TRABAJO ESPECIAL DE GRADO

Diseño de código computacional para la simulación de procesos de corte en el Torno Ernault Somua 500

Presentado ante la ilustre
Universidad Central de Venezuela
Por el Br. Yégüez Montaño,
Wilfredo José,
Para optar al Título de
Ingeniero Mecánico

Caracas, 2001.

Agradecimientos

Al Profesor Ing. José Gregorio La Riva por tener la osadía de asumir la guía de este trabajo y al Ing. Ángel Rondón quien propuso e inició esta idea en la Escuela de Ingeniería Mecánica de la Facultad de Ingeniería de Universidad Central de Venezuela.

Sin el valioso aporte de mis amigos los técnicos y empleados de esta Escuela en especial el Técnico Erlis Araque y el Sr. Hernández que siempre con sus consejos y observaciones que siempre vienen al caso me ayudaron en momentos bien difíciles. Gracias de verdad.

Erasmo Clemente, Antonio Acosta, Enzo Molina. Juntos pasamos las verdes y las maduras dándonos ánimo y riendo de los propios errores. Les debo bastante, eso lo sabe bien el negro.

Yuby, se supone que uno debe agradecer cuando ya no hay más favores y ya no se necesita mas apoyo porque las dificultades se han superado. Pero contigo no se cómo hacer porque se que aunque ya terminé este trabajo, mi vida te debe y deberá aún más. Gracias por acompañarme todo este trecho.

Rigoberto y Charito este logro lento pero seguro es bastante de ustedes, gracias por tener tantas ganas de luchar y por todas las cosas que me han dado incluido el orgullo de ser parte de sus vidas.

Yégüez M., Wilfredo J.,

Diseño de código computacional para la simulación de procesos de corte en el Torno Ernault Somua 500

Tutor Académico: Prof. Ing. José G. La Riva. Tesis. Caracas, U.C.V. Facultad de Ingeniería. Escuela de Ingeniería Mecánica. 2001. 103 pág.

Máquinas – Herramienta, Control Numérico, Mecanizado

Desarrollo de una herramienta computacional para la simulación de procesos de corte en el torno a control numérico marca Ernault Somua 600 que se encuentra instalado y operativo en la Escuela de Ingeniería Mecánica. El sistema está orientado al sistema Operativo Windows y su diseño se basa en el modelo de programación fundamentado en Objetos. Basados en el entorno de programación Visual C++ 6.0 contenido en la suite de programación Microsoft Visual Studio 6.0, se diseña y construye una aplicación que brinda la posibilidad de editar el código NC que luego se ejecuta en el simulador presentando una retroalimentación visual del proceso. La aplicación permite al usuario dimensionar los elementos principales del proceso como lo son La pieza en su geometría original y las diferentes herramientas de corte involucradas. Adicionalmente cuenta con una interfaz de comunicación vía puerto paralelo DB – 25 para la transferencia de datos entre una PC y el bus de datos del control numérico durante la ejecución real del proceso.

Indice

Introducción.

La máquina herramienta juega un papel muy importante en el desarrollo tecnológico toda vez que con estos equipos se pueden fabricar elementos constructivos que sin su aplicación sería imposible lograrlos. Los grandes sistemas de producción requieren optimizar sus procesos en la medida que se tornan mas complejos. En este sentido, la búsqueda se ha orientado hacia la posibilidad de realizar diferentes operaciones tales como fresado, mandrinado y perforado en una sola máquina eliminando de una vez el cambio y desmontaje de piezas y herramientas y mejorando la precisión al mismo tiempo. Productividad, precisión, rapidez y flexibilidad son las variables de producción que se pretenden mejorar cada vez más con el uso de dispositivos automáticos en los lugares de fabricación. El control numérico fue ideado para dar respuesta a esta necesidad. Sin embargo, la utilización del control numérico ha provocado grandes cambios en amplios sectores de la industria. Se requiere un cambio en la mentalidad de los operarios acostumbrados a las máquinas convencionales.

El control numérico se ha implementado en numerosos procesos de producción hasta el punto que algunos no se podrían ejecutar sin aplicar este método de control. En nuestro país, este hecho es mas común a medida que el parque industrial se incrementa por lo que se hace necesario la existencia de personal calificado en esta área. El proceso de formación de operarios de estos costosos equipos implica la ejecución de tareas aplicando los conceptos y maquinaria relacionada. El error por impericia es bastante

probable y el deterioro o pérdida total sería una consecuencia evidente en estos casos.

La simulación surge como alternativa junto al desarrollo de técnicas gráficas aplicadas al computador que ayudan no solo en la enseñanza sino también en el desarrollo de rutinas y procesos de fabricación de piezas complejas. A esto último se le conoce como *CAM (Computer Aided Manufacturing)* y se trata de la aplicación de técnicas de CAD para la fabricación de piezas en máquinas –herramienta.

La Escuela de Ingeniería Mecánica de la Facultad de Ingeniería de la Universidad Central de Venezuela cuenta con dos equipos de Control Numérico de los cuales el Torno H. Ernault – Somua 600 es el objeto de este trabajo. Luego de habérsele incorporado una interfaz de comunicación vía PC, la Escuela de Ingeniería Mecánica adquirió un paquete CAM (SMART CAM). Este software permite el diseño y simulación del proceso de fabricación de una pieza a partir de un dibujo en CAD que es procesado y como salida, se obtiene la rutina en código NC. Esta rutina es simulada y representada gráficamente en pantalla por el simulador hasta tanto el código no sea editado para introducir cambios que el operador considere pertinentes. Al introducir modificaciones en el código generado por la aplicación, es imposible acceder al módulo de simulación con esta rutina editada.

Esta es la principal motivación de este trabajo, generar un software que permita la edición y la simulación de ejecución de una rutina NC de

manera interactiva. Los beneficios que se obtienen de este trabajo además de compensar la deficiencia principal de SMART-CAM es que se obtiene una herramienta de amplio uso en la enseñanza y entrenamiento de operarios sin poner en peligro la integridad de la máquina.

1.- Marco Teórico

1.1.- El Torno a Control Numérico

1.1.1.- Arquitectura General de un Control Numérico.

Un equipo de control numérico es un automatismo que ejecuta una secuencia de operaciones programadas por el operador del equipo. Las acciones se reducen a la realización de movimientos de elementos mecánicos accionados por motores en sucesión según un código de programa que es leído por el equipo y guardado en memoria. Cada línea de programa es tomada de la memoria e interpretada paso a paso para su ejecución en la sucesión prescrita. Esta estructura es similar a la de un computador personal en la que se pueden ver una Unidad de Entrada, una Unidad de Memoria, una Unidad Central de Procesamiento y una Unidad de Salida, siendo la unidad Central de procesamiento la que sen encarga de los cálculos y toma de decisiones lógicas para la ejecución de acciones por parte del equipo de control numérico. Para ilustrar esta organización, se puede ver en la página siguiente, en la Figura Nro. 1, un esquema propuesto por Alique (1981) en el que se presenta un panorama general de un equipo de control numérico y que en esencia hasta nuestros días no ha cambiado mucho.

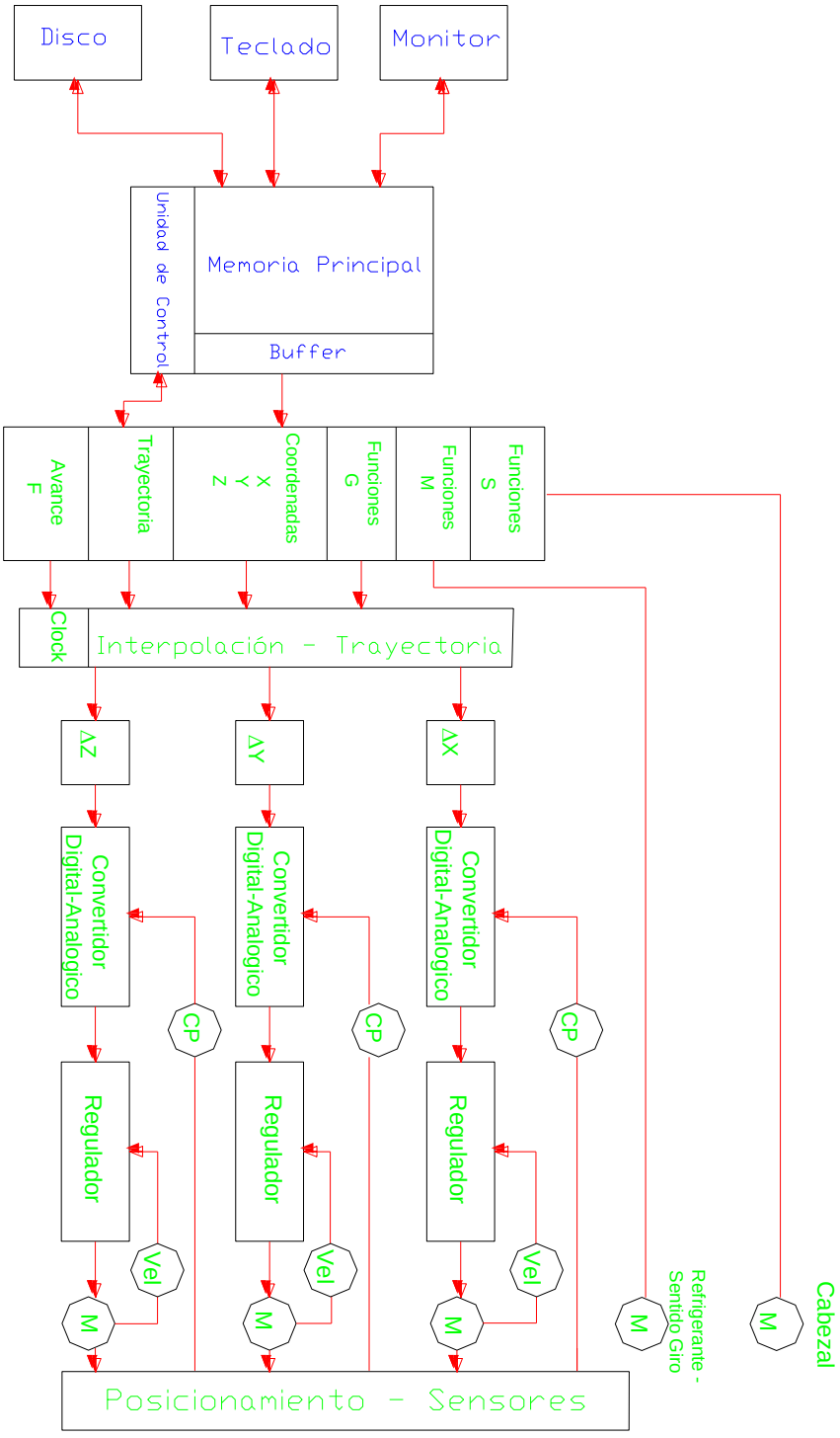


Fig. 1.1 Esquema General de un Control Numérico

1.1.1.1.- La unidad de Entrada y Salida de datos

La unidad de entrada y salida de datos admite datos para ser procesados y ejecutar las operaciones y genera información para el usuario que lo necesita para verificar la secuencia de ejecución del proceso. Al principio los datos eran transmitidos al control numérico mediante cintas perforadas que eran leídas por el equipo mediante lectores ópticos. Hoy en día los métodos gráficos han avanzado una enormidad y se cuenta con la gran capacidad gráfica de los computadores y la interacción hombre máquina puede llegar a ser en tiempo real mediante el uso de monitores.

1.1.1.2.- La Memoria Principal

En la memoria interna del equipo se almacena el código del programa , los datos de la máquina, datos de compensación, ceros, aceleración. Las instrucciones del programa se guardan allí como bloques de datos que se van leyendo secuencialmente y la unidad de interpretación se encarga de definir que tipo de operación debe ejecutar y en qué forma.

1.1.1.3.- La Unidad de Control

1.1.1.3.1.- Cálculo de la trayectoria.

El control lee de la memoria, un bloque de programa que define un ciclo de trabajo. El bloque es interpretado de tal forma que el control define:

- Nueva posición a alcanzar.
- Velocidad a la que debe realizarse el trayecto.
- Forma en que debe realizarse.

- Compensación de la herramienta.

El cálculo de la nueva posición se realiza de la siguiente manera.

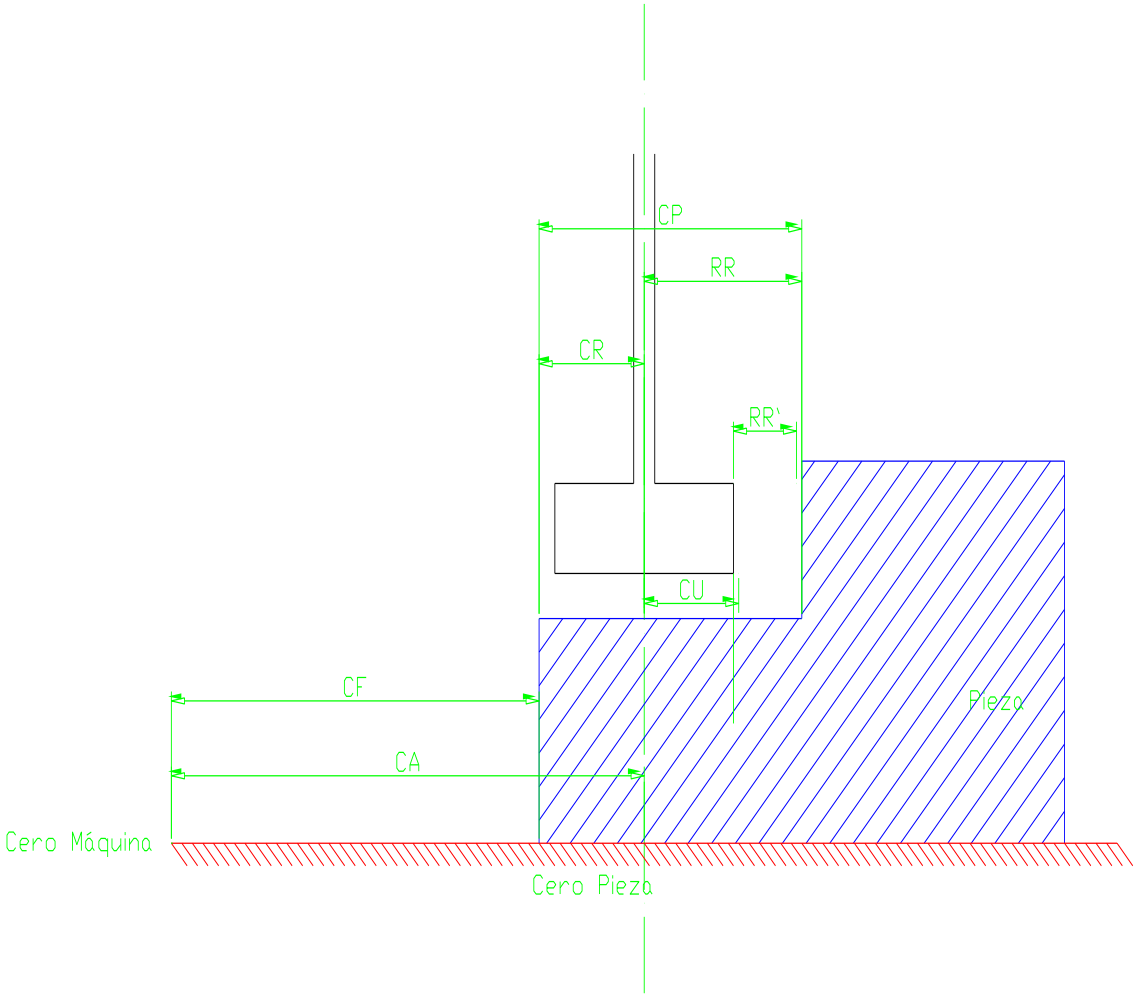


Fig. 1.2 Cálculo de la Trayectoria

CA es la cota absoluta o posición de la herramienta respecto al cero absoluto o cero de la máquina. Este valor se encuentra almacenada en una dirección de memoria que es constantemente actualizada según el movimiento de la herramienta.

CF es el cero flotante y es la distancia entre el cero de la máquina y el extremo de la pieza que se toma como origen para medir todos los desplazamientos programados en el programa.

CR es la cota relativa al cero flotante.

CP es la cota predeterminada o requerida para alcanzar después de completar el movimiento.

RR es la distancia a recorrer para alcanzar la cota programada y RR' es el camino tomando en cuenta la compensación de la herramienta.

$$CR = CA - CF$$

$$RR = CR - CP$$

$$RR' = RR \pm CU$$

1.1.1.3.2.- Compensación.

La compensación se refiere a la consideración de la distancia entre el centro de la herramienta y el radio de la herramienta. Esto con el fin de establecer la relación entre estas dos variables y poder escribir programas en

los que se toma en cuenta el contorno de la pieza y no la trayectoria de la herramienta .

1.1.1.3.3.- Interpolación

Las máquinas-herramienta tienen dos formas fundamentales de ejecutar el movimiento de un punto a otro y es básicamente la característica que diferencia uno y otro tipo. ***Un control numérico de posicionamiento punto a punto*** ejecuta el cambio de posición sin importar la trayectoria recorrida puesto que lo esencial es alcanzar la coordenada programada. Este tipo de movimiento se suele realizar a la máxima velocidad que alcanza el equipo y es de aplicación general en maquinas de taladrar. Cuando el movimiento punto a punto se realiza en direcciones paralelas a los ejes principales y ortogonales el sistema de posicionamiento es ***punto a punto y paraxial.*** El otro tipo de sistema de posicionamiento es aquel que controla la trayectoria en cada instante y para lo cual monitorea la razón de cambio de posición en cada eje además de la posición final. Con estos automatismos se pueden conseguir trayectorias lineales, arcos de circunferencia y otras curvas definidas analíticamente. Es lógico pensar que un equipo de contorneo puede realizar el trabajo de una máquina de posicionamiento punto a punto y utilizado en tareas de torneado, fresado y por ende en taladrado.

1.1.1.3.3.1.- Interpolación Analógica.

En función de la trayectoria definida mediante el programa se obtienen salidas de movimiento para cada eje. Estas son cambios de posición que en cada intervalo de tiempo Δt guardan una relación definida

por la curva que se está generando. El interpolador analógico produce estás respuestas en forma continua e instantánea.

1.1.1.3.3.2.- Interpolación Digital.

Los sistemas de control numérico cubren su alta precisión basados en la interpolación digital puesto que su contraparte analógica tiene limitaciones de respuesta en el tiempo. En este caso la salida de control consiste de trenes de impulsos para cada eje controlado que guardan una relación funcional en cada intervalo discreto de tiempo Δt. La precisión esta limitada por la longitud de la palabra digital que registra los valores de posicionamiento.

1.1.1.3.3.3.- Analizador Diferencial Digital.

Cuando la curva se puede representar en su forma paramétrica:

$$X = f(\Phi)$$
$$Y = f(\Phi)$$
$$Z = f(\Phi)$$

Estas relaciones son usadas por el interpolador Analizador Diferencial Digital y es el que en principio se usa en forma más extendida en los automatismos de control numérico puesto que establece una relación entre los cambios en las coordenadas en intervalos de tiempo requeridos. Basados en intervalos de tiempo discretos muy pequeños Δt en cada momento se puede establecer el cambio de posición Δx, Δy, Δz.

1.1.1.4.- Servomecanismos.

La salida del sistema de control es un tren de pulsos que acciona a los elementos motores encargados de producir el movimiento. Este conjunto de control de movimiento y elemento actuador es el servomecanismo y es el elemento final de la cadena de control y encargado de producir el cambio físico de posición en el tiempo.

1.1.1.4.1.- Servomecanismos de Lazo Abierto.

En este caso las órdenes de accionamiento al motor se origina en la unidad de cálculo y que establece la distancia a recorrer y la velocidad de posicionamiento. El servomecanismo no está recibiendo durante la ejecución del desplazamiento, información alguna que afecte el régimen de movimiento ni en cuanto a posición ni velocidad, no existe retroalimentación.

1.1.1.4.2.- Servomecanismos de Lazo Cerrado.

El servomecanismo de lazo cerrado constantemente recibe información tanto de la unidad de cálculo como de sensores que miden la posición real y la velocidad en tiempo de ejecución. Esta información es procesada y en todo momento afecta al régimen de ejecución del posicionamiento. El tren de pulsos es transformado en todo momento en un voltaje continuo, señal analógica que sirve para alimentar al sistema de regulación de velocidad.

1.1.1.4.3.- Posicionamiento

En los sistemas de lazo cerrado el posicionamiento es controlado básicamente de dos maneras. La primera de ellas es la desaceleración por etapas y es usada en sistemas de bajo costo en los que se disponen de grandes juegos mecánicos, con el fin de ganar tiempo para activar el bloqueo del desplazamiento. La segunda técnica es basada en la desaceleración continua y controlada usando sistemas reguladores de velocidad reversibles que dan mayores prestaciones de precisión.

1.1.1.4.4.- Sistemas de Accionamiento.

- Motores Paso a Paso.
- Sistemas Hidráulicos Paso a Paso.
- Motores de corriente continua.
- Motores de corriente alterna con cambio de velocidad mecánico.
- Motores de corriente alterna con cambio de velocidad por variación de frecuencia.
- Sistemas hidráulicos.

1.1.1.4.5.- Esquematización de un servomecanismo de control de posición.

Un servomecanismo de control de posición produce un movimiento relativo entre dos elementos según lo indica la unidad de cálculo y basado en las señales que genera el interpolador. En un sistema de lazo cerrado el tren de pulsos es transformado en una señal analógica de voltaje que es función de la frecuencia del tren de pulsos.

Este voltaje continuo ingresa a un comparador junto con una señal de retroalimentación proveniente de los sensores de posición y velocidad para establecer la magnitud del error. Este error es la diferencia entre la posición actual y la posición deseada. Así mismo cualquier desviación de la velocidad deseada es detectada y corregida en todo momento.

