, solo se espera a que el trabajo se vuelva monótono y desgastante para que un nuevo empleado sea incorporado, por ineficiencia o fatiga del anterior.

En algunas pequeñas y medianas empresas algo organizadas, se adquiere equipo o herramientas de apoyo en la etapa de decoración al igual que en otras etapas de producción.

En realidad para que la fatiga ocurra, requiere de unos meses o quizá años, pero es mejor evitar que esto ocurra, pues no es reversible que la persona recupere su salud visual al emplearse tanto tiempo en la labor de la decoración. Además, esto de alguna manera puede solucionarse; turnándose esta labor entre varios para que el cuerpo descanse y se reparta el desgaste. Pero finalmente, esto requiere de más personal.

Mejora la velocidad y calidad

Si se considera una superficie de 50cm x 40cm, que corresponde a una torta de pastel común para fiestas.

El tiempo de decoración manual de los trazos de una imagen, se consiguen entre los 10 a 30 minutos dependiendo de la complejidad del dibujo y la habilidad del decorador cuando se emplea solo una persona en este proceso.

El tiempo de decoración manual apoyada por un dispositivo guía, reduce el tiempo entre los 5 y 10 minutos, pero no reduce el desgaste físico en el decorador ni el tiempo invertido en adquirir la imagen guía.

Lograr trazos definidos, a una velocidad continua y con el doble de velocidad de un trazo manual ó mas, se puede tomar en consideración para decidirse a emplear una máquina que lo haga, además de aminorar el desgaste físico del empleado.

Competencia en el mercado

Una razón más general para emplear una máquina de este tipo, se da desde su creación misma, pues la competencia en el mercado de las máquinas automatizadas va creciendo constantemente, y se introducen máquinas en el mercado para elaborar cualquier tipo de producto, con el fin de hacer cada vez mas eficientes, rápido y de calidad la producción, por ejemplo, las máquinas que produce la empresa UNFILLER® para la decoración y elaboración de productos de repostería, buscan un incremento en la producción y la calidad.

Con esta idea y la actual información de máquinas de control numérico en el mercado, es competente diseñar y construir una maquina tipo CNC que realice la decoración de una torta de pastel utilizando la mayor flexibilidad posible y para lograr esto, es necesario el desarrollo de una aplicación que pueda controlar el proceso.

TEMAS

El presente trabajo esta dividido en 6 capítulos.

En el capítulo 1 se da una introducción a las técnicas de decoración de pasteles, empezando con una técnica artesanal, pasando a una forma mediamente avanzada y terminando en un proceso automatizado.

En el capítulo 2 se describe en breve en análisis funcional del prototipo, mostrando con una serie de pasos, las acciones que debe realizar la interfaz.

En el capítulo 3 se da una introducción al ambiente de desarrollo donde se ejecuta el programa, se desarrollan diversos filtros para la detección de contornos en una imagen y se selecciona en base a los resultados un filtro que proporcione una imagen adecuada para la vectorización (tratado en el capítulo 4).

En el capítulo 4 se desarrolla el algoritmo para vectorizar los contornos de una imagen y se desarrollan herramientas para generar contornos en el momento de aplicarlas (trazador de líneas, rectángulos, elipses, letras y trazado libre). Se crean dos funciones para la manipulación de archivos con vectores ordenados, uno para guardar los vectores generados por el usuario y otra función para abrir los archivos de vectores guardados.

En el capítulo 5, se desarrolla un simulador que interpreta los datos obtenidos en la vectorización como código binario. El simulador se ejecuta en una ventana de WPF (del ingles *Windows Presentation Foundation*), con vista tridimensional de las partes que componen al simulador, el cual se comporta como un dispositivo de control numérico.

En el capítulo 6, se describen líneas de código en C# para enviar información a través del puerto paralelo además que se describen las partes de puerto paralelo que son usadas para enviar la información a un dispositivo externo.

Capitulo 1

Introducción: Software y decoración de una torta de pastel.

1. PROTOTIPO VIRTUAL PARA UNA DECORADORA DE PASTELES USANDO CONTROL NUMÉRICO POR COMPUTADORA (CNC).

El prototipo virtual para una decoradora de pasteles de CNC, consiste en una interfaz gráfica con la capacidad de procesar imágenes, generar figuras y convertir la información en vectores, para simular esta información a través de un dispositivo CNC virtual. Ya que se ha demostrado a través del dispositivo virtual su funcionalidad, se puede proceder a aplicar a un dispositivo real, como podría ser un router[1] CNC con una adaptación de un sistema de inyección de fluido viscoso.

1.1 Software de control numérico.

Son diversos los programas que se han desarrollado sobre control numérico por computadora. A partir de 1955, muchos de ellos se han desarrollado a base de grandes investigaciones e ingeniería, muchos arrojan resultados óptimos en el proceso de producción, lamentablemente las investigaciones y desarrollo de aplicaciones, generalmente están enfocadas en el desarrollo de piezas y procesos pesados, en los que se agregan herramientas sofisticadas para cada proceso. Es por este motivo, que es difícil emplear alguna aplicación existen en el mercado a un prototipo para decorar pasteles.

Las aplicaciones de diseño para la creación de vectores, se les conoce como aplicaciones CAD, del ingles *Computer Aided Design (diseño asistido por computadora),* y los vectores son los puntos que presenta el ordenador para formar líneas de dibujo en un proyecto y así representar la imagen de un objeto. La cantidad de aplicaciones del tipo CAD es amplia, tanto en vectores de 2 dimensiones como los de 3 dimensiones. El problema que presentan estas herramientas esta en su complejidad, pues muchas herramientas se basan en comandos para formar vectores y las que no cuentan con tantas herramientas son difíciles de adaptar a cualquier proceso.

Las aplicaciones que interpretan vectores para usarlos en el control numérico se les conoce como aplicaciones CAM del ingles *computer-aided manufacturing (fabricación asistida por ordenador),* que en base al diseño e instrucciones del ordenador, envían la información a los motores para realizar el proceso de manufactura basándose en la información de los vectores obtenidos en el diseño CAM.

Actualmente se integran los dos sistemas (CAD-CAM) en aplicaciones como: SolidEdge, SolidWorks, CATIA, etc, pero estos programas además de ser muy costosos son complicados de usar y más para personas que no tienen conocimientos de computación.

Por otro lado existen aplicaciones libres de licencia y disponibles en la red, especializadas en el Control Numérico, por ejemplo el MACH3. Esta aplicación puede ser elegida como interfaz gráfica, ya que puede resolver el problema del trazado de las trayectorias de una imagen por medio de un algoritmo de segmentación, además de convertir la información en códigos GyM[2] para ser utilizada en el proceso de manufactura.

[1] Un router CNC, es una máquina-herramienta que unida a un ordenador y el programa adecuado, traza formas en una mesa rectangular, siguiendo el patrón de diseño de la figura creada en el programa.
[2] Los códigos GyM, son los códigos de programación que incluyen algunas máquinas de CNC y están dirigidos a la manipulación de herramientas, velocidad y posición.

Esta aplicación es muy popular por sus funciones, pues se adapta a casi cualquier tipo de máquina CNC en el mercado y cuenta con una diversidad de herramientas como: una cámara web para reconocer imágenes y trazar su forma con el CNC, la mencionada segmentación de imágenes BMP y JPG, etc.

Para conocer la aplicación, en la figura 1.1, se muestra la interfaz gráfica del programa MACH3.

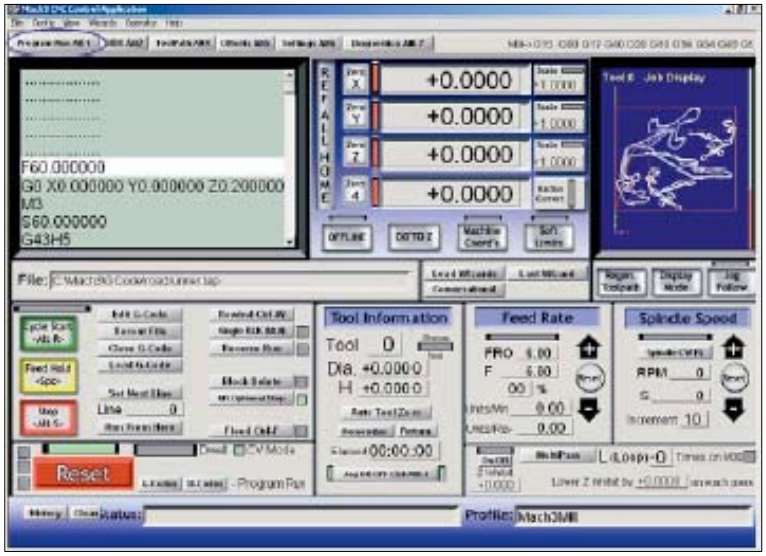


Figura 1.1. Interfaz gráfica de Mach3 ® que simula la consola de una máquina de Control Numérico.

La figura 1.1 muestra los elementos que incluye la interfaz gráfica para operar una máquina de control numérico. A simple vista se puede observar que el programa cuenta con demasiados controles en pantalla, lo que hace difícil su aplicación para usuarios que desconocen la herramienta y más complicado para el usuario que se esta iniciando en el uso del ordenador y las máquinas de control numérico.

La interfaz gráfica que se describe a lo largo de este documento, presenta un ambiente con menos elementos en pantalla, haciendo más sencilla la forma de operar los elementos de diseño y los de transformación de vectores.

1.2 Técnicas de decoración.

Una forma muy sencilla de grabar una imagen sobre un pastel sin necesidad de herramientas costosas, consiste en lo siguiente:

1.- Tener la zona embetunada del pastel que se va a decorar. La secuencia de la figura 1.2, muestra como queda un pastel embetunado.

Figura 1.2. Forma tradicional de decorar un pastel.

Figura 1.2. Secuencia de pasos para embetunar con una espátula una torta de pastel.

2.- Contar con un objeto: delgado, ligero, con punta e higiénico (como un palillo de madera por ejemplo) y trazar sobre la zona embetunada con el objeto, la imagen que se desea dibujar en el pastel.

Figura 1.3. Trazado de una imagen guía con un objeto delgado sobre un pastel.

En la figura 1.3, se muestra un ejemplo donde se han trazado 3 estrellas con un palillo de madera, para que sirvan como guía de trazado.

3.- Sobre la guía formada con el objeto se decora el pastel inyectando con la duya el merengue, de tal manera que adquiera la forma del trazo guía.

Figura 1.4. Decoración del pastel siguiendo una imagen guía trazada sobre el betún.

La figura 1.4, muestra el resultado de decorar con merengue sobre una imagen formada por un palillo de madera en la superficie embetunada de un pastel.

Otra técnica de decoración, es una técnica asistida por una herramienta visual de apoyo, como lo es un proyector de transparencias. En el mercado existen diferentes modelos especiales para la decoración de pasteles. Esta técnica es parecida a la anterior, solo que al

proyectarse el dibujo de la transparencia sobre el pastel, se ahorra el tiempo que se emplea en el paso mostrado en la figura 1.3 y se mejora la calidad de la imagen, pues la copia tiene mayor parecido a la original. La figura 1.5 muestra los pasos para realizar esta técnica.

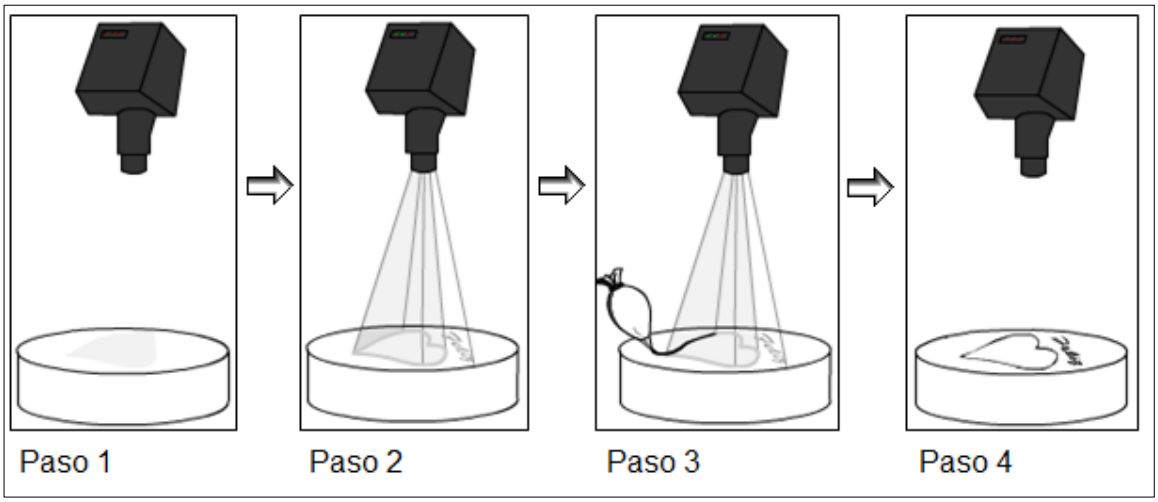


Figura 1.5. Decoración de un pastel asistido por un proyector de transparencias.

En la figura 1.5 paso1, se tiene la torta de pastel embetunada, lista para ser decorada.

En la figura1.5 paso 2, se enfoca la proyección de la transparencia sobre la superficie del pastel.

En la figura 1.5 paso 3, se esta grabando la imagen con una manga pastelera siguiendo el patrón de líneas que se forma con la proyección.

En la figura 1.5 paso 4, se muestra el resultado final.

Un dispositivo proyector de transparencias especial para la decoración, cuesta en el extranjero alrededor de 225.00 dólares, el más económico, por ejemplo el Kopykake 300xk[3], por mencionar alguno. Pero existen otros modelos que alcanzan un valor de hasta 500 dólares.

Decorar con un proyector como guía toma alrededor de 5 minutos para un solo pastel, considerando que ya se tiene la transparencia elaborada, caso contrario a la primera técnica, que requiere más tiempo y habilidad al tener que trazar la imagen guía.

Como se vio en esta técnica, contar con un dispositivo proyector de transparencia, aporta beneficios a una empresa, en cuanto a rapidez y calidad. Pero hay que mencionar que en las dos técnicas, las posturas físicas que se toman para la decoración de un pastel no cambian mucho (ejemplo de la figura 1.6) y los requisitos de enfoque visual y buen manejo de la decoración son necesarios para hacer el trabajo.

[3] Kopukake 300xk. Proyector de transparencias utilizado para la decoración de pasteles, publicado a la venta por internet en la pagina http://www.kopykake.com/ac_projector_300xk.html

Figura 1.6. Posición que toma un decorador para poder grabar una imagen en un pastel.
(© Fotografías de la escuela de Missouri para sordos MSD.

La figura 1.6 muestra que tanto la vista como la postura son afectadas en las dos técnicas, el inciso b y c de la figura 1.6 describen el tipo de técnica usada y las regiones afectadas por la postura.

Observando la posición que toma un decorador y considerando la cantidad de pasteles que se piden en un día en algunas pastelerías, se puede referir como malo el desgaste físico que requiere una persona para realizar estas rutinas.

Este desgaste es un problema que se pretende solucionar con la implementación de este trabajo a las máquinas de control numérico.

1.3 Máquinas industriales de decoración.

La decoración de pasteles con máquinas industriales automatizadas, proporciona beneficios de producción, cuando las cantidades de materia que se manejan son elevadas, pues esto permite cubrir el costo de un equipo.

Algunas máquinas de decorado industrial que se conocen, son llamadas depositadoras, funcionan a través de electroválvulas[4]. Su aplicación es limitada, pues el trazado de líneas para formar una imagen, no entra dentro de sus funciones.

La figura 1.7, se muestra una secuencia de imágenes, donde se puede ver el trabajo que realiza una depositadora semiautomática en un pastel de un ¼.

Figura 1.7. Secuencia de imágenes de un pastel decorado por medio de rotación y depositado semiautomático.

[4] Una electroválvula es un dispositivo diseñado para controlar el flujo de un fluido a través de un conducto.

En la figura 1.7 se observa una secuencia de imágenes, la base del pastel gira en su centro, mientras que por medio de una electroválvula manipulada por un operador, se depositan puntos de merengue el la superficie rotante del pastel, para dar el efecto de decorado mostrado.

La idea principal de este trabajo, consiste en sustituir el efecto semiautomático por el efecto de control numérico, eliminando la limitante que hay, incluso hasta en un proceso automático, en los cuales solo se decora de forma radial sobre el pastel o de forma preestablecida por la máquina decoradora[5].

[5] Para ver el video-catalogo de ejemplos de las máquinas de decoración de pasteles que ofrece Unfiller®, visitar la pagina http://www.unifiller.com/products.php#

Capitulo 2

Análisis funcional del proyecto.

2. ANÁLISIS FUNCIONAL DEL PROTOTIPO.

Para empezar a resolver el problema se separaron las partes funcionales que componen el prototipo. Para lograr esto, fue necesario un análisis funcional de las partes que actúan en el proceso. A continuación se describe el proceso, para tener una idea de las partes funcionales que están involucradas.

Paso 1: Se adquiere una imagen digital como la mostrada en la figura 2.1.

Figura 2.1. Imagen digital inicial.

La imagen digital puede ser de formato BMP, JPG, GIF o PNG, parecida a la mostrada en la figura 2.1, con pocos detalles de luz y degradación de color, para optimizar el proceso de filtrado y conversión de vectores.

Paso 2: Por medio de un procesamiento de imagen como filtro de canny ó binariazión de la imagen, se filtran los bordes de la imagen original para obtener la imagen de la figura 2.2.

Figura 2.2. Contornos de la imagen inicial.

La figura 2.2 muestra el resultado de aplicar un filtro de detección de contornos a la imagen de la figura 2.1.

Paso 3: Se invierte el color de la imagen para obtener el resultado de la figura 2.3.

Figura 2.3. Imagen invertida en colores.

La figura 2.3 muestra el resultado de invertir los colores de la imagen 2.2, esto es necesario, por que el programa escanea los pixeles negros para ordenarlos en una matriz, y como el proceso de filtrado mostrado en la figura 2.2 produce un fondo negro, no se estarían ordenando en la matriz los bordes sino el fondo, produciendo a la vez un error al no existir una trayectoria definida.

Paso 4: Las trayectorias formadas por pixeles negros se convierten a vectores ordenados y objetos, tal y como lo representa la figura 2.4.

Figura 2.4. Polilíneas que forman la imagen.

La figura 2.4 es una representación de líneas que forman una imagen, esto se consigue con una matriz que almacena y ordena la posición de los pixeles negros, de tal manera que tengan vecindad cada posición y al momento de emplear una línea, forme un segmento de la imagen entre el punto inicial de la línea y el final, con la finalidad de que al recorrer toda la matriz de posiciones ordenada, resulte la misma imagen de la figura 2.3.

Segunda forma de obtener vectores ordenados.

Otra forma de adquirir información de las imágenes es generándolas, para lograr esto será necesario elaborar herramientas para generar vectores. Las herramientas utilizadas en el programa serán las que aparecen comúnmente en un programa de dibujo como son: la línea, el rectángulo, la elipse y el lápiz, tal y como lo muestra la figura 2.5.

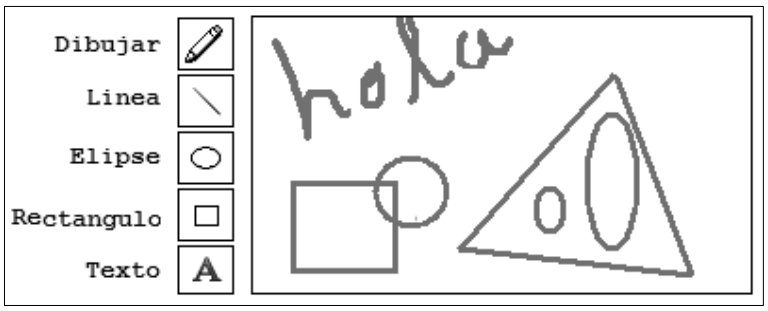


Figura 2.5. Herramientas de dibujo para la interfaz gráfica.

La figura 2.5, muestra las herramientas con las que se consigue dibujar la información en el momento de aplicar cada una de ellas, por lo que ya no es necesaria la detección de bordes ni las operaciones para ordenar la posición de los vectores, ya que al momento de hacer un trazo, la información se estará almacenando en una matriz de manera ordenada.

Paso 5: Los valores ordenados de la posición de los pixeles son usados para formar trayectorias como lo muestra la figura 2.6, la cual representa un segmento de una imagen.

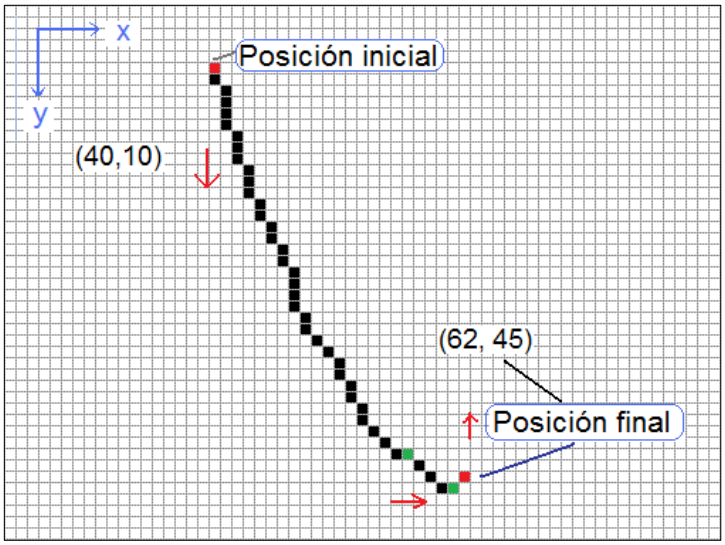


Figura 2.6. Trayectoria formada por líneas.

La figura 2.6, representa una trayectoria formada por múltiples líneas, en la figura 2.6 se puede observar que a partir de la posición inicial todas las trayectorias tienen incrementos continuos en el eje “y” y discontinuos en el eje “x”, pero cuando llega la trayectoria al punto indicado en verde, los incrementos en el eje “y” no son continuos por lo que se determina un cambio de dirección, cuando llega al segundo punto verde ocurre otro cambio de dirección, estos cambios de dirección están indicados por una flecha y sirven para señalar como cambia de dirección la trayectoria, lo cual se determina en el paso 6.

Paso 6: Para determinar como cambia la dirección de la trayectoria formada por líneas se resta la posición del primer vector con la del segundo de manera sucesiva hasta recorrer toda la matriz (como se muestra en el ejemplo de la figura 2.7), el resultado obtenido es una matriz que contiene información de los incrementos y dirección que tiene cada incremento, lo cual puede ser usado como referencia para enviar pulsos a unos motores y moverlos con incrementos de 1, 0 y -1.

pixel n+1 – pixel n		pixel n+1 – pixel n		pixel n+1 – pixel n		x, y	x, y	x, y	x, y
1: pixel 2 - pixel 1	···	1: (40,10)-(40, 9)	···	21: (53,40)-(53,39)		1: (0, 1)	11: (0, 1)	21: (0, 1)	21: (0, 1)
2: pixel 3 - pixel 2	···	2: (41,11)-(40,10)	···	22: (54,41)-(53,40)		2: (1, 1)	12: (1, 1)	22: (1, 1)	22: (1, 1)
3: pixel 4 - pixel 3	···	3: (41,12)-(41,11)	···	23: (55,42)-(54,41)		3: (0, 1)	13: (0, 1)	23: (0, 1)	23: (1, 1)
4: pixel 5 - pixel 4	···	4: (41,13)-(41,12)	···	24: (56,43)-(55,42)		4: (0, 1)	14: (1, 1)	24: (1, 1)	24: (1, 1)
5: pixel 6 - pixel 5	··· =	5: (41,14)-(41,13)	···	25: (57,43)-(56,43)	=	5: (0, 1)	15: (0, 1)	25: (1, 1)	25: (1, 0)
6: pixel 7 - pixel 6	···	6: (42,15)-(41,14)	···	26: (58,44)-(58,43)		6: (1, 1)	16: (1, 1)	26: (1, 1)	26: (1, 1)
7: pixel 8 - pixel 7	···	7: (42,16)-(42,15)	···	27: (59,45)-(58,44)		7: (0, 1)	17: (0, 1)	27: (0, 1)	27: (1, 1)
8: pixel 9 - pixel 8	···	8: (42,17)-(42,16)	···	28: (60,46)-(59,45)		8: (0, 1)	18: (1, 1)	28: (1, 1)	28: (1, 1)
9: pixel 10 - pixel 9	···	9: (43,18)-(42,17)	···	29: (61,46)-(60,46)		9: (1, 1)	19: (0, 1)	29: (0, 1)	29: (1, 0)
10: pixel 11 - pixel 10	···	10: (43,19)-(43,18)	···	30: (62,45)-(61,46)		10: (0, 1)	20: (0, 1)	30: (1, 1)	30: (1,-1)

Figura 2.7. Salto necesario para avanzar de un píxel a otro.

En la figura 2.7, se muestra el resultado de restar las posiciones de un vector con su vecino, dando así el paso necesario en (x, y) para llegar a la otra posición. Las muestras 25 y 29, se encuentran marcadas para indicar un desplazamiento diferente a los anteriores, la muestra 30 para indicar que la curva empieza a caer.

Un valor diferente a 1,-1 ó 0 dentro del resultado de cada resta, indica que comienza una nueva trayectoria o trazo de la imagen.

Se puede reconstruir la imagen de la figura 2.6 solo con usar incrementos en X e Y equivalentes a los valores de los datos obtenidos en la figura 2.7, por ejemplo en el primer punto se tiene incremento en Y de 1, en el segundo punto incremento en X e Y de 1, en el tercer punto incremento en Y de 1, de esta manera hasta recorrer todos los puntos de la matriz. Haciendo el recorrido completo de la matriz de incrementos se obtiene la imagen de la figura 2.8.

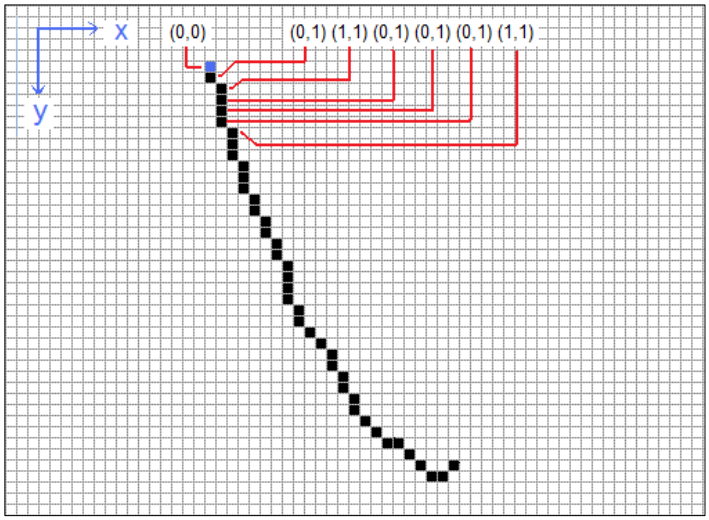


Figura 2.8. Trazado de una trayectoria por medio de incrementos en X e Y.

Paso 7: Los datos obtenidos se utilizan en la simulación para verificar el resultado que proporciona el paso 4, haciendo uso de los elementos mostrados en la figura 2.9.

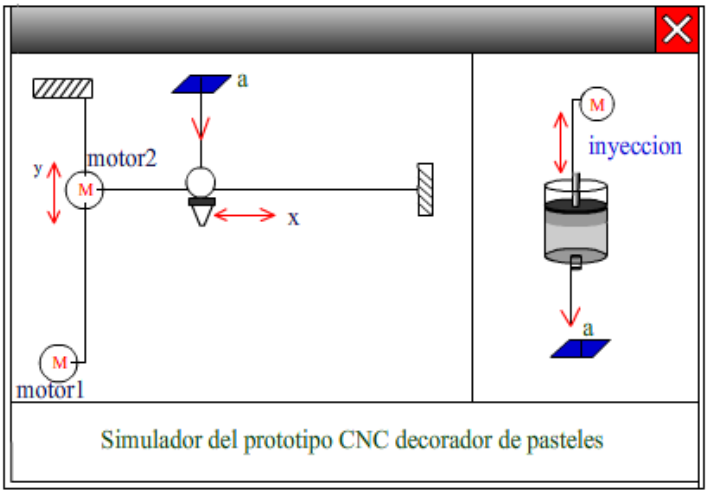


Figura 2.9. Prototipo virtual de una decoradora de pasteles.

La figura 2.9, muestra la idea principal de prototipo virtual, haciendo referencia en la división izquierda a los motores que moverán las partes X-Y de la mesa y que en pantalla se deben simular esos movimientos, como lo haría una máquina real. La división de la parte derecha muestra un sistema de inyección de fluido viscoso, que de acuerdo a los datos obtenidos en los vectores, deberá actuar cuando se este trazando una trayectoria y quedar inactiva cuando solo se este desplazando el cabezal de la duya[6] a otra posición, este parte hace analogía al eje Z, pero solo gira en una dirección cuando esta activado, se detiene cuando no esta activado, y se activa en el otro sentido solo cuando se va a recargar el deposito de merengue, por lo que no interviene en las operaciones realizadas a las imágenes digitales.

Paso 8: Verificada en la simulación la información obtenida en el paso 6, se convierte en pulsos eléctricos que serán enviados por algún puerto de salida del ordenador.

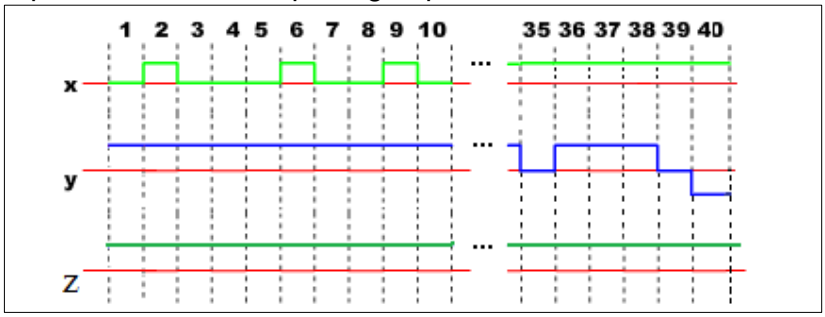


Figura 2.10. Pulsos de salida del ordenador a la unidad de control.

La figura 2.10, representa una de las etapas finales de la aplicación, donde los datos obtenidos en la resta de vecindad se convierten en pulsos eléctricos que salen por el puerto paralelo. Esto se puede ver si se observa la figura 2.7, en la que sus valores están representados en la figura 2.10 en forma de pulsos eléctricos. Como se observa, los pulsos en el eje Z permanecen constantes debido a que el motor de este eje debe estar activo mientras se sigue la trayectoria de la imagen, permanecerá inactivo solo cuando se esté trasladando de posición la duya sin seguir la trayectoria de la imagen.

[6] Duya es el accesorio que se coloca en la punta o parte más pequeña de una manga pastelera.

Paso 9: Se selecciona un puerto de salida como el mostrado en la figura 2.11, se asignan las salidas de datos y las funciones que realizaran.

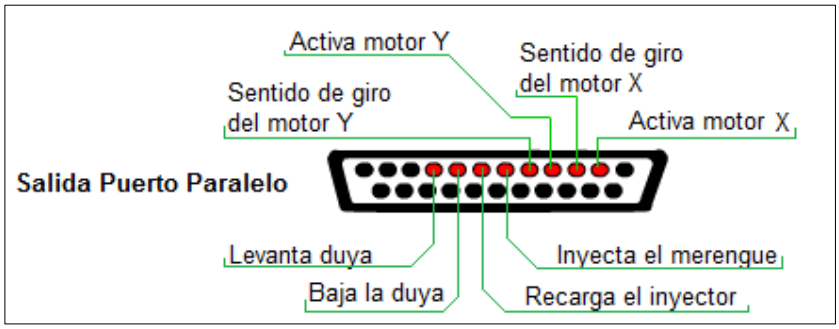


Figura 2.11. Puerto paralelo del ordenador usado para transmitir datos.

En la figura 2.11, se muestra una representación del puerto paralelo, que será usado como medio de comunicación para transmitir los datos en este trabajo. En esta figura se indican los bits de salida del puerto paralelo que pueden ser usados para enviar datos y la manera de usarlos. Dentro de las instrucciones asignadas al puerto paralelo, la salida que levanta la duya y la salida que la baja pueden ser despreciadas, pues este proceso se puede realizar de manera manual, ya que estas salidas solo posicionan la duya a una altura considerable respecto a la superficie del pastel que se va a decorar.

La información enviada por el puerto paralelo, puede ser decodificada por un dispositivo programable (como un microcontrolador) y adaptada a algún dispositivo de desplazamiento basado en motores, ya que esta información será enviada en forma binaria y puede ser usada a conveniencia.

Finalmente, la aplicación debe tener la posibilidad de respaldar la información, por lo que en algún momento puede ser reutilizada. Haciendo esto, se logra obtener una serie de plantillas de vectores para dibujar de forma directa con el simulador o con interacción al puerto usando algún dispositivo externo que pueda interpretar la información.

De acuerdo a lo anterior, se obtuvieron los siguientes requerimientos para realizar dichos pasos, estos requerimientos son los que se usan para realizar el análisis funcional del programa.

- Se requiere que una parte del programa puede leer imágenes, las modifique y las guarde.
- Se requiere una rutina para generar vectores de desplazamiento y control numérico.
- Se necesita una función para guardar la información obtenida en algún medio de almacenamiento.
- Se requiere una simulación virtual de los datos obtenidos en la vectorización.
- Se necesita una parte del programa que pueda enviar información al exterior del ordenador.
- Se requiere representar la visualización de los desplazamientos X-Y en pantalla.

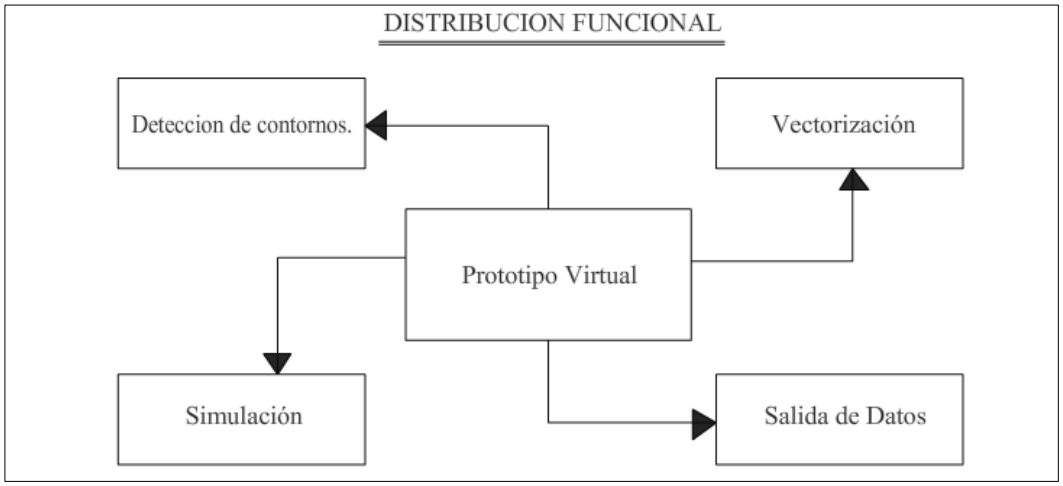


Figura 2.12. Análisis funcional del prototipo.

La figura 2.12, muestra las partes funcionales que integran el prototipo por medio de un análisis funcional. Las partes funcionales, pueden funcionar por separado, y el objetivo final de este método es integrar de manera eficientes las partes para lograr una sola función que cumpla con el objetivo del prototipo.

En los capítulos posteriores se describe detalladamente como se realiza cada uno de los pasos mostrados en este análisis.

Capitulo 3

Interfaz gráfica.

3. INTERFAZ GRÁFICA.

Es el medio visual donde se pondrán las herramientas para extraer la información de las imágenes o el medio para crear la imagen que será convertida en pulsos eléctricos para activar las instrucciones que saldrán por el puerto paralelo.

3.1 Interfaz Gráfica

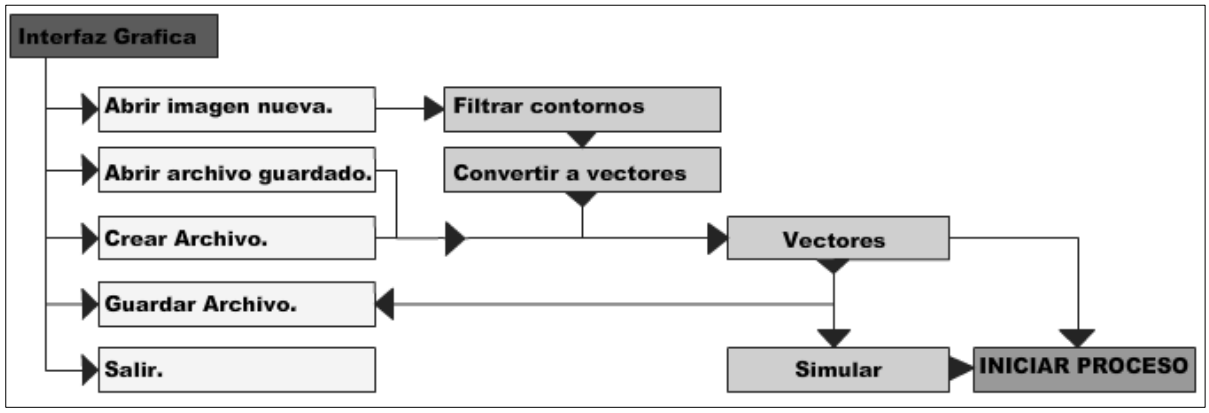


Figura 3.1. Representación del modelo de controles para la interfaz gráfica del sistema.

En la figura 3.1, se muestra un esquema del funcionamiento de controles para operar la interfaz gráfica del sistema. Las instrucciones principales son:

- ***Abrir Imagen Nueva***. Permite adquirir una imagen digital almacenada en el disco duro o algún medio de almacenamiento de información.

- ***Abrir Archivo Guardado***. Permite abrir un archivo que ya ha sido tratado y convertido a vectores.

- ***Crear Archivo.*** Permite abrir un cuadro en blanco, para poder dibujar a mano libre la imagen por medio del cursor, generando en el mismo instante los vectores.

- ***Guardar Archivo.*** Permite guardar un archivo que contiene los vectores resultantes del diseño o procesamiento de imagen previamente realizado.

- ***Salir.*** Permite abandonar la aplicación.

- ***Filtrar contornos.*** Esta función filtra los contornos de una imagen, dando como resultado una imagen formada por múltiples líneas de un mismo color para ser procesada. Solo se aplica a imágenes con formato BMP, JPG, GIF y PNG.

- ***Convertir Vectores***. Es una función que es permitida cuando se ha conseguido obtener los contornos de una imagen. Esta función convierte las líneas que forman la imagen en vectores de desplazamiento.

- ***Vectores***. Es el resultado que se obtiene de convertir las líneas de una imagen a vectores, de abrir un archivo que contiene una lista de vectores ó de crear un archivo a base de un diseño en pantalla por medio del cursor.

- ***Simular.*** Es una función que se permite cuando se ha logrado obtener un archivo de vectores. Esta función permite simular la información antes de disponer de la máquina real para visualizar el posible resultado de llevar a cabo la ejecución del prototipo decorador de pasteles.

- ***INICIAR PROCESO.*** Es la función que permite el inicio de arranque de los motores, permitiendo que la imagen que se muestra en la pantalla de vectores, se empiece a trazar por medio del flujo de señales que se mandan por el puerto de salida del ordenador y que son traducidos por medio de un dispositivo programable.

3.2 Acondicionamiento del Ambiente en Visual C# 2008.

Para empezar a desarrollar la aplicación de interfaz gráfica, se selecciono un compilador y el lenguaje de programación de acuerdo a las necesidades que demanda la aplicación y el conocimiento de las herramientas del lenguaje de programación.

Para este caso se ha seleccionado el lenguaje de programación C#, por que cuenta con las herramientas, el poder de procesamiento para manipular imágenes y operaciones necesarias en el proceso, además de ser una herramienta de programación orientada a objetos. Por otro lado, se podría basar el programa en lenguaje JAVA, por sus amplias librerías, su código abierto y por que es un programa orientado a objetos, pero a la hora de cargar un programa y ejecutarlo, resulta lento en comparación con la velocidad que C# presenta, que para este proyecto, la velocidad es un factor de suma importancia. Comparado con C++, existe una polemica, pues C++ es una herramienta que podría estar por encima de C#, debido a que la mayoría de las aplicaciones de control e interfaz de usuario están basadas en código C++, pero el surgimiento de C# se basa en corregir los errores que presenta C++, como lo son la extracción de librerías y la forma de depurar la información, además, realizar una aplicación basada en C# permite ampliar los conocimientos de una tecnología poderosa en el desarrollo de las tecnologías de la información (TI).

Figura 3.2.1 Herramienta de programación a usar.

La figura 3.2.1, muestra la ventana que aparece cuando se inicia la aplicación de visual C# 2008.

Para empezar a trabajar con una aplicación con ventanas y controles dentro del Visual C# 2008, hay que ir a: ***Archivo → Nuevo Proyecto → Aplicación para Windows.*** Antes de aceptar la plantilla o tipo de proyecto, se asigna un nombre al proyecto en el ítem "*nombre*" y se procede a aceptar.

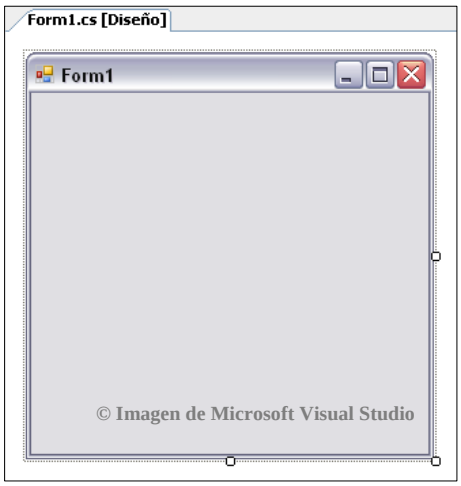


Figura 3.2.2 Ventana de un proyecto en Visual C# 2008.

La figura 3.2.2, muestra la *ventana principal* del programa que ejecuta todas las acciones de este proyecto. Para trabajar con la ventana y las herramientas de visual estudio, se puede acondicionar el ambiente de trabajo de la siguiente manera:

1.- Abrir las ventanas: Cuadro de herramientas, Explorador de Soluciones y Ventana de propiedades, como lo muestran los iconos de la figura 3.2.3.

Figura 3.2.3 Acondicionamiento de Ventanas.

2.- Modificar las siguientes propiedades: "Name", "Text" y "WindowState", de la "ventana principal" usando la ventana de propiedades de Visual C#.

