TRABAJO ESPECIAL DE GRADO

MÓDULOS DE AUTOMATIZACIÓN DE DIBUJOS DE VIGAS Y CONEXIONES EN ACERO ESTRUCTURAL, OPERANDO DENTRO DE UN SISTEMA CAD

TUTOR ACADÉMICO: Prof. Tomás Osers

Presentado ante la ilustre
Universidad Central de
Venezuela para optar al título
de Ingeniero Civil

Por el Br. José E. Herrera Trujillo

Caracas, mayo de 2002

TRABAJO ESPECIAL DE GRADO

MÓDULOS DE AUTOMATIZACIÓN DE DIBUJOS DE VIGAS Y CONEXIONES EN ACERO ESTRUCTURAL, OPERANDO DENTRO DE UN SISTEMA CAD

Presentado ante la ilustre
Universidad Central de
Venezuela para optar al título
de Ingeniero Civil

Por el Br. José E. Herrera Trujillo

Caracas, mayo de 2002

MÓDULOS DE AUTOMATIZACIÓN DE DIBUJOS DE VIGAS Y CONEXIONES EN ACERO ESTRUCTURAL, OPERANDO DENTRO DE SISTEMAS CAD

Elaborado por:

José E. Herrera Trujillo

Trabajo Especial de Grado presentado ante la Escuela de Ingeniería Civil de la Universidad Central de Venezuela en cumplimiento parcial de los requisitos exigidos para optar al título de Ingeniero Civil.

Aprobada su presentación

Ing. Tomás Osers

Caracas, mayo de 2002

VEREDICTO

Los suscritos, miembros del jurado para examinar el **Trabajo Especial de Grado** presentado por el bachiller:

José E. Herrera Trujillo

Para optar al **Título de Ingeniero Civil** en la **Universidad Central de Venezuela**, hacen constar que han examinado el mismo, otorgándole la calificación de:

Sin hacerse solidarios de las ideas ni conclusiones en él expuestas.

Caracas, de mayo de 2002

Observaciones:

Notas:

1. Calificación de 0 a 20 puntos.
2. El jurado podrá señalar los aspectos que considere de interés sobre el Trabajo presentado. Podrá hacer recomendaciones en cuanto a la conveniencia de continuar la investigación, la presentación del trabajo para optar a algún premio, la difusión del mismo, etc.. En caso de aplazamiento, el Jurado podrá razonar sus motivos indicando si el Trabajo es susceptible de mejorar o debe descartarse totalmente.

AGRADECIMIENTOS

A mi Familia, por su apoyo, confianza y amor.

A Yenny Báez, Aura Fernández y Leonardo Terreros por haberme ayudado en los momentos difíciles de la carrera.

A los integrantes del DIOC a quienes les agradezco su constante colaboración en la realización del programa.

A Tomás Osers, quien con su ejemplo y dedicación demostró que ser Profesor es mucho mas que impartir conocimientos en el aula.

.

ÍNDICE

pág.

SUMARIO

SUMARIO

El presente Trabajo Especial de Grado es una aplicación de "la computación gráfica" a la creación de planos de Taller de conexiones estructurales de miembros de acero.

El mismo comprende rutinas para el dibujo de conexiones estructurales de acero con la finalidad de lograr disminuir el tiempo de realización de los planos de taller.

Las rutinas antes mencionadas fueron implementadas en computadora IBM o compatibles, en el lenguaje "Visual Basic" bajo el ambiente de "AutoCAD".

Para el correcto uso de la aplicación, los planos dibujados con la ayuda de las rutinas de la barra de herramientas "Perfiles de Acero" deben ser representados en escala 1:1 en milímetros.

El programa realizado aumenta la eficiencia técnica y económica en el dibujo de conexiones estructurales de acero utilizando el Sistema AutoCAD. Además, constituye una herramienta muy importante para el Ingeniero Civil, ya que facilita el entendimiento de los planos de Taller a la vez que reduce el tiempo de elaboración de los mismos y agiliza cualquier cambio en el diseño.

A través del Sistema AutoCAD, se facilita la programación debido a que éste maneja instrucciones de graficación, con lo cual se permite que la elaboración del programa sea de una manera mas sencilla.

Es importante resaltar que al ver la forma tan rápida en que puede elaborarse un plano de Taller con la utilización de este programa, no podría percibirse el grado de dificultad y lo complejo del trabajo realizado para la elaboración del mismo, sobre todo por la diversidad de criterios de modelaje estructural de éstos y lo minucioso del detallado en el dibujo de los planos.

1.- INTRODUCCIÓN

1.- INTRODUCCIÓN

Este trabajo Especial de Grado utiliza un instrumento de alta tecnología como es la computación, para aumentar la eficiencia técnica (precisión) y económica (tiempo) en el dibujo de conexiones estructurales de acero para automatizar el dibujo de detalles constructivos de vigas y conexiones en acero estructural, bajo el ambiente del sistema de AutoCAD.

Haciendo historia; a lo largo de los años se ha podido constatar que el dibujo de conexiones de acero es una labor ardua y tediosa, en muchos casos los planos deben ser realizados en escala 1:1 lo que significa una gran pérdida de tiempo debido a la cantidad de detalles que deben incluirse en ellos. Hoy en día debido al avance tecnológico, y gracias a la existencia de las computadoras y especialmente con la creación de la computación gráfica se facilitan las labores antes mencionadas.

Actualmente resulta bastante común el uso de Sistemas Computarizados para la resolución y cálculo de los problemas frecuentes de la Ingeniería, sin embargo, el tiempo de culminación de estos proyectos que ha sido optimizado con el uso de estos sistemas, se ve retrasado debido al dibujo de los planos correspondientes, lo cual puede tomar un tiempo considerable; así como también cualquier cambio en el diseño, que implicaría repetir todo de nuevo. Con la ayuda de programas de este tipo resulta mas sencillo y a la vez se requiere de menor tiempo para elaborar los cambios en el diseño de las conexiones.

El programa desarrollado resultó de concatenar conocimientos de computación gráfica y de Ingeniería Civil, especialmente en el diseño de estructuras de acero. En los proyectos de estructuras de acero se distinguen fundamentalmente tres tipos de planos: planos de Proyecto, planos de Construcción y planos de Taller. Este programa facilita la elaboración de los planos de Taller, los cuales se preparan en base a la información suministrada por los planos de proyecto, y tienen como finalidad señalar los detalles necesarios para proceder a la fabricación de las partes de los componentes que integran la estructura. El objetivo fundamental de estos planos es mostrar la localización, el tipo y las

dimensiones de todos los conectores, indicando los que serán ejecutados en el Taller y en la Obra.

Se elaboraron un conjunto de subrutinas escritas en Visual Basic, las cuales fueron creadas con la misma lógica, de manera de programar las subrutinas comunes una sola vez, optimizando de esta forma el tiempo de programación.

Cada subrutina fue desarrollada con el mismo esquema, explicando su uso.

Por último cabe destacar que el objetivo específico del presente trabajo no es la utilización, manejo y aprendizaje de Visual Basic, sino el de facilitar la elaboración y el entendimiento de los planos de Taller.

2.- OBJETIVOS

2.2- OBJETIVOS GENERALES

El objetivo fundamental del presente Trabajo Especial de Grado, es desarrollar un conjunto de rutinas en lenguaje VBA (Visual Basic For Aplications) dentro del sistema AutoCAD (Computting Aided Design), para automatizar el dibujo de detalles constructivos de vigas y conexiones en acero estructural.

2.3- OBJETIVOS ESPECÍFICOS

- Estudiar y comprender el modelo de objetos del motor gráfico del sistema AutoCAD2000.
- Estudiar y comprender la base de datos y su manejo dentro del sistema AutoCAD2000.
- Estudiar y entender el manejo de VBA dentro del sistema AutoCAD2000.
- Desarrollar algoritmos de automatización de dibujos en CAD, de diversos detalles constructivos de vigas y conexiones en Acero Estructural.
- Implementar los algoritmos desarrollados en el objetivo inmediato anterior, en el lenguaje VBA para operar dentro del sistema AutoCAD2000.
- Crear diálogos de comunicación entre el usuario y el CAD.

3.- MARCO TEÓRICO

3.2- PROGRAMACIÓN CON VISUAL BASIC

3.2.2- VISUAL BASIC Y ACTIVEX AUTOMATION

Visual Basic es la desembocadura de lo que en un principio se llamó lenguaje BASIC (*Beginner's All-purpose Symbolic Instruction Code*), que fue diseñado para iniciados en Estados Unidos en 1964. Fue una derivación del lenguaje FORTRAN que se caracterizaba por su sencillez de manejo y aprendizaje(1).

Microsoft a lo largo de su historia ha venido proporcionando una serie de editores BASIC cuyos más altos exponentes fueron GWBASIC y QuickBASIC. Cuando apareció Windows como entorno las cosas comenzaron a cambiar: aparecen los lenguajes visuales. La programación visual, conocida como orientada a objetos y conducida por eventos, permite desarrollar aplicaciones para Windows de una manera sencilla e intuitiva (2).

Hoy en día, esta programación orientada a objetos se sirve de una de las últimas tecnologías de programación, esto es *ActiveX Automation*. La tecnología ActiveX, presente en las últimas versiones de Windows, supera con creces al sistema anterior OLE, proporcionando una interfaz de programación que permite manejar los objetos de una aplicación desde fuera de la misma. La intercomunicación ahora entre aplicaciones que manejen objetos ActiveX es total: los usuarios serán capaces desde un programa en Visual Basic de abrir una sesión de **AutoCAD**, añadir una serie de líneas a modo de tabla y, a continuación abrir una sesión en Excel, por ejemplo, para leer determinados datos, operar con ellos y volver a **AutoCAD** para añadir los datos a la tabla. Y todo esto mediante letreros de diálogo, botones y demás componentes típicos y sencillos de manejar de Microsoft Windows.

Los objetos propios de una aplicación son así expuestos a las demás aplicaciones como objetos Automation. En **AutoCAD** son objetos Automation todos los objetos de dibujo, las denominadas tablas de símbolos como bloques, capas, estilos de cota, etc. y también las *Preferencias*. Las aplicaciones que proporcionan un entorno de programación en el que los desarrolladores o usuarios pueden

escribir macros y rutinas para acceder y controlar objetos ActiveX se denominan *controladores de Automation*. Estos controladores pueden ser aplicaciones de Windows como Word y Excel, o entornos de programación como Visual Basic o Visual C++.

Los objetos ActiveX exponen para su control dos conceptos específicos: métodos y propiedades. Los métodos son funciones que ejercen una acción sobre los objetos. Las propiedades son funciones que definen o devuelven información sobre el estado de un objeto. Las propiedades y métodos dependen de cada tipo de objeto y se describen a través de una biblioteca de tipos. Los desarrolladores o usuarios tienen a su disposición un examinador de biblioteca de tipos, para saber en todo momento los métodos y propiedades existentes. **AutoCAD** dispone de sus propios objetos ActiveX para ser manejados desde un programa (1,3).

La biblioteca de tipos donde se encuentran definidos los métodos y propiedades a través de los cuales expone **AutoCAD** sus objetos *Automation* se encuentra en el archivo ACAD.TLB. Las referencias se establecen desde el editor de VBA, desde el menú *Herramientas>Referencias...*, activando la casilla *AutoCAD Object Library*. Los objetos de una aplicación pueden usarse sin hacer referencia a la biblioteca de objetos, pero es preferible añadir dicha referencia por motivos de facilidad, fiabilidad y comprobación.

Es posible acceder a objetos Automation desde muchas aplicaciones de Windows. Estas aplicaciones pueden ser ejecutables independientes, archivos de bibliotecas de enlace dinámico .DLL y macros dentro de aplicaciones como Word y Excel. Mediante Automation resulta muy sencillo desarrollar y mantener una interfaz gráfica de usuario. La interfaz puede ir desde un solo cuadro de diálogo o una barra de herramientas que mejore una aplicación existente, hasta una aplicación propia completa.

3.2.3- PROCESO GENERAL DE CREACIÓN DE UN PROGRAMA

El proceso de creación de un programa se puede dividir en tres etapas características (3):

1. **Generación del código fuente.** Comprende el diseño del formulario con todos sus controles, y la generación de los procedimientos que se van a ejecutar al actuar el usuario sobre el formulario (ver Figura # 1). El módulo VBA proporciona:

   a. Objetos de formularios tomados de Visual Basic, pero adaptados a las necesidades del AutoCAD.

   b. Herramientas de formato para los objetos de control: alineación, centrado, tamaño, orden de superposición, etc.

   c. Examinador de objetos para acceder a todos los tipos de objetos existentes, así como sus métodos y propiedades.

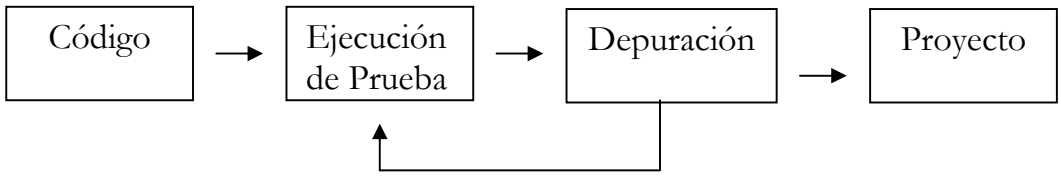


Figura # 1. Generación del Código fuente

2. **Ensayo y depuración de programas.** Una vez generado el código fuente, se trata de ejecutar el programa, ya sea total o parcialmente, para someterlo a todas las contingencias posibles y poner en evidencia los fallos que pudiera tener. El módulo VBA proporciona:

    a. Posibilidad de detener la ejecución del programa en puntos estratégicos mediante puntos de ruptura o interrupción.

    b. Simulaciones paso a paso de ejecución del programa, pudiéndose analizar simultáneamente las líneas del código fuente y su efecto en AutoCAD.

    c. Inspección de valores de variables para examinar el funcionamiento del programa durante su ejecución.

3. **Utilización de programas.** Una vez completado y depurado el programa, el paso final consiste en obtener un proyecto ejecutable para que se pueda utilizar desde AutoCAD. El módulo VBA proporciona:

    a. Creación de archivos de proyecto directamente ejecutables desde AutoCAD para aumentar la eficiencia y seguridad de utilización de los programas.

    b. Posibilidad de importar archivos de formularios, código Basic o clase para utilizar dentro del proyecto de VBA.

3.2.4- INICIACIÓN A VBA

Una aplicación o programa de VBA está formado por módulos. Una aplicación debe tener por lo menos un módulo, que son los que contiene el código de la aplicación. Este código puede estar formado por declaraciones de variables u objetos, procedimientos y eventos, aunque estos últimos solo en el módulo de Formulario o en ThisDrawing. El objeto ThisDrawing es el documento de AutoCAD asociado con el proyecto, que puede llevar el código como cualquier otro módulo, declaraciones, procedimientos y eventos.

3.2.5- OBJETOS (3)

Los objetos son las entidades con las que se trabaja en VBA. Un objeto ha sido definido previamente en un módulo de clase. El lenguaje de Visual Basic tiene ya definido un conjunto de objetos para trabajar con ellos como son los formularios y los controles que sobre él aparecen. Algunos de estos controles son cuadros de texto, botones de comando, etiquetas, etc.

3.3- PROGRAMACIÓN ORIENTADA A OBJETOS

Un programa tradicional consiste en un conjunto de instrucciones que se van ejecutando en un determinado orden. La actuación del usuario se limita a introducir datos cuando el programa se los pide, y éste se encarga de procesarlos leyendo las instrucciones una a una, en función de las estructuras de control (alternativas, repetitivas y llamadas a subrutinas) diseñadas por el programador.

Un programa orientado a objetos asocia bloques de instrucciones a cada objeto. En el momento de ejecutarse, el usuario puede actuar sobre cada objeto de varias maneras: haciendo *clic*, doble *clic*, pulsar y arrastrar, etc. Según el objeto seleccionado y la manera de actuar sobre él, se ejecutará el bloque de instrucciones asociado. Estas posibilidades de actuación sobre los objetos es lo que se conoce como sucesos o eventos. De ahí viene la denominación de programación orientada a objetos y conducida por eventos (4).

Así pues, un programa de este tipo contiene una serie de bloques de instrucciones que se llaman procedimientos, y que se pueden agrupar en módulos. La estructura completa se representa en la Figura # 2 y los conceptos se resumen en los siguientes términos:

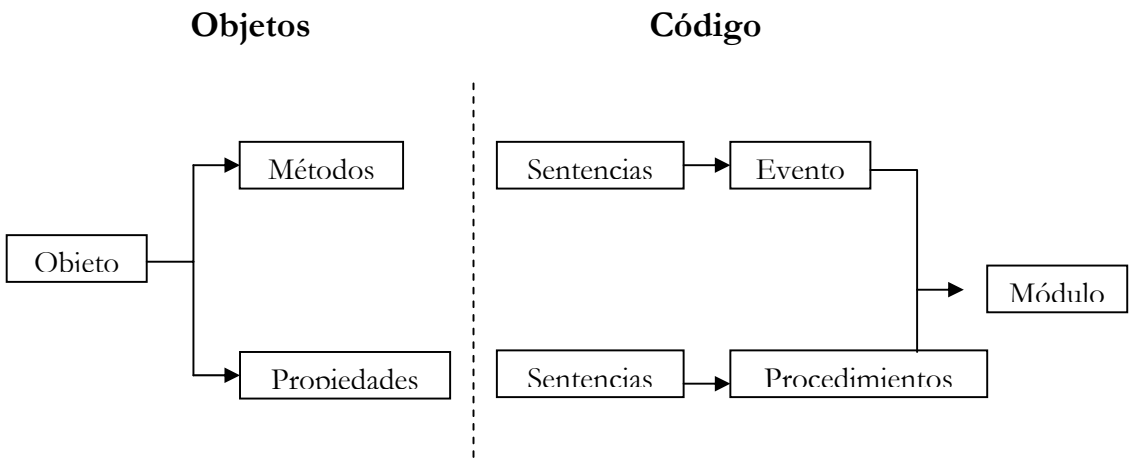


Figura # 2. Estructura general de un programa orientado a un objeto (5)

- **Aplicación o Programa:** Conjunto de bloques de instrucciones y objetos de control cuya función consiste en desarrollar un objetivo común. Se puede dividir en varios módulos.

- **Módulo:** Subconjunto de un programa. Hay tres módulos posibles:

    - **Formulario:** Contiene declaraciones de variables, procedimientos, eventos y la ventana de formulario.
    - **Estándar:** Contiene declaraciones de variables y procedimientos.
    - **Clase:** Contiene definiciones de nuevos objetos, con sus métodos y propiedades.

- **Ventana de Formulario:** Ventana en pantalla donde se sitúan los objetos de control sobre los que va a actuar el usuario. Es el mismo concepto que el cuadro de diálogo de AutoCAD.

- **Procedimiento:** Cada uno de los bloques de instrucciones o sentencias que forman el código. Pueden ser subrutinas (tipo *Sub*) o funciones (tipo *Function*).

- **Sentencia:** Cada instrucción de código dentro de un procedimiento.

- **Objeto:** Cada uno de los elementos sobre los que actúa un programa. Se agrupan por clases y se pueden destacar tres:

    - **Objetos del Sistema:** Contiene objetos especiales del sistema operativo: *Err*, *Font*, *Printer*, *Debug*, etc.
    - **Objetos de Formulario:** Contiene el propio formulario y todos los objetos de control que se pueden situar en él: casillas, botones, listas, deslizadores, etc.
    - **Objetos de AutoCAD.** Contiene todos los objetos del dibujo, tablas de símbolos, objetos no gráficos y preferencias. Se distribuyen en una estructura jerarquizada.

- **Evento:** Código que se ejecuta cada vez que se ejerce una actuación sobre un objeto: *clic*, doble *clic*, pulsar y arrastrar, cargar, modificar, etc.

- **Propiedad:** Cada uno de los atributos de un objeto. Así, un objeto de formulario tiene las propiedades de *Name*, *Caption*, *backColor*, etc. Un objeto cualquiera de dibujo de AutoCAD tiene las propiedades de *Color*, *Layer*, *Linetype*, etc. Un objeto específico de dibujo, en el caso de un círculo por ejemplo, tendrá las propiedades *Center*, *Radius*, *Area*, etc.

- **Método:** Instrucción para actuar sobre los objetos. Así, por ejemplo, *Add*, *Close* o *GetFormat* para los objetos de Formulario, y *AddCircle*, *Move*, *Copy* o *GetAngle* para los objetos de AutoCAD.

3.3.2- LA PLANTILLA DE OBJETOS

Los objetos ActiveX que proporciona **AutoCAD** para su manejo desde programas VBA están divididos según una jerarquía que se debe seguir a la hora de llamarlos o referirse a ellos. La plantilla que se muestra en la Figura # 3 es muy útil a la hora de programar, ya que establece dicha jerarquía.

En Visual Basic es factible añadir al entorno nuevos objetos creados por el programador para luego ser utilizados. Lo que se ha hecho en VBA es precisamente eso. Estos objetos tienen sus propiedades y métodos, al igual que los demás. Existen objetos de entidades individuales de dibujo (líneas, círculos, arcos...) con sus propiedades (color, capa, tipo de línea...) y métodos (copiar, mover, escalar).

También se han definido otros objetos no gráficos como son el Espacio Modelo, el Espacio Papel y los bloques. Éstos se consideran una colección de objetos de entidades individuales de dibujo y tienen también sus propiedades para, por ejemplo, saber cuántas entidades simples contienen, y sus métodos para, por ejemplo, añadir nuevas entidades a la colección.

El propio documento actual de **AutoCAD** está definido como un objeto y tiene sus propiedades (camino de acceso, límites...) y métodos (guardar, regenerar...). Dentro de él se encuentran los mencionados anteriormente, además de otras colecciones como el conjunto de capas, de estilos de texto, etcétera, cada una con propiedades y métodos.

Y todo ello está incluido en el objeto más exterior, que es la aplicación de **AutoCAD**.

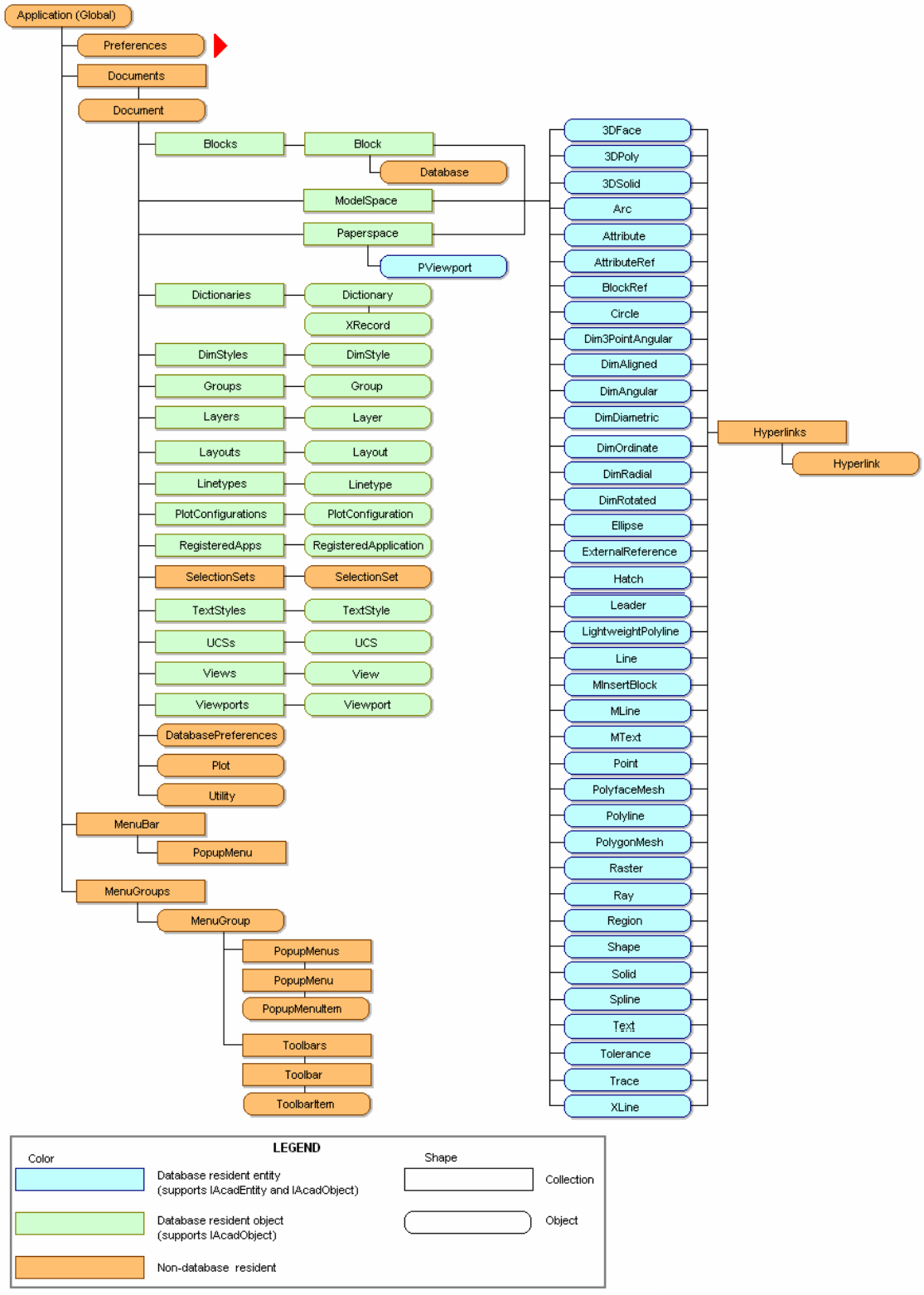


Figura # 3. Jerarquía de los Objetos (5)

3.3.3- EMPEZAR UN PROGRAMA

El primer paso que se debe dar al realizar un programa en VBA es casi siempre el mismo, y se refiere a la declaración de las variables de objeto que serán necesarias para acceder a los distintos aspectos de **AutoCAD**.