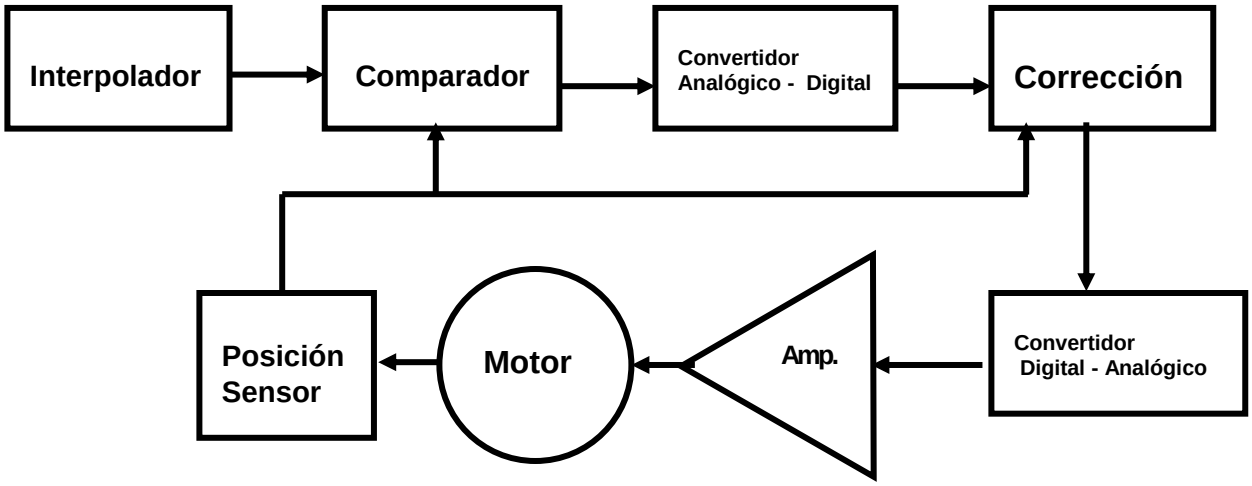


Fig. 1.3 Esquema de un Servomecanismo

1.1.2.- Programación de un control Numérico

Un operador humano debe crear un programa o secuencia de operaciones y cargarlo en la memoria del control numérico. Este programa debe especificar los parámetros necesarios y la secuencia de operaciones a realizar, en un idioma que la máquina pueda reconocer e interpretar. Mediante el código, el operador establece con exactitud la velocidad de avance, velocidad de rotación del husillo, cambios de herramienta, desplazamientos. En la actualidad este código se genera con programas de computadoras que generan código en forma automática en función de información de la geometría de la pieza y de las características de los materiales a mecanizar. Estos programas son lo mas avanzado hasta el momento en el área de automatización y han generado una nueva especialidad en la ingeniería como es el CAM, Computer Aided Manufacturing. Utilizando estos instrumentos se puede generar rápidamente una rutina de mecanizado y cargarla para su ejecución en el equipo de control numérico.

1.1.2.1.- Programación Manual.

Cuando no se cuenta con el apoyo de estas herramientas computacionales, el usuario de una máquina de control numérico debe crear un código en base a su conocimiento de las operaciones de mecanizado y del material y herramientas de corte involucrado. Finalmente el código obtenido debe ser de igual manera introducido en la memoria del equipo para su ejecución. Tanto la rutina generada manualmente como la generada en forma automática debe cumplir con algunas reglas que se han ido estandarizando en busca de un código mas global para estos equipos. Ya

en las normas ISO 1056 se regulariza el uso de un código que tiende a ser el estándar y se conoce mas generalmente como código G.

Un programa de control numérico se estructura en tres elementos

- Principio de programa constituido por comentarios que no representan operaciones a ejecutar sino que resultan de uso en la documentación de la rutina. Esta parte precede siempre a un símbolo de inicio de instrucciones ejecutables y que en algunos sistemas se usa el %.

- Programa de mecanizado conformado por las líneas de programa o bloques que suministran los parámetros de corte y su secuencia. Un comando o instrucción esta formado por una letra que representa una *Dirección de la Memoria* del control numérico y debe estar acompañada de un valor numérico. La memoria del control numérico está dividida en bloques que afectan por separado una función específica del equipo.

- El fin de programa que indica la culminación de la secuencia de instrucciones en la memoria del equipo, que también se usa entre otros el símbolo de %.

1.1.2.2.- Direcciones de memoria

1.1.2.2.1.- Dirección N.

Esta dirección de memoria corresponde al numero de secuencia del bloque de instrucción y el valor que le sigue es un número que se incrementa según la secuencia de ejecución de la rutina.

1.1.2.2.2.- Dirección X, Y, Z.

Aquí se establecen las cotas de movimiento según los ejes principales del equipo, a saber de los ejes X, Y, Z.

1.1.2.2.3.- Dirección G, funciones preparatorias.

Sirve para la especificación del tipo de mecanizado en cuanto a tipo de interpolación, corrección de la herramienta, trayectoria y algunas mas dependiendo del fabricante de la máquina herramienta. Esta dirección se usa en los equipos de mas reciente data y es parte de esa tendencia a estandarizar el código de programación que se viene usando en los equipos de control numérico.

1.1.2.2.4.- Dirección M, funciones auxiliares.

Dirección que contiene las indicaciones de operaciones espaciales como sentido de rotación del husillo, paradas programadas, aplicación de refrigerante.

1.1.2.2.5.- Dirección F.

La velocidad de avance de la herramienta es afectada por esta dirección y según el fabricante es interpretada en función del valor numérico o de la cantidad de dígitos que sigan a la letra F en el código

1.1.2.2.6.- Dirección S.

El husillo rotará a la velocidad especificada por esta dirección y será interpretada de diferentes maneras según el fabricante. En algunos casos el valor numérico representa las revoluciones por minuto directamente o por el contrario especifica un valor dentro de una gama fija de velocidades.

1.1.2.2.7.- Dirección I, J, K.

Cuando se realiza la interpolación circular, estas direcciones se usan para establecer el centro de la circunferencia de la trayectoria cada una con respecto a un sistema de coordenadas cuyos ejes son paralelos a los ejes principales X, Y, Z.

1.1.2.2.8.- Dirección T

La secuencia de cambios de las herramientas en la torreta es especificada según la dirección T. En el equipo que nos ocupa, esta dirección viene acompañada de la orden M06 que autoriza el cambio de herramienta o rotación de la torreta.

1.1.2.3.- Compensación de la herramienta.

En la norma ISO 2539 se mencionan los caracteres que se usan para indicar mediante el programa los valores de compensación de la herramienta. El sistema Ernault Somua objeto de este trabajo, no utiliza este tipo de dirección sino que junto con la cifra que indica el número de útil, se agrega un dígito que especifica el numero de la dirección de memoria en la que se encuentra la corrección que le corresponde. Es así como un bloque tal como T021 se refiere a la herramienta en la posición dos (02) y se le aplica la corrección #1. Esta compensación está fijada en el tablero de control del equipo.

1.1.3.- Descripción del equipo.

1.1.3.1.- Características Técnicas.

En la página siguiente, figuara 1.4 tomada del manual de Operación del equipo de control numérico Ernault Somua se pueden apreciar las dimensiones principales del equipo:

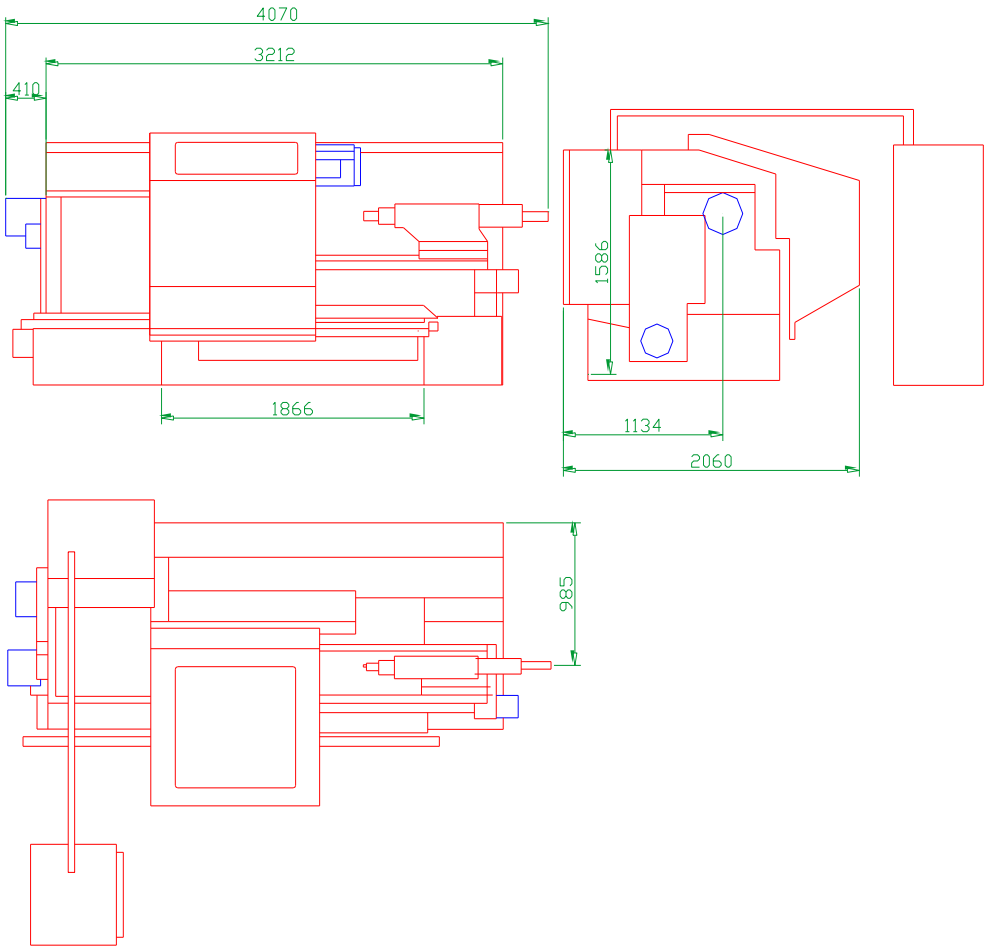


Fig. 1.4 Vistas principales

1.1.3.1.1.- Componentes básicos

- Cabezal con dos rangos de 12 velocidades según selector manual.
- Motor de 2 velocidades, 750 r.p.m, 11 KW y 1500 r.p.m, 15 KW respectivamente.

- Bancada con rieles de deslizamiento
- Tortea con plato de 8 estaciones.
- Tanque para el sistema hidráulico de accionamiento.
- Gabinete para el sistema de suministro de energía eléctrica principal.
- Gabinete para el sistema de control y entrada de datos.
- Bomba eléctrica para el sistema de lubricación.

1.1.3.1.1.1.- El Cabezal

Es accionado por los motores de 750 y 1500 r.p.m. que según se combinen junto con el uso del sistema de clutches E1 a E5 proporcionan el rango de velocidades del cual se dispone para los diversos trabajos de mecanizado requeridos. 5 electro válvulas son los elementos finales de control en la selección de estos clutches y el frenado. Las dirección S con su respectivo valor es la que especifica la selección de una de las velocidades disponibles.

1.1.3.1.1.2.- El Husillo

Provee el movimiento transversal y es guiado por medio de los rieles asimétricos de sección prismática que dan estabilidad estructural al sistema en movimiento. El contacto entre los rieles de deslizamiento y la tortea se mantiene por rodamientos que se acoplan a las caras del prisma en todo momento. Los movimientos son sensados en todo momento por un rodamiento esférico en cual a su vez está conectado al sistema de control del torno.

1.1.3.1.1.3.- Sensores de final de carrera

El final de carrera en las dos direcciones está asegurado por dos microswitches. Un tercer microswitch, indica el cero de la máquina para los desplazamientos transversales. El accionamiento de estos microswitches desconecta cualquier acción del torno para evitar sobrepasar el límite de movimiento posible en cualquiera de las dos direcciones.

1.1.3.1.1.4.- La torreta

tiene dos opciones de montaje que permiten un arreglo horizontal o en su defecto un arreglo vertical que es el que corresponde al arreglo de nuestro equipo. El cabezal de la torreta es accionado por un motor hidráulico y la precisión en la selección es obtenido por un sistema de seguidor y acoplamiento curvo, para soltar la posición seleccionada y tomar una nueva se utiliza un pistón hidráulico. Un conjunto de 12 contactos magnéticos dan información de la posición relativa de las herramientas para que éstas puedan ser utilizadas en cualquier orden y un posicionamiento correcto de la herramienta es indicado por un microswitches que habilita la señal Start Cycle. Las posiciones se pueden seleccionar con los siguientes valores numéricos.

01 02 04 06 07 08 10 12

1.1.3.2.- La Programación

La unidad de control es un sistema 550 Lathe Control de General Electric y es una unidad para el control de mecanizado por contorneo. mediante este sistema, el operador puede suministrar paso a paso y manualmente, un conjunto de instrucciones para un mecanizado o puede ingresar un programa mediante cinta perforada o usando una interfaz de puerto paralelo con conector DP25.

Esta unidad usa el formato de información basado en letras de dirección y números que completan una palabra o comando. *Un comando* es una unidad básica de información que tiene todos los datos para una acción mínima. M06 es un comando que habilita la selección de una herramienta programada mediante el comando T02. La letra es una dirección de la memoria del sistema de control y en función de su tipo, el valor numérico que acompaña a ésta tiene un significado específico.

El formato en el que se presentan los datos del programa está basado en la norma EIA Standard RS – 274 y es como se indica a continuación:

- **Unidades inglesas:**

  N3 X+14 Z+14 I+14 K+14 F03 S+2 T3 M2

- **Unidades métricas**

  N3 X+32 Z+32 I+32 K+32 F12 S+2 T3 M2

A los fines de la programación basada en lectura por cinta perforada la secuencia en la estructura del programa es como a continuación se indica:

1. Encabezamiento 10 pulgadas si no se está usando carrete.
2. Carácter de parada.
3. Algunos caracteres de avance de cinta
4. Carácter de fin de bloque (EOB End of Block).
5. El cuerpo del programa conformado por la secuencia de caracteres en el formato indicado anteriormente.
6. Caracter de parada si es requerido.
7. Caracter de fin de bloque.
8. Algunas pulgadas de cinta sin perforar
9. 10 pulgadas de cinta libre.

El mencionar esta estructura es importante por cuanto esta era la forma original de introducir programas en el sistema de control numérico y luego de la implementación de la comunicación vía puerto paralelo de un PC muchas de estas características en la transferencia de data aún se mantienen. El ingreso de código para la ejecución se realiza a través de la cinta perforada lo cual es posible directamente o través de una interfaz de comunicación via puerto paralelo de una PC. Esto no se realiza como un protocolo propiamente dicho sino que cada uno de los pines de la interfaz DB 25 de puerto paralelo se ha puesto en paralelo con cada uno de los bits que ingresan por cinta perforada. Mediante software se ingresa una secuencia de datos como la que se produciría al correr una cinta perforada. Existe un módulo del programa SMART – CAM que implementa esta comunicación vía puerto paralelo y también es aprovechada en el presente trabajo. Cuando la cinta corre a través del sistema lector óptico, se detiene un intervalo de

tiempo suficiente para excitar los transductores y presentar la palabra digital correspondiente a cada carácter. Cada símbolo está representado por una palabra de 8 bits y cada bit corresponde a cada una de las pistas de la cinta perforada. Una perforación es atravesada por el haz de luz y representa un 1 mientras que la no perforación es un cero binario. Por ejemplo, el caracter 3 es leído en cinta perforada como:

00010011 en Código EIA

Su equivalente decimal es **19**

El puerto paralelo paralelo de 25 pines utiliza los pines 2 al 9 para poner la data de salida en paquetes o palabras de 8 bits. Al conectar estos puntos en paralelo con el bus de datos del lector óptico se presentan los elementos de información provenientes de la PC como si se originaran de la cinta perforada. Cada carácter en el puerto paralelo se pone en su representación binaria correspondiente al código ASCII y debido a esto se requiere su transformación a Código EIA para hacerlo compatible con la unidad de control del sistema Ernault Somua.

Estas consideraciones son importantes y son utilizadas mas adelante en la confección del módulo que transfiere el código editado en el programa de simulación y lo pone en la memoria del control numérico mediante el puerto paralelo del PC. A continuación, la figura 1.5 es una tabla que representa cad uno de loscaracteres del código EIA que acepta el equipo de control numérico en el que se desarrola este trabajo.

	Pista								
Carácter	**8**	**7**	**6**	**5**	**4**	**arrast**	**3**	**2**	**1**
PARADA					●	•		●	●
A		●	●			•			●
B		●	●			•		●	
C		●	●	●		•		●	●
D		●	●			•	●		
E		●	●	●		•	●		●
F		●	●	●		•	●	●	
G		●	●			•	●	●	●
H		●	●		●	•			
I		●	●	●	●	•			●
J		●		●		•			●
K		●		●		•		●	
L		●				•		●	●
M		●				•	●		
N		●				•	●		●
O		●				•	●	●	
P		●		●		•	●	●	●
Q		●		●	●	•			
R		●			●	•			●
S			●	●		•		●	
T			●			•		●	●
U			●			•		●	●
V			●			•	●		●
W			●			•	●	●	
X			●	●		•	●	●	●
Y			●	●	●	•			
Z			●		●	•			●
+		●	●	●		•			
-		●				•			
Delete		●	●	●	●	•	●	●	●
Tab			●	●	●	•	●	●	
Space				●		•			
						•			
CR	●					•			
/			●	●		•			●
.		●	●		●	•		●	●
0			●			•			
1						•			●
2						•		●	
3				●		•		●	●
4						•	●		
5				●		•	●		●
6				●		•	●	●	
7						•	●	●	●
8				●		•			
9			●	●		•			

Fig. 1.5 Código EIA

El carácter de fin de bloque EOB es un carácter de retorno de carro CR (Carriage Return) que en EIA Standard es 244 y en ASCII es 13.

El caracter de tabulación puede ser usado para mejorar la legibilidad del programa y sin embargo este no representa información alguna ejecutable para el torno a control numérico.

Cada comando esta conformado por un número específico de dígitos pero cuando no se requieran dígitos a la izquierda los ceros en estos espacios pueden ser obviados.

Si se requiere un movimiento en el sentido negativo de cualquiera de las dos direcciones, el valor numérico debe estar precedido por un signo menos. En el caso contrario, el signo positivo puede omitirse y el control numérico lo asume así.

La información en las direcciones S, T yM es retenida hasta que una nueva lectura correspondiente a un nuevo bloque de ejecución es leido.

1.1.3.2.1.- Comandos

1.1.3.2.1.1.- Número de Secuencia N

El número de secuencia se programa con la dirección N y un número de 3 dígitos y este valor es mantenido en un registro de desplazamiento hasta

que un nuevo número de secuencia es programado. Esta data es presentada en la pantalla del gabinete de control cuando el selector esta en la posición N

1.1.3.2.1.2.- Desplazamientos

X±3.2 Z±3.2 Programan la magnitud del desplazamiento en los ejes X y Z respectivamente y un número de 5 dígitos cuantifica la magnitud de ese desplazamiento en centésimas de milímetro. El mayor valor de cambio de posición que se puede programar es de 999.99 milímetros con una resolución de 0.01 milímetros.

1.1.3.2.1.3.- Arcos de circunferencia

I±3.2 K±3.2 Programan una interpolación circular entre el punto actual y la coordenada final programada en la direcciones X y Z tomando como centro un punto que se encuentra según las coordenadas I y K relativas al punto de inicio y donde la dirección I es paralela a la dirección X y la dirección K es paralela a la dirección Z. En la figura 1.6 se esquematiza la relación entre los diferente ejes coordenados.

La información combinada de distancia, signo y desfase del centro de circunferencia indica el sentido de giro tomando en consideración que el equipo no maneja arcos de mas de 90 grados. La ausencia de valores numéricos para I y K implica un movimiento lineal.

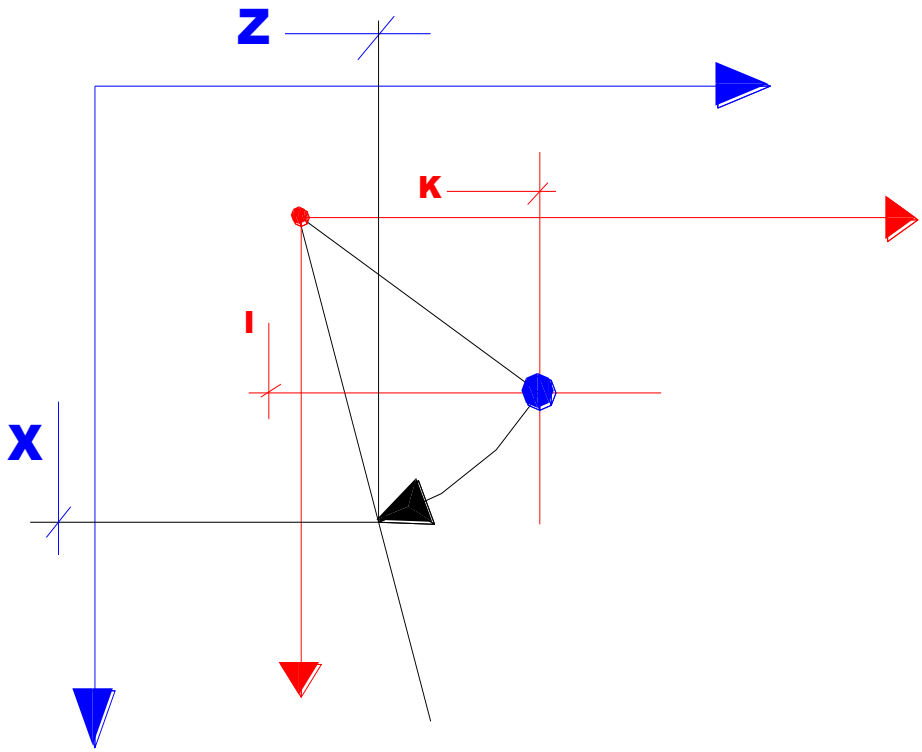


Fig. 1.6 Interpolación Circular

1.1.3.2.1.4.- Avance

F0.01 a F1.99 El avance en milímetros por revolución se programa con la letra F seguida de un valor numérico de tres dígitos que va desde F0.01 a F1.99 milímetros por revolución. El valor programado es retenido hasta que removido al producirse una nueva lectura en esa dirección de memoria. Este

parámetro junto con la dirección S definen por completo la velocidad de corte a la cual se realiza la operación de mecanizado. La rata de movimiento en cada eje es ajustada en todo momento en cada eje individualmente para obtener el vector de velocidad programado según esta dirección. Por debajo de velocidades de 30 r.p.m. se pueden observar variaciones indeseables en este parámetro. Es posible que aunque el valor máximo programable sea de 1.99 este no pueda ser alcanzado realmente.