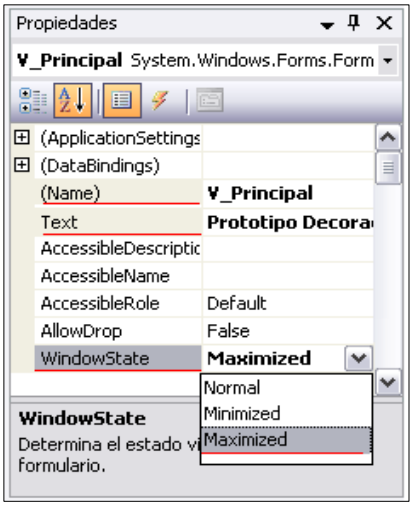


Figura 3.2.4. Ventana de propiedades de un objeto en Visual C# 2008.

En la figura 3.2.4, se muestra la ventana de propiedades de la v*entana principal,* como se observa, aparece un campo llamado *(name)*, este campo permite declarar cual será el nombre de la ventana principal. En el campo *Text* se indica el mensaje que aparecerá en la parte superior izquierda de la *ventana principal*. Y el campo *WindowState* maximiza la ventana al ancho y alto de la pantalla cuando se ejecuta el programa, sin importar que tamaño tenía inicialmente.

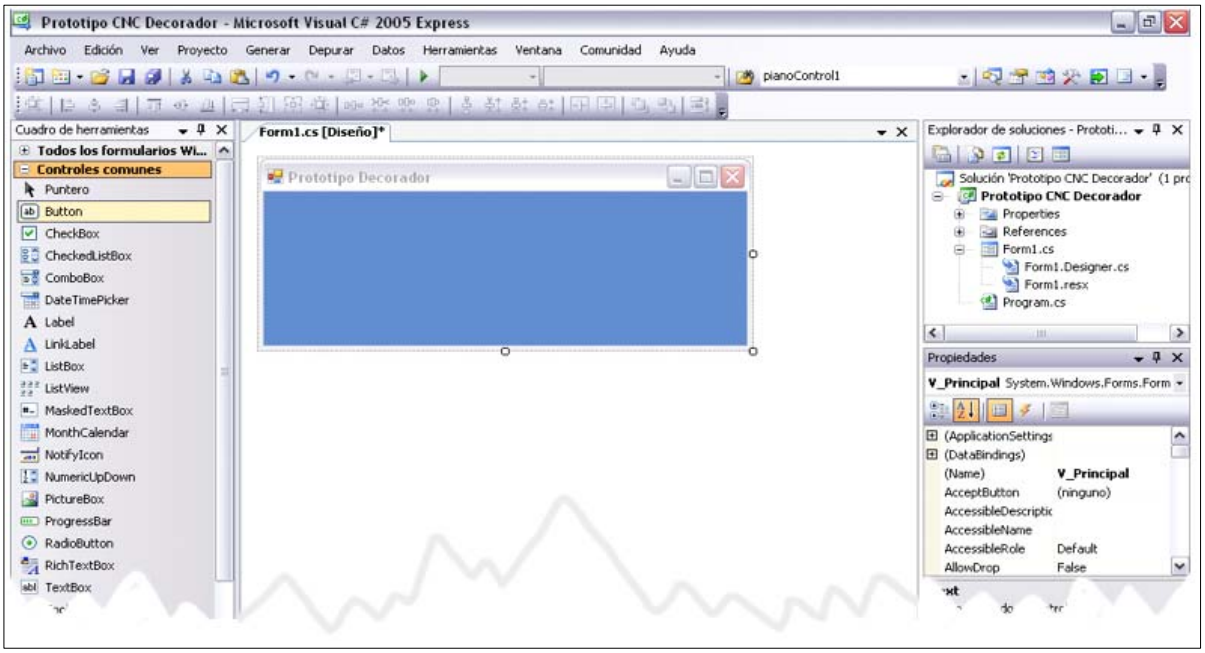


Figura 3.2.5. Acondicionamiento del ambiente de programación en Visual Studio C#.

La figura 3.2.5, muestra como queda el ambiente visual de desarrollo donde se realizará la aplicación de la interfaz gráfica. Acondicionado de esa manera, se puede tener acceso rápido a alguna de las herramientas de diseño y propiedades de los objetos seleccionados dentro del ambiente.

Ahora, siguiendo el modelo de la figura 3.1, se insertan una serie de botones que sirvan para: Abrir una imagen, abrir un archivo de vectores, crear una imagen y salir. Estos botones son el menú principal del programa y pueden ser presentados por medio de un *MenuStrip,* que es una herramienta que crea botones en forma de menú, solo se selecciona y se arrastra sobre *V_Principal* para que la herramienta aparezca. Agregado a esto, se inserta en el primer campo que se forma, una palabra que agrupe a las 5 funciones que se están contemplando, por ejemplo, “Abrir”. Sucesivamente se llenan los campos con los nombres de las 5 funciones mencionadas en la figura 3.1. La figura 3.2.6, muestra como podría quedar este procedimiento.

Figura 3.2.6. Inserción de un “MenuStrip” para las funciones principales de la interfaz gráfica.

La figura 3.2.6, muestra la manera de agregar botones a la interfaz gráfica, de igual manera expresa que se pueden ir agregando de manera consecutivas más botones a la interfaz.

Si en cada botón del menú se da doble clic durante el diseño, se crea un *evento de clic*[7], este evento es una función que se ejecuta cuando se hace un clic en el objeto durante el tiempo de ejecución.

Dando doble clic, en el primer ítem *Abrir Imagen Nueva* se obtiene el siguiente resultado.

```
private void abrirImagenNuevaToolStripMenuItem_Click(object sender, EventArgs e)
{
//Agregar aquí las intrucciones.///
}
```

Cuando se da clic en el botón del menú se produce el evento y se ejecutan las instrucciones que están entre las llaves de la función.

Haciendo lo mismo en el botón *Salir*, ocurre lo siguiente:

```
private void salirToolStripMenuItem_Click(object sender, EventArgs e)
{
    this.Close();
}
```

Para esta evento se introduce una función llamada *this.Close().* Cuando *this.Close()* es llamado, se cierra por completo todo el programa que esta en ejecución.

3.3 Abrir una Imagen.

Para abrir una imagen es necesario:

- Aplicar alguna herramienta para explorar archivos.

Figura 3.3.1. Herramienta para Abrir Archivos en C#.

La figura 3.3.1 representa un objeto de la barra de herramientas de Visual C# 2008, que sirve para explorar y abrir archivos que se encuentran en las carpetas del explorador.

- Aplicar un filtro en la herramienta para visualizar solo imágenes BMP, JPG, GIF y PNG.

  Filtro OpenFileDialog = *.BMP, *.JPG, *.GIF, *.PNG.

- Asignar la imagen seleccionada a una variable del tipo Bitmap.

  Variable Bitmap NombreVariableIMG = new Bitmap();

[7] Un evento de clic en programación se produce cuando se da un clic sobre un objeto que tiene ese evento. Existen diversos eventos como pueden ser: Evento de doble clic, de mover el objeto, pasar por encima del objeto, etc.

- Verificar que se ha cargado alguna imagen.

```
Si (NombreVariableIMG < 100x100 Pixeles) //La imagen se considera pequeña.
        {
        NombreVariableIMG = null // variable sin valor.
        }

Si (NombreVariableIMG > 800x618 Pixeles) //La imagen se considera muy grande
        {
      //Reescalar imagen con un factor de conversión a 800X622 pixeles
        }

Si (NombreVariableIMG ≠ null)
     {
        //Se a cargado la imagen.
        }
```

- Convertir la imagen al plano RGB de24 bits.

```
Si (NombreVariableIMG.Formato ≠ Formato24BitsRGB)
 {
    NombreVariableIMG = ConvertirA24BitsRGB ( NombreVariableIMG.Formato )
 }
```

- Presentar imagen en un cuadro de dibujo considerando un ancho y alto de pixeles adecuado.

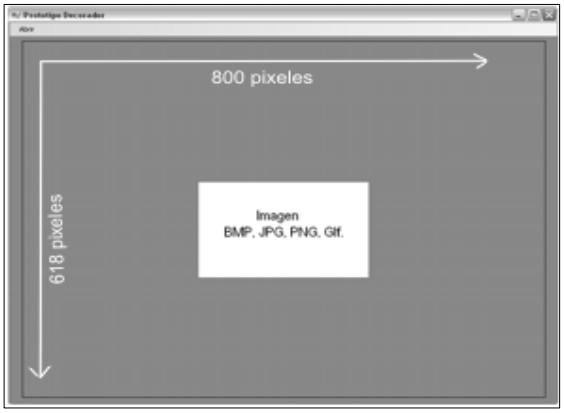


Figura 3.3.2. Área para visualizar la imagen seleccionada y extraer las características necesarias.

El la figura 3.3.2, se muestra las dimensiones máximas que puede tomar la imagen en pantalla. Para el ancho de la imagen se consideró como máximo 800 pixeles y para el alto de la imagen, un máximo de 618 pixeles.
Estos valores tienen su origen al considerar un estándar en medidas de charolas para repostería, tomando como referencia la de mayor ancho y largo. En el anexo B se puede observar la ficha de medidas de charolas.

La resolución de pantalla se obtuvo de la siguiente manera:

El valor de área más grande para charolas es 53x41cm, esto tiene un factor de 41cm/53cm = 0.7735, si se considera un área en pantalla de 800 pixeles para presentar el ancho de la imagen, el alto de la imagen se obtiene multiplicando el factor.

800 pixeles x 0.7735 = 618 pixeles.

Por lo tanto: Máximo Ancho = 800 pixeles.
Máximo Alto = 618 Pixeles.

Se considero en un inicio un ancho de 800 pixeles debido a que es una resolución de pantalla muy común, pues corresponde a las pantallas de 15 pulgadas, que junto con las de 17 pulgadas son las mas frecuentes de encontrar. [8]

En el apéndice A parte I se muestra el código en C# que genera la función para abrir una imagen y mostrarla en pantalla.

3.4. Filtros para imágenes digitales.

A continuación hay que crear la función para extraer los contornos de la imagen. Para esto se emplean los filtros de imagen. El problema en esta situación, es aplicar el filtro adecuado, pues en la actualidad existen diversos filtros para la detección de bordes, y cada uno muestra resultados diferentes. Algunos de los filtros más comunes son: filtro de Canny, filtro de Laplace, filtro de Sobel, filtro de Prewiit, filtro de Roberts.

Debido a que existen varios filtros y es necesario tomar en consideración solo uno, se debe realizar una prueba a cada uno de los más comunes y seleccionar el que proporcione mejores resultados.

Es conveniente generar una clase en C# que contenga los filtros para poder manejar la información de manera ordenada, para esto se crea y agrega una nueva clase al proyecto a la que se le ha llamado filtros.cs.

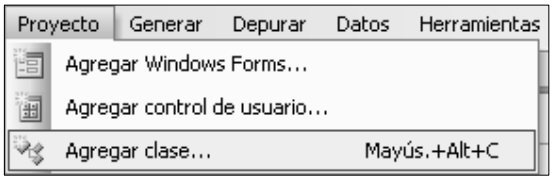


Figura 3.4.1. Herramienta para agregar una clase nueva en C#.

La figura 3.4.1, muestra como agregar una nueva clase al proyecto en el que se esta trabajando dentro del ambiente de programación de visual C# 2008.

Contando con una nueva clase, se pueden agregar métodos a la clase para ser usados conforme sean invocados en el programa, la clase que se ha creado en este punto corresponde a la de los filtros y en esta clase de pueden declarar los métodos que realizan las operaciones de filtrado.

[8] La información de resolución de pantalla más común es un dato extraído de la pagina web: http://www.webtaller.com/maletin/articulos/resolucion-pantalla.php

Para este proyecto se han considerado las siguientes funciones, que servirán para probar algunos de los filtros más comunes para la detección de bordes.

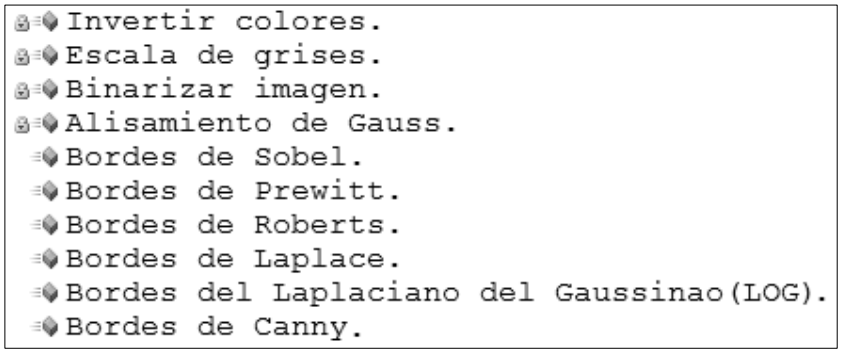


Figura 3.4.2. Funciones de filtros para hacer pruebas de contorno.

La figura 3.4.2 muestra las funciones que se han declarado en la clase “filtros.cs”, las funciones marcadas con un candado, son procedimientos auxiliares. Por ejemplo, en las operaciones de detección de contornos, los resultados producen un fondo negro con líneas blancas, con la función *“Invertir colores”* se cambia el fondo a blanco y las líneas a negro, que para el cálculos posteriores en esta aplicación es importante tener un fondo blanco y las líneas en negro.

Para empezar a probar los filtros se tomará como base la imagen de la figura 3.4.3, la cual tiene las características de una imagen usada frecuentemente en la decoración de pasteles.

Figura 3.4.3. Imagen de prueba para la decoración de un pastel.

La figura 3.4.3, representa una imagen típica en la decoración de pasteles, ya que no cuenta con efectos de luz, sombra o degradación de color, que para un decorador son efectos despreciados a la hora de copiar una imagen a un pastel. La figura 3.4.3, es la imagen base para hacer pruebas en los filtros citados en la figura 3.4.2.

3.4.1 Función invertir colores.

La inversion de un color de una imagen consiste en determinar el color opuesto en ese modelo de color. Es una operación que afecta a cada pixel, independiente de su vecindad. La imagen inversa puede ser imaginada como “la inversa” o como el “negativo” de una imagen.

Esta funcion es necesaria en el programa debido a que en un punto de la aplicación es necesario invertir los pixeles que resultaron de la convolución a su negativo o invertir alguna imagen que cuente con un fondo negro con líneas blancas.

En el caso del modelo RGB, se determina el valor inverso de un color, restando el valor del canal RGB al valor máximo de cada canal, que en este caso corresponde a 255. Con esto se propone la siguiente función.

$$\text{Dada la funcion } f(x,y) = \begin{vmatrix} 255 - p(r) \\ 255 - p(g) \\ 255 - p(b) \end{vmatrix}$$

Donde f(x,y) representa la imagen, p(r) representa al canal rojo de la imagen, p(g) el canal verde y p(b) el canal azul.

En el siguiente procedimiento se describe como aplicar esta función a una imagen.

Pseudocódigo.

```
Para cada pixel de Imagen f(x,y).
   {
   p(r) = 255 - p(r);    //invierte valor del color rojo
   p(g) = 255 - p(g);    //invierte valor del color verde
   p(b) = 255 - p(b);    //invierte valor del color azul
   }
```

Figura 3.4.4. Resultado de invertir los colores de una imagen.

La figura 3.4.4 muestra el resultado de aplicar la función *invertir colores* a la imagen de la figura 3.4.3.

El código fuente en C# para generar esta función, se encuentra en el Apéndice A parte II.

3.4.2 Función escala de grises.

Convertir una imagen a escala de grises, es importante, pues se logra que los canales de color RGB tengan el mismo valor, facilitando la segmentación de la imagen. Para generar la

función de escala de grises, se ha recurrido al estándar de televisión YIQ[9] que consiste en un sistema de nivelación de color, Y regula la cantida de luz en la imagen e IQ la cantidad de color.

Los valores que fueron extraidos de este estándar, pertenecen a los de conversión de RGB a Y, debido a que Y es el que determina la cantidad de luz que presenta el pixel en la pantalla, mostrando un brillo que va de 0 a 255. En tanto que los valores de conversion de IQ fueron despreciados, por que solo representan valores de conversion de color, que en este caso no son requeridos.

De acuerdo a la información del estándar YIQ: $Y = 0.299\,R + 0.587\,G + 0.114\,B$

Donde Y es el resultado de la conversion de los canales RGB, R es el valor del pixel en el canal rojo, G es el valor del pixel en el canal verde y B es el valor de pixel en el canal azul.

Con esto se puede formar la sigueinte función que muestra como obtener la imagen en escala de grises.

f(x,y) = R * 0.299 + G * 0.587 + B * 0.114.

Donde, f(x,y) representa el resultado de la imagen en escala de grises.

En el siguiente procedimientos describe como aplicar esta función a la imagen.

Pseudocódigo.

```
Para cada pixel de Imagen f(x,y)
    {
    valor_Gris = ( p(r)* 0.299 + p(g) * 0.587 + p(b) * 0.114);

    p(r) = p(g) = p(b) = valor_Gris ;
    }
```

En este algoritmo p(r) corresponde al valor del canal rojo, p(g) al verde, p(b) al canal azul y valor_gris al tono en gris que se obtiene al realizar la suma de los 3 canales de color.

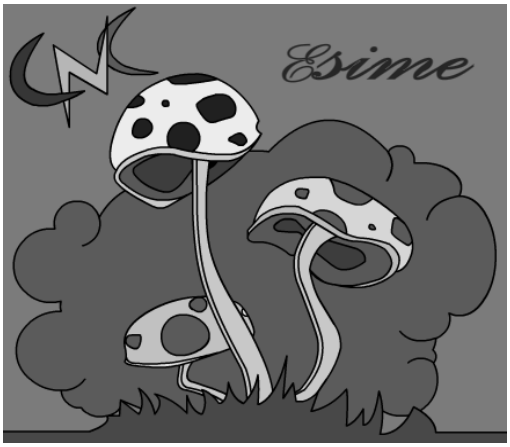


Figura 3.4.5. Resultado de convertir la imagen a escala de grises.

[9] El modelo de Color YIQ - Tratamiento Digital de Imágenes, Rafael C. González – Richard E. Woods, Pag 248.

La figura 3.4.5 es el resultado de aplicar la función “escala de grises” a la imagen de la figura 3.4.3.

El código fuente en C# para generar esta función, se encuentra en el Apéndice A parte III.

3.4.3 Función binarizar imagen.

La función “binarizar imagen” sirve para ver si un pixel esta dentro de un valor de umbral o si esta fuera, dando como resultado solo dos valores en una imagen, con lo que se obtiene por medio de esta función, una imagen monocromática o binaria.

La siguiente funcion muestra la operación que se realiza para obtener la imagen binaria.

Dada la función $s(x,y) = \begin{vmatrix} 0, & \text{si } f(x,y) < \text{umbral} \\ 255, & \text{si } f(x,y) \geq \text{umbral} \end{vmatrix}$

Donde s(x,y) es una imagen binaria y umbral es el nivel de separación para considerar blanco o negro a un pixel de la imagen.

En el siguiente procedimiento se describe como aplicar esta función a una imagen.

Pseudocódigo.

```
Para cada pixel de Imagen f(x,y).
{
     Si(f(x,y) < umbral)    //Si los valores RGB estan por  debajo
     f(x,y) = 0;            //del umbral todos los valores RGB seran 0

     Sino
     f(x,y) = 255;        //Por ensima del valor de umbral seran 255.
}
```

Figura 3.4.6a. Función binarizar imagen con umbral = 90.

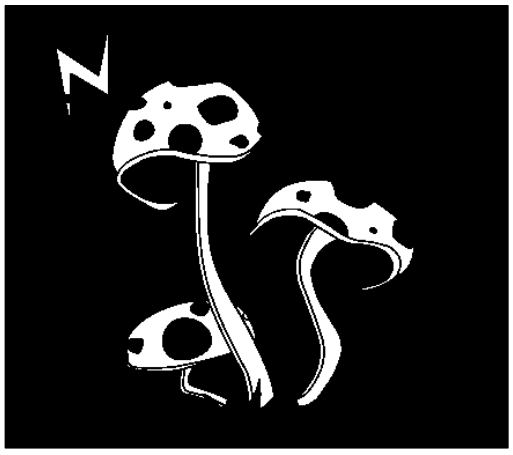


Figura 3.4.6b. Función binarizar imagen con umbral = 128.

La figura 3.4.6a y 3.4.6b muestran el resultado de aplicar la función *binarizar imagen* a la imagen de la figura 3.4.5, que esta en escala de grises. Para el caso de la figura 3.4.6a se ha usado un valor de umbral igual a 90, esto significa que los pixeles que estén por debajo de ese valor adquieren el valor mínimo de intensidad que es 0 y los pixeles que están por encima del valor de 90 adquieren el valor máximo de intensidad que es 255. Para la figura 3.4.6b se ha usando un valor de umbral de 128, debido a esto se presentan más pixeles negros que el resultado de la figura 3.4.6a, pues el umbral de pixeles negros se abrió de 90 a 128 convirtiendo más pixeles que no entraban en el rango cuando solo llega a 90 de umbral.

El código fuente en C# para generar esta función, se encuentra en el Apéndice A parte IV.

3.4.4 Máscaras de convolución.

Una máscara de convolución es una representación matricial de coeficientes que en procesamiento de imágenes sirven para suavizado de imágenes, el afilado de imágenes, detección de bordes, y otros efectos.

Si se considera una imagen como una función matricial f(x, y) y a una matriz de convolución como g(x, y), se puede expresar la operación de convolución[10] como lo muestra la ecuación 3.4.1.

$$s(x, y) = f(x, y) * g(x, y) \qquad \textbf{Ecuación 3.4.1}$$

Donde s(x, y) representa el resultado de la convolución entre f(x, y) y g(x, y).

Como las funciones f(x, y) representa la matriz de una imagen y g(x, y) la matriz de los coeficientes de un filtro, su convolución puede ser expresada como lo muestra la figura 4.3.7.

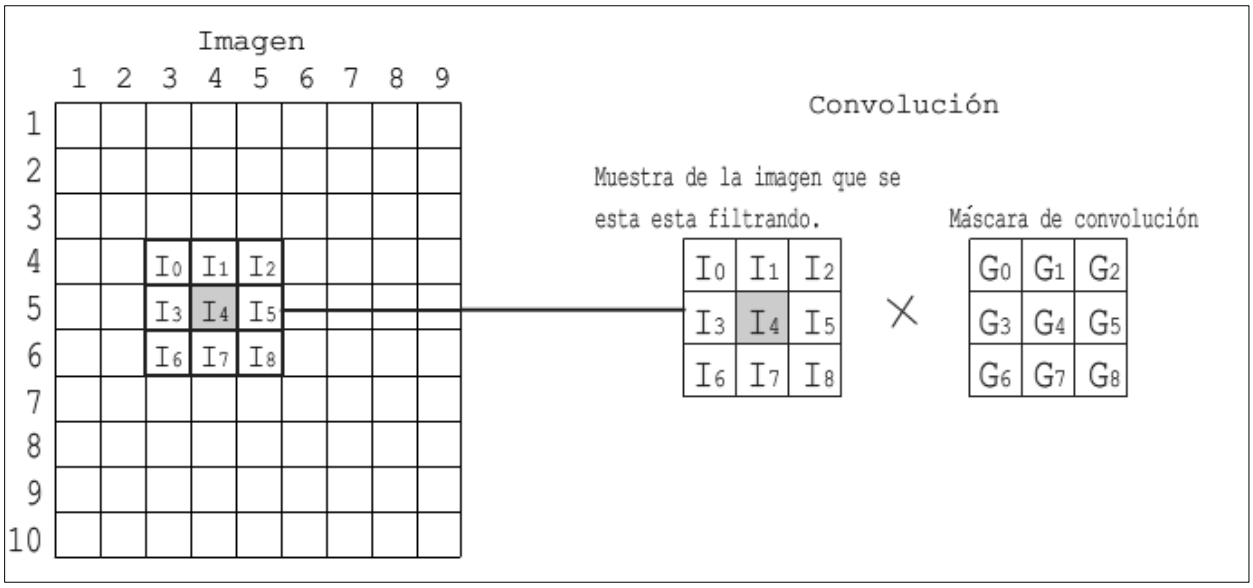


Figura 4.3.7. Convolución con una muestra de una imagen y una máscara de convolución.

[10] Una convolucíon es una operación matemática que transforma dos funciones f(x,y) y g(x,y) en una tercera función s(x,y), que representa la magnitud en la que se superpone f(x,y) con la función translada e invertida g(x,y).

La figura 3.4.7, muestra la convolución de un pixel de una imagen con una máscara de convolución, donde I_4 representa al pixel que será transformado por la convolución, I_0, I_1, I_2, I_3, I_5, I_6, I_7, I_8, son los pixeles vecinos que sirven como referencia para determinar el valor de I_4 cuando entran en operación con los coeficientes de la máscara de convolución (G_0 a G_8). I_0 a I_8 representa una muestra de 3x3 de la matriz imagen f(x, y).

El resultado de la figura 3.4.7 queda expresado como se muestra en la figura 3.4.8.

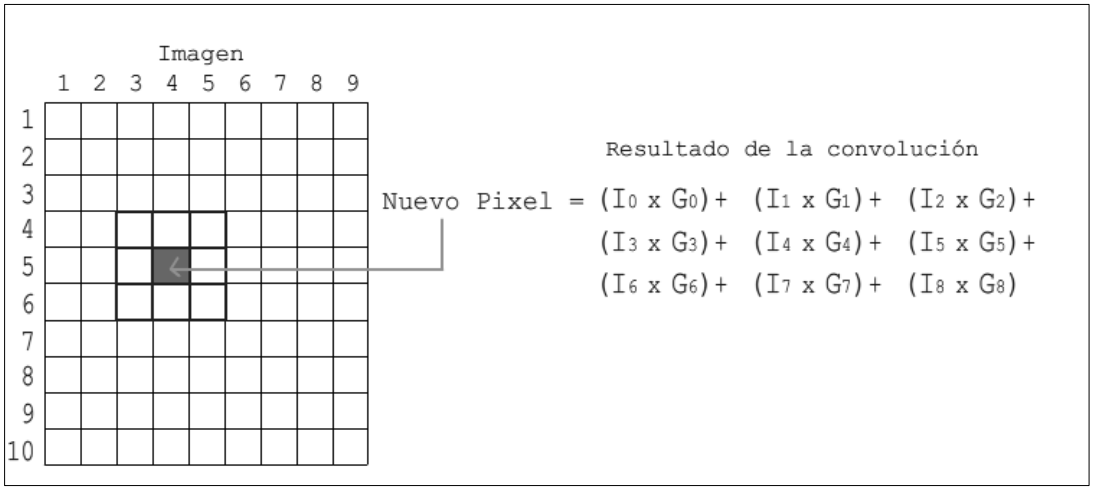


Figura 3.4.8. Resultado de realizar una operación de convolución entre matrices.

La figura 3.4.8, muestra la operación que resulta de convolucionar los valores de los coeficientes de la máscara de convolución con una muestra de 3x3 pixeles de una imagen, donde se obtiene un pixel con valor filtrado. Si se realiza un recorrido de toda la imagen con la misma máscara de convolución se obtiene los pixeles filtrados que forman la nueva imagen con el efecto del filtro utilizado.

3.4.5 Función alisamiento de Gauss.

La función alisamiento de Gauss, consiste en un filtro que representa la máscara de convolución de Gauss para la eliminación del ruido en una imagen, su efecto de eliminación de ruido se debe a un desenfoque sobre la imagen. Esta función es importante en la aplicación, ya que los filtros de Canny y Laplaciano del Gaussiano hacen uso de este filtro, además puede ser usada simplemente para eliminar pixeles indeseados en una imagen.

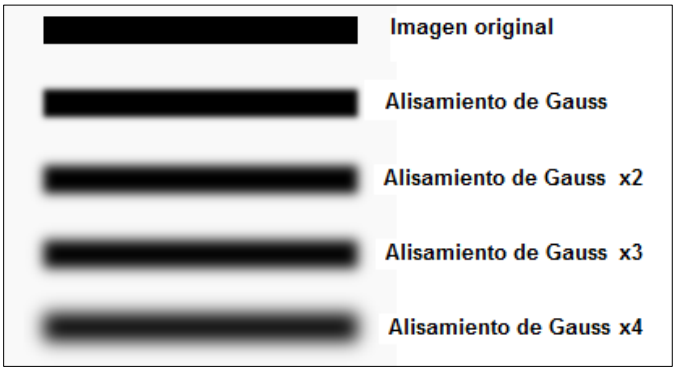


Figura 3.4.9. Resultado de aplicar de forma gradual alisamiento de Gauss a una imagen.

La figura 3.4.9 muestra un rectángulo negro como imagen inicial, al aplicar el filtro de Gauss, la imagen se ve desenfocada o suavizada, si se vuelve a aplicar el filtro sobre a imagen suavizada, esta se suavizara más, teniendo como resultado, el alisamiento que presenta el rectángulo con *alisamiento de Gauss x2*, de tal manera que si aplicamos 4 veces el alisamiento se tendrá una imagen como la muestra el ultimo rectángulo de la figura 3.4.9, en el que se observa mayor suavizado que los demás.

Este efecto de alisamiento es posible lograrlo convolucionando la imagen con una máscara del filtro de Gauss[11].

2	4	5	4	2
4	9	12	9	4
5	12	15	12	5
4	9	12	9	4
2	4	5	4	2

1/115

Figura 3.4.10. Máscara de convolución para obtener el filtro de alisamiento de Gauss.

La figura 3.4.10 representa la máscara de convolución de Gauss para alisar una imagen, esta máscara consiste en una matriz de 5 x 5 y un divisor a la derecha que corresponde a un factor de alisamiento. La operación con la máscara se realiza recorriendo toda la matriz de la imagen, multiplicando los valores de los pixeles de la imagen y promediando la suma de estos resultados con el factor de alisamiento (como se explico en el subtema 3.4.4 *máscaras de convolución*). La forma de representar esta función es la que se muestra a continuación.

Dada la función de convolución GAUSS(x,y) = f(x, y) $*$ h(x, y).

Donde GAUSS(x, y) es la imagen suavizada, resultado de la convolución entre f(x, y) y h(x, y), siendo f(x, y) la imagen original y h(x, y) la máscara de convolución de Gauss.

En el siguiente procedimientos describe como aplicar esta máscara a una imagen.

Pseudocódigo.

```
        Para cada pixel de Imagen f(x, y).
{

GAUSS(x, y) =
   [ f(x-2, y-2) * 2 + f(x-1, y-2) * 4 + f(x, y-2) * 5 + f(x+1, y-2) * 4 + f(x+2, y-2) * 2
     f(x-2, y-1) * 4 + f(x-1, y-1) * 9 + f(x, y-1) *12 + f(x+1, y-1) * 9 + f(x+2, y-1) * 4
     f(x-2,  y ) * 5 + f(x-1,  y ) *12 + f(x,  y ) *15 + f(x+1,  y ) *12 + f(x+2,  y ) * 5
     f(x-2, y+1) * 4 + f(x-1, y+1) * 9 + f(x, y+1) *12 + f(x+1, y+1) * 9 + f(x+2, y+1) * 4
     f(x-2, y+2) * 2 + f(x-1, y+2) * 4 + f(x, y+2) * 5 + f(x+1, y+2) * 4 + f(x+2, y+2) * 2 ]
   x 1/115

}
f(x, y) = GAUSS(x, y);    // La imagen original adquiere el
                          // valor de la imagen alisada.
```

[11] La máscarade convolución del filtro de Gauss fue obtenida del libro Hill Grenn, Canny Edge Detection tutorial 2002.

De este algoritmo, hay que mencionar que los números enteros son constantes que pertenecen a los establecidos en la máscara de la figura 3.4.10 con su respectivo factor de alisamiento a la derecha, los otros elementos corresponde a la función f(x, y) y sus respectivos desplazamientos en la matriz.

Figura 3.4.11a. Función Alisamiento de Gauss aplicado 1 vez.

Figura 3.4.11b. Función Alisamiento de Gauss aplicado 3 veces.

La figura 3.4.11a, es el resultado que se obtiene al aplicar la máscara de convolución de Gauss a la imagen de la figura 3.4.3, esta imagen presenta un ligero alisamiento, poco perceptible para la vista, pero para demostrar su efecto, se aplicó la máscara de convolución 3 veces a la imagen que resultaba de cada alisamiento, obteniendo la imagen que se muestra en la figura 3.4.11b, en la que se observa un desenfoque más pronunciado.

El código fuente en C# para generar esta función, se encuentra en el Apéndice A parte V.

3.4.6 Función bordes de Sobel.

Esta función es un filtro de detección de bordes por medio del cálculo de gradiente.

Este método obtiene los bordes en las direcciones (x, y) de una imagen, la magnitud del gradiente y la dirección del vector gradiente.

La ecuación 3.4.2 expresa la forma de obtener el vector gradiente.

$$\nabla f(x,y) = i\frac{df(x,y)}{dx} + j\frac{df(x,y)}{dy} \qquad \textbf{Ecuación 3.4.2}$$

Donde *i, j,* son los vectores unitarios en las direcciones X e Y.

De la ecuación anterior se obtienen los operadores de Sobel (Gx, Gy)[12], para detectar los bordes en la dirección horizontal X y vertical Y, los cuales se muestran en la figura 3.4.12.

[12] Operador de Sobel. Feature extraction and imagen processing. Mark Nixon & Alberto Aguado. Pag. 124

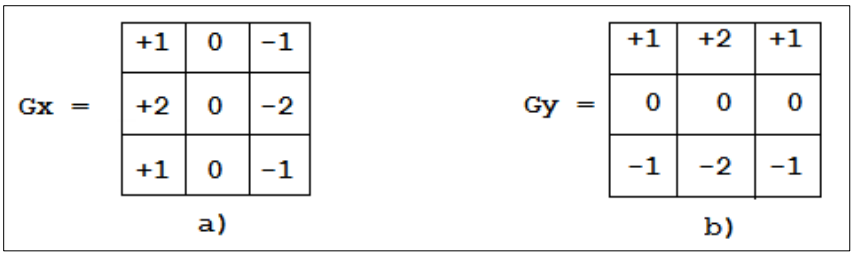


Figura 3.4.12. Máscaras de convolución de Sobel para la detección de bordes en la dirección horizontal (Gx) y vertical (Gy).

En la figura 3.4.12, se muestran dos máscaras de convolución, la máscara del inciso a) pertenece al operador Gx de Sobel que sirve para encontrar los bordes en la dirección horizontal de la imagen, en el inciso b) se expresa el operador Gy, que consiste en una máscara de convolución, que detecta los borde en la dirección vertical de la imagen.

Los operadores Gx y Gy sirven para encontrar la magnitud del gradiente G, el cual determina los bordes de Sobel para este caso. La ecuación 3.4.3 indica como calcular la magnitud del gradiente G.

$$G = \sqrt{Gx^2 + Gy^2} \qquad \textbf{Ecuación 3.4.3}$$

Con estos operadores también se obtiene el ángulo de dirección del gradiente G, tal y como lo muestra la ecuación 3.4.4.

$$\theta_G = \text{ArcTang}\left(\frac{Gy}{Gx}\right) \qquad \textbf{Ecuación 3.4.4}$$

Pseudocódigo.

```
Para cada pixel de Imagen f(x,y).
{

  Gx = f(x-1, y-1)*1 + f(x-1, y)*2 + f(x-1, y+1)*1
      -f(x+1, y-1)*1 - f(x+1, y)*2 - f(x+1, y+1)*1 //horizontal

  Gy = f(x-1, y-1)*1 + f(x, y-1)*2 + f(x+1, y-1)*1
     -f(x-1, y+1)*1 - f(x, y+1)*2 - f(x+1, y+1)*1 //vertical

  G = (Gx^2 + Gy^2 )^1/2  //Magnitud del vector Gradiente.

  θG = ArcTan(Gy/Gx) //Direccion del vector Gradiente.

}

  f(x,y) = G(x,y)
```

El primer resultado de esta función son los bordes de la imagen original en la dirección del gradiente Gx y Gy.

Figura 3.4.13a. Bordes del gradiente horizontal (Gx) de Sobel.

Figura 3.4.13b. Bordes del gradiente vertical (Gy) de Sobel.

La figura 3.4.13a y 3.4.13b muestran el resultado de aplicar la máscara de convolución de gradiente en Gx, Gy, además de aplicar la función Invertir al resultado de cada gradiente para cambiar el color de fondo.

Después de haber obtenido el valor de los gradientes Gx y Gy, se procede a calcular la magnitud del gradiente G, para esto se aplica la ecuación 3.4.3.

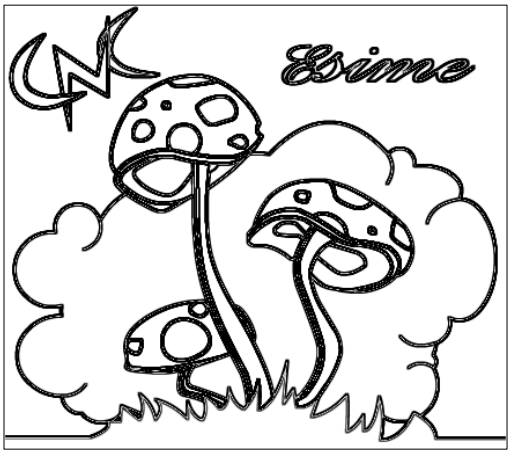


Figura 3.4.14. Resultado del filtro de *Sobel.*

La figura 3.4.14 muestra el resultado obtenido de la magnitud del gradiente G con la ecuación 3.4.3, que consiste finalmente en la detección de bordes de Sobel. Adicionalmente al resultado de la detección de Sobel, se le aplicó la función *invertir colores* y la función *binarizar imagen*, para obtener un fondo blanco con líneas de un solo tono de color.

3.4.7 Función bordes de Prewitt.

Es un filtro de detección de bordes por medio del cálculo de gradiente al igual que Sobel, la diferencia radica en su máscara de convolución, en la cual los operadores Gx y Gy de Prewitt[13] presentan las máscaras de convolución que muestra la figura 3.4.15.

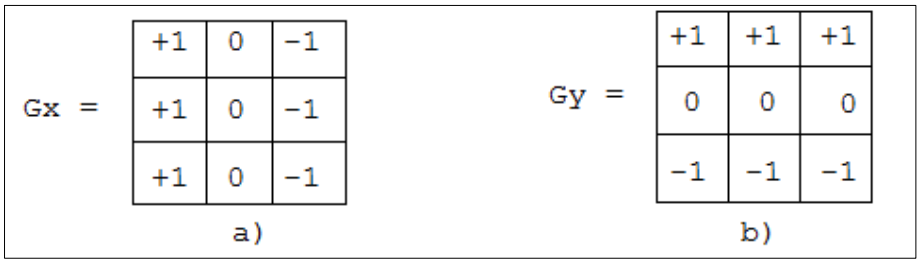


Figura 3.4.15. Máscaras de convolución de Prewitt para la detección de bordes en la dirección horizontal Gx y vertical Gy.

En la figura 3.4.15, se muestran dos máscaras de convolución, la máscara del inciso a) pertenece al operador Gx de prewitt, que sirve para encontrar los bordes de la imagen en la dirección horizontal, en el inciso b) se expresa el operador Gy, que consiste en una máscara de convolución que detecta los borde en la dirección vertical.

Después, los resultados de las aproximaciones de los gradientes horizontales y verticales (Gx, GY), son usados para obtener la magnitud del gradiente de la ecuación 3.4.3 y el ángulo de dirección del gradiente mediante la ecuación 3.4.4, vistas en la función de Sobel.

Pseudocódigo.

```
Algoritmo

    Para cada pixel de Imagen f(x,y).
    {

      Gx = f(x-1, y-1)*1 + f(x-1, y)*1 + f(x-1, y+1)*1
          -f(x+1, y-1)*1 - f(x+1, y)*1 - f(x+1, y+1)*1 //horizontal

      Gy = f(x-1, y-1)*1 + f(x, y-1)*1 + f(x+1, y-1)*1
          -f(x-1, y+1)*1 - f(x, y+1)*1 - f(x+1, y+1)*1 //vertical

      G = (Gx^2 + Gy^2 )^1/2   //Magnitud del vector Gradiente.

      θG = ArcTan(Gy/Gx) //Direccion del vector Gradiente.
    }

      f(x,y) = G(x,y)
```

[13] Operador de Prewitt. Feature extraction and imagen processing. Mark Nixon & Alberto Aguado. Pag. 122

Figura 3.4.16a. Bordes del gradiente horizontal (Gx) de Prewitt.

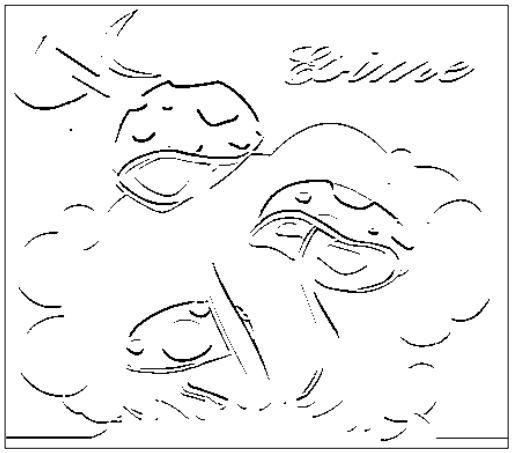


Figura 3.4.16b. Bordes del gradiente vertical (Gy) de Prewitt.

La figura 3.4.16a y 3.4.16b muestran el resultado de aplicar el gradiente en X, Y de Prewitt, además de aplicar la función Invertir al resultado de cada gradiente para cambiar el color de fondo. En estos resultados se puede observar que hay similitud con los obtenidos con los gradientes propuestos por Sobel, pero en el método de Prewitt la cantidad de pixeles pertenecientes a bordes son menores que los que se obtienen con Sobel.

Después de haber obtenido el valor de estos pixeles se procede a calcular la magnitud de la imagen, para esto se aplica la ecuación 3.4.3.

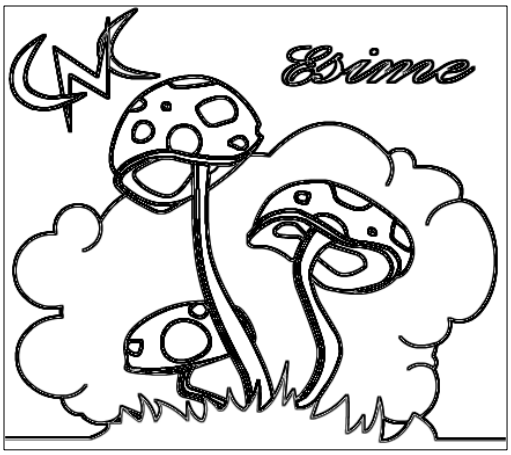


Figura 3.4.17. Resultado del filtro de *Prewitt.*

La figura 3.4.17 muestra el resultado que se obtiene de calcular la magnitud del gradiente G utilizando la ecuación 3.4.3 con los parámetros Gx y Gy obtenidos con los operadores de Prewitt, adicionalmente al resultado de la detección de Prewitt, se le aplicó la función *invertir colores* y la función *binarizar imagen*, para obtener un fondo blanco con líneas de color negro.