Según plantilla mostrada en la Figura # 3, la propia aplicación **AutoCAD**, el documento activo, el Espacio Modelo o el Espacio Papel, entre otros, son objetos que se deben de definir en un principio para luego poder referirse a ellos a lo largo del programa fácilmente. Como objetos que son se declararán como tales, así por ejemplo, la aplicación en sí se podría declarar así (en `General_Declaraciones`) (6):

```
AcadApp as Object
```

el documento activo:

```
AcadDoc as Object
```

y el Espacio Modelo y Papel:

```
AcadModel as Object
```

y

```
AcadPapel as Object
```

Esto en sí no tiene mucho sentido hasta que no se le den unos valores coherentes. Para ello se utilizará el procedimiento `UserForm_Initialize` (en el caso de que el programa disponga de un formulario; no es una macro), ya que habrán de tomar valores al iniciar el programa, así:

```
Set AcadApp = GetObject(, "AutoCAD.Application")
Set AcadDoc = AcadApp.ActiveDocument
Set AcadModel = AcadDoc.ModelSpace
Set AcadPapel = AcadDoc.PaperSpace
```

Para añadir valores a variables del tipo `Object`, es necesario utilizar la instrucción `Set`.

La primera línea es la que realmente hace referencia a la librería o biblioteca de objetos de **AutoCAD**. No se incluye el camino por encontrarse el archivo `ACAD.TLB` en el directorio principal de instalación de **AutoCAD**, esto es, por estar en directorio de archivos de soporte. Sin embargo, la coma necesaria de la función `GetObject` que separa los dos parámetros es imprescindible incluirla.

Las líneas restantes hacen referencia a los demás objetos que, como se ve, cuelgan todos del primero. El documento activo (`ActiveDocument`) cuelga directamente de la aplicación **AutoCAD**, y tanto el Espacio Modelo (`ModelSpace`) como el Espacio Papel (`PaperSpace`) del documento actual. Esta jerarquía se puede apreciar perfectamente en la plantilla mostrada en la Figura # 3.

Esta asignación de objetos no es de obligatoria realización, ya que se puede referir a ellos con toda la secuencia de nombres. Pero parece lógico utilizar, por ejemplo, un nombre de variable como `AcadModel` cada vez que se tenga que referir al Espacio Papel (que serán muchas veces), que utilizar la secuencia entera hasta `ModelSpace(7)`.

Además en el caso de trabajar directamente en VBA se puede sustituir las llamadas al documento actual por `ThisDrawing` así:

```
Set AcadDoc = ThisDrawing
```

o incluso utilizar `ThisDrawing` posteriormente en las llamadas durante el programa. El problema de esto reside en que sólo el VBA de **AutoCAD** sabe lo que es `ThisDrawing`. En el momento en que se desee abrir un programa en un entorno Visual Basic externo para, por ejemplo compilarlo, se necesitará realizar las llamadas pertinentes a la librería de objetos, si no nunca funcionará. Es por ello que es mejor que el programador se acostumbre a esta técnica mucho más versátil y elegante (8).

Todo esto es necesario porque se le debe decir al VBA que se va a trabajar con una aplicación denominada **AutoCAD**, que utilizará el documento actual activo y, por ejemplo, su Espacio Modelo. A la hora de añadir un círculo, por ejemplo,

se deberá indicar que ha de ser al Espacio Modelo del documento actual de **AutoCAD**, sobre todo si se trabaja con un Visual Basic externo o el archivo está ya compilado.

Se emplea en estas declaraciones la función `GetObject`, cuya finalidad consiste en devolver una referencia al objeto que se le solicita, en el caso a estudiar a `AutoCAD.Application`, que engloba a todo el modelo de objetos. Para que esta función responda adecuadamente es necesario que **AutoCAD** esté cargado y con un dibujo abierto, que será referenciado como `ActiveDocument(9)`.

3.4- DIBUJO Y REPRESENTACIÓN DE ENTIDADES

A continuación, se explican los diferentes métodos que existen para añadir entidades individuales de dibujo mediante programas en VBA, se explicarán los métodos de las entidades mas utilizadas en la realización de "Perfiles de Acero", tal y como son las líneas, los círculos y los textos.

La manera de dibujar entidades tiene relación con métodos que pertenecen a las colecciones de Espacio Modelo, Espacio Papel y Bloques (como se ve en la plantilla mostrada en la Figura # 3).

3.4.2- LÍNEAS (10)

La sintaxis que se utiliza a continuación es: instrucciones, funciones, métodos, propiedades y demás términos reservados como aparecen en el editor VBA una vez aceptados; los textos en cursiva son mnemotécnicos que han de ser sustituidos por su valor; los textos no en cursiva deben escribirse como tales; las sintaxis en colores: métodos en azul y propiedades en verde; en las listas las propiedades y métodos de objetos nuevos (sin explicar) en negrilla, los ya explicados no; los listados de programas se muestran como aparecen en el editor; una barra vertical indica una dualidad de valores.

La sintaxis del método `AddLine` para dibujar líneas es la que sigue:

```
Set ObjLínea = ObjColección.AddLine(DblPtoInicial,
DblPtoFinal)
```

Propiedades	Métodos
Application	ArrayPolar
Color	ArrayRectangular
EndPoint	Copy
EntityName	Erase
EntityType	GetBoundingBox
Handle	GetXData
Layer	Highlight
Linetype	IntersectWith
LinetypeScale	Mirror
Normal	Mirror3D
ObjectID	Move
StartPoint	Offset
Thickness	Rotate
Visible	Rotate3D
	ScaleEntity
	SetXData
	TransformBy
	Update

Tabla # 1. Propiedades y Métodos

Los objetos gráficos de dibujo se han de declarar previamente como tales. Para ello se debe definir un nombre de variable que almacenará el objeto; es a lo que se refiere *ObjLínea*. Esta variable puede ser declarada como `Object` simplemente o como un objeto especial de VBA para **AutoCAD** que representa el tipo de objeto que almacenará. Este objeto especial tiene diferentes nombres según el objeto que almacene; su sintaxis podría definirse así: `IAcad`*`Objeto`*, es decir, primero la cadena fija `IAcad` y luego el nombre del objeto.

De esta manera, una variable que tuviera que guardar una línea podría declararse como `Object` o como `IAcadLine`. La diferencia es que si se declara como `Object` podrá almacenar cualquier objeto a lo largo del programa: una línea, un círculo, un rayo, un texto. Si se declara como `IAcadLine` esa variable única y exclusivamente podrá almacenar entidades de línea de **AutoCAD**, una o varias a lo largo de la ejecución, pero sólo líneas. Es recomendable hacerlo con la primera opción (`Object`), sino a veces hay problemas.

Las entidades de **AutoCAD**, para ser dibujadas desde VBA, han de ser guardadas en una variable de objeto, por lo tanto con `Set`.

`ObjColección` se refiere a la colección donde se almacenará la línea, es decir, si se dibujará en Espacio Modelo o Espacio Papel, o si formará parte de un bloque. Los puntos inicial y final de la línea (*`DblPtoInicial`* y *`DblPtoFinal`*) deben ser agrupaciones de tres coordenadas (X, Y, y Z), por lo tanto han de definirse como matrices (*arrays* o tablas) de tres elementos, cada uno de ellos de tipo `Double`, generalmente.

Una macro en VBA se crea y se añade automáticamente al módulo estándar de código existente si existe, si no existe se crea una nuevo. La macro se ejecutará sin necesidad de letrero de diálogo.

Para crear una macro VBA se hace desde el menú *Herramientas>Macros.*. Esto abre un cuadro que permite introducir un nombre para la nueva macro (también se puede eliminar, editar o ejecutar una existente). Tras pulsar el botón *Crear* se añadirá un nuevo procedimiento `Sub` al módulo de código existente (o se creará uno). Dicho procedimiento tendrá el nombre de la macro. Para ejecutar una macro se puede hacer desde el mismo cuadro de creación (eligiendo una) o desde el menú de **AutoCAD** *VBA>Run Macro...*, esto último arranca un cuadro de diálogo que permite elegir el módulo estándar en cuestión donde se encuentra la macro y la macro por su nombre (en el caso de que hubiera varias dentro de un mismo módulo).

A continuación se creará un ejemplo de una macro llamada `DibujoLínea` y se le añadirán las siguientes líneas de código:

```
Option Explicit
Dim AcadDoc As Object
Dim AcadModel As Object
Dim ObjLínea As IAcadLine
Dim PuntoInicial (1 To 3) As Double
Dim PuntoFinal (1 To 3) As Double
```

```
Sub DibujoLínea()
  Set AcadDoc = GetObject( ,"Autocad.Application").ActiveDocument
  Set AcadModel = AcadDoc.ModelSpace
  PuntoInicial(1) = 100: PuntoInicial(2) = 100: PuntoInicial(3) = 0
  PuntoFinal(1) = 200: PuntoFinal(2) = 200: PuntoFinal(3) = 0
  Set ObjLínea = AcadModel.AddLine(PuntoInicial, PuntoFinal)
End Sub
```

Se recomienda el uso de `Option Explicit` para depurar errores en tiempo de corrida.

Lo primero que se hace en `General_Declaraciones`, tras `Option Explicit` que obliga a declarar todas las variables utilizadas, es definir o declarar las variables que se van a usar. La que representa al documento actual y la del Espacio Modelo; la que contendrá la línea, como un objeto de línea de **AutoCAD**; y las del punto inicial y final de la línea, como matrices del tipo `Double`.

En el cuerpo de la macro propiamente dicha se asigna su valor a cada variable, tanto a las de la aplicación y el dibujo actual, como a los puntos inicial y final. Se utiliza el método explicado para dibujar la línea en el Espacio Modelo.

Existen los correspondientes objetos `IAcadModelSpace` e `IAcadDocument`, pero se recomienda la sintaxis utilizada en el ejemplo para estos objetos.

La ventaja que lleva implícita el almacenamiento en una variable de objeto de la entidad dibujada, dice relación a su posterior utilización para la aplicación de propiedades. No es necesario acceder a la Base de Datos interna de **AutoCAD** para modificar las propiedades de un objeto, ya que dichas propiedades se relacionan directamente con sus objetos. Así por ejemplo, si tras trazar la línea del ejemplo anterior se quisiera cambiar al color rojo, únicamente se deberá añadir la siguiente línea al programa (tras *`Set ObjLínea...`*):

```
ObjLínea.Color = 1
```

`Color` es una propiedad de la línea (véase en la lista tras la sintaxis), por lo que lo único que se hace es cambiarle el valor como a cualquier otra propiedad en Visual Basic: indicando el objeto, seguido de un punto de separación, la propiedad, un signo de igual y el nuevo valor. Algunas propiedades se tratan de otra forma.

A continuación se explican a fondo cada una de las propiedades de los objetos de línea y se explican de manera ambigua, es decir, sin referirse a las líneas en sí, ya que estas propiedades son comunes a muchos otros objetos VBA.

- `Application`. Obtiene el objeto `Application` de la entidad. Este objeto representa la aplicación de **AutoCAD**. La sintaxis es:

```
Set ObjAplicación = ObjGráfico.Application
```

siendo *`ObjAplicación`* una variable definida como `Object` y *`ObjGráfico`* un objeto gráfico. En el caso del ejemplo anterior de dibujo de una línea, *`ObjGráfico`* sería la variable `ObjLínea`.

- `Color`. Obtiene y/o asigna el color de/a una entidad. El color se representa como un número entero de `0` a `256` (colores de **AutoCAD**). También se pueden emplear algunas constantes predefinidas como:

```
AcByBlock
AcByLayer
AcCyan
AcRed
AcBlue
AcYellow
AcMagenta
AcGreen
AcWhite
```

La sintaxis para esta propiedad es:

```
ObjGráfico.Color = IntNumColor
```

Siendo *IntNumColor* el número de color o alguna de las constantes. Ésta es la forma en que se ha cambiado el color a la línea en el ejemplo. Existe otra sintaxis para obtener el color de un objeto dibujado y guardado en una variable:

```
IntNumColor = ObjGráfico.Color
```

donde *IntNumColor* es ahora la variable (tipo `Integer`) que guardará el número de color y `ObjGráfico` la variable (tipo `Object`) que almacena la línea. En el ejemplo anterior la línea estaba almacenada en `ObjLínea` (tipo `IAcadLine`). Si se quisiera obtener su color y guardarlo en una variable denominada `NúmeroColor` se tendrá que hacer, por ejemplo:

```
NúmeroColor = ObjLine.Color
```

Antes de esto, y si existe una instrucción `Option Explicit`, se habrá de declarar la variable `NúmeroColor` como `Integer`.

- `EndPoint`. Obtiene el punto final de un objeto arco, elipse o línea y, en el caso de las líneas, también puede asignarse. Así pues, la sintaxis para obtener el punto final de los objetos mencionados es:

```
VarPtoFinal = ObjGráfico.EndPoint
```

siendo *VarPtoFinal* una variable que almacena un punto. Estas variables de punto siempre han de definirse como `Variant`. En el ejemplo se podría haber obtenido el punto final de la línea así, aunque esto no tiene mucho sentido (como con el color) porque ya se sabe cual es su punto final; se lo ha dado el programador.

Si lo que se desea es asignar un punto final (sólo para líneas) se utilizará la siguiente sintaxis:

```
ObjLínea.EndPoint = DblPtoFinal
```

donde *ObjLínea* es una variable tipo objeto que contiene una línea y *DblPtoFinal* es una variable que contiene un punto, es decir una matriz de tres valores tipo `Double`.

• `EntityName`. Obtiene el nombre de un objeto. Este nombre es el de la clase a la que pertenece el objeto. La sintaxis de obtención de un nombre es:

```
StrNombre = ObjGráfico.EntityName
```

donde *`ObjNombre`* es una variable de tipo `String` que guardará el nombre del objeto, y `ObjGráfico` es la variable que guarda el objeto. Si en el ejemplo anterior, se define una variable llamada por ejemplo `NombreLínea` como `String`, y se escribe al final:

```
NombreLínea = ObjLínea.EntityName
```

`NombreLínea` guardaría la cadena `AcDbLine`, nombre de clase del objeto de entidad línea.

Cuando se pregunta por objetos de **AutoCAD** se puede emplear la siguiente propiedad `EntityType`.

• `EntityType`. Obtiene el tipo de entidad. Éste ha de ser guardado en una variable de tipo `Integer` así:

```
IntNumTipo = ObjGráfico.EntityType
```

El número de tipo devuelto se corresponde con los expuestos en la Tabla # 2.

Tipo de entidad	Descripción
1	AcDB3DFace
2	AcDB3DPolyLine
3	AcDB3DSolid
4	AcDBArc
5	AcDBAttribute
6	AcDBAttributeReference
7	AcDBBlockReference
8	AcDBCircle

Tipo de entidad	Descripción
9	AcDBDimAligned
10	AcDBDimAngular
11	AcDBDimDiametric
12	AcDBDimOrdinate
13	AcDBDimRadial
14	AcDBDimRotated
15	AcDBEllipse
16	AcDBGroup
17	AcDBHatch
18	AcDBLeader
19	AcDBLine
20	AcDBMText
21	AcDBPoint
22	AcDBPolyline
23	AcDBPolylineLight
24	AcDBPolyMesh
25	AcDBPViewPort
26	AcDBRaster
27	AcDBRay
28	AcDBRegion
29	AcDBShape
30	AcDBSolid
31	AcDBSpline
32	AcDBText
33	AcDBTolerance
34	AcDBTrace
35	AcDBXLine

Tabla # 2 . Tipos de Entidad

En el ejemplo, una sentencia de este tipo devolvería en código `19`.

• `Handle`. Obtiene el rótulo o código del objeto. Se trata de una cadena de texto que identifica el objeto durante toda la vida de ese objeto en el dibujo. La sintaxis pues de esta propiedad es:

```
StrRótulo = ObjGráfico.Handle
```

`StrRotulo` es una cadena (`String`).

• `Layer`. Permite obtener y/o asignar la capa de/a un objeto. La sintaxis para asignar una capa a un objeto ya dibujado es:

```
ObjGráfico.Layer = StrNombreCapa
```

donde *`ObjGráfico`* es la variable que guarda el objeto y *`StrNomCapa`* una cadena de texto con el nombre de una capa existente. Si la capa no existe VBA proporciona un error de ejecución. Para evitar eso habría que crearla previamente. En el ejemplo de la línea, si existiera una capa llamada `CUERPO` en el dibujo actual y hubiese querido colocar o introducir dicha línea en esa capa, se habría hecho:

```
ObjLínea.Layer = "Cuerpo"
```

La sintaxis para obtener la capa de un objeto es:

```
StrNomCapa = ObjGráfico.Layer
```

`StrNomCapa` habrá de ser una variable del tipo `String`, es decir una cadena alfanumérica.

• `Linetype`. Permite obtener y/o asignar un tipo de línea de/a una entidad de dibujo. Para asignar un tipo de línea:

```
ObjGráfico.Linetype = StrNombreTipoLínea
```

siendo *ObjGráfico* el objeto en cuestión (como la línea en `ObjLínea`) y *StrNomTipoLínea* una cadena que especifica el nombre del tipo de línea. Si éste no está cargado se producirá un error. Como tipos de línea especiales se pueden indicar:

```
ByLayer
ByBlock
Continuous
```

Para obtener el tipo de línea de un objeto:

```
StrNomTipoLínea = ObjGráfico.Linetype
```

StrNomTipoLínea será una cadena (`String`).

• `LinetypeScale`. Obtiene y asigna la escala de tipo de línea de/a un objeto. Para asignar:

```
ObjGráfico.LinetypeScale = RealEscalaTipoLínea
```

RealEscalaTipoLínea es el valor de la nueva escala.

Para obtener:

```
RealEscalaTipoLínea = ObjGráfico.LinetypeScale
```

RealEscalaTipoLínea es una variable positiva y real del tipo `Double`.

• `Normal`. Obtiene y asigna la normal de/a una entidad. La normal es un vector unitario que indica la dirección Z en el SCE (Sistema de Coordenadas de la Entidad). El vector se almacena en una variable del tipo `Variant`, que recogerá una matriz de tres elementos `Double`. Para asignar una normal:

```
ObjGráfico.Normal = DblVector
```

Para obtener una normal:

```
VarVector = ObjGráfico.Normal
```

• `ObjectID`. Extrae el identificador interno de una entidad. Este número es devuelto en notación decimal. Para obtenerlo:

```
LngNúmeroID = ObjGráfico.ObjectID
```

Donde *`LngNúmeroID`* es una variable que habrá sido declarada como `Long`.

• `StartPoint`. Funciona igual que `EndPoint` pero para el punto inicial de líneas, arcos y elipses. En estos dos últimos casos sólo se puede obtener su valor; con las líneas se puede modificar. Para asignar un nuevo punto inicial a una línea la sintaxis es:

```
ObjLínea.StartPoint = DblPtoInic
```

donde *`ObjLínea`* sólo puede ser un objeto de línea exclusivamente y *`DblPtoInic`* una variable que almacene un punto o una matriz de tres valores tipo `Double`.

Para obtener el punto inicial de los objetos mencionados:

```
VarPtoInic = ObjLínea.StartPoint
```

`VarPtoInic` habrá sido declarada como `Variant`.

• `Thickness`. Obtiene y/o asigna un valor que representa la altura en Z de un objeto 2D. Para asignar, la sintaxis es la siguiente:

```
ObjGráfico.Thickness = DblAlturaObjeto
```

y para obtener la elevación de un objeto:

```
DblAlturaObjeto = ObjGráfico.Thickness
```

`DblAlturaObjeto` habrá sido declarada como `Double`.

• `Visible`. Obtiene y/o asigna la visibilidad de/a una entidad. Esta propiedad es del tipo `Boolean`, por lo que sus resultados sólo podrán ser `True` o `False`. De querer asignar un estado de visibilidad a un objeto se utilizara la siguiente sintaxis:

```
ObjGráfico.Visible = BooEstadoVis
```

Por ejemplo, en el caso de la macro que dibujaba una línea, si al final del cuerpo de la macro (antes de `End Sub`) se escribiera:

```
ObjLínea.Visible = False
```

y se ejecutara, la línea se habría dibujado, pero permanecería invisible. A veces puede ser interesante mantener objetos invisibles para mostrarlos en un momento dado.

De querer extraer el estado de visibilidad de un objeto se utilizará la sintaxis que sigue:

```
BooEstadoVis = ObjGráfico.Visible
```

donde *`BooEstadoVis`* habrá sido declarado, como norma general, como `Boolean`.

A parte de las propiedades, los nuevos objetos VBA para **AutoCAD** también poseen métodos. A continuación se explicarán todos los métodos existentes para las líneas.

Al igual que para las propiedades, los métodos de los objetos son muchos de ellos comunes, por lo que se refiere a ellos como globales, no como particulares de la creación de líneas.

• `ArrayPolar`. Este método crea una matriz polar del objeto indicado, especificando un centro, los grados abarcados y el número de objetos. La sintaxis de este método es la siguiente:

```
VarMatriz = ObjGráfico.ArrayPolar(IntNúmero, DblÁngulo, DblCentro)
```

donde *VarMatriz* ha de ser una variable declarada como `Variant` que guardará todos los objetos de la matriz o *array*. `ObjGráfico` es el objeto ya creado del que se realizará la matriz polar, *IntNúmero* es el número de objetos de la matriz (`Integer`), *DblÁngulo* los ángulos cubiertos en radianes (`Double`) y *DblCentro* el centro de la matriz (un punto de tres elementos `Double`).

Así, y retomando el ejemplo que se ha presentado en el dibujo de una línea, se ampliará a continuación:

```
Option Explicit

Dim AcadDoc As Object
Dim AcadModel As Object
Dim ObjLínea As Object
Dim PuntoInicial (1 To 3) As Double
Dim PuntoFinal (1 To 3) As Double
Dim MatrizLin As Variant
Dim PuntoBase (1 To 3) As Double
```

```
Sub DibujoLínea()
  Set AcadDoc = GetObject( ,"Autocad.Application").ActiveDocument
  Set AcadModel = AcadDoc.ModelSpace
  PuntoInicial(1) = 100: PuntoInicial(2) = 100: PuntoInicial(3) = 0
  PuntoFinal(1) = 200: PuntoFinal(2) = 200: PuntoFinal(3) = 0
  Set ObjLínea = AcadModel.AddLine(PuntoInicial, PuntoFinal)
  PuntoBase(1) = 10: PuntoBase(2) = 10: PuntoBase(3) = 0
  MatrizLin = ObjLínea.ArrayPolar(10, 3.14159, PuntoBase)
End Sub
```

En este nuevo ejemplo, a parte de lo anterior se declara la variable `MatrizLin` como `Variant` y `PuntoBase` como una tabla de tres valores `Double`. Después se realiza la matriz (tras dar valores a `PuntoBase`). El número de elementos y los

ángulos cubiertos pueden ser también variables, evidentemente declaradas como `Integer` y `Double` respectivamente.