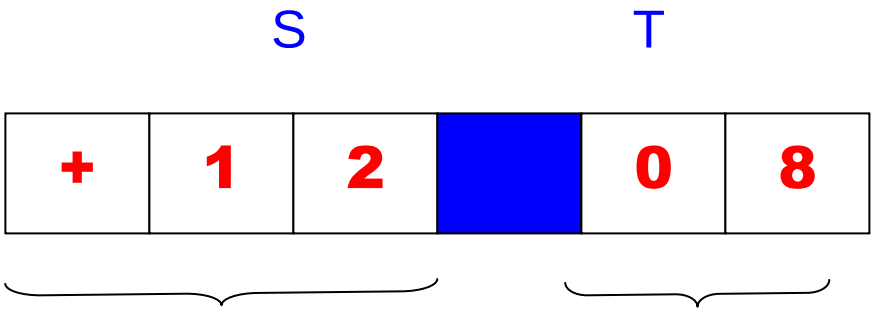
1.1.3.2.1.5.- Velocidad del Cabezal

S+2 La velocidad del cabezal se programa con la dirección S y es retenida hasta una nueva lectura y sirve para escoger un valor específico de velocidad dentro de los valores discretos disponibles en cada uno de los dos rangos disponibles en el equipo de control numérico.

1.1.3.2.1.6.- Selección de la herramienta en la torreta

T3 Un número de tres dígitos indica la posición activa de la torreta con una herramienta seleccionada. Los dos primeros dígitos de izquierda a derecha indican la posición en si y el último dígito indica la corrección que se aplica a esta herramienta y la cual está ajustada mediante un selector para cada corrección del 1 al 9 en la consola de control LM 550.

En la pantalla del gabinete de control se presentan la información relativa a la torreta y velocidad del cabezal en el siguiente formato.



En la tabla que sigue se presentan los valores correspondientes a las dos gamas disponibles de velocidades del cabezal según la dirección S correspondiente.

Rango	S01	S02	S03	S04	S05	S06	S07	S08	S09	S10	S11	S12
1	1600	1000	800	630	500	400	315	250	200	160	125	80
2	630	400	315	250	200	160	125	100	80	63	50	32

1.1.3.2.1.7.- Comandos Misceláneos

M2 Mediante estos comandos se programan acciones diversas a realizar por el equipo como son las paradas y encendido de la bomba de refrigerante, en el cuadro siguiente se presentan con su respectiva función.

Código	**Nombre**	**Función**
M00	Parada de Programa	*Detiene la ejecución del programa en curso. *Enciende la luz indicadora Cycle Start.
M02	Fin de Programa	*Detiene la ejecución del programa en curso. *Enciende la Luz de Fin de Programa (End of Program) *Borra todos los datos almacenados. *Cancela la verificación de paridad.
M03	Sentido Horario	Inicia la rotación del cabezal en el sentido horario a la velocidad que indique la dirección S.
M04	Sentido Antihorario	Inicia la rotación del cabezal en el sentido antihorario a la velocidad que indique la dirección S.
M05	Refrigerante	Activa la circulación de refrigerante
M06	Selección Herramienta	Activa el posicionamiento de la torreta en la herramienta especificada por la dirección T

1.2.- El Entorno Windows

Windows es un sistema operativo basado en una interfaz gráfica de usuario **(GUI**, ***Graphical User Interface*)**. Una GUI utiliza objetos gráficos en la pantalla de video para comunicar información e interactuar con el usuario del sistema. El mismo sistema de video que tradicionalmente funcionó como dispositivo de salida, presenta iconos y elementos visuales, llamados controles, que transforman el monitor en un medio de entrada de datos. El flujo unidireccional de información desde el teclado al programa y luego al video fue sustituido por la interacción bidireccional del usuario con los objetos del video.

El programa toma para sí un área rectangular de la pantalla donde se produce toda la actividad que se genera y se identifica con una barra de título. La mayor parte de las posibilidades del software están dispuestas en un elemento grafico conocido como menú que esta compuesto por una secuencia de botones que pueden ser accionados por el usuario. Algunas opciones de ese menú pueden requerir una entrada adicional de datos y son requeridos a través de la aparición de otro elemento visual conocido como el cuadro de dialogo. En este cuadro, el usuario ingresa mediante teclado o cualquier otro medio, la información requerida para continuar el trabajo.

Los sucesos que se desencadenan al seleccionar un control u otro elemento gráfico son conocidos como *eventos*. Los eventos son el resultado de una secuencia de llamadas a funciones específicas encargadas de realizar una tarea en particular, como puede ser el pintar la ventana con su contenido, mover texto, disminuir la ventana del programa, etc. Esto es posible debido a que el sistema operativo Windows puede mandar mensajes

al programa solicitando una actividad a realizar. La programación tradicional no permitía este intercambio, toda vez que existía un solo sentido en el tránsito de información del programa en ejecución hacia el sistema operativo. Esto es, cuando se requería visualizar algo en un programa BASIC por ejemplo, la instrucción PRINT pasaba los datos al sistema operativo DOS y este se encargaba de gestionar todo lo necesario para hacer posible el cumplimiento de esta solicitud. El programa no tenía manera de saber si la operación se había ejecutado y peor aún, ni siquiera si se había realizado en forma correcta.

1.2.1.- La Programación Orientada a objetos

Cuando se desarrolla software orientado al sistema operativo Windows, el programador se está comprometiendo con la ingeniería de programación basada en el concepto de objetos. ***Un objeto*** es el resultado de la abstracción en torno a los atributos de un ente real y ***La programación Orientada a Objetos*** procura encapsular en elementos de código estos atributos y comportamientos y simularlos para tratar de mostrar estas características. La programación orientada a objetos genera paquetes que contienen elementos simples de datos ***(atributos)*** y funciones ***(comportamientos)*** que se relacionan estrechamente y conforman un solo elemento que se conoce como Objeto. El objeto creado de esta forma, es capaz de interactuar con otros objetos aprovechando sus posibilidades sin necesidad de saber como están implementadas sus propiedades internamente.

Un objeto estructuralmente se construye con ***Datos Miembro y Funciones miembro*** donde los primeros son datos elementales como variables enteras, variables alfanuméricas y arreglos de datos y las funciones miembro son bloques de código que pueden tomar datos o no y realizar tareas específicas que modelan las respuestas y propiedades de ese objeto en su interacción con otros. Un objeto modelado de esta forma constituye una ***Clase*** y viene a ser un elemento unitario de dato conocido como ***tipos de datos creados por el usuario.*** A partir de clases ya creadas se pueden modelar objetos que utilizan las propiedades existentes de esas clases e incrementar su potencia agregando nuevas propiedades a las clases derivadas. ***Un Objeto se deriva de una clase*** cuando toma para si propiedades de una clase existente.

Toda la actividad de un programa para el entorno Windows se desarrolla en un área rectangular llamada ventana. Al parecer, este objeto grafico sabe como comportarse ante la solicitud de cualquier tarea. Solamente hay que indicar qué tarea se requiere y el resultado se obtiene en forma gráfica en la pantalla o una salida por impresora, por ejemplo***. En Windows el objeto principal es la ventana***, ella cuenta con funciones que permiten ese comportamiento al cual ya estamos tan acostumbrados. Son funciones específicas las que se encargan de pintar la ventana en la pantalla la primera vez, repintar una vez que se restituye una ventana minimizada, procesar el teclado y gestionar los eventos desencadenados por el simple accionar del botón del mouse. Cada uno de estas situaciones generan mensajes específicos a la ventana y esta sabe como tratar cada uno.

1.2.2.- Los Mensajes y eventos en Windows

Un programa Windows está formado por un núcleo central de código y un procedimiento de ventana. El núcleo central, *WinMain( ),* se encarga de iniciar y mantener la ejecución de la aplicación, crear el objeto ventana con todos sus atributos desde el inicio del programa y adicionalmente cuenta con un *bucle de mensajes.* El bucle de mensajes es una estructura de repetición que se encarga de procesar una a una, las solicitudes que el sistema operativo hace al programa como respuesta a cada evento que genera el usuario de la aplicación. Cada evento hace que el sistema operativo Windows produzca una llamada a una función específica del objeto ventana, cada llamada es un **Mensaje** y este se acomoda en secuencia, en un elemento contenedor que gestiona el sistema operativo a la espera de ser atendido por el bucle de mensajes. Según la cantidad de eventos que se producen estos se apilan en ***la cola de mensajes*** que Windows crea para cada programa en ejecución. Cuando el bucle de mensajes atiende uno de los mensajes lo pasa al ***procedimiento de ventana de la aplicación*** que es una función especial del programa que realiza las tareas inherentes a cada mensaje según su tipo.

Cuando el usuario del programa Windows produce algún cambio como consecuencia por ejemplo de minimizar la ventana, se produce un mensaje que Windows manda al programa y este es alojado en la cola de mensajes. *El evento* de hacer clic con el mouse en el botón de minimizar a generado esta llamada a una función auxiliar de la función WinMain( ); que se encarga de englobar todo el código necesario para procesar cada mensaje según su

tipo. *El Mensaje* es un objeto Windows que contiene información específica acerca de tipo y procedencia del mismo.

Cada mensaje que es obtenido de la cola de mensajes del programa es procesado según su naturaleza por *el procedimiento de ventana.* Este bloque de código reúne en si mismo, subprogramas que reciben como dato el mensaje apropiado para cada cual y realizan una secuencia específica de tareas como respuesta a ese mensaje en particular. Como es el caso de minimizar la ventana, se genera un mensaje tipificado por el sistema operativo Windows. *WM_SIZE* es un mensaje que se aloja en el bucle de mensajes cuando se cambia el tamaño de la ventana hasta que es procesado por el procedimiento de ventana. Este mensaje le indica al programa que el tamaño de la ventana a cambiado y debe repintarse con las nuevas dimensiones de la ventana en virtud de ese evento. Dentro del procedimiento de ventana existe código que atiende la solicitud de este mensaje y como resultado se genera un mensaje adicional, *WM_PAINT*.

Cuando se obtiene un mensaje WM_PAINT el procedimiento de ventana responde dibujando la ventana con sus nuevas dimensiones en la pantalla y al terminar esta tarea, el control se devuelve al bucle de mensajes del programa para seguir procesando los siguientes mensajes. Este ciclo se repite hasta que el usuario cierra el programa y se genera un mensaje *WM_QUIT* que obliga a que se gestione todo lo necesario para borrar la ventana y recobrar la memoria ocupada por el programa.

Por lo tanto, la mayor parte del trabajo de codificar un programa orientado al sistema operativo Windows se dedica a la gestión de los mensajes que el usuario pueda generar. Cada mensaje es pasado al

procedimiento de ventana como parámetro y según su tipo existirá código que lo procesa y devuelva el control a Windows.

1.2.3.- La gestión gráfica.

Toda la funcionalidad y aceptación de Windows se basa en que cualquier suceso o tarea ejecutado por un programa termina con una retroalimentación visual para el usuario rica en elementos gráficos. El resultado de cualquier mensaje atendido por el programa se traduce en un cambio en la ventana del programa que está presente en la pantalla de video. La ventana del programa esta delimitada por un elemento rectangular, el ***Marco Principal*** y se identifica con ***La barra de título****.* Dentro de este cuadro se encuentra alineado en el lado superior la *barra de menú principal.* El área restante es el ***Rectángulo Cliente*** y es donde se produce toda la acción gráfica de un programa Windows. Para llevar a cabo estas tareas el programa utiliza las funciones GDI *(Graphics Device Interface)* de Windows, estas son un conjunto de funciones que gestionan la aparición de todos los elementos gráficos que construyen lo que vemos en pantalla. Cuando el procedimiento de ventana procesa eventos que afectan lo que se ve en el rectángulo cliente, un mensaje WM_PAINT se produce y esto *Invalida el área de cliente*. Invalidar el área de cliente significa que lo que esta en pantalla ya no es válido y debe ser actualizado o redibujado. Entonces es cuando el programa, a través de la interfaz gráfica de dispositivos, opera cambios en la ventana usando funciones para el manejo de caracteres gráficos, dibujo de líneas, mapas de bits, etc.

La visualización en el área de cliente debe ser independiente del dispositivo gráfico en el que se presenta el programa. El programa debe poder aprovechar la potencia de un sistema de alta resolución con un gran

monitor y de igual manera aparecer también en un área de cliente bastante reducida. Windows mantiene información de las posibilidades de cada dispositivo gráfico mediante El Contexto de Dispositivo. ***El Contexto de Dispositivo*** es un objeto que maneja Windows y que almacena toda la información necesaria del dispositivo gráfico (hardware) por el que se pretenda manejar la salida visual y el cual está asociado con un periférico en particular como lo es una impresora, plotter o pantalla de video. Cualquier representación gráfica se hace en función de los parámetros que este contexto de dispositivo resguarda como datos. Algunos de los valores del contexto de dispositivo son el color del texto, el color del fondo del texto, el tipo de fuente, tamaños, etc. Cuando un programa requiere dibujar algo primero obtiene el contexto de dispositivo y luego aplica las funciones gráficas con esos datos.

2.- La Arquitectura General del Programa

En su totalidad, el programa para la simulación de procesos de corte desarrollado en este trabajo de grado está basado en el *armazón de aplicación* que brinda la MFC. La *Microsoft Foundation Class* es un conjunto de integrado de componentes de software orientado a objetos que aporta todo lo que se necesita en la construcción de una aplicación genérica orientada a Windows. Basados en el entorno de programación Visual C++ 6.0 de la suite Visual Studio de Microsoft, la MFC cuenta con los objetos comunes de una aplicación de esta naturaleza como lo son la clase de ventana, el documento, etc., que de otro modo requeriría de trabajo adicional el programarlos por cuenta propia. Disponiendo de estos elementos contenidos en clases, solo es necesario implementarlos ajustándolos a la necesidad particular que se quiere cubrir. Esto es la mayor parte del trabajo de programación que se a realizado en este simulador.

Toda aplicación Windows tiene que realizar trabajos comunes de gestión de memoria, inicio de la aplicación, crear los objetos ventanas. Todo esto es código redundante que ya está contenido en estos objetos prefabricados de software. Al usar estos productos reutilizables se ahorra trabajo y tiempo de codificación y se concentra el esfuerzo en el desarrollo de lo que realmente es nuevo. Lo nuevo en realidad es todo lo relativo a la simulación, esto es, la pieza, las herramientas de corte, el movimiento relativo entre ellas, el cálculo de la interferencia, la visualización de estos eventos.

Basados en la MFC de Visual Studio 6.0 se ha desarrollado la aplicación para la simulación de procesos de corte en torno a control

numérico utilizando ***La Arquitectura Documento - Vista*** que fundamenta su funcionamiento en dos objetos principales que son el ***Objeto Vista*** y el ***Objeto Documento.*** Un Objeto de la Clase Documento, contiene todos los datos necesarios para que el Objeto Vista presente ante el usuario, mediante el dispositivo grafico de visualización, toda la información que genera la ejecución del programa.

2.1.- El Simulador Control Numérico: La Arquitectura Documento – Vista

El Simulador Control Numérico es un software orientado al sistema operativo Windows que ha sido construido basado en la estructura que genera Visual C++ para aplicaciones que usan La Microsoft Foundation Class y cuyos elementos constructivos principales son piezas de software reutilizables llamadas clases de las cuales se derivan cada uno de los objetos que interactúan y dan funcionalidad al programa logrado. La aplicación, El marco principal, El Documento y la Vista son los cuatro objetos que constituyen el esqueleto del programa y sobre los cuales los demás objetos que modelan al control numérico se apoyan para realizar la simulación. De estos cuatro objetos, el objeto documento y el objeto vista concentran la mayor parte de código adicional y el cual es precisamente el que se encarga de realizar la simulación y presentar estos eventos en la pantalla del sistema de video en forma gráfica.

En el documento del simulador, que representa la memoria principal del control numérico en la simulación, se encapsula el código que procesa la

rutina de mecanizado que el usuario del programa introduce para simular y ejecutar desde la captura del código ASCII proveniente del teclado hasta la transformación en una *Tabla Simbólica* necesaria para dar significado desde el punto de vista de la máquina al texto del código NC. A partir de allí, el código NC está disponible para el objeto vista y ésta tomará la información necesaria del documento para presentar visualmente el movimiento relativo de los elementos en pantalla. El documento también guarda para la vista, la información relativa a las herramientas de corte y la pieza a mecanizar, como lo es la geometría y dimensiones principales. En general, la simulación se ejecuta con un intercambio de información entre el objeto vista y el objeto documento y esto es precisamente la piedra angular de la filosofía de un programa Documento – Vista. En los títulos siguientes se desarrolla una descripción un poco mas profunda del programa con el fin de presentar los otros objetos que esta aplicación contiene y que modelan los elementos móviles de la aplicación.

2.2.- El Armazón de Aplicación

El armazón de aplicación de la MFC genera una aplicación que básicamente está compuesta por cuatro objetos que interactúan entre sí para producir toda la funcionalidad requerida. Estos cuatro objetos, a saber de, La Aplicación, El Marco Principal, El Documento y la Vista realizan por separado cada uno una tarea en particular y cada uno afecta al otro según la implementación de los detalles.

Toda aplicación Windows requiere para su correcto funcionamiento una serie de tareas de inicialización que son comunes a todos los programas

Windows. La creación de las clases de ventanas, la inicialización de los parámetros de clase de ventana, la visualización por primera vez de la ventana de aplicación, la implementación del bucle de mensajes, la gestión de los mensajes. Todos estos trabajos son ejecutados por el objeto Aplicación derivado de la clase ***CWinApp*** de la MFC. Este objeto contiene a la función WinMain( ) que encapsula la inicialización, corrida y terminación de la aplicación al cierre. Inicializado el objeto aplicación este se conecta con otro objeto que representa el marco principal de la aplicación. El marco principal sustenta su funcionalidad según la clase de la MFC llamada ***CMainFrame***, encargada de la gestión del marco de ventana, barra de menús, barra de estatus, manejo de las diferentes ventanas de visualización de la aplicación, entre otras. Todos los procesos que atañen a la modificación de datos, ejecución de tareas y cálculos, guardado de datos se encapsulan en el objeto documento. El objeto documento se deriva de la clase ***CDocument*** también de la MFC y en el se llevan a cabo todos los procesos que modifican los datos que afectan lo que se ve en la pantalla. Esto último, el efecto gráfico y su gestión se concentra en el área de cliente y ésta es manejada por la clase de vista. El simulador desarrollado usa dos objetos vista, uno para mostrar un espacio para la edición del código NC y el otro para presentar la simulación de los movimientos de la herramienta de corte y sus efectos sobre la pieza. La MFC ofrece diferentes clases de vista cuya escogencia dependerá del tipo de visualización que se requiere. Pero básicamente la clase de vista contiene todas las funciones gráficas que nos permiten dibujar en pantalla y en cualquier dispositivo de visualización lo que el usuario final de la aplicación requiere ver. También contiene funciones de administración y gestión de recursos de hardware y memoria para a hacer eficiente el hecho gráfico.

2.2.1.- CWinApp: La Clase de aplicación

Es la clase de la aplicación que se encarga de dar identidad e individualidad al programa en ejecución y, como se mencionó anteriormente, encapsula en un solo objeto la inicialización, funcionamiento y terminación de una aplicación para Windows. Un programa Windows se construye basado en la creación de un único objeto derivado de la clase ***CWinApp***. Este objeto es el primero que existe en memoria incluso antes de crear la ventana de la aplicación. Podemos tener tantas instanciasde la aplicación en ejecución como el dispositivo de visualización permita. Cada instancia en ejecución, representa para Windows un ***Hilo de Ejecución (Thread)***. La Clase CWinApp contiene las funciones miembros que se requieren para que el sistema operativo administre los hilos de ejecución. Estas funciones ya están contenidas en una clase especial de la MFC llamada ***CWinThread*** y el objeto aplicación las incorpora por lo que se dice que un objeto de la clase CWinApp se deriva de esta última. Las funciones encargadas del inicio, ejecución, salida y control de hilo de ejecución se encuentran encapsuladas en la clase ***CWinThread***.

No es necesario codificar la función WinMain( ) que ya se ha descrito anteriormente porque se encuentra contenida en las bibliotecas de clases de la MFC y es llamada cuando el programa se inicia. WinMain( ) realiza servicios como el registro de las clases de ventana y luego llama a las funciones miembro de la clase de aplicación que inician y corren la aplicación. Para iniciar la aplicación la función WinMain( ) llama a las funciones miembros de la clase de aplicación ***InitApplication( )*** y ***InitInstance( )***. Hecho esto, el bucle de mensajes empieza a operar cuando WinMain( ) llama a la función miembro de la clase de aplicación ***Run( )***.

Cuando el usuario termina la aplicación, al cerrar la ventana, WinMain llama a ***ExitInstance( )*** también del objeto aplicación y que se encarga de la gestión de memoria al ser liberada por el programa.

En la figura 2.1, se muestra la secuencia principal de llamadas a función que hacen posible la ejecución de un programa desarrollado en base a la MFC. Donde WinMain(), la función central de una aplicación que se encuentra contenida en el objeto aplicación llama a InitInstance( ) que inicializa la aplicación y prepara el bucle de mensajes. La función Run( ) mantiene el funcionamiento del bucle de mensajes para procesar los diferentes mensajes generados por los múltiples eventos que generados por el usuario. Todo programa windows ocupa recursos de memoria que deben ser administrados eficientemente y luego ser liberados al terminar el programa. Cuando se finaliza la corrida estos recursos son devueltos al sistema según rutinas que están contenidas en ExitInstance( ).