Los resultados obtenidos en los filtros de Sobel y Prewitt tienen tanta similitud que hace difícil ver la diferencia, pero usando un contador de pixeles negros en las imágenes resultantes se determina la diferencia, en la imagen resultante de Sobel se obtuvieron 17017 pixeles negros y en la imagen resultante de Prewitt se obtuvieron 14751 pixeles negros.

3.4.8 Función bordes de Roberts.

El operador gradiente de Roberts[14] es el método no lineal más simple utilizado para la detección de bordes. Presenta la desventaja que, dependiendo de la dirección, ciertos bordes son más realzados que otros, inclusive teniendo igual magnitud. Al igual que Sobel y Prewitt, este detector ocupa 2 máscaras de convolución para encontrar los bordes del gradiente G en la dirección horizontal (Gx) y vertical (Gy).

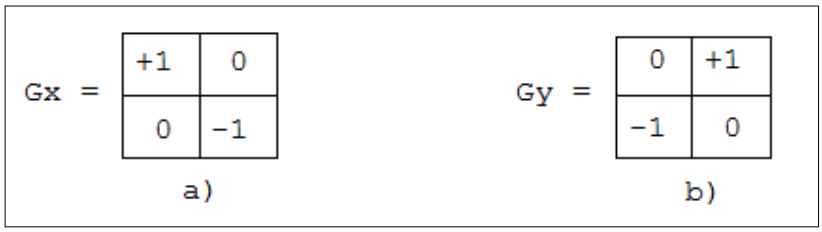


Figura 3.4.18. Máscaras de convolución de Roberts para la detección de bordes en la dirección horizontal (Gx) y vertical (Gy).

La figura 3.4.18, muestras dos máscaras de convolución, con la máscara del inciso a) se obtiene el operador Gx y con la máscara del inciso b) el operador Gy, sabiendo que estos elementos sirven para encontrar la magnitud del gradiente G usando la ecuación 3.4.3 y el ángulo de dirección del gradiente G con la ecuación 3.4.4.

El siguiente procedimiento muestra como calcular los bordes de la imagen usando los operadores de Roberts.

Pseudocódigo.

```
Para cada pixel de Imagen f(x,y).
{

  Gx = f(x , y)*1 + f(x+1, y+1)*-1;   //horizontal

  Gy = f(x+1, y)*1 + f(x, y-1)*-1;    //vertical

  G = (Gx^2 + Gy^2 )^1/2   //Magnitud del vector Gradiente.

  θG = ArcTan(Gy/Gx) //Dirección del vector Gradiente.
}

  f(x,y) = G(x,y)
```

Calculando los operadores Gx y Gy que propone Roberts se obtienen los resultados que muestra la figura 3.4.19a y 3.4.19b.

[14] Filtro de Roberts. Feature extraction and imagen processing. Mark Nixon & Alberto Aguado. Pag. 104.

Figura 3.4.19a. Bordes del gradiente horizontal (Gx) de Roberts.

Figura 3.4.19b. Bordes del gradiente vertical (Gy) de Roberts.

La figura 3.4.19a y 3.4.19b, son las componentes del vector gradiente de Roberts, que son utilizadas para calcular la magnitud de este vector.

Figura 3.4.20. Resultado del filtro de Roberts

La imagen de la figura 3.4.20 se obtuvo realizando la operación de la ecuación 3.4.3 con los operadores de Roberts obtenidos.

3.4.9 Función bordes de Canny.

El filtro de Canny[15] para detectar bordes, consta de 4 pasos.

1.- Aplicar un filtro de alisamiento (Gauss) a la imagen que se va trabajar.

2.- Obtener la magnitud del vector gradiente de la imagen usando algún filtro de gradiente (Sobel, Prewiit, Roberts).

[15] Filtro de Canny. Feature extraction and imagen processing. Mark Nixon & Alberto Aguado. Pag. 129.

3.- Aplicar la supresión de no máximos al resultado: En este paso se logra el adelgazamiento del grosor de los bordes obtenidos con el gradiente, hasta lograr bordes de un píxel de grosor.

4.- Aplicar histéresis de umbral: en este paso se aplica una función de histéresis basada en dos umbrales, es parecido al proceso de binarización, pero en este caso se establecen dos umbrales a la función. Con este proceso se pretende reducir la posibilidad de aparición de contornos falsos.

Desarrollo de los pasos propuestos por Canny.

1.- Alisamiento de la imagen.

Usar la función *alisamiento de Gauss*.
f(x, y) = GAUSS(x, y). Donde f(x, y) representa la imagen original y s(x, y) es la imagen alisada.

2.- Obtención del gradiente y ángulo de dirección

Usar la función *bordes de Sobel.*
f(x, y) = G(x, y). Donde f(x, y) representa la imagen alisada y G(x, y) el resultado de la imagen filtrada con Sobel.

$\theta_G(x, y) = Gx / Gy$. Donde $\theta_G(x, y)$ es el ángulo del dirección del gradiente y Gx, Gy son los operadores del vector gradiente.

3.- Supresión de no máximos.

Para realizar la supresión de no máximos, se compara el ángulo de dirección $\theta_G(x, y)$ con 4 valores de orientación respecto al eje horizontal. Los 4 valores son:

d1 = 0º
d2 = 45º
d3 = 90º
d4 = 135º
La comparación se realiza de la siguiente manera:

Si ($\theta_G(x, y) < 22.5°$)
$\theta_G(x, y) = 0°$;
Sino, si ($\theta_G(x, y) < 67.5°$)
$\theta_G(x, y) = 45°$;
Sino, si ($\theta_G(x, y) < 112.5°$)
$\theta_G(x, y) = 90°$;
Sino, si ($\theta_G(x, y) < 157.5°$)
$\theta_G(x, y) = 135°$;
Sino ($\theta_G(x, y) = 0°$;

Con esto se asigna un valor discreto a el ángulo $\theta_G(x, y)$ para determinar en que dirección tiene vecindad con otro pixel adyacente y si debe ser suprimido el pixel o no.

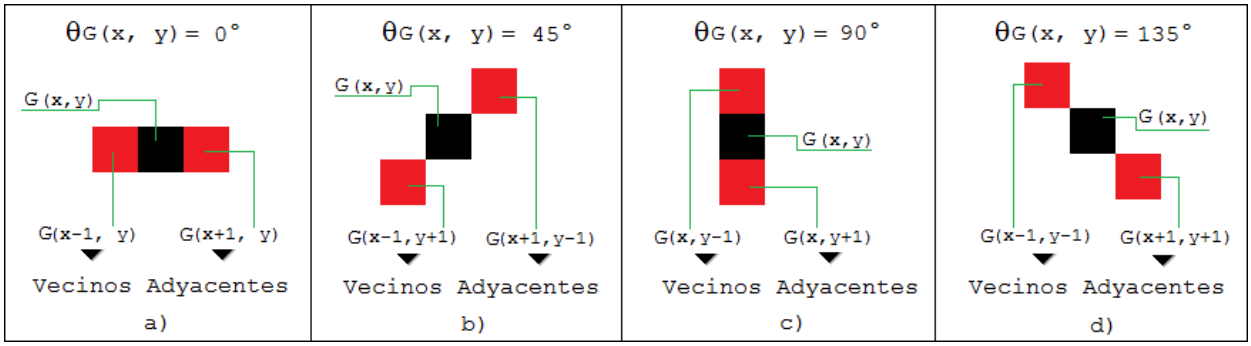


Figura 3.4.21. Pixeles adyacentes a la dirección del vector G(x,y).

La figura 3.4.21, muestra la posición que tiene un pixel cuando es adyacente a su angulo de dirección. El inciso a) de la figura 3.4.21 muestra la posición de los pixeles adyacentes cuando el ángulo de dirección del vector gradiente es 0°, el inciso b) muestra la posición de los pixeles adyacentes cuando el ángulo es de 45°, el inciso c) muestra la posición de los pixeles adyacentes cuando el ángulo de dirección es de 90° y el inciso d) muestra la posición de los pixeles cuando el ángulo de dirección es de 135°.

Tomando en cuenta lo anterior, para determinar si un pixel debe ser suprimido o no se realiza la siguiente comparación.

Si ($G_{(x, y)}$ < que al menos uno de sus dos vecinos adyacentes en la dirección $\theta_G(x, y)$)

$G_{(x, y)}$ = 0. Significa que a sido suprimido.

Si no se cumple la condición $G_{(x, y)}$ mantiene su valor.

Terminando de recorrer la matriz de $G_{(x, y)}$, se asigna el valor de la supresión de no máximos a la imagen f(x, y).

4.- Histéresis de umbral.

La histéresis de umbral es el último paso que propone Canny, sirve para eliminar algunos pixeles que se consideren falsos en la detección de contornos.

Para realizar este paso se proponen 2 valores de umbral en un rango de 0 a 255. El de menor valor será el umbral1 y el de mayor valor será umbral2 como se indica a continuación.

0 < umbral1 < umbral2 < 255.

Después de asignar los 2 valores de umbral, se recorre la matriz de los pixeles que resultaron de la supresión de no máximos, haciendo la siguiente comparación a cada pixel.

Si($f_{(x, y)}$ < umbral2) , si se cumple la condición se compara:

Si($f_{(x, y)}$ < umbral1) , si se cumple la condición $f_{(x, y)}$ se hace 0.

Si no , si no se cumplió la anterior condición se vuelve a comparar:

Si($f_{(x+1, y)}$ < umbral2 y $f_{(x+1, y+1)}$ < umbral2 y
$f_{(x, y+1)}$ < umbral2 y $f_{(x-1, y+1)}$ < umbral2 y
$f_{(x-1, y)}$ < umbral2 y $f_{(x-1, y-1)}$ < umbral2 y
$f_{(x, y-1)}$ < umbral2 y $f_{(x+1, y-1)}$ < umbral2)
Si se cumple esta condición el valor del pixel en $f_{(x, y)}$ se hace 0 por que todos los pixeles vecinos de este pixel son menores que el umbral2.

Aplicando estos pasos a la imagen de la figura 3.4.4, se obtienen las imágenes filtradas por Canny que se muestran en las figuras 3.4.22a y 3.4.22b.

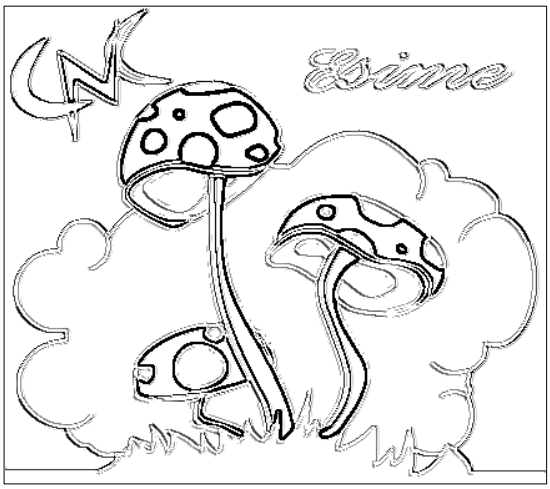


Figura 3.4.22a. Bordes del Canny con umbral1 = 20 y umbral2 = 200.

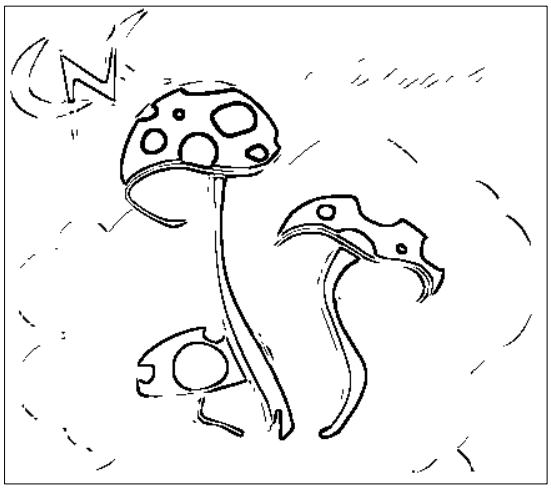


Figura 3.4.22b. Bordes del Canny con umbral1 = 200 y umbral2 = 255.

La figura 3.4.22a y 3.4.22b, muestran el resultado de aplicar el filtro de Canny, en las que se ha procedido usando el gradiente de Sobel. Para la figura 3.4.22a se aplicó un umbral menor= 20 y un umbral mayor = 200, mientras que en la figura 3.4.22b se aplicó un umbral menor = 200 y un umbral mayor = 255.

3.4.10 Función bordes de Laplace.

El filtro de Laplace[16], es un filtro de segunda derivada que remueve los pixeles resaltando a los que presentan cambios bruscos con respecto a sus vecinos. La expresión matemática para este filtro esta descrita en la ecuación 3.4.5.

$$\nabla^2 f_{(x,y)} = (d^2f/dx^2, d^2f/dy^2) \qquad \textbf{Ecuación 3.4.5}$$

[16] Filtro de Laplace. Feature extraction and imagen processing. Mark Nixon & Alberto Aguado. Pag. 137.

Los filtros de segunda derivada, basados en la ecuación 3.4.5 presentan la desventaja de ser sensibles al ruido que hay en algunas imágenes.
Para la detección de bordes de una imagen, Laplace propone 2 máscaras de convolución, cada una presenta diferentes resultados, por lo que es necesario determinar cual aporta mejores resultados.

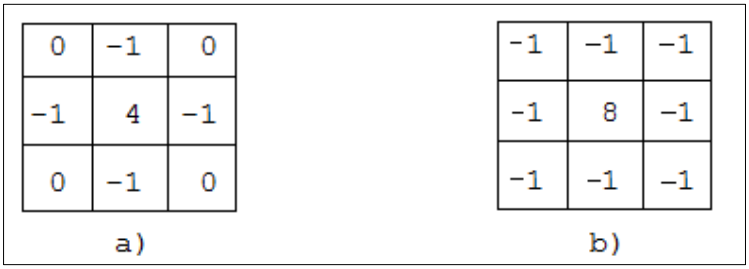


Figura 3.4.23. Máscaras de convolución de Laplace.

La figura 3.4.23. muestra las máscaras de convolución propuestas por Laplace para la detección de bordes. El inciso a) de la figura 3.4.23, muestra la máscara de Laplace que aparece comunmente en los textos que describen este filtro, mientras que en el inciso b) se muestra una máscara no tan comun de encontrar en los textos sobre filtros de Laplace, pero que debe ser considerada para observar los resultados que proporciona.

El siguiente procedimiento indica como desarrollar el código para generar la función de deteccion de bordes de laplace.

Pseudocódigo.

```
//Para la máscara de convolución del inciso a)

     Para cada pixel de Imagen f(x,y)
     {
      lp(x,y) = f(x, y)*4 - (f(x, y-1) + f(x-1, y)
                          + f(x+1, y) + f(x, y+1))
     }


//Para la máscara de convolución del inciso b)

     Para cada pixel de Imagen f(x,y).
     {
      lp(x,y) = f(x, y)* 8 - (f(x-1, y-1) + f(x, y-1) + f(x+1, y-1)
                           + f(x-1, y ) +  f(x+1, y ) + f(x-1, y+1)
                          + f(x, y+1) + f(x+1, y+1))
      }
```

Realizando esto pasos se obtienen las imágenes que se muestran en la figura 3.4.24a y 3.4.24b.

Figura 3.4.24a. 1er Filtro de Laplace

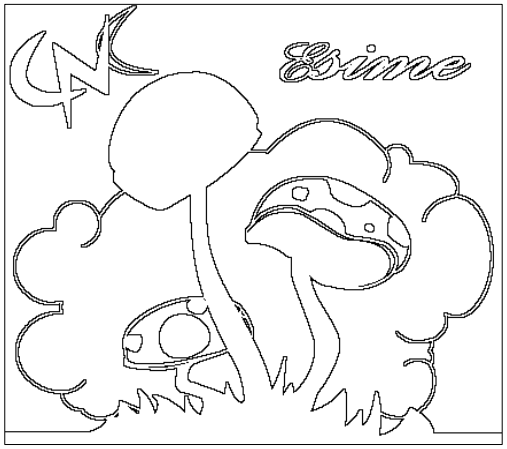


Figura 3.4.24a. 2do Filtro de Laplace

La figura 3.4.24a, muestra el resultado de aplicar la máscara de convolución de la figura 3.4.23 inciso a) y la figura 3.4.24b muestra el resultado de aplicar la máscara de la figura 3.4.23 inciso b), en ambos casos se aplico al resultado la función *invertir colores.* A simple vista las dos imágenes pueden parecer iguales, pero para el caso de la figura 3.4.24a se obtuvo una imagen con bordes de un pixel de grosor, mientras que en la figura 3.4.24b se obtuvo una imagen con algunos bordes más gruesos. Haciendo un acercamiento a las imágenes se puede observar esta diferencia.

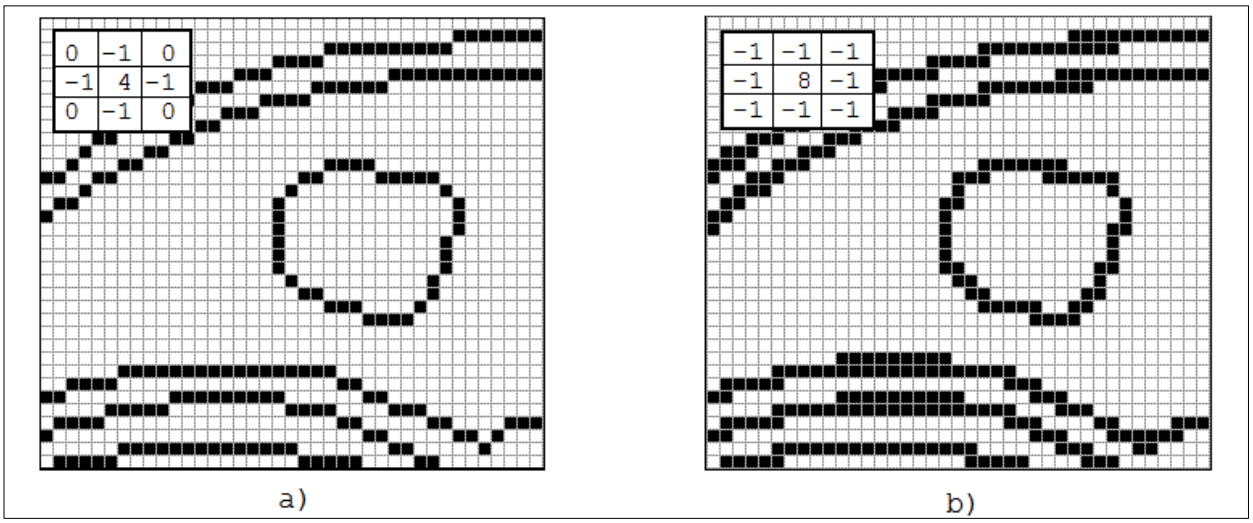


Figura 3.4.25. Acercamiento para observar los pixeles resultantes en la detección de bordes de Laplace.

El inciso a) de la figura 3.4.25, representa un fragmento amplificado en tamaño, de la imagen que resulta de aplicar el 1er filtro de Laplece (mostrado en la figura 3.4.23 inciso a)), el cual muestra como resultado, bordes delgados de un pixel de grosor, mientras que para el inciso b) de la figura 3.4.25, se muestra un fragmento amplificado en tamaño, de la imagen que resulta de aplicar el 2do filtro de Laplace (mostrado en la figura 3.4.23 inciso b)), en el cual se pueden observar algunos bordes con más de un pixel de grosor.

El código fuente en C# para generar esta función, se encuentra en el Apéndice A parte VI.

3.4.11 Función bordes Laplaciano del Gaussiano.

El filtro Laplaciano del Gaussiano, basa la detección de bordes reduciendo el ruido de la imagen por medio del alisamiento de la imagen y calculando a la vez, los cambios críticos de color haciendo uso de la segunda derivada. Este operador combina el filtro Gaussiano con el filtro de Laplace para lograr este proceso y se conoce como filtro LOG[17].

La expresión matemática del operador LOG es la que se muestra en la ecuación 3.4.6.

$$\nabla^2 f(x,y) * GAUSS = (d^2f/dx^2, d^2f/dy^2) * GAUSS \qquad \textbf{Ecuación 3.4.6}$$

Y su máscara de convolución esta representada en la figura 3.4.26.

0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

Figura 3.4.26. Máscaras de convolución del Laplaciano del Gaussinao (LOG).

La figura 3.4.26. representa la máscara de convolución propuesta por el filtro Laplaciano del Gaussinao (LOG).

Para desarrollar este filtro se ha propuesto el siguiente procedimiento.

Pseudocódigo.

```
   Para cada pixel de Imagen f(x,y).
    {
     //Usar la matriz 5x5 creada en alisamiento de Gauss y cambiar
     //los coeficientes por los establecidos en el operador LOG.

LOG(x,y) =
    f(x-2, y-2) *  0 + f(x-1, y-2) *  0 + f(x, y-2) * -1 + f(x+1, y-2) *  0 + f(x+2, y-2) *  0
    f(x-2, y-1) *  0 + f(x-1, y-1) * -1 + f(x, y-1) * -2 + f(x+1, y-1) * -1 + f(x+2, y-1) *  0
    f(x-2,  y ) * -1 + f(x-1,  y ) * -2 + f(x,  y ) * 16 + f(x+1,  y ) * -2 + f(x+2,  y ) * -1
    f(x-2, y+1) *  0 + f(x-1, y+1) * -1 + f(x, y+1) * -2 + f(x+1, y+1) * -1 + f(x+2, y+1) *  0
    f(x-2, y+2) *  0 + f(x-1, y+2) *  0 + f(x, y+2) * -1 + f(x+1, y+2) *  0 + f(x+2, y+2) *  0
  }
f(x, y) = LOG(x, y);      // La imagen original adquiere el
                          // valor de la imagen filtrada.
```

[17] Laplaciano del Gaussiano extraído de - http://leibniz.iimas.unam.mx/~yann/Vision/Clase02.pdf - Pag. 22.

Aplicando este procedimiento se obtiene la imagen de la figura 3.4.27.

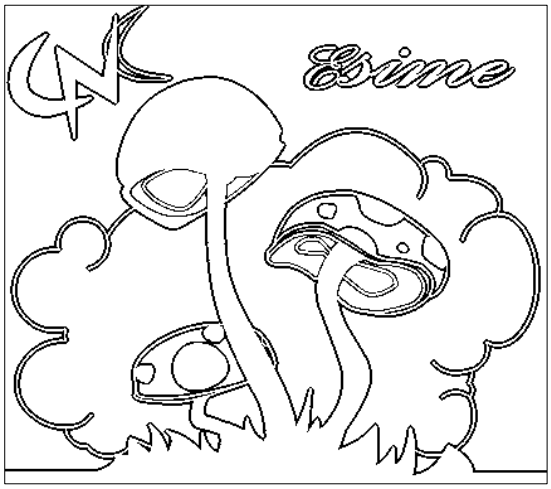


Figura 3.4.27 Laplaciano del Gaussinao (LOG).

La figura 3.4.27, muestra el resultado de aplicar el filtro LOG a la imagen de la figura 3.4.4 (no olvidando que se ha usado la función *invertir colores* para obtener el fondo blando y los bordes negros). Esta imagen al ser comparada con los resultados obtenidos en el filtro de Laplace presenta bordes más gruesos.

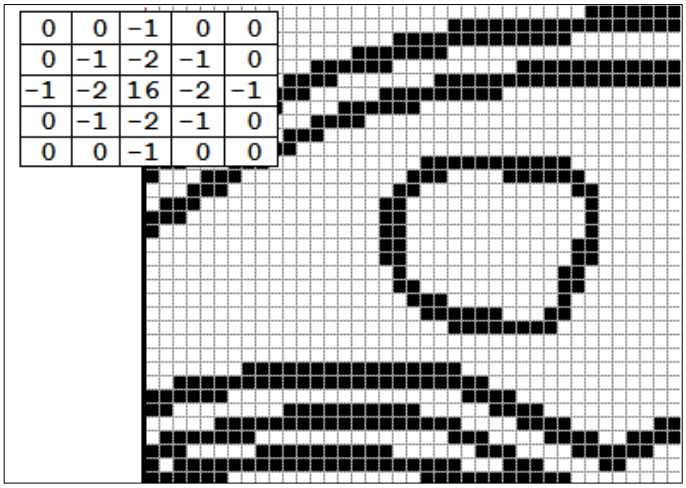


Figura 3.4.28. Acercamiento de la imagen para observar los pixeles resultantes en la detección de bordes LOG.

La figura 3.4.28 representa un fragmento de la imagen en la figura 3.4.27 aumentada en tamaño para observar los pixeles. Con este acercamiento se logra observar el grosor de los bordes, con lo que se determinó que son más gruesos que los obtenidos en los filtros de Laplace.

3.5 Evaluación de Filtros.

A continuación se muestra un resumen las imágenes obtenidas en los filtros desarrollados en este capitulo, para hacer una evaluación y determinar cual será utilizado para detectar los bordes en el programa principal.

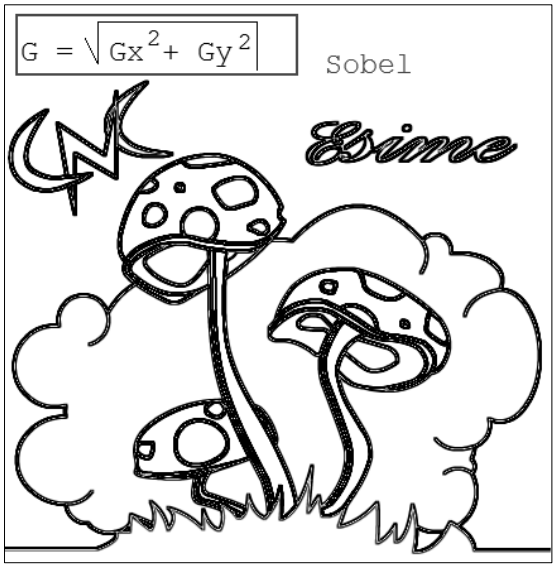


Figura 3.5.1. Filtro de Prewitt.

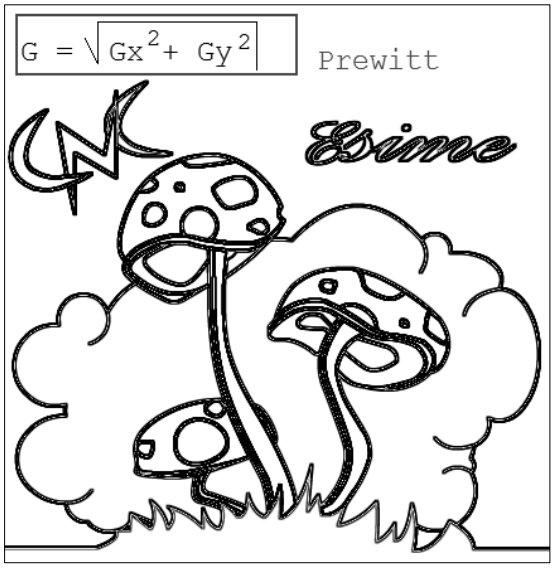


Figura 3.5.2. Filtro de Sobel.

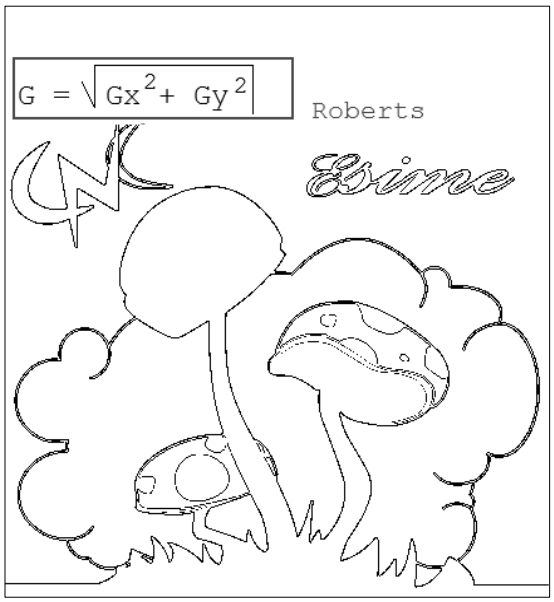


Figura 3.5.3. Filtro de Roberts.

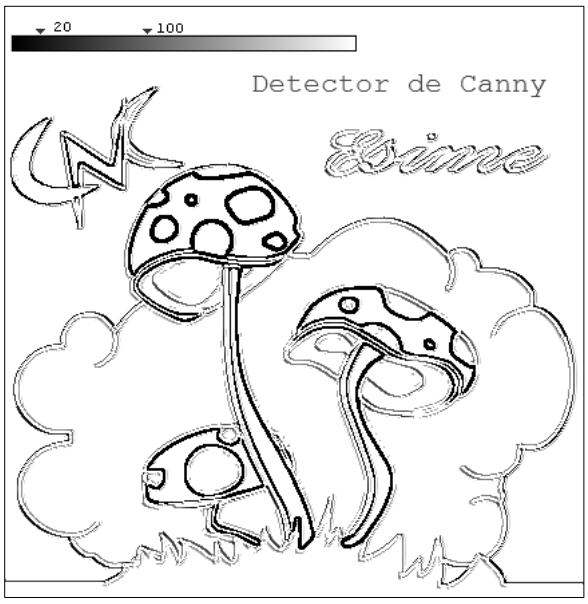


Figura 3.5.4. Filtro de Canny con umbral1 = 20 y umbral2 = 200.

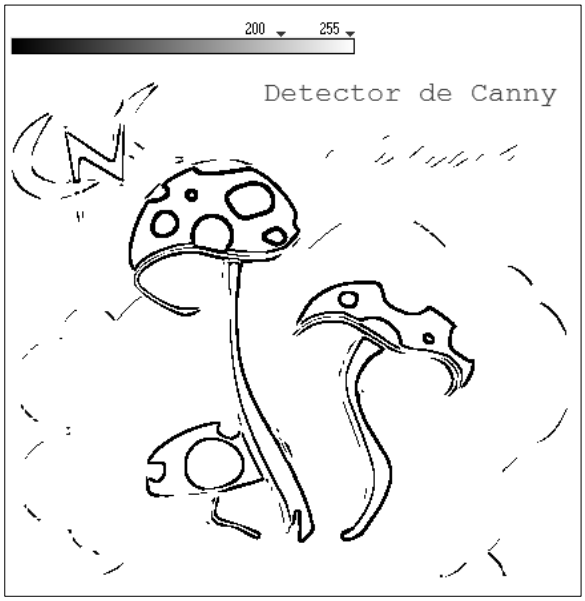


Figura 3.5.5. Filtro de Canny con umbral1 = 200 y umbral2 = 255

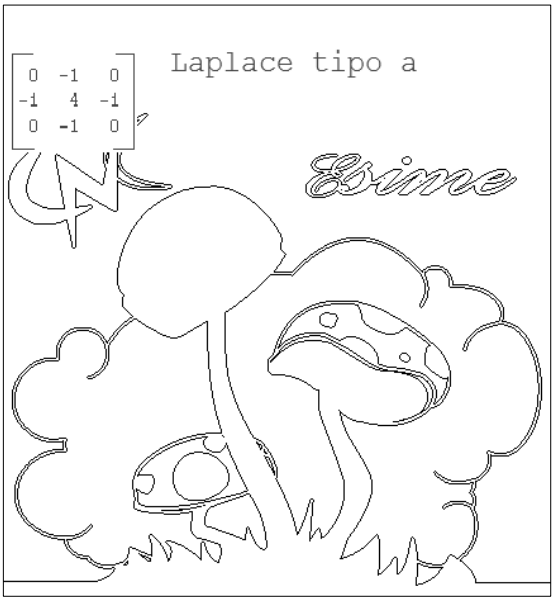


Figura 3.5.6. 1er Filtro de Laplace.

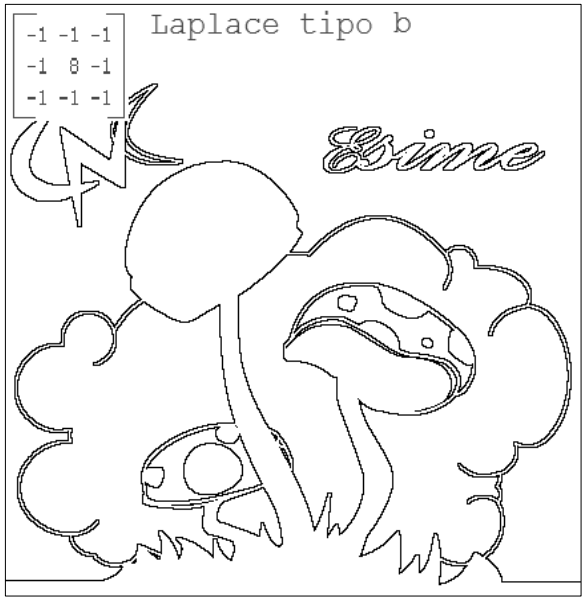


Figura 3.5.7. 2do Filtro de Laplace

Figura 3.5.8. Filtro LOG.

Observando las figuras (3.5.1 a 3.5.8) contenidas en este subtema, se puede determinar a simple vista, cuales presentan bordes útiles para la *Vectorización* (tema que se desarrolla con detalle en el capitulo 4), la cual requiere algunas características en la imagen para funcionar de manera optima. Las características que debe presentar una imagen para poder ser vectorizada son las siguientes:

- Contar con bordes definidos y trayectorias completas.

- Tener bordes delgados de 1 pixel de grosor.
- Presentar la mayor cantidad de información de contornos de la imagen original (figura 3.4.3).
- Presentar la menor cantidad de pixeles no pertenecientes a la imagen original (figura 3.4.3).

La última y penúltima característica, pueden ser consideras como características secundarias, ya que la vectorización de las trayectorias puede realizarse correctamente si se siguen las primeras dos características.

Observando la figura 3.5.1 y 3.5.2, se tienen 2 imágenes similares, ambas presentan bordes definidos con trayectorias completas, pero no presentan bordes delgados de 1 pixel, por lo que no pueden ser considerados estos filtros para ser usados en la vectorizacion.

La figura 3.5.3, presenta algunos bordes delgados de 1 pixel, pero otros son más gruesos, además algunos bordes tiene pixeles que fueron suprimidos, por lo que no se completan algunas trayectorias de los bordes, debido a esto, este filtro queda descartado.

La figura 3.5.4 y 3.5.5, muestran imágenes que no presentan ninguna característica útil para la vectorización, debido a esto, el filtro de Canny fue descartado.

La figura 3.5.6 muestra una imagen con bordes bien definidos y con un pixel de grosor, lo que permite realizar la vectorización sobre la imagen que produce este filtro, además, no genera bordes falsos. El filtro tiene un inconveniente, pues no presenta toda la información de bordes que contiene la imagen original (figura 3.4.3).

La figura 3.5.7, puede ser similar a la figura 3.5.6, pero presenta algunos bordes con 2 pixeles de grosor, quedando descartado para su uso en la vectorización.

La figura 3.5.8, presenta mayor información de los bordes que la figura 3.5.6 y 3.5.7, ya que muestra más bordes correspondientes a la imagen original, además las trayectorias son completas y bien definidas, pero el resultado de este filtro muestra contornos de 3 pixeles de grosor en algunas partes, por lo que ha quedado descartado para ser aplicado en la vectorización.

En un principio, se pensaba que el filtro de Canny (figura 3.5.4 y 3.5.5) proporcionaría las características que se requieren para la vectoriación, pero los resultados no fueron favorables, en cambio, el filtro de Laplace (figura 3.5.6), a pesar de ser considerado un filtro de alta sensibilidad al ruido, proporcionó el resultado que se esperaba de Canny y cubrió las principales características para realizar la vectorización, debido a esto, el filtro de Laplace visto en la figura 3.4.23 inciso a), ha sido seleccionado como filtro predeterminado para la detección de bordes.

Capitulo 4

Vectorización

4. VECTORIZACIÓN.

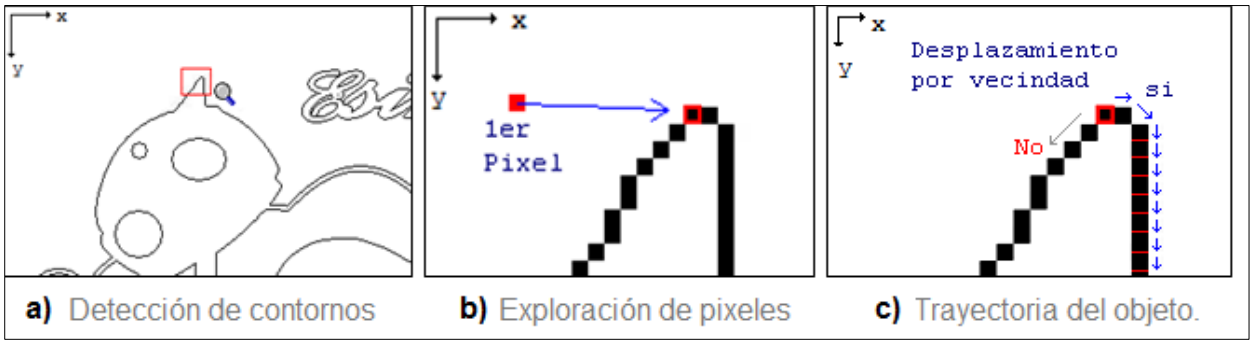
La vectorización consiste en el arreglo u ordenamiento de las coordenadas de los pixeles de una imagen, de tal manera que la secuencia de sus coordenadas toma la forma de trayectorias definidas cuando se recorre la matriz de los puntos almacenados.

Durante el desarrollo de este capítulo se definen 2 formas de vectorizar:

- Convirtiendo los contornos de la imagen en vectores.
- Creando vectores a partir de una paleta de herramientas.

4.1 Convertir Imagen a Vectores.

Después de obtener una imagen con líneas definidas y un píxel de grosor, se puede proceder a convertir los datos de la imagen en datos de conectividad, lo que significa extraer información de los pixeles que se encuentran conectados entre si (que tengan vecindad) y que de acuerdo a la conexión formen un objeto o una trayectoria.



Figuras 4.1.1- Método para la obtención de la trayectoria formada por vecindad entre pixeles.

La figura 4.1.1 muestra un procedimiento para la obtención de las trayectorias que forman una imagen. El inciso a) de la figura 4.1.1, es el resultado obtenido de filtrar una imagen con el filtro propuesto en el capitulo 3, el inciso b) muestra un acercamiento que corresponde a un fragmento de la imagen del inciso a) marcado con una lupa, esta imagen señala la posición del primer pixel detectado, en el inciso c) se observa la trayectoria que deberá seguir el recorrido de los pixeles vecinos.

Siguiendo el procedimiento de la figura 4.1.1, se pretenden obtener los vectores ordenados para formar los bordes de la imagen. En la figura 4.1.2 se observa la trayectoria que se obtendría si se ordenan los vectores de los pixeles conectados entre si o que son vecinos y corresponden a un borde de la imagen.

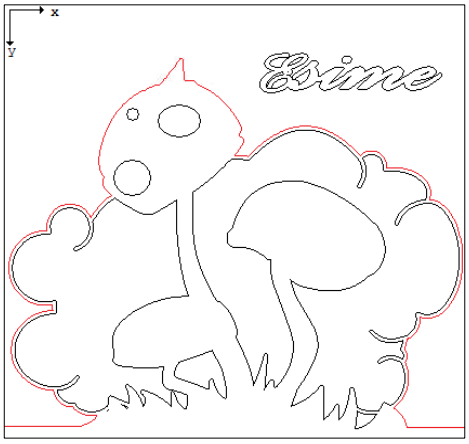


Figura 4.1.2. Trayectoria que se forma si se sigue la secuencia de vecindad.

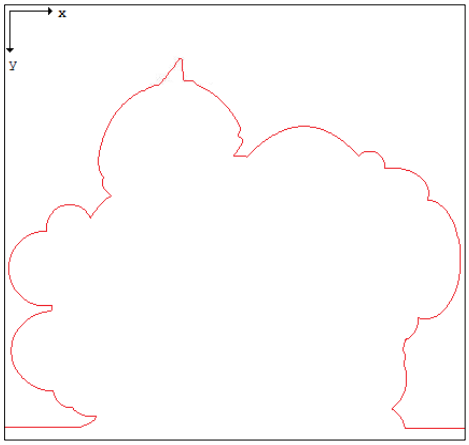


Figura 4.1.3. Objeto aislado que se forma cuando se ha terminado el camino de la exploración inicial.

La figura 4.1.2, muestra el resultado de una trayectoria generada por la conectividad entre pixeles vecinos y la figura 4.1.3, muestra la trayectoria de la figura 4.1.2 de manera aislada de los pixeles que no corresponde a la trayectoria o que no son vecinos entre si de la primera exploración de conectividad.

Para empezar, se hace un recorrido en la matriz de la imagen en busca de pixeles con valor 0 (pixeles color negro que representan los bordes de una imagen), cuando se encuentre un pixel con ese valor, se almacena la posición del pixel en una matriz de vectores. Al finalizar el recorrido en la matriz de la imagen, se tendrá una matriz con vectores de posición X-Y de los pixeles negros que pertenecen a la imagen.

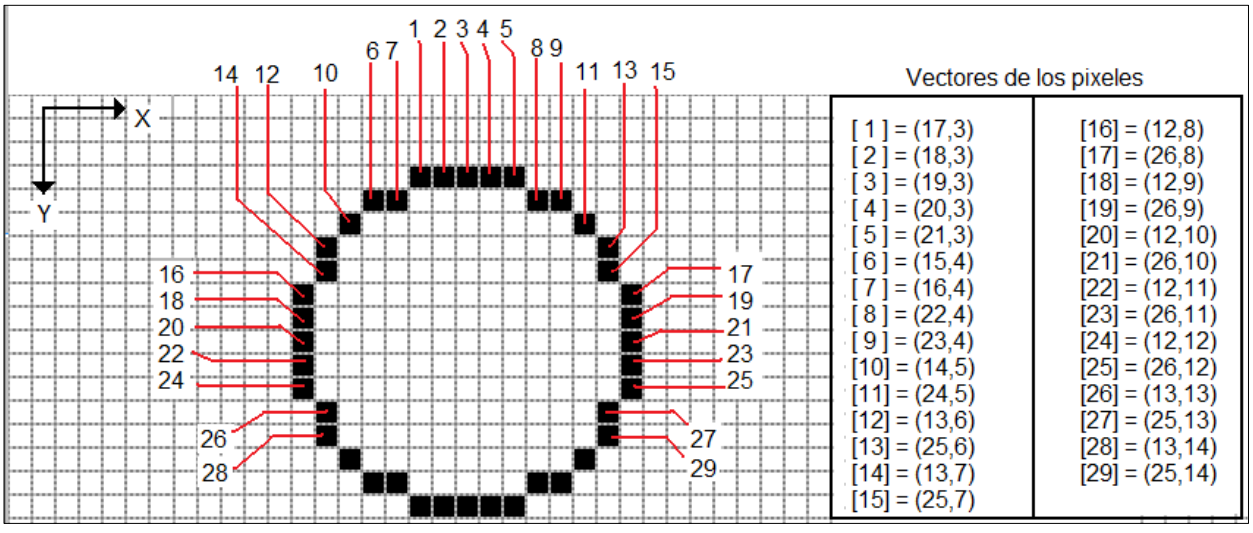


Figura 4.1.4. Almacenamiento de las posiciones de los pixeles en el plano X, Y de la imagen.