Se puede acceder a la tabla que guarda los objetos mediante índices para obtener los propios objetos de la matriz. En el ejemplo anterior, únicamente se deben añadir las siguientes dos líneas al listado (antes de `End Sub`):

```
Dim Línea6 As Object
Set Línea6 = MatrizLin(6)
```

para guardar en la variable `Línea6` la sexta de las líneas de la matriz en este caso. Esta variable tiene que ser declarada como `Object` porque va a guardar un objeto. Además el índice habrá de ser coherente con el número de objetos claro está, sino se produce un error de ejecución en el módulo VBA.

- `ArrayRectangular`. Este método crea una matriz polar de dos o tres dimensiones repitiendo el objeto al que se le aplica. Hay que indicar el número de filas, columnas y niveles, así como las distancias correspondientes, de la siguiente manera:

```
VarMatriz = ObjGráfico.ArrayRectangular(IntNúmFil,
IntNumCol, IntNumNiv, DblDistFil, DblDistCol,
DblDistNiv)
```

Los números de filas, columnas y niveles serán enteros (`Integer`) y las distancias lo más lógico es que sean de doble precisión (`Double`).

Las demás consideraciones, así como la manera de acceder a los objetos simples de la matriz, son las mismas que para `ArrayPolar`.

- `Copy`. Método que copia la entidad a la que se aplica en la misma posición que el objeto original. Su sintaxis es simplemente:

```
Set ObjCopia = ObjGráfico.Copy
```

ObjCopia será un objeto, por lo que estará definido como tal. Para copiar la línea sexta de la matriz del último ejemplo, la cual se había guardado en Línea6, se haría por ejemplo:

```
Dim Copia6 As Object
Set Copia6 = Línea6.Copy
```

Para copiar un objeto y moverlo a su vez, se utiliza primero este método y luego el método Move.

- Erase. Elimina la entidad a la que se aplica el método. Como en este caso no se ha de guardar objeto ni valor alguno en ninguna variable, se puede llamar al método con Call, pero al no necesitar argumentos no hace falta. Su sintaxis es pues:

```
ObjGráfico.Erase
```

Para borrar por ejemplo la línea sexta, del ejemplo que se viene manejando, guardada en Línea6 se hace:

```
Línea6.Erase
```

- GetBoundingBox. Este método devuelve las coordenadas inferior izquierda y superior derecha de la caja de abarque del objeto al que se le aplique. Esta caja de abarque es el rectángulo que se ajusta al objeto abarcándolo por completo. La sintaxis de GetBoundingBox es:

```
Call ObjGráfico.GetBoundingBox(VarInfIzq, VarSupDcha)
```

La forma de trabajar este método es un poco extraña. *VarInfIzq* y *VarSupDcha* han de ser forzosamente dos variables declaradas, antes de llamar al método, como Variant. Al final de la ejecución guardarán las coordenadas en cuestión. Si se desea ahora utilizar esos puntos para por ejemplo dibujar una línea, no se podrán utilizar como tales, sino que se habrá de realizar un trasvase de variables e introducir los diferentes valores en una matriz de tres elementos. A continuación se muestra una ampliación más de nuestro ejemplo de la línea:

```
Option Explicit
Dim AcadDoc As Object
Dim AcadModel As Object
Dim ObjLínea As Object
Dim PuntoInicial (1 To 3) As Double
Dim PuntoFinal (1 To 3) As Double
Dim MatrizLin As Variant
Dim PuntoBase (1 To 3) As Double
```

```
Sub DibujoLínea()
  Set AcadDoc = GetObject( ,"Autocad.Application").ActiveDocument
  Set AcadModel = AcadDoc.ModelSpace

  PuntoInicial(1) = 100: PuntoInicial(2) = 100: PuntoInicial(3) = 0
  PuntoFinal(1) = 200: PuntoFinal(2) = 200: PuntoFinal(3) = 0
  Set ObjLínea = AcadModel.AddLine(PuntoInicial, PuntoFinal)
  PuntoBase(1) = 10: PuntoBase(2) = 10: PuntoBase(3) = 0
  MatrizLin = ObjLínea.ArrayPolar(10, 3.14159, PuntoBase)
  Dim Línea6 As Object
  Set Línea6 = MatrizLin(6)
   Dim EsqInfIzq As Variant
  Dim EsqSupDch As Variant
  Call Línea6.GetBoundingBox(EsqInfIzq, EsqSupDch)
  Dim Pto1 (1 To 3) As Double
  Dim Pto2 (1 To 3) As Double
  Pto1(1) = EsqInfIzq(0): Pto1(2) = EsqInfIzq(1): Pto1(3) = EsqInfIzq(2)
  Pto2(1) = EsqSupDch(0): Pto2(2) = EsqSupDch (1): Pto2(3) = EsqSupDch(2)
  Set ObjLínea = AcadModel.AddLine(Pto1, Pto2)
End Sub
```

Se ve cómo se realiza el trasvase de variables. En el caso de las variables `Pto1` y `Pto2`, han sido definidas con un rango de `1` a `3`, sin embargo en `EsqInfIzq` y

`EsqSupDch` se utiliza el rango `0` a `2`, por lo que los índices están desfasados una unidad.

- `GetXData`. Permite recuperar los datos extendidos de una entidad. Su sintaxis es la que sigue:

```
Call ObjGráfico.GetXData(StrNomAplicación, VarTipoXDato,
VarValorXDato)
```

Los datos extendidos no los crea **AutoCAD** sino una aplicación AutoLISP o ADS. Sin embargo, estos datos se almacenan en la base de datos con los restantes del dibujo.

`StrNomAplicación` es el nombre de la aplicación que ha creado los datos extendidos; es una cadena (`String`). Si se especifica una cadena vacía se devuelven todos los datos extendidos de todas las aplicaciones.

`VarTipoXDato` y *`VarValorXDato`* son obligatoriamente dos variables de salida definidas como `Variant` antes de llamar al método. Esta variables almacenarán un *array* de códigos de tipo de datos y un *array* con el tipo de dato para cada código respectivamente.

- `Highlight`. Asigna lo que se denomina puesta en relieve de una entidad, es decir, su resalte o designación (línea de trazos en lugar de continua). La sintaxis es la siguiente:

```
ObjGráfico.Highlight(BooRelieve)
```

`BooRelieve` es `Boolean`. El valor `True` hace que la entidad se resalte y `False` que no se resalte. A veces puede ser necesario realizar una regeneración o utilizar el método `Update` para que los resultados sean visibles. La mayoría de las veces no hace falta.

• `IntersectWith`. Calcula todos los puntos de intersección de dos entidades y los devuelve en forma de matriz de puntos. La sintaxis es:

```
VarMatrizPtos = ObjGráfico1.IntersectWith(ObjGráfico2, OpciónExt)
```

Se indican los objetos gráficos con los que se desea calcular las intersecciones y una opción de extensión cuyos valores se presentan en la Tabla # 3.

Opción	Descripción
AcExtendNone	No se prolonga ninguna entidad
AcExtendThisEntity	Se prolonga *ObjGráfico1*
AcExtendOtherEntity	Se prolonga *ObjGráfico2*
AcExtendBoth	Se prolongan ambas entidades

Tabla # 3. Opciones de Extensión

El resultado es una matriz de valores de puntos tipo `Variant`. Si no hay puntos de intersección, su valor es nulo.

• `Mirror`. Produce una simetría de la entidad en cuestión con respecto a un eje de simetría que viene definido por dos puntos, ambos *arrays* de tres elementos tipo `Double`. El valor devuelto es un nuevo objeto simétrico al inicial (variable tipo `Object`).

La sintaxis de este método es:

```
Set ObjSimétrico = ObjGráfico.Mirror(DblPto1, DblPto2)
```

Los objetos no originales no son eliminados y, si se desea hacer, se deberá utilizar el método `Erase` tras la ejecución de la simetría.

A continuación se presenta un ejemplo de este método:

```
Option Explicit

Dim AcadDoc As Object
Dim AcadModel As Object
Dim ObjLínea As Object
Dim PuntoInicial(1 To 3) As Double
Dim PuntoFinal(1 To 3) As Double
Dim PuntoSim(1 To 3) As Double
Dim SimLin As Object
```

```
Sub DibujoLínea()
  Set AcadDoc = GetObject(, "Autocad.Application").ActiveDocument
  Set AcadModel = AcadDoc.ModelSpace
  PuntoInicial(1) = 100: PuntoInicial(2) = 100: PuntoInicial(3) = 0
  PuntoFinal(1) = 200: PuntoFinal(2) = 200: PuntoFinal(3) = 0
  Set ObjLínea = AcadModel.AddLine(PuntoInicial, PuntoFinal)
  PuntoSim(1) = 200: PuntoSim (2) = 100: PuntoSim (3) = 0
  Set SimLin = ObjLínea.Mirror(PuntoFinal, PuntoSim)
End Sub
```

En este ejemplo, tras dibujar una línea se realiza una simetría de la misma, apoyándose en un punto propio de la línea origen y en otro recién creado.

- `Mirror3D`. De manera similar al método anterior, `Mirror3D` produce una simetría tridimensional de un objeto con respecto a un plano definido por tres puntos. Los tipos de datos son iguales que en `Mirror`. La sintaxis de `Mirror3D` se presenta a continuación:

```
Set ObjSimétrico3D = ObjGráfico.Mirror3D(DblPto1, DblPto2, DblPto3)
```

• `Move`. Se utiliza para desplazar un objeto apoyándose en un punto de base que se moverá a otro punto indicado. Ambos puntos son matrices de tres elementos del tipo `Double`. Como no se obtiene ningún objeto, matriz de puntos o valor como devolución, sino simplemente un desplazamiento, se debe llamar al método con `Call`.

La sintaxis de este método es:

```
Call ObjGráfico.Move(DblPto1, DblPto2)
```

Si en el último ejemplo se hubiese querido desplazar el objeto simétrico, llevando el segundo punto del eje de simetría al último punto de la primera línea, se habría añadido el siguiente renglón tras la simetría:

```
Call SimLin.Move(PuntoSim, PuntoFinal)
```

• `Offset`. Este método crea una copia paralela o equidistante a un objeto y según una distancia dada. La copia será una matriz de objetos `Variant`; lo más lógico es que la distancia sea una variable `Double` (si es variable y no se indica directamente un valor):

```
VarEquidist = ObjGráfico.Offset(DblDistancia)
```

Para acceder posteriormente a cada objeto se puede utilizar la técnica de las matrices.

• `Rotate`. Gira el objeto en cuestión con respecto a un punto base (tabla o *array* de tres elementos `Double`) y según un ángulo indicado en radianes (`Double`). Su sintaxis es:

```
Call ObjGráfico.Rotate(DblPtoBase, DblÁngulo)
```

Para girar, por poner un ejemplo, la simetría que se ha movido en el ejemplo anterior, se añadirá al código del programa:

```
Call SimLin.Rotate(PuntoFinal, (3.14159 / 2))
```

• `Rotate3D`. Al igual que `Rotate` gira una entidad, esta vez en el espacio 3D y tomando como eje de giro el que determinan dos puntos. Los tipos de datos son los mismos. Su sintaxis es:

```
Call ObjGráfico.Rotate3D(DblPto1, DblPto2, DblÁngulo)
```

• `ScaleEntity`. Aplica un factor de escala al objeto con el cual esté asociado y a partir de un punto base indicado. El tipo de dato de la escala dependerá de la conveniencia del usuario, pero es lógico que sea `Double`. Su sintaxis es la que sigue:

```
Call ObjGráfico.ScaleEntity(DblPtoBase, DblFactorEscala)
```

• `SetXData`. Permite asignar datos extendidos a una entidad. Se suministra un *array* `Integer` con el código para cada tipo de dato y un *array* `Variant` con los datos, así:

```
Call ObjGráfico.SetXData(IntTipoXDato, VarValorXDato)
```

• `TransformBy`. Mueve, escala y gira el objeto mediante una matriz de transformación de 4× 4. Los elementos de la matriz son de tipo `Double` y deben tener una disposición concreta:

R00	R01	R02	T0
R10	R11	R12	T1
R20	R21	R22	T2
0 0	0	1	

La rotación se indica en la submatriz 3×3 que lleva la letra `R`, y en la columna marcada con `T` se indica la traslación. También en la submatriz de 3×3 es donde se indica el factor de escala. La sintaxis de `TransformBy` es:

```
Call ObjGráfico.TransformBy(DblMatriz)
```

• `Update`. Actualiza el objeto al que se le aplica, que puede ser una entidad o incluso todo el dibujo. Su sintaxis es la que sigue:

```
ObjGráfico.Update
```

Realizar una actualización de un objeto puede ser necesaria en determinadas ocasiones en que una operación en cuestión no produce el efecto deseado hasta que se actualiza. De querer por ejemplo actualizar la primera línea dibujada del ejemplo que se viene manejando hasta ahora se haría:

```
ObjLínea.Update
```

Si lo que se quiere es actualizar la aplicación, se habría de definir un objeto así por ejemplo:

```
Dim AcadApp As Object
```

para darle un valor así:

```
Set AcadApp = GetObject(, "AutoCAD.Application")
```

y luego:

```
AcadApp.Update
```

En este caso se declararía el objeto de documento activo simplemente así:

```
Set AcadDoc = AcadApp.ActiveDocument
```

Después de estudiar todas las propiedades y métodos del objeto de línea, se explicarán a continuación los demás métodos de las colecciones existentes para la creación de objetos o entidades gráficas de dibujo desde VBA. Muchas propiedades y/o métodos son comunes a todos los objetos gráficos, por lo que, de aquí en adelante, únicamente se expondrán aquellos que no se hayan visto ya.

Es decir, se explicarán algunas características nuevas de cada objeto gráfico sino se han visto con anterioridad en otro. De cada objeto se proporcionan listadas todas sus propiedades y listados sus métodos, resaltando en negrilla los que se explican en la sección en curso.

3.4.3- CÍRCULOS (10)

El método `AddCircle` permite dibujar círculos en la colección de objetos de Espacio Modelo, Espacio Papel o formando parte de un bloque. Hay que indicar las coordenadas del punto del centro, que será una matriz de tres elementos (coordenadas X, Y y Z) de tipo de dato `Double`, y el radio del círculo, que también será `Double`. La sintaxis de `AddCircle` es:

```
Set ObjCírculo = ObjColección.AddCircle(DblCentro, DblRadio)
```

Propiedades	Métodos
Application	ArrayPolar
Area	ArrayRectangular
Center	Copy
Color	Erase
EntityName	GetBoundingBox
EntityType	GetXData
Handle	Highlight
Layer	IntersectWith
Linetype	Mirror
LinetypeScale	Mirror3D
Normal	Move
ObjectID	Offset
Radius	Rotate
Thickness	Rotate3D
Visible	ScaleEntity
	SetXData
	TransformBy
	Update

Tabla # 4. Propiedades y Métodos de los Círculos

Como se puede observar en la Tabla # 4, los métodos son los mismos que para las líneas. En cuestión de propiedades desaparecen dos evidentemente (`StartPoint` y `EndPoint`) y aparecen `Area`, `Radius` y `Center`, que son las que se van a comentar a continuación.

- `Area`. Obtiene el área de una entidad cerrada (círculo, elipse, arco, polilínea, región o spline). La variable que guarde el dato de salida habrá sido declarado como `Double`. La sintaxis de esta propiedad es:

```
DblÁrea = ObjGráfico.Area
```

- `Center`. Obtiene y asigna el centro de la entidad. La sintaxis para asignar un nuevo centro es:

```
ObjGráfico.Center = DblPtoCentro
```

siendo *`DblPtoCentro`* un *array* de tres elementos `Double` que guarda las coordenadas X, Y y Z del nuevo centro. Para obtener el centro se utiliza la sintaxis siguiente:

```
VarPtoCentro = ObjGráfico.Center
```

donde *`VarPtoCentro`* será una variable declarada como `Variant` que guardará las coordenadas del centro. Se debe recordar que para utilizar luego estas coordenadas, a la hora de suministrarlas como puntos de dibujo para objetos gráficos, habrá que realizar un trasvase de variables y guardarlas en una matriz o *array* de tres elementos tipo `Double`.

- `Radius`. Obtiene y asigna el radio de un círculo (o de un arco). El valor en cuestión, tanto uno nuevo para asignar como el que recogerá una variable al obtener, será del tipo `Double`. Por lo que si se utilizan variables (al obtener seguro, pero el asignar no es obligatorio) deberán declararse como de doble precisión.

La sintaxis para asignar un radio es:

```
ObjGráfico.Radius = DblRadio
```

y la sintaxis para obtener el radio de una entidad ya dibujada es:

```
DblRadio = ObjGráfico.Radius
```

3.4.4- PUNTOS (10)

Para añadir puntos como entidades gráficas se utiliza el método `AddPoint`, ya sea en cualquiera de las colecciones que lo poseen (Espacio Modelo, Papel o bloques). La manera es sencilla, ya que únicamente hay que suministrar al método un punto que es, como siempre, una matriz o *array* de tres elementos (coordenadas X, Y y Z) de tipo de dato `Double`.

La sintaxis para este método es:

```
Set ObjPunto = ObjColección.AddPoint(DblPunto)
```

Propiedades	Métodos
Application	ArrayPolar
Color	ArrayRectangular
Coordinates	Copy
EntityName	Erase
EntityType	GetBoundingBox
Handle	GetXData
Layer	Highlight
LineType	IntersectWith
LinetypeScale	Mirror
Normal	Mirror3D
ObjectID	Move
Thickness	Rotate
Visible	Rotate3D
	ScaleEntity
	SetXData
	TransformBy
	Update

Tabla # 5. Propiedades y Métodos de los Puntos

En la Tabla # 5 se presenta un resumen de las propiedades y Métodos de los Puntos. Una vez explicados los métodos, sólo faltaría explicar una nueva propiedad.

- `Coordinates`. Obtiene y/o asigna una matriz con las coordenadas de cada uno de los vértices del objeto dado; en el caso del punto hay un único vértice.

La variable que se asigna a estos vértices (si la hubiera) se declara con una sentencia tipo `Dim Vértices (1 To n * 3) As Double`, donde *n* es el número de vértices o puntos.

Para asignar una matriz de coordenadas a un objeto punto:

```
ObjGráfico.Coordinates = DblMatrizVértices
```

y para obtenerla:

```
VarVértices = ObjGráfico.Coordinates
```

donde *VarVértices* es una variable declarada como `Variant`. Se debe recordar que si se quiere utilizar después las coordenadas habrá que hacer un trasvase a una matriz de tres elementos `Double`.

3.4.5- TEXTO (10)

El método `AddText` permite añadir textos en una línea al dibujo. Se debe de indicar la cadena de texto en cuestión (variable `String`), el punto de inserción en el SCU (matriz `Double` de tres valores) y la altura (`Double`). La sintaxis es pues la que sigue:

```
Set ObjTexto = ObjColección.AddText(StrTexto, DblPtoIns, DblAltura)
```

Propiedades	Métodos
Application	ArrayPolar
Color	ArrayRectangular
EntityName	Copy
EntityType	Erase
Handle	GetBoundingBox
Heigth	GetXData
HorizontalAlignment	Highlight
InsertionPoint	IntersectWith
Layer	Mirror
LineType	Mirror3D
LinetypeScale	Move
Normal	Rotate
ObjectID	Rotate3D
ObliqueAngle	ScaleEntity
Rotation	SetXData
ScaleFactor	TransformBy
StyleName	Update
TextAlignmentPoint	
TextGenerationFlag	
TextString	
Thickness	
VerticalAlignment	
Visible	

Tabla # 6. Propiedades y Métodos de los Textos

En la Tabla # 6 se presentan las Propiedades y Métodos de los Textos. Después de explicar los métodos, a continuación se estudian las nuevas propiedades.

- `Heigth`. Obtiene y/o asigna la altura de un objeto. En el caso del texto se refiere a la altura de las letras mayúsculas. La sintaxis para asignar una altura a un objeto es:

```
ObjGráfico.Heigth = DblAltura
```

y para obtenerla:

```
DblAltura = ObjGráfico.Heigth
```

siendo *DblAltura* una variable declarada como Double.

• HorizontalAlignment. Obtiene o asigna la alineación horizontal de y a los textos exclusivamente. Las sintaxis son:

para asignar:

```
ObjTexto.HorizontalAlignment = IntAlineaciónH
```

para obtener:

```
IntAlineaciónH = ObjTexto.HorizontalAlignment
```

siendo los valores para *IntAlineaciónH* del tipo Integer, aunque también se admiten las siguientes constantes:

```
acHorizontalAlignmentLeft
acHorizontalAlignmentCenter
acHorizontalAlignmentRight
acHorizontalAlignmentAligned
acHorizontalAlignmentMiddle
acHorizontalAlignmentFit
```

• InsertionPoint. Esta propiedad obtiene y asigna el punto de inserción de una entidad. El punto de inserción es una matriz de tres elementos Double, como cualquier otro punto:

```
ObjGráfico.InsertionPoint = DblPtoIns
```

con esta sintaxis se asigna un nuevo punto de inserción a un objeto. Para extraer u obtener el punto de inserción de una entidad (ahora Variant) se utilizará:

```
VarPtoIns = ObjGráfico.InsertionPoint
```

• ObliqueAngle. Asigna y obtiene el ángulo de oblicuidad de un texto medido en radianes. Para valores positivos se produce una inclinación a la derecha; los valores negativos se transforman en su equivalente positivo para almacenarlos, para ello se les suma 2Π.

para asignar un ángulo de oblicuidad:

```
ObjTexto.ObliqueAngle = DblÁngulo
```

para obtener el ángulo de oblicuidad de un texto:

```
DblÁngulo = ObjTexto.ObliqueAngle
```

DblÁngulo es un variable de doble precisión `Double`.

- `Rotation`. Obtiene y/o asigna un ángulo de rotación en radianes (`Double`) para un texto, medido a partir del eje X y en sentido antihorario.

para asignar:

```
ObjTexto.Rotation = DblÁngulo
```

para obtener:

```
DblÁngulo = ObjTexto.Rotation
```

- `ScaleFactor`. Obtiene y/o asigna el factor de escala de anchura (`Double`) para una entidad de texto.

para asignar:

```
ObjTexto.ScaleFactor = DblEscala
```

para obtener:

```
DblEscala = ObjTexto.ScaleFactor
```

- `StyleName`. Obtiene y/o asigna el nombre (`String`) de estilo de texto (o acotación) empleado para el objeto. El estilo deberá estar definido en el dibujo, si no es así se producirá un error. Si no se asigna estilo se asume por defecto el actual.

para asignar:

```
ObjGráfico.StyleName = StrEstilo
```

para obtener:

```
StrEstilo = ObjGráfico.StyleName
```

• `TextAlignmentPoint`. Obtiene y/o asigna la posición del punto de alineación del texto. Se trata pues de un *array* de tres elementos de tipo `Double` a la hora de asignar, y de una variable `Variant` a la hora de recoger los valores de un objeto texto.

para asignar pues:

```
ObjTexto.TextAlignmentPoint = DblPtoAlineación
```

para obtener:

```
VarPtoAlineación = ObjTexto. TextAlignmentPoint
```

• `TextGenerationFlag`. Obtiene y/o asigna el tipo de generación del texto. Es un valor `Integer` pero que posee también estas dos constantes:

```
acTextFlagBackward
acTextFlagUpsideDown
```

para aplicar una generación de "atrás hacia adelante" y de "arriba a abajo" respectivamente. Si se quieren aplicar los dos efectos se deben ejecutar dos instrucciones, una con cada valor.

para asignar pues:

```
ObjTexto.TextGenerationFlag = IntGeneración
```

para obtener:

```
IntGeneración = ObjTexto. TextGenerationFlag
```

• `TextString`. Obtiene y/o asigna la cadena de texto de una entidad texto. Se pueden incluir en la cadena (valor tipo `String`) los caracteres especiales de **AutoCAD** (los precedidos por `%%`).

para asignar una cadena la sintaxis es:

```
ObjTexto.TextString = StrTexto
```

para obtener una cadena la sintaxis es:

```
StrTexto = ObjTexto.TextString
```

• `VerticalAlignment`. Obtiene o asigna la alineación vertical de y a los textos exclusivamente. Las sintaxis son:

para asignar:

```
ObjTexto.VerticalAlignment = IntAlineaciónV
```

para obtener:

```
IntAlineaciónV = ObjTexto.VerticalAlignment
```

siendo los valores para *`IntAlineaciónV`* del tipo `Integer`, aunque también se admiten las siguientes constantes:

```
acVerticalAlignmentBaseline
acVerticalAlignmentBottom
acVerticalAlignmentMiddle
acVerticalAlignmentTop
```

4.- METODOLOGÍA

4.- METODOLOGÍA

A continuación se explican los pasos que se siguieron para la realización de cada una de las rutinas que comprenden la barra de herramientas "Perfiles de Acero"

4.2- RUTINA "Meter Datos"

Esta rutina permite que el usuario introduzca en el archivo de AutoCAD las características que definen a cada uno de los perfiles que componen el plano.