Secuencia de ejecución:

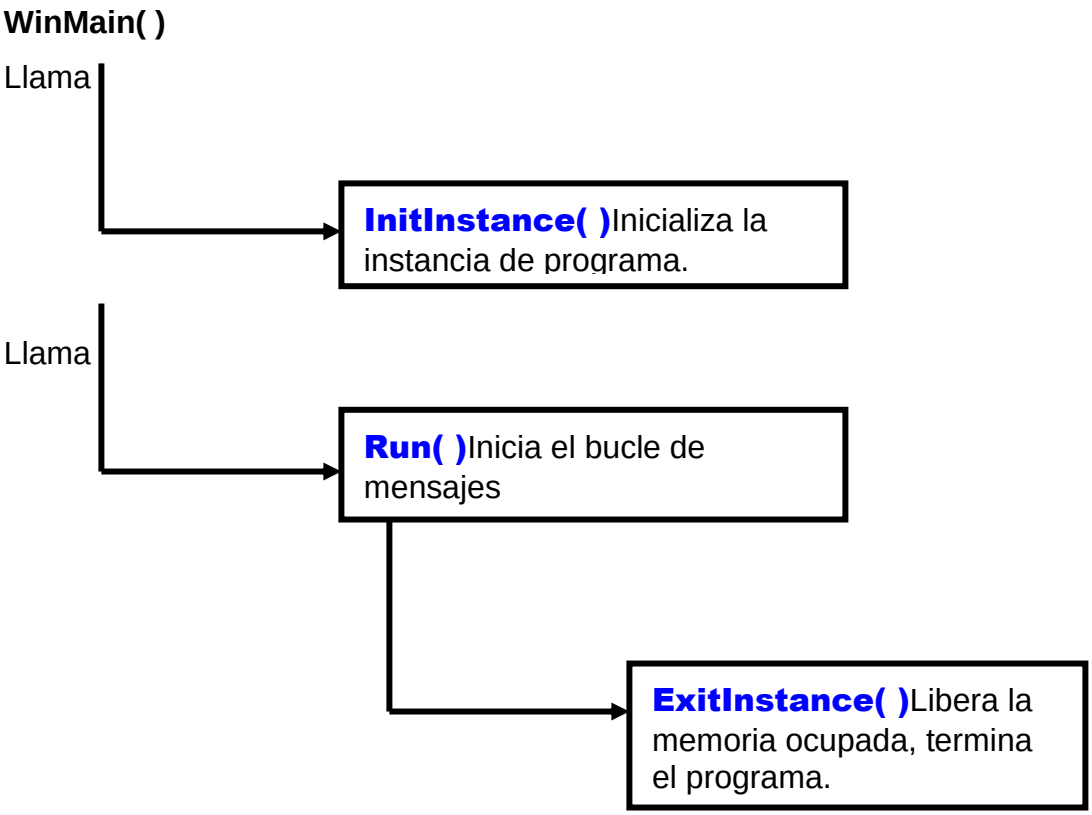


Fig 2.1 Secuencia de ejecución

2.2.2.- CFrameWnd: La clase de marco Principal

Cuando una aplicación Windows aparece en pantalla, esta se muestra contenida en un marco rectangular que la delimita con respecto a otras. Este rectángulo es el marco principal y contiene adicionalmente a la barra de título, la barra de status y las barras de menú. Este conjunto es modelado según la clase ***CFrameWnd*** de la MFC. Un objeto derivado de la clase *CFrameWnd* encierra la funcionalidad del marco principal de la aplicación. Cuando este objeto se construye al ser llamada su función miembro ***Create( )*** por el objeto aplicación, se crea la estructura real de la ventana y el armazón

de aplicación la enlaza y comienza a utilizar para utilizar las funciones miembro ***ShowWindow( )*** *y* ***UpdateWindow( )*** que presentan por primera vez la ventana de aplicación en la pantalla de video.

La clase CframeWnd, es el código empaquetado que provee todas las características de funcionalidad de la ventana de marco principal que es la que encuadra la ventana cliente. La ventana de marco principal es la ***Ventana Padre*** de la vista. Antes de que se dibuje cualquier cosa en nuestro programa, se crea un marco que encierra a nuestra futura visualización y que engloba adicionalmente a la barra de menú, la barra de status y la vista. Estas últimas son ***Ventanas Hijas*** del marco principal porque todo lo que pase en ellas está controlado por éste y en todo momento es notificado de cualquier evento. Mediante este mecanismo el armazón de aplicación se asegura un comportamiento armónico entre los diferentes objetos que conforman la aplicación, tal como lograr que las ventanas hijas se contraigan y expandan de manera acorde con los cambios que el usuario impone en el tamaño de la ventana de la aplicación.

El marco principal lleva cuenta de la ventana activa de la aplicación, todos los eventos que afectan a la aplicación se reflejan en la ventana hija que tiene el foco de la acción. El posicionamiento de las barras de control, las vistas, los cuadros de dialogo, la activación de los diferentes botones de la barra de menú y las solicitudes de ayuda sensible al contexto son gestionados por este objeto.

2.2.3.- CDocument: El Documento

El objeto Documento contiene los datos que el objeto vista visualiza. Específicamente para el simulador que ha desarrollado, el documento representa la memoria del programa porque en el se resguarda todo el código NC editado, la tabla simbólica que se genera al procesar el texto del código NC y los atributos dimensionales y geométricos de la pieza en bruto y las herramientas de corte. Una aplicación construida con los objetos de la MFC contiene un único objeto documento y puede tener múltiples objetos vista para manejar las diferentes maneras de presentar la respuesta gráfica. Se produce un intercambio de información entre la vista y el documento al momento de ejecutar una aplicación construida de esta forma.

El documento representa la unidad de datos que el usuario accede en el programa con los comandos de abrir, salvar y cerrar de la opción Archivo del menú principal de cualquier programa orientado a Windows. Este contiene los datos que utilizará el otro elemento complementario de una aplicación basada en la arquitectura Documento – Vista para mostrar información al usuario. El documento administra los datos que la vista usa para representar información gráfica mediante funciones de gestión gráfica.

Para ello, la clase CDocument de la que se derivan los objeto documento de cualquier aplicación de esta naturaleza esta provista de una interfaz coherente que le permite la gestión de los datos de la aplicación. El documento participa en la lectura y escritura de archivo desde y hacia los dispositivos de almacenamiento. Pone disponibles estos datos para la visualización en pantalla y la impresión.

Cuando los datos sufren modificaciones que deban ser mostradas al usuario del programa, la vista debe ser actualizada para reflejar estos cambios y para ello el documento cuenta con funciones miembro como

UpdateAllViews( ) que ejecutan los intercambios de información con las vista relacionadas al documento. ***Serialize( )*** es otra de las funciones del documento que mas importan por cuanto encierra todo el código que se requiere para conectar los datos de la aplicación con un archivo o elemento de respaldo de esa información en sistemas periféricos de resguardo de memoria como son los discos, cintas, etc.

El usuario de un programa Windows interactúa con los datos del documento por medio del objeto Vista asociado con el documento. La vista construye una imagen del documento en pantalla e interpreta las entradas del usuario como operaciones que afectan al documento y este debe contar con el código que ejecute esas tareas y luego la vista debe ser redibujada.

En el momento que se inicia la aplicación Simulador de torno NC se presenta una ventana de edición en la que el usuario ingresa el texto del código NC. Hasta este punto esa información es una secuencia de caracteres alfanuméricos sin significado alguno para el programa. Pero es antes de proceder a la simulación que una función miembro del documento procesa este texto y lo transforma en valores numéricos con significado. Elementos Sintacticos es esta función y de la cual se presenta una copia en el apédice de este trabajo. En su trabajo, esta función se conecta con la vista de edición y va tomando uno a uno cada carácter y según una estructura de selección va construyendo la sintaxis del código NC. Es decir, determina que dirección de memoria corresponde a una letra y en la tabla simbólica que va construyendo agrega su respectivo valor numérico. La tabla simbólica es construida en base a un objeto contenedor de tipo lista que apila en la misma secuencia de aparición cada registro correspondiente a una instrucción dada. Un registro de esta tabla simbólica contiene una columna para cada dirección

de memoria (N, X, Y, Z, G, F, S, I, J, K, M y T). En el apéndice se encuentran tanto la función ***ValorNumerico( )*** como la función miembro del documento ***InsertarFilaTablaSibolica( )***, la primera recupera el valor numérico para cada campo del registro y la última, una vez completado este proceso, inserta el registro en la tabla simbólica. Cuando la vista de simulación está lista para realizar su trabajo toma de esta tabla simbólica en el documento los valores para iniciar los movimientos de la herramienta.

El documento, a través de tres objetos del tipo cuadro de diálogo obtiene del usuario los datos necesarios para dimensionar las herramientas de corte y la pieza en bruto. Los atributos de la herramienta se concentran en un objeto que se denomina ***CHerramienta*** que una estructura de datos que contiene los datos miembro correspondientes a los parámetros principales que en el manual SANDVIK se utilizan para la selección de las herramientas.

- m_nAngulo: Cuando es un polígono, el ángulo interno principal
- m_nDimPrincipal: Cuando es un polígono, la longitud de los lados.
- m_nGeometria: Un valor entero para cada tipo de forma
- m_nPosicion: Posición en la torreta

Del objeto de diálogo correspondiente a la definición de la torreta los datos son pasados a un arreglo de 8 objetos CHerramienta de nombre Portaherramientas que representa a la torreta con las herramientas de corte en su posición. Si estos datos, el Código NC, las herramientas de corte y el

tocho se encuentran perfectamente definidos, se puede efectuar la simulación sin contratiempos.

2.2.4.- CView: La Vista

Todo lo que se ve en pantalla en un programa Windows es debido a las muchas funciones de gestión gráfica con las que cuenta la Clase *Cview( )* de la cual se derivan todos los objetos vista que están asociados al documento. La vista es la que se encarga de la visualización de los datos, que residen en forma abstracta en la memoria del computador y que se encuentran encapsulados en el documento.

Siempre que el usuario final del programa ingresa algún cambio mediante el teclado o por medio de la interfaz gráfica con el mouse, el documento recibe la notificación de la vista y se encarga de ejecutar las tareas requeridas. El resultado de esto es pasado a la vista y ésta se encarga de pintar la pantalla para reflejar estos cambios.

Con objetos gráficos sencillos como líneas, arcos de circunferencia patrones de fondo, polígonos simples e imágenes raster ajustadas al dispositivo gráfico del que se dispone, se llena el área de cliente y es lo que el usuario ve e interpreta como la información que busca. La responsable de dibujar al final de toda una secuencia de llamadas a funciones miembro de la vista como consecuencia de una modificación, es la función *OnDraw( ).* En esa función se concentra todo el código que llama a las funciones de la GUI (Graphical User Interface) de Windows que menciono al principio de esta

parte y que generan líneas, círculos, texto y procesan la interacción del usuario con el mouse, el teclado y cualquier otro dispositivo de entrada. Cuando el usuario introduce un cambio, se genera una llamada a una función de Windows que establece como no válida la información gráfica presente. La función *Invalidate( )*, genera un mensaje que va a la cola de mensajes y que termina en la llamada a OnDraw( ).

El Simulador de control numérico utiliza dos objetos vista para presentar al usuario los datos contenidos en el documento. El primero es el objeto destinado a la edición del texto del código NC que luego se conecta con el documento para que éste construya la tabla simbólica que servirá para ejecutar la simulación de los procesos de corte. El segundo objeto de vista es el que encapsula todas las funciones gráficas que representan el movimiento de las herramientas de corte y su efecto sobre la pieza mecanizada.

2.2.4.1.- El Editor.

El editor es un objeto derivado de la clase de base CView de la MFC y que incrementa su versatilidad incorporando para sí un objeto derivado de la clase ***CRichEditCtrl***. Un objeto de esta clase contiene toda la funcionalidad de un editor de texto como son las opciones de cortar, pegar, copiar entre otras. ***CSimuladorControlNumericoView*** es el objeto editor del SimuladorControlNumerico y es el que se conecta con el documento para que éste construya la tabla simbólica que sirve para la realización de las tareas de simulación que ejecuta el otro objeto de vista de la aplicación.

2.2.4.2.- La Simulación

La Simulación se lleva a cabo en el objeto ***CPasoRunPasoView*** derivado de la clase de vista CView. La vista de simulación toma del documento los atributos geométricos y dimensionales que se encuentran resguardados en el documento y construye los objetos gráficos que los representan y que interactúan como imágenes raster mediante operaciones a nivel de bits para dejar en pantalla el efecto de interferencia entre ellas y representar en el caso apropiado, el fenómeno del arranque de viruta.

Al iniciar la simulación, una función miembro de la vista construye los objetos raster correspondientes a las herramientas involucradas. En el Apéndice, sección 6.2.1, se encuentra una copia del código correspondiente a la función que construye los elementos móviles en base a objetos gráfico poligonales. ***GenerarHerramientas( )*** es la función miembro de la vista que toma del documentos los datos de las herramientas y crea un arreglo de imágenes raster construidas con objetos gráficos sencillos y que corresponden a 6 de las geometrías de plaquitas que el manual SANDVIK 1999 propone para la selección de herramientas de corte.

Una vez construidos estos elementos, se puede comenzar a iterar a lo largo de la tabla simbólica del documento para ir ejecutando una a una las instrucciones del código NC. Para cada instrucción se realiza una llamada a la función miembro de la vista ***OnSimulacionInstruccionSiguiente( ),*** en este bloque de código se diferencia entre una operación de movimiento o no movimiento. Las operaciones de no movimiento en general son aquellas que provienen de comandos misceláneos y a diferencia de estas, las de movimientos se detectan cuando los campos correspondientes a las

direcciones X y Z contienen un valor entero diferente de cero. Se puede ir al apéndice y revisar el código de esta función para ver que cuando esto sucede, se activa un objeto de Windows llamado ***Temporizador***. El temporizador es una función Windows que aprovecha la señal del clock del procesador para enviar un mensaje Windows del Tipo **WM_TIMER** a intervalos regulares de tiempo y que se toman como base para producir una actualización de la pantalla según la aparición de estos mensajes. En cada ciclo del temporizador (10 cada segundo), la función ***OnTimer( )*** de la vista, calcula la nueva posición de la herramienta de corte basada en los parámetros de velocidad, cambio neto de posición y tipo de interpolación para esa instrucción y pasa el control a la función ***OnDraw( )*** de la vista para presentar en intervalos discretos de tiempo, una nueva imagen que representa el estado actual de los elementos de la simulación. El temporizador se detiene cuando se alcanzan los cambios de posición prescritos en ambas coordenadas.

OnDraw( ) antes de realizar cualquier operación gráfica durante el movimiento de la herramienta, realiza una operación booleana entre el objeto raster herramienta activa y el objeto raster pieza, elementos que previamente han sido construidos basados en los datos del documento. Esta operación es una resta en la que queda como resultado otra región raster que corresponde al área de la pieza que no es común al área de la herramienta. Es de esta forma como se simula el proceso de arranque de viruta. La región correspondiente a la herramienta de corte se mantiene igual por cuanto se considera para esta simulación, que este elemento no pierde material.

La función miembro de la vista llamada ***OnTimer( ),*** es la encargada de atender el proceso de los mensajes WM_TIMER introducidos por

Windows debidos al temporizador. Para lograr un efecto de continuidad en el movimiento, se generan 10 actualizaciones de la pantalla por minuto por cuanto una cantidad mayor sería practicamente imperceptible para el usuario. En OnTimer se contiene toda la rutina que determina el cambio de posición de la herramienta transcurrido un intervalo de tiempo correspondiente a un milisegundo (1 ms) en tiempo real. Para efectos de documentación, se incluye esta rutina en el apéndice, sección 6.2.3.

3.- El Simulador de Torno NC

Cuando se inicia la ejecución del programa Simulador Control Numérico, se presenta la ventana de marco principal, La barra de menú, la barra de herramientas, la barra de status y la ventana de edición de código NC.

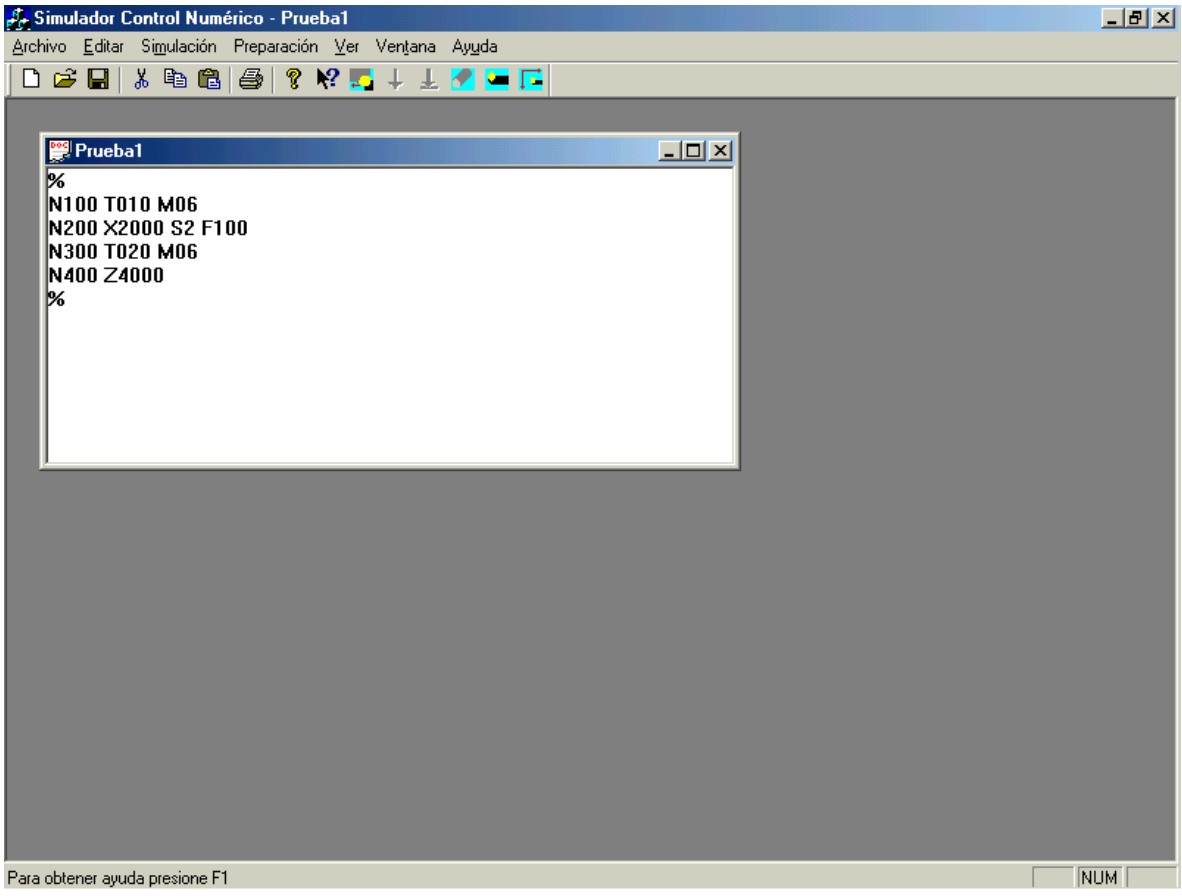


Fig 3.1 La ventana de la aplicación.

La barra de menú principal presenta para su selección las siguientes opciones:

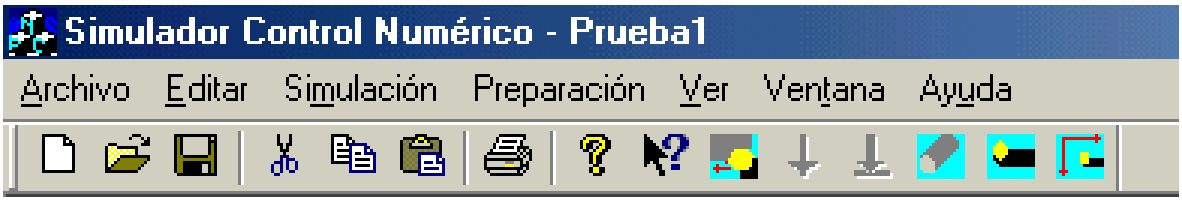


Fig 3.2 El menú principal y la barra de herramientas

- ***Archivo*** Operaciones de respaldo y recuperación de archivos desde o hacia los diferentes dispositivos de memoria que el equipo recono en ese momento.
- ***Editar*** Prestaciones para la modificación del contenido de los archivos de código NC activos.
- ***Simulación*** Opciones para la ejecución y visualización de la simulación del código NC activo.
- ***Preparación*** Presenta los diferentes cuadros de diálogo para indicar los diferentes atributos geométricos y dimensionales de los elementos involucrados en la simulación.
- ***Ver*** Brinda la posibilidad de modificar el nivel de amplificación de las operaciones en la pantalla.

- ***Ventana*** Sirve para la selección de los diferentes arreglos de ventana.
- ***Ayuda*** Permite el acceso a la ayuda en línea.

3.1.- Lectura y respaldo de archivos

La opción Archivo del menú principal permite la lectura y guardado de los archivos de código NC de nuestra aplicación, asi como las operaciones típicas de impresión. Su selección despliega submenús que son bastante conocidos por los usuarios de aplicaciones Windows y por lo que no se dedica una explicación exhaustiva.

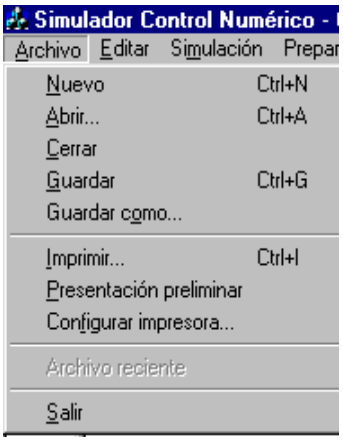


Fig 3.3.- El Menú Archivo

3.2.- Edición del código NC

Cuando se selecciona Nuevo, Abrir o al iniciar la aplicación se presenta la ventana de edición que es la configuración por defecto. Dentro de esta ventana es posible introducir mediante teclado o leer desde cualquier dispositivo de almacenamiento, el texto de código NC que representa la rutina de control numérico que se habrá de ejecutar en la aplicación. La opción Editar del menú principal, da herramientas para la modificación de los archivos como son copiar, cortar y Ejecutar en el torno. Esta última transfiere directamente al control numérico, el código para realizar la operación en el equipo sin visualizar simulación alguna.