La figura 4.1.4, muestra una imagen, donde se pueden apreciar los pixeles con valor 0 que la forman, estos pixeles están marcados por una numeración que indica la forma en la que fueron almacenados en la matriz de vectores, la numeración se debe a que el programa escanea la imagen de izquierda a derecha (dirección de X) en busca de pixeles con valor 0 (color negro) y cuando termina baja 1 pixel (dirección Y) y regresa a la posición inicial de X,

repitiéndose el ciclo hasta terminar el recorrido en la matriz imagen, de tal manera que se almacenan las posiciones de los pixeles como se observa en la parte derecha de la figura 4.1.4.

Como se observa, estos valores almacenados, representan los vectores de la imagen, pero no están ordenados por conectividad, ya que si se hace el recorrido de esta matriz, redibujando por medio de las posiciones almacenadas, se formaría la imagen renglón por renglón y si se adapta la información de vectores adquirida hasta el momento a una máquina que interprete los datos, se tendrá un efecto no deseado.

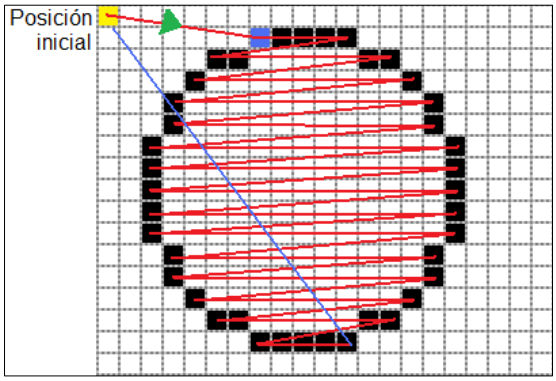


Figura 4.1.5. Trayectoria que se sigue para trazar una imagen sin vectores ordenados.

La figura 4.1.5 muestra el recorrido que tiene que seguir un dispositivo de desplazamientos para trazar los contornos de un circulo si la posición de sus pixeles no estas ordenados por conectividad, esto se debe a que los vectores están ordenados renglón por renglón, dando el efecto que se muestra. Este efecto seria valido si se pretende rellenar el área del contorno, pero para efectos de trazado no tiene validez la información, por lo que es necesario ordenar la posición de los pixeles por medio de la conectividad.

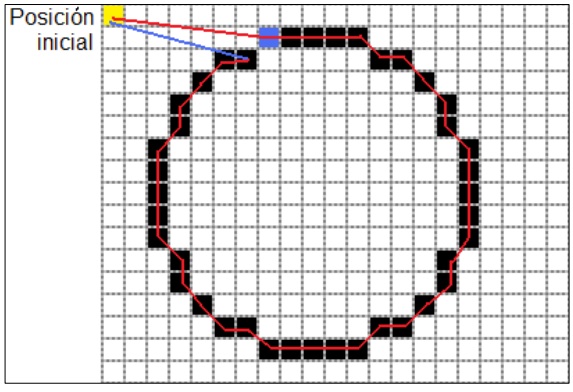


Figura 4.1.6. Trayectoria que se sigue para trazar una imagen con vectores ordenados.

La figura 4.1.6, muestra la trayectoria que debe seguir un dispositivo si utiliza la información de los vectores ordenados por conectividad. Esta forma de trazar una figura es la que generalmente realiza un dispositivo de control numérico, donde solo se desplaza sobre una línea recta para llegar a la posición inicial de la figura, después se desplaza sobre la trayectoria de la figura y finalmente en línea recta a la posición inicial.

Condición de conectividad.

Para obtener trayectorias por conectividad, se debe verificar si un pixel tiene vecindad con otro, para esto se resta la posición de un pixel con otro y se verifica el resultado.

Sea $p_m(x_m, y_m)$ la posición de un pixel color negro de la imagen f(x, y) y $p_n(x_n, y_n)$ la posición de otro pixel color negro perteneciente a la misma imagen f(x, y).

Habrá conectividad si $p_n(x_n, y_n) - p_m(x_m, y_m) =$
(1, 1) ó (0, 1) ó (-1, 1) ó (-1, 0)
ó (-1,-1) ó (0,-1) ó (1,-1) ó (1, 0)

Donde el resultado son 8 posibles valores que determinan si un pixel esta conectado a otro o no. Esto se puede observar en la figura 4.1.7.

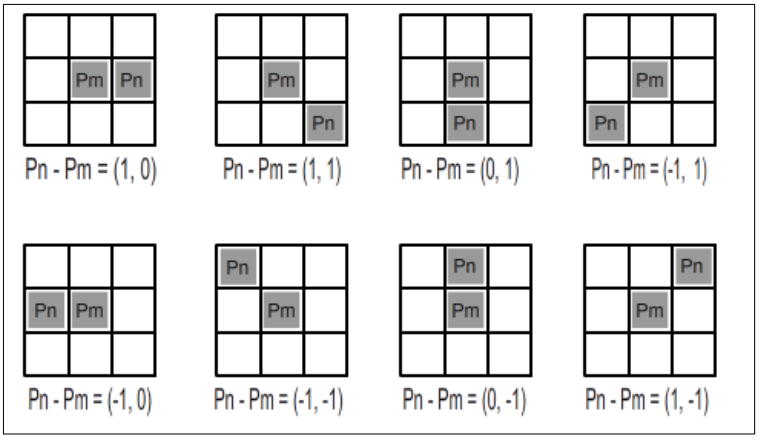


Figura 4.1.7. Existencia de conectividad entre pixeles.

La figura 4.1.7, expresa los 8 valores obtenidos en la condición de conectividad, donde se indica que posición toma cada pixel dependiendo del valor que se obtuvo en la resta de posiciones entre pixeles.

El siguiente ejemplo muestra si existe conectividad entre la posición de los pixeles.

Si el pixel $p_m(x_m, y_m) = (100,25)$ y el pixel $p_n(x_n, y_{jn}) = (100,27)$

$p_n(x_n, y_n) - p_m(x_m, y_m) = (100, 27) – (100, 25) = (0, 2)$

No existe conectividad por que no es igual a alguno de los valores establecidos.

Si el pixel $p_m(x_m, y_m) = (100,27)$ y el pixel $p_n(x_n, y_{jn}) = (100,26)$

$p_n(x_n, y_n) - p_m(x_m, y_m) = (100, 26) – (100, 27) = (0, -1)$

Tiene conectividad por que es igual a uno de los valores establecidos (0, -1).

De esta manera se puede verificar si un pixel tiene conectividad con otro y si existe conectividad, se almacena el valor de la posición del primero (en este caso $p_m(x_m, y_m)$) en

una matriz de vectores ordenados y el primero adquiere el valor de la posición del segundo ($p_n(x_n, y_n)$), para que el segundo adquiera un nuevo valor y se pueda encontrar el siguiente vector con conectividad.

La codificación del procedimiento para encontrar la conectividad entre pixeles, es la que se muestra a continuación:

Pseudocódigo.

Sea p un píxel en (x, y)

```
Hay un píxel vecino horizontal o vertical si:
{
     P(x, y) = P(x+1, y)  // Píxel vecino a la derecha
     P(x, y) = P(x-1, y)  // Píxel vecino a la izquierda
     P(x, y) = P(x, y+1)  // Píxel vecino a abajo
     P(x, y) = P(x, y-1)  // Píxel vecino a arriba
}
Hay un píxel vecino diagonal si:
{
     P(x, y) = P(x+1, y+1)  // Píxel vecino esquina derecha-abajo
     P(x, y) = P(x+1, y-1)  // Píxel vecino esquina derecha-arriba
     P(x, y) = P(x-1, y+1)  // Píxel vecino esquina izquierda-abajo
     P(x, y) = P(x-1, y-1)  // Píxel vecino esquina izquierda-arriba
}
```

Si se cumple cualquiera de estas condiciones, el píxel p(x, y) tendrá a lo menos un pixel vecino, de ser así, se asigna el vecino como nuevo píxel y el anterior se guarda en un arreglo de vectores ordenados y se vuelve a comprobar si hay un pixel vecino para el nuevo pixel, realizando esta operación hasta que ya no se encuentre pixeles vecinos.

Cuando se buscan ordenar los pixeles por medio de la vecindad entre ellos, se encuentran algunos problemas durante la exploración y determinación del vector a ordenar, por lo que es necesario establecer algunas condiciones que solucionen el problema.

Problema de Vecindad 1.

Un problema de conectividad se presenta cuando un píxel encuentra múltiples pixeles vecinos, por lo que hay que determinar un camino ideal.

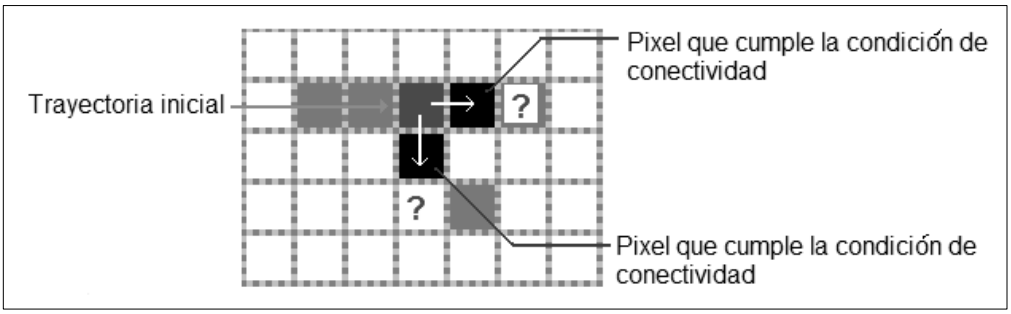
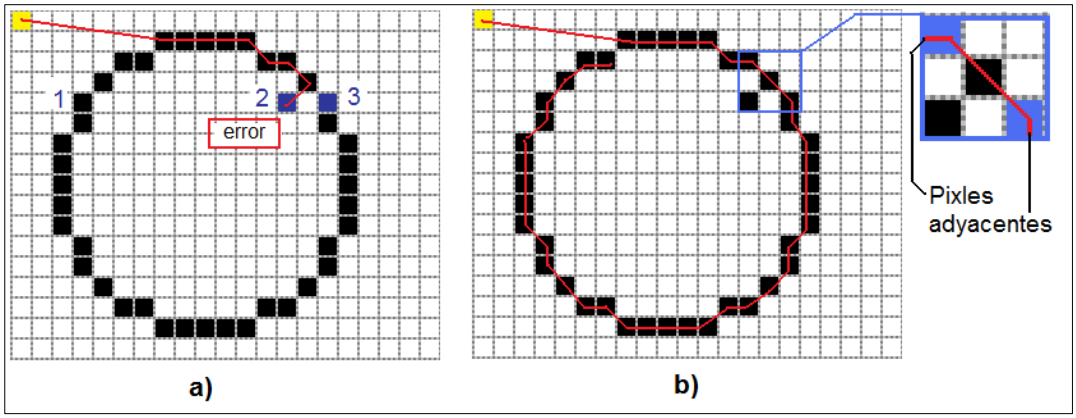


Figura 4.1.8. Problema de conectividad múltiple.

La figura 4.1.8, muestra un problema que suele presentarse cuando las trayectorias de los contornos de una imagen tienen múltiples caminos o pixeles conectados. Si se ignora esta cuestión, la selección del camino será definida por el primero que detecte la exploración de caminos (trayectoria por jerarquía), que muchas veces no es el correcto, pues existe más probabilidad de que el camino correcto que debe seguir la trayectoria es el pixel que se encuentra adyacente a la trayectoria que lleva la conectividad de pixeles, pues muchas veces en una trayectoria suelen aparecer pixeles falsos provocados por el ruido en una imagen. La figura 4.1.9 muestra un ejemplo de esta situación.



La figura 4.1.9. Diferencia de elegir trayectoria por jerarquía y por adyacencia.

La figura 4.1.9, muestra la diferencia de elegir almacenar una trayectoria por jerarquía (inciso a) y almacenar una por adyacencia (inciso b). En la figura 4.1.9, se puede apreciar los pixeles que forman un círculo en ambos incisos, pero debido al ruido en la imagen se pudo haber formado un pixel falso, que por sentido común, se puede determinar cual es el pixel que no pertenece a la circunferencia, pero el programa de exploración carece de sentido común para detectar este posible error, debido a eso, si se selecciona un camino por jerarquía, que consiste en tomar el primer pixel que detecte como vecino, el programa almacenará esa trayectoria y seguirá el camino que muestra el inciso a), debido a que el primer pixel que detecta es el numero 2 y no el numero 3, terminándose el recorrido en ese punto. Por otro lado, si se toma el primer pixel, pero se detecta que existe un pixel adyacente a la trayectoria de la conectividad de vectores y se toma esta trayectoria, se tiene mayor probabilidad que se esté tomando el camino correcto, tal y como lo muestra la figura 4.1.9 inciso b). Esto no quiere decir que siempre sea este el camino correcto, pero hay mayor probabilidad que un pixel adyacente a otro no sea ruido de una imagen, sino parte de la trayectoria de una imagen.

Ahora lo que se debe determinar, es como reconocer que un pixel es adyacente con otro. Para esto se asignan etiquetas a la dirección que lleva la trayectoria de conectividad como lo muestra la figura 4.1.10.

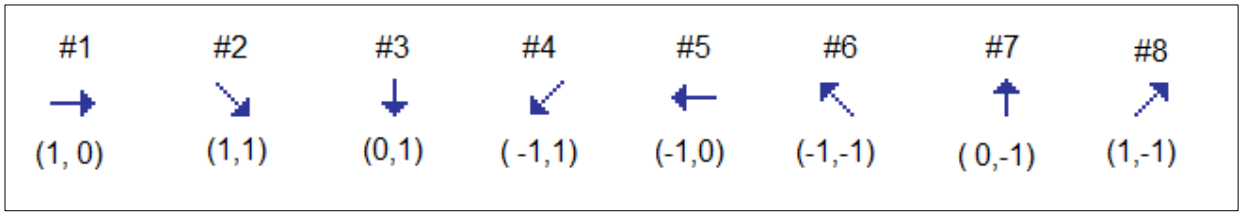


Figura 4.1.10. Etiquetas de dirección de una trayectoria.

La figura 4.1.10, muestra las 8 posibles direcciones que puede tomar una trayectoria siguiendo los resultados de la figura 4.1.7, pero en la figura 4.1.10, se ha asignado a cada dirección un valor numérico de 1 al 8. Con esta numeración se puede determinar que dirección tiene una trayectoria de acuerdo al resultado de la resta entre la posición de los pixeles. Si en la exploración de conectividad se tiene una conectividad con el valor (1,0), se almacena la etiqueta que corresponde al #1 de acuerdo a la figura 4.1.10 y si en la siguiente exploración se tienen dos pixeles vecinos, con identificar cual corresponde a la etiqueta #1 se determina cual es adyacente. De esta manera se puede estimar en determinado momento el camino adecuado que debe seguir el trazo.

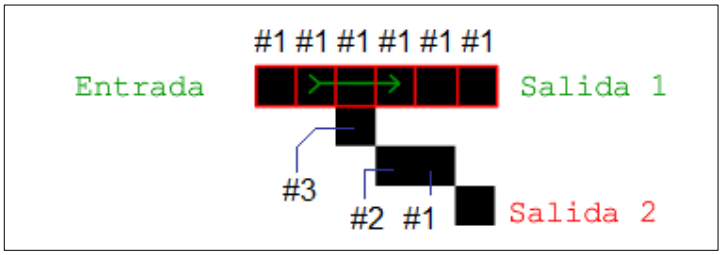


Figura 4.1.11. Selección de caminos usando adyacencia entre vecinos.

El la figura 4.1.11, se muestra la exploración de una imagen que presenta dos posible caminos para la entrada de una trayectoria, la selección por adyacencia otorga la salida 1, ignorando un desplazamiento por la salida 2.

Problema de Vecindad 2.

Otro problema que suele presentarse, es producido por múltiples vecinos sin vecinos adyacentes, como se muestra en la figura 4.1.12.

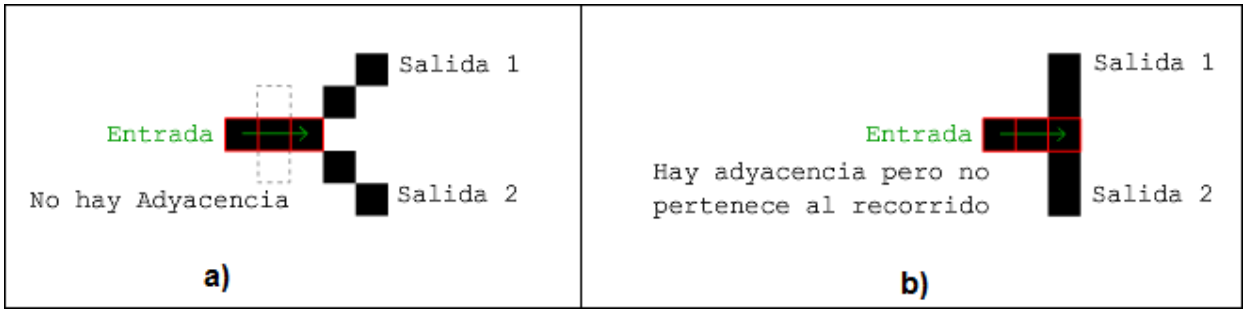


Figura 4.1.12. Falta de adyacencia entre vecindad de pixeles en la trayectoria.

La figura 4.1.12, muestra 2 casos: en el inciso a) el pixel no tiene vecinos adyacentes para determinar una selección de camino, en el inciso b) se tienen vecinos adyacentes, pero no son adyacentes al recorrido que lleva el pixel.

Para solucionar el problema se tienen que tomar algunas consideraciones:

- Verificar si no existe adyacencia inmediata entre vecinos:
    - Si es el primer píxel de exploración, no habrá registro de vecino adyacente, por lo que se asignará una ruta por jerarquía (primer pixel vecino encontrado).

- Si los pixeles posteriores al inicial no cambiaron de dirección y se presentan múltiples caminos sin adyacencia se asigna la ruta por jerarquía.

- Si la ruta seleccionada por jerarquía se explora previamente y si se encuentra un camino con una longitud menor a 2 pixeles conectados se debe descartar esta dirección.

- Si la segunda ruta por jerarquía es igual de pequeña que la primera se descarta.

- Si no existe alguna ruta mayor a la primera, se marcará la primera y se eliminaran las otras posibles.

- Si existe una ruta con longitud mayor a las anteriores se almacenan los vectores de esa trayectoria.

- Verificar si no existe adyacencia a lo largo de la trayectoria.

  - Si a lo largo de un una trayectoria hubo cambio en la dirección del píxel, se registra con una etiqueta de dirección (adyacencia acumulada) para ser usada cuando existan múltiples vecinos, otorgando prioridad al que sea adyacente al registrado, de lo contrario se asignará por jerarquía.

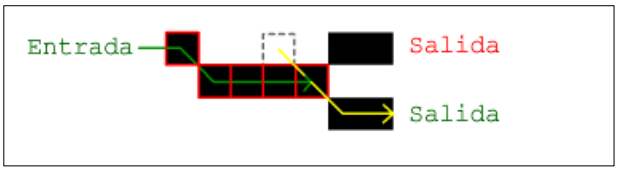


Figura 4.1.13. Determinación de un camino por medio de adyacencia registrada previamente (adyacencia almacenada).

La figura 4.1.13, muestra la trayectoria que seguiría un pixel si sus vecinos no son adyacentes, pero uno de los pixeles vecinos es adyacente a un cambio de trayectoria que se tuvo en pixeles anteriores (adyacencia almacenada).

Esta condición resuelve el problema si se trata de una trayectoria parecida a una línea ó una curva poco pronunciada, en la que hay pixeles colocados en su orilla que no deben ser tomados en cuenta.

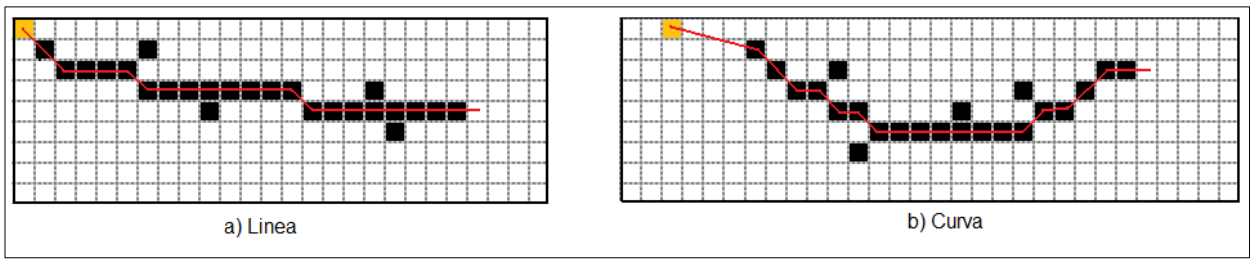


Figura 4.1.14. *Trayectoria que se almacena en la matriz de vectores ordenados usando la condición de adyacencia.*

La figura 4.1.14, muestra la trayectoria que se forma si se obedece la condición de adyacencia y adyacencia almacenada, en la que los pixeles que no fueron tomados en cuenta, al no tener más de 1 conectividad son descartados en el almacenamiento de vectores ordenados.

Por otro lado, esta condición presenta inconvenientes cuando algunos pixeles aparecen en posiciones indeseables de la imagen.

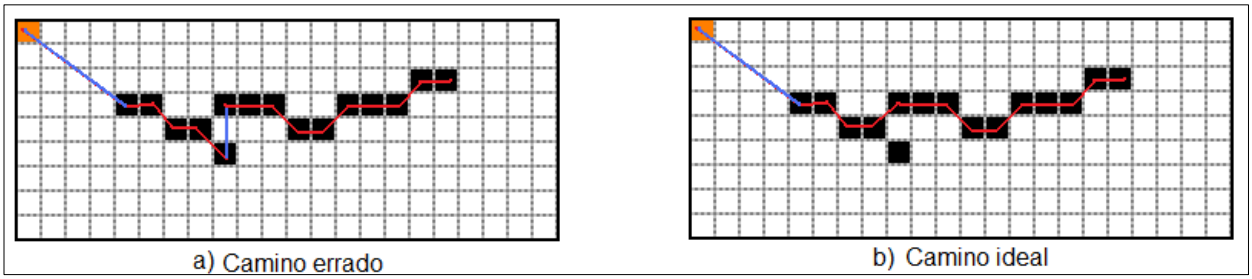


Figura 4.1.15, *Error que se presenta con la condición de adyacencia.*

La figura 4.1.15, muestra un error simple que se puede presentar con la adyacencia, en el inciso a) de la figura 4.1.15 se ha tomado la trayectoria siguiendo la condición de adyacencia, pero como se observa, siguiendo esta trayectoria no se completa un segmento largo y se tiene que desplazar al próximo segmento. Si se ignora la trayectoria por adyacencia y se elige el camino que genera un segmento más largo se obtiene la figura 4.1.15 del inciso b), por lo que se ha generado una nueva condición para la elección de caminos conectados.

Problema de Vecindad 3.

Un problema muy sencillo de resolver, se presenta cuando en una imagen encontramos un pixel aislado o dos, en los que no existen trayectorias, para estos pixeles se ha elegido su eliminación, debido a que pueden ser considerados como ruido de la imagen.

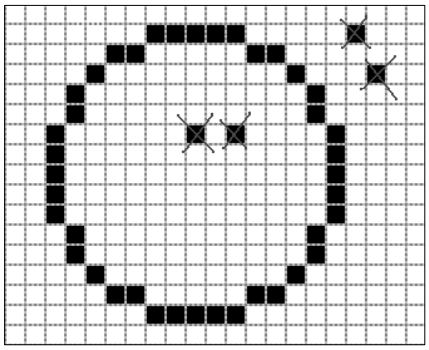


Figura 4.1.16. Eliminación de pixeles aislados sin una sola conectividad.

La figura 4.1.16, muestra 4 pixeles descartados para el almacenamiento de vectores ordenados, debido a que estos pixeles no producen trayectorias y si llegaran a ser necesarios, pueden ser fácilmente agregados.

Hasta el momento estos son los casos más importantes de vecindad que puede resolver el algoritmo del programa para determinar que caminos tomar, pero no se han resuelto todos

los problemas que pueden presentarse en la elección del camino ideal, debido a que existe una infinidad de formas de presentar las trayectorias de una imagen y cada que se genere una nueva condición para resolver el problema, se puede generar un nuevo problema.

En el Apéndice B parte I se encuentra el código fuente escrito en C# que genera la función de convertir los contornos en trayectorias.

4.2 Generar Vectores.

Para generar vectores en la aplicación se introducen algunas herramientas útiles en el dibujo, para facilitar la creación de un diseño con vectores. Las herramientas que se van a desarrollar en este capitulo son:

- Trazador de líneas.
- Trazador de rectángulos.
- Trazador de elipses,
- Trazador libre.
- Trazador de Texto.

MÉTODOS MATEMÁTICOS.

Los métodos usados en la formación de vectores, están basados en funciones geométricas, ya que estas devuelven la posición de cada punto en el plano x-y para formar una figura geométrica. Otra ventaja no visible del método matemático es que las funciones pueden ser usadas de manera analógica o discreta, pudiendo generarse una figura con la cantidad de puntos que se desee.

C# proporciona las herramientas para generar algunas figuras geométricas como: línea, rectángulo, elipse, texto, etc. y proporciona la información de la posición inicial y final de los puntos con los que se genero la figura, pero no proporciona la información de la posición de cada pixel usado para la formación de la figura, haciendo necesario el cálculo de estas posiciones por medio de los métodos matemáticos.

4.2.1 Trazador de líneas.

Para hacer funcionar esta herramienta se requieren 3 pasos:

1. Seleccionar la herramienta (se activa el uso de la función línea).
2. Conocer la posición del cursor dentro del área de diseño cuando se ha presionado el botón izquierdo del mouse (pixel inicial de la trayectoria).
3. Mover y capturar la posición del cursor dentro del área de diseño cuando se ha soltado el botón derecho del mouse (pixel final de la trayectoria).

Ahora, suponiendo que se ha oprimido el botón del mouse en la coordenada (100,100) y se ha soltado en la coordenada (126, 116), se obtiene una imagen como se muestra en la figura 4.2.1.

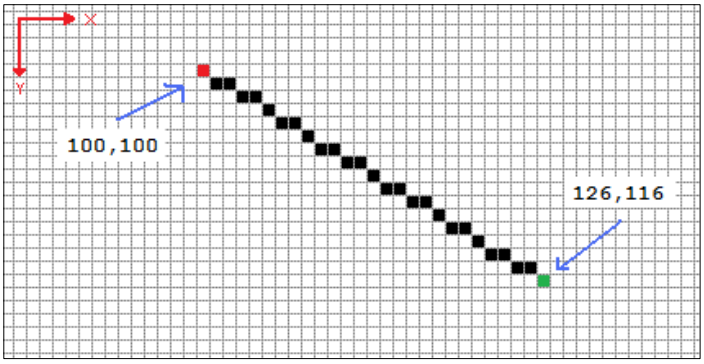


Figura 4.2.1. Coordenadas del pixel inicial y pixel final de una línea.

La figura 4.2.1, muestra el trazo que se forma al usar la herramienta línea, en la que se han establecido dos coordenadas para formarla.

Teniendo las coordenadas del punto inicial y del punto final de la línea, se pude determinar por medio de la pendiente de una recta (ecuación 4.2.1), de que manera se desplazan los vectores para obtener la trayectoria que lleva la línea.

$$m = \frac{yf - yi}{xf - xi}$$ **Ecuación 4.2.1**

donde: xi, yi son los puntos iniciales

xf, son los puntos finales

m es el valor de la pendiente

sustituyendo

$$m = \frac{116 - 100}{126 - 100} = \frac{16}{26} = \boxed{\frac{8}{13}}$$

Como `m = 8/13`, significa que por cada pixel desplazado una unidad en X se desplazara 8/13 de unidad en Y. Esto se puede expresar de la siguiente manera:

Por cada iteración X_k `= xi + k,` Y_k `= yi + (8/13 * k).`

Donde X_k, Y_k, son las coordenadas del pixel que serán almacenados en la matriz de vectores ordenados y `xi, yi,` son la posición inicial de los vectores.

En la línea recta que forma esta herramienta, los pixeles siempre se desplazan una unidad en la coordenada del eje más grande, observando este detalle, se determina que la cantidad de elementos que puede contener la matriz, es igual al eje de mayor longitud en la recta, debido a que los elementos en este eje no se repiten.

Para determinar cual es el eje de mayor longitud se obtiene la diferencia entre la distancia de separación de cada eje.

$$|Xi - Xf| = |100 - 126| = |-26| = 26$$
$$|yi - yf| = |100 - 116| = |-16| = 16$$

En este caso, el eje X tiene mayor longitud, por lo que su valor representa el número de elementos en la matriz más 1. Por lo tanto, el número de iteración para obtener los puntos de la recta y el tamaño de la matriz para almacenarlos debe ser de 27.

Tomando los valores anteriores y realizando las respectivas iteraciones con incrementos de 1 en k se tiene la tabla 4.2.1.

Item	$X_k = xi + k$	$Y_k = yi + (8/13 * k)$	Matriz Vector
k = 0	$X_0 = 100 + 0$	$Y_0 = 100 + (8/13 * 0)$	(100,100)
k = 1	$X_0 = 100 + 1$	$Y_0 = 100 + (8/13 * 1)$	(101,101)
k = 2	$X_0 = 100 + 2$	$Y_0 = 100 + (8/13 * 2)$	(102,101)
k = 3	$X_0 = 100 + 3$	$Y_0 = 100 + (8/13 * 3)$	(103,102)
k = 4	$X_0 = 100 + 4$	$Y_0 = 100 + (8/13 * 4)$	(104,102)
k = 5	$X_0 = 100 + 5$	$Y_0 = 100 + (8/13 * 5)$	(105,103)
k = 6	$X_0 = 100 + 6$	$Y_0 = 100 + (8/13 * 6)$	(106,104)
k = 7	$X_0 = 100 + 7$	$Y_0 = 100 + (8/13 * 7)$	(107,104)
⋮	⋮	⋮	⋮
k = 26	$X_0 = 100 + 26$	$Y_0 = 100 + (8/13 * 26)$	(126,116)

Tabla 4.2.1. Almacenamiento de los valores de la trayectoria formada por el trazador de líneas.

Con los valores de la tabla 4.2.1 se puede hacer una comparación con los puntos de la figura 4.2.1, para determinar si los vectores ordenados corresponden a la de la línea trazada.

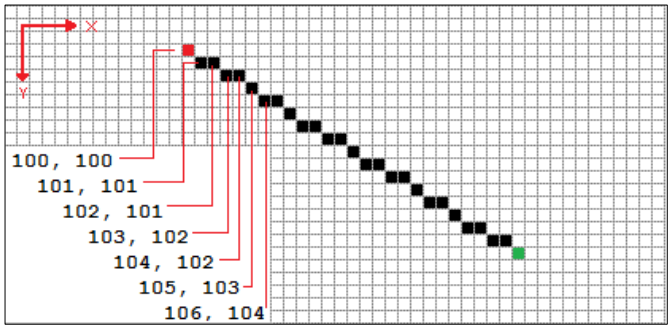


Figura 4.2.2. Identificación de la posición de los pixeles formados por el trazador de líneas.

La figura 4.2.2, muestra una serie de coordenadas en la parte inferior izquierda, que es utilizada para comparar los resultados de la tabla 4.2.1. De esta manera se obtienen los puntos del vector para almacenarlos en una matriz y usarlos en la simulación del prototipo. De igual manera, con estos datos se pueden calcular los saltos de los pixeles y con ello, los pulsos necesarios para mover los motores, como se propone en el paso 6 del análisis funcional.

4.2.2 Trazador de rectángulos.

El trazador de rectángulos es una herramienta para generar cuadros o rectángulos en la interfaz gráfica, almacena los puntos que forman al rectángulo en un matriz de vectores ordenados y pulsos para los motores. Para hacer funcionar esta herramienta se requieren los siguientes pasos:

1. Seleccionar la herramienta (se activa el uso de la función rectángulo).

2. Conocer la posición del cursor dentro del área de diseño cuando se ha presionado el botón izquierdo del mouse (pixel inicial de la posición del rectángulo).

3. Mover y capturar la posición del cursor dentro del área de diseño cuando se ha soltado el botón derecho del mouse (pixel final del dimensionamiento del rectángulo).

Con estos pasos es fácil obtener la información que será almacenada para trazar un rectángulo en la simulación.

La longitud de la matriz será igual al perímetro del rectángulo.

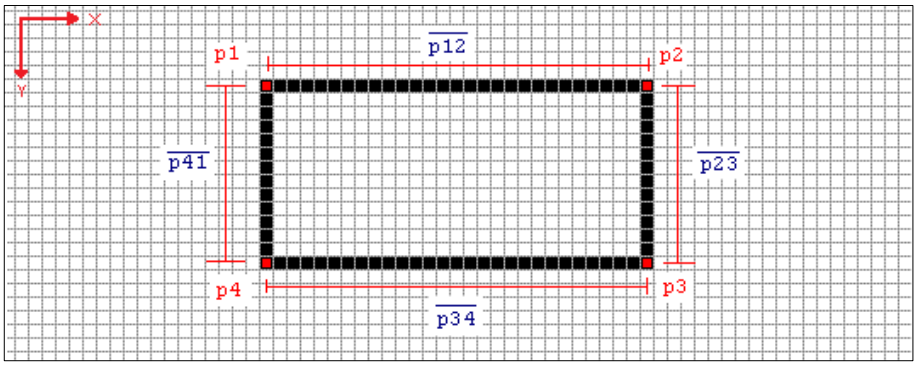


Figura 4.2.3. Identificación de los lados del rectángulo.

La figura 4.2.3 muestra como obtener los puntos de las rectas que forman el rectángulo y con estos valores se obtiene el perímetro y la longitud de la matriz. Los puntos p1, p2, p3 y p4 pertenecen a las esquinas del rectángulo y p12, p23, p34, p41 son los segmentos que forman el rectángulo.

Por ejemplo, si el usuario selecciona la herramienta rectángulo y oprime el botón del mouse en la posición (30,30), después el usuario suelta el botón del mouse en la posición (41, 35), establece de esta manera el punto inicial (posición) y el punto final (dimensionamiento) del rectángulo.

Teniendo estos datos se procesa la información de la siguiente manera:

El punto P1 corresponde al punto inicial establecido por el usuario y el punto p3 al punto final. Con esto puntos es posible obtener p2 y p4.

```
p2  = (X = coordenada X de p3, Y = coordenada Y de p1) = (41, 30).

p4 = (X = coordenada X de p1, Y = coordenada Y de p3) = (30, 35).
```

Con lo que se tienen las 4 esquinas del rectángulo.

```
p1 = (30, 30).  p2 = (41, 30).    p3 = (41, 35).    p4 = (30, 35).
```

Con estas esquinas se pueden establecer límites en los segmentos e ir almacenando los vectores por segmentos.

El siguiente procedimiento sirve para almacenar los puntos de los 4 segmentos del rectángulo.

Seudocódigo

```
La longitud de la matriz será:   lm = p12 + p23 + p34 + p41.
//Los valores se almacenan en la matriz de la siguiente manera.

i = 0.
MatrizVector[i] =  p1 + i,  i = i+1,   mientras px1 ≤ px2

MatrizVector[i] =  p2 + i,  i = i+1,   mientras py2 ≤ py3

j = 0.
MatrizVector[i + j] =  p3 + j,  j = j -1,   mientras px3 ≥ p4x

MatrizVector[i + j] =  p4 + j,  j = j -1,   mientras p1 ≤ p2
```

Lo que hace este proceso es almacenar en un matriz de vectores los puntos del primer segmento que va de p1 a p2 y como solo se desplaza los pixeles en el eje X, se establecen los límites en este eje, después se almacena los puntos del segmento que va de p2 a p3, estableciendo los limites en el eje Y, debido a que solo se desplazan los pixeles en este eje, los siguientes puntos que se almacenan son los del segmento que va de p3 a p4, en este segmento, se almacenan los puntos en decrementos de X, debido a que el vector de desplazamiento de un pixel a otro es en la dirección –X, el segmento que cierra la trayectoria del rectángulo va de p4 a p1, el cual tiene desplazamiento en la dirección –Y, por lo que los limites se establecen en decrementos de Y.

Ítem	Segmento P_{12}	Ítem	Segmento P_{23}	Ítem	
V0	P(30, 30)	V12	P(41, 31)	V18	P(34, 35)
V1	P(31, 30)	V13	P(41, 32)	V19	P(33, 35)
V2	P(32, 30)	V14	P(41, 33)	V20	P(32, 35)
V3	P(33, 30)	V15	P(41, 34)	V21	P(31, 35)
V4	P(34, 30)	V16	P(41, 35)	V22	P(30, 35)
V5	P(35, 30)		Segmento P34		Segmento P41
V6	P(36, 30)	V12	P(40, 35)	V23	P(30, 34)
V7	P(37, 30)	V13	P(39, 35)	V24	P(30, 33)
V8	P(38, 30)	V14	P(38, 35)	V25	P(30, 32)
V9	P(39, 30)	V15	P(37, 35)	V26	P(30, 31)
V10	P(40, 30)	V16	P(36, 35)		
V11	P(41, 30)	V17	P(35, 35)		

La tabla 4.2.2 Tabla de puntos obtenidos por cada segmento del rectángulo.

La tabla 4.2.2, representa los vectores ordenados que forman los segmentos del rectángulo, en la columna ítem se marca índice de la matriz y de lado derecho el vector que se incluye en ese índice. En la tabla 4.2.2, se indica en que momento empieza un segmento del rectángulo y después se despliegan los vectores que compone ese segmento, los valores presentados en esta tabla, pertenecen a los del ejemplo en el que se ha formado un rectángulo a partir de las coordenadas p1 = (30, 30) y p2 = (41, 35).

4.2.3 Trazador de elipses.

El trazador de elipses es una herramienta para generar elipses y círculos en la interfaz gráfica, almacenando los puntos que forman la elipse en un matriz de posiciones y pulsos para los motores.

Para hacer funcionar esta herramienta se requieren los siguientes pasos:

1. Seleccionar la herramienta (se activa el uso de la función elipse o círculo).

2. Conocer la posición del cursor dentro del área de diseño cuando se ha presionado el botón izquierdo del mouse (pixel inicial de la posición de la elipse).

3. Mover y capturar la posición del cursor dentro del área de diseño cuando se ha soltado el botón derecho del mouse (pixel final del dimensionamiento de la elipse).

Para generar los vectores que forman la trayectoria de una elipse, solo es necesario conocer el punto inicial y el punto final que establece el usuario con el cursor. Conociendo estos puntos se puede determinar de que tamaño son lo ejes menor y mayor del elipse.

Suponiendo que el usuario selecciona la herramienta *elipse* y presiona con el cursor en la coordenada (11, 21) y termina la figura geométrica soltando el botón del mouse en la coordenada (30,30), se obtiene una imagen como se muestra en la figura 4.2.4.

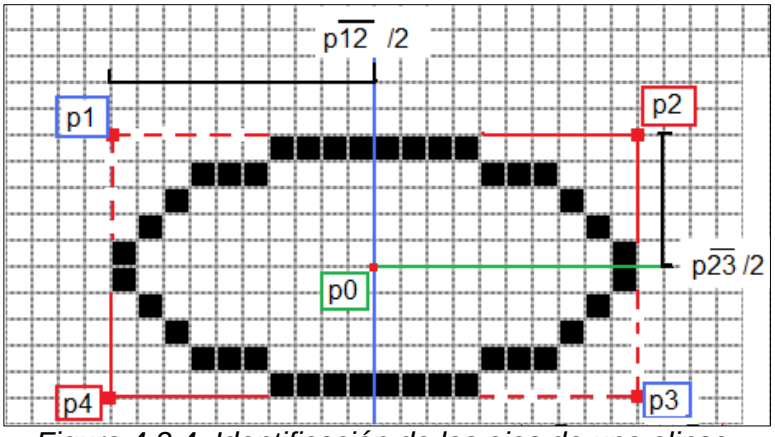


Figura 4.2.4. Identificación de los ejes de una elipse.

La figura 4.2.4, muestra el trazado de una elipse por medio de dos coordenadas, p1 es la primera coordenada establecida por el usuario y p3 la segunda coordenada establecida por el usuario que dimensiona la elipse. Para obtener las coordenadas de p2 y p4, se realiza el mismo procedimiento que en el rectángulo.

p2 = (X = coordenada X de p3, Y = coordenada Y de p1) = (30, 21).

p4 = (X = coordenada X de p1, Y = coordenada Y de p3) = (11, 30).

Así se obtienen las 4 esquinas del rectángulo:

p1 = (11, 21). p2 = (30, 21). p3 = (30, 30). p4 = (11, 30)

Con estos puntos se puede obtener el radio de los ejes X e Y de la elipse. De acuerdo a la figura 4.2.4 uno de los radios (semieje) se obtiene con la longitud del segmento p12 divido a la mitad y el otro radio con la longitud del segmento p23 dividido a la mitad. Con esto se tiene que:

Semieje1 = p12 / 2 = (p2 – p1) / 2
= [(30, 21) - (11, 21)] / 2= (9.5, 0) = (10, 0)

Semieje2 = p23 / 2 = (p3 – p2) / 2
= [(30, 30) - (30, 21)] / 2= (0, 4.5) = (0, 5)

De estos resultados se observa que el semieje1 solo tiene valor en X por lo que representa al semieje X y semieje2 solo tiene valor en Y por lo que representa el semieje Y.

Semieje X = 10. Semieje Y = 5.

Y con estos semiejes se establece el centro del origen de la elipse, marcado como p0 en la figura 4.2.4.

p0 = (semieje X, semieje Y) + P1 = (10, 5) + (11, 21) = (21, 26)

Ahora que se cuenta con todos los datos, se introduce una función que genere los puntos de una elipse. Para esto se puede usar la ecuación de la elipse (Ecuación 4.2.2).