A continuación se explica el funcionamiento de esta rutina:

Se creó la siguiente forma (ver Figura # 4):

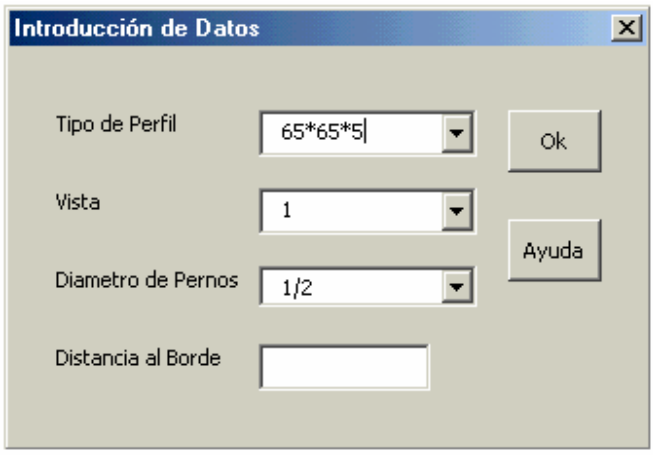


Figura # 4. Forma para definir característica de perfil

En la forma se puede observar que:

El combobox del Tipo de Perfil se rellenó con los datos de la Tabla # 7.

El combobox de Vista se rellenó con los números del 1 al 8, que representan los distintos tipos de vistas de cada perfil.

El combobox de Diámetro de Pernos se rellenó con los datos suministrados por la Tabla # 8.

Con el dato de Tipo de Perfil, el programa busca en el archivo de texto "Datos.txt" las dimensiones de este perfil "L" y " t", observar Figura # 5.

La distancia al borde (dB) es la distancia que existe entre el borde del perfil y la línea de pernos, véase Figura # 5.

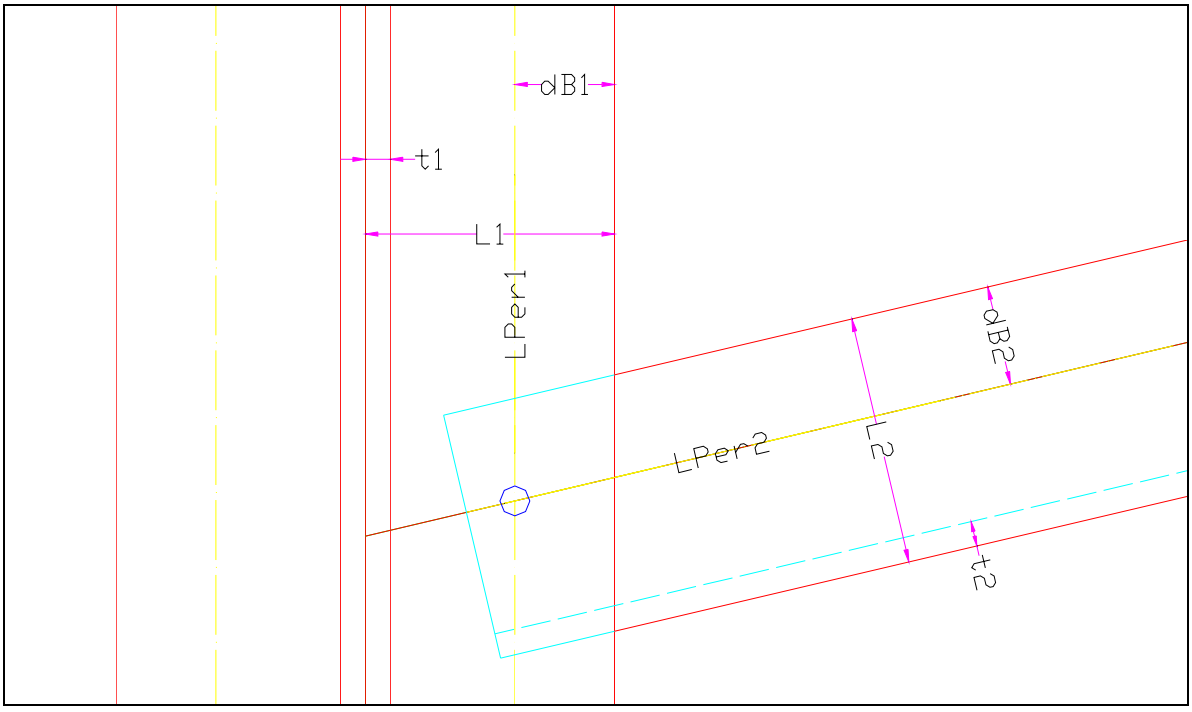


Figura # 5. Imagen de las dimensiones del perfil (L, t y dB)

El AutoCAD permite crear un archivo de datos adjuntos a cada objeto que se encuentre en el archivo (11).

El usuario al llenar esta forma está dando a conocer las siguientes variables L, t, vista, diámetro y dB, y al seleccionar una línea permite que estos datos sean introducidos en el archivo de datos adjuntos que AutoCAD tiene reservado para esta línea que representa a un perfil con estas características.

Una vez introducidos los datos en la línea, ésta cambiará de la capa "0" a "Línea de Cg" y se tornará de color verde.

A continuación, se presentan los perfiles y diámetros de pernos creados en el programa, fíjese en Tabla # 7 y Tabla # 8.

Tipo de Perfil
L*L*t (mm * mm * mm)
20*20*3
20*20*4
25*25*3
25*25*4
30*30*3
30*30*5
35*35*4
35*35*6
40*40*4
40*40*6
50*50*4
50*50*5
50*50*7
65*65*5
65*65*6
65*65*7
75*75*7
75*75*8
90*90*7
100*100*8
100*100*10
110*110*8
110*110*10
120*120*10
120*120*12

Tabla # 7. Perfiles L Sidor de Alas Iguales (12)

Diámetro de Pernos
1 / 2 ”
5 / 8 ”
3 / 4 ”
7 / 8 ”
1 ”
1 1/8 ”

Tabla # 8. Diámetros Comerciales de Pernos

4.3- RUTINA “Unión de Perfiles”

Esta rutina realiza los cortes necesarios para la conexión de perfiles angulares unidos mediante un perno. Para utilizar esta instrucción es necesario haber definido anteriormente las características de los perfiles con la rutina “Meter Datos”. A continuación se explica el funcionamiento de dicha rutina:

Se apagan todos las capas (layers) con la excepción de la capa “Línea de Cg”, la capa “Línea de Cg” es la que contiene las líneas que definen la geometría y las características que definen a cada uno de los perfiles de la estructura, esto se hace con la finalidad de restringir los objetos que el usuario pueda seleccionar en el plano.

Subsecuentemente, se definen los tipos de líneas con las cuales se van a dibujar las líneas que representan a la conexión.

Se activan todos las capas (layers), con la finalidad de que se vean las líneas que definen el plano en pantalla.

Posteriormente, se le pide al usuario que seleccione las líneas que definen a la conexión. Estas líneas son Linea1v y Linea2v respectivamente.

Se extraen los datos que definen las características de cada uno de los perfiles.

Con la rutina "Saca_datos", los datos que se van a extraer se encuentran en el archivo de datos adjuntos que AutoCAD tiene reservado para cada objeto del dibujo y que el usuario debió haber definido previamente.

Se verifica que el diámetro de los pernos del perfil #1 y del perfil #2 sean iguales.

Consecutivamente, se dibuja Linea1, como la línea comprendida entre los puntos medios de Linea1v definidos por esta conexión con las conexiones próximas, con la rutina "recorta_linea_1".

Se dibuja Linea2, como la línea comprendida entre los puntos medios de Linea2v definidos por esta conexión con las conexiones próximas, con la rutina "recorta_linea_2".

Se ajustan Linea1 y Linea2, de tal manera que el punto de inicio de ambas líneas esté a la izquierda de la pantalla, esto se hace para poder referenciar con mayor facilidad los puntos de inicio y terminación de dichas líneas, con la rutina "arregloID".

Se verifica que el perfil #2 sea un perfil sencillo, dado que este tipo de conexión no se puede realizar entre dos perfiles dobles.

Se dibuja el perfil #1 de la siguiente manera:

Si el perfil #1 es un perfil doble Linea1 coincide con L12.

L11s se dibuja como una paralela a una distancia l1 del lado superior en la pantalla de L12.

L11i se dibuja como una paralela a una distancia l1 del lado inferior en la pantalla de L12.

L13s se dibuja como una paralela a una distancia t1 del lado superior en la pantalla de L12.

L13i se dibuja como una paralela a una distancia t1 del lado inferior en la pantalla de L12.

LPer1s se dibuja como una paralela a una distancia l1xt1 del lado superior en la pantalla de L12.

LPer1i se dibuja como una paralela a una distancia l1-x1 del lado inferior en la pantalla de L12.

En el caso de que L13i intercepte a Linea2 entonces L11, L13 y Lper1 serán una copia de L11i, L13i y LPeri respectivamente, de no ser así, L11, L13 y Lper1 serán una copia de L11s, L13s y LPers.

El diseño de las líneas que definen el perfil y la definición de L11, L13 y Lper1 lo hace la rutina "Dibuja_Perfiles".

En el caso de que el perfil #1 sea un perfil sencillo se dibujará de la misma manera que el perfil #2.

Observar Figura # 6.

Se dibuja el perfil #2 de la siguiente manera:

Se hace una copia de Linea2 llamada Lper2, en vista de que el perfil # 2 es un perfil sencillo, Linea2 coincide con la línea de pernos LPer2.

L21 se dibuja como una paralela a una distancia x2 de la línea LPer2 (el lado en el cual se dibujara esta línea lo define el tipo de vista del perfil).

L22 se dibuja como una paralela a una distancia l2-x2 de LPer2.

L23 se dibuja como una paralela a una distancia l2-x2-t2 de LPer2.

Observar Figura # 6.

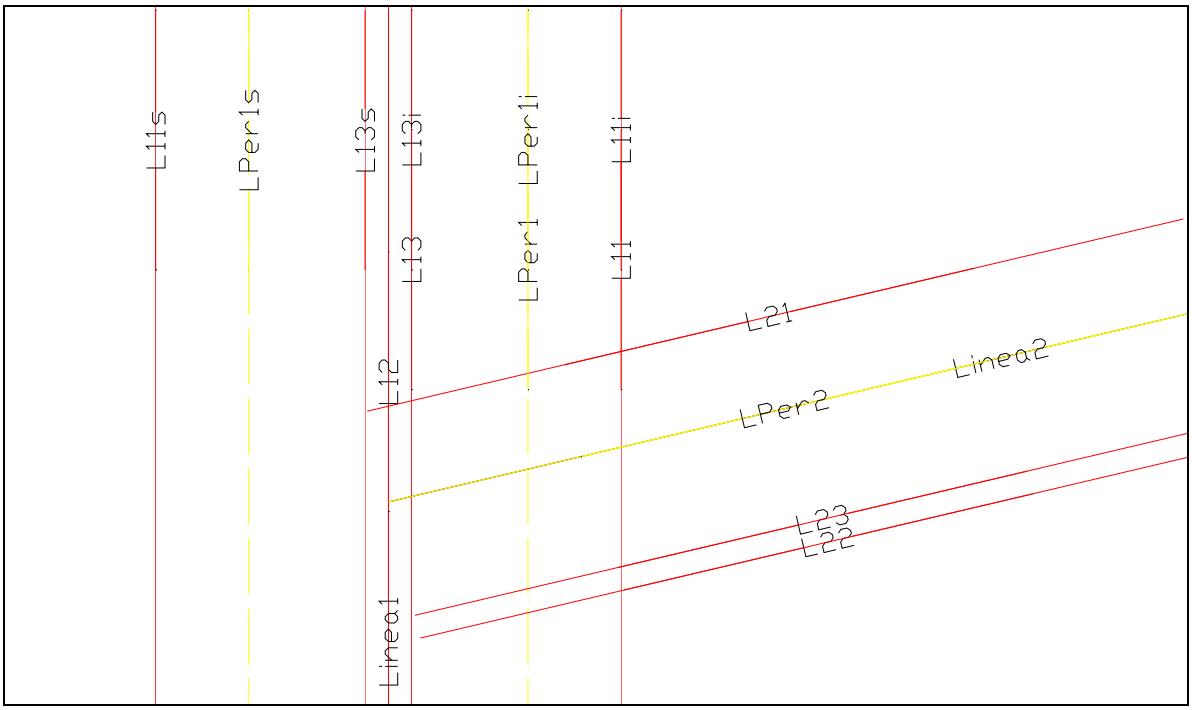


Figura # 6. Compendio de Líneas a Trazar para la Construcción de Conexiones

A continuación, se crea una línea LPer2 nueva que va desde la intercepción de LPer2 con LPer1 hasta el punto de comienzo de LPer2, se borra la línea vieja de LPer2. La intercepción de LPer1 con LPer2 define el punto en el cual va el perno de la conexión. Con la rutina "extiende_linea_a_linea".

Se ordenan L21 y L22 de manera tal que el punto de terminación de dichas líneas sea el punto que esté más cercano al centro del perno. Con la rutina "arregloSE_de_linea_con_punto".

Se dibuja LAUX, que es una línea que parte el punto de terminación de desde L21, pasa por centro del perno y termina en el punto de terminación de L22.

Se verifica que LAUX no intercepte a L13, de hacerlo los perfiles estarían chocando y el programa mostraría una ventana de diálogo informándolo.

Se dibuja el perno en la intersección de LPer1 con LPer2.

Se calculan las líneas de cada perfil que se deben interceptar:

En perfil #1 se verifica si L11 o L12 interceptan a LPer2, de ser L11 entonces L1m1 será una copia de L11 y L1m2 será una copia de L12, y viceversa. Con la rutina "Calcula_lineas_que_mandan1".

En perfil #2 se interceptan L11 con L21 y L22, y L13 con L21 y L22, d1 será la distancia formada entre la intercepción de L21 con L11 y la intercepción de L22 con L13, d2 será la distancia formada entre la intercepción de L21 con L13 y la intercepción de L22 con L11, de ser d1 mayor que d2 L2m1 será una copia de L21 y L2m2 será una copia de L22, de ser d2 mayor que d1, entonces L2m1 será una copia de L22 y L2m2 será una copia de L21. Con la rutina "Calcula_lineas_que_mandan2".

Observar Figura # 7.

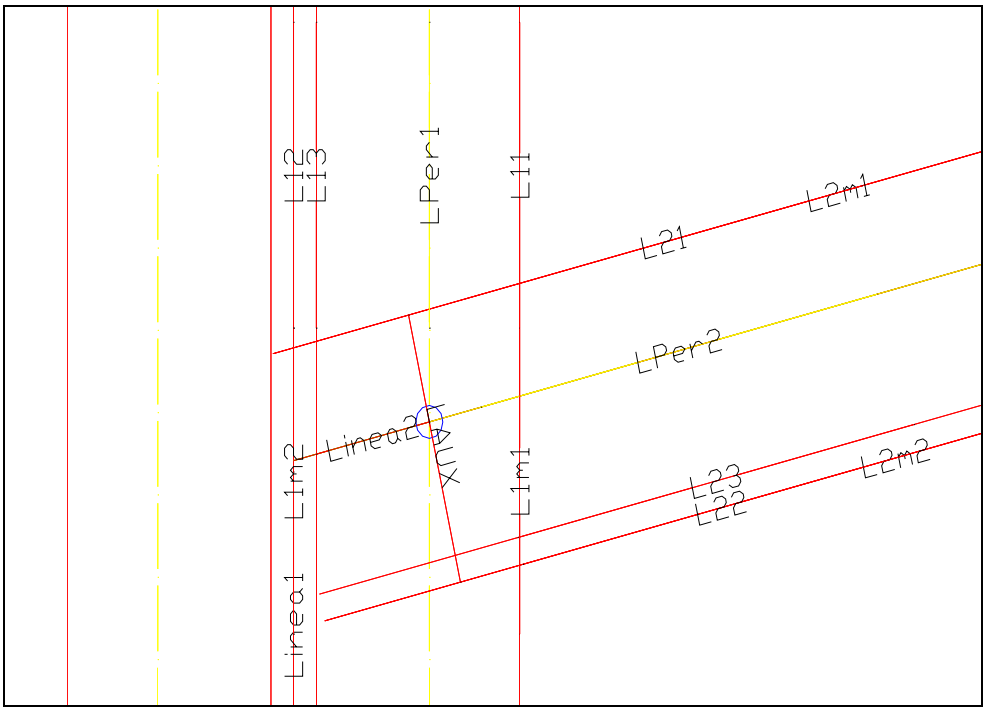


Figura # 7. Sumario de Líneas a Esbozar para la Construcción de Conexiones

La distancia dmax será la menor de las siguientes distancias: la comprendida entre la intercepción de L21 con L13 y el punto de inicio de LAUX o la comprendida entre la intercepción de L22 con L13 y el punto de final de LAUX. Esto se hace con la finalidad de asegurar que los perfiles no choquen. En el caso de que dmax sea menor que el radio del perno el programa mostraría una ventana de diálogo informándolo.

Con la rutina "distancias_posibles" se crea "d"; dónde el usuario proporciona la distancia "d" a partir del perno, comprendida entre el radio del perno y dmax.

Con la rutina "extiende_linea_a_linea" y conocida la distancia "d", se crea Lper2x, la cual es una copia de Lper2, y se extiende una distancia "d" en la dirección en la cual se encuentra ubicado el perfil #1.

Se eliminan Lper2, L21, L22, L23, LAUX.

Se dibuja el perfil #2, a partir de LPer2x repitiendo el procedimiento anterior.

Con la rutina "arregloSE_de_linea_con_punto", se acopla LPer2x de manera tal que el punto de culminación de dichas líneas sea el punto que esté más cercano al centro del perno.

Se desplaza LAUX desde el centro del perno hasta el punto de terminación de LPer2x.

Para realizar la visibilidad se intercepta L1m1 con L21 y L22, se crean 3 líneas nuevas desde la intercepción de L1m1 con L21 hasta el punto de inicio de L21, desde la intercepción de L1m1 con L22 hasta el punto de inicio de L22 y desde el punto de inicio de L21 hasta el punto de inicio de L22., las cuales tendrán un color "Cyan" en el caso de que se deban ver en el plano.

Se eliminan L1m1, L1m2, L2m1, L2m2, Linea1, Linea2 y Lper2.

Se revisa en todo el plano que las líneas no estén dibujadas 2 veces, en caso de estarlo el programa borrara una de estas, esto se hizo para asegurar que si el usuario realiza 2 veces la misma conexión no queden líneas sobrepuestas.

Observar las Figuras # 8 y # 9.

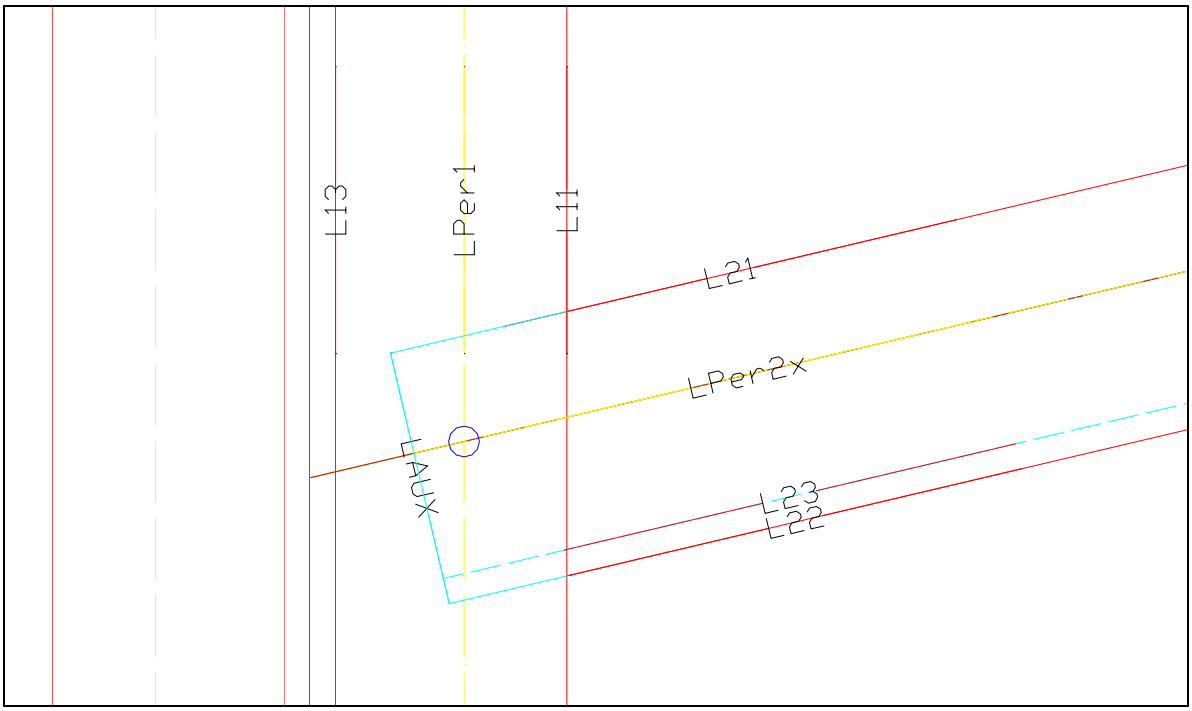


Figura # 8. Conexión de Perfiles por un Perno

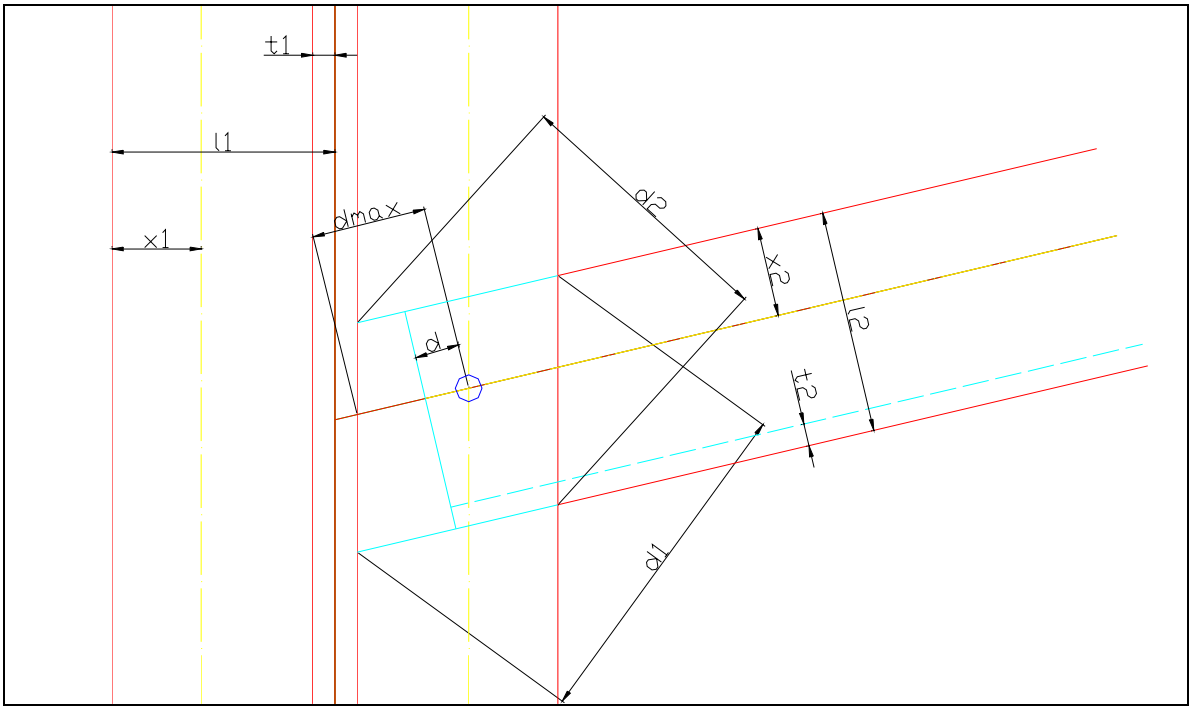


Figura # 9. Distancias Utilizadas

4.4- RUTINA "Unión para Planchas"

Esta rutina realiza los cortes necesarios para la conexión de perfiles angulares unidos mediante una plancha pero sin dibujar la plancha. Para utilizar esta instrucción es necesario haber definido anteriormente las características de los perfiles con la instrucción "Meter Datos".