Fig 3.4 El Menú Editar

El código que se encuentra en la ventana de edición es el que será ejecutado por el simulador en la pantalla y si así es requerido, ejecutado en el equipo de control numérico. Esto es posible mediante la siguiente opción del menú principal, ***Simulación.***

3.3.- Preparación.

Antes de realizar cualquier operación de mecanizado es necesario suministrar al programa los datos de geometría y dimensiones de las herramientas de corte y de la pieza en bruto. La opción Preparación del menú principal, ofrece los submenú necesarios para la entrada de estos datos mediante los cuadros de diálogo correspondientes.

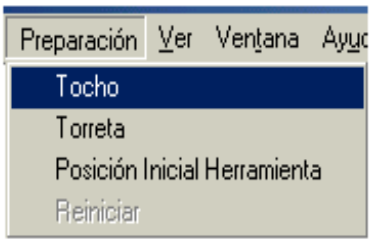


Fig 3.5.- Menú Preparación.

El Submenú tocho activa el cuadro de diálogo que permite el dimensionamiento de la pieza en bruto con sus magnitudes originales. En este cuadro de diálogo como el que se presenta en la figura, se indica el diámetro, longitud y se define si está previamente perforado.

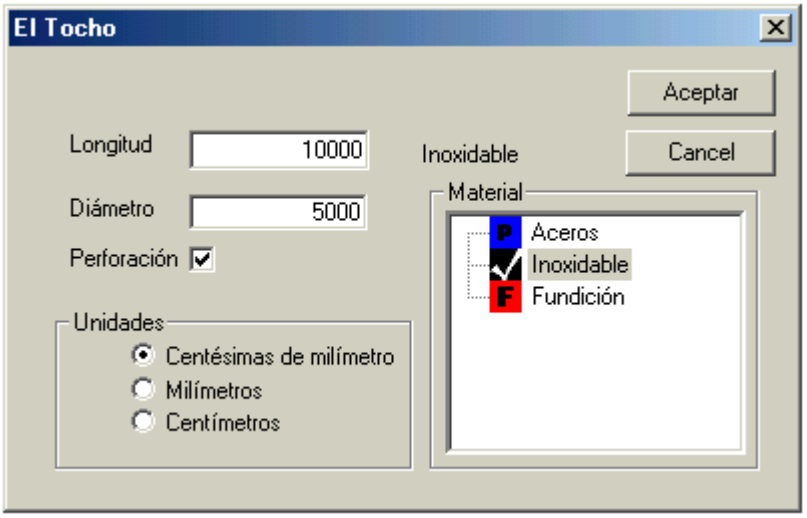


Fig 3.6 Dimensiones de la pieza en bruto.

Cuando el control Perforación está seleccionado y se aceptan los valores que se ingresan en el cuadro de diálogo de la figura 3.6, aparece inmediatamente el cuadro de diálogo perforación y cuya apariencia debe ser similar a la figura 3.7. Aquí se introducen los atributos dimensionales de la perforación que previamente tiene el tocho para trabajos de mecanizado interno.

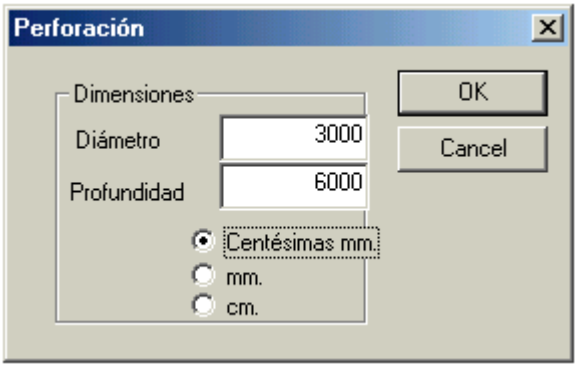


Fig. 3.7 El diálogo Perforación.

El paso siguiente es establecer las propiedades de las herramientas de corte y esto es posible mediante la selección de la opción ***Torreta*** del menú Preparación a lo cual se presenta el cuadro de diálogo de la figura 3.8. En él se introducen las dimensiones según las propiedades del manual para selección de herramientas Corokey de SANVIK del año 1999. Existen 8 posiciones y 6 geometrías disponibles para realizar la simulación. La herramienta queda definida por su posición en la torreta, el tipo de geometría, la longitud principal en milímetro y el ángulo con respecto a la superficie de transición.

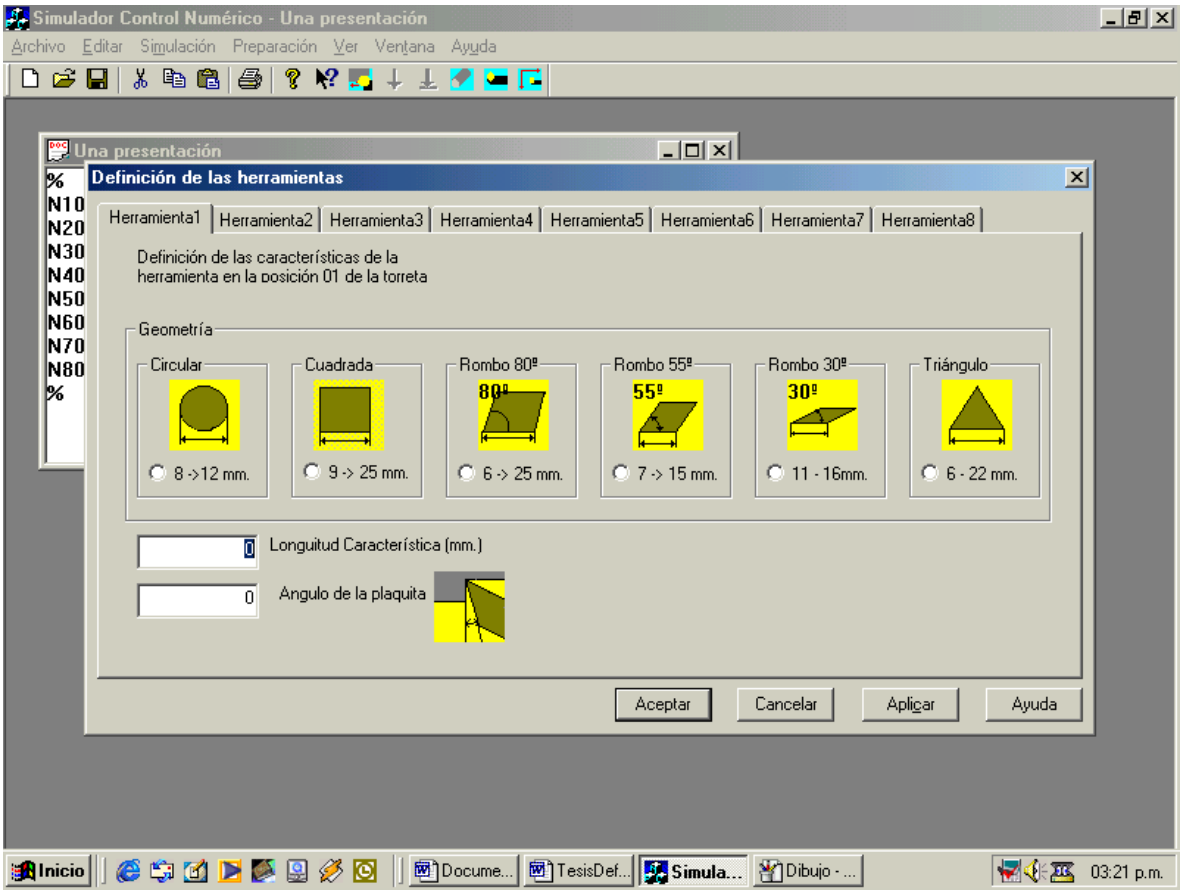


Fig 3.8 Definición de las herramientas.

La hoja de diálogo ***Definición de las herramientas*** presenta 8 fichas correspondientes a 8 puestos de herramientas de corte y en cada una el usuario selecciona una de las 6 geometrías disponibles de plaquita, cada una tomada de las propuestas en el manual SANVIK mencionado anteriormente. Las geometrías que se presentan para selección son:

- Circular: con el diámetro como dimensión principal
- Cuadrada: la dimensión principal es la magnitud de los lados.

- Rombo: con ángulo interior de 80 grados y se define al igual que el anterior y los siguientes polígono por la magnitud de su lado.
- Rombo: con ángulo interior de 55 y 30 grados.
- Triangular: es un triángulo equilátero.

Adicionalmente se agrega una información relativa al ángulo de posicionamiento con respecto a la superficie de transición y las magnitudes de los lados son suministradas en milímetros.

Finalmente se puede especificar la posición inicial de la herramienta seleccionando ***Posición Inicial Herramienta*** del menú Preparación. En este espacio se especifican las coordenadas iniciales del cero de la herramienta. Este cuadro de diálogo presenta la apariencia de la figura 3.9.

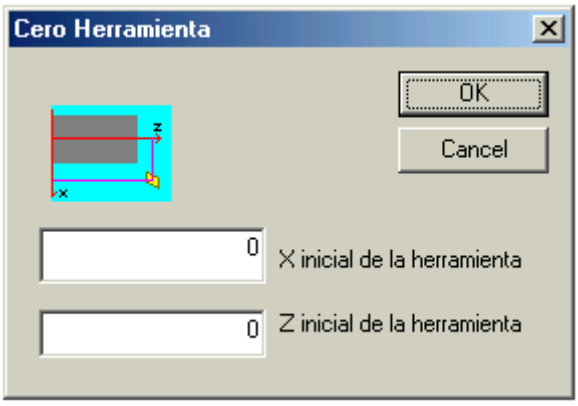


Fig 3.9 El Diálogo Cero Herramienta

Con los pasos previos se han definido por completo las herramientas de corte y el tocho y ya se puede pasar a la simulación del código NC. En la barra de menú se encuentra la opción ***Simulación.*** La selección de cualquiera de las opciones de este elemento del menú principal, pone a disposición del usuario la ventana de visualización de la simulación del código NC activo.

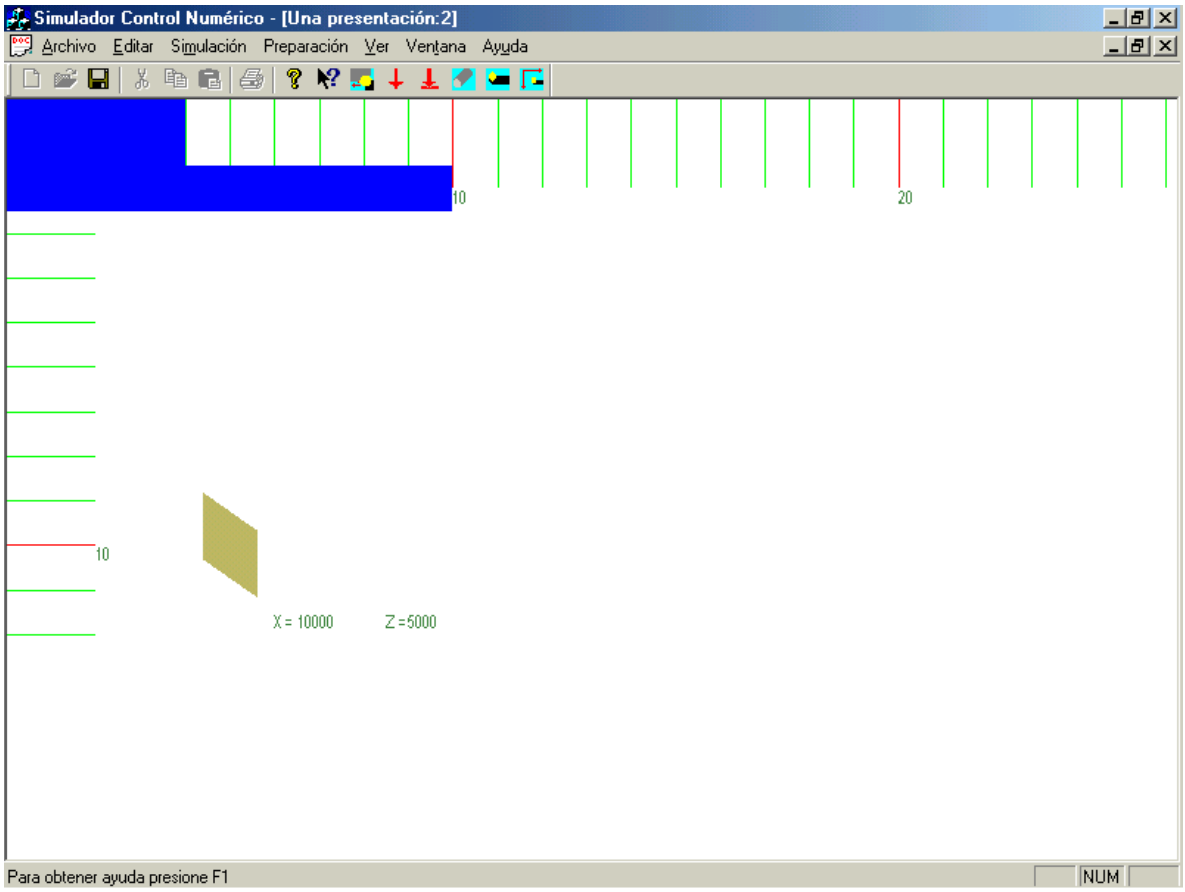
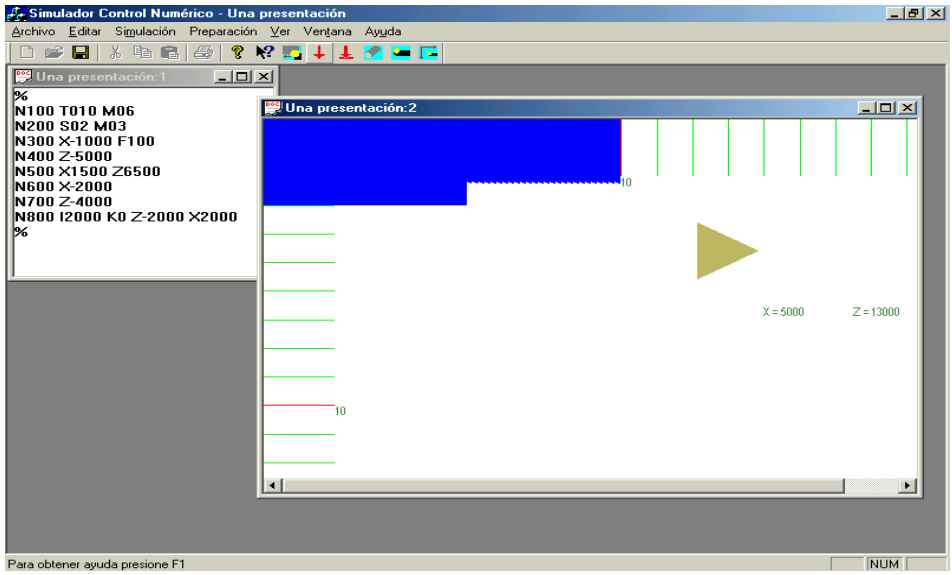


Fig 3.10 La ventana de simulación

3.4.- La Simulación

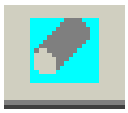
Esta opción del programa es la que permite la visualización de la simulación del código NC activo y las diferentes alternativas del submenú indican en que forma se lleva a cabo. De este modo, es posible ejecutar el código instrucción por instrucción bajo el control del usuario quien indicará cuando se ejecuta la próxima. También existe la alternativa de ejecutar toda la rutina de mecanizado sin parar hasta el final y finalmente se puede realizar el mecanizado en forma real en el equipo de control numérico bajo el control de esta aplicación mientras se recibe una retroalimentación visual de la ejecución en el simulador.

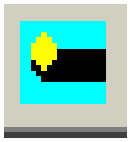
En la figura 3.10 se puede apreciar el aspecto de la ventana de simulación tras la ejecución de un cilindrado según las instrucciones del código NC de la ventana de edición.

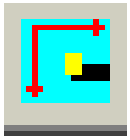


Se cuenta con botones de acceso directo para la definición de los elementos de la simulación que complementan al menú principal.

 Presenta la ventana de simulación.

 Definición del tocho.

 Definición de las herramientas de corte.

 Posición inicial de la herramienta de corte.

En términos generales, esta es una descripción de las prestaciones que este software de simulación de procesos de corte ofrece al usuario que principalmente esté interesado en un contacto con las técnicas de programación de estos equipos.

4.- Conclusiones

Lejos de pretender obtener una aplicación tan robusta como un sistema CAM, se ha buscado generar una herramienta que facilite el aprendizaje de los elementos mas importantes de la programación de un torno a control numérico. Toda vez el que el contacto por primera vez con un equipo de esta naturaleza es difícil y cualquier error puede significar un gasto importante en términos de reparaciones en el caso de choques indeseados.

El programa presenta algunas prestaciones para recrear algunas de las condiciones reales en la operación del torno a control numérico en lo que se refiere a geometría de la pieza y las herramientas disponibles. La estructura del programa induce a pensar que es posible incluir muchos mas elementos para hacer mas rico el paquete y el entorno de programación utilizado para el desarrollo de esta herramienta es generoso en las posibilidades de modificación y mejoras. La arquitectura misma de la aplicación, queda abierta a modificaciones que aumenten su potencia como herramienta de aprendizaje y trabajo.

La simulación paso a paso y completa están complementadas con una interfaz que comunica vía puerto paralelo el torno con el PC y así de este modo, es posible accionar la ejecución real de una rutina de mecanizado, mientras se representa en la pantalla este trabajo.

Es posible que el presente trabajo sea un punto de inicio en el desarrollo de este tipo de aplicaciones CAM que puedan llegar a ser tan robustas como las que se encuentran en el mercado y que faciliten el trabajo de diseño y mecanizado y hagan mas amigable la interacción del usuario con equipos de mecanizado

5.- Recomendaciones.

- El presente programa es principalmente una herramienta para el aprendizaje en el área de control numérico. Sería provechoso el diseño de actividades prácticas basadas en este simulador para complementar la información recibida en la cátedra de Tecnología Mecánica.

- Con este trabajo no se pretende cubrir toda la potencialidad de una aplicación CAM, pero si puede ser el primer paso hacia este objetivo.

- Este trabajo puede ser el punto de inicio para un proyecto de investigación que desarrolle herramientas cada vez mas potentes en el campo del mecanizado y diseño de procesos.

- La simulación es una herramienta que también se puede aplicar a muchas área de conocimiento dictadas en esta escuela.

- Existen muchas posibilidades si la información sobre materiales disponible en este paquete se amplía mucho mas.

- Se puede también incorporar información sobre herramientas de corte adicionales que ampliarán las posibilidades de simulación, como sería el caso del taladrado.

- Un próximo paso sería llevar la simulación al campo tridimensional toda vez que resulta mas rica en información al evaluar los procesos de mecanizado.

6.- Apéndice

6.1.- El Documento::CSimuladorControlNumericoDoc.