$(X^2 / a^2) + (Y^2 / b^2) = 1$ **Ecuación 4.2.2.**

Donde X es la posición del pixel en el eje X, Y la posición del pixel en el eje Y, “a” es el valor que tiene el semieje X y “b” es el valor del semieje Y.

Despejando a Y para obtener su valor haciendo iteraciones con X, se tiene la siguiente función.

$Y = \pm ((1 - X^2 / a^2) * b^2)^{1/2}$

Sustituyendo valores $Y = \pm ((1- (X^2 / 100)) * 25)^{1/2}$

Para realizar las iteraciones, se debe tomar en cuenta que X no se desplaza desde cero, sino que cubre una distancia de –X a X debido a que es un semieje de la elipse, por lo que las iteraciones se hacen desde -10 hasta 10.

Haciendo las iteraciones en incrementos de 1 pixel de -10 a 10 se tiene la tabla 4.2.3.

Item	$Y = \pm ((1- (X^2 / 10^2)) * 5^2)^{1/2}$	Y		Item	$Y = \pm ((1- (X^2 / 100)) * 25)^{1/2}$	Y
X= -10	$Y = + ((1- (-10^2 / 100)) * 25)^{1/2}$	1		X= -10	$Y = - ((1- (-10^2 / 100)) * 25)^{1/2}$	-1
X= -9	$Y = + ((1- (-9^2 / 100)) * 25)^{1/2}$	2		X= -9	$Y = - ((1- (-9^2 / 100)) * 25)^{1/2}$	-2
X= -8	$Y = + ((1- (-8^2 / 100)) * 25)^{1/2}$	3		X= -8	$Y = - ((1- (-8^2 / 100)) * 25)^{1/2}$	-3
X= -7	$Y = + ((1- (-7^2 / 100)) * 25)^{1/2}$	4		X= -7	$Y = - ((1- (-7^2 / 100)) * 25)^{1/2}$	-4
X= -6	$Y = + ((1- (-6^2 / 100)) * 25)^{1/2}$	4		X= -6	$Y = - ((1- (-6^2 / 100)) * 25)^{1/2}$	-4
X= -5	$Y = + ((1- (-5^2 / 100)) * 25)^{1/2}$	4		X= -5	$Y = - ((1- (-5^2 / 100)) * 25)^{1/2}$	-4
X= -4	$Y = + ((1- (-4^2 / 100)) * 25)^{1/2}$	5		X= -4	$Y = - ((1- (-4^2 / 100)) * 25)^{1/2}$	-5
X= -3	$Y = + ((1- (-3^2 / 100)) * 25)^{1/2}$	5		X= -3	$Y = - ((1- (-3^2 / 100)) * 25)^{1/2}$	-5
X= -2	$Y = + ((1- (-2^2 / 100)) * 25)^{1/2}$	5		X= -2	$Y = - ((1- (-2^2 / 100)) * 25)^{1/2}$	-5
X= -1	$Y = + ((1- (-1^2 / 100)) * 25)^{1/2}$	5		X= -1	$Y = - ((1- (-1^2 / 100)) * 25)^{1/2}$	-5
X= 0	$Y = + ((1- (0^2 / 100)) * 25)^{1/2}$	5		X= 0	$Y = - ((1- (0^2 / 100)) * 25)^{1/2}$	-5
X= 1	$Y = + ((1- (1^2 / 100)) * 25)^{1/2}$	5		X= 1	$Y = - ((1- (1^2 / 100)) * 25)^{1/2}$	-5
X= 2	$Y = + ((1- (2^2 / 100)) * 25)^{1/2}$	5		X= 2	$Y = - ((1- (2^2 / 100)) * 25)^{1/2}$	-5
X= 3	$Y = + ((1- (3^2 / 100)) * 25)^{1/2}$	5		X= 3	$Y = - ((1- (3^2 / 100)) * 25)^{1/2}$	-5
X= 4	$Y = + ((1- (4^2 / 100)) * 25)^{1/2}$	5		X= 4	$Y = - ((1- (4^2 / 100)) * 25)^{1/2}$	-5
X= 5	$Y = + ((1- (5^2 / 100)) * 25)^{1/2}$	4		X= 5	$Y = - ((1- (5^2 / 100)) * 25)^{1/2}$	-4
X= 6	$Y = + ((1- (6^2 / 100)) * 25)^{1/2}$	4		X= 6	$Y = - ((1- (6^2 / 100)) * 25)^{1/2}$	-4
X= 7	$Y = + ((1- (7^2 / 100)) * 25)^{1/2}$	4		X= 7	$Y = - ((1- (7^2 / 100)) * 25)^{1/2}$	-4
X= 8	$Y = + ((1- (8^2 / 100)) * 25)^{1/2}$	3		X= 8	$Y = - ((1- (8^2 / 100)) * 25)^{1/2}$	-3
X= 9	$Y = + ((1- (-2^2 / 100)) * 25)^{1/2}$	2		X= 9	$Y = - ((1- (-2^2 / 100)) * 25)^{1/2}$	-2
X= 10	$Y = + ((1- (-1^2 / 100)) * 25)^{1/2}$	0		X= 10	$Y = - ((1- (-1^2 / 100)) * 25)^{1/2}$	-0

La tabla 4.2.3 muestra el resultado de las iteraciones en X con la función Y.

En la tabla 4.2.3, se realizan las iteraciones desplazando el valor de X de -10 a 10 con radio en X = 10 y radio en Y = 5. Del lado derecho de la tabla se muestran los valores positivos de la raíz y del otro lado los valores negativos.

De esta tabla, al extraer los valores y colocarlos los pixeles en las vectores obtenidos, se consigue la imagen de la figura 4.2.5.

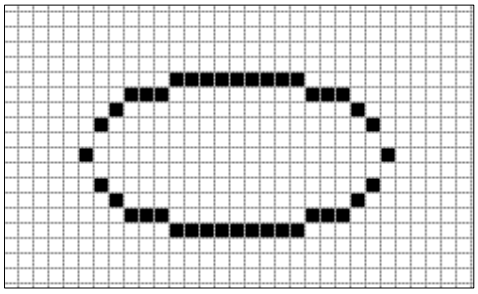


Figura 4.2.5. Pixeles formados al usar la ecuación de la elipse con iteraciones en X.

Como puede observarse, la figura 4.2.5, muestra parte de lo que corresponde a una elipse, pero no esta completamente cerrada la trayectoria, esto se debe a que los desplazamientos en incrementos de 1 no alcanzan a cubrir todos los valores que cierren la trayectoria, para lograr cerrar la trayectoria, tendrían que obtenerse los valores de las posiciones con respecto a X y superponerlos en la imagen para tener todos los puntos que componen la circunferencia de la elipse.

Si se realiza el mismo procedimiento para encontrar los puntos de la elipse, pero haciendo las iteraciones con desplazamientos en Y y la función con respecto a X, se obtienen las posiciones de los pixeles respecto a X como lo muestra la figura 4.2.6.

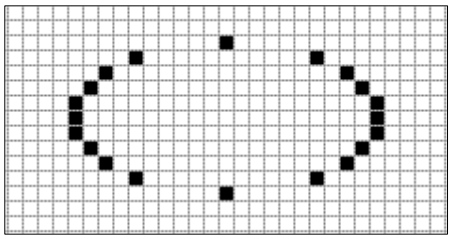


Figura 4.2.6. Pixeles formados al usar la ecuación de la elipse con iteraciones en Y.

La figura 4.2.6, muestra la posición de los pixeles que se obtiene cuando las iteraciones se hacen respecto a Y. En esta figura se puede observar que aparecen los pixeles que no aparecen en la iteración con respecto a X, de lo cual se puede deducir que si se superponen ambas imágenes por medio de una suma (OR), se obtendrá la trayectoria completa de la elipse.

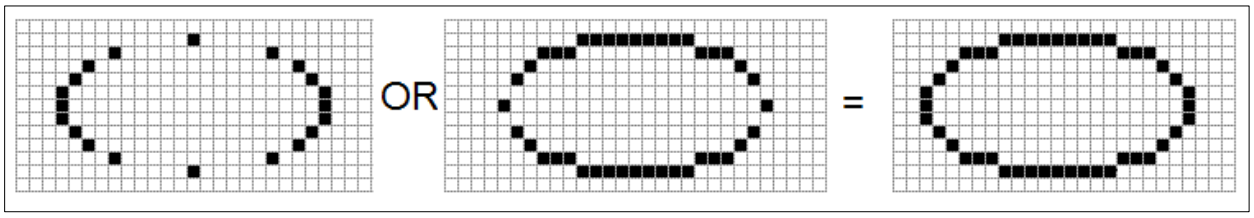


Figura 4.2.7. Superposición de pixeles para obtener la trayectoria completa de la elipse.

La figura 4.2.7, muestra una suma OR entre imágenes, en la que los pixeles que no tiene una imagen son cubiertos por la otra para obtener de esa manera la trayectoria cerrada de la elipse.

Analizando este método, se deduce que no cubre las necesidades, pues a pesar de que la imagen a sido redibujada, el ordenamiento de pixeles no se a demostrado, ya que para formar la elipse cerrada se requieren dos imágenes, las cuales se tratan por separado, lo que provoca que el ordenamiento de los vectores que forma la imagen tenga que hacerse de nuevo.

Otro método en el que se ha pensado, esta basado en las funciones de senos y cosenos, debido a que estas funciones generan curvas cerradas cuando se aplican a 360°, con lo que se obtiene las coordenadas en X e Y si se desplaza el ángulo de una función. Las funciones de seno y coseno se muestran en la ecuación 4.2.3.

$$v(x, y) = (a \text{ seno } (\theta) \ , \ b \text{ coseno } (\theta)) \qquad \textbf{Ecuación 4.2.3.}$$

De la ecuación 4.2.3, v(x, y) representa la posición de los vectores ordenados, “a” representa el valor del semieje X, “b” el valor del semieje Y y θ el ángulo de 0° a 360° que cierra la trayectoria del elipse, donde el θ es el elemento al que se le aplican las iteraciones en incrementos definidos por los radios de la elipse, que van de 0° a 360°.

Por ejemplo, si se toman los valores que se han estado manejando anteriormente.

a = 10 (eje x). b = 5 (eje y).

Y se consideran incrementos de 45° ($\theta = \theta + 45°$) a la función de senos y cosenos hasta cerrar la trayectoria llegando a 360°, se obtienen los puntos de la tabla 4.2.4.

κ, θ	$X_k = a * Seno(\theta)$	$Y_k = b * Coseno(\theta)$	Vectores ordenados $V_k = (X_k, Y_k)$
1, 0°	$X_1 = 10 * Seno(0°)$	$Y_1 = 5 * Coseno(0°)$	$V_1 = (0, 5)$
2, 45°	$X_2 = 10 * Seno(45°)$	$Y_2 = 5 * Coseno(45°)$	$V_2 = (7, 4)$
3, 90°	$X_3 = 10 * Seno(90°)$	$Y_3 = 5 * Coseno(90°)$	$V_3 = (10, 0)$
4, 135°	$X_4 = 10 * Seno(135°)$	$Y_4 = 5 * Coseno(135°)$	$V_4 = (7, -4)$
5, 180°	$X_5 = 10 * Seno(180°)$	$Y_5 = 5 * Coseno(180°)$	$V_5 = (0, -5)$
6, 225°	$X_6 = 10 * Seno(225°)$	$Y_6 = 5 * Coseno(225°)$	$V_6 = (-7,-4)$
7, 270°	$X_7 = 10 * Seno(270°)$	$Y_7 = 5 * Coseno(270°)$	$V_7 = (-10,0)$
8, 315°	$X_8 = 10 * Seno(315°)$	$Y_8 = 5 * Coseno(315°)$	$V_8 = (-7, 4)$

Tabla 4.2.4. Vectores obtenidos con incrementos de 45°

La tabla 4.2.4, muestra los vectores que se obtienen si se usa la función de senos y cosenos con los valores de “a” (semieje X) y “b” (semieje Y) haciendo incrementos de 45°.

Si se colocan los pixeles usando los vectores de la tabla 4.2.4, se obtiene la imagen de la figura 4.2.8.

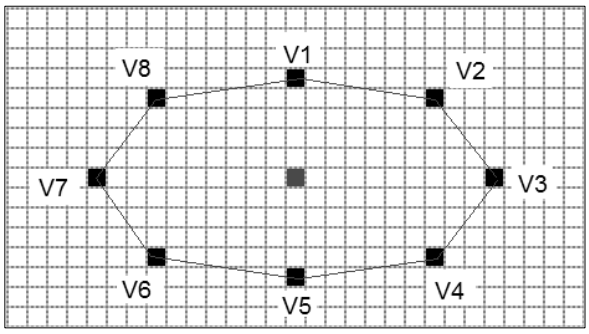


Figura 4.2.8. Puntos obtenidos por la función del seno y coseno con incrementos de 45°.

La figura 4.2.8, muestra la posición de los pixeles que se obtuvieron cuando se utilizaron incrementos de 45° en sus respectivas amplitudes o ancho del eje. Como se puede notar en esta figura, al hacer las iteraciones con incrementos de 45°, las posiciones se van colocando de manera ordenada, que como lo describe la figura, se mueven los vectores en el sentido de las manecillas del reloj.

Ahora lo único que falta es cerrar la trayectoria. Para lograrlo se debe reducir el tamaño de los incrementos, pero hay que considerar que si se reduce demasiado un incremento, se producirán demasiados vectores y algunos se repetirán. Cuando esto sucede es recomendable no almacenar los vectores que se repiten y establecer un número de incrementos que alcance a cubrir todos los puntos. Por ejemplo si se establecen incrementos de 1°, se necesitaran 360 iteraciones para cerrar la trayectoria de la elipse y se

estarán creando 360 vectores, pero si se observa la imagen de la figura 4.2.8, a simple vista no requiere mas de 60 pixeles para cerrar la trayectoria, por lo que los incrementos de 1° no son necesarios, pero si la elipse tiene ejes más grandes, por ejemplo de 100x50, se requieren de al menos 400 pixeles para cerrar la trayectoria, por lo que se los incrementos deben ser más pequeños.

Una manera de determinar un valor para el incremento, que no genere una cantidad exagerada de iteraciones y que a la vez cubra todos los pixeles necesarios para cerrar la trayectoria de la elipse, consiste en calcular el tamaño de los semiejes de la elipse, de tal manera que las iteraciones sean 4 veces el tamaño de la suma de los 2 semiejes, así, si un semieje mide 100 pixeles y el segundo 50 pixeles, las iteraciones deberán ser (50+100)x4 = 600, con lo que los incrementos del ángulo deberán ser de 360/600 = 0.6° para producir la cantidad de iteraciones establecida.

Usando esta información para el ejemplo donde se tiene un semieje a = 10 y semieje b = 5, se obtiene que el número de iteraciones máximo es: (10+5) * 4 = 60 y los incrementos deben ser de 360°/60 = 6°. Tomando el valor de este incremento y aplicando las iteraciones a la funciones de senos y cosenos se obtiene la tabla 4.2.5.

θ	Vector *		θ	Vector *		θ	Vector *		θ	Vector *
0°	(0, 5)		96°	(10, -1)		192°	(-2, -5)		288°	(-9, 2)
6°	(1, 5)		102°	(10, -1)		198°	(-3, -5)		294°	(-9, 2)
12°	(2, 5)		108°	(9, -2)		204°	(-4, -5)		300°	(-8, 3)
18°	(3, 5)		114°	(9, -2)		210°	(-5, -4)		306°	(-8, 3)
24°	(4, 5)		120°	(8, -3)		216°	(-5, -4)		312°	(-7, 3)
30°	(5, 4)		126°	(8, -3)		222°	(-6, -4)		318°	(-6, 4)
36°	(5, 4)		132°	(7, -3)		228°	(-7, -3)		324°	(-5, 4)
42°	(6, 4)		138°	(6, -4)		234°	(-8, -3)		330°	(-5, 4)
48°	(7, 3)		144°	(5, -4)		240°	(-8, -3)		336°	(-4, 5)
54°	(8, 3)		150°	(5, -4)		246°	(-9, -2)		342°	(-3, 5)
60°	(8, 3)		156°	(4, -5)		252°	(-9, -2)		348°	(-2, 5)
66°	(9, 2)		162°	(3, -5)		258°	(-10, -1)		354°	(-1, 5)
72°	(9, 2)		168°	(2, -5)		264°	(-10,-1)			
78°	(10, 1)		174°	(1, -5)		270°	(-10, 0)			
84°	(10, 1)		180°	(0, -5)		276°	(-10,1)			
90°	(10, 0)		186°	(-1, -5)		282°	(-10,1)			

***Se obtiene el vector por medio de**: X= 10 * Seno(incremento), Y = 5 * Coseno(incremento).

Tabla 4.2.5. Obtención de los vectores que componen la trayectoria de una elipse.

La tabla 4.2.5, muestra los vectores que representan las posiciones de los pixeles para formar la elipse, donde cada uno de los vectores se encuentra ordenado, de tal manera que si se recorre la matriz de vectores, se formara la elipse en un trayectoria que va en sentido de las manecilla del reloj, empezando del punto (0,5) y terminando en (-1,5). En la tabla se observan valores que se repiten, solo debe almacenarse 1 y los demás deben ser descartados. La imagen que representa los valores de la tabla 4.2.5, se muestra en la figura 4.2.9.

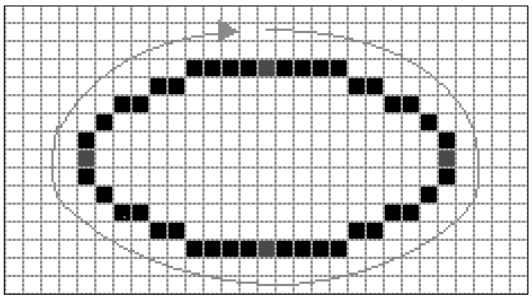


Figura 4.2.9. Elipse formada por vectores ordenados en una matriz.

La figura 4.2.9, es una representación de la elipse, pero dibujada siguiendo el patrón de vectores ordenados que se muestran en la tabla 4.2.5, de tal manera que la flecha a su alrededor indica la dirección que llevan los vectores para formar la imagen, consiguiendo de esta manera convertir un figura en vectores de desplazamiento para ser usados en la simulación o transformados en pulsos eléctricos y ser enviados por el puerto paralelo.

4.2.4 Trazador libre.

La herramienta de trazador libre, permite crear vectores a partir de cualquier trazo realizado por el usuario cuando mueve el cursor a través de área de diseño de la interfaz gráfica.

Para hacer funcionar esta herramienta se requieren los siguientes pasos:

1. Seleccionar la herramienta (se activa el uso de la función dibujar).

2. Conocer la posición del cursor dentro del área de diseño cuando se esta presionando el botón izquierdo del mouse y se esta moviendo el cursor.

Cuando se usa esta herramienta, en el momento de oprimir el botón del mouse en el área de diseño, se genera un pixel en la posición del cursor, mismo que se almacena en la matriz de vectores ordenados, cuando el cursor se mueve, se registra un nuevo punto que corresponde a la coordenada en la que se encuentra ahora el cursor, de tal manera que mientras se mueva el cursor y se mantenga apretado el botón del cursor, se estarán registrando en una matriz los puntos por donde ha pasado el cursor.

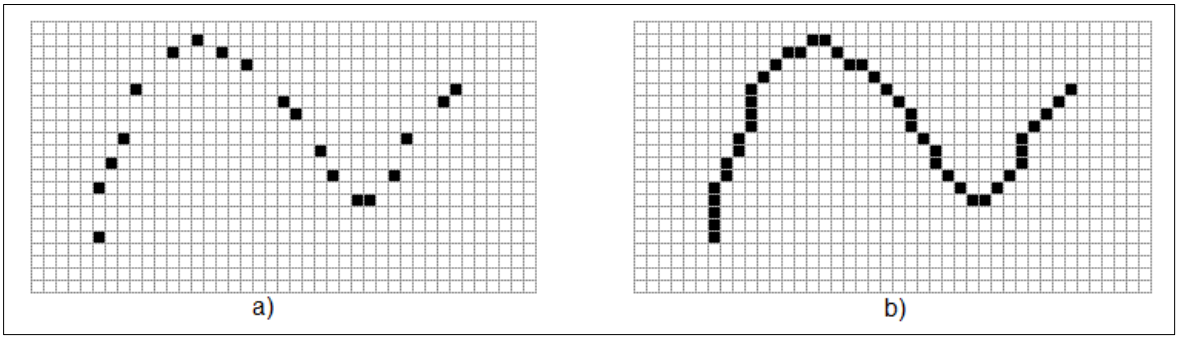


Figura 4.2.10. Trayectoria realizada con la herramienta trazos libres.

La figura 4.2.10, muestra un ejemplo de un trazo que se puede hacer con la herramienta de trazos libres. El inciso a) de la figura 4.2.10, representa solo los puntos que fueron almacenados en la matriz debido a que muchos puntos por los que pasa el cursor no llegan a ser almacenados por falta de velocidad en el proceso de almacenamiento. El inciso b) muestra la trayectoria que se forma con la herramienta de trazo libre, donde por medio de líneas unidas, se cierran los puntos para obtener un solo trazo.

Hasta el momento los puntos almacenados en la matriz, no representan la información necesaria para redibujar la imagen, pues si se envía esta información, solo se estaría redibujando por medio de vectores la imagen de la figura 4.2.10 inciso a). Para obtener todos los puntos que forman el trazo, se utilizan los vectores almacenados hasta el momento, usando la idea de unirlos de manera consecutiva con líneas rectas.

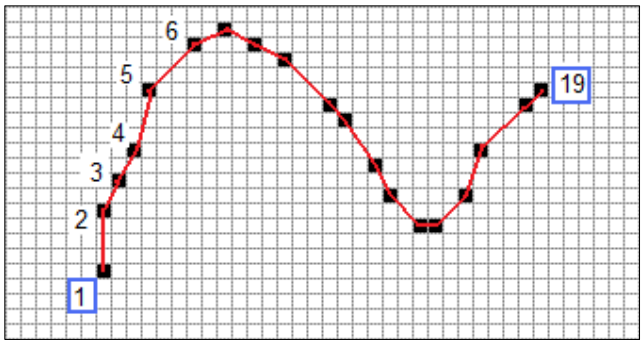


Figura 4.2.11. Formación de una curva por medio de la unión de vectores con líneas rectas.

La figura 4.2.11, muestra de que manera se puede redibujar el trazo de la figura 4.2.10, inciso b), basándose en sus puntos almacenados. Esta figura muestra una secuencia ordenada en la que se almacenaron los vectores, de esta manera, si se toman los dos primeros puntos almacenados y se usa el algoritmo para obtener los puntos de una recta (4.2.1 trazador de líneas), se obtiene los vectores ordenados entre estos dos puntos y si se realiza el mismo procedimiento llevando la secuencia de puntos almacenada inicialmente con el cursor, se obtienen al final todos los vectores necesarios para formar la trayectoria de la figura 4.2.11 de manera ordenada.

4.2.5 Trazador de texto.

El trazador de texto es una herramienta que sirve para escribir palabras en forma de vectores para ser simuladas y enviadas al puerto de salida del ordenador.

Para hacer funcionar esta herramienta se siguen los siguientes pasos:

- Se selecciona la herramienta.
- Se escribe el texto deseado.

Esta herramienta no recurre al procesamiento de imagen, pero si a la identificación de archivos de vectores que se encuentran almacenadas en la aplicación y que serán comparadas con cada carácter que escriba el usuario.

Por ejemplo, al momento de seleccionar la herramienta texto, el programa espera a que se oprima una letra del teclado, cuando se oprime, el programa busca el archivo correspondiente que contiene la matriz de vectores ordenados que genera ese carácter.

Finalmente, si se colocan las herramientas de dibujo que generan vectores, creadas hasta el momento, en un cuadro que las contenga, se obtiene una paleta de herramientas como la que se muestra en la figura 4.2.12.

Figura 4.2.12. Paleta de herramientas para el diseño de figuras.

La figura 4.2.12, muestra una representación gráfica de las herramientas para dibujar vectores, descritas en este capitulo.

En el apéndice B parte II se muestran las partes de código para generar estas herramientas.

4.3. Guardar y abrir vectores.

Es importante tener la posibilidad de guardar los resultados obtenidos en la vectorización y de igual manera poder abrir los archivos guardados, por lo que se ha desarrollado en este subtema las herramientas para guardar y abrir archivos de vectores.

4.3.1 Guardar Vectores.

Una vez que se ha logrado un diseño por medio de la vectorización y se desea guardar en el disco duro ó algún otro medio de almacenamiento, es necesaria una función que pueda hacerlo, además de utilizar un formato especifico para identificar el archivo en el ordenador.

C# tiene la facilidad de guardar archivos por medio de una clase llamada SaveFileDialog, con esta clase se puede abrir una ventana de dialogo y guardar el archivo en la ubicación que desee, además de poder definir la extensión del archivo, que en la interfaz gráfica ha sido definida como *.vtx, ya que es un archivo de texto con contenido de vectores.

Para guardar los archivos de vectores, se ha elegido guardar los vectores en forma de cadena de caracteres, pero con un orden y coherencia para determinar en algún momento que dato pertenece a la posición X y que dato a la posición Y. Para lograr esto se hizo lo siguiente:

Cada vector de la imagen esta incluido dentro de dos parentesis “(“ , “)”.

Los ejes X-Y están separados por una coma “,”.

De esta manera cuando se lea un carácter “(“, indica que el siguiente carácter deber ser numérico y pertenece al eje x, hasta que encuentra un carácter “,” que indicará que la siguiente línea es numérica y pertenece al eje Y, hasta que encuentre un “)”, de esta manera se define una coordenada, se almacena en una matriz de vectores y se prosigue a leer la siguiente secuencia de caracteres.

Siguiendo estos pasos, se puede generar un archivo *vtx como muestra en la figura 4.3.1.

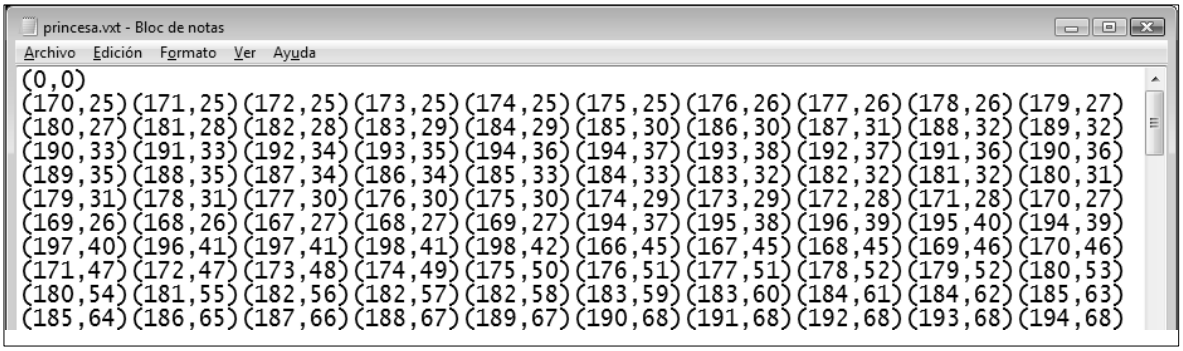


Figura 4.3.1. Archivo de Texto con extensión VTX para guardar vectores ordenados.

La figura 4.3.1, muestra un archivo de texto que contiene información de vectores ordenados, en el cual se ha agrupado la información de tal manara que pueda ser leída por medio de algún algoritmo que interprete la secuencia que se muestra.

En el apéndice B parte III se incluye el fragmento de código que genera esta herramienta en C#.

4.3.2 Abrir Vectores.

Si se pueden guardar vectores, es necesario que se puedan abrir en un determinado momento. Esto es lo mismo que se hizo para abrir una imagen, pero en esta ocasión se trata de leer un archivo de texto que contiene los vectores en forma de cadenas de caracteres, con un orden determinado que puede ser interpretado de manera sencilla.

Para agrupar los vectores y detectar que caracteres pertenecen al eje X y que caracteres al eje Y, se realiza un escaneo del archivo, siguiendo el siguiente procedimiento:

Primero se obtiene el número de líneas o párrafos que tiene el archivo, después se escanea línea por línea en busca de un carácter “(“, cuando se encuentre uno, se recorre al siguiente carácter, que se ha determinado previamente será un carácter numérico de acuerdo a la forma en que se codifico, pero el siguiente carácter se desconoce, por lo que debe ser comparado, si resulta ser un carácter numérico, se concatena con el anterior, si no lo es, debe tratarse de una “,” la cual determina la separación entre la coordenada X y la coordenada Y, por lo que el siguiente carácter debe ser numérico correspondiente al eje Y, después de este carácter se compara el que le prosigue y si resulta ser carácter numérico se concatena con el valor del carácter anterior de lo contrario, debe tratarse de un carácter “)”, que determina que las coordenadas de ese vector terminan en este punto y los siguiente caracteres a analizar corresponden a un nuevo vector.

Este procedimiento es sencillo de aplicar y lo que resulta al final del proceso es una matriz que contiene n elementos de vectores con las coordenadas X-Y que forman la imagen o figura que se guardo después de la sectorización. Los valores que proporciona la codificación de los archivos vtx, son utilizados para realizar la simulación, de tal manera que la información mostrada en la simulación pueda ser enviada a través del puerto paralelo.

En el apéndice B parte IV se encuentra el fragmento de código que sirve para leer un archivo de texto con contenido de vectores y reconocer la información.

En el capitulo 5 se hace uso de los vectores ordenados que se han tratado en este capitulo, para poner en marcha la simulación y el proceso de control numérico.

Capitulo 5

Simulador:
Prototipo Virtual CNC

5. SIMULADOR VIRTUAL

El simulador virtual consiste en una máquina virtual de 3 ejes capaz de producir los mismos movimientos que produciría una máquina real, ya que los datos que procesa son los mismos que serán enviados al dispositivo exterior. La simulación verifica los resultados que se pueden tener cuando se lleve a cabo el proceso de decoración.

Para su funcionamiento, hace uso de la información adquirida en la vectorización, o de archivos que contienen información de vectores que se encuentran en una carpeta de plantillas con la extensión vtx. Estos archivos presentan un formato como el que se mostro en el capitulo 4, figura 4.3.1.

5.1. Adquisición de Control Numérico.

Los archivos vtx contienen información de vectores, más no información que pueda procesar una máquina comercial, que por lo general basa la información en pulsos determinados por código binario. Pero con los vectores obtenidos es fácil adquirir la información binaria.

Para obtener la información binaria primero se abre un archivo de vectores.

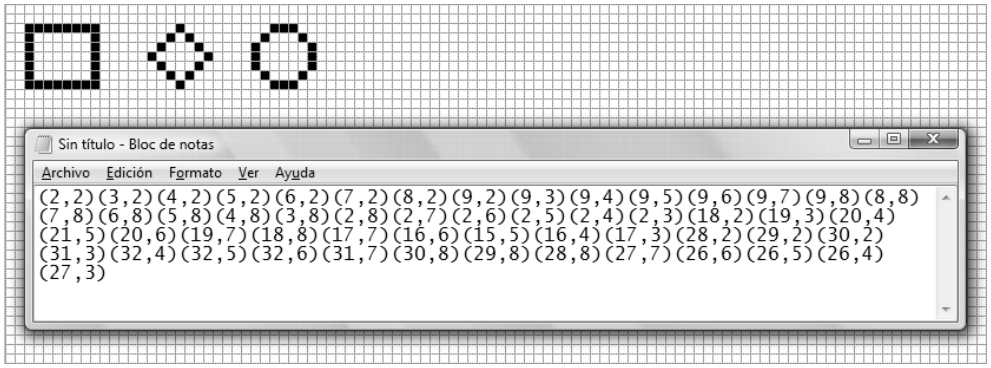


Figura 5.1.1. Archivo de vectores que representa las figuras mostradas.

De acuerdo a la figura 5.1.1, se observa un archivo en un block de notas que contiene la información que representa las coordenadas necesarias para formar las figuras geométricas mostradas en la parte superior. El procedimiento para obtener el código binario, solo consiste en restar el elemento antecesor al elemento sucesor de la lista de coordenadas y el resultado compararlo con *etiquetas de funcionalidad (cap. 5.2)*.

Por ejemplo:

1 (3, 2) – (2, 2) = (1, 0)
2 (4, 2) – (3, 2) = (1, 0)

Estas operaciones se muestran en la tabla 5.1.

1	(3,2) – (2,2)	=(1,0)		21	(2,7) – (2,8)	=(0,-1)
2	(4,2) – (3,2)	=(1,0)		22	(2,6) – (2,7)	=(0,-1)
3	(5,2) – (4,2)	=(1,0)		23	(2,5) – (2,6)	=(0,-1)
4	(6,2) – (5,2)	=(1,0)		24	(2,4) – (2,5)	=(0,-1)
5	(7,2) – (6,2)	=(1,0)		25	(2,3) – (2,4)	=(0,-1)
6	(8,2) – (7,2)	=(1,0)		26	(18,2) – (2,3)	=(16,1)
7	(9,2) – (8,2)	=(1,0)		27	(19,3) – (18,2)	=(1,1)
8	(9,3) – (9,2)	=(0,1)		28	(20,4) – (19,3)	=(1,1)
9	(9,4) – (9,3)	=(0,1)		29	(21,5) – (20,4)	=(1,1)
10	(9,5) – (9,4)	=(0,1)		30	(20,6) – (21,5)	=(-1,1)
11	(9,6) – (9,5)	=(0,1)		31	(19,7) – (20,6)	=(-1,1)
12	(9,7) – (9,6)	=(0,1)		32	(18,8) – (19,7)	=(-1,1)
13	(9,8) – (9,7)	=(0,1)		33	(17,7) – (18,8)	=(-1,-1)
14	(8,8) – (9,8)	=(-1,0)		34	(16,6) – (17,7)	=(-1,-1)
15	(7,8) – (8,8)	=(-1,0)		35	(15,5) – (16,6)	=(-1,-1)
16	(6,8) – (7,8)	=(-1,0)		36	(16,4) – (15,5)	=(1,-1)
17	(5,8) – (6,8)	=(-1,0)		37	(17,3) – (16,4)	=(1,-1)
18	(4,8) – (5,8)	=(-1,0)		38	(28,2) – (17,3)	=(11,-1)
19	(3,8) – (4,8)	=(-1,0)		39	(28,3) – (28,2)	=(0,1)
20	(2,8) – (3,8)	=(-1,0)		40	(28,4) – (28,3)	=(0,1)

Tabla 5.1. Movimiento de los vectores

La tabla 5.1, muestra por medio de la resta por iteración de coordenadas como se mueve el vector para trazar la imagen. La interpretación de la tabla 5.1 es la siguiente:

Los valores que se restan son las comparaciones a etiquetar.

Los vectores con el signo " = " son los resultados de las restas.

Las filas señaladas en 2 colores indican que hubo un cambio de desplazamiento del vector, por ejemplo si el vector se estuvo moviendo con saltos de (1, 0) y cambia su desplazamiento en (0, 1), (1, 1), (1, -1), (-1, 1), (-1, -1), (-1,0), (0, -1).

Cuando la fila esta señalada por un solo color indica que se requiere un desplazamiento mayor a los anteriores para llegar al otro vector, por lo que se tiene que interrumpir la impresión y solo desplazar el punto.

En el ejemplo de la figura 5.1.2, se tiene que (18,2) y (2,3) son puntos aislados, por lo que el cabezal de máquina debe desplazarse desde (2,3) al punto (18,2) sin imprimir la trayectoria que se forma entre estos dos puntos, como lo indica la línea recta en color verde.

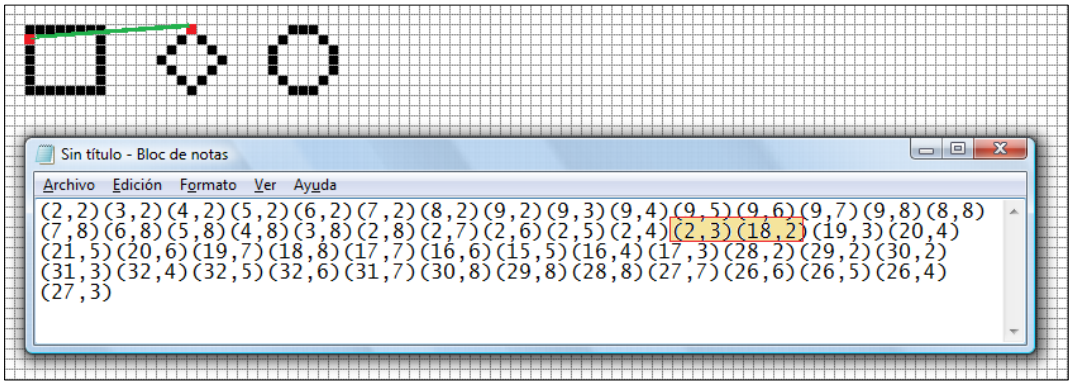


Figura 5.1.2. Trayectoria que se forma entre dos puntos aislados que no pertenece a la imagen.

La figura 5.1.2 muestra un archivo de texto que representa los vectores que forman las imágenes geométricas de la parte superior. Los puntos marcados en la imagen pertenecen a la trayectoria que debe recorrerse sin inyectar materia prima sobre el pastel. Para que esto ocurra, se deben establecer puntos de control que simulará la máquina. Los puntos de control son los que determina si se desplaza en X, en Y o al mismo tiempo, imprimiendo o solo desplazándose. Estos puntos de control son los códigos binarios que serán insertados a la máquina real, y por esto la máquina virtual debe de interpretarlos de la misma manera.

5.2. Códigos Binarios.

Los códigos binarios son etiquetas de funcionalidad, están hechas para simular el comportamiento de una máquina de 3 ejes (desplazarse en X, -X, Y, -Y, Z, -Z). Los códigos binarios se han establecido de la siguiente manera:

Si (**X** = 0, **Y** = 0, **Z** = 0) → Desplazar a la Derecha. Código Binario = (10000000).
Máquina activada.

Si (**X** = 1, **Y** = 0, **Z** = 0) → Desplazar a la Derecha. Código Binario = (10000001).
Desplazamiento en X.

Si (**X** = 0, **Y** = 1, **Z** = 0) → Desplazar a la Derecha. Código Binario = (10000100). 
Desplazamiento en Y.

Si (**X** = 0, **Y** = 0, **Z** = 1) → Desplazar a la Derecha. Código Binario = (10010000). 
Desplazamiento en Z.

Si (**X** = -1, **Y** = 0, **Z** = 0) → Desplazar a la Derecha. Código Binario = (10000010). 
Desplazamiento en -X.

Si (**X** = 0, **Y** = -1, **Z** = 0) → Desplazar a la Derecha. Código Binario = (10001000). 
Desplazamiento en -Y.

Si (**X** = 0, **Y** = 0, **Z** = -1) → Desplazar a la Derecha. Código Binario = (10100000). 
Desplazamiento en -Z.

Si (**X** = 1, **Y** = 1, **Z** = 0) → Desplazar a la Derecha. Código Binario = (10000101). 
Desplazamiento en X, Y.

Si (**X** = 1, **Y** = 0, **Z** = 1) → Desplazar a la Derecha. Código Binario = (10010001). 
Desplazamiento en X, Z.

Si (**X** = 0, **Y** =1, **Z** = 1) → Desplazar a la Derecha. Código Binario = (10010100). 
Desplazamiento en Y, Z.

Si (**X** = -1, **Y** = 1, **Z** = 0) → Desplazar a la Derecha. Código Binario = (10000110). 
Desplazamiento en -X, Y.

Si (**X** = -1, **Y** = 0, **Z** = 1) → Desplazar a la Derecha. Código Binario = (10010010). 
Desplazamiento en -X, Z.

Si (**X** = 0, **Y** = -1, **Z** = 1) → Desplazar a la Derecha. Código Binario = (10011000).
Desplazamiento en -Y, Z.

Si (**X** = -1, **Y** = 1, **Z** = 0) → Desplazar a la Derecha. Código Binario = (10001001). 
Desplazamiento en X, -Y.

Si (**X** = -1, **Y** = 1, **Z** = 0) → Desplazar a la Derecha. Código Binario = (10001010). 
Desplazamiento en -X, -Y.

Si (**X** = 1, **Y** = 1, **Z** = 1) → Desplazar a la Derecha. Código Binario = (10010101).
Desplazamiento en X, Y, Z.

Si (**X** = -1, **Y** = 1, **Z** = 1) → Desplazar a la Derecha. Código Binario = (10010110). 
Desplazamiento en -X, Y, Z.

Si (**X** = 1, **Y** = -1, **Z** = 1) → Desplazar a la Derecha. Código Binario = (10011001). 
Desplazamiento en X, -Y, Z.

Si (**X** = -1, **Y** = -1, **Z** = 1) → Desplazar a la Derecha. Código Binario = (10011010).
Desplazamiento en -X, -Y, Z.

(Nota: No hay desplazamientos en –Z cuando se desplaza en X, en Y ó ambas.

Estos códigos se introducen en la aplicación para asignarles una función a los datos que se obtuvieron a partir de la adquisición de control numérico (subtema 5.1).

FUNCIONES DE DESPLAZAMIENTO			
Binario	Hexadecimal	Decimal	Funcionamiento
10000000	80	128	Activado
10000001	81	129	X
10000100	84	132	Y
10010000	90	144	Z
10000010	82	130	-X
10001000	88	136	-Y
10100000	A0	160	-Z
10000101	85	133	X,Y
10010001	91	145	X,Z
10010100	94	148	y,Z
10000110	86	134	-X,Y
10010010	92	146	-X,Z
10011000	98	152	-Y,Z
10001001	89	137	X,-Y
10001010	8A	138	-X,-Y
10010101	95	149	-X, Y,Z
10010110	96	150	X, Y,Z
10011001	99	153	X,-Y,Z
10011010	9C182	154	-X,-Y,Z

Tabla 5.2. Instrucciones de código binario.

La tabla 5.2, muestra una lista de etiquetas de funcionalidad que pueden estar representadas por números binarios, decimales y hexadecimales.

Con las etiquetas que presenta la tabla 5.2, se puede decir que si se tiene que mover el motor X a la derecha inyectando materia prima se debe enviar al puerto de salida la activación de X, Z que corresponde al valor 145 en decimal, equivalente a 10010001 en código binario.

El código fuente en C# para generar esta codificación, se encuentra en el apéndice C parte I.

5.3. Diseño del simulador y las partes móviles.

El diseño del simulador es la apariencia que tiene la máquina en la pantalla, de tal manera que se asemeje a una máquina real o al sistema de decoración real.

La siguiente imagen muestra las partes esenciales que componen al decorador virtual.