A continuación se explica el funcionamiento de esta rutina:

Se apagan todos las capas (layers) con la excepción de la capa (layer) "Línea de Cg", la capa (layer) "Línea de Cg" es el que contiene las líneas que definen la geometría y las características que definen a cada uno de los perfiles de

la estructura, esto se hace con la finalidad de restringir los objetos que el usuario pueda seleccionar en el plano.

Se definen los tipos de líneas con las cuales se van a dibujar las líneas que representan a la conexión.

Se le pide al usuario que seleccione las líneas que definen a la conexión. Estas líneas son Linea1v y Linea2v respectivamente.

Se activan todos los layer. Con la finalidad de que se vean las líneas que definen el plano en pantalla.

Se extraen los datos que definen las características de cada uno de los perfiles. Con la rutina "Saca_datos". Los datos que se van a extraer se encuentran en el archivo de datos adjuntos que AutoCAD tiene reservado para cada objeto del dibujo y que el usuario debe de haber definido previamente.

Se le pide al usuario el número de pernos (np) que tendrá el perfil #2.

Se le pide al usuario la distancia que separa a los pernos (dP) que tendrá el perfil #2.

Se le pide al usuario la distancia entre el perno más próximo al borde y el borde (dB) del perfil #2.

Se le pide al usuario la distancia perpendicular al perfil #1 que separará a los perfiles (disPer).

Con la rutina "recorta_linea_1", se dibuja Linea1, como la línea comprendida entre los puntos medios de Linea1v definidos por esta conexión con las conexiones próximas.

Con la rutina "recorta_linea_2", se dibuja Linea2, ésta es la línea comprendida entre los puntos medios de Linea2v definidos por esta conexión hasta las conexiones próximas.

Con la rutina "arregloID", se ordenan Linea1 y Linea2, de tal manera que el punto de inicio de ambas líneas esté a la izquierda de la pantalla, esto se hace para poder referenciar con mayor facilidad los puntos de inicio y terminación de dichas líneas.

Se dibuja el perfil # 1, con la rutina "dibuja_perfil", de igual manera que en el módulo anterior.

Se calculan las líneas que rigen en el perfil #1, con la rutina "Call calcula_lineas_que_mandan1", de igual manera que en el módulo anterior.

A continuación, se dibujó una línea horizontal La, en el origen del plano.

Posteriormente, se trazó una línea Lb, comprendida entre el punto de inicio de L1m1 y el punto de inicio de L1m2.

De allí, se mueve La desde su punto de terminación hasta el punto de inicio de Lb. Se rota La un ángulo igual al de Lb. Se dibuja una copia de L1m1, llamada Lc.

Subsiguiente a eso, Lc se mueve desde el punto de inicio de L1m1 hasta el punto de inicio de La. Lc quedará como una línea paralela a L1m1 a una distancia disPer en la dirección en la que se encuentra el perfil #2. Véase Figura # 10.

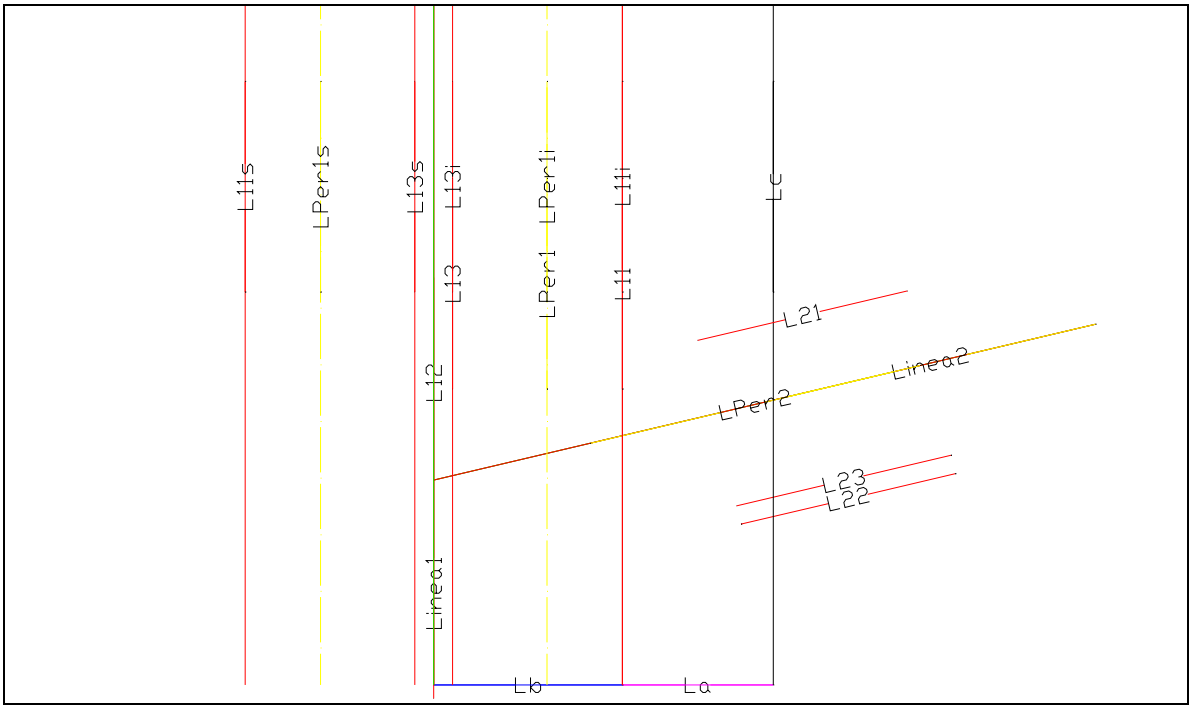


Figura # 10. Conjunto de Líneas Trazadas para Unión de Planchas

Se dibuja el perfil #2 con la rutina "dibuja_perfil" , de igual manera que en el módulo anterior.

Se calculan las líneas que determinan en el perfil #2 con la rutina "calcula_lineas_que_mandan2", de igual manera que en el módulo anterior.

Con la rutina "arregloSE_de_linea_con_punto", se ordenan L21, L22, L23 y Lper2 de manera tal que el punto de terminación de dichas líneas sea el punto que esté mas cercano al centro del perno.

Consecutivamente, se traza LAUX desde el punto de inicio de L2m1 hasta el punto de inicio de L2m2. Se mueve LAUX desde el punto de inicio de L2m1 hasta el punto de intercepción de L2m1 con Lc. Se borra L21, L22, L23, Lper2, L21i, L23i, L21s, L23s, LPer2i y LPer2s.

Luego, se dibuja Linea22, donde se intercepta LAUX con Linea2 hasta el punto de inicio de Linea2. Se acomoda Linea22, de tal manera que el punto de inicio esté a la izquierda de la pantalla, con la rutina "arregloID". Ver Figura # 11.

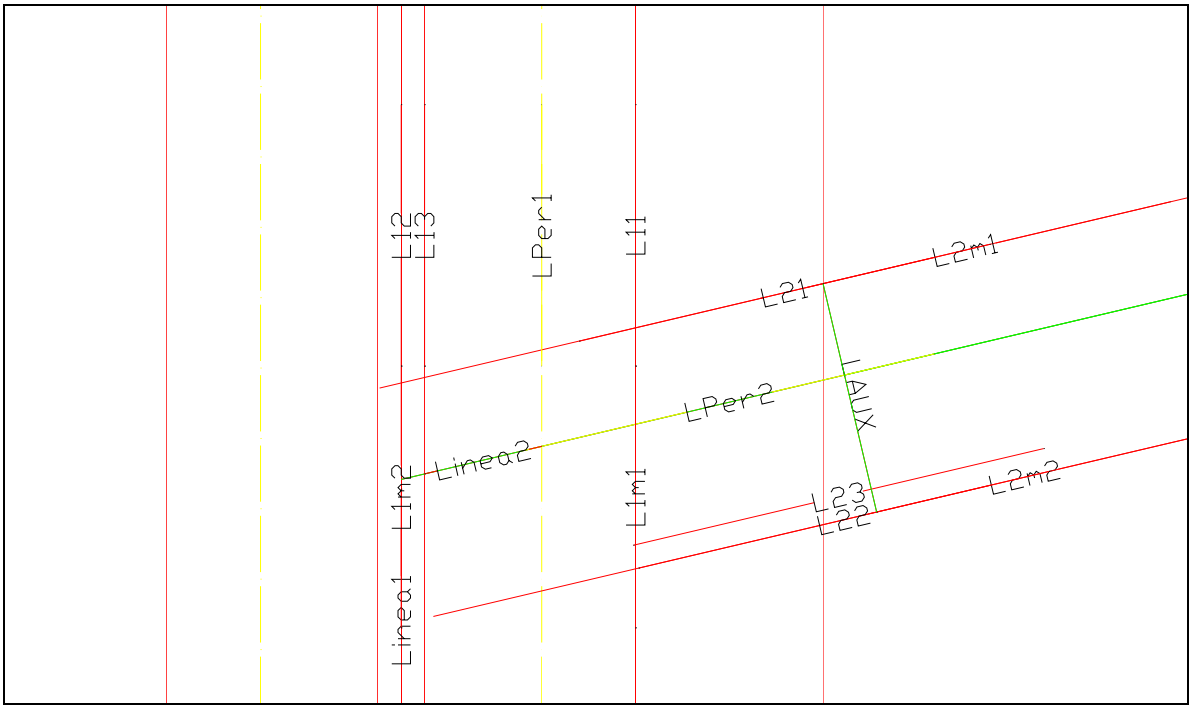


Figura # 11. Líneas Dibujadas para Construcción de Unión de Planchas

Ahora, se dibuja el perfil # 2 a partir de Linea22 con la rutina "dibuja_perfil". Se esboza una línea horizontal LdB que tiene una longitud dB, se mueve desde su punto de comienzo hasta el punto de terminación de Lper2, se rota esta línea un ángulo igual al de Lper2.

Posteriormente, se delinea una raya horizontal LdP que tiene una longitud dP, se mueve desde su punto de comienzo hasta el punto de terminación de Lper2, se rota esta línea un ángulo igual al de Lper2 y se mueve desde su punto de terminación hasta el punto de comienzo de LdB. Se dibuja el primer perno en el punto de terminación de LdP.

Se mueve LdP desde su punto de terminación hasta su punto de comienzo y se dibuja otro perno, esta acción se repite np-1 veces. Se borran LdP y LdB.

Obsérvese Figura # 12.

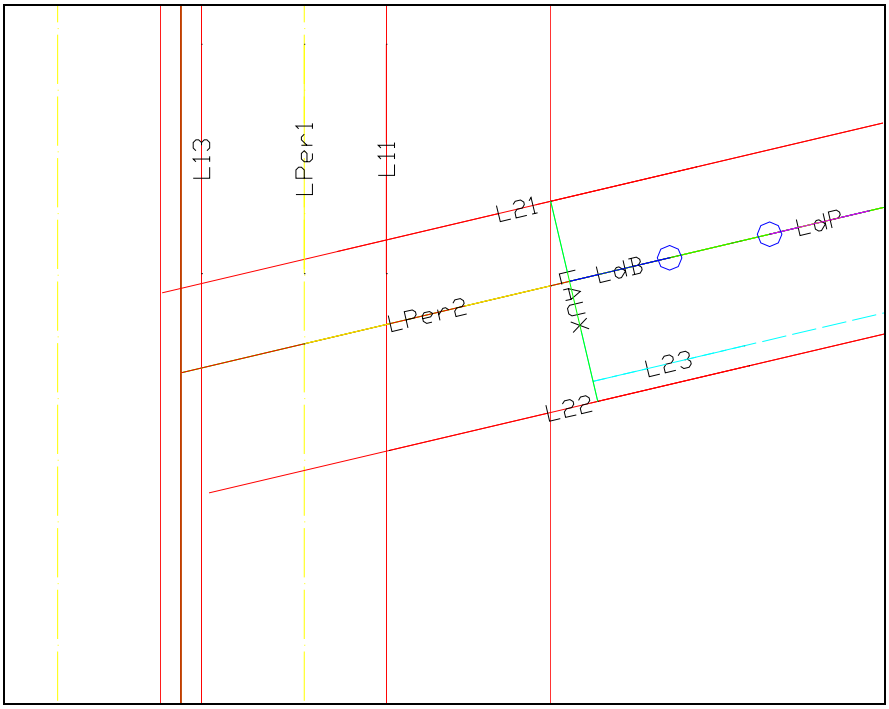


Figura # 12. Líneas Trazadas para la Realización de Planchas

Estas acciones se hacen con la finalidad de dibujar todos los pernos del perfil 2 y se hace con la rutina "Dibuja_Pernos". En el caso de que el perfil 2 sea doble se repite el procedimiento pero en vez de utilizarse Lper2, se utilizarían Lper2s y Lper2i respectivamente.

Luego, se borran La, Lb, Lc, Linea1, Linea2, Linea22, L1m1, L1m2, L2m1 y L2m2.

Véase Figura # 13.

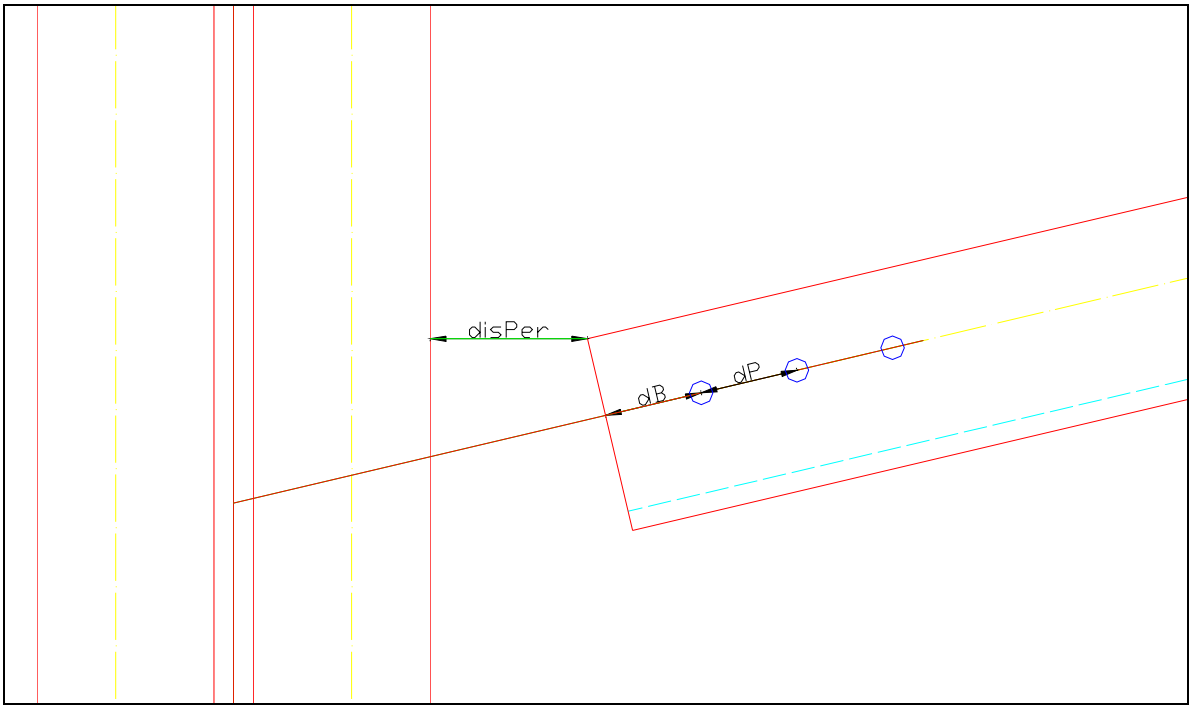


Figura # 13. Unión para Plancha

4.5- RUTINA "Separadores"

Esta rutina permite al usuario determinar el tipo de separador que desea dibujar. Para utilizar esta instrucción es necesario haber realizado las distintas conexiones del plano.

A continuación se explica el funcionamiento de esta rutina:

Se creó la siguiente interfaz (Figura # 14), con la finalidad de que el usuario tenga la opción de escoger que tipo de separador desea dibujar.

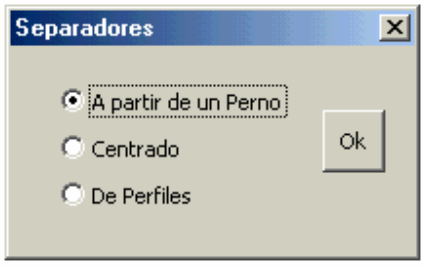


Figura # 14. Interfaz para la Construcción de Separadores

4.5.2- RUTINA "Separador a partir de un Perno"

Esta subrutina crea un separador a partir de una conexión ya realizada por el módulo Unir_Perfiles.

A continuación se explica el funcionamiento de esta subrutina:

Se desactivan todos las capas (layers) con la excepción de "Línea de Cg" y de "Pernos", esto se hace con la finalidad de restringir los objetos que el usuario pueda seleccionar en el plano.

Se le pide al usuario que seleccione el perno (Perno) a partir del cual se va a referenciar el separador.

Seguidamente, se le demanda al usuario que seleccione la línea (Línea) que define el perfil en el cual va a estar el separador.

Consecutivamente, se extraen los datos que definen las características del perfil; con la rutina "Saca_datos". Los datos que se van a extraer se encuentran en el archivo de datos adjuntos que AutoCAD tiene circunspecto para cada objeto del dibujo y que el usuario debió haber definido previamente.

Posteriormente, se activan todas las capas (layers). Con la finalidad de que se vean las líneas que definen el plano en pantalla.

A partir de allí, se verifica que el perfil en el cual se va a encontrar el separador sea un perfil doble, de no ser así el programa mostrará un cuadro de diálogo informándolo.

Se ajusta Línea, de tal manera que el punto de inicio esté a la izquierda de la pantalla, esto se hace para poder hacer punto de referencia con mayor facilidad el punto de inicio y terminación de la línea. Con la rutina "arregloID".

A continuación, se le exige al usuario que defina la posición de Perno (Pos) en el separador, con la finalidad de poder hacer referencia el dibujo del separador al Perno. Pos puede ser a la izquierda, a la derecha o central.

Se le pide al usuario que defina el número de pernos por ala (np), que va a tener dicho separador.

De ser la posición de Perno central, se verifica que el número de pernos sea un número impar, con la finalidad de que la distribución de los pernos sea simétrica.

Se le pide al usuario que defina la distancia que separa a los pernos (dP).

Se le pide al usuario que defina la distancia que separa al perno mas cercano del borde del separador con el borde del separador (db).

Se dibuja el perfil, con la rutina "dibuja_perfil_solo", esta rutina dibuja el perfil de la misma forma que "dibuja_perfil", pero se diferencian en que "dibuja_perfil_solo" no define cuales son L11, L13 y Lper1.

Se le hace un reflejo de Perno (PernoAux) desde el punto de inicio hasta el punto de terminación de Línea.

Se dibuja Lp1 desde el punto de inicio de L11i hasta el punto de inicio de Línea. Se dibuja Lp2 desde el punto de inicio de Línea hasta el punto de inicio de L11s.

Se intercepta Lper1i con Perno, de no interceptarse, Lper1i con Perno se intercepta Lper1s con Perno dando como resultado v, que es un vector que contiene las coordenadas de los 2 puntos de intercepción.

De ser la posición del perno de referencia igual a Derecha entonces NpD será igual a np.

De ser la posición del perno de referencia igual a Izquierda entonces NpI será igual a np.

De ser la posición del perno de referencia igual a Central entonces NpD será igual a la parte entera de la mitad de Np + 1 y NpI será igual a la parte entera de la mitad de Np + 1.

Posteriormente, se realizan una serie de operaciones: se dibuja una línea horizontal (LdP) en el origen del plano, de longitud igual a dP. Se mueve LdP desde el origen del plano hasta el centro de Perno. Se rota LdP un ángulo igual al de Línea. Se dibuja un perno (PernoAux) en el punto de terminación de LdP. Se mueve LdP desde su punto de inicio hasta su punto de terminación. Se hace un espejo de PernoAux desde el punto de inicio hasta el punto de terminación de LdP.

Ver Figura # 15.

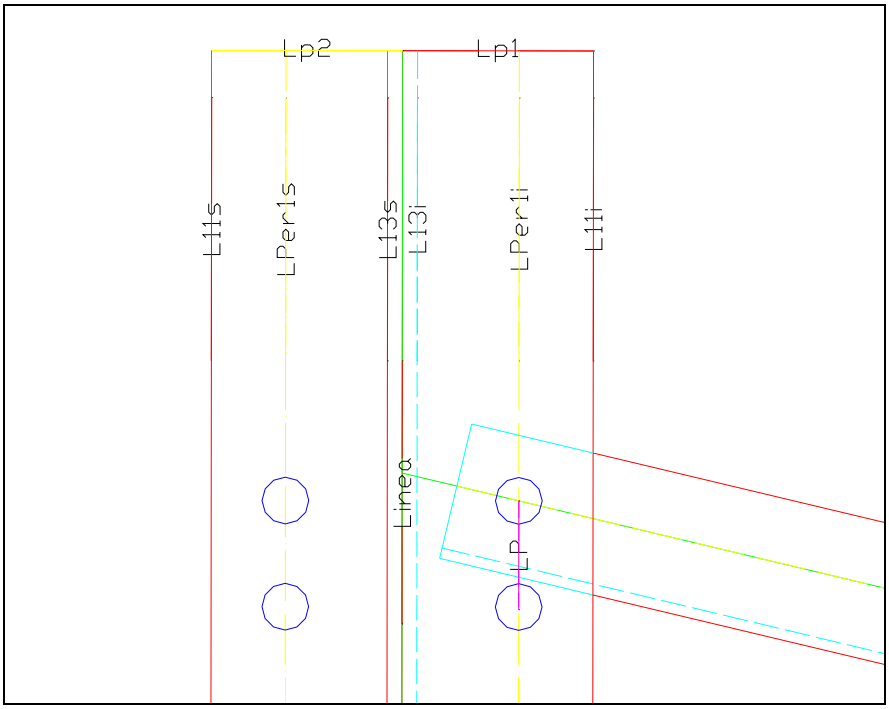


Figura # 15. Líneas a Realizar para la Construcción del Separador

Esto se repite NpD – 1 veces, con la finalidad de que se dibujen todos los pernos que se encuentran del lado derecho del perno de referencia.