6.1.1.- ElementosSintacticos( ).

```
void CSimuladorControlNumericoDoc::ElementosSintacticos( )
{
m_nContLineas = 0;
TRACE("\n\nEntrando en elementos sintacticos\n\n");
m_posicionTablaSimbolica = m_tablaSimbolicaActiva.GetHeadPosition();
POSITION posicion;
int N = 0, X = 0, Y = 0, Z = 0, G = -1, T = 1, M = -1, I = 0, K = 0, F = 0, S = 0;
posicion = GetFirstViewPosition();
CSimuladorControlNumericoView* pFirstView = (CSimuladorControlNumericoView*)(GetNextView( posicion ));
pFirstView->m_richEditCtrl.GetWindowText(m_lineaDePrograma.m_strCodigo);
indice = 0;
indice = m_lineaDePrograma.m_strCodigo.Find('N', 0);//Busca el inicio de la parte ejecutable
TRACE("\n\nEl primer N se encuentra en la posición %d\n\n", indice);

N = ValorNumerico();
TRACE("\nEl Valor numérico de N es %d\n", N);
caracter = m_lineaDePrograma.m_strCodigo.GetAt(indice);
while(indice < m_lineaDePrograma.m_strCodigo.GetLength())
      {
       switch(caracter)
              {
              case 'N':
                     N = 0;
                      N = ValorNumerico();
                      break;
             case 'X':
                      X = 0;
                      X = ValorNumerico();
                       break;
              case 'Y':
                      Y = 0;
                      Y = ValorNumerico();
                      break;
              case 'Z':
                     Z = 0;
                     Z = ValorNumerico();
                     Break
              case 'G':
                      G = 0;
                     G = ValorNumerico();
                     break;
              case 'T':
                     T = 0;
                     T = ValorNumerico();
                     break;
              case 'M':
                     M = 0;
                     M = ValorNumerico();
                     break;
               case 'I':
```

```
                I = 0;
                I = ValorNumerico();
                break;
        case 'K':
                K = 0;
                K = ValorNumerico();
                break;
        case 'F':
                F = 0;
                F = ValorNumerico();
                break;
        case 'S':
                S = 0;
                S = ValorNumerico();
                break;
        case '\n':
                if (indice < m_lineaDePrograma.m_strCodigo.GetLength())
                {
                indice++;
                        if (indice < m_lineaDePrograma.m_strCodigo.GetLength())
                        {
                        caracter = m_lineaDePrograma.m_strCodigo.GetAt(indice);
                        }
                }
                break;
        case 13:
                if (indice < m_lineaDePrograma.m_strCodigo.GetLength())
                {
                InsertarFilaTablaSibolica(m_posicionTablaSimbolica, indice, N, X, Y, Z, G, T, M, I, K, F, S);
                G = -1;
                K = 0;
                I = 0;
                //J = 0;
                X = 0;
                Y = 0;
                Z = 0;
                M = -1;
                indice++;
                caracter = m_lineaDePrograma.m_strCodigo.GetAt(indice);
                }
                break;
        default:
                indice++;
                if (indice < m_lineaDePrograma.m_strCodigo.GetLength())
                        {
                        caracter = m_lineaDePrograma.m_strCodigo.GetAt(indice);
                        }
                break;
        }
}

        TRACE("\nSaliendo de elementos sintacticos\n");
}
```

6.1.2.- ValorNumerico( )

```
int CSimuladorControlNumericoDoc::ValorNumerico( )
{
int valorNumerico = 0;
indice++;
```

```
int pivote = indice;
BOOL negativo = FALSE;
caracter = m_lineaDePrograma.m_strCodigo.GetAt(indice);
if(caracter == 45)
        {
        negativo = TRUE;
        indice++;
        caracter = m_lineaDePrograma.m_strCodigo.GetAt(indice);
        }
while(caracter > 47 && caracter < 58 && indice < m_lineaDePrograma.m_strCodigo.GetLength())
        {
        valorNumerico = valorNumerico * 10 + (caracter - 48);
        if(indice < m_lineaDePrograma.m_strCodigo.GetLength())
                {
                caracter = m_lineaDePrograma.m_strCodigo.GetAt(indice);
                }
        }
        if (negativo)
        {
        return -1 * valorNumerico;
        }
        else
        {
        return valorNumerico;
        }
```

6.1.3.- InsertarFilaTablaSibolica( ).

```
void CSimuladorControlNumericoDoc::InsertarFilaTablaSibolica(POSITION posicion,
                                                             int indice,
                                                             int N,
                                                             int X,
                                                             int Y,
                                                             int Z,
                                                             int G,
                                                             int T,
                                                             int M,
                                                             int I,
                                                             int K,
                                                             int F, int S)
                                                            {
        CTablaSimbolica* pFilaTablaSimbolica = new CTablaSimbolica;
        pFilaTablaSimbolica->m_nFavance                 =       F;
        pFilaTablaSimbolica->m_nGPreparacion            =       G;
        pFilaTablaSimbolica->m_nICentro                 =       I;
        pFilaTablaSimbolica->m_nIndice                  =       indice;
        pFilaTablaSimbolica->m_nKcircunferencia         =       K;
        pFilaTablaSimbolica->m_nMmiscelaneos            =       M;
        pFilaTablaSimbolica->m_nNinstruccion            =       N;
        pFilaTablaSimbolica->m_nScabezal                =       S;
        pFilaTablaSimbolica->m_nTherramienta            =       T;
        pFilaTablaSimbolica->m_nXcoordenada             =       X;
        pFilaTablaSimbolica->m_nYcoordenada             =       Y;
        pFilaTablaSimbolica->m_nZcoordenada             =       Z;
        m_posicionTablaSimbolica                        =
        m_tablaSimbolicaActiva.InsertAfter(m_posicionTablaSimbolica,
                                           pFilaTablaSimbolica);
                                           m_nContLineas++;
                                   }
```

6.2.- La Vista:: CPasoRunPasoView

6.2.1.- GenerarHerramientas( ).

```
void CPasoRunPasoView::GenerarHerramientas()
{
for(int Tool = 1; Tool <= 8; Tool++)
{
Torreta.m_nPosicion        =        m_pDocumento ->Portaherramienta[Tool].m_nPosicion;
Torreta.m_nGeometria       =        m_pDocumento->Portaherramienta[Tool].m_nGeometria;
Torreta.m_nDimPrincipal    =        m_pDocumento->Portaherramienta[Tool].m_nDimPrincipal * 100;
Torreta.m_nAngulo          =        m_pDocumento->Portaherramienta[Tool].m_nAngulo;
CPoint Poligono80[4];
CPoint Rectangulo[4];
CPoint Poligono55[4];
CPoint Poligono35[4];
CPoint Triangulo[3];
switch(Torreta.m_nGeometria)
{
case 1:
Tools[Tool].CreateEllipticRgn(m_fFactorZoom * (m_pointProcesoZ - Torreta.m_nDimPrincipal / 2),
                                                    m_fFactorZoom * (-m_pointProcesoX + Torreta.m_nDimPrincipal /
2),                                                         m_fFactorZoom * (m_pointProcesoZ +
Torreta.m_nDimPrincipal / 2),
                                                    m_fFactorZoom * (-m_pointProcesoX - Torreta.m_nDimPrincipal /
2));
break;
case 3:
Rectangulo[0].x = m_fFactorZoom * (m_pointProcesoZ - 0.7071 * Torreta.m_nDimPrincipal * cos((45 +
Torreta.m_nAngulo) * 3.1415 / 180));

Rectangulo[0].y = m_fFactorZoom * (-m_pointProcesoX + 0.7071 * Torreta.m_nDimPrincipal * cos((45 +
Torreta.m_nAngulo) * 3.1415 / 180));

Rectangulo[1].x = m_fFactorZoom * (m_pointProcesoZ + 0.7071 * Torreta.m_nDimPrincipal * cos((45 +
Torreta.m_nAngulo) * 3.1415 / 180));

Rectangulo[1].y = m_fFactorZoom * (-m_pointProcesoX + 0.7071 * Torreta.m_nDimPrincipal * cos((45 +
Torreta.m_nAngulo) * 3.1415 / 180));

Rectangulo[2].x = m_fFactorZoom * (m_pointProcesoZ + 0.7071 * Torreta.m_nDimPrincipal * cos((45 +
Torreta.m_nAngulo) * 3.1415 / 180));

Rectangulo[2].y = m_fFactorZoom * (-m_pointProcesoX - 0.7071 * Torreta.m_nDimPrincipal * cos((45
Torreta.m_nAngulo) * 3.1415 / 180));

Rectangulo[3].x = m_fFactorZoom * (m_pointProcesoZ - 0.7071 * Torreta.m_nDimPrincipal * cos((45 +
Torreta.m_nAngulo) * 3.1415 / 180));

Rectangulo[3].y = m_fFactorZoom * (-m_pointProcesoX - 0.7071 * Torreta.m_nDimPrincipal * cos((45 +
Torreta.m_nAngulo) * 3.1415 / 180));

Tools[Tool].CreatePolygonRgn(Rectangulo, 4, ALTERNATE);
break;

case 4:
Poligono80[0].x = m_fFactorZoom * (m_pointProcesoZ - Torreta.m_nDimPrincipal * cos(40 * 3.1416 / 180) * sin((40
+ Torreta.m_nAngulo) * 3.1415 / 180));
```

```
Poligono80[0].y = m_fFactorZoom * (-m_pointProcesoX + Torreta.m_nDimPrincipal * cos(40 * 3.1416 / 180) *
cos((40 + Torreta.m_nAngulo) * 3.1415 / 180));
Poligono80[1].x = m_fFactorZoom * (m_pointProcesoZ + Torreta.m_nDimPrincipal * sin(40 * 3.1416 / 180) * cos((40
+ Torreta.m_nAngulo) * 3.1415 / 180));
Poligono80[1].y = m_fFactorZoom * (-m_pointProcesoX + Torreta.m_nDimPrincipal * sin(40 * 3.1416 / 180) * sin((40
+ Torreta.m_nAngulo) * 3.1415 / 180));
Poligono80[2].x = m_fFactorZoom * (m_pointProcesoZ + Torreta.m_nDimPrincipal * cos(40 * 3.1416 / 180) * sin((40
+ Torreta.m_nAngulo) * 3.1415 / 180));
Poligono80[2].y = m_fFactorZoom * (-m_pointProcesoX - Torreta.m_nDimPrincipal * cos(40 * 3.1416 / 180) *cos((40
+ Torreta.m_nAngulo) * 3.1415 / 180));Poligono80[3].x = m_fFactorZoom * (m_pointProcesoZ -
Torreta.m_nDimPrincipal * sin(40 * 3.1416 / 180) * cos((40 + Torreta.m_nAngulo) * 3.1415 / 180));
Poligono80[3].y = m_fFactorZoom * (-m_pointProcesoX - Torreta.m_nDimPrincipal * sin(40 * 3.1416 / 180) * sin((40
+ Torreta.m_nAngulo) * 3.1415 / 180));

Tools[Tool].CreatePolygonRgn(Poligono80, 4, ALTERNATE);
break;

case 5:

Poligono55[0].x = m_fFactorZoom * (m_pointProcesoZ - Torreta.m_nDimPrincipal * cos(27.5 * 3.1416 / 180) *
sin((27.5 + Torreta.m_nAngulo) * 3.1415 / 180));
Poligono55[0].y = m_fFactorZoom * (-m_pointProcesoX + Torreta.m_nDimPrincipal * cos(27.5 * 3.1416 / 180) *
cos((27.5 + Torreta.m_nAngulo) * 3.1415 / 180));
Poligono55[1].x = m_fFactorZoom * (m_pointProcesoZ + Torreta.m_nDimPrincipal * sin(27.5 * 3.1416 / 180) *
cos((27.5 + Torreta.m_nAngulo) * 3.1415 / 180));
Poligono55[1].y = m_fFactorZoom * (-m_pointProcesoX + Torreta.m_nDimPrincipal * sin(27.5 * 3.1416 / 180) *
sin((27.5 + Torreta.m_nAngulo) * 3.1415 / 180));
Poligono55[2].x = m_fFactorZoom * (m_pointProcesoZ + Torreta.m_nDimPrincipal * cos(27.5 * 3.1416 / 180) *
sin((27.5 + Torreta.m_nAngulo) * 3.1415 / 180));
Poligono55[2].y = m_fFactorZoom * (-m_pointProcesoX - Torreta.m_nDimPrincipal * cos(27.5 * 3.1416 / 180)
*cos((27.5 + Torreta.m_nAngulo) * 3.1415 / 180));
Poligono55[3].x = m_fFactorZoom * (m_pointProcesoZ - Torreta.m_nDimPrincipal * sin(27.5 * 3.1416 / 180) *
cos((27.5 + Torreta.m_nAngulo) * 3.1415 / 180));
Poligono55[3].y = m_fFactorZoom * (-m_pointProcesoX - Torreta.m_nDimPrincipal * sin(27.5 * 3.1416 / 180) *
sin((27.5 + Torreta.m_nAngulo) * 3.1415 / 180));

Tools[Tool].CreatePolygonRgn(Poligono55, 4, ALTERNATE);
break;

case 6:

Poligono35[0].x = m_fFactorZoom * (m_pointProcesoZ - Torreta.m_nDimPrincipal * cos(17.5 * 3.1416 / 180) *
sin((17.5 + Torreta.m_nAngulo) * 3.1415 / 180));
Poligono35[0].y = m_fFactorZoom * (-m_pointProcesoX + Torreta.m_nDimPrincipal * cos(17.5 * 3.1416 / 180) *
cos((17.5 + Torreta.m_nAngulo) * 3.1415 / 180));
Poligono35[1].x = m_fFactorZoom * (m_pointProcesoZ + Torreta.m_nDimPrincipal * sin(17.5 * 3.1416 / 180) *
cos((17.5 + Torreta.m_nAngulo) * 3.1415 / 180));
Poligono35[1].y = m_fFactorZoom * (-m_pointProcesoX + Torreta.m_nDimPrincipal * sin(17.5 * 3.1416 / 180) *
sin((17.5 + Torreta.m_nAngulo) * 3.1415 / 180));
Poligono35[2].x = m_fFactorZoom * (m_pointProcesoZ + Torreta.m_nDimPrincipal * cos(17.5 * 3.1416 / 180) *
sin((17.5 + Torreta.m_nAngulo) * 3.1415 / 180));
Poligono35[2].y = m_fFactorZoom * (-m_pointProcesoX - Torreta.m_nDimPrincipal * cos(17.5 * 3.1416 / 180)
*cos((17.5 + Torreta.m_nAngulo) * 3.1415 / 180));
Poligono35[3].x = m_fFactorZoom * (m_pointProcesoZ - Torreta.m_nDimPrincipal * sin(17.5 * 3.1416 / 180) *
cos((17.5 + Torreta.m_nAngulo) * 3.1415 / 180));
Poligono35[3].y = m_fFactorZoom * (-m_pointProcesoX - Torreta.m_nDimPrincipal * sin(17.5 * 3.1416 / 180) *
sin((17.5 + Torreta.m_nAngulo) * 3.1415 / 180));

Tools[Tool].CreatePolygonRgn(Poligono35, 4, ALTERNATE);
break;

case 7:
```

```
Triangulo[0].x = m_fFactorZoom * (m_pointProcesoZ - Torreta.m_nDimPrincipal * cos(30 * 3.1416 / 180) * sin((30 +
Torreta.m_nAngulo) * 3.1415 / 180));
Triangulo[0].y = m_fFactorZoom * (-m_pointProcesoX + Torreta.m_nDimPrincipal * cos(30 * 3.1416 / 180) * cos((30
+ Torreta.m_nAngulo) * 3.1415 / 180));
Triangulo[1].x = m_fFactorZoom * (m_pointProcesoZ + Torreta.m_nDimPrincipal * sin(30 * 3.1416 / 180) * cos((30 +
Torreta.m_nAngulo) * 3.1415 / 180));
Triangulo[1].y = m_fFactorZoom * (-m_pointProcesoX + Torreta.m_nDimPrincipal * sin(30 * 3.1416 / 180) * sin((30 +
Torreta.m_nAngulo) * 3.1415 / 180));
Triangulo[2].x = m_fFactorZoom * (m_pointProcesoZ - Torreta.m_nDimPrincipal * sin(30 * 3.1416 / 180) * cos((30 +
Torreta.m_nAngulo) * 3.1415 / 180));
Triangulo[2].y = m_fFactorZoom * (-m_pointProcesoX - Torreta.m_nDimPrincipal * sin(30 * 3.1416 / 180) * sin((30 +
Torreta.m_nAngulo) * 3.1415 / 180));

Tools[Tool].CreatePolygonRgn(Triangulo, 3, ALTERNATE);
break;
                                        }
                }

                m_boolHerramRgnCreado = TRUE;

}
```