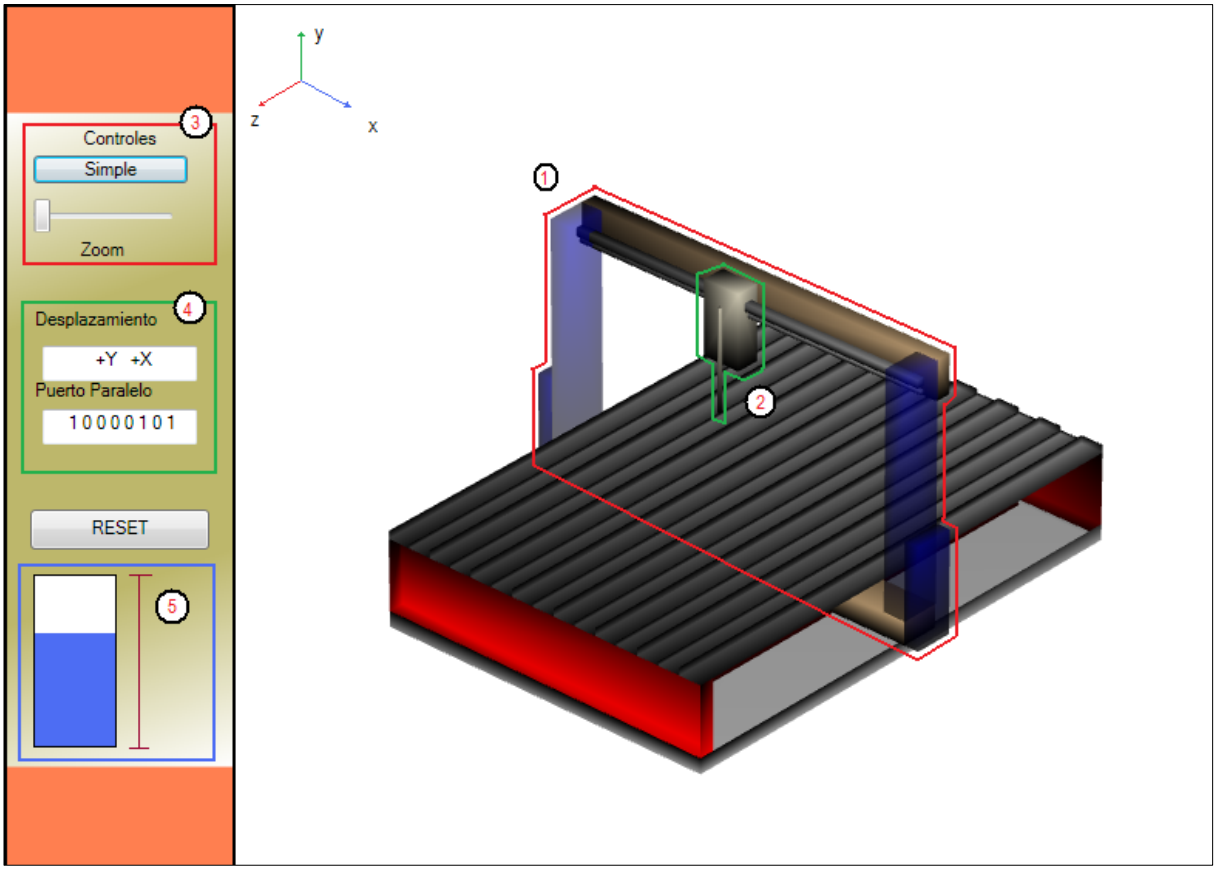


Figura 5.3.1. Partes esenciales que componen al decorador virtual de pasteles.

La figura 5.3.1 representa una vista de la pantalla donde se muestra el prototipo virtual decorador de tortas de pastel, en la imagen se pude notar algunos objetos numerados, los cuales son las partes esenciales de la simulación.

Para los puntos numerados, se tiene que:

(1) Pertenece al **brazo** principal del decorador virtual y se muestra indicado en un contorno rojo en la figura. Esta pieza del simulador se mueve a lo largo del eje Z en las coordenadas del sistema 3D, pero su movimiento real o basado en imagen es a lo largo del eje X.

(2) Pertenece al **cabezal** que soporta la duya o cono que permite la salida del fluido para decorar el pastel. Este objeto virtual se mueve a lo largo de X en el sistema de coordenadas 3D, pero en coordenadas reales y de imagen se mueve a los largo de Y. Este objeto esta fijo en el brazo por lo que los movimiento del brazo afectaran a esta pieza.

(3) Pertenece al **control** de simulación. Este objeto puede controlar el inicio del proceso de simulación, parar la simulación en un punto y mostrar por pasos el proceso o detener completamente el proceso para iniciar de nuevo.

(4) Pertenece al cuadro de información que muestra los **datos de salida**. Esta imagen proporcionará información de los datos que se están leyendo para producir los movimientos en la máquina virtual, los cuales son los mismos que salen por el puerto de datos.

(5) Pertenece a la **inyectora**. Este objeto representa el desplazamiento del eje Y en coordenadas 3D, que seria análogo al desplazamiento en Z en coordenadas reales. La imagen muestra un nivel de materia en un contenedor, el cual se ira reduciendo en color azul conforme se vaya imprimiendo o trazando la imagen en el simulador.

5.4. Construcción del simulador usando WPF.

Para realizar el simulador virtual, se recurrió al sistema de presentación de gráficos de WPF (por sus siglas en ingles “Windows Presentation Foundation”) que consiste en una de las novedosas tecnologías de Microsoft, con la potencia para desarrollar interfaces usando una arquitectura *Modelo Vista Controlador,* el cual es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. Permite la aplicación de gráficos 3D y separa su lenguaje de compilación en su lenguaje nativo que es el XALM y los lenguajes de programación de .NET aumentando las posibilidades de programación e interacción.

Debido a la interacción que tiene WPF con .NET, fue posible crear la aplicación basándose en código C#.

5.4.1. Integrando WPF a C#.

En el capitulo 3 se mostro como crear una ventana WindowsForm, para poder visualizar un control de WPF es necesario crear una nueva ventana WindowsForm. Una vez creada una ventana Form, se agrega la referencia de librería “System Integration” al programa como se muestra la figura 5.4.1.

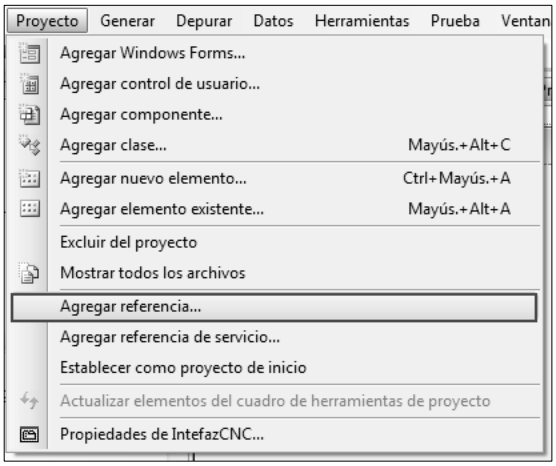


Figura 5.4.1. Como agregar una referencia o librería de programación en Visual C# 2008.

La figura 5.4.1 muestra los elementos que despliega la pestaña “Proyectos” del compilador de Visual C# 2008 y en rojo la parte que interesa de estos elementos.

Cuando se ha presionado “Agregar referencia” se obtiene un cuadro con 5 pestañas, en la pestaña “.Net” se encuentra al final un componente que se llama “WindowsFormsIntegration”, de esta manera se acepta y se integra el componente a la aplicación en desarrollo.

Después, en el cuadro de herramientas del compilador, se arrastra el elemento “ElementHost” a la ventana Form creada hasta ahora.

Figura 5.4.2. Componente para visualizar WPF en un Windows Form.

Después de arrastrar el elemento de la figura 5.4.2 a la ventana Form creada, se agrega un Control de Usuario (WPF) al proyecto.

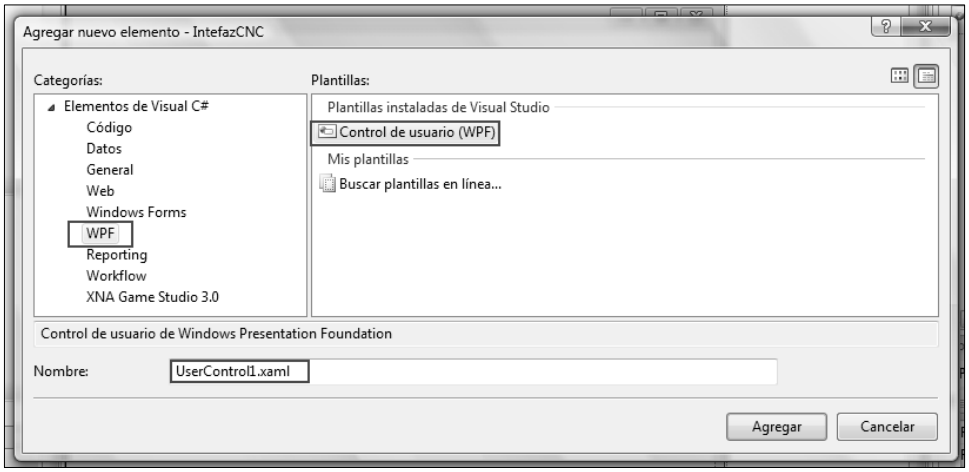


Figura 5.4.3. Agregando un control de usuario WPF al proyecto.

La figura 5.4.3, muestra como agregar un control de usuario WPF al proyecto, en el cual se señala en un recuadro las partes que deben estar seleccionadas, la parte del nombre es importante tenerla en cuenta, por que esta nos permite el llamado al control de usuario que se esta creando.

Una vez creado el control de usuario "UserControl1.xalm", es posible agregarlo al "elementHost1" añadido anteriormente al control de usuario. Para lograr esto solo se aplica el siguiente código de programa en el constructor de la clase que se autocreo cuando se inserto la nueva ventana Form.

```
public WPF_ventana()
    {
       InitializeComponent();
       UserControl1 wpfctl = new UserControl1 ();
       elementHost1.Child = wpfctl;
       this.Controls.Add(elementHost1);
     }
```

En este código, UserControl1 pertenece a la clase que se creo cuando se agrego el control de usuario WPF, al cual se le asigna un nombre de objeto "wpfctl". "elementHost1 es el elemento que se añade para visualizar el WPF al cual se le esta introduciendo el objeto "wpfctl".

Siguiendo este codigo, es posible visualizar lo que se desarrolle en el control de usuario desde la ventana "windows form" y solo para mandar a llamar la ventana "Form" creada se usa la funcion Show() por medio de un objeto de la clase Form de la ventana creada.

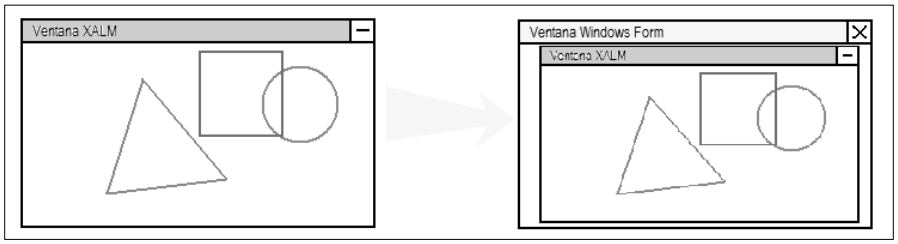


Figura 5.4.4. Ventana WPF vista desde una ventana "windowForm".

5.4.2 Creación de piezas 3D.

Para empezar a programar las piezas 3D que componen la mesa de decoración, se creo una función que devuelve la forma geométrica para cada pieza.

Figura 5.4.5. Funciones para crear objetos 3D usados en el decorador virtual

Las funciones presentadas en la figura 5.4.5 sirven para formar cada una de las piezas que representan a la mesa de decoración virtual. El valor que devuelve cada función es un "GeometryModel3D" que es una representación de puntos y mallas colocados en el espacio 3D para forma una figura tridimensional.

Las siguientes imágenes muestran el objeto "GeometryModel3D" que devuelve cada función.

La Función AnclaBrazos( ) genera las piezas tridimensionales que se muestran en la figura 5.4.6, que corresponden a los sujetadores de los brazos de la mesa de decoración.

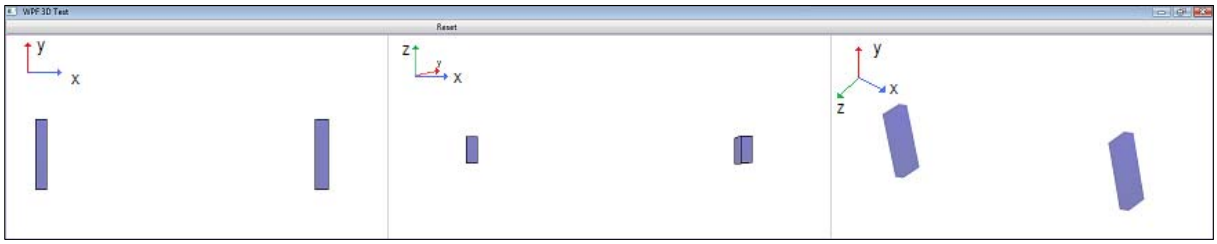


Figura 5.4.6. Vistas tridimensionales del objeto *AnclaBrazos.*

La Función Base( ) genera la pieza tridimensional que se muestran en la figura 5.4.7, que corresponden a la base de la mesa de decoración.

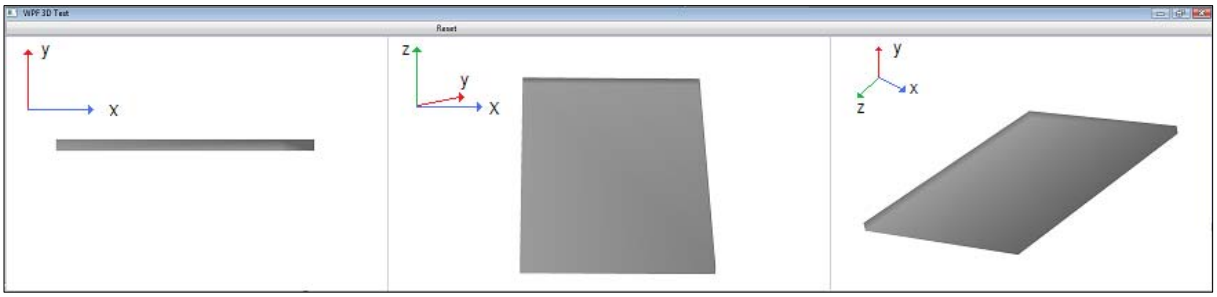


Figura 5.4.7. Vistas tridimensionales del objeto *AnclaBrazos.*

La Función BaseSuperior( ) genera la pieza tridimensional que se muestran en la figura 5.4.8, que corresponden a la base superior de la mesa de decoración.

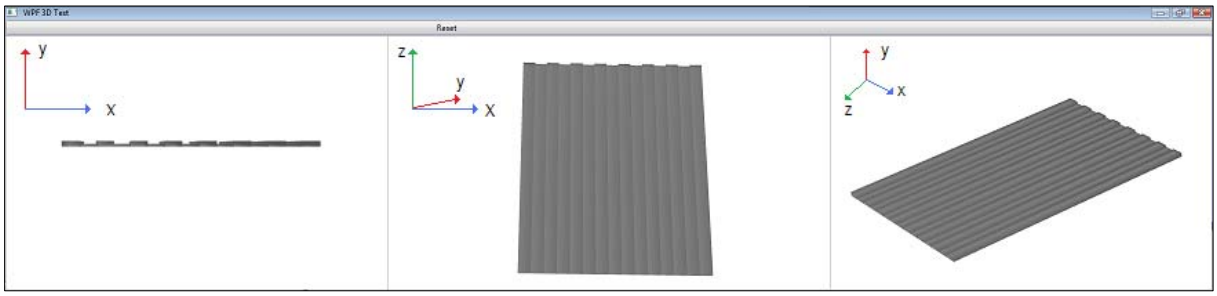


Figura 5.4.8. Vistas tridimensionales del objeto *BaseSuperior.*

La Función Brazos( ) genera las piezas tridimensionales que se muestran en la figura 5.4.9, que corresponden a los brazos de la mesa de decoración.

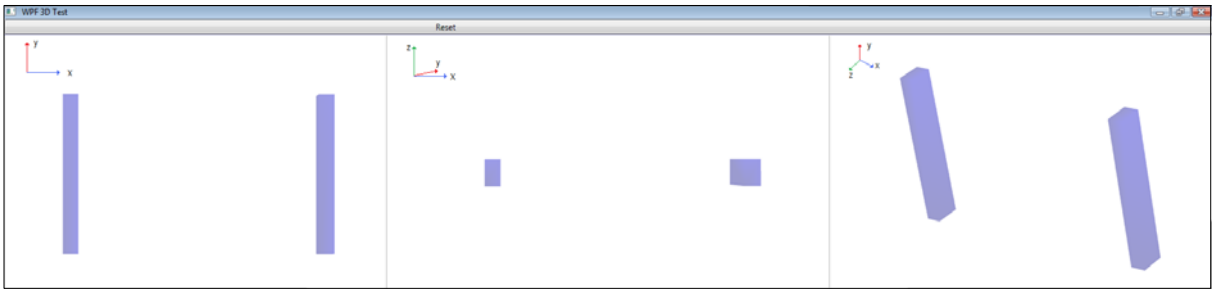


Figura 5.4.9. Vistas tridimensionales del objeto *Brazos.*

La Función PortaDuya( ) genera la pieza tridimensional que se muestra en la figura 5.4.10, que corresponden a la pieza que sostiene la duya, también llamada cabezal de la mesa de decoración.

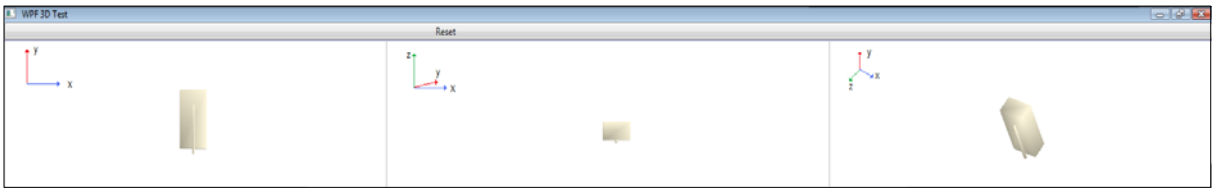


Figura 5.4.10. Vistas tridimensionales del objeto *PortaDuya.*

La Función Postes( ) genera las piezas tridimensionales que se muestran en la figura 5.4.11, que corresponden a los postes de la mesa de decoración.

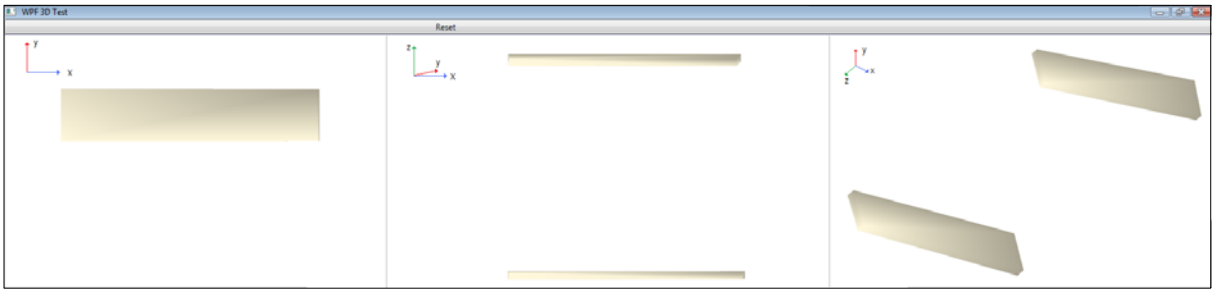


Figura 5.4.11. Vistas tridimensionales del objeto *Postes.*

La Función Prismas( ) genera las piezas tridimensionales que se muestran en la figura 5.4.12, que corresponden a las guías prismáticas de la mesa de decoración.

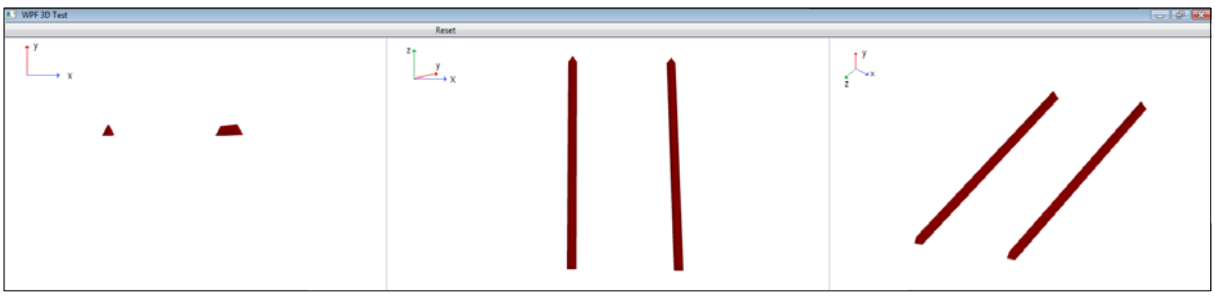


Figura 5.4.12. Vistas tridimensionales del objeto *Prismas.*

La Función SoporteBrazo( ) genera la pieza tridimensional que se muestran en la figura 5.4.13, que corresponden a la base que lleva los brazos de la mesa de decoración.

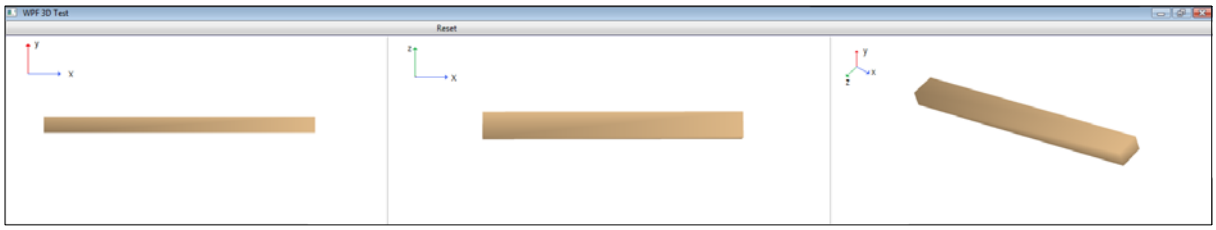


Figura 5.4.13. Vistas tridimensionales del objeto *SoporteBrazo.*

La Función SoporteCabezal( ) genera la pieza tridimensional que se muestran en la figura 5.4.14, que corresponden a la guía donde se mueve el cabezal de la mesa de decoración.

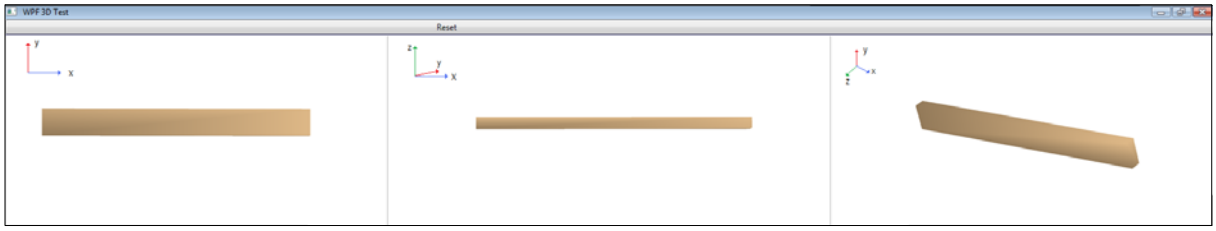


Figura 5.4.14. Vistas tridimensionales del objeto *SoporteCabezal.*

La Función TornilloCabezal( ) genera la pieza tridimensional que se muestran en la figura 5.4.15, que corresponden al tornillo que mueve el cabezal donde se mueve el cabezal de la mesa de decoración.

Función TornilloCabezal( )

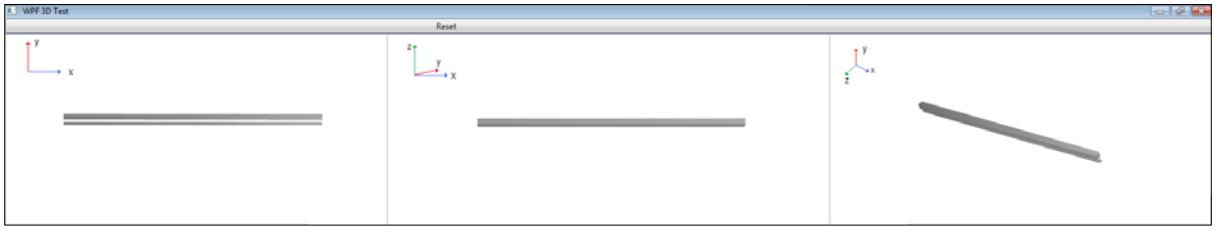


Figura 5.4.14. Vistas tridimensionales del objeto *TornolloCabezal.*

Para poder usar estas funciones se aplico un “ViewPort3D” en el panel de WPF, dentro del ViewPort3D, se colocaron las figuras geométricas que devuelve cada una de las funciones de la figura 5.4.5. Al haber colocado todas las piezas en el panel viewport3D que se ejecuta desde la función “Mesa_3D( )”, se obtuvo la mesa de decoración tal y como se muestra en la figura 5.4.15.

Función Mesa3D( )

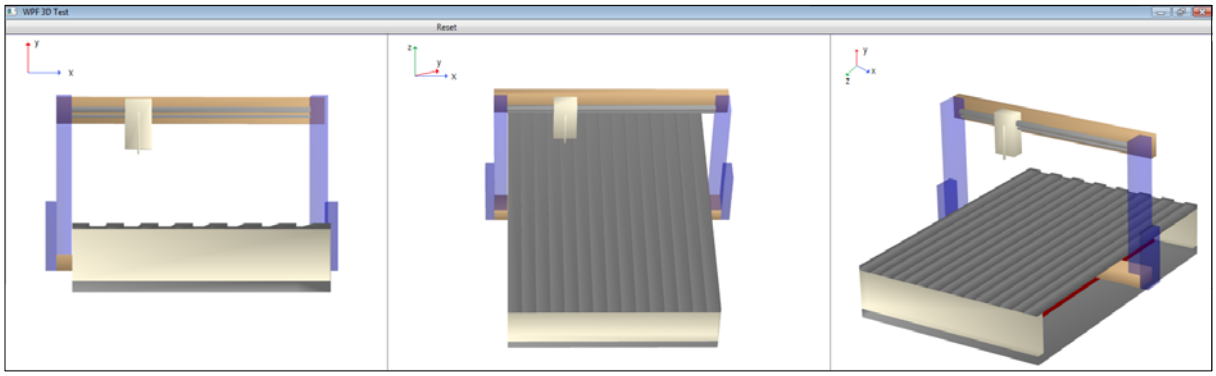


Figura 5.4.15. Vistas tridimensionales de las piezas unidas en el campo visual 3D.

La figura 5.4.15 muestra el resultado de unir las piezas mostradas de la figura 5.4.6 a la 5.4.14, formada por cada una de sus respectivas funciones.

De esta manera se ha logrado crear una mesa de decoración en 3D con la unión de diferentes figuras compuestas.

El código C# utilizado para generar las funciones y la mesa virtual se encuentra en el apéndice C parte II.

5.5. Funcionamiento del Simulador.

El siguiente paso consiste en dar movimiento a las figuras que forman el simulador virtual, respetando que los movimientos no son aleatorios ni siguiendo un patrón fijo, sino que se mueven respecto a los datos obtenidos en el proceso de vectorización del capitulo 4.

La información es leída por líneas de código y representada en un cuadro de texto como lo indica el panel de datos de la figura 5.3.1, para poder mover las piezas en las direcciones X, Y, Z, se sigue la codificación que se ha establecido en la tabla 5.2.

En la tabla 5.2, el código 1000 0000 solo activa la máquina, por lo que no realiza función alguna en el simulador. Por otro lado, el código 1000 0001 indica un movimiento solo en el eje X.

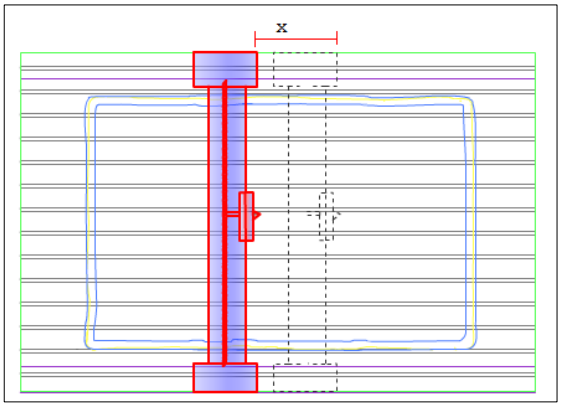


Figura 5.5.1. Desplazamiento del objeto brazo en el simulador.

La figura 5.5.1, muestra el objeto brazo del simulador, el cual tiene un movimiento en X de acuerdo a la instrucción establecida por el código binario. Para que el programa pueda realizar esta operación, se incluyeron los objetos en una clase que recibe incrementos o decrementos en las coordenadas, en este caso los objetos que se desplazan son todos los que están fijos al brazo ("SoporteBrazo", "SoporteCabezal", "AnclaBrazos", "Brazos", "TornilloCabezal"). PortaDuya no se incluyo en los objetos, por que este objeto además de moverse en X, se mueve en Y.

Para mover un objeto 3D se aplica una función de translación como lo es "TranslateTransform3D" que permite desplazar el objeto en X, Y o Z. Para mover todos los objetos en un solo llamado de función de translación se incluye la translación a una función que incluye a los objetos agrupados "Transform3DGroup".

En este caso, para mover el brazo en X, deberá aplicarse una translación de objetos en Z, debido a que las coordenadas en 3D para este programa son diferentes a las coordenadas de imagen 2D o para alguna máquina real.

Por ejemplo para mover el brazo en X una posición se aplico el siguiente código.

```
TranslateTransform3D translacion = new TranslateTransform3D(
                    new Vector3D(0, 0, increX));
Transform3DGroup brazo = new Transform3DGroup();

brazo.Children.Add(translacion);

FigurasBrazo.Transform = brazo;
```

En el código se declaro una variable llamada translación, en la cual se a establecido un incremento de valor "increX" en el eje Z de la función, por lo que se moverá en el eje Z de la pantalla 3D y en el eje X de una imagen.

Después se declaro una variable del tipo Transfor3DGroup, en la cual se contiene la transformación establecida en "translación", por lo que al aplicar a una figura esta propiedad, se moverá en el eje Z el valor contenido en increX.

De esta manera, cada que se incremente o decremente el valor de "increX", el grupo de figuras que forma el brazo será desplazado en la dirección Z o –Z del plano 3D y la dirección X o -X del plano real.

Para el caso en el que se mueve el PortaDuya se necesita un incremento adicional, pues este objeto se mueve tanto en X como en Y, que corresponde a los ejes Z y X del plano 3D, por lo que el fragmento de código que mueve este objeto es similar al anterior, pero con un variable en el Vector X y una en el Vector Z.

```
TranslateTransform3D translacion2 = new TranslateTransform3D(
                    new Vector3D(increY, 0, increX));
Transform3DGroup portDuya = new Transform3DGroup();

portDuya.Children.Add(translacion2);

FiguraDuya.Transform = portDuya;
```

En este caso, si la variable “increY” es la única que cambia se tendrá el efecto que muestra la figura 5.5.2.

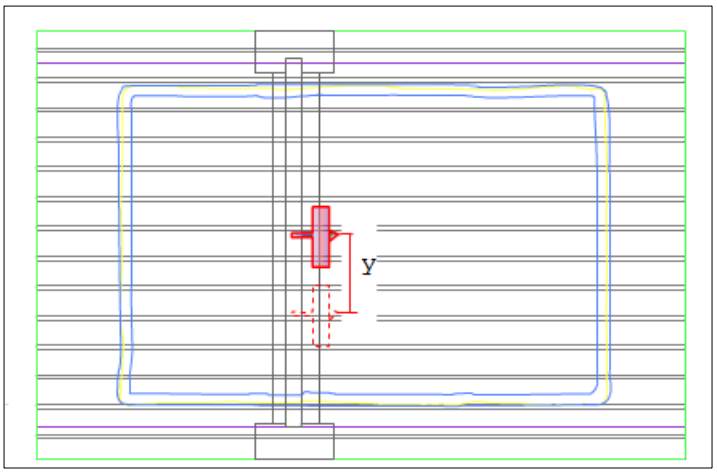


Figura 5.5.2. Desplazamiento del objeto PortaDuya en el simulador.

En el caso en el que la variable “increX” e “increY” tienen una variación en su valor, se tendrá movimiento en los dos ejes, por lo que todos los objetos con efecto de translación se moverán, tal y como se muestra en la figura 5.3.4.

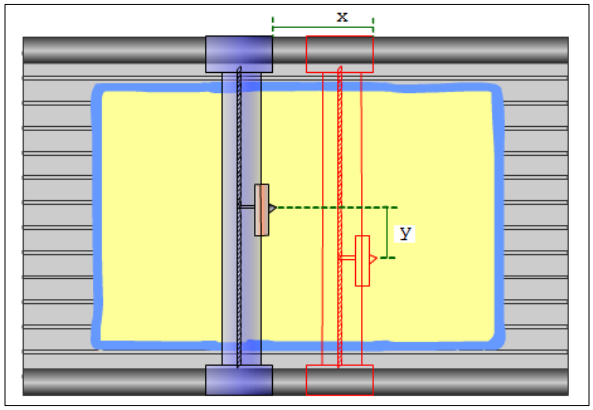


Figura 5.3.4. Desplazamiento del objeto cabezal en el simulador.

Cuando se tiene un código que involucra la activación del eje Z, indica que se esta moviendo el motor de inyección, por lo que se debe estar imprimiendo una imagen mientras este activado. Para simular esta parte se incluyo en el código una tercera variable del tipo booleana, en la que si la variable permanece como “true” se estará trazando una imagen y si permanece como “false” no existe desplazamiento en el eje Z por lo solo se estarán moviendo los objetos.

Conforme la variable del eje Z permanece en estado “true” y se esta trazando una imagen, un indicador muestra el efecto de inyección y vaciado de un contenedor. Tal y como se definió y describió en el elemento 5 de la figura 5.3.1.

El código fuente escrito en C# que controla el flujo de esta información se encuentra el apéndice C parte III.

Capitulo 6

Puerto de Datos

6 PUERTO DE DATOS (PUERTO PARALELO)

La selección del puerto de datos para este proyecto no es la más conveniente, por que no todos los ordenadores cuentan con este puerto de comunicación, pero es un medio de comunicación que ofrece facilidades en el orden en que se manda la información, es fácil de programar y los recursos que ofrece son suficientes para controlar la información. Por otro lado, existen en el mercado convertidores de serie a paralelo ó de USB a paralelo, que pueden ser usados cuando el ordenador no cuente con el puerto paralelo, pero si con el puerto serial o un puerto USB.

6.1. Utilización del puerto paralelo.

El puerto paralelo cuenta con 8 líneas de comunicación para trasmitir datos del ordenador a otro dispositivo. La disposición de estas líneas, servirán para indicar a la unidad de control los movimientos que efectuaran los motores a pasos.

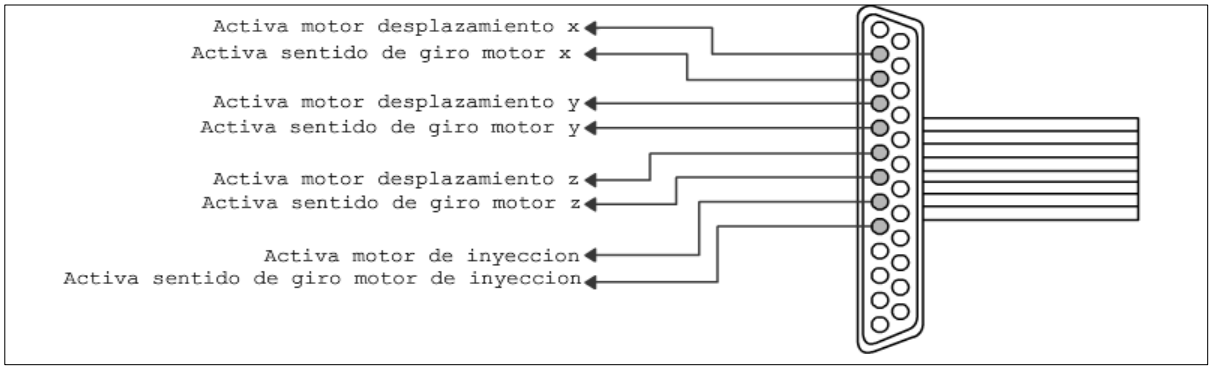


Figura 6.1.1. Representación la disposición que brinda el puerto paralelo para transmitir datos a los dispositivos externos.

La figura 6.1.1 muestra la manera en que se puede utilizar el puerto paralelo para comunicarse con la unidad de control o dispositivo de control al que estén conectados los motores. La facilidad que brinda el puerto paralelo es la de comunicar 8 datos sin necesidad de multiplexar la información.

El puerto paralelo a su vez cuenta con 5 líneas de comunicación que reciben datos. Estas líneas son utilizadas para enviar información al ordenador sobre el estado del dispositivo, en los casos en los que el dispositivo presenta un mal funcionamiento. Un ejemplo es el que se muestra en la figura 6.1.2.

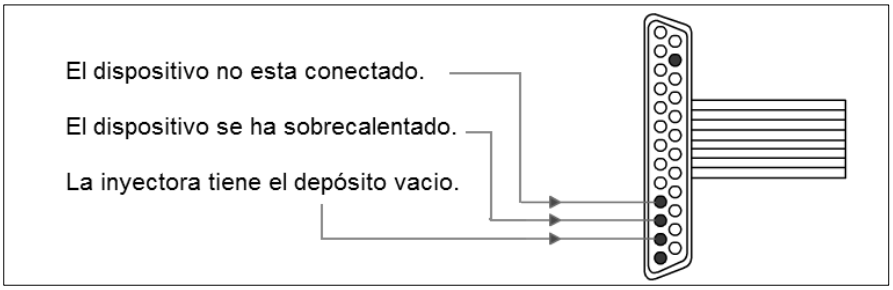


Figura 6.1.2. Disposición de las líneas de comunicación del puerto paralelo para recibir datos de dispositivos externos.

La figura 6.1.2, muestra la distribución de los bits de entrada del puerto paralelo, en donde se indica de que manera pueden ser usadas estas entradas.

6.2. Programando el puerto paralelo en C#.

C# como la mayoría de los compiladores actuales, tiene la capacidad de conectarse con dispositivos externos, en este caso el puerto paralelo. La manera en que se comunica es por medio de una librería llamada “inpout32.dll”. Esta librería es llamada en el compilador de la siguiente manera:

```
public class PortAccess
{
        [DllImport("inpout32.dll", EntryPoint="Out32")]
        public static extern void Output(int adress, int value);
}
```

La función “Output” presenta dos variables enteras, *adress* y *value*, que son utilizadas para comunicarse con el puerto paralelo.

La variable “adress” es la direccion LPT1 del puerto paralelo que sirve para activar la salida de datos al exterior. Para saber que número corresponde la direccion del puerto paralelo, se puede encontrar el valor viendo las propiedades del puerto paralelo con el administrador de dispositivos.

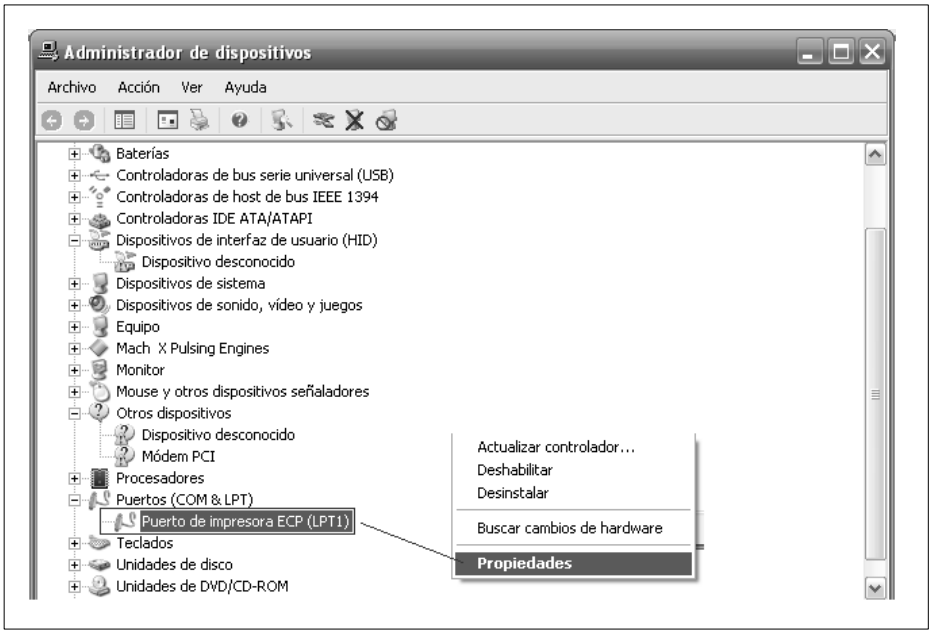


Figura 6.2.1. Administrador de dispositivos de Windows.

La figura 6.2.1, muestra la ventana del administrador de dispositivos de Windows, en la lista de dispositivos se encuentra el puerto paralelo y para conocer su dirección es necesario entrar a las propiedades del dispositivo y seleccionar la pestaña de recursos que muestra la ventana.

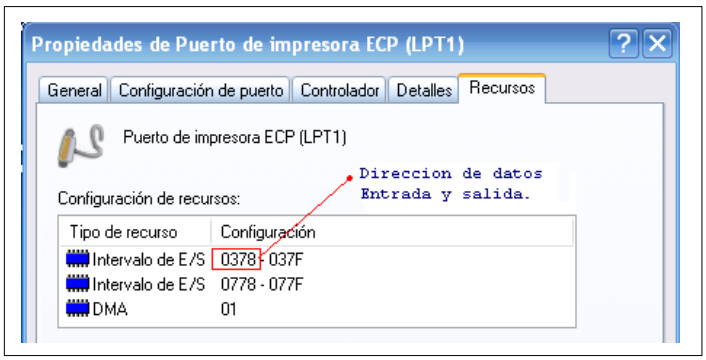


Figura 6.2.2. Propiedades del puerto paralelo.

La figura 6.2.2 muestra como se encuentra configurada las salidas del puerto paralelo, por lo tanto, el valor de la dirección *adress* es 0378, que es un número hexadecimal y 888 seria su valor en decimal.

Por otro lado, la variable "value" es el número de bits que sale por el puerto configurado de salida. Este valor pude ir de 0 a 255.

De esta manera si se llama a Output y se le asigna a *adress* = 888 y *value* = 8, se esta enviando un solo bit por el puerto paralelo correspondiente al 3 bit de salida.

Figura 6.2.3. Puerto de salida y activación de 1 bit.