Luego, se dibuja una línea horizontal (LdB) en el origen del plano, de longitud igual a dB. De ser Pos distinta a Derecha, se mueve LdB desde el punto de inicio de LdB hasta el centro del último perno dibujado. De ser Pos igual a Derecha se mueve LdB desde el punto de inicio de LdB hasta el centro de Perno. Se rota LdB un ángulo igual al de Línea. Se crea LpA1 como una copia de Lp1. Se crea LpB1 como una copia de Lp2. Se mueven LpA1 y LpB1 desde v hasta el punto de terminación de LdB.

Prosiguiendo, se borran LdB y LdP.

Se dibuja una línea horizontal (LdP) en el origen del plano, de longitud igual a dP. Se mueve LdP desde el origen del plano hasta el centro de Perno. Se rota LdP un ángulo igual al de Línea. Se mueve LdP desde su punto de terminación hasta su punto de inicio. Se vuelve a mover LdP desde su punto de terminación hasta su punto de inicio.

De allí, se dibuja un perno (PernoAux) en el punto de terminación de LdP. Se mueve LdP desde su punto de inicio hasta su punto de terminación. Se hace un reflejo de PernoAux desde el punto de inicio hasta el punto de terminación de LdP.

Esto se repite NpI – 1 veces, con la finalidad de que se dibujen todos los pernos que se encuentran del lado izquierdo del perno de referencia.

Se dibuja una línea horizontal (LdB) en el origen del plano, de longitud igual a dB. De ser Pos distinta a Izquierda, se mueve LdB desde el punto de inicio de LdB hasta el centro del último perno dibujado. De ser Pos igual a Izquierda se mueve LdB desde el punto de inicio de LdB hasta el centro de Perno.

Luego, se rota LdB un ángulo igual al de Línea. Se mueve LdB desde su punto de terminación hasta su punto de comienzo. Se crea LpA2 como una copia de Lp1. Se crea LpB2 como una copia de Lp2. Se mueven LpA2 y LpB2 desde v hasta el punto de inicio de LdB. Se revisa con el tipo de vista el color que debe tener cada una de las líneas del separador.

Culminando, se borran LdP, lp1, lp2, L11i.„ L11s, L13i, L13s, LPer1i y LPer1s. Ver Figura # 16.

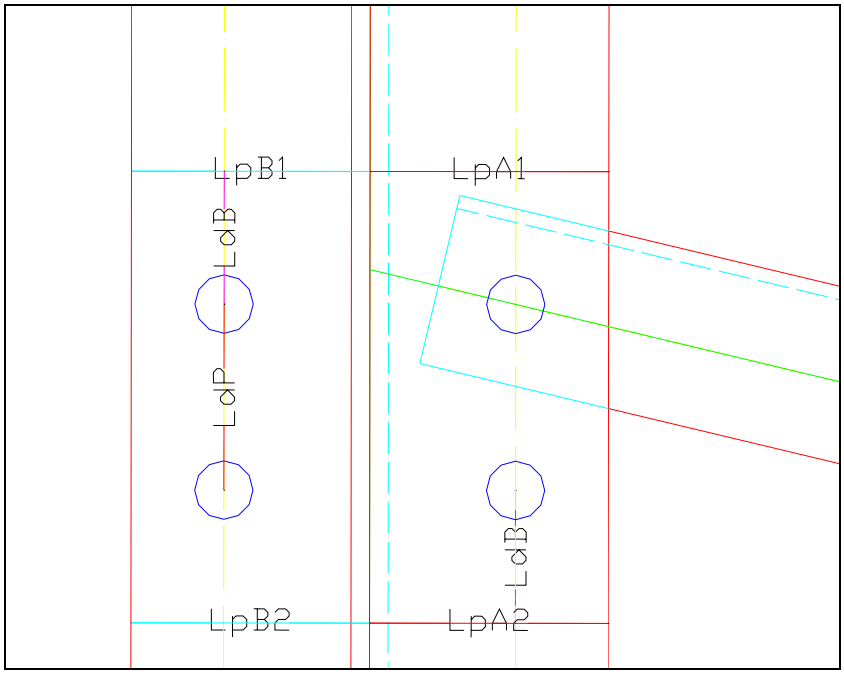


Figura # 16. Separador de Pernos

Para dibujar el detalle se llama a la rutina "detalle_de_separador" (Ver Figura # 17) que se explica a continuación:

Se le pide al usuario que defina el punto en el cual quiere que vaya el detalle del separador.

Se calcula una distancia (dis) que define la longitud del separador.

Se dibuja una línea horizontal Lh1 de longitud dist, que parte desde el punto.

Se dibuja Lh2 como una paralela a Lh1 a una distancia L.

Se dibuja Lh3 como una paralela a Lh1 a una distancia L – t.

Se dibuja Ldv como una paralela a Lh1 a una distancia dv.

Se dibuja Lph como una paralela a Lh1 a una distancia dv.

Se dibuja Lv1 desde los puntos de comienzo de Lh1 y Lh2.

Se dibuja Lv2 desde los puntos de terminación de Lh1 y Lh2.

Se dibuja LdB como una horizontal desde el punto, con una longitud dB.

Se mueve LdB desde el punto hasta el punto de comienzo de Lph.

Se dibuja LdP como una horizontal desde el punto, con una longitud dP.

Se mueve LdP desde el punto hasta el punto de terminación hasta el comienzo de LdB.

Se dibuja un perno con centro en el punto de inicio de LdP.

Se guardan en un vector p las coordenadas del perno.

Se mueve LdP desde su punto de inicio hasta su punto de terminación.

Esto se repite Np veces hasta que todos los pernos queden dibujados.

Se dimensionan todas las líneas dibujadas y las distancias entre los pernos.

Se hace un espejo de todas las líneas dibujadas según se van dibujando.

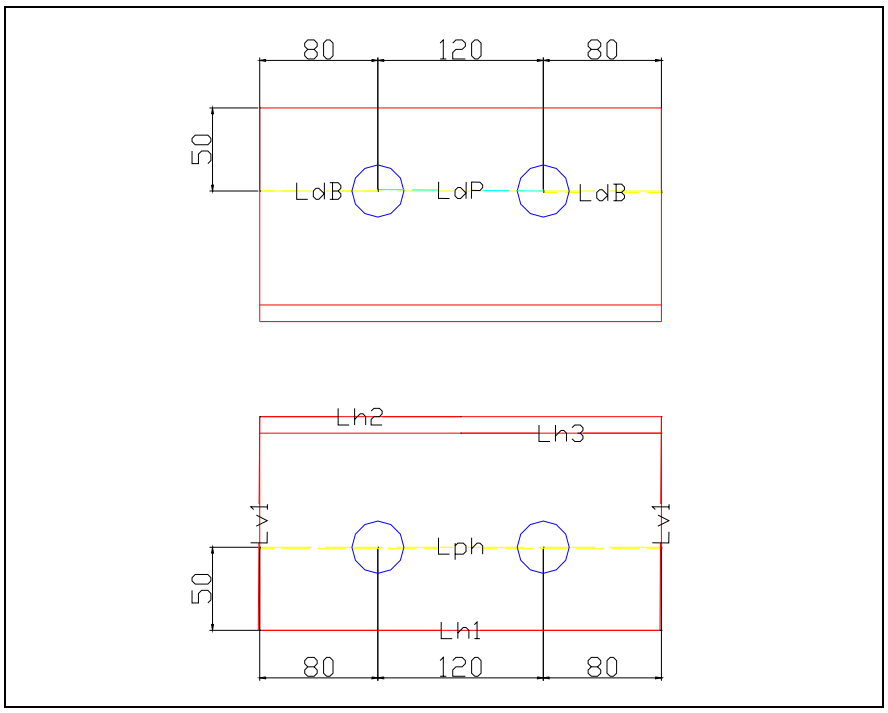


Figura #17. Detalle del Separador.

4.5.3- RUTINA "Separador de perfiles"

Esta rutina dibuja un separador ubicado en el medio de dos conexiones ya realizadas, en este tipo de separador los perfiles se encuentran separados y el separador que se va a dibujar los conecta.

A continuación se explica el funcionamiento de esta subrutina:

Se apagan todos las capas (layers) con la excepción de "Línea de Cg" y de "Pernos", esto se hace con la finalidad de restringir los objetos que el usuario pueda seleccionar en el plano.

Se le demanda al usuario que seleccione dos pernos (Perno1 y Perno2), a partir de los cuales se va a referencias el separador.

Se le pide al usuario que seleccione la línea (Línea) que define el perfil en el cual se va a encontrar el separador.

Se extraen los datos que definen las características del perfil, con la rutina "Saca_datos". Los datos que se van a extraer se encuentran en el archivo de datos adjuntos que AutoCAD tiene reservado para cada objeto del dibujo y que el usuario debe de haber definido previamente.

Se activan todas las capas (layers), con la finalidad de que se vean las líneas que definen el plano en pantalla.

Se verifica que el perfil en el cual se va a encontrar el separador sea un perfil doble, de no serlo el programa mostrará un cuadro de diálogo informándolo.

Al interesado se le pide que defina el punto (punto) en el cual desea pegar el detalle. Luego que defina si desea que los pernos horizontales y los verticales vayan intercalados.

Posteriormente, se le pide al usuario que defina la separación entre los perfiles (SP), además el número de pernos por ala (np), que va a tener dicho separador, el número debe ser un número par, de no serlo el programa lo informará.

Luego, se le pide al usuario que defina la distancia que separa a los pernos (dP). Conjuntamente que defina la distancia que separa al perno mas cercano del borde del separador con el borde del separador (db).

Con la rutina "arregloID", se acomoda Línea, de tal manera que el punto de inicio esté a la izquierda de la pantalla, esto se hace para poder referenciar con mayor facilidad el punto de inicio y terminación de la línea.

Se dibuja el perfil, con la rutina "dibuja_perfil_solo", esta rutina dibuja el perfil de la misma forma que "dibuja_perfil", pero se diferencian en que "dibuja_perfil_solo" no define cuales son L11, L13 y Lper1.

Se dibuja Lp1 como una línea comprendida entre los puntos de inicio de L11i y L11s. Se dibuja Lp2 como una línea comprendida entre los puntos de inicio de L11i y L11s. Se mueve Lp1 desde el punto de inicio de L11i hasta el centro de Perno1. Se mueve Lp2 desde el punto de inicio de L11i hasta el centro de Perno2. Se intercepta Lp1 con Línea obteniendo como resultado el punto P1. Se intercepta Lp2 con Línea obteniendo como resultado el punto P2.

Culminando, se borran Lp1 y Lp2. Se calcula el punto Pm, que será el punto medio entre P1 y P2. Pm define el punto central entre los pernos de referencia, punto que será el punto de referencia del separador. Ver Figura # 18.

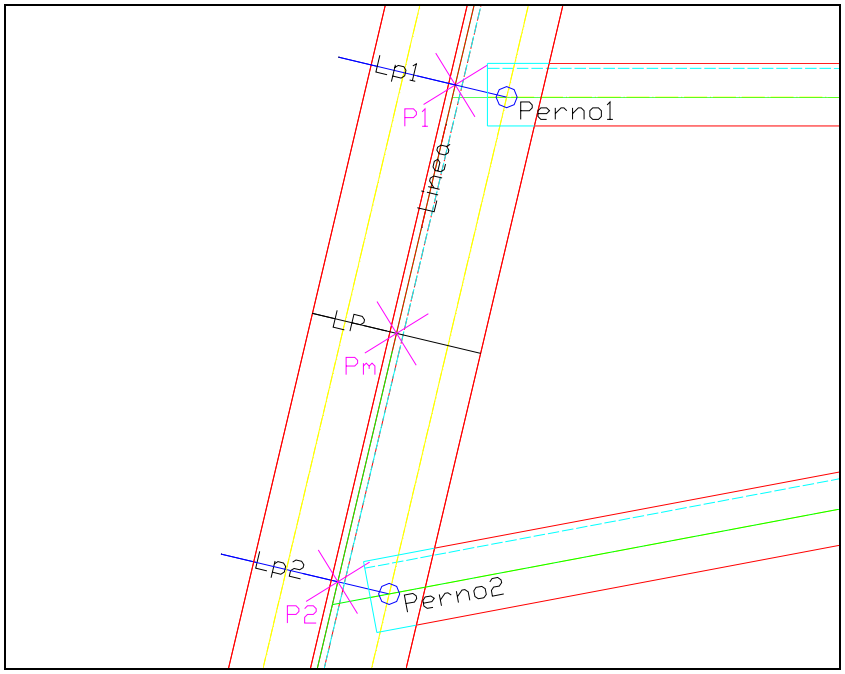


Figura # 18. Definición del Punto Medio entre dos Pernos

A continuación se describen una serie de acciones a tomar para la construcción de este tipo de separador (Ver Figura # 19):

Se dimensiona un vector LPer de 1 a 5 de Acadline.

Se dimensiona un vector pernoV de 0 a 2*np de Acadline

Se dimensiona un vector perno de 1 a 2*np de Acadline.

Se dibuja una línea vertical que tiene como punto medio punto (pernV(0)) de longitud 3*t.

De ir los pernos intercalados, se dibujará una línea horizontal LPer(1) en punto de longitud 4 * db + 2 * (np - 1) * dP + SP + dP.

De no ir los pernos intercalados, se dibujará una línea horizontal LPer(1) en punto de longitud 4 * db + 2 * (np - 1) * dP + SP .

Se dibuja una línea paralela LPer(2) a LPer(1) a una distancia L.

Se dibuja una línea paralela LPer(3) a LPer(1) a una distancia t.

Se dibuja una línea LPer(4) comprendida entre los puntos de inicio de LPer(1) y LPer(2).

Se dibuja una línea LPer(5) comprendida entre los puntos de terminación de LPer(1) y LPer(2).

Se dibuja una línea paralela LPernos a LPer(1) a una distancia L - X.

Se dibuja una línea horizontal LdB1 con una longitud dB a partir del punto.

Se mueve LdB1 desde su punto de inicio hasta el punto de terminación de Lpernos.

De ir los pernos intercalados, se dibujará una línea horizontal LdB2 de longitud db + dP / 2, en el punto.

De no ir los pernos intercalados, se dibujará una línea horizontal LPer(1) de longitud db, en el punto.

Se dibuja una línea horizontal LdP con una longitud dP ubicada en el punto.

Se dibuja un círculo perno(1) con centro en LdB1.

Se hace una copia PernoV(1) de PernoV(0) y se mueve desde el punto de inicio hasta el punto de terminación de LdB2.

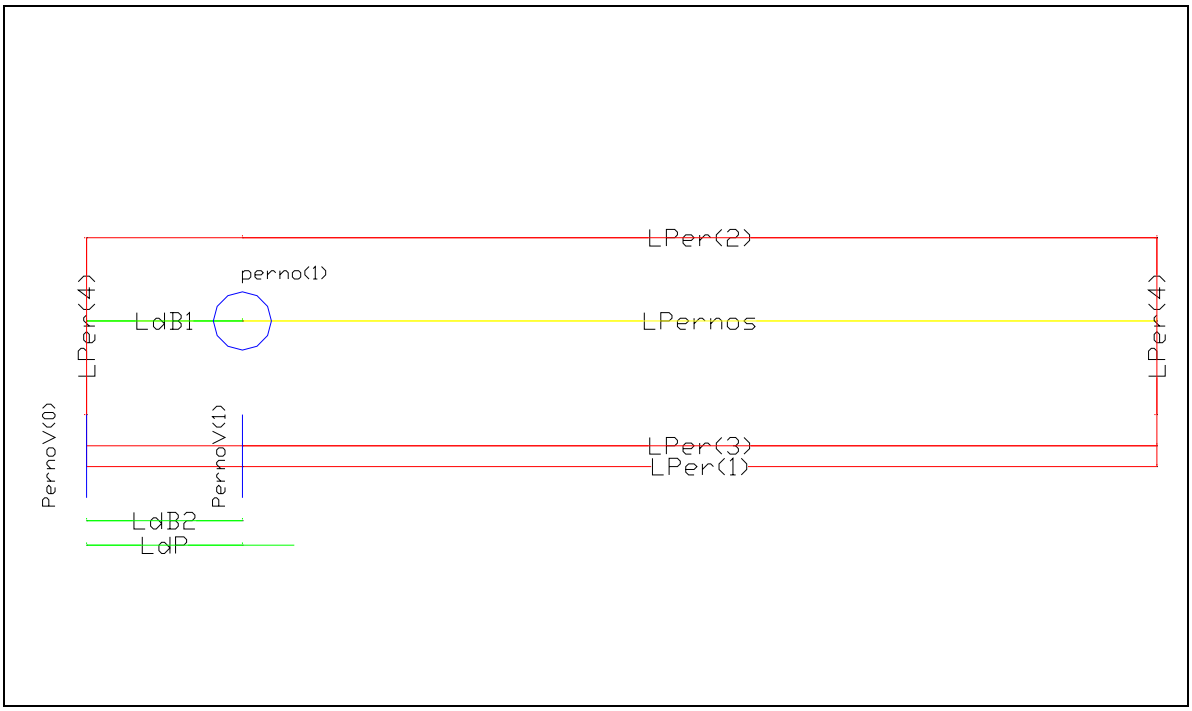


Figura # 19. Líneas para la Construcción del Detalle del Separador

Luego, se hace una copia perno(i+1) de perno(i) (partiendo de i=1) y se mueve desde el punto de inicio hasta el punto de terminación de LdP.

Se hace una copia PernoV(i+1) de PernoV(i) (partiendo de i=1) y se mueve desde el punto de inicio hasta el punto de terminación de LdP.

Se repiten estas dos últimas instrucciones np-1 veces.

De ir los pernos intercalados, se dibujará una línea horizontal Ln en punto de longitud (4 * db + 2 * (np - 1) * dP + dP) / 2 + SP). De no ir los pernos intercalados, se dibujará una línea horizontal Ln en punto de longitud (4 * db + 2 * (np - 1) * dP) / 2 + SP).

Por np veces se hace una copia perno(I+np) de perno (i) (partiendo de I=1) y se mueve desde el punto de inicio hasta el punto de terminación de Ln. Por np veces se hace una copia pernoV(I+np) de perno (i) (partiendo de I=1) y se mueve desde el punto de inicio hasta el punto de terminación de Ln.

Posteriormente, se hace una paralela Mirr de Lper(1) a una distancia –L.

Por 2*np veces se hace un espejo de pernoV(i) (partiendo de I=1) desde el punto de inicio hasta el punto de terminación de Mirr. Por 2*np veces se hace un espejo de perno(i) (partiendo de I=1) desde el punto de inicio hasta el punto de terminación de Mirr. Por 5 veces se hace un espejo de LPer(i) (partiendo de I=1) desde el punto de inicio hasta el punto de terminación de Mirr.

Subsecuentemente, se borran Ln, LdB1, LdB2, LdP, Mirr y se acotan las líneas horizontales, las verticales y los pernos. Observar Figura # 20.

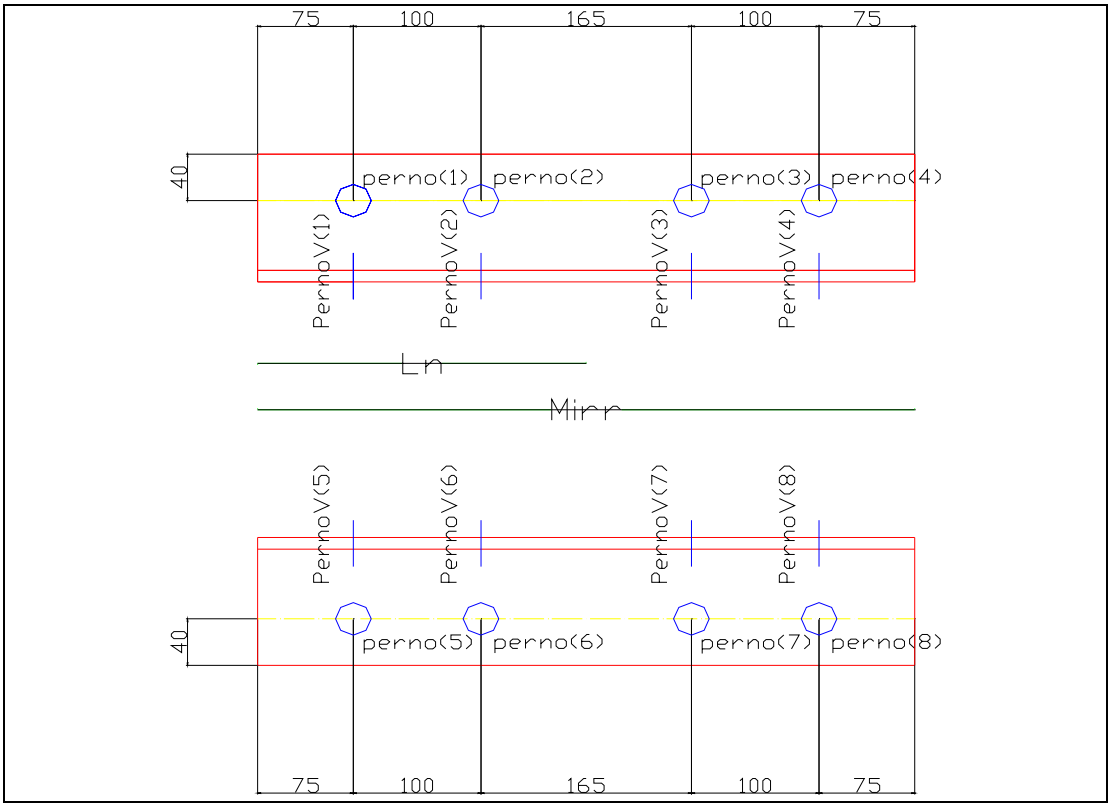


Figura # 20. Líneas para la Construcción del Detalle del Separador

De esta manera se termina el detalle del separador, a partir de aquí se empieza a dibujar el separador en el perfil. Se calcula el punto medio po de LPer(1).

Desde I = 1 hasta 2 * np veces se hace una copia de perno(i) que se mueve desde po hasta pm, se rota un ángulo igual al de Línea y se le hace un espejo desde el punto de inicio hasta el punto de terminación de Línea.

Desde I = 1 hasta 2 * np veces se hace una copia de pernoV(i) que se mueve desde po hasta pm, se rota un ángulo igual al de Línea y se le hace un espejo desde el punto de inicio hasta el punto de terminación de Línea.

Se traza Lpa como una línea comprendida entre los puntos de inicio de L11i y Línea.

Luego, se mueve Lpa desde el punto de inicio de Lpa hasta pm. Se dibuja Lpb como una línea comprendida entre los puntos de inicio de L11s y Línea. Se mueve Lpa desde el punto de inicio de Lpb hasta pm.

De no ir los pernos intercalados se dibuja una línea paralela Lp1 a Lpa a una distancia 2 * db + (SP / 2) + (np - 1) * dP.

De ir los pernos intercalados se dibuja una línea paralela Lp1 a Lpa a una distancia 2 * db +(SP / 2) + (np - 1) * dP +(dP / 2).

De allí se hace un espejo de Lp1 desde el punto de inicio hasta el punto de terminación de Lpa.

De no ir los pernos intercalados se dibuja una línea paralela Lp2 a Lpb a una distancia 2 * db +(SP / 2) + (np - 1) * dP.

De ir los pernos intercalados se dibuja una línea paralela Lp2 a Lpb a una distancia 2 * db +(SP / 2) + (np - 1) * dP + (dP / 2).

Y se hace un espejo de Lp2 desde el punto de inicio hasta el punto de terminación de Lpb. Se esboza una línea paralela a Lpa a una distancia SP/2. Luego, se hace un espejo de esa línea desde el punto de inicio hasta el punto de terminación de Lpa.

Se dibuja una línea paralela a Lpb a una distancia SP/2. Se hace un espejo de esa línea desde el punto de inicio hasta el punto de terminación de Lpb. Y se borran Lpa, Lpb, L11i, L11i, L13s, L13s, LPer1i y LPer1s.

Véase Figura # 21.