6.2.2.- OnSimulacionInstruccionSiguiente( ).

```
void CPasoRunPasoView::OnSimulacionInstruccionSiguiente()
{
m_boolCombinarProcede = FALSE;
// TODO: Add your command handler code here
if(!m_boolParadaProgramada)
{
m_pTabSimb = m_pTablaSimbolica->GetAt(m_posicionTablaSimbolica);

m_nDeltaX = m_pTabSimb->m_nXcoordenada;

m_nDeltaY = m_pTabSimb->m_nYcoordenada;

m_nDeltaZ = m_pTabSimb->m_nZcoordenada;

m_SvelocidadCabezal = m_pTabSimb->m_nScabezal;

m_FvelocidadAvance = m_pTabSimb->m_nFavance;

m_GfuncionPreparatoria = m_pTabSimb->m_nGPreparacion;

m_Icentro = m_pTabSimb->m_nICentro;

m_Kcentro = m_pTabSimb->m_nKcircunferencia;

m_MfuncionEspecial = m_pTabSimb->m_nMmiscelaneos;

int nherramienta = 0;

if (m_pTabSimb->m_nTherramienta >= 10 && m_pTabSimb->m_nTherramienta < 20)
                {
                nherramienta = 1;
                }
if (m_pTabSimb->m_nTherramienta >= 20 && m_pTabSimb->m_nTherramienta < 30)
                {
                nherramienta = 2;
                }
```

```
if (m_pTabSimb->m_nTherramienta >= 30 && m_pTabSimb->m_nTherramienta < 40)
		{
		nherramienta = 3;
		}
if (m_pTabSimb->m_nTherramienta >= 40 && m_pTabSimb->m_nTherramienta < 50)
		{
		nherramienta = 4;
		}
if (m_pTabSimb->m_nTherramienta >= 50 && m_pTabSimb->m_nTherramienta < 60)
		{
		nherramienta = 5;
		}
if (m_pTabSimb->m_nTherramienta >= 60 && m_pTabSimb->m_nTherramienta < 70)
		{
		nherramienta = 6;
		}
if (m_pTabSimb->m_nTherramienta >= 70 && m_pTabSimb->m_nTherramienta < 80)
		{
		nherramienta = 7;
		}

if (m_pTabSimb->m_nTherramienta >= 80 && m_pTabSimb->m_nTherramienta < 90)
		{
		nherramienta = 8;
		}

  if(m_Tool != nherramienta && (m_MfuncionEspecial = 6))//m_Tool = 1 por defecto
		{
		m_Tool = nherramienta;
		for (int j = 0; j <= 8; j++)
			{
			Tools[j].DeleteObject();
			}
		GenerarHerramientas();
		m_boolCambioDeHerramienta = TRUE;
		Invalidate();
		}
if (m_boolCambioDeHerramienta)
		{
		CSimuladorControlNumericoDoc* pDoc = (CSimuladorControlNumericoDoc*)GetDocument();
		if(pDoc->m_boolHerramientasPreparado)
			{
			Torreta.m_nPosicion			= pDoc->Portaherramienta[m_Tool].m_nPosicion;
			Torreta.m_nGeometria		= pDoc->Portaherramienta[m_Tool].m_nGeometria;
			Torreta.m_nDimPrincipal = pDoc->Portaherramienta[m_Tool].m_nDimPrincipal * 100;
			m_boolHerramientasPreparado = TRUE;
			}
		}
switch(m_pTabSimb->m_nMmiscelaneos)
		{
		case 0:
			m_boolParadaProgramada = TRUE;
			break;
		case 2:
			m_boolFinalPrograma = TRUE;
			m_boolHorario = FALSE;
			m_boolAntiHorario = FALSE;
			m_boolRefrigerante = FALSE;
			break;
		case 3:
			m_boolHorario = TRUE;
			m_boolParadaCabezal = FALSE;
			break;
		case 4:
```

```
                m_boolAntiHorario = TRUE;
                m_boolParadaCabezal = FALSE;
                break;
        case 5:
                m_boolParadaCabezal = TRUE;
                m_boolHorario = FALSE;
                m_boolAntiHorario = FALSE;
                break;
        case 8:
                m_boolRefrigerante = TRUE;
                break;
        case 9:
                m_boolRefrigerante = FALSE;
                break;
        }

if((m_nDeltaX != 0) || (m_nDeltaY != 0) || (m_nDeltaZ != 0))//Verifica si hay un cambio de posición para iniciar una
temporización
        {
        MSG mensaje;
        if (m_nDeltaX < 0)
                {
                m_nXnegativo = TRUE;
                }



        if (m_nDeltaZ < 0)
                {
                m_nZnegativo = TRUE;
                }
        switch(m_GfuncionPreparatoria)
                {
                case -1:
                        m_funSeleccionVelocidad_S();
                        if (m_Icentro == 0 && m_Kcentro == 0)//Solamente traslación
                                {
                                m_pointFinalX = m_pointInicialX + m_nDeltaX;
                                m_pointFinalY = m_pointInicialY + m_nDeltaY;
                                m_pointFinalZ = m_pointInicialZ + m_nDeltaZ;
                        if(m_FvelocidadAvance > 0 && m_FvelocidadAvance <= 199)//Avance de Trabajo en
                                 centésimas de mm/rev
                                {
                                m_nVx = (int)(m_SvelocidadCabezal * m_FvelocidadAvance * m_nDeltaX) /
                        (
                                        600 * sqrt(pow(m_nDeltaX, 2) + pow(m_nDeltaY, 2) +
                                        pow(m_nDeltaZ, 2)));  //Centesimas de milimetro / decimas de
                                        segundo
                                m_nVy = (int)(m_SvelocidadCabezal * m_FvelocidadAvance * m_nDeltaY) /
(
                                        600 * sqrt(pow(m_nDeltaX, 2) + pow(m_nDeltaY, 2) +
                                pow(m_nDeltaZ, 2)));
                                m_nVz = (int)(m_SvelocidadCabezal * m_FvelocidadAvance * m_nDeltaZ) /
                                (
                                        600 * sqrt(pow(m_nDeltaX, 2) + pow(m_nDeltaY, 2) +
                                pow(m_nDeltaZ, 2)));
                                }
                        if (m_FvelocidadAvance > 199 && m_FvelocidadAvance <= 500)//Avance Rápido en
mts/min
                                {
                                m_nVx = ((m_FvelocidadAvance * m_nDeltaX) / sqrt(pow(m_nDeltaX, 2) +
                                         pow(m_nDeltaY, 2) + pow(m_nDeltaZ, 2))) * 1000 / 600;
                                m_nVy = ((m_FvelocidadAvance * m_nDeltaY) / sqrt(pow(m_nDeltaX, 2) +
                                         pow(m_nDeltaY, 2) + pow(m_nDeltaZ, 2))) * 1000 / 600;
```

```
                    m_nVz = ((m_FvelocidadAvance * m_nDeltaZ) / sqrt(pow(m_nDeltaX, 2) +
                            pow(m_nDeltaY, 2) + pow(m_nDeltaZ, 2))) * 1000 / 600;
                    }
            if (m_FvelocidadAvance == 800)
                    {

                    }
            }//Final del if de movimiento de traslación
if(m_Icentro != 0 || m_Kcentro != 0)
            {
            m_nCentroX = m_pointProcesoX + m_Icentro;
            m_nCentroZ = m_pointProcesoZ + m_Kcentro;
            m_nRadioGiro = ceil(sqrt(pow(m_Icentro, 2) + pow(m_Kcentro, 2)));
            if(m_Icentro != 0)
                    {
                    m_nAnguloDeGiro = (int)((atan(m_Kcentro / m_Icentro)) * 180 / 3.14);
                    }
            else
                    {
                    if(m_Kcentro < 0)
                            {
                            if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) > 0)
                            m_nAnguloDeGiro = -90;
                            if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) < 0)
                            m_nAnguloDeGiro = 90;
                            }
                    if(m_Kcentro > 0)
                            {
                            if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) > 0)
                                    m_nAnguloDeGiro = -90;
                            if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) < 0)
                                    m_nAnguloDeGiro = 90;
                            }
                    }
            m_nAnguloGiroProceso = m_nAnguloDeGiro;
            m_pointFinalX = m_pointProcesoX + m_nDeltaX;
            m_pointFinalY = m_pointProcesoY + m_nDeltaY;
            m_pointFinalZ = m_pointProcesoZ + m_nDeltaZ;
            Invalidate();
            }//Final del If de movimiento circular
break;

case 0://G00 Posicionamiento Rápido
            m_SvelocidadCabezal = 1600;
            m_FvelocidadAvance = 199;
            m_nVx = (int)(m_SvelocidadCabezal * m_FvelocidadAvance * m_nDeltaX) / (
                    600 * sqrt(pow(m_nDeltaX, 2) + pow(m_nDeltaY, 2) + pow(m_nDeltaZ,
                    2)));//Centesimas de milimetro / decimas de segundo
            m_nVy = (int)(m_SvelocidadCabezal * m_FvelocidadAvance * m_nDeltaY) / (
                    600 * sqrt(pow(m_nDeltaX, 2) + pow(m_nDeltaY, 2) + pow(m_nDeltaZ, 2)));
            m_nVz = (int)(m_SvelocidadCabezal * m_FvelocidadAvance * m_nDeltaZ) / (
                    600 * sqrt(pow(m_nDeltaX, 2) + pow(m_nDeltaY, 2) + pow(m_nDeltaZ, 2)));
            m_pointFinalX = m_pointInicialX + m_nDeltaX;
            m_pointFinalY = m_pointInicialY + m_nDeltaY;
            m_pointFinalZ = m_pointInicialZ + m_nDeltaZ;
            break;
case 1: //Interpolación Lineal
            m_funSeleccionVelocidad_S();
                    m_pointFinalX = m_pointInicialX + m_nDeltaX;
            m_pointFinalY = m_pointInicialY + m_nDeltaY;
            m_pointFinalZ = m_pointInicialZ + m_nDeltaZ;
            if(m_FvelocidadAvance > 0 && m_FvelocidadAvance <= 199)//Avance de Trabajo en
            centésimas de mm/rev
                    {
```

```
                m_nVx = (int)(m_SvelocidadCabezal * m_FvelocidadAvance * m_nDeltaX) /
(
                        600 * sqrt(pow(m_nDeltaX, 2) + pow(m_nDeltaY, 2) +
                        pow(m_nDeltaZ, 2)));//Centesimas de milimetro / decimas de
                        segundo
                m_nVy = (int)(m_SvelocidadCabezal * m_FvelocidadAvance * m_nDeltaY) /
(
                        600 * sqrt(pow(m_nDeltaX, 2) + pow(m_nDeltaY, 2) +
pow(m_nDeltaZ, 2)));
                m_nVz = (int)(m_SvelocidadCabezal * m_FvelocidadAvance * m_nDeltaZ) /
(
                600 * sqrt(pow(m_nDeltaX, 2) + pow(m_nDeltaY, 2) + pow(m_nDeltaZ, 2)));
                }
                if (m_FvelocidadAvance > 199 && m_FvelocidadAvance <= 500)//Avance
                Rápido en mts/min
                        {
                        m_nVx = ((m_FvelocidadAvance * m_nDeltaX) /
                        sqrt(pow(m_nDeltaX, 2) + pow(m_nDeltaY, 2) + pow(m_nDeltaZ,
                        2))) * 1000 / 600;
                        m_nVy = ((m_FvelocidadAvance * m_nDeltaY) /
                        sqrt(pow(m_nDeltaX, 2) + pow(m_nDeltaY, 2) + pow(m_nDeltaZ,
                        2))) * 1000 / 600;
                        m_nVz = ((m_FvelocidadAvance * m_nDeltaZ) /
                        sqrt(pow(m_nDeltaX, 2) + pow(m_nDeltaY, 2) + pow(m_nDeltaZ,
                        2))) * 1000 / 600;
                        }
                if (m_FvelocidadAvance == 800)
                        {
                        }
                break;
        case 2:
                m_funSeleccionVelocidad_S();
                if(m_FvelocidadAvance > 0 && m_FvelocidadAvance <= 199)//Avance de
                Trabajo en centésimas de mm/rev
                        {
                        m_nVx = (int)(m_SvelocidadCabezal * m_FvelocidadAvance *
                        m_nDeltaX) / (600 * sqrt(pow(m_nDeltaX, 2) + pow(m_nDeltaY, 2)
                        + pow(m_nDeltaZ, 2)));//Centesimas de milimetro / decimas de
                        segundo
                        m_nVy = (int)(m_SvelocidadCabezal * m_FvelocidadAvance *
                        m_nDeltaY) / (600 * sqrt(pow(m_nDeltaX, 2) + pow(m_nDeltaY, 2)
                        + pow(m_nDeltaZ, 2)));
                        m_nVz = (int)(m_SvelocidadCabezal * m_FvelocidadAvance *
                        m_nDeltaZ) / (600 * sqrt(pow(m_nDeltaX, 2) + pow(m_nDeltaY, 2)
                        + pow(m_nDeltaZ, 2)));
                        }
                if (m_FvelocidadAvance > 199 && m_FvelocidadAvance <= 500)//Avance
                Rápido en mts/min
                        {
                        m_nVx = ((m_FvelocidadAvance * m_nDeltaX) /
                        sqrt(pow(m_nDeltaX, 2) + pow(m_nDeltaY, 2) + pow(m_nDeltaZ,
                        2))) * 1000 / 600;
                        m_nVy = ((m_FvelocidadAvance * m_nDeltaY) /
                        sqrt(pow(m_nDeltaX, 2) + pow(m_nDeltaY, 2) + pow(m_nDeltaZ,
                        2))) * 1000 / 600;

                        m_nVz = ((m_FvelocidadAvance * m_nDeltaZ) /
                        sqrt(pow(m_nDeltaX, 2) + pow(m_nDeltaY, 2) + pow(m_nDeltaZ,
                        2))) * 1000 / 600;
                        }
                m_nCentroX = m_pointProcesoX + m_Icentro;
                m_nCentroZ = m_pointProcesoZ + m_Kcentro;
                m_nRadioGiro = ceil(sqrt(pow(m_Icentro, 2) + pow(m_Kcentro, 2)));
                if(m_Icentro != 0)
```

```
                                        {
                                        m_nAnguloDeGiro = (int)((atan(m_Kcentro / m_Icentro)) * 180 /
3.14);
                                }
                        else
                                {
                                if(m_Kcentro < 0)
                                        {
                                        if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) > 0)
                                                m_nAnguloDeGiro = -90;
                                        if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) < 0)
                                                m_nAnguloDeGiro = 90;
                                        }
                                if(m_Kcentro > 0)
                                        {
                                        if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) > 0)
                                                m_nAnguloDeGiro = -90;
                                        if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) < 0)
                                                m_nAnguloDeGiro = 90;
                                        }
                                }
                        m_nAnguloGiroProceso = m_nAnguloDeGiro;
                        m_pointFinalX = m_pointProcesoX + m_nDeltaX;
                        m_pointFinalY = m_pointProcesoY + m_nDeltaY;
                        m_pointFinalZ = m_pointProcesoZ + m_nDeltaZ;
                        Invalidate();
                        break;

                case 3:
                        m_funSeleccionVelocidad_S();
                        if(m_FvelocidadAvance > 0 && m_FvelocidadAvance <= 199)//Avance de Trabajo en
                                                                                centésimas de mm/rev
                                {
                                m_nVx = (int)(m_SvelocidadCabezal * m_FvelocidadAvance * m_nDeltaX) /
                                (
                                        600 * sqrt(pow(m_nDeltaX, 2) + pow(m_nDeltaY, 2) +
                                        pow(m_nDeltaZ, 2)));//Centesimas de milimetro / decimas de
                                        segundo
                                m_nVy = (int)(m_SvelocidadCabezal * m_FvelocidadAvance * m_nDeltaY) /
(
                                        600 * sqrt(pow(m_nDeltaX, 2) + pow(m_nDeltaY, 2) +
                                pow(m_nDeltaZ, 2)));
                                m_nVz = (int)(m_SvelocidadCabezal * m_FvelocidadAvance * m_nDeltaZ) /
                                (
                                        600 * sqrt(pow(m_nDeltaX, 2) + pow(m_nDeltaY, 2) +
                                pow(m_nDeltaZ, 2)));
                                }
                        if (m_FvelocidadAvance > 199 && m_FvelocidadAvance <= 500)//Avance Rápido en
                mts/min
                                {
                                m_nVx = ((m_FvelocidadAvance * m_nDeltaX) / sqrt(pow(m_nDeltaX, 2) +
                                         pow(m_nDeltaY, 2) + pow(m_nDeltaZ, 2))) * 1000 / 600;
                                m_nVy = ((m_FvelocidadAvance * m_nDeltaY) / sqrt(pow(m_nDeltaX, 2) +
                                         pow(m_nDeltaY, 2) + pow(m_nDeltaZ, 2))) * 1000 / 600;
                                m_nVz = ((m_FvelocidadAvance * m_nDeltaZ) / sqrt(pow(m_nDeltaX, 2) +
                                         pow(m_nDeltaY, 2) + pow(m_nDeltaZ, 2))) * 1000 / 600;
                                }
                        m_nCentroX = m_pointProcesoX + m_Icentro;
                        m_nCentroZ = m_pointProcesoZ + m_Kcentro;
                        m_nRadioGiro = ceil(sqrt(pow(m_Icentro, 2) + pow(m_Kcentro, 2)));
                        if(m_Icentro != 0)
                                {
                                m_nAnguloDeGiro = (int)((atan(m_Kcentro / m_Icentro)) * 180 / 3.14);
                                }
```

```
                    else
                        {
                        if(m_Kcentro < 0)
                            {
                            if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) > 0)
                                m_nAnguloDeGiro = -90;
                            if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) < 0)
                                m_nAnguloDeGiro = 90;
                            }
                        if(m_Kcentro > 0)
                            {
                            if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) > 0)
                            m_nAnguloDeGiro = -90;
                            if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) < 0)
                                m_nAnguloDeGiro = 90;
                            }
                        }
                    m_nAnguloGiroProceso = m_nAnguloDeGiro;
                    m_pointFinalX = m_pointProcesoX + m_nDeltaX;
                    m_pointFinalY = m_pointProcesoY + m_nDeltaY;
                    m_pointFinalZ = m_pointProcesoZ + m_nDeltaZ;
                    Invalidate();
                    break;
                }
                m_nTimerPasoPaso = SetTimer(1, 100, NULL);//100/1000 = Un mensaje cada 1/10 segundos
                m_boolDeltaX = m_boolDeltaY = m_boolDeltaZ = FALSE;
                if (m_nXnegativo)
                    {
                    m_nVx = -1 * abs(m_nVx);
                    }
                if (m_nZnegativo)
                    {
                    m_nVz = -1 * abs(m_nVz);
                    }
                while((m_pointProcesoX != m_pointFinalX) || (m_pointProcesoY != m_pointFinalY) ||
                (m_pointProcesoZ != m_pointFinalZ) )
                    {
                    if (::PeekMessage(&mensaje, NULL, 0, 0, PM_REMOVE))
                        {
                        ::TranslateMessage(&mensaje);
                        ::DispatchMessage(&mensaje);
                        }
                    }
                KillTimer(1);
                m_nXnegativo = FALSE;
                m_nZnegativo = FALSE;
                m_pointInicialX = m_pointProcesoX;
                m_pointInicialY = m_pointProcesoY;
                m_pointInicialZ = m_pointProcesoZ;
                m_nVx = m_nVz = m_nVy = 0;
                m_Icentro = m_Kcentro = 0;
                }
        if(m_posicionTablaSimbolica != NULL)
                {
                m_pTabSimb = m_pTablaSimbolica->GetNext(m_posicionTablaSimbolica);
                boolFinalPrograma = FALSE;
                }
        if(m_posicionTablaSimbolica == NULL)
                {
                CDialogoFinalPasoPasoRun dialogo;
                int ret = dialogo.DoModal();
                boolFinalPrograma = TRUE;
                }
        }//Final ParadaProgramada
```

```
}
```