La figura 6.2.3 muestra la salida que produce si se asigna Output(888,8), en el llamado de la función.

Para poder recibir datos, se puede recurrir a la misma clase, pero con una nueva función llamada:

```
[DllImport("inpout32.dll", EntryPoint="Inp32")]
        public static extern int Input(int adress);
```

La nueva función es Input y solo contiene la variable entera *adress*. Hay que notar que esta función es distinta a la anterior, por que regresa un valor entero, que es el valor que esta recibiendo el puerto paralelo.

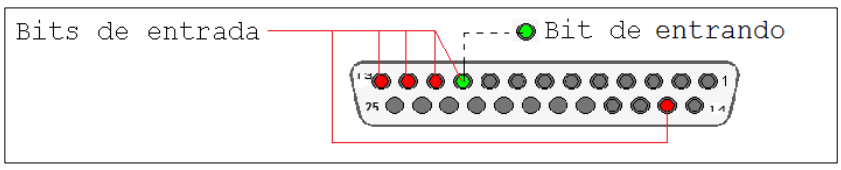


Figura 6.2.4. Bits de entrada y activación de 1 bit de entrada.

La figura 6.2.4. Muestra un bit que esta entrando por el puerto paralelo, este bit entrega un valor entero en la función Input, este valor entero puede ser usado a conveniencia para asignar alguna de las instrucciones escritas en la figura 6.1.2.

La variable *adress* para la función Input será una arriba de la función Output, de tal manera que es 0279 en hexadecimal o 889 en decimal, con lo que el llamado de la función es el siguiente:

```
bit = Input(889);
```

De esta manera se podrá saber si la máquina debe detenerse, iniciar el proceso nuevamente o indicar que no se puede proceder mientras no se solucione el problema.

CONCLUSIONES.

De acuerdo a lo establecido en los objetivos, se ha logrado determinar cual es el filtro adecuado para el sistema propuesto, esto después de una serie de pruebas, en las cuales el filtro de Laplace demostró mejores resultados para este proyecto. Así, con los resultados obtenidos por medio de este filtro, se pudo trabajar en el método de vectorización propuesto en el desarrollo del problema y detallado en el capitulo 4. Este método no es 100% eficaz en encontrar los trazos que mejor realicen el trabajo, debido a que tendría que procesar la información de líneas, como la procesa nuestro cerebro, por lo que se tendría que entrar en otro tema de estudio; pero los datos arrojados por este método son aceptables, pues cumplen con el requisito de llevar a cabo el trazado completo de la imágenes que se usaron.

Finalmente, con la vectorización de las imágenes y la adquisición de datos de desplazamiento, se pudo procesar la información a través de un simulador virtual, que prácticamente, redibuja la imagen obtenida en la detección de contornos, pero emplea los modelos que se utilizarían en una máquina real. De esta manera se pretende que la información enviada por el puerto paralelo, produzca un comportamiento similar al de la simulación.

Por otro lado, las herramientas de dibujo para crear vectores, funcionan de acuerdo a los modelos matemáticos especificados en el tema, pudiendo reimprimirse en la simulación, como se propuso en los modelos matemáticos.

Apéndice A

Apéndice A Código Fuente C#.

En el apéndice A se encuentran fragmentos de código que realizan las funciones de procesamientos de imagen, así como el control de guardar y abrir archivos editados.

Parte I. Abrir Imagen.

```
private void Abrir(  )
 {
 OpenFileDialog ofd = new OpenFileDialog();

 ofd.InitialDirectory = "c:\\Imagenes\\";
 ofd.Filter = "Image files (*.jpg,*.png,*.tif,*.bmp,*.gif)|*.jpg;*.png;*.tif;*.bmp;*.gif|JPG fil" +
              "es (*.jpg)|*.jpg|PNG files (*.png)|*.png|TIF files (*.tif)|*.tif|BMP files (*.bm" +
              "p)|*.bmp|GIF files (*.gif)|*.gif";

 ofd.RestoreDirectory = true;

 if (DialogResult.OK == ofd.ShowDialog())
   {
   m_Bitmap = (Bitmap)Bitmap.FromFile(ofd.FileName, false);
   this.AutoScroll = true;
   this.AutoScrollMinSize = new Size((int)(m_Bitmap.Width * Zoom), (int)(m_Bitmap.Height * Zoom));
   this.Invalidate();
   }

  //Condición para convertir imagen GIF a RGB24bits
 if (m_Bitmap.PixelFormat == PixelFormat.Format8bppIndexed)
   {
   Bitmap tmp = m_Bitmap;

  // Convertir imagen GiF a 24bitsRGB
   m_Bitmap = Clone(tmp, PixelFormat.Format24bppRgb);
   tmp.Dispose();
   }
 width = m_Bitmap.Width;
 height = m_Bitmap.Height;

 } //--------> Termina Abrir Archivo


public static Bitmap Clone(Bitmap source, PixelFormat format)
      {
         int width = source.Width;
         int height = source.Height;

         Bitmap bitmap = new Bitmap(width, height, format);

         // Dibuja la imagen en un nuevo cuadro usando Graphics
         Graphics g = Graphics.FromImage(bitmap);
         g.DrawImage(source, 0, 0, width, height);
         g.Dispose();

         return bitmap;
      }//----> Termnina Función para Convertir GIF a RGB
```

Parte II. Invertir Imagen

```
public static bool Invertir(Bitmap b)
    {
          //Img_data extrae la información de los datos de la imagen para modificarla.
            BitmapData Img_data = b.LockBits(new Rectangle(0, 0, b.Width, b.Height),
            ImageLockMode.ReadWrite, PixelFormat.Format24bppRgb);

        int stride = Img_data.Stride;
        System.IntPtr Scan0 = Img_data.Scan0;

        unsafe
        {


            byte* p = (byte*)(void*)Scan0; //Apuntador a los canales de color de la imagen.

          int nOffset = stride - b.Width * 3;
          int xWidht = b.Width * 3;

          for (int y = 0; y < b.Height; ++y)
          {
             for (int x = 0; x < xWidht; ++x)
             {
                p[0] = (byte)(255 - p[0]);//Invierte el valor de cada pixel de la imagen
                ++p;
             }
             p += nOffset;
          }
        }

        b.UnlockBits(Img_data);

        return true;
    }
```

Parte III. Convertir a escala de grises.

```
public static Int32 GrayScale(Bitmap b)
    {

          BitmapData Img_data = b.LockBits(new Rectangle(0, 0, b.Width,
          b.Height),ImageLockMode.ReadWrite, PixelFormat.Format24bppRgb);
      Int32 contador = 0;

        int stride = Img_data.Stride;
        System.IntPtr Scan0 = Img_data.Scan0;

        unsafe
        {
          byte* p = (byte*)(void*)Scan0;

          int nOffset = stride - b.Width * 3;

          byte red, green, blue;

          for (int y = 0; y < b.Height; ++y)
          {
             for (int x = 0; x < b.Width; ++x)
             {
```

```
                blue = p[0];
                green = p[1];
                red = p[2];
                 //Multiplica los valores por el estandar deYIQ para obtener la escala el tono gris
                p[0] = p[1] = p[2] = (byte)(.299 * red + .587 * green + .114 * blue);
                p += 3;
            }
            p += nOffset;
        }

    }

    b.UnlockBits(Img_data);
    return contador;
}
```

Parte IV. Binarización.

```
public static bool Threshold(Bitmap b, int umbral)
    {

        GrayScale(b);

        BitmapData Img_data = b.LockBits(new Rectangle(0, 0, b.Width, b.Height), ImageLockMode.ReadWrite,
PixelFormat.Format24bppRgb);


        int stride = Img_data.Stride;
        System.IntPtr Scan0 = Img_data.Scan0;

        int nVal = 0;

        unsafe
        {
            byte* p = (byte*)(void*)Scan0;

            int nOffset = stride - b.Width * 3;
            int xWidht = b.Width * 3;

            for (int y = 0; y < b.Height; ++y)
            {
                for (int x = 0; x < xWidht; ++x)
                {
                    nVal = (int)(p[0]);

                            //Establece solo dos tonos de color en la imagen 0 ó 255.
                    if (nVal < umbral) nVal = 0;
                    else nVal = 255;

                    p[0] = (byte)nVal;

                    ++p;
                }
                p += nOffset;
            }
        }
        b.UnlockBits(Img_data);

        return true;
    }
```

Parte V. Alisamiento de Gauss.

```
public static bool Conv5x5(Bitmap b)
    {
        // Se crea un objeto para llamar a la clase ConvMatrix5x5 que contiene los valores de  alisamiento.
        ConvMatrix5x5 m = new ConvMatrix5x5();
        Bitmap bSrc = (Bitmap)b.Clone();

        // GDI+ - regresa en forma BGR, no RGB.
        BitmapData bmData = b.LockBits(new Rectangle(0, 0, b.Width, b.Height), ImageLockMode.ReadWrite,
PixelFormat.Format24bppRgb);
        BitmapData bmSrc = bSrc.LockBits(new Rectangle(0, 0, bSrc.Width, bSrc.Height),
ImageLockMode.ReadWrite, PixelFormat.Format24bppRgb);

        int s = bmData.Stride;
        int s2 = s * 2;
        int s3 = s * 3;
        int s4 = s * 4;
        int mOffset = 0;
        System.IntPtr Scan0 = bmData.Scan0;
        System.IntPtr SrcScan0 = bmSrc.Scan0;

        unsafe
        {
           byte* p = (byte*)(void*)Scan0;
           byte* pSrc = (byte*)(void*)SrcScan0;

           int nOffset = s + 12 - b.Width * 3;
           int nWidth = b.Width - 4;
           int nHeight = b.Height - 4;

           int nPixel;

           for (int y = 0; y < nHeight; ++y)
           {
              for (int x = 0; x < nWidth; ++x)
              {
                  nPixel = ((((pSrc[2] * m.sq11) + (pSrc[5] * m.sq12) + (pSrc[8] * m.sq13) + (pSrc[11] * m.sq14) +
                 (pSrc[14] * m.sq15) +
                 (pSrc[2 + s] * m.sq21) + (pSrc[5 + s] * m.sq22) + (pSrc[8 + s] * m.sq23) +   (pSrc[11 + s] *
                 m.sq24) + (pSrc[14 + s] * m.sq25) +
                 (pSrc[2 + s2] * m.sq31) + (pSrc[5 + s2] * m.sq32) + (pSrc[8 + s2] * m.sq33) + (pSrc[11 + s2] *
                 m.sq34) + (pSrc[14 + s2] * m.sq35) +
                 (pSrc[2 + s3] * m.sq41) + (pSrc[5 + s3] * m.sq42) + (pSrc[8 + s3] * m.sq43) + (pSrc[11 + s3] *
                 m.sq44) + (pSrc[14 + s3] * m.sq45) +
                 (pSrc[2 + s4] * m.sq51) + (pSrc[5 + s4] * m.sq52) + (pSrc[8 + s4] * m.sq53) + (pSrc[11 + s4] *
                 m.sq54) + (pSrc[14 + s4] * m.sq55)) / m.Factor) + mOffset);

                 if (nPixel < 0) nPixel = 0;
                 if (nPixel > 255) nPixel = 255;

                 p[8 + s2] = (byte)nPixel;

                  nPixel = ((((pSrc[1] * m.sq11) + (pSrc[4] * m.sq12) + (pSrc[7] * m.sq13) + (pSrc[10] * m.sq14) +
                  (pSrc[13] * m.sq15) +
                  (pSrc[1 + s] * m.sq21) + (pSrc[4 + s] * m.sq22) + (pSrc[7 + s] * m.sq23) +     (pSrc[10 + s] *
                  m.sq24) + (pSrc[13 + s] * m.sq25) +
                  (pSrc[1 + s2] * m.sq31) + (pSrc[4 + s2] * m.sq32) + (pSrc[7 + s2] * m.sq33) + (pSrc[10 + s2] *
                  m.sq34) + (pSrc[13 + s2] * m.sq35) +
                  (pSrc[1 + s3] * m.sq41) + (pSrc[4 + s3] * m.sq42) + (pSrc[7 + s3] * m.sq43) + (pSrc[10 + s3] *
                  m.sq44) + (pSrc[13 + s3] * m.sq45) +
```

```
                (pSrc[1 + s4] * m.sq51) + (pSrc[4 + s4] * m.sq52) + (pSrc[7 + s4] * m.sq53) + (pSrc[10 + s4] *
                m.sq54) + (pSrc[13 + s4] * m.sq55)) / m.Factor) + mOffset);

                if (nPixel < 0) nPixel = 0;
                if (nPixel > 255) nPixel = 255;

                p[7 + s2] = (byte)nPixel;

                nPixel = ((((pSrc[0] * m.sq11) + (pSrc[3] * m.sq12) + (pSrc[6] * m.sq13) + (pSrc[9] * m.sq14) +
                (pSrc[12] * m.sq15) +
                (pSrc[0 + s] * m.sq21) + (pSrc[3 + s] * m.sq22) + (pSrc[6 + s] * m.sq23) + (pSrc[9 + s] * m.sq24)
                + (pSrc[12 + s] * m.sq25) +
                (pSrc[0 + s2] * m.sq31) + (pSrc[3 + s2] * m.sq32) + (pSrc[6 + s2] * m.sq33) + (pSrc[9 + s2] *
                m.sq34) + (pSrc[12 + s2] * m.sq35) +
                (pSrc[0 + s3] * m.sq41) + (pSrc[3 + s3] * m.sq42) + (pSrc[6 + s3] * m.sq43) + (pSrc[9 + s3] *
                m.sq44) + (pSrc[12 + s3] * m.sq45) +
                (pSrc[0 + s4] * m.sq51) + (pSrc[3 + s4] * m.sq52) + (pSrc[6 + s4] * m.sq53) + (pSrc[9 + s4] *
                m.sq54) + (pSrc[12 + s4] * m.sq55)) / m.Factor) + mOffset);

                if (nPixel < 0) nPixel = 0;
                if (nPixel > 255) nPixel = 255;

                p[6 + s2] = (byte)nPixel;

                p += 3;
                pSrc += 3;
              }

              p += nOffset;
              pSrc += nOffset;
            }
          }

          b.UnlockBits(bmData);
          bSrc.UnlockBits(bmSrc);

          return true;
      }//----> Termina convolución

public class ConvMatrix5x5
  {
      public int sq11 = 1, sq12 = 4, sq13 = 7, sq14 = 4, sq15 = 1;
      public int sq21 = 4, sq22 = 16, sq23 = 26, sq24 = 16, sq25 = 4;
      public int sq31 = 7, sq32 = 26, sq33 = 41, sq34 = 26, sq35 = 7;
      public int sq41 = 4, sq42 = 16, sq43 = 26, sq44 = 16, sq45 = 4;
      public int sq51 = 1, sq52 = 4, sq53 = 7, sq54 = 4, sq55 = 1;

      public int Factor = 273;
      public int Offset = 0;
      public void SetAll(int nVal)

      {
          sq11 = sq12 = sq13 = sq14 = sq15 = nVal;
          sq21 = sq22 = sq23 = sq24 = sq25 = nVal;
          sq31 = sq32 = sq33 = sq34 = sq35 = nVal;
          sq41 = sq42 = sq43 = sq44 = sq45 = nVal;
          sq51 = sq52 = sq53 = sq54 = sq55 = nVal;
      }

  }
```

Parte VI – Detección de bordes – Máscara de Laplace 1ra forma

```
public static bool MascaraLaplace(Bitmap b)
    {
       ConvMatrix m = new ConvMatrix();
       m.ArriMed = m.AbaMed = m.MedIzq = m.MedioDer = -1;
       m.ArrIzq = m.ArrDer = m.AbaIzq = m.AbaDer = 0;
       m.Centro = 4;
       m.Offset = 0;

       return Filtros.Conv3x3(b, m);
    }


public static bool Conv3x3(Bitmap b, ConvMatrix m)
{
       // para evitar error por divisor de cero
   if (0 == m.Factor) return false;

   Bitmap Img_copy = (Bitmap)b.Clone();

       // GDI+ - regresa en forma BGR, no RGB.
       BitmapData Img_data = b.LockBits(new Rectangle(0, 0, b.Width, b.Height),
       ImageLockMode.ReadWrite, PixelFormat.Format24bppRgb);
       BitmapData data_copy = Img_copy.LockBits(new Rectangle(0, 0, Img_copy.Width, Img_copy.Height),
       ImageLockMode.ReadWrite, PixelFormat.Format24bppRgb);

    int stride = Img_data.Stride;
    int stride2 = stride * 2;
    System.IntPtr Scan0 = Img_data.Scan0;
    System.IntPtr SrcScan0 = data_copy.Scan0;

    unsafe
       {
          byte* p = (byte*)(void*)Scan0;
          byte* p_copy = (byte*)(void*)SrcScan0;

          int nOffset = stride + 6 - b.Width * 3;
          int xWidht = b.Width - 2;
          int xHeight = b.Height - 2;

          int nPixel;

          for (int y = 0; y < xHeight; ++y)
          {

             for (int x = 0; x < xWidht; ++x)
             {
                          nPixel = ((((p_copy[2] * m.ArrIzq) + (p_copy[5] * m.ArriMed) +          (p_copy[8] *
                       m.ArrDer) + (p_copy[2 + stride] * m.MedIzq) +
                       (p_copy[5 + stride] * m.Centro) +  (p_copy[8 + stride] * m.MedioDer) +
                       (p_copy[2 + stride2] * m.AbaIzq) + (p_copy[5 + stride2] * m.AbaMed) + (p_copy[8 +
                       stride2] * m.AbaDer)) / m.Factor) + m.Offset);

                if (nPixel < 0) nPixel = 0;
                if (nPixel > 255) nPixel = 255;

                p[5 + stride] = (byte)nPixel;

                          nPixel = ((((p_copy[1] * m.ArrIzq) + (p_copy[4] * m.ArriMed) +
                          (p_copy[7] * m.ArrDer) + (p_copy[1 + stride] * m.MedIzq) +
                          (p_copy[4 + stride] * m.Centro) + (p_copy[7 + stride] * m.MedioDer) +
```

```
                    (p_copy[1 + stride2] * m.AbaIzq) + (p_copy[4 + stride2] * m.AbaMed) + (p_copy[7 +
                    stride2] * m.AbaDer)) / m.Factor) + m.Offset);

            if (nPixel < 0) nPixel = 0;
            if (nPixel > 255) nPixel = 255;

            p[4 + stride] = (byte)nPixel;

                    nPixel = ((((p_copy[0] * m.ArrIzq) + (p_copy[3] * m.ArriMed) +
                    (p_copy[6] * m.ArrDer) + (p_copy[0 + stride] * m.MedIzq) +
                    (p_copy[3 + stride] * m.Centro) + (p_copy[6 + stride] * m.MedioDer) +
                    (p_copy[0 + stride2] * m.AbaIzq) + (p_copy[3 + stride2] * m.AbaMed) + (p_copy[6 +
                    stride2] * m.AbaDer)) / m.Factor) + m.Offset);

            if (nPixel < 0) nPixel = 0;
            if (nPixel > 255) nPixel = 255;

            p[3 + stride] = (byte)nPixel;

            p += 3;
            p_copy += 3;
          }

          p += nOffset;
          p_copy += nOffset;
        }
      }

      b.UnlockBits(Img_data);
      Img_copy.UnlockBits(data_copy);

      return true;
    }//----> Termina convolución
```

Apéndice B

Apéndice B Código Fuente C#.

En el apéndice B se encuentran fragmentos de código que realizan las funciones de generación de vectores, esta sección contiene procedimientos de cálculo de desplazamiento de los pixeles para formar trayectorias y algunas herramientas para leer archivos de vector y guardar archivos de vector.

Parte I – Convertir Imagen a Vector.

```
public ArrayList px = new ArrayList();
public ArrayList py = new ArrayList();
public ArrayList vx = new ArrayList();
public ArrayList vy = new ArrayList();
public int[] vecinosx = { 1, 1, 0, -1, -1, -1, 0, 1 };
public int[] vecinosy = { 0, 1, 1,  1,  0, -1,-1,-1 };

public bool road(ArrayList ppx, ArrayList ppy, ArrayList vvx, ArrayList vvy, int mm2, int nn2, int llongitud)
 {
 int m1, n1, m2, n2;
 int caminos = 0;
 bool respuesta = true;

 m1 = mm2;
 n1 = nn2;


 for (int i = 0; i < ppx.Count; i++)
   {
   m2 = (int)ppx[i];
   n2 = (int)ppy[i];

   for (int j = 0; j < 8; j++)
    {
      if (m2 - m1 == vecinosx[j] && n2 - n1 == vecinosy[j])
        {
        caminos = caminos + 1;
        for (int k = 0; k < vvx.Count; k++)
            if (m2 == (int)vvx[k] && n2 == (int)vvy[k])
              caminos = caminos - 1;
        }
      if (caminos > 0)
        {
        respuesta = false;
        }
      if (llongitud < 3)
        {
        respuesta = false;
        }
      }

  }
      return respuesta;
}

public bool Vectores2(Bitmap b)
  {

  px.Clear(); py.Clear(); vx.Clear(); vy.Clear();
  int m1, m2, n1, n2;
  int caminos = 0;
  int anteriorx = 0;
```

```
int anteriory = 0;
int[] movx = { 1, 1, 0, -1, -1, -1, 0, 1 };
int[] movy = { 0, 1, 1, 1, 0, -1, -1, -1 };
int longitud = 0;

ArrayList direccion = new ArrayList();
ArrayList longitudes = new ArrayList();

Bitmap tmp = b;
BitmapData bmData = b.LockBits(new Rectangle(0, 0, b.Width, b.Height),
        ImageLockMode.ReadWrite, PixelFormat.Format24bppRgb);
int stride = bmData.Stride;
System.IntPtr Scan0 = bmData.Scan0;

unsafe
 {
 byte* p = (byte*)(void*)Scan0;
 int nOffset = stride - b.Width * 3;

 for (int y = 0; y < b.Height; ++y)
    {
    for (int x = 0; x < b.Width; ++x)
       {
       if (p[0] == 255)
         {
         px.Add(x);
         py.Add(y);
         }
       p += 3;
       }
    p += nOffset;
    }
 }
 b.UnlockBits(bmData);

 for (int h = 0; h < px.Count; h++)
    {
    m1 = (int)px[h];
    n1 = (int)py[h];

    for (int i = 0; i < py.Count; i++)
       {
       m2 = (int)px[i];
       n2 = (int)py[i];

       for (int j = 0; j < 8; j++)
          if (m2 - m1 == movx[j] && n2 - n1 == movy[j])
            {
            caminos = caminos + 1;
            for (int k = 0; k < vx.Count; k++)
               if (m2 == (int)vx[k] && n2 == (int)vy[k])
                  caminos = caminos - 1;

         }
          if (caminos == 1)
            {
               anteriorx = m1;
               anteriory = n1;
               vx.Add(m1);
               vy.Add(n1);

               m1 = m2;
```

```
                n1 = n2;
                i = 0;
                caminos = 0;
                longitud = longitud + 1;

                if (road(px, py, vx, vy, m2, n2, longitud))
                {
                    vx.Add(m2);
                    vy.Add(n2);
                    longitudes.Add(longitud);
                    longitud = 0;
                }
             //después de este  si ya no hay trayectoria otorgar el valor de m2 a vx.
            }
        }
    }
    return true;
}
```

Parte II – Crear vectores.

// Función que recibe los parámetros de punto inicial de X,Y y punto final de X,Y para formar una liena recta entre estos dos puntos.

```
public static ArrayList ConvertirLinea(int xi, int yi, int xf, int yf)
    {
        ArrayList vlxy = new ArrayList();
        int xr, yr;
        double k;
        double d = 0;

        xr = xf - xi;
        yr = yf - yi;

        #region Construye linea

        if (Math.Abs(xr) > Math.Abs(yr))
        {
            k = (double)yr / (double)xr;

            if (xr >= 0 && yr >= 0)     //(+, +)
            {
                for (int i = 0; i <= xr; i++)
                {
                    vlxy.Add(i + xi);
                    vlxy.Add(Convert.ToInt16(d) + yi);
                    d = d + k;
                }

            }
            if (xr < 0 && yr >= 0)     //(-, +)
            {
                for (int i = 0; i <= Math.Abs(xr); i++)
                {
                    vlxy.Add(xi - i);
                    vlxy.Add(Convert.ToInt16(d) + yi);
                    d = d - k;


                }
            }
```

```
    if (xr < 0 && yr < 0)     //(-, -)
    {
        for (int i = 0; i <= Math.Abs(xr); i++)
        {
            vlxy.Add(xi - i);
            vlxy.Add(Convert.ToInt16(d) + yi);
            d = d - k;
        }
    }
    if (xr >= 0 && yr < 0)     //(+, -)
    {
        for (int i = 0; i <= xr; i++)
        {
            vlxy.Add(xi + i);
            vlxy.Add(Convert.ToInt16(d) + yi);
            d = d + k;
        }
    }
}

else
{
    k = (double)xr / (double)yr;

    if (xr >= 0 && yr >= 0)     //(+, +)
    {
        for (int i = 0; i <= yr; i++)
        {

            vlxy.Add(Convert.ToInt16(d) + xi);
            vlxy.Add(yi + i);
            d = d + k;
        }
    }

    if (xr < 0 && yr >= 0)     //(-, +)
    {
        for (int i = 0; i <= yr; i++)
        {

            vlxy.Add(Convert.ToInt16(d) + xi);
            vlxy.Add(yi + i);
            d = d + k;
        }
    }

    if (xr < 0 && yr < 0)     //(-, -)
    {

        for (int i = 0; i <= Math.Abs(yr); i++)
        {

            vlxy.Add(Convert.ToInt16(d) + xi);
            vlxy.Add(yi - i);
            d = d - k;
        }
    }

    if (xr >= 0 && yr < 0)     //(+, -)
    {
        for (int i = 0; i <= Math.Abs(yr); i++)
        {
```

```
                vlxy.Add(Convert.ToInt16(d) + xi);
                vlxy.Add(yi - i);
                d = d - k;
            }
        }
    }
    #endregion

    return vlxy;
}
```

// Función que recibe los parámetros de punto inicial de X,Y y punto final de X,Y para formar una rectangulo de izquierda a derecha con estos valores.

```
public static ArrayList ConvertirRectangulo(int xi, int yi, int xf, int yf)
{
    ArrayList vlxy = new ArrayList();

    for (int i = xi; i <= xf; i++)
    {
        vlxy.Add(i);
        vlxy.Add(yi);
    }

    for (int i = yi + 1; i <= yf; i++)
    {
        vlxy.Add(xf);
        vlxy.Add(i);
    }

    for (int i = xf - 1; i >= xi; i--)
    {
        vlxy.Add(i);
        vlxy.Add(yf);
    }

    for (int i = yf - 1; i >= yi; i--)
    {
        vlxy.Add(xi);
        vlxy.Add(i);
    }
    return vlxy;
}
```

// Función que recibe los parámetros de punto inicial de X,Y y punto final de X,Y para formar una elipse.

```
public static ArrayList ConvertirElipse(int xi, int yi, int xf, int yf)
{
    ArrayList vlxy = new ArrayList();
    int posicionX, posicionY;
    int radio;
    int radiox;
    int radioy;

    radiox = (int)((xf - xi) / 2);
    radioy = (int)((yf - yi) / 2);
    posicionX = (int)((xi + xf) / 2);
    posicionY = (int)((yi + yf) / 2);
```

```
if (radiox > radioy)
    radio = radiox;
else
    radio = radioy;


/*Se hace cero el primer valor de vlx y vly para poder tener un valor inicial y
  poder comparar los valores que proporcione la funcion de senos y coseno y no
  tener valores repetidos ni tener errores de comparar con un vlx o vly sin contenido.
 */
vlxy.Add(0); vlxy.Add(0);
int xc = 0;
int yc = 0;

if (radio >= 380)
{
    for (int i = 0; i < 3600; i++)
    {
        float x = (float)Math.Cos(2 * Math.PI * i / 3600) * radiox + posicionX;
        float y = -(float)Math.Sin(2 * Math.PI * i / 3600) * radioy + posicionY;

        int xa = Convert.ToInt16(x);
        int ya = Convert.ToInt16(y);



        if (xa == xc && ya == yc)
        {
            xc = xa;
            yc = ya;
        }
        else
        {
            xc = xa;
            yc = ya;
            vlxy.Add(xa);
            vlxy.Add(ya);
        }
    }
}

if (radio >= 290 && radio < 380)
{
    for (int i = 0; i < 2400; i++)

     {
        float x = (float)Math.Cos(2 * Math.PI * i / 2400) * radiox + posicionX;
        float y = -(float)Math.Sin(2 * Math.PI * i / 2400) * radioy + posicionY;

        int xa = Convert.ToInt16(x);
        int ya = Convert.ToInt16(y);

        if (xa == xc && ya == yc)
        {
            xc = xa;
            yc = ya;
        }
        else
        {
            xc = xa;
```

```
            yc = ya;
            vlxy.Add(xa);
            vlxy.Add(ya);
         }
      }
   }

   if (radio >= 200 && radio < 290)
   {
      for (int i = 0; i < 1800; i++)
      {
         float x = (float)Math.Cos(2 * Math.PI * i / 1800) * radiox + posicionX;
         float y = -(float)Math.Sin(2 * Math.PI * i / 1800) * radioy + posicionY;

         int xa = Convert.ToInt16(x);
         int ya = Convert.ToInt16(y);

         if (xa == xc && ya == yc)
         {
            xc = xa;
            yc = ya;
         }
         else
         {
            xc = xa;
            yc = ya;
            vlxy.Add(xa);
            vlxy.Add(ya);
         }
      }
   }

   if (radio >= 100 && radio < 200)
   {
      for (int i = 0; i < 1200; i++)
      {
         float x = (float)Math.Cos(2 * Math.PI * i / 1200) * radiox + posicionX;
         float y = -(float)Math.Sin(2 * Math.PI * i / 1200) * radioy + posicionY;

         int xa = Convert.ToInt16(x);
         int ya = Convert.ToInt16(y);
         if (xa == xc && ya == yc)
         {
            xc = xa;
            yc = ya;

         }
         else
         {
            xc = xa;
            yc = ya;
            vlxy.Add(xa);
            vlxy.Add(ya);
         }
      }
   }

   if (radio >= 60 && radio < 100)
   {
      for (int i = 0; i < 720; i++)
      {
         float x = (float)Math.Cos(2 * Math.PI * i / 720) * radiox + posicionX;
```

```
            float y = -(float)Math.Sin(2 * Math.PI * i / 720) * radioy + posicionY;

            int xa = Convert.ToInt16(x);
            int ya = Convert.ToInt16(y);

            if (xa == xc && ya == yc)
            {
               xc = xa;
               yc = ya;
            }
            else
            {
               xc = xa;
               yc = ya;
               vlxy.Add(xa);
               vlxy.Add(ya);
            }
         }
      }

      if (radio < 60)
      {
         for (int i = 0; i < 360; i++)
         {
            float x = (float)Math.Cos(2 * Math.PI * i / 360) * radiox + posicionX;
            float y = -(float)Math.Sin(2 * Math.PI * i / 360) * radioy + posicionY;

            int xa = Convert.ToInt16(x);
            int ya = Convert.ToInt16(y);

            if (xa == xc && ya == yc)
            {
               xc = xa;
               yc = ya;
            }
            else
            {
               xc = xa;
               yc = ya;
               vlxy.Add(xa);
               vlxy.Add(ya);
            }
         }
      }
      vlxy.RemoveAt(0);
      vlxy.RemoveAt(0);
      return vlxy;

   }
```

Parte III – Guardar Vectores.

```
private void guardarVectores( )
   {
      if (vvx.Count > 1)
      {

         Stream myStream;
         SaveFileDialog saveFileDialog1 = new SaveFileDialog();

         saveFileDialog1.Filter = "Vectores files (*.vxt)|*.vxt";
```

```
            saveFileDialog1.FilterIndex = 2;
            saveFileDialog1.RestoreDirectory = true;

            if (saveFileDialog1.ShowDialog() == DialogResult.OK)
            {
               if ((myStream = saveFileDialog1.OpenFile()) != null)
               {
                  StreamWriter wText = new StreamWriter(myStream);

                  for (int i = 0; i < vvx.Count; i++)
                  {
                     wText.Write("(" + vvx[i].ToString() + "," + vvy[i].ToString() + ")");
                     if (i % 10 == 0)
                        wText.WriteLine();
                  }
                  wText.Close();
               }
            }

         }
         else
         {
            Aviso1 ventana1 = new Aviso1();
            ventana1.Aviso_msg = "¡Aun no se han generado los vectores de la imagen!";
            ventana1.Show();
         }
          }
```

Parte VI – Abrir Vectores.

```
private void abrirVectores()
   {
      OpenFileDialog ofd = new OpenFileDialog();
      //ofd.Filter = " Archivos Vxt (*.txt)|*.txt | Todos los archivos (*.*)|*.*";
      ofd.Filter = " Archivos Vxt (*.vxt)|*.vxt";
      ofd.Title = " Selecciona un archivo";
      ofd.FilterIndex = 2;

      if (ofd.ShowDialog() == DialogResult.OK)
         {
         ArrayList vxy = new ArrayList();
         StreamReader archivoVxt = new StreamReader(ofd.FileName);
         StreamReader copiaVxt = new StreamReader(ofd.FileName);

         vxy = (ArrayList)ConversorVtx.ConverTxt_a_Points(archivoVxt, copiaVxt).Clone();

         for (int i = 0; i < vxy.Count - 2; i += 2)
            {
               vvx.Add(vxy[i]);
               vvy.Add(vxy[i + 1]);
            }

          VectorData ventana3 = new VectorData(vvx, vvy);
          ventana3.Show();
          if (vvx.Count > 1)
            imageToolBar.Visible = true;

             }
    }
```

```
        //Pertenece a la clase Class ConversorVtx
public static ArrayList ConverTxt_a_Points(StreamReader VxtFile, StreamReader VtxCopy)
    {
        string num;
        int nlinea = 0;
        ArrayList vx = new ArrayList();
        while (VtxCopy.ReadLine() != null)
        {
            nlinea++;
        }

        string[] ncadena = new string[nlinea+1];

        for (int i = 0; i < nlinea; i++)
        {
            ncadena[nlinea] = VxtFile.ReadLine();

            for (int j = 0; j < ncadena[nlinea].Length; j++)
            {
                if (ncadena[nlinea].Substring(j, 1) == "(")
                {
                    j++;
                    num = ncadena[nlinea].Substring(j, 1);

                    if (ncadena[nlinea].Substring(j + 1, 1) != ",")
                    {
                        num = ncadena[nlinea].Substring(j, 2);
                        j++;
                        if (ncadena[nlinea].Substring(j + 1, 1) != ",")
                        {
                            num = ncadena[nlinea].Substring(j - 1, 3);
                            j++;

                        }
                    }

                    vx.Add((int)Convert.ToInt16(num));
                }

                if (ncadena[nlinea].Substring(j, 1) == ",")
                {
                    j++;
                    num = ncadena[nlinea].Substring(j, 1);
                    if (ncadena[nlinea].Substring(j + 1, 1) != ")")
                    {
                        num = ncadena[nlinea].Substring(j, 2);
                        j++;
                        if (ncadena[nlinea].Substring(j + 1, 1) != ")")
                        {
                            num = ncadena[nlinea].Substring(j - 1, 3);
                            j++;

                        }
                    }
                     vx.Add((int)Convert.ToInt16(num));
                }
            }
        }
        VxtFile.Close();
        VtxCopy.Close();
        return vx;
        }
```

Apéndice C

Apéndice C Código Fuente C#.

En el apéndice C se encuentran fragmentos de código que realizan las funciones de generar código binario a partir de la información obtenida en la adquisición de vectores. De igual forma se incluye en esta sección el código para generar las piezas de la mesa 3D y los elementos necesarios para el prototipo virtual.

Parte I – Codificación de Vectores a Bits de Salida.

```
//Declaración de variables globales
private ArrayList DataPort = new ArrayList();
private ArrayList vx = new ArrayList();
private ArrayList vy = new ArrayList();

private void conversor()
    {
        int xi, yi, xf, yf, xr, yr;
        double k;
        double d = 0;

        vx.Insert(0, 0);
        vy.Insert(0, 0);
        vx.Add(0);
        vy.Add(0);
        ArrayList vlx = new ArrayList();
        ArrayList vly = new ArrayList();

        #region Etiquetas de Funcionalidad - Codigo Binario
        /*
         * FUNCIONES DE DESPLAZAMIENTO.
         * Binario   hexadecimal   Decimal
         * 10000000    80              128  activado
         * 10000001        81          129    X
         * 10000100        84          132    Y
         * 10010000        90          144    Z
         * 10000010        82          130   -X
         * 10001000        88          136   -Y
         * 10100000        A0          160   -Z
         * 10000101        85          133    X,Y
         * 10010001        91          145    X,Z
         * 10010100        94          148    y,Z
         * 10000110        86          134   -X,Y
         * 10010010        92          146   -X,Z
         * 10011000        98          152   -Y,Z
         * 10001001        89          137    X,-Y
         * 10001010        8A          138   -X,-Y
         * 10010101        95          149    X, Y,Z
         * 10010110        96          150   -X, Y,Z
         * 10011001        99          153    X,-Y,Z
         * 10011010        98C182       154   -X,-Y,Z
         */
        #endregion

        for (int h = 0; h < vx.Count - 1; h++)
        {
            xi = (int)vx[h]; yi = (int)vy[h];
            xf = (int)vx[h + 1]; yf = (int)vy[h + 1];
            xr = xf - xi;
            yr = yf - yi;

            //Obtiene los bits de salida para puerto paralelo con efecto en Z.
            #region Conversion a Bit de SAIIDA
```

```
if (xr == 1 && yr == 0)
{
   DataPort.Add(145);     //X,Z
}
if (xr == 1 && yr == 1)
{
   DataPort.Add(149);     //X,Y,Z
}
if (xr == 0 && yr == 1)
{
   DataPort.Add(148);     //Y,Z
}
if (xr == -1 && yr == 1)
{
   DataPort.Add(150);     //-X,Y,Z
}
if (xr == -1 && yr == 0)
{
   DataPort.Add(146);     //-X,Z
}
if (xr == -1 && yr == -1)
{
   DataPort.Add(154);     //-X,-Y,Z
}
if (xr == 0 && yr == -1)
{
   DataPort.Add(152);     //-Y,Z
}
if (xr == 1 && yr == -1)
{
   DataPort.Add(153);     //X,-Y,Z
}
#endregion

//Obtiene los bits de salida para puerto paralelo sin efecto en Z.
//Es decir que aquí se obtienen trayectorias en liena recta para mover solo el cabezal.

if (Math.Abs(xr) >= 2 || Math.Abs(yr) >= 2)
{
   #region Construye linea

   if (Math.Abs(xr) > Math.Abs(yr))
   {
      k = (double)yr / (double)xr;

      if (xr >= 0 && yr >= 0)     //(+, +)
      {
         for (int i = 0; i <= xr; i++)
         {
            vlx.Add(i + xi);
            vly.Add(Convert.ToInt16(d) + yi);
            d = d + k;

         }
      }
      if (xr < 0 && yr >= 0)     //(-, +)
      {
         for (int i = 0; i <= Math.Abs(xr); i++)
         {
            vlx.Add(xi - i);
            vly.Add(Convert.ToInt16(d) + yi);
            d = d - k;
```

```
            }
        }
        if (xr < 0 && yr < 0)     //(-, -)
        {
            for (int i = 0; i <= Math.Abs(xr); i++)
            {
                vlx.Add(xi - i);
                vly.Add(Convert.ToInt16(d) + yi);
                d = d - k;
            }
        }
        if (xr >= 0 && yr < 0)     //(+, -)
        {
            for (int i = 0; i <= xr; i++)
            {
                vlx.Add(xi + i);
                vly.Add(Convert.ToInt16(d) + yi);
                d = d + k;
            }
        }
    }


    else
    {
        k = (double)xr / (double)yr;

        if (xr >= 0 && yr >= 0)     //(+, +)
        {
            for (int i = 0; i <= yr; i++)
            {
                vly.Add(yi + i);
                vlx.Add(Convert.ToInt16(d) + xi);
                d = d + k;
            }
        }

        if (xr < 0 && yr >= 0)     //(-, +)
        {
            for (int i = 0; i <= yr; i++)
            {
                vly.Add(yi + i);
                vlx.Add(Convert.ToInt16(d) + xi);
                d = d + k;
            }
        }

        if (xr < 0 && yr < 0)     //(-, -)
        {
            for (int i = 0; i <= Math.Abs(yr); i++)
            {
                vly.Add(yi - i);
                vlx.Add(Convert.ToInt16(d) + xi);
                d = d - k;
            }
        }

        if (xr >= 0 && yr < 0)     //(+, -)
        {
            for (int i = 0; i <= Math.Abs(yr); i++)
            {
                vly.Add(yi - i);
```

```
                    vlx.Add(Convert.ToInt16(d) + xi);
                    d = d - k;
                }
            }
        }
        #endregion
        d = 0;
        k = 0;
        for (int b = 0; b < vlx.Count - 1; b++)
        {
            int xi2, yi2, xf2, yf2, xr2, yr2;
            xi2 = (int)vlx[b]; yi2 = (int)vly[b];
            xf2 = (int)vlx[b + 1]; yf2 = (int)vly[b + 1];
            xr2 = xf2 - xi2;
            yr2 = yf2 - yi2;

        //Convierte los vectores a bits de salida para puerto paralelo sin Efecto en Z
            #region Conversion a Bit de SAIIDA
            if (xr2 == 1 && yr2 == 0)
            {
                DataPort.Add(129);     //X
            }
            if (xr2 == 1 && yr2 == 1)
            {
                DataPort.Add(133);     //X,Y
            }
            if (xr2 == 0 && yr2 == 1)
            {
                DataPort.Add(132);     //Y
            }
            if (xr2 == -1 && yr2 == 1)
            {
                DataPort.Add(134);     //-X,Y
            }
            if (xr2 == -1 && yr2 == 0)
            {
                DataPort.Add(130);     //-X
            }
            if (xr2 == -1 && yr2 == -1)
            {
                DataPort.Add(138);     //-X,-Y
            }
            if (xr2 == 0 && yr2 == -1)
            {
                DataPort.Add(136);     //-Y
            }
            if (xr2 == 1 && yr2 == -1)
            {
                DataPort.Add(137);     //X,-Y
            }
            #endregion
        }
        vlx.Clear();
        vly.Clear();
    } //termina condición para formación de translado sin impresión.
  }
}
//Al finalizar esta funcion el arreglo DataPort contiene los bites de salida para el puerto paralelo //que
corresponden a los pulsos necesarios en los motores para formar la imagen.
```