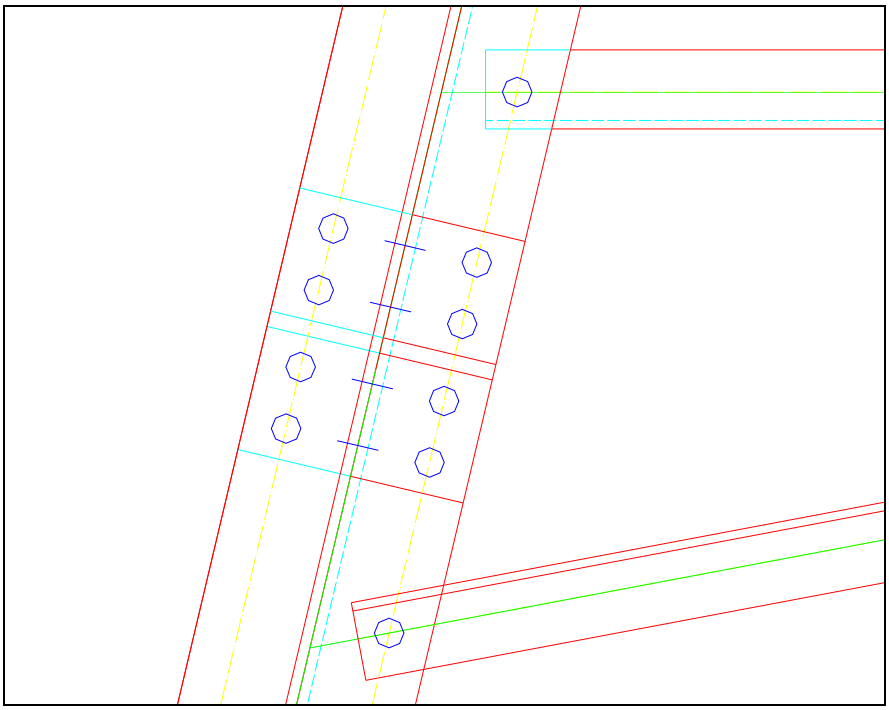


Figura # 21. Separador de Perfiles

4.5.4- RUTINA "Separador Centrado"

Para utilizar esta instrucción es necesario haber realizado las distintas conexiones del plano**.** Esta rutina dibuja un separador ubicado en el medio de dos conexiones ya realizadas, en este tipo de separador los perfiles no se encuentran separados.

A continuación se explica el funcionamiento de esta subrutina:

Se apagan todas las capas (layers) con la excepción de "Línea de Cg" y de "Pernos",, esto se hace con la finalidad de restringir los objetos que el usuario pueda seleccionar en el plano. Luego se le pide al usuario que seleccione dos pernos (Perno1 y Perno2), a partir de los cuales se va a referenciar el separador.

Posteriormente, se le demanda al usuario que seleccione la línea (Línea) que define el perfil en el cual va a estar el separador. Con la rutina "Saca_datos" se extraen los datos que definen las características del perfil. Los datos que se van a extraer se encuentran en el archivo de datos adjuntos que AutoCAD tiene reservado para cada objeto del dibujo y que el usuario debe de haber definido previamente.

Se activan todas las capas (layers). Con la finalidad de que se vean las líneas que definen el plano en pantalla. Se verifica que el perfil en el cual se va a encontrar el separador sea un perfil doble, de no serlo el programa mostrará un cuadro de diálogo informándolo.

Luego, se le pide al usuario que defina el número de pernos por ala (np), que va a tener dicho separador, que defina la distancia que separa a los pernos (dP), además de introducir la distancia que separa al perno mas cercano del borde del separador con el borde del separador (db).

De allí, se acomoda Línea, con la rutina "arregloID", de tal manera que el punto de inicio esté a la izquierda de la pantalla, esto se hace para poder referenciar con mayor facilidad el punto de inicio y terminación de la línea.

Se dibuja el perfil, con la rutina "dibuja_perfil_solo", esta rutina dibuja el perfil de la misma forma que "dibuja_perfil", pero se diferencian en que "dibuja_perfil_solo" no define cuales son L11, L13 y Lper1.

Se traza Lp1 como una línea comprendida entre los puntos de inicio de L11i y L11s.

Se delinea Lp2 como una línea comprendida entre los puntos de inicio de L11i y L11s.

Consecutivamente, se realizan los pasos que describen a continuación (Ver Figura # 22):

Se mueve Lp1 desde el punto de inicio de L11i hasta el centro de Perno1.

Se mueve Lp2 desde el punto de inicio de L11i hasta el centro de Perno2.

Se intercepta Lp1 con Línea obteniendo como resultado el punto P1.

Se intercepta Lp2 con Línea obteniendo como resultado el punto P2.

Se calcula el punto Pm, que será el punto medio entre P1 y P2.

Se borran Lp1 y Lp2.

Pm define el punto central entre los pernos de referencia, punto que será el punto de referencia del separador.

Se dibuja LP como una línea comprendida entre los puntos de inicio de L11i y L11s.

Se mueve LP desde el punto de inicio de Línea hasta Pm.

Se acomoda LP, de tal manera que el punto de inicio esté a la izquierda de la pantalla, esto se hace para poder referenciar con mayor facilidad el punto de inicio y terminación de la línea. Con la rutina "arregloID".

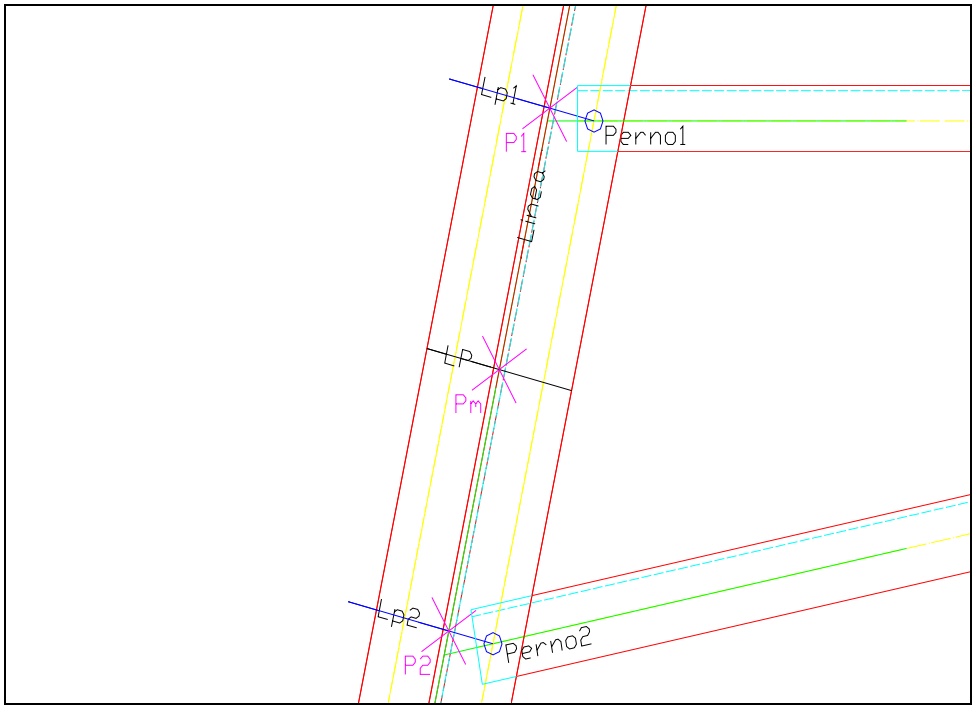


Figura # 22. Definición del Punto Medio entre dos Pernos

Se intercepta Lp con Lper1i, obteniendo como resultado el punto P1.

De ser np un número par entonces:

- Se dibuja una línea horizontal LdB en el origen del plano con una longitud dP / 2.
- Se mueve LdB desde el origen el plano hasta P1.
- Se rota LdB un ángulo igual al de Línea.
- Se dibuja Perno1 en el punto de terminación de LdB.
- Se hace un espejo de Perno1, con respecto a Línea llamado Perno2.
- Se hace un espejo de Perno1 y Perno2 con respecto a Lp.
- Se dibuja una línea horizontal LdP en el origen del plano con una longitud dP.
- Se mueve LdP desde el origen el plano hasta el punto de terminación de LdB.
- Se borra LdB.
- Se rota LdP un ángulo igual al de Línea.
- Se calcula n como n = (np / 2) - 1
- Se calcula K como (dP / 2) + (n * dP) + db

De ser np un número impar entonces:

- Se dibuja Perno1 en P1.

- Se hace un espejo de Perno1, con respecto a Línea llamado Perno2.
- Se hace un espejo de Perno1 y Perno2 con respecto a Lp.
- Se dibuja una línea horizontal LdP en el origen del plano con una longitud dP.
- Se mueve LdP desde el origen el plano hasta el punto de terminación de P1.
- Se rota LdP un ángulo igual al de Línea.
- Se calcula n como la parte entera de (np / 2) - 1
- Se calcula K como (n * dP) + db

Se continuará con los siguientes pasos:

Se dibuja Perno1 en el punto de terminación de LdP.

Se hace un espejo de Perno1, con respecto a Línea llamado Perno2.

Se hace un espejo de Perno1 y Perno2 con respecto a Lp.

Se mueve LdP desde su punto de comienzo hasta su punto de terminación.

Esto se repite n veces, con la finalidad de que se dibujen todos los pernos que definen al separador. (Ver Figura # 23)

Se continúa con el esbozo de Lpa como una línea comprendida entre los puntos de inicio de L11i y Línea. Se intercepta LPer1i con Lpa, obteniendo como resultado el punto P1. Se intercepta LP con LPer1i, obteniendo como resultado el punto P2.

Luego, se mueve Lpa desde P1 hasta P2. Se dibuja Lpb como una línea comprendida entre los puntos de inicio de L11s y Línea. Se intercepta LPer1s con Lpb, obteniendo como resultado el punto P1. Se intercepta LP con LPer1s, obteniendo como resultado el punto P2 y se mueve Lpb desde P1 hasta P2.

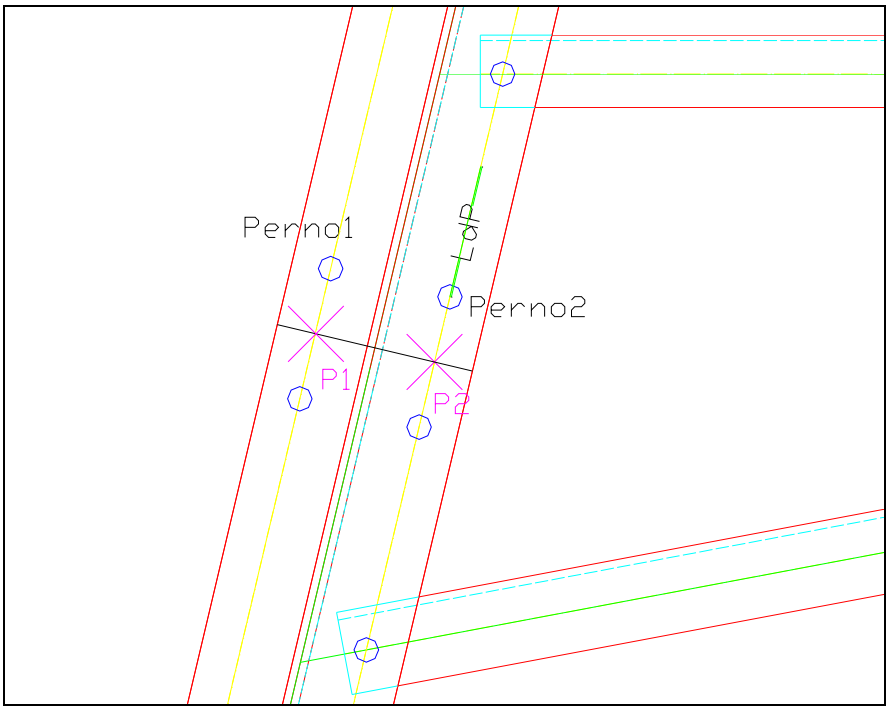


Figura # 23. Líneas a Realizar para la Construcción del Separador

Dependiendo del tipo de vista se le asignará un color rojo o cyan a Lpa y Lpb. Véase Figura # 24.

Se dibuja una línea Lp1 paralela a Lpa a una distancia k.

Se hace un espejo de Lp1 con respecto a LP.

Se dibuja una línea Lp1 paralela a Lpb a una distancia k.

Se hace un espejo de Lp1 con respecto a LP.

Posteriormente, se borran Lpa, Lpb, LP, LdB, LdP, L11i, L11s, L13i, L13s, Lper1s y LPer1i. Obsérvese Figura #25.

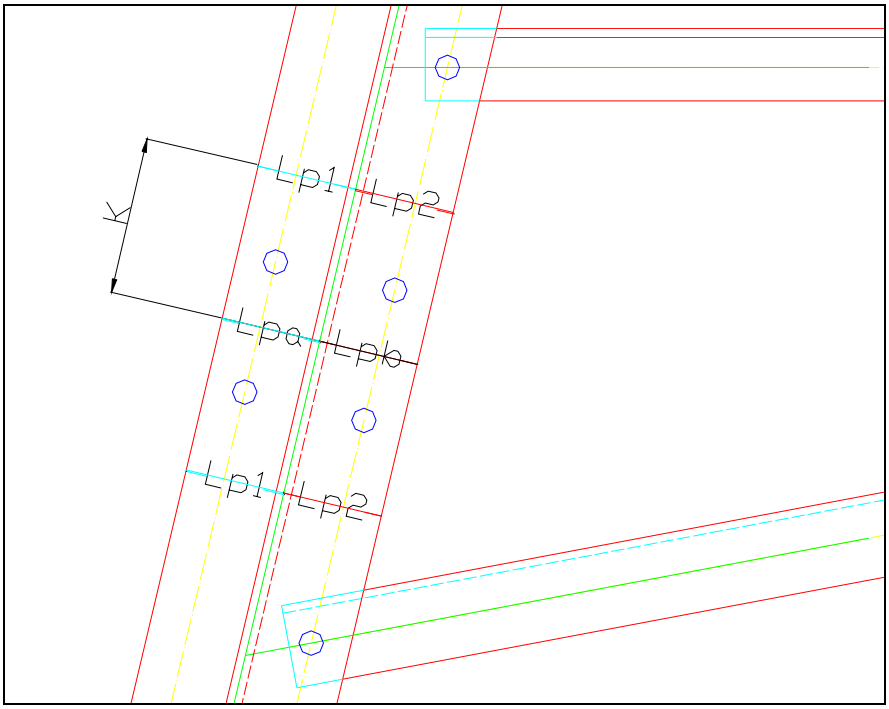


Figura # 24. Líneas a Realizar para la Construcción del Separador

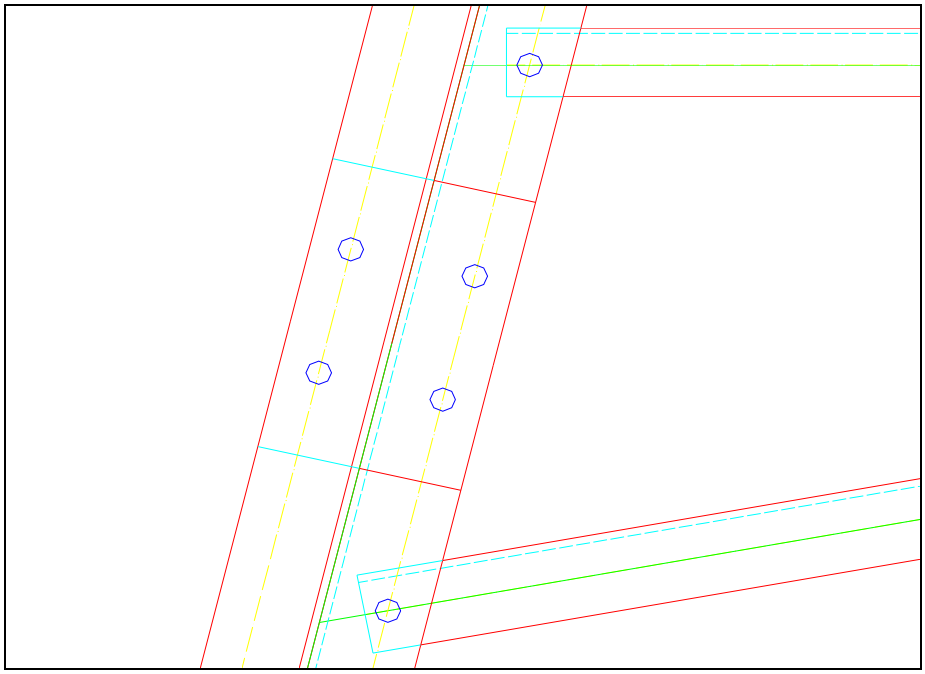


Figura # 25. Separador Centrado

Para culminar, se llama a la rutina "detalle_de_separador" que se explicó en la rutina "Separador a partir de un perno". Detalle la Figura # 26.

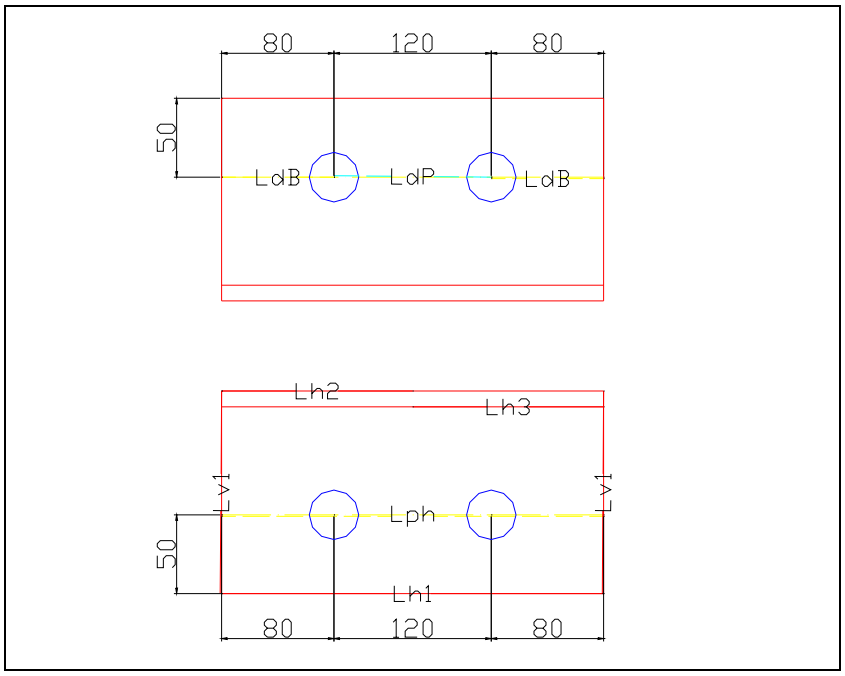


Figura # 26. Detalle del Separador Centrado

4.6- RUTINA "Planchas"

Esta rutina hace todos cortes y cálculos necesarios para que se dibuje alrededor de los pernos seleccionados una plancha que cumple con los requerimientos del usuario. Para utilizar esta instrucción es necesario haber definido anteriormente las conexiones que definen a los perfiles que forman parte de la plancha, ya sea con la rutina "Union_de_perfiles_con_un_perno" o con la rutina "Union_para_planchas".

A continuación se explica el funcionamiento de esta rutina:

Se muestra un cuadro de diálogo informando que los pernos a escoger deben ser escogidos en forma antihoraria.

Se le pide al usuario que introduzca los pernos que definen la forma geométrica de la plancha.

Se le asigna a la variable n el número de pernos escogidos.

En caso de n ser menor que 3 se sale de la rutina. Esto debido a que una plancha debe de estar definida por lo menos por 3 pernos.

Se le pide al usuario la distancia entre el perno mas próximo al borde y el borde (dB) de la plancha, además que defina si las esquinas de la plancha deben ir picadas.

Se dimensiona un vector p, el cual contendrá las coordenadas de los pernos que definen la geometría de la plancha.

Se dibuja una polilínea cerrada líneas definida por todas las coordenadas del vector p.

De allí se siguen una serie de pasos:

Se le hace una paralela L a una distancia dB a líneas.

Se define un vector pn, que contendrá las coordenadas que definen la polilínea L.

Se borra la polilínea cerrada líneas.

En caso de no querer las esquinas picadas se explota L y se terminará la rutina.

Obsérvese la Figura # 27.

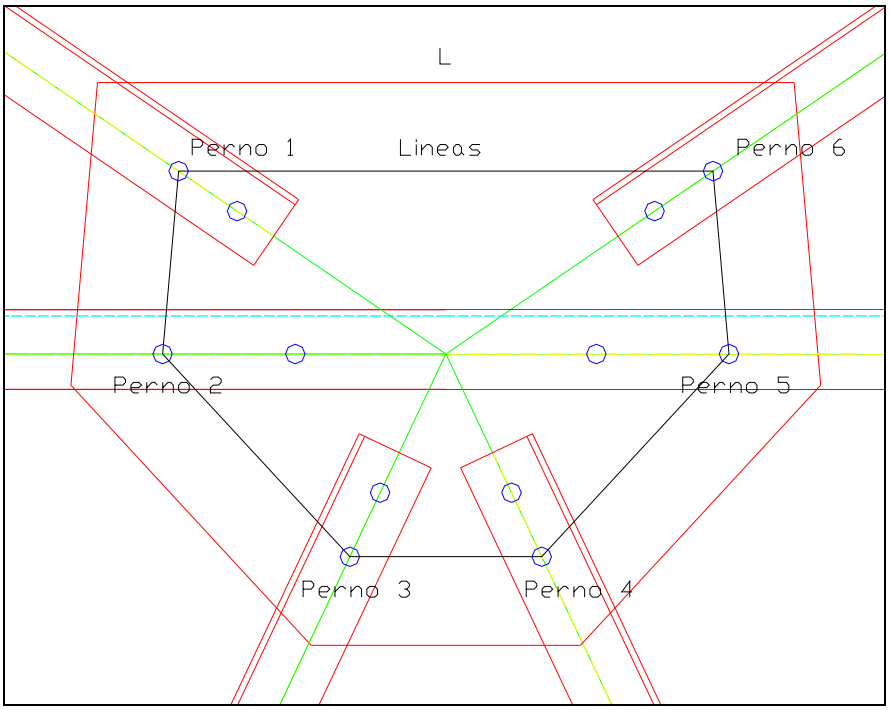


Figura # 27. Líneas Trazadas para la Construcción de la Plancha

En caso de requerir las esquinas picadas se seguirán los siguientes pasos por cada una de las esquinas de la plancha:

Se define pa como las coordenadas del p que definen una esquina.

Se define pb como las coordenadas del pn que definen una esquina.

Se dibuja una línea Aux desde pa hasta pb.

Se dibuja una línea horizontal LdN en el origen del plano de longitud dB.

Se mueve LdN desde el punto de origen del plano hasta pa.

Se rota LdN un ángulo igual al de Aux.

Se dibuja una línea horizontal LdC en el origen del plano de longitud dB.

Se mueve LdC desde el punto de origen del plano hasta el punto de terminación de LdN.

Se rota LdC un ángulo igual al de Aux

Se intercepta LdC con L extendiendo LdC.

Si el caso es que no se interceptan o de que se intercepten en un solo punto, se llena un vector ldef con las coordenadas de pn que definen esa esquina.

Si el caso es que se interceptan en 2 puntos, se llenará el vector ldef con las coordenadas de los 2 puntos, ordenados en forma antihoraria. Véase la Figura # 28.