6.2.3.- OnTimer( ).

```
void CPasoRunPasoView::OnTimer(UINT nIDEvent)
{
m_boolCombinarProcede = TRUE;
CRgn rgnVirutaAnt;
rgnVirutaAnt.CreateRectRgn(0, 0, 0, 0);
rgnVirutaAnt.CombineRgn(&m_rgnTocho, &Tools[m_Tool], RGN_AND);
InvalidateRgn(&rgnVirutaAnt, FALSE);
rgnVirutaAnt.DeleteObject();
InvalidateRgn(&Tools[m_Tool], FALSE);
InvalidateRgn(&m_rgnTocho,FALSE);
CRect rectTexto(m_pointProcesoZ,
                    -m_pointProcesoX,
                     m_pointProcesoZ + 3000,
                    -m_pointProcesoX - 3000);
InvalidateRect(rectTexto);
m_pointInicialX = m_pointProcesoX;
m_pointInicialZ = m_pointProcesoZ;
m_nPointXoffset = 0;
m_nPointZoffset = 0;
if (m_Icentro == 0 && m_Kcentro == 0)//Solamente traslación
        {//Inicio del If de Movimiento lineal en OnTimer
         if(m_nXnegativo)
                {//Comiienzo del If de Velocidad negativa en X
                if(m_pointProcesoX > m_pointFinalX)
                        {//Inicio del If para acercándose por la derecha
                        m_nPointXoffset = m_pointProcesoX;
                        m_pointProcesoX += m_nVx;
                         m_nPointXoffset = m_pointProcesoX - m_nPointXoffset;
                        }//Final del If para acercándose por la dereceha
                  if(m_pointProcesoX < m_pointFinalX)
                        {//Inicio del If para rebasar el límite por la derecha
                         m_nPointXoffset = m_pointProcesoX;
                        m_pointProcesoX = m_pointFinalX;
                        m_nPointXoffset = m_pointProcesoX - m_nPointXoffset;
                        }//Final del If para rebasar el límite por la derecha
                }//Final del if de Velocidad negativa en X
        else
                {//Comienzo del else de Velocidad positiva en X
                if(m_pointProcesoX < m_pointFinalX)
                        {//Inicio del If para acercándose por la izquierda
                        m_nPointXoffset = m_pointProcesoX;
                        m_pointProcesoX += m_nVx;
                        m_nPointXoffset = m_pointProcesoX - m_nPointXoffset;
                        }//Final del If para acercándose por la izquierda
                if(m_pointProcesoX > m_pointFinalX)
                        {//Inicio del If para rebasar el límite por la izquierda
                        m_nPointXoffset = m_pointProcesoX;
                        m_pointProcesoX = m_pointFinalX;
                        m_nPointXoffset = m_pointProcesoX - m_nPointXoffset;
                        }//Final del If para rebasar el límite por la izquierda
                }//Final del else de Velocidad positiva en X
                //***************Final cambio de posición en X
//**********Cambio de posición en Z
        if(m_nZnegativo)
                {//Comiienzo del If de Velocidad negativa en Z
                if(m_pointProcesoZ > m_pointFinalZ)
                        {//Inicio del If para acercándose por la derecha
                                m_nPointZoffset = m_pointProcesoZ;
```

```
                                        m_pointProcesoZ += m_nVz;
                                        m_nPointZoffset = m_pointProcesoZ - m_nPointZoffset;
                                }//Final del If para acercándose por la dereceha
                        if(m_pointProcesoZ < m_pointFinalZ)
                                {//Inicio del If para rebasar el límite por la derecha
                                m_nPointZoffset = m_pointProcesoZ;
                                m_pointProcesoZ = m_pointFinalZ;
                                m_nPointZoffset = m_pointProcesoZ - m_nPointZoffset;
                                }//Final del If para rebasar el límite por la derecha
                        }//Final del if de Velocidad negativa en Z
                else
                        {//Comienzo del else de Velocidad positiva en Z
                        if(m_pointProcesoZ < m_pointFinalZ)
                                {//Inicio del If para acercándose por la izquierda
                                m_nPointZoffset = m_pointProcesoZ;
                                m_pointProcesoZ += m_nVz;
                                m_nPointZoffset = m_pointProcesoZ - m_nPointZoffset;
                                }//Final del If para acercándose por la izquierda
                        if(m_pointProcesoZ > m_pointFinalZ)
                                {//Inicio del If para rebasar el límite por la izquierda
                                m_nPointZoffset = m_pointProcesoZ;
                                m_pointProcesoZ = m_pointFinalZ;
                                m_nPointZoffset = m_pointProcesoZ - m_nPointZoffset;
                                }//Final del If para rebasar el límite por la izquierda
                        }//Final del else de Velocidad positiva en Z
//***************Final cambio de posición en Z
                }//Final de Movimiento Lineal en OnTimer
if((m_Icentro != 0 || m_Kcentro != 0))
                {
                if(m_Icentro < 0)
                        {//Centro por encima del punto en movimiento
                        if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) > 0)//Ojo aqui***********
                                m_nAnguloGiroProceso++;
                        if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) < 0)//Ojo Aquí**********
                                m_nAnguloGiroProceso--;
                        if(m_Kcentro > 0)
                                {//Centro a la derecha y por encima del punto en movimiento
                                if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) < 0)
                                        {//Inicio sentido horario con centro a la derecha y por encima delpunto en
movimiento
                                        if(m_pointProcesoX > m_pointFinalX)
                                                {
                                                m_nPointXoffset = m_pointProcesoX;
                                                m_pointProcesoX = m_nCentroX + m_nRadioGiro *
                                                cos(m_nAnguloGiroProceso * 3.1416 / 180);
                                                m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
                                                }
                                        else
                                                {
                                                m_nPointXoffset = m_pointProcesoX;
                                                m_pointProcesoX = m_pointFinalX;
                                                m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
                                                }
                                        if(m_pointProcesoZ > m_pointFinalZ)
                                                {
                                                m_nPointZoffset = m_pointProcesoZ;
                                                m_pointProcesoZ = m_nCentroZ + m_nRadioGiro *
                                                sin(m_nAnguloGiroProceso * 3.1416 / 180);
                                                m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
                                                }
                                        else
                                                {
                                                m_nPointZoffset = m_pointProcesoZ;
                                                m_pointProcesoZ = m_pointFinalZ;
```

```
                                        m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
                                        }
                }//Final sentido horario con centro a la derecha y por encima delpunto en movimiento
        if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) > 0)
                {//Inicio sentido antihorario con centro a la derecha y por encima delpunto en movimiento
                if(m_pointProcesoX < m_pointFinalX)
                        {
                        m_nPointXoffset = m_pointProcesoX;
                        m_pointProcesoX = m_nCentroX + m_nRadioGiro * cos(m_nAnguloGiroProceso *
                        3.1416 / 180);
                        m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
                        }
                else
                        {
                        m_nPointXoffset = m_pointProcesoX;
                        m_pointProcesoX = m_pointFinalX;
                        m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
                        }
                if(m_pointProcesoZ < m_pointFinalZ)
                        {
                        m_nPointZoffset = m_pointProcesoZ;
                        m_pointProcesoZ = m_nCentroZ + m_nRadioGiro * sin(m_nAnguloGiroProceso *
                3.1416 / 180);
                        m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
                        }
                else
                        {
                        m_nPointZoffset = m_pointProcesoZ;
                        m_pointProcesoZ = m_pointFinalZ;
                        m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
                        }

                }//Final sentido antihorario con centro a la derecha y por encima delpunto en movimiento
        }//Final centro a la derecha y por encima del punto en movimiento
if(m_Kcentro < 0)
        {//Inicio centro a la izquierda y por encima del punto en movimiento
        if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) < 0)
                {///Inicio sentido horario con centro a la izquierda y por encima delpunto en movimiento
                if(m_pointProcesoX < m_pointFinalX)
                        {
                        m_nPointXoffset = m_pointProcesoX;
                        m_pointProcesoX = m_nCentroX + m_nRadioGiro * cos(m_nAnguloGiroProceso *
                        3.1416 / 180);
                        m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
                        }
                else
                        {
                        m_nPointXoffset = m_pointProcesoX;
                        m_pointProcesoX = m_pointFinalX;
                        m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
                        }
        if(m_pointProcesoZ > m_pointFinalZ)
                {
                m_nPointZoffset = m_pointProcesoZ;
                m_pointProcesoZ = m_nCentroZ + m_nRadioGiro * sin(m_nAnguloGiroProceso * 3.1416 / 180);
                m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
                }
        else
                {
                m_nPointZoffset = m_pointProcesoZ;
                m_pointProcesoZ = m_pointFinalZ;
                m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
                }
        }//Final sentido horario con centro a la izquierda y por encima delpunto en movimiento
```

```
if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) > 0)
	{//Inicio sentido antihorario con centro a la izquierda y por encima delpunto en movimiento
	if(m_pointProcesoX > m_pointFinalX)
		{
		m_nPointXoffset = m_pointProcesoX;
		m_pointProcesoX = m_nCentroX + m_nRadioGiro * cos(m_nAnguloGiroProceso * 3.1416 / 180);
		m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
		}
	else
		{
		m_nPointXoffset = m_pointProcesoX;
		m_pointProcesoX = m_pointFinalX;
		m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
		}
	if(m_pointProcesoZ < m_pointFinalZ)
		{
		m_nPointZoffset = m_pointProcesoZ;
		m_pointProcesoZ = m_nCentroZ + m_nRadioGiro * sin(m_nAnguloGiroProceso * 3.1416 / 180);
		m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
		}
	else
		{
		m_nPointZoffset = m_pointProcesoZ;
		m_pointProcesoZ = m_pointFinalZ;
		m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
		}
	}//Final sentido antihorario con centro a la izquierda y por encima delpunto en movimiento
}//Final Centro a la izquierda y por encima del punto en movimiento
}//Final Centro por encima del punto en movimiento
if(m_Kcentro == 0)
	{//Inicio centro por encima del punto en movimiento y en el medio
	if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) > 0)
		{//Inicio I < 0 K = 0 y antihorario
		if(m_pointProcesoX > m_pointFinalX)
			{
			m_nPointXoffset = m_pointProcesoX;
			m_pointProcesoX = m_nCentroX + ceil(m_nRadioGiro * cos(m_nAnguloGiroProceso *
			3.1416 / 180));
			m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
			}
		else
			{
			m_nPointXoffset = m_pointProcesoX;
			m_pointProcesoX = m_pointFinalX;
			m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
			}
		if(m_pointProcesoZ < m_pointFinalZ)
			{
			m_nPointZoffset = m_pointProcesoZ;
			m_pointProcesoZ = m_nCentroZ + ceil(m_nRadioGiro * sin(m_nAnguloGiroProceso *
			3.1416 / 180));
			m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
			}
		else
			{
			m_nPointZoffset = m_pointProcesoZ;
			m_pointProcesoZ = m_pointFinalZ;
			m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
			}
	}//Final I < 0 K = 0 y horario
if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) < 0)
	{//Inicio I < 0 K = 0 y Antihorario
	if(m_pointProcesoX > m_pointFinalX)
		{
```

```
                        m_nPointXoffset = m_pointProcesoX;
                        m_pointProcesoX = m_nCentroX + ceil(m_nRadioGiro * cos(m_nAnguloGiroProceso * 3.1416 /
180));
                        m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
                        }
                else
                        {
                        m_nPointXoffset = m_pointProcesoX;
                        m_pointProcesoX = m_pointFinalX;
                        m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
                        }
                if(m_pointProcesoZ > m_pointFinalZ)
                        {
                        m_nPointZoffset = m_pointProcesoZ;
                        m_pointProcesoZ = m_nCentroZ + floor(m_nRadioGiro * sin(m_nAnguloGiroProceso * 3.1416 /
                180));
                        m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
                        }
                else
                        {
                        m_nPointZoffset = m_pointProcesoZ;
                        m_pointProcesoZ = m_pointFinalZ;
                        m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
                        }
                }//Final I < 0 K = 0 y horario
}//Final centro por encima del punto en movimiento y en el medio
///////////////*********************************************
//A Partir de aquí esta copiado el código para Centro por debajo del movimiento
if(m_Icentro > 0)
        {//Centro por debajo del punto en movimiento
        if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) > 0)//Ojo aqui**********
                m_nAnguloGiroProceso++;
        if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) < 0)//Ojo Aquí**********
                m_nAnguloGiroProceso--;
        if(m_Kcentro > 0)
                {//Centro a la derecha y por debajo del punto en movimiento
                if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) < 0)
                        {//Inicio sentido horario con centro a la derecha y por debajo delpunto en movimiento
                        if(m_pointProcesoX > m_pointFinalX)
                                {
                                m_nPointXoffset = m_pointProcesoX;
                                m_pointProcesoX = m_nCentroX - m_nRadioGiro *
                                cos(m_nAnguloGiroProceso * 3.1416 / 180);
                                m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
                                }
                        else
                                {
                                m_nPointXoffset = m_pointProcesoX;
                                m_pointProcesoX = m_pointFinalX;
                                m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
                                }
                        if(m_pointProcesoZ < m_pointFinalZ)
                                {
                                m_nPointZoffset = m_pointProcesoZ;
                                m_pointProcesoZ = m_nCentroZ - m_nRadioGiro *
                        sin(m_nAnguloGiroProceso *  3.1416 / 180);
                                m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
                                }
                        else
                                {
                                m_nPointZoffset = m_pointProcesoZ;
                                m_pointProcesoZ = m_pointFinalZ;
                                m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
                                }
```

```
                }//Final sentido horario con centro a la derecha y por debajo delpunto en movimiento
                if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) > 0)
                        {//Inicio sentido antihorario con centro a la derecha y por debajo delpunto en
movimiento
                        if(m_pointProcesoX < m_pointFinalX)
                                {
                                m_nPointXoffset = m_pointProcesoX;
                                m_pointProcesoX = m_nCentroX - m_nRadioGiro *
                                cos(m_nAnguloGiroProceso * 3.1416 / 180);
                                m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
                                }
                        else
                                {
                                m_nPointXoffset = m_pointProcesoX;
                                m_pointProcesoX = m_pointFinalX;
                                m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
                                }
                        if(m_pointProcesoZ > m_pointFinalZ)
                                {
                                m_nPointZoffset = m_pointProcesoZ;
                                m_pointProcesoZ = m_nCentroZ - m_nRadioGiro *
                                sin(m_nAnguloGiroProceso * 3.1416 / 180);
                                m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
                                }
                        else
                                {
                                m_nPointZoffset = m_pointProcesoZ;
                                m_pointProcesoZ = m_pointFinalZ;
                                m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
                                }
                        }//Final sentido antihorario con centro a la derecha y por debajo delpunto en
movimiento
                }//Final centro a la derecha y por debajo del punto en movimiento
        if(m_Kcentro < 0)
                {//Inicio centro a la izquierda y por debajo del punto en movimiento
                if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) < 0)
                        {//Inicio sentido horario con centro a la izquierda y por debajo delpunto en movimiento
                        if(m_pointProcesoX < m_pointFinalX)
                                {
                                m_nPointXoffset = m_pointProcesoX;
                                m_pointProcesoX = m_nCentroX - m_nRadioGiro *
                                cos(m_nAnguloGiroProceso * 3.1416 / 180);
                                m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
                                }
                        else
                                {
                                m_nPointXoffset = m_pointProcesoX;
                                m_pointProcesoX = m_pointFinalX;
                                m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
                                }

                        if(m_pointProcesoZ < m_pointFinalZ)
                                {
                                m_nPointZoffset = m_pointProcesoZ;
                                m_pointProcesoZ = m_nCentroZ - m_nRadioGiro *
                                sin(m_nAnguloGiroProceso * 3.1416 / 180);
                                m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
                                }
                        else
                                {
                                m_nPointZoffset = m_pointProcesoZ;
                                m_pointProcesoZ = m_pointFinalZ;
                                m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
                                }
```

```
                    }//Final sentido horario con centro a la izquierda y por debajo delpunto en movimiento
                if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) > 0)
                    {//Inicio sentido antihorario con centro a la izquierda y por debajo delpunto en
movimiento
                    if(m_pointProcesoX > m_pointFinalX)
                        {
                        m_nPointXoffset = m_pointProcesoX;
                        m_pointProcesoX = m_nCentroX - m_nRadioGiro *
                        cos(m_nAnguloGiroProceso * 3.1416 / 180);
                        m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
                        }
                    else
                        {
                        m_nPointXoffset = m_pointProcesoX;
                        m_pointProcesoX = m_pointFinalX;
                        m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
                        }
                    if(m_pointProcesoZ > m_pointFinalZ)
                        {
                        m_nPointZoffset = m_pointProcesoZ;
                        m_pointProcesoZ = m_nCentroZ - m_nRadioGiro *
                        sin(m_nAnguloGiroProceso * 3.1416 / 180);
                        m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
                        }
                    else
                        {
                        m_nPointZoffset = m_pointProcesoZ;
                        m_pointProcesoZ = m_pointFinalZ;
                        m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
                        }
                    }//Final sentido horario con centro a la izquierda y por debajo delpunto en movimiento
                }//Final Centro a la izquierda y por debajo del punto en movimiento
            if(m_Kcentro == 0)
                {//Inicio centro en el medio y por debajo del punto en movimiento
                if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) < 0)
                    {//Inicio sentido horario con centro en el medio y por debajo delpunto en movimiento
                    if(m_pointProcesoX < m_pointFinalX)
                        {
                        m_nPointXoffset = m_pointProcesoX;
                        m_pointProcesoX = m_nCentroX - m_nRadioGiro *
                        cos(m_nAnguloGiroProceso * 3.1416 / 180);
                        m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
                        }
                    else
                        {
                        m_nPointXoffset = m_pointProcesoX;
                        m_pointProcesoX = m_pointFinalX;
                        m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
                        }
                    if(m_pointProcesoZ < m_pointFinalZ)
                        {
                        m_nPointZoffset = m_pointProcesoZ;
                        m_pointProcesoZ = m_nCentroZ - m_nRadioGiro *
                        sin(m_nAnguloGiroProceso * 3.1416 / 180);
                        m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
                        }
                    else
                        {
                        m_nPointZoffset = m_pointProcesoZ;
                        m_pointProcesoZ = m_pointFinalZ;
                        m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
                        }
                    }//Final sentido horario con centro en el medio y por debajo delpunto en movimiento
                if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) > 0)
```

```
                                {//Inicio sentido antihorario con centro a la izquierda y por debajo delpunto en
movimiento
                                if(m_pointProcesoX < m_pointFinalX)
                                        {
                                        m_nPointXoffset = m_pointProcesoX;
                                        m_pointProcesoX = m_nCentroX - m_nRadioGiro *
                                        cos(m_nAnguloGiroProceso * 3.1416 / 180);
                                        m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
                                        }
                                else
                                        {
                                        m_nPointXoffset = m_pointProcesoX;
                                        m_pointProcesoX = m_pointFinalX;
                                        m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
                                        }
                                if(m_pointProcesoZ > m_pointFinalZ)
                                        {
                                        m_nPointZoffset = m_pointProcesoZ;
                                        m_pointProcesoZ = m_nCentroZ - (floor(m_nRadioGiro *
                                        sin(m_nAnguloGiroProceso * 3.1416 / 180)));
                                        m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
                                        }
                                else
                                        {
                                        m_nPointZoffset = m_pointProcesoZ;
                                        m_pointProcesoZ = m_pointFinalZ;
                                        m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
                                        }
                                }//Final sentido horario con centro a la izquierda y por debajo delpunto en movimiento
                        }//Final Centro a la izquierda y por debajo del punto en movimiento
}//Final Centro por debajo del punto en movimiento
///////////////************************************************
if(m_Icentro == 0)
        {//Inicio I = 0
        if(m_Kcentro > 0)
                {//Inicio I = 0 y K > 0
                if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) > 0)
                        {//Inicio I = 0 y K > 0 y Sentido antihorario
                        m_nAnguloGiroProceso++;
                        if(m_pointProcesoX < m_pointFinalX)
                                {
                                m_nPointXoffset = m_pointProcesoX;
                                m_pointProcesoX = m_nCentroX + ceil(m_nRadioGiro *
                                cos(m_nAnguloGiroProceso * 3.1416 / 180));
                                m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
                                }
                        else
                                {
                                m_nPointXoffset = m_pointProcesoX;
                                m_pointProcesoX = m_pointFinalX;
                                m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
                                }
                        if(m_pointProcesoZ < m_pointFinalZ)
                                {
                                m_nPointZoffset = m_pointProcesoZ;
                                m_pointProcesoZ = m_nCentroZ + ceil(m_nRadioGiro *
                                sin(m_nAnguloGiroProceso * 3.1416 / 180));
                                m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
                                }
                        else
                                {
                                m_nPointZoffset = m_pointProcesoZ;
                                m_pointProcesoZ = m_pointFinalZ;
                                m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
```

```
                                        }
                                }//Final I = 0 y K > 0 y Sentido antihorario
                        if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) < 0)
                                {//Inicio I = 0 y K > 0 y Sentido horario
                                m_nAnguloGiroProceso--;
                                if(m_pointProcesoX > m_pointFinalX)
                                        {
                                        m_nPointXoffset = m_pointProcesoX;
                                        m_pointProcesoX = m_nCentroX - ceil(m_nRadioGiro *
                                        cos(m_nAnguloGiroProceso * 3.1416 / 180));
                                        m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
                                        }
                                else
                                        {
                                        m_pointProcesoX = m_pointFinalX;
                                        }
                                if(m_pointProcesoZ < m_pointFinalZ)
                                        {
                                        m_nPointZoffset = m_pointProcesoZ;
                                        m_pointProcesoZ = m_nCentroZ - ceil(m_nRadioGiro *
                                        sin(m_nAnguloGiroProceso * 3.1416 / 180));
                                        m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
                                        }
                                else
                                        {
                                        m_pointProcesoZ = m_pointFinalZ;
                                        }

                                }//Final I = 0 y K > 0 y Sentido horario
                        }//Final I = 0 y K > 0
//*****************************************************
//Aqui copio I = 0 y k < 0
if(m_Kcentro < 0)
        {//Inicio I = 0 y K < 0
        if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) > 0)
                {//Inicio I = 0 y K < 0 y Sentido antihorario
                m_nAnguloGiroProceso++;
                if(m_pointProcesoX > m_pointFinalX)
                        {
                        m_nPointXoffset = m_pointProcesoX;
                        m_pointProcesoX = m_nCentroX - ceil(m_nRadioGiro * cos(m_nAnguloGiroProceso *
                        3.1416 / 180));
                        m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
                        }
                else
                        {
                        m_nPointXoffset = m_pointProcesoX;
                        m_pointProcesoX = m_pointFinalX;
                        m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
                        }
                if(m_pointProcesoZ > m_pointFinalZ)
                        {
                        m_nPointZoffset = m_pointProcesoZ;
                        m_pointProcesoZ = m_nCentroZ - ceil(m_nRadioGiro * sin(m_nAnguloGiroProceso *
                        3.1416 / 180));
                        m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
                        }
                else
                        {
                        m_nPointZoffset = m_pointProcesoZ;
                        m_pointProcesoZ = m_pointFinalZ;
                        m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
                        }
```

```
                }//Final I = 0 y K < 0 y Sentido antihorario
        if((m_Kcentro * m_nDeltaX - m_Icentro * m_nDeltaZ) < 0)
                {//Inicio I = 0 y K < 0 y Sentido horario
                m_nAnguloGiroProceso--;
                if(m_pointProcesoX < m_pointFinalX)
                        {
                        m_nPointXoffset = m_pointProcesoX;
                        m_pointProcesoX = m_nCentroX + m_nRadioGiro * cos(m_nAnguloGiroProceso *
                        3.1416 / 180);
                        m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
                        }
                else
                        {
                        m_nPointXoffset = m_pointProcesoX;
                        m_pointProcesoX = m_pointFinalX;
                        m_nPointXoffset = m_pointProcesoX  - m_nPointXoffset;
                        }
                if(m_pointProcesoZ > m_pointFinalZ)
                        {
                        m_nPointZoffset = m_pointProcesoZ;
                        m_pointProcesoZ = m_nCentroZ + m_nRadioGiro * sin(m_nAnguloGiroProceso *
                3.1416 / 180);
                        m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
                        }
                else
                        {
                        m_nPointZoffset = m_pointProcesoZ;
                        m_pointProcesoZ = m_pointFinalZ;
                        m_nPointZoffset = m_pointProcesoZ  - m_nPointZoffset;
                        }
                }//Final I = 0 y K > 0 y Sentido horario
        }//Final I = 0 y K > 0
}//Final I = 0
}
TRACE("\n\nSaliendo de OnTimer\n\n");
CRgn rgnVirutaSig;
rgnVirutaSig.CreateRectRgn(0, 0, 0, 0);
rgnVirutaSig.CombineRgn(&m_rgnTocho, &Tools[m_Tool], RGN_AND);
InvalidateRgn(&rgnVirutaSig, FALSE);
rgnVirutaSig.DeleteObject();
InvalidateRgn(&Tools[m_Tool], FALSE);
InvalidateRgn(&m_rgnTocho,FALSE);

}
```

7.- Bibliografía

Abdel-MaIeK, K.; H. Yeh y N. Maropis, 1998, "Determining Interference between Pairs of Defined Constructively in Computer Animation", Engineering with Computers, Vol 14, No. 1, pp. 48-58.

Atfison, C., 1993, "Code Capsules: A C++ Data Class, Parte I". The Users Journal, Vol. 11, No 2, Febrero 1993, pp. 120-131.

ANSI, 1990,"American National Standard for Information Systems Programming Language(ANSI Document ANSI/ISO 9899)", American National Standard Institute, New York.

Ban-on, Ch. ,1982, "Numerical control for machine tool". Me Graw-hBl, New York.

Bruins, D.H.. 1981, "Herramientas y maquinas herram»enta", Urmo, Bilbao.

Byron, D., 1993, "The Case for Object Technology Standards", CASE Trends, septiembre, pp. 22-26.

Candia, C., Ribis, S., 1984, "Acoplamiento de una unidad de control numerico de tres ejes a una fresadora", Caracas.

Chads, J., 1969, "Principles of numerical control". Industrial Press, New York.

Computerworid, 1993, "TTie CW Guide to Object Oriented Programming", Computerworid, Junio, pp. 107-130.

De Garmo, E. P., 1969, "Materiales y procesos de Fabricadon", Reverte, Buenos Aires.

De Lima, H., 1974, "Programación del control numérico de maquinas herramienta", Caracas.

Deitel, H. M.. 1994. "Como programar en C/C++", Prentice Hall, Mexico.

Duran, p.; 0., Avila, 1968, "Aplicación del control numérico a la fabricación de partes mecánicas", Caracas.

H. Ernault Somua, “Operator`s Manual, Setting and Maintenance 600N”, H. Ernault Somua, Versailles.

Hagan. T., 1993, "C++ Class Libraries for GUIs", Open Systems Today, Febrero 15, pp. 54-58.

Horten, H. E., 1969, "Control numérico de las máquinas herramienta,

Garcia, M. A., 1971, "Maquinas Herramienta para ingenieros", Urmo, Bilbao.

General Electric, “Mark Century 550LM Two Axes Contourning Control”, General electric, numerical equipment control department, Waynesboro. V.A.

Geriing, H., 1986, "Alrededor de las maquinas herramienta", Reverte, Barcelona.

Harden, C., 1996, An integrated System for Computer Aided Dosing and Analisys of Muttibody", Engineering with Computers, VolH.No.l.i pp. 23-33.

Hsieh, S. H. y E. D. Sotelino, 1997, "A Message-Passing Class Library C++ for Portable Parallel Programming", Engineering with Computers, Vol.13.No. 1, pp. 20-34.

Kibbe, P., 1985, "Manual de maquinas herramienta, Umusa. Mexico.

Kozaczynski, W.; A. Kuntzmann-Combelles, 1993, "What It Takes to Make OO Work", 1EE Software Magazine, Vol 10, No. 1, Enero. pp.20-23.

Kruglinsky, D. K, 1997, “Programación avanzada con Microsoft Visual C++, McGraw-Hill, Madrid.

Lakmazaheri, S., 1998, "Logic-Based 2D Geometric Modeling in a Cad Environment", Engineering with Computers, Vol. 14, No 2, pp. 123-138.

Lo, S. H., 1998, "Analysis and Verification of The Boundary of Solids Objects", Engineering with Computers, Vol. 14, No. 1, pp 36-47.

Matsche, J., 1993, "Object-oriented programming in Standard C', Object Magazine, Vol. 2, No 5, Enero-Febrero, pp. 71-74.

Medland, A.; G. Muffineux y A. H. Rentoul, 1995, "Introducing Solids Objects into a constraint Modelling System.", Engineering with Computers, Vol 11, No. 1, pp. 27-35.

Ogata, K., 1990, "Ingenieria de Control Modema", Prentice Hall, Mexico.

Patton, W., 1972, "Numerical Control practice and application", Reston Pub. Co., Reston.

Petzold, Ch., 1996, “Programación en Windows 95”, McGraw-Hill, Madrid.

Pittman, M., 1993, "Lessons Learned In Managing Object-Oriented Development", IEEE Software Magazine, Vol. 10, No. 1, Enero, pp. 43-53

Prieto, R., 1993, "Status Report: Software Reusability", IEEE Software Magazine, Vol 10, No. 3, Mayo. pp. 61-66.

Redondo, J., 1969, "Maquinas Herramienta con control numerico", Dossat, Madrid.

Reed, D.R., 1991, "Moving from C to C++". Object Magazine, Vol. 1, No. 3, Septiembre/Octubre, pp. 46-60.

Rettkj, M.; G. Simmons; J. Thompson, 1993, "Extended Objects", Communications os the ACM, Vol 36, No. 8, Agosto, pp. 19-24.

Ritctte, D.; S. Johson; M. LesK; B. Kernighan, 1978, UNIX Time-Snaring System: The C Programming Language", The Bell System Technical Journal, Vol. 57, No. 6 Part 2, Julio – Agosto 1978, pp. 1991-2019.

Ritchie, D. M., 1984, The UNIX System: The Evolution of the UNIX Time-Sharing System[1]', AT&T Bell Laboratories Technical Journal,' Vol. 63, No 8, Part 2, octubre de 1984, pp 1577-1593.

Rodnguez, 0.; J.. Sabarich, 1980, "Montaje, puesta en marcha y prueba de un torno a control numérico", Caracas.

Rosler, L, 1984, The UNIX System: The evolution of C - Past and Future", AT&T Bell Laboratories Technical Journal, Vol. 63, No. 8, Parte 2, Octubre de 1984, pp. 1685-1699.

.

Saks, D., 1993, "Inheritance, Part 2", The C Users Journal, Mayo, pp. 81-89.

Sandvik Coromant, 1999, “Guía de selección de herramientas”, SANDVIK, Suecia.

Shiffman, H., 1991, "C++ Object-Oriented Extensions to C", Sun World, Vol. 4, No. 5, Mayo, pp. 63-70.

SkeBy, C., 1993, "Pointer Power in C and C++. Part I", The C Users Journal, Vol. 11, No. 2, February, pp. 93-98. Snyder, A., 1993, The essence of objects: Concepts and Terms", IEEE Software Magazine. Vol. 10, No. 1, Enero, pp. 31-42.

Stroustrup. B., 1984, "The UNIX System: Data Abstraction In C". AT&T Bell Laboratories Technical Journal, Vol. 63, No. 8, Part 2, October 1984, pp. 1701-1732.

Vaffis, P. y J. S. Cotton, 1996, "A Method of Object-oriented Information Management for Layout Design", Engineering with Computers, Vol. 12, No. 1, pp. 34-45.

Voss, G., 1993, "Objects and Messages", Windows Tech Journal, February, pp. 15-16.

Wiebel, M; S. Halliday, 1992, "Using OOP Techniques Instead of

Switch in C++", The C Users Journal, October, pp. 105-106.

Wilde, N., P. Matthews; R. Huftt, 1993, 'Maintaining Object-Oriented Software", IEEE Software Magazine, Vol. 10, No 1, January, pp. 75-80.

WB, N., 1993, Templates in C++", The C Users Journal, Mayo, pp. 33-51