Parte II – Creación de figuras geométricas para la mesa 3D.

```
private GeometryModel3D AnclaBrazos()
    {
        GeometryModel3D mGeometry = new GeometryModel3D();
        mesh = new MeshGeometry3D();
        double separacion = 0.53;
        int barras = 0;

        //Medida del AnclaBrazos Largo X = 0.02  Alto Y = 0.13  Ancho Z = 0.05

        mesh.Positions.Add(new Point3D(-0.05, 0.03, 0.3)); //creación de lo puntos
        mesh.Positions.Add(new Point3D(-0.03, 0.03, 0.3)); //en el espacio tridi--
        mesh.Positions.Add(new Point3D(-0.03, 0.16, 0.3)); //mensional.
        mesh.Positions.Add(new Point3D(-0.05, 0.16, 0.3));

        mesh.Positions.Add(new Point3D(-0.05, 0.03, 0.25));
        mesh.Positions.Add(new Point3D(-0.03, 0.03, 0.25));
        mesh.Positions.Add(new Point3D(-0.03, 0.16, 0.25));
        mesh.Positions.Add(new Point3D(-0.05, 0.16, 0.25));


        mesh.Positions.Add(new Point3D(-0.05 + separacion, 0.03, 0.3));
        mesh.Positions.Add(new Point3D(-0.03 + separacion, 0.03, 0.3));
        mesh.Positions.Add(new Point3D(-0.03 + separacion, 0.16, 0.3));
        mesh.Positions.Add(new Point3D(-0.05 + separacion, 0.16, 0.3));

        mesh.Positions.Add(new Point3D(-0.05 + separacion, 0.03, 0.25));
        mesh.Positions.Add(new Point3D(-0.03 + separacion, 0.03, 0.25));
        mesh.Positions.Add(new Point3D(-0.03 + separacion, 0.16, 0.25));
        mesh.Positions.Add(new Point3D(-0.05 + separacion, 0.16, 0.25));


        for (int i = 0; i < 2; i++)//Formando triangulos en el espacio 3D.
        {
            // Cara frontal
            mesh.TriangleIndices.Add(0 + barras);
            mesh.TriangleIndices.Add(1 + barras);
            mesh.TriangleIndices.Add(2 + barras);
            mesh.TriangleIndices.Add(2 + barras);
            mesh.TriangleIndices.Add(3 + barras);
            mesh.TriangleIndices.Add(0 + barras);


            // Cara trasera
            mesh.TriangleIndices.Add(6 + barras);
            mesh.TriangleIndices.Add(5 + barras);
            mesh.TriangleIndices.Add(4 + barras);
            mesh.TriangleIndices.Add(4 + barras);
            mesh.TriangleIndices.Add(7 + barras);
            mesh.TriangleIndices.Add(6 + barras);

            // Cara lateral izquierda
            mesh.TriangleIndices.Add(1 + barras);
            mesh.TriangleIndices.Add(5 + barras);
            mesh.TriangleIndices.Add(2 + barras);
            mesh.TriangleIndices.Add(5 + barras);
            mesh.TriangleIndices.Add(6 + barras);
            mesh.TriangleIndices.Add(2 + barras);
```

```
        // Cara superior
        mesh.TriangleIndices.Add(2 + barras);
        mesh.TriangleIndices.Add(6 + barras);
        mesh.TriangleIndices.Add(3 + barras);
        mesh.TriangleIndices.Add(3 + barras);
        mesh.TriangleIndices.Add(6 + barras);
        mesh.TriangleIndices.Add(7 + barras);

        // Cara inferior
        mesh.TriangleIndices.Add(5 + barras);
        mesh.TriangleIndices.Add(1 + barras);
        mesh.TriangleIndices.Add(0 + barras);
        mesh.TriangleIndices.Add(0 + barras);
        mesh.TriangleIndices.Add(4 + barras);
        mesh.TriangleIndices.Add(5 + barras);

        // Cara lateral derecha
        mesh.TriangleIndices.Add(4 + barras);
        mesh.TriangleIndices.Add(0 + barras);
        mesh.TriangleIndices.Add(3 + barras);
        mesh.TriangleIndices.Add(3 + barras);
        mesh.TriangleIndices.Add(7 + barras);
        mesh.TriangleIndices.Add(4 + barras);
        barras += 8;
    }

    Color color = Color.FromArgb(200, 0, 0, 150);
    SolidColorBrush fillBrush = new SolidColorBrush(color);
    // Creación de la geometria
    mGeometry = new GeometryModel3D(mesh, new DiffuseMaterial(fillBrush));
    return mGeometry;
}

private GeometryModel3D Base()
{
    GeometryModel3D mGeometry = new GeometryModel3D();
    mesh = new MeshGeometry3D();

    //medida del brazo
    // Largo 45cm =  0    a    45    X
    // Alto  2cm  = -0.01 a  0.01    Y
    // Ancho 65cm  = 0    a    65    Z

    mesh.Positions.Add(new Point3D(0.0, -0.01, 0.0));
    mesh.Positions.Add(new Point3D(0.45, -0.01, 0.0));
    mesh.Positions.Add(new Point3D(0.45, 0.01, 0.0));
    mesh.Positions.Add(new Point3D(0.0, 0.01, 0.0));

    mesh.Positions.Add(new Point3D(0.0, -0.01, 0.65));
    mesh.Positions.Add(new Point3D(0.45, -0.01, 0.65));
    mesh.Positions.Add(new Point3D(0.45, 0.01, 0.65));
    mesh.Positions.Add(new Point3D(0.0, 0.01, 0.65));

    mesh.TriangleIndices.Add(0);
    mesh.TriangleIndices.Add(1);
    mesh.TriangleIndices.Add(2);
    mesh.TriangleIndices.Add(2);
    mesh.TriangleIndices.Add(3);
    mesh.TriangleIndices.Add(0);
```

```
    mesh.TriangleIndices.Add(4);
    mesh.TriangleIndices.Add(5);
    mesh.TriangleIndices.Add(6);
    mesh.TriangleIndices.Add(6);
    mesh.TriangleIndices.Add(7);
    mesh.TriangleIndices.Add(4);

    mesh.TriangleIndices.Add(5);
    mesh.TriangleIndices.Add(1);
    mesh.TriangleIndices.Add(2);
    mesh.TriangleIndices.Add(2);
    mesh.TriangleIndices.Add(6);
    mesh.TriangleIndices.Add(5);

    mesh.TriangleIndices.Add(2);
    mesh.TriangleIndices.Add(6);
    mesh.TriangleIndices.Add(3);
    mesh.TriangleIndices.Add(3);
    mesh.TriangleIndices.Add(6);
    mesh.TriangleIndices.Add(7);

    mesh.TriangleIndices.Add(5);
    mesh.TriangleIndices.Add(1);
    mesh.TriangleIndices.Add(0);
    mesh.TriangleIndices.Add(0);
    mesh.TriangleIndices.Add(4);
    mesh.TriangleIndices.Add(5);

    mesh.TriangleIndices.Add(4);
    mesh.TriangleIndices.Add(0);
    mesh.TriangleIndices.Add(3);
    mesh.TriangleIndices.Add(3);
    mesh.TriangleIndices.Add(7);
    mesh.TriangleIndices.Add(4);

    Color color = Color.FromRgb(20, 20, 20);
    SolidColorBrush fillBrush = new SolidColorBrush(color);
    // asignando nombre a la figura Geometrica
    mGeometry = new GeometryModel3D(mesh, new DiffuseMaterial(Cool()));
    return mGeometry;
}

private GeometryModel3D BaseSuperior()
{
    GeometryModel3D mGeometry = new GeometryModel3D();
    mesh = new MeshGeometry3D();
    double separation = 0.0;
    int barras = 0;

    //medida del brazo
    // Largo 45cm =  0    a    45    X
    // Alto  2cm  = -0.01 a  0.01    Y
    // Ancho 65cm  = 0    a    65    Z

    for (int i = 0; i < 8; i++)
    {
        mesh.Positions.Add(new Point3D(0.0 + separation, 0.10, 0.0));
        mesh.Positions.Add(new Point3D(0.03 + separation, 0.10, 0.0));
        mesh.Positions.Add(new Point3D(0.03 + separation, .11, 0.0));
        mesh.Positions.Add(new Point3D(0.0 + separation, 0.11, 0.0));
```

```
    mesh.Positions.Add(new Point3D(0.0 + separation, 0.10, 0.65));
    mesh.Positions.Add(new Point3D(0.03 + separation, 0.10, 0.65));
    mesh.Positions.Add(new Point3D(0.03 + separation, .11, 0.65));
    mesh.Positions.Add(new Point3D(0.0 + separation, 0.11, 0.65));

    separation += 0.06;
}

separation = 0.0;

for (int i = 0; i < 7; i++)
{
    mesh.Positions.Add(new Point3D(0.03 + separation, 0.10, 0.0));
    mesh.Positions.Add(new Point3D(0.06 + separation, 0.10, 0.0));
    mesh.Positions.Add(new Point3D(0.06 + separation, .105, 0.0));
    mesh.Positions.Add(new Point3D(0.03 + separation, 0.105, 0.0));

    mesh.Positions.Add(new Point3D(0.03 + separation, 0.10, 0.65));
    mesh.Positions.Add(new Point3D(0.06 + separation, 0.10, 0.65));
    mesh.Positions.Add(new Point3D(0.06 + separation, .105, 0.65));
    mesh.Positions.Add(new Point3D(0.03 + separation, 0.105, 0.65));

    separation += 0.06;
}

for (int i = 0; i < 15; i++)
{
    mesh.TriangleIndices.Add(0 + barras);
    mesh.TriangleIndices.Add(1 + barras);
    mesh.TriangleIndices.Add(2 + barras);
    mesh.TriangleIndices.Add(2 + barras);
    mesh.TriangleIndices.Add(3 + barras);
    mesh.TriangleIndices.Add(0 + barras);

    mesh.TriangleIndices.Add(6 + barras);
    mesh.TriangleIndices.Add(5 + barras);
    mesh.TriangleIndices.Add(4 + barras);
    mesh.TriangleIndices.Add(4 + barras);
    mesh.TriangleIndices.Add(7 + barras);
    mesh.TriangleIndices.Add(6 + barras);

    mesh.TriangleIndices.Add(1 + barras);
    mesh.TriangleIndices.Add(5 + barras);
    mesh.TriangleIndices.Add(2 + barras);
    mesh.TriangleIndices.Add(5 + barras);
    mesh.TriangleIndices.Add(6 + barras);
    mesh.TriangleIndices.Add(2 + barras);

    mesh.TriangleIndices.Add(2 + barras);
    mesh.TriangleIndices.Add(6 + barras);
    mesh.TriangleIndices.Add(3 + barras);
    mesh.TriangleIndices.Add(3 + barras);
    mesh.TriangleIndices.Add(6 + barras);
    mesh.TriangleIndices.Add(7 + barras);

    mesh.TriangleIndices.Add(5 + barras);
    mesh.TriangleIndices.Add(1 + barras);
    mesh.TriangleIndices.Add(0 + barras);
    mesh.TriangleIndices.Add(0 + barras);
    mesh.TriangleIndices.Add(4 + barras);
    mesh.TriangleIndices.Add(5 + barras);
```

```
            mesh.TriangleIndices.Add(4 + barras);
            mesh.TriangleIndices.Add(0 + barras);
            mesh.TriangleIndices.Add(3 + barras);
            mesh.TriangleIndices.Add(3 + barras);
            mesh.TriangleIndices.Add(7 + barras);
            mesh.TriangleIndices.Add(4 + barras);
            barras += 8;
        }

        mGeometry = new GeometryModel3D(mesh, new DiffuseMaterial(Brushes.Gray));
        return mGeometry;

    }

    private GeometryModel3D Brazos()
    {
        GeometryModel3D mGeometry = new GeometryModel3D();
        double separacion = 0.481;
        mesh = new MeshGeometry3D();
        int barras = 0;

        mesh.Positions.Add(new Point3D(-0.03, 0.06, 0.3));
        mesh.Positions.Add(new Point3D(-0.001, 0.06, 0.3));

        mesh.Positions.Add(new Point3D(-0.001, 0.36, 0.3));
        mesh.Positions.Add(new Point3D(-0.03, 0.36, 0.3));

        mesh.Positions.Add(new Point3D(-0.03, 0.06, 0.25));
        mesh.Positions.Add(new Point3D(-0.001, 0.06, 0.25));
        mesh.Positions.Add(new Point3D(-0.001, 0.36, 0.25));
        mesh.Positions.Add(new Point3D(-0.03, 0.36, 0.25));


        mesh.Positions.Add(new Point3D(-0.03 + separacion, 0.06, 0.3));
        mesh.Positions.Add(new Point3D(0 + separacion, 0.06, 0.3));
        mesh.Positions.Add(new Point3D(0 + separacion, 0.36, 0.3));
        mesh.Positions.Add(new Point3D(-0.03 + separacion, 0.36, 0.3));

        mesh.Positions.Add(new Point3D(-0.03 + separacion, 0.06, 0.25));
        mesh.Positions.Add(new Point3D(0 + separacion, 0.06, 0.25));
        mesh.Positions.Add(new Point3D(0 + separacion, 0.36, 0.25));
        mesh.Positions.Add(new Point3D(-0.03 + separacion, 0.36, 0.25));

        for (int i = 0; i < 2; i++)
        {
            mesh.TriangleIndices.Add(0 + barras);
            mesh.TriangleIndices.Add(1 + barras);
            mesh.TriangleIndices.Add(2 + barras);
            mesh.TriangleIndices.Add(2 + barras);
            mesh.TriangleIndices.Add(3 + barras);
            mesh.TriangleIndices.Add(0 + barras);

            mesh.TriangleIndices.Add(6 + barras);
            mesh.TriangleIndices.Add(5 + barras);
            mesh.TriangleIndices.Add(4 + barras);
            mesh.TriangleIndices.Add(4 + barras);
            mesh.TriangleIndices.Add(7 + barras);
            mesh.TriangleIndices.Add(6 + barras);

            mesh.TriangleIndices.Add(1 + barras);
            mesh.TriangleIndices.Add(5 + barras);
            mesh.TriangleIndices.Add(2 + barras);
```

```
            mesh.TriangleIndices.Add(5 + barras);
            mesh.TriangleIndices.Add(6 + barras);
            mesh.TriangleIndices.Add(2 + barras);

            mesh.TriangleIndices.Add(2 + barras);
            mesh.TriangleIndices.Add(6 + barras);
            mesh.TriangleIndices.Add(3 + barras);
            mesh.TriangleIndices.Add(3 + barras);
            mesh.TriangleIndices.Add(6 + barras);
            mesh.TriangleIndices.Add(7 + barras);

            mesh.TriangleIndices.Add(5 + barras);
            mesh.TriangleIndices.Add(1 + barras);
            mesh.TriangleIndices.Add(0 + barras);
            mesh.TriangleIndices.Add(0 + barras);
            mesh.TriangleIndices.Add(4 + barras);
            mesh.TriangleIndices.Add(5 + barras);

            mesh.TriangleIndices.Add(4 + barras);
            mesh.TriangleIndices.Add(0 + barras);
            mesh.TriangleIndices.Add(3 + barras);
            mesh.TriangleIndices.Add(3 + barras);
            mesh.TriangleIndices.Add(7 + barras);
            mesh.TriangleIndices.Add(4 + barras);
            barras += 8;
        }


        Color color = Color.FromArgb(150, 0, 0, 200);
        SolidColorBrush fillBrush = new SolidColorBrush(color);
        // asignando nombre a la figura Geometrica
        mGeometry = new GeometryModel3D(mesh, new DiffuseMaterial(fillBrush));
        return mGeometry;
    }

    private GeometryModel3D PortaDuya()
    {
        GeometryModel3D mGeometry = new GeometryModel3D();
        mesh = new MeshGeometry3D();

        mesh.Positions.Add(new Point3D(0.1, 0.26, 0.28));
        mesh.Positions.Add(new Point3D(0.15, 0.26, 0.28));
        mesh.Positions.Add(new Point3D(0.15, 0.355, 0.28));
        mesh.Positions.Add(new Point3D(0.1, 0.355, 0.28));

        mesh.Positions.Add(new Point3D(0.1, 0.26, 0.25));
        mesh.Positions.Add(new Point3D(0.15, 0.26, 0.25));
        mesh.Positions.Add(new Point3D(0.15, 0.355, 0.25));
        mesh.Positions.Add(new Point3D(0.1, 0.355, 0.25));

        mesh.Positions.Add(new Point3D(0.123, 0.20, 0.284));
        mesh.Positions.Add(new Point3D(0.127, 0.20, 0.284));
        mesh.Positions.Add(new Point3D(0.127, 0.33, 0.284));
        mesh.Positions.Add(new Point3D(0.123, 0.33, 0.284));

        mesh.Positions.Add(new Point3D(0.123, 0.20, 0.28));
        mesh.Positions.Add(new Point3D(0.127, 0.20, 0.28));
        mesh.Positions.Add(new Point3D(0.127, 0.33, 0.28));
        mesh.Positions.Add(new Point3D(0.123, 0.33, 0.28));
```

```
    for(int i = 0; i<9; i += 8)
       {
    mesh.TriangleIndices.Add(0 + i);
    mesh.TriangleIndices.Add(1 + i);
    mesh.TriangleIndices.Add(2 + i);
    mesh.TriangleIndices.Add(2 + i);
    mesh.TriangleIndices.Add(3 + i);
    mesh.TriangleIndices.Add(0 + i);

    mesh.TriangleIndices.Add(6 + i);
    mesh.TriangleIndices.Add(5 + i);
    mesh.TriangleIndices.Add(4 + i);
    mesh.TriangleIndices.Add(4 + i);
    mesh.TriangleIndices.Add(7 + i);
    mesh.TriangleIndices.Add(6 + i);

    mesh.TriangleIndices.Add(1 + i);
    mesh.TriangleIndices.Add(5 + i);
    mesh.TriangleIndices.Add(2 + i);
    mesh.TriangleIndices.Add(5 + i);
    mesh.TriangleIndices.Add(6 + i);
    mesh.TriangleIndices.Add(2 + i);

    mesh.TriangleIndices.Add(2 + i);
    mesh.TriangleIndices.Add(6 + i);
    mesh.TriangleIndices.Add(3 + i);
    mesh.TriangleIndices.Add(3 + i);
    mesh.TriangleIndices.Add(6 + i);
    mesh.TriangleIndices.Add(7 + i);

    mesh.TriangleIndices.Add(5 + i);
    mesh.TriangleIndices.Add(1 + i);
    mesh.TriangleIndices.Add(0 + i);
    mesh.TriangleIndices.Add(0 + i);
    mesh.TriangleIndices.Add(4 + i);
    mesh.TriangleIndices.Add(5 + i);

    mesh.TriangleIndices.Add(4 + i);
    mesh.TriangleIndices.Add(0 + i);
    mesh.TriangleIndices.Add(3 + i);
    mesh.TriangleIndices.Add(3 + i);
    mesh.TriangleIndices.Add(7 + i);
    mesh.TriangleIndices.Add(4 + i);
    }
   mGeometry = new GeometryModel3D(mesh, new DiffuseMaterial(Brushes.Cornsilk));
   return mGeometry;
}

private GeometryModel3D Postes()
{
    GeometryModel3D mGeometry = new GeometryModel3D();
    mesh = new MeshGeometry3D();

    mesh.Positions.Add(new Point3D(0.0, 0.01, 0.0));
    mesh.Positions.Add(new Point3D(0.45, 0.01, 0.0));
    mesh.Positions.Add(new Point3D(0.45, .10, 0.0));
    mesh.Positions.Add(new Point3D(0.0, 0.10, 0.0));

    mesh.Positions.Add(new Point3D(0.0, 0.01, 0.02));
    mesh.Positions.Add(new Point3D(0.45, 0.01, 0.02));
    mesh.Positions.Add(new Point3D(0.45, .10, 0.02));
    mesh.Positions.Add(new Point3D(0.0, 0.10, 0.02));
```

```
    mesh.Positions.Add(new Point3D(0.0, 0.01, 0.63));
    mesh.Positions.Add(new Point3D(0.45, 0.01, 0.63));
    mesh.Positions.Add(new Point3D(0.45, .10, 0.63));
    mesh.Positions.Add(new Point3D(0.0, 0.10, 0.63));

    mesh.Positions.Add(new Point3D(0.0, 0.01, 0.65));
    mesh.Positions.Add(new Point3D(0.45, 0.01, 0.65));
    mesh.Positions.Add(new Point3D(0.45, .10, 0.65));
    mesh.Positions.Add(new Point3D(0.0, 0.10, 0.65));

     for(int i = 0; i<9; i += 8)
      {
    mesh.TriangleIndices.Add(0 + i);
    mesh.TriangleIndices.Add(1 + i);
    mesh.TriangleIndices.Add(2 + i);
    mesh.TriangleIndices.Add(2 + i);
    mesh.TriangleIndices.Add(3 + i);
    mesh.TriangleIndices.Add(0 + i);

    mesh.TriangleIndices.Add(6 + i);
    mesh.TriangleIndices.Add(5 + i);
    mesh.TriangleIndices.Add(4 + i);
    mesh.TriangleIndices.Add(4 + i);
    mesh.TriangleIndices.Add(7 + i);
    mesh.TriangleIndices.Add(6 + i);

    mesh.TriangleIndices.Add(1 + i);
    mesh.TriangleIndices.Add(5 + i);
    mesh.TriangleIndices.Add(2 + i);
    mesh.TriangleIndices.Add(5 + i);
    mesh.TriangleIndices.Add(6 + i);
    mesh.TriangleIndices.Add(2 + i);

    mesh.TriangleIndices.Add(2 + i);
    mesh.TriangleIndices.Add(6 + i);
    mesh.TriangleIndices.Add(3 + i);
    mesh.TriangleIndices.Add(3 + i);
    mesh.TriangleIndices.Add(6 + i);
    mesh.TriangleIndices.Add(7 + i);

    mesh.TriangleIndices.Add(5 + i);
    mesh.TriangleIndices.Add(1 + i);
    mesh.TriangleIndices.Add(0 + i);
    mesh.TriangleIndices.Add(0 + i);
    mesh.TriangleIndices.Add(4 + i);
    mesh.TriangleIndices.Add(5 + i);

    mesh.TriangleIndices.Add(4 + i);
    mesh.TriangleIndices.Add(0 + i);
    mesh.TriangleIndices.Add(3 + i);
    mesh.TriangleIndices.Add(3 + i);
    mesh.TriangleIndices.Add(7 + i);
    mesh.TriangleIndices.Add(4 + i);
    }
    mGeometry = new GeometryModel3D(mesh, new DiffuseMaterial(Brushes.Red));
    return mGeometry;
}

private GeometryModel3D Prismas()
{
    GeometryModel3D mGeometry = new GeometryModel3D();
    mesh = new MeshGeometry3D();
```

```
    double xmed = 0.45 / 4;
    double xmed2 = 0.45 - (0.45 / 4);
    //puntos primera guia
    mesh.Positions.Add(new Point3D(xmed, 0.011, 0.03));
    mesh.Positions.Add(new Point3D(xmed + 0.02, 0.011, 0.03));
    mesh.Positions.Add(new Point3D(xmed + 0.01, 0.03, 0.03));

    mesh.Positions.Add(new Point3D(xmed, 0.011, 0.62));
    mesh.Positions.Add(new Point3D(xmed + 0.02, 0.011, 0.62));
    mesh.Positions.Add(new Point3D(xmed + 0.01, 0.03, 0.62));

    mesh.Positions.Add(new Point3D(xmed2, 0.011, 0.03));
    mesh.Positions.Add(new Point3D(xmed2 + 0.02, 0.011, 0.03));
    mesh.Positions.Add(new Point3D(xmed2 + 0.01, 0.03, 0.03));

    mesh.Positions.Add(new Point3D(xmed2, 0.011, 0.62));
    mesh.Positions.Add(new Point3D(xmed2 + 0.02, 0.011, 0.62));
    mesh.Positions.Add(new Point3D(xmed2 + 0.01, 0.03, 0.62));

     for(int i = 0; i<9; i+= 6 )
        {
    mesh.TriangleIndices.Add(0 + i);
    mesh.TriangleIndices.Add(1 + i);
    mesh.TriangleIndices.Add(2 + i);

    mesh.TriangleIndices.Add(3 + i);
    mesh.TriangleIndices.Add(4 + i);
    mesh.TriangleIndices.Add(5 + i);

    mesh.TriangleIndices.Add(2 + i);
    mesh.TriangleIndices.Add(5 + i);
    mesh.TriangleIndices.Add(1 + i);

    mesh.TriangleIndices.Add(5 + i);
    mesh.TriangleIndices.Add(4 + i);
    mesh.TriangleIndices.Add(1 + i);

    mesh.TriangleIndices.Add(2 + i);
    mesh.TriangleIndices.Add(0 + i);
    mesh.TriangleIndices.Add(3 + i);

    mesh.TriangleIndices.Add(3 + i);
    mesh.TriangleIndices.Add(5 + i);
    mesh.TriangleIndices.Add(2 + i);

    mesh.TriangleIndices.Add(0 + i);
    mesh.TriangleIndices.Add(1 + i);
    mesh.TriangleIndices.Add(3 + i);

    mesh.TriangleIndices.Add(3 + i);
    mesh.TriangleIndices.Add(1 + i);
    mesh.TriangleIndices.Add(4 + i);
   }

    mGeometry = new GeometryModel3D(mesh, new DiffuseMaterial(Brushes.DarkRed));
    return mGeometry;
}

private GeometryModel3D SoporteBrazo()
{
    GeometryModel3D mGeometry = new GeometryModel3D();
    MeshGeometry3D mesh = new MeshGeometry3D();
```

```
    //medida del brazo
    // Largo 51cm = 0.48 a -0.03   X
    // Alto  3cm  = 0.06 a  0.03   Y
    // Ancho 5cm  = 0.30 a  0.25   Z

    mesh.Positions.Add(new Point3D(-0.03, 0.03, 0.3));
    mesh.Positions.Add(new Point3D(0.48, 0.03, 0.3));
    mesh.Positions.Add(new Point3D(0.48, 0.06, 0.3));
    mesh.Positions.Add(new Point3D(-0.03, 0.06, 0.3));

    mesh.Positions.Add(new Point3D(-0.03, 0.03, 0.25));
    mesh.Positions.Add(new Point3D(0.48, 0.03, 0.25));
    mesh.Positions.Add(new Point3D(0.48, 0.06, 0.25));
    mesh.Positions.Add(new Point3D(-0.03, 0.06, 0.25));

    mesh.TriangleIndices.Add(0);
    mesh.TriangleIndices.Add(1);
    mesh.TriangleIndices.Add(2);
    mesh.TriangleIndices.Add(2);
    mesh.TriangleIndices.Add(3);
    mesh.TriangleIndices.Add(0);

    mesh.TriangleIndices.Add(6);
    mesh.TriangleIndices.Add(5);
    mesh.TriangleIndices.Add(4);
    mesh.TriangleIndices.Add(4);
    mesh.TriangleIndices.Add(7);
    mesh.TriangleIndices.Add(6);

    mesh.TriangleIndices.Add(1);
    mesh.TriangleIndices.Add(5);
    mesh.TriangleIndices.Add(2);
    mesh.TriangleIndices.Add(5);
    mesh.TriangleIndices.Add(6);
    mesh.TriangleIndices.Add(2);

    mesh.TriangleIndices.Add(2);
    mesh.TriangleIndices.Add(6);
    mesh.TriangleIndices.Add(3);
    mesh.TriangleIndices.Add(3);
    mesh.TriangleIndices.Add(6);
    mesh.TriangleIndices.Add(7);

    mesh.TriangleIndices.Add(5);
    mesh.TriangleIndices.Add(1);
    mesh.TriangleIndices.Add(0);
    mesh.TriangleIndices.Add(0);
    mesh.TriangleIndices.Add(4);
    mesh.TriangleIndices.Add(5);

    mesh.TriangleIndices.Add(4);
    mesh.TriangleIndices.Add(0);
    mesh.TriangleIndices.Add(3);
    mesh.TriangleIndices.Add(3);
    mesh.TriangleIndices.Add(7);
    mesh.TriangleIndices.Add(4);

    mGeometry = new GeometryModel3D(mesh, new DiffuseMaterial(Brushes.BurlyWood));
    return mGeometry;
}
```

```
private GeometryModel3D SoporteCabezal()
{
   GeometryModel3D mGeometry = new GeometryModel3D();
   mesh = new MeshGeometry3D();

   mesh.Positions.Add(new Point3D(-0.03, 0.31, 0.25));
   mesh.Positions.Add(new Point3D(0.48, 0.31, 0.25));
   mesh.Positions.Add(new Point3D(0.48, 0.36, 0.25));
   mesh.Positions.Add(new Point3D(-0.03, 0.36, 0.25));

   mesh.Positions.Add(new Point3D(-0.03, 0.31, 0.23));
   mesh.Positions.Add(new Point3D(0.48, 0.31, 0.23));
   mesh.Positions.Add(new Point3D(0.48, 0.36, 0.23));
   mesh.Positions.Add(new Point3D(-0.03, 0.36, 0.23));

   mesh.TriangleIndices.Add(0);
   mesh.TriangleIndices.Add(1);
   mesh.TriangleIndices.Add(2);
   mesh.TriangleIndices.Add(2);
   mesh.TriangleIndices.Add(3);
   mesh.TriangleIndices.Add(0);

   mesh.TriangleIndices.Add(6);
   mesh.TriangleIndices.Add(5);
   mesh.TriangleIndices.Add(4);
   mesh.TriangleIndices.Add(4);
   mesh.TriangleIndices.Add(7);
   mesh.TriangleIndices.Add(6);

   mesh.TriangleIndices.Add(1);
   mesh.TriangleIndices.Add(5);
   mesh.TriangleIndices.Add(2);
   mesh.TriangleIndices.Add(5);
   mesh.TriangleIndices.Add(6);
   mesh.TriangleIndices.Add(2);

   mesh.TriangleIndices.Add(2);
   mesh.TriangleIndices.Add(6);
   mesh.TriangleIndices.Add(3);
   mesh.TriangleIndices.Add(3);
   mesh.TriangleIndices.Add(6);
   mesh.TriangleIndices.Add(7);

   mesh.TriangleIndices.Add(5);
   mesh.TriangleIndices.Add(1);
   mesh.TriangleIndices.Add(0);
   mesh.TriangleIndices.Add(0);
   mesh.TriangleIndices.Add(4);
   mesh.TriangleIndices.Add(5);

   mesh.TriangleIndices.Add(4);
   mesh.TriangleIndices.Add(0);
   mesh.TriangleIndices.Add(3);
   mesh.TriangleIndices.Add(3);
   mesh.TriangleIndices.Add(7);
   mesh.TriangleIndices.Add(4);

   mGeometry = new GeometryModel3D(mesh, new DiffuseMaterial(Brushes.BurlyWood));
   return mGeometry;
}

private GeometryModel3D TornilloCabezal()
```

```
{
    GeometryModel3D mGeometry = new GeometryModel3D();
    mesh = new MeshGeometry3D();

    mesh.Positions.Add(new Point3D(-0.02, 0.33, 0.27));
    mesh.Positions.Add(new Point3D(0.47, 0.33, 0.27));
    mesh.Positions.Add(new Point3D(0.47, 0.34, 0.27));
    mesh.Positions.Add(new Point3D(-0.02, 0.34, 0.27));

    mesh.Positions.Add(new Point3D(-0.02, 0.33, 0.26));
    mesh.Positions.Add(new Point3D(0.47, 0.33, 0.26));
    mesh.Positions.Add(new Point3D(0.47, 0.34, 0.26));
    mesh.Positions.Add(new Point3D(-0.02, 0.34, 0.26));

    mesh.Positions.Add(new Point3D(-0.02, 0.32, 0.265));
    mesh.Positions.Add(new Point3D(0.47, 0.32, 0.265));
    mesh.Positions.Add(new Point3D(0.47, 0.325, 0.265));
    mesh.Positions.Add(new Point3D(-0.02, 0.325, 0.265));

    mesh.Positions.Add(new Point3D(-0.02, 0.32, 0.26));
    mesh.Positions.Add(new Point3D(0.47, 0.32, 0.26));
    mesh.Positions.Add(new Point3D(0.47, 0.325, 0.26));
    mesh.Positions.Add(new Point3D(-0.02, 0.325, 0.26));

    for (int i = 0; i < 2; i++)
    {
        mesh.TriangleIndices.Add(0 + (i * 8));
        mesh.TriangleIndices.Add(1 + (i * 8));
        mesh.TriangleIndices.Add(2 + (i * 8));
        mesh.TriangleIndices.Add(2 + (i * 8));
        mesh.TriangleIndices.Add(3 + (i * 8));
        mesh.TriangleIndices.Add(0 + (i * 8));

        mesh.TriangleIndices.Add(6 + (i * 8));
        mesh.TriangleIndices.Add(5 + (i * 8));
        mesh.TriangleIndices.Add(4 + (i * 8));
        mesh.TriangleIndices.Add(4 + (i * 8));
        mesh.TriangleIndices.Add(7 + (i * 8));
        mesh.TriangleIndices.Add(6 + (i * 8));

        mesh.TriangleIndices.Add(1 + (i * 8));
        mesh.TriangleIndices.Add(5 + (i * 8));
        mesh.TriangleIndices.Add(2 + (i * 8));
        mesh.TriangleIndices.Add(5 + (i * 8));
        mesh.TriangleIndices.Add(6 + (i * 8));
        mesh.TriangleIndices.Add(2 + (i * 8));

        mesh.TriangleIndices.Add(2 + (i * 8));
        mesh.TriangleIndices.Add(6 + (i * 8));
        mesh.TriangleIndices.Add(3 + (i * 8));
        mesh.TriangleIndices.Add(3 + (i * 8));
        mesh.TriangleIndices.Add(6 + (i * 8));
        mesh.TriangleIndices.Add(7 + (i * 8));

        mesh.TriangleIndices.Add(5 + (i * 8));
        mesh.TriangleIndices.Add(1 + (i * 8));
        mesh.TriangleIndices.Add(0 + (i * 8));
        mesh.TriangleIndices.Add(0 + (i * 8));
        mesh.TriangleIndices.Add(4 + (i * 8));
        mesh.TriangleIndices.Add(5 + (i * 8));

        mesh.TriangleIndices.Add(4 + (i * 8));
```

```
            mesh.TriangleIndices.Add(0 + (i * 8));
            mesh.TriangleIndices.Add(3 + (i * 8));
            mesh.TriangleIndices.Add(3 + (i * 8));
            mesh.TriangleIndices.Add(7 + (i * 8));
            mesh.TriangleIndices.Add(4 + (i * 8));
        }

        mGeometry = new GeometryModel3D(mesh, new DiffuseMaterial(Brushes.DarkGray));
        return mGeometry;
    }
```

Parte III – Manipulación de objetos por Reloj.

```
private void timer_Tick(object sender, EventArgs e)
    {
        //timer.delay --> usar para cambiar el tiempo.
        if (contPulsos < DataPort.Count)//aumentar la velocidad con suma 4.
        {

            #region Decodificacion Binaria simple
            //------Codificación de Datos……
            #endregion

                    TranslateTransform3D tbrazo = new TranslateTransform3D(new Vector3D(0, 0,-
                (double)increX));
            Transform3DGroup tgroup = new Transform3DGroup();

                    TranslateTransform3D tDuya = new TranslateTransform3D(new Vector3D(
                    (double)increY,0,-(double)increX));
            Transform3DGroup tgroup_1 = new Transform3DGroup();

            tgroup.Children.Add(tbrazo);
            Brazos.Transform = tgroup;
            AnclaBrazos.Transform = tgroup;
            SoporteBrazo.Transform = tgroup;
            SoporteCabezal.Transform = tgroup;
            TornilloCabezal.Transform = tgroup;

            tgroup_1.Children.Add(tDuya);
            PortaDuya.Transform = tgroup_1;

            if (activeInyectora) //Dibuja si la inyectora esta activa.
            {
                ModelVisual3D extraCube = new ModelVisual3D();
                Model3DGroup modelGroup = new Model3DGroup();
                GeometryModel3D model3D = new GeometryModel3D();

                model3D = (GeometryModel3D)fluido(increY, increX);
                modelGroup.Children.Add(model3D);
                extraCube.Content = modelGroup;
                mainViewport.Children.Add(extraCube);
            }
        }
        else  //Detiene el reloj cuando se han termiando los datos.
        {
            timer.Stop();
        }

        contPulsos++;
    }
```

Anexos

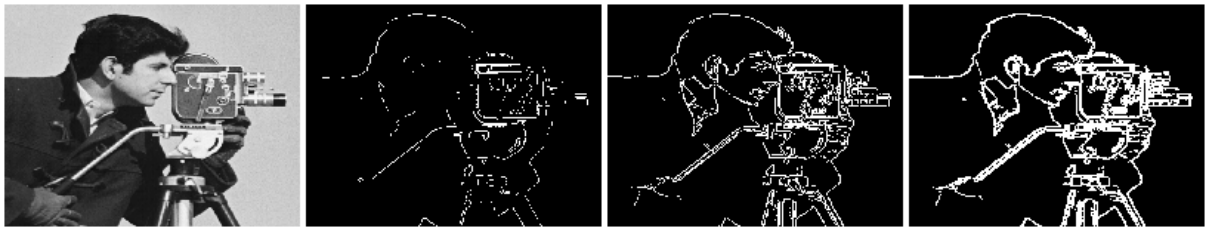
Anexo A Fichas Técnicas.

Ficha de Medidas de Charolas de Plástico proporcionada por MultiEmpac SA. De CV.

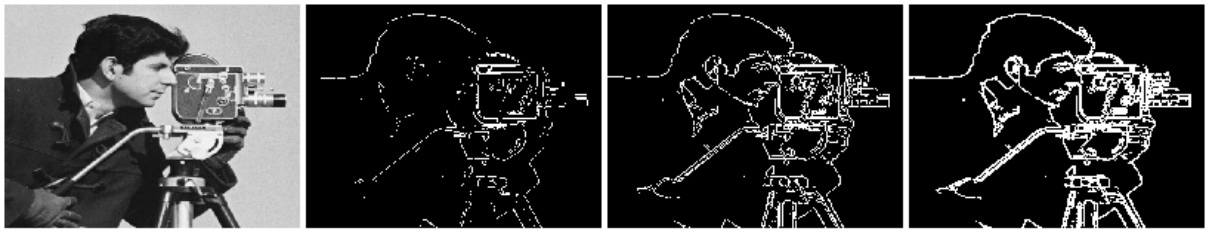
Charolas de Plástico Rectangulares			Charolas de Plástico Redondas		
					
Alto 1-2 cm	Largo 21-53 cm	Ancho 16-41 cm	Alto 1-2 cm	Largo 25-41 cm	Ancho 25-41 cm
Calibre -20-22	Capacidad -		Calibre -	Capacidad - 20-22	
Presentación pza			Presentación pza		

Anexo B. Evaluación de Resultados sobre la imagen utilizando diversos Filtros.

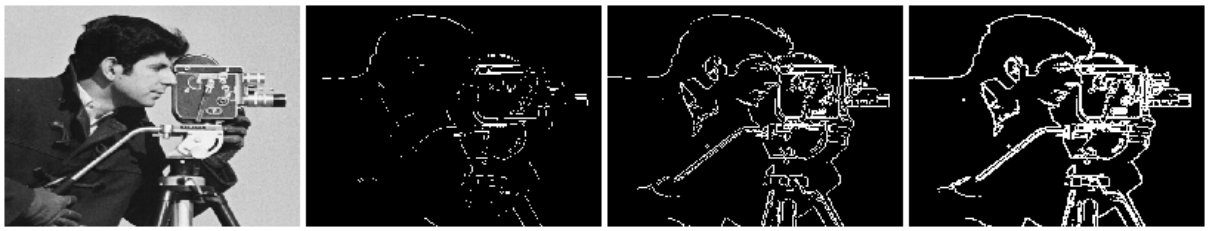
Prewitt



Sobel



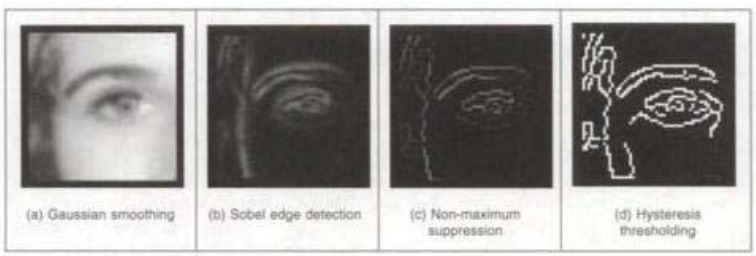
Frie Chen



Canny

Recorte del libro: Feature Extraction & Image Processing

1. use Gaussian smoothing (as in Section 3.4.4), Figure **4.15**(a);
2. use the Sobel operator, Figure **4.15**(b);
3. use non-maximal suppression, Figure **4.15**(c);
4. threshold with hysteresis to connect edge points, Figure **4.15**(d).



Resultado de la imagen al usar el método de Canny para la detección de contornos

BLIBLIOGRAFIA

C# La biblia
Editorial Anaya.

GDI+Custom Controls whit Visual C#2005
Editorial: Packt.Publishing
www.packtpub.com

Practical WPF Graphics Programming
Editorial: *Jack Xu, Ph.D*

Feature Extraction & Image Processing
Mark Nixon & Alberto Aguado
Editorial: Newnes
Paginas: 103-121.

Tratamiento digital de Imágenes.
Addison-Wesley/Diaz de Santos
Paginas: 43 - 51, 109 - 121, 339, 447 – 495.

Computer imaging: digital image analysis and processing.
Scott E. Umbaugh – 2005.

Pymes y CNC en Mexico.
8º CONGRESO IBEROAMERICANO DE INGENIERIA MECANICA
Tecnología de CNC a bajo costo para la automatización de la pequeña y mediana empresa de países en vías de desarrollo.

MSDN Microsotf Refences.

Matematicas avanzadas para Ingenieria/ Advance Mathematics For Engineering
de Peter V. Oneil - 2008
- Página 123

URLs

UNFILLER
http://www.unifiller.com/
Product Guide → Depositors cake.

Artsoft MACH 3
http://www.machsupport.com/

CNC - PUERTO PARALELO.
http://r-luis.xbot.es/cnc/electro01.html

CNC CATALOGO.
http://www.frs-cnc.com/

PROYECTOR CATALOGO.
http://www.walkers-online-sales.com/kopykake_k1000_projector.php

Elaboración de pasteles.
http://saearthurhouse.homestead.com/ACEalbum.html
http://saearthurhouse.homestead.com/cake_decorating_3.JPG
http://saearthurhouse.homestead.com/cake_decorating.JPG
http://www.bakingandbakingscience.com/dec.htm Dedicada a informar como se decoran pasteles.
http://www.msd.k12.mo.us/vocational/Culinary%20Arts/bakery02-03.htm