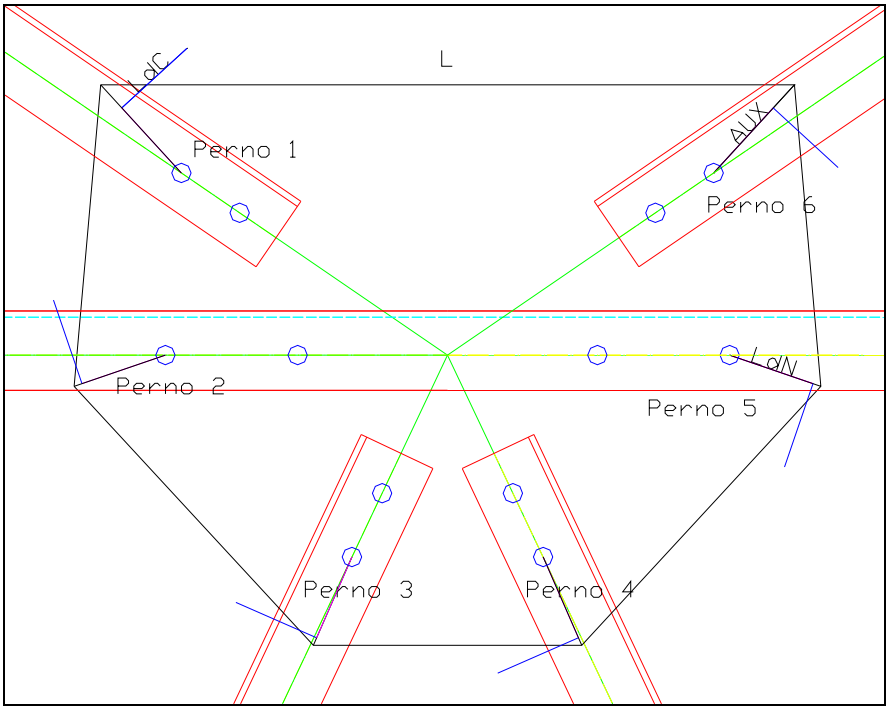


Figura # 28. Construcción de la Plancha

Una vez llenado el vector con todas las coordenadas de las esquinas, se dibujará una poli línea líneas con el vector ldef. Luego, se borra L y se explota líneas. Ver Figura # 29.

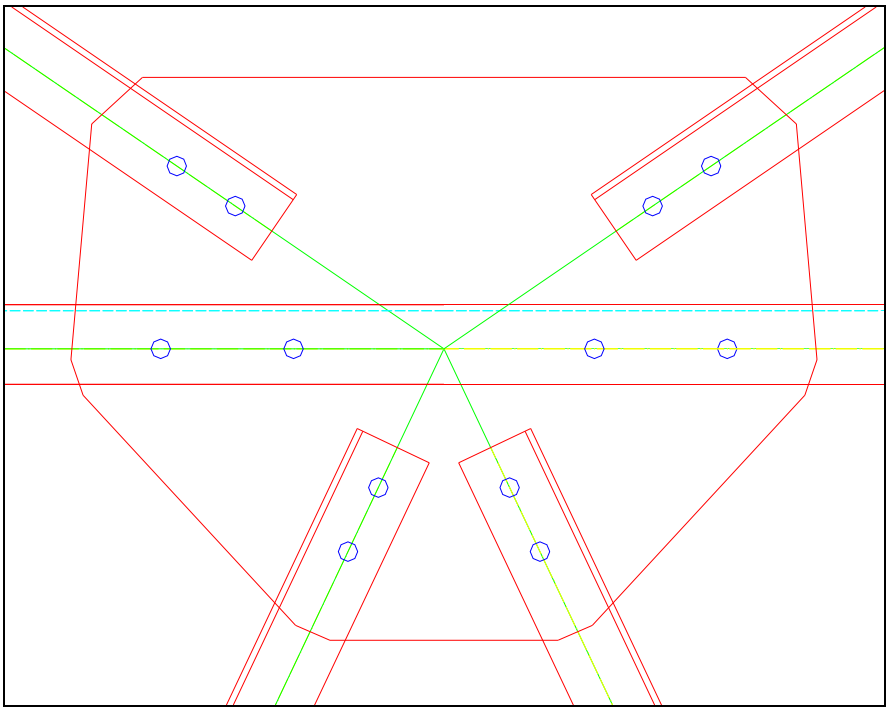


Figura # 29. Dibujo de la Plancha

4.7- RUTINA "Dimensionar"

Para utilizar esta instrucción se recomienda haber terminado el plano. Esta subrutina asiste al usuario para acotar las distancias existentes entre los bordes de los perfiles y los pernos de manera automática.

A continuación se explica el funcionamiento de esta subrutina:

Se apagan todas las capas (layers) con la excepción de "Línea de Pernos". Luego, se le pide al usuario la línea de pernos Línea que desea acotar.

Después, se activan todas las capas (layers). Con la finalidad de que se vean las líneas que definen el plano en pantalla. Posterior a eso, se le demanda al usuario que defina el punto en el cual desea que vaya el acotamiento.

Con la rutina "Limpia_pernos", se verifica de que ningún perno se encuentre dibujado 2 veces en el plano, esto se hace verificando de que los objetos de layer "Pernos" en el dibujo no tengan el mismo centro y radio.

Con la rutina "Limpia_pernos", se verifica que la línea de pernos seleccionada no se encuentre dibujado 2 veces en el plano, esto se hace comprobando de que ningún objeto de layer "Línea de Pernos" tenga el mismo punto de inicio y de terminación.

Se acomoda Línea, de tal manera que el punto de inicio esté a la izquierda de la pantalla, esto se hace para poder referenciar con mayor facilidad el punto de inicio y terminación de la línea, con la rutina "arregloID".

Se hace una copia Linead de Línea.

Se mueve desde su punto de inicio hasta punto.

Con la rutina "Calcula_distancia_perpendicular_entre_punto_y_linea2", se rota Linead un ángulo de - 90 grados.

Luego, se intercepta Linead con Línea extendiendo las 2 líneas, obteniendo como resultado el punto pv. Se calcula la distancia d entre punto y pv, esta distancia será la perpendicular entre punto y Línea.

Enseguida se dibuja una línea paralela Laux a Línea, a una distancia d, en la dirección de punto.

Se traza una línea Lp desde el punto de inicio de Línea hasta el punto de inicio de Laux. Y se ordenan los bordes del perfil y los centros de los pernos contenidos en Línea de izquierda a derecha, con la rutina "ordena_puntos"

Acto seguido, se hace el acotamiento entre todos los puntos contenidos en Línea, utilizando como punto de referencia el punto de inicio de Lp, que se moverá en cada acotamiento desde su punto de inicio hasta el punto medio de los dos puntos que definen el acotamiento. Observar Figura # 30.

A continuación, se borran Lp y Laux (Ver Figura # 31).

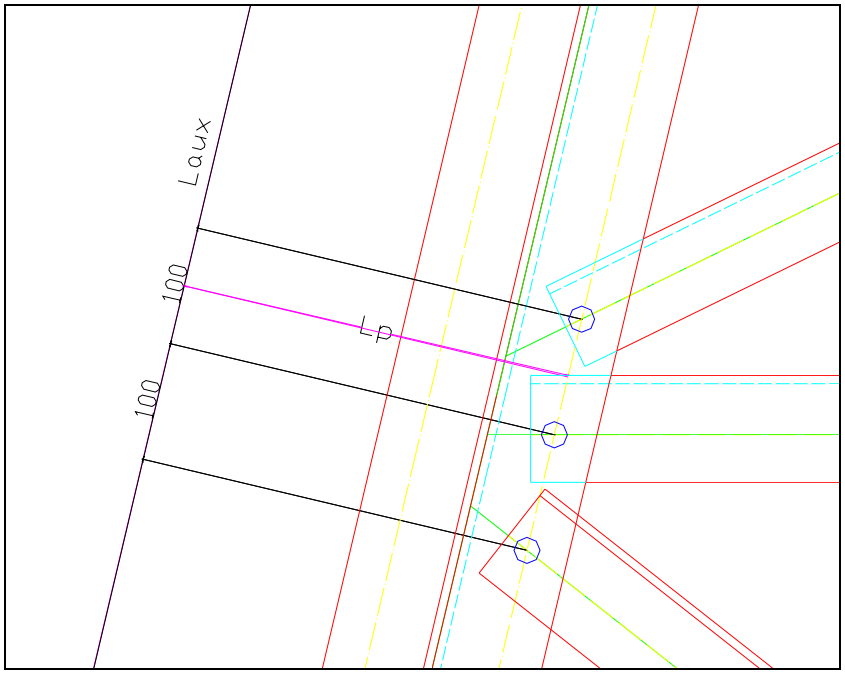


Figura # 30. Líneas Trazadas para la Realización del Acotamiento

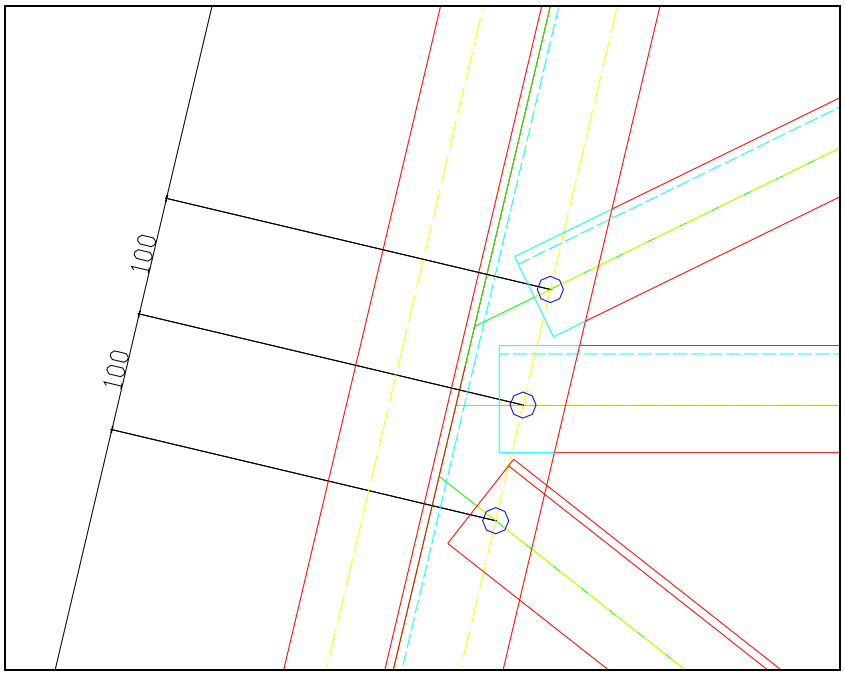


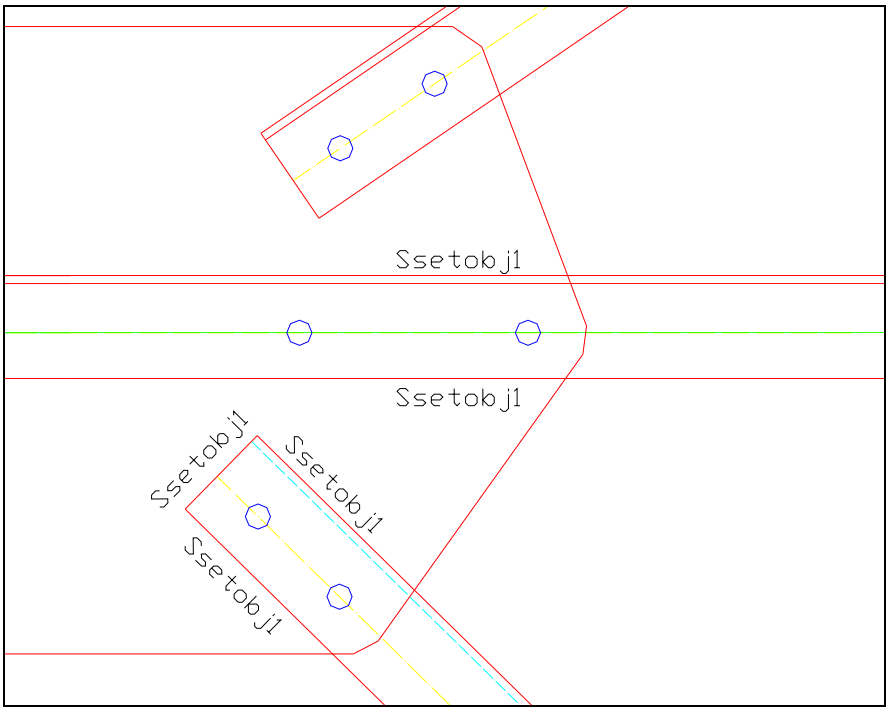
Figura # 31. Acotamiento del Perfil

4.8- RUTINA "Visibilidad"

Esta rutina permite realizar la visibilidad de una manera manual de los perfiles que forman una conexión. Para el uso de esta instrucción es necesario haber realizado anteriormente la conexión o la plancha.

A continuación se explica el funcionamiento de esta rutina:

Al comenzar, se le pide al usuario que seleccione las líneas que definen el perfil, éstas serán las que representan las líneas externas del perfil y que se encontrarán en la AcadSelectionSet Ssetobj1. En la Figura #32 se muestra el ejemplo de las líneas pertenecientes a ssetobj1 para dos casos distintos de visibilidad.



Figuras # 32. Definición de las Líneas del Perfil

Luego, se le pide al usuario que seleccione las líneas que definen la plancha, estas líneas serán La, Lb y Lc, que vienen formadas por las líneas que cortan, que se encuentran en la parte interna del perfil y que se encontrarán en una AcadSelectionSet Ssetobj2. En la Figura #33 se muestra el ejemplo de las líneas pertenecientes a Ssetobj2 para dos casos distintos de visibilidad.

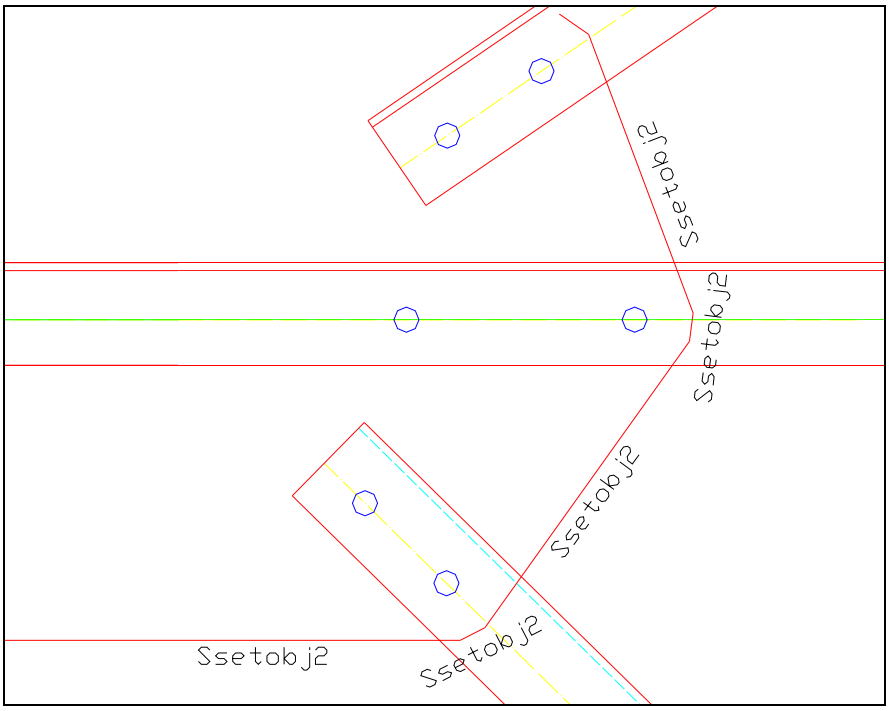


Figura # 33. Definición de las Líneas de la Plancha

Subsecuentemente, se le pide al usuario que defina cuales son las líneas que no se deben ver en el plano (si son las del perfil o las de la plancha), esto se hará seleccionando cualquiera de las líneas seleccionadas anteriormente.

En el caso de que las líneas que no se vean sean las de la plancha se le asignará a Ssetobj1 el contenido de ssetobj2 y a Ssetobj2 el contenido de Ssetobj1.

Se llama a la rutina "visivilidad_lista".

Se intercepta una de las líneas L1 de Ssetobj1 con cada una de las líneas de Ssetobj2 (sin extender ninguna).

En el caso de que no existan puntos de intercepción se le asignará a L1 el color Cyan.

En el caso de que exista un solo punto de intercepción se verificará que L1 tenga el mismo punto de inicio o terminación que las líneas de Ssetobj1, en el caso de tenerlo se creará una línea L que partirá desde el punto de inicio de L1 y terminará en pn, se le asignará a L el color Cyan, se creará otra línea L que partirá desde pn y llegará al punto de terminación de L1, y se le asignará el color rojo.

En el caso de que exista un solo punto de intercepción se verificará que L1 tenga el mismo punto de inicio o terminación que las líneas de Ssetobj1, en el caso de que no se cumpla la condición anterior, se calcula la distancia d1 comprendida entre el punto de inicio de L1 y pn y la distancia d2 entre pn y el punto de terminación de L1, se creará una línea L comprendida entre el punto de inicio de L1 y pn, y se le asignará color rojo, se creará otra línea L comprendida entre el punto de inicio de L1 y pn, y también se le asignará el color rojo, se creará una línea L adicional comprendida entre pn y el punto de terminación de L1, se le asignará nuevamente el color rojo.

En el caso de que existan 2 puntos de intercepción pa y pb, se calculará d1 que es la distancia comprendida entre el punto de inicio de L1 y pa, se calculará d2 que es la distancia comprendida entre pb y pa, se calculará d3 que es la distancia comprendida entre el punto de terminación de L1 y pb, en el caso de que d1 + d2 + d3 sean iguales a la longitud de L1 se creará una línea L de color rojo comprendida entre el punto de inicio de L1 y pa, se creará otra línea comprendida entre pa y pb de color Cyan y se creará una línea adicional de color rojo comprendida entre pa y el punto de terminación de L1. En el caso contrario se creará una línea L comprendida entre el punto de inicio de L1 y pb de color rojo, se creará una línea L comprendida entre pb y pa de color Cyan y se creará otra comprendida entre pb y el punto de terminación de L1 de color rojo.

Se repiten estos pasos con cada una de las líneas pertenecientes a L1.

Se borran Ssetobj1 y Ssetobj2.

Observar Figura # 34.

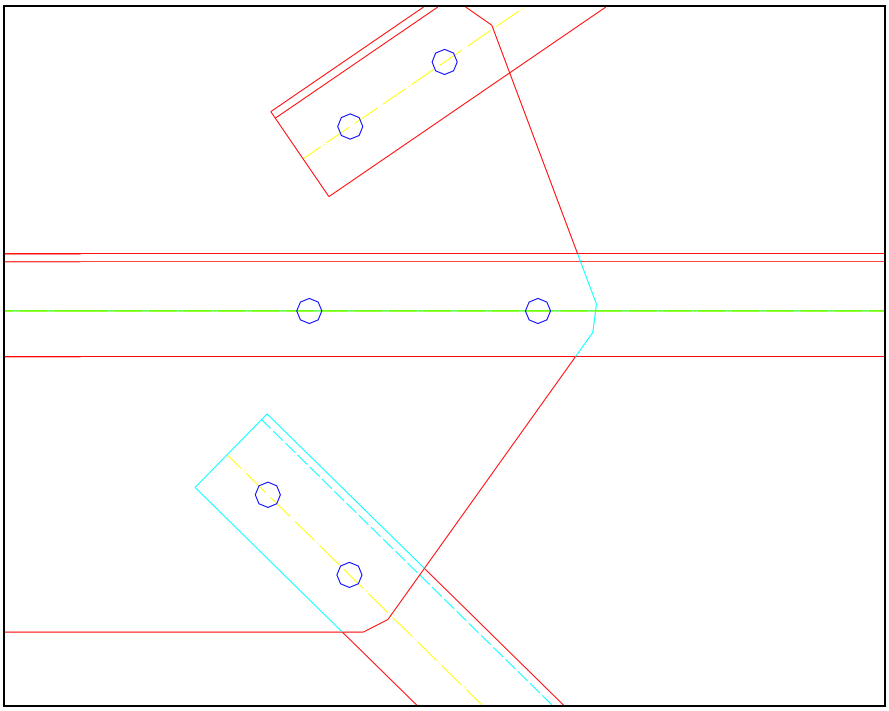


Figura # 34. Visibilidad

4.9- RUTINA "Detalle Cruz"

Esta rutina realiza los cortes necesarios para realización de una vista longitudinal de los perfiles. Para utilizar esta instrucción es necesario haber definido anteriormente las características de los perfiles con la rutina "Meter Datos".

A continuación se explica el funcionamiento de esta rutina:

Primero, se apagan todos las capas (layers) con la excepción de la capa "Línea de Cg", la capa "Línea de Cg" es la que contiene las líneas que definen la

geometría y las características que definen a cada uno de los perfiles de la estructura, esto se hace con la finalidad de restringir los objetos que el usuario pueda seleccionar en el plano.

Empezando, se le pide al usuario que seleccione la línea Línea que define al perfil.

Luego, se activan todas las capas (layers). Con la finalidad de que se vean las líneas que definen el plano en pantalla. Se extraen los datos que definen las características de cada uno de los perfiles, con la rutina "Saca_datos". Los datos que se van a extraer se encuentran en el archivo de datos adjuntos que AutoCAD tiene reservado para cada objeto del dibujo y que el usuario debe de haber definido previamente.

Posterior a eso, se le pide al usuario que seleccione el punto p en el cual quiere pegar el detalle. Se define un vector de líneas Ln.

Subsecuentemente, se realizan los siguientes pasos (Figura # 35):

- Se dibuja una línea vertical Ln(0), desde p con longitud L.
- Se dibuja una línea horizontal Ln(1), desde el punto de terminación de Ln(0) con longitud t.
- Se dibuja una línea vertical Ln(2), desde el punto de terminación de Ln(1) con longitud L -t.
- Se dibuja una línea horizontal Ln(3), desde el punto de terminación de Ln(2) con longitud L -t.
- Se dibuja una línea vertical Ln(4), desde el punto de terminación de Ln(3) con longitud t.
- Se dibuja una línea horizontal Ln(5), desde el punto de terminación de Ln(4) con longitud -L
- Se dibuja una línea vertical perno de una longitud (2 * t) a una distancia L - db de L(0).
- Se hace un relleno en el espacio definió por el vector de líneas Ln.

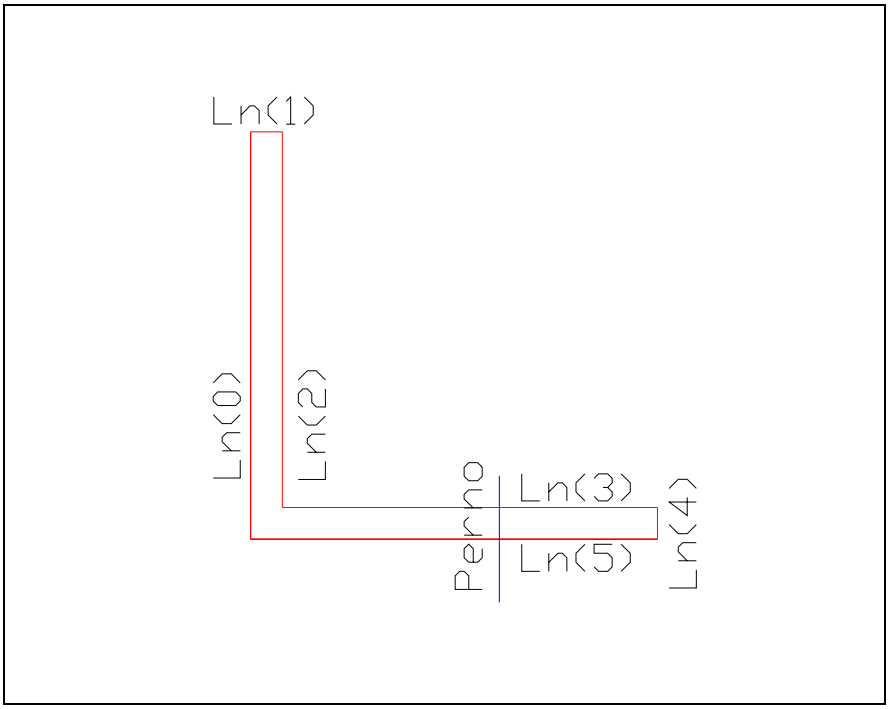


Figura # 35. Líneas Trazadas para la Construcción del Detalle

Para culminar, se llevan a cabo los siguientes pasos (Figura # 36):

- Se hace una copia de Ln y pernos que se rota 90º.
- Se hace una copia de Ln, pernos y del relleno que se rota 180º.
- Se hace una copia de Ln y de pernos que se rota 270º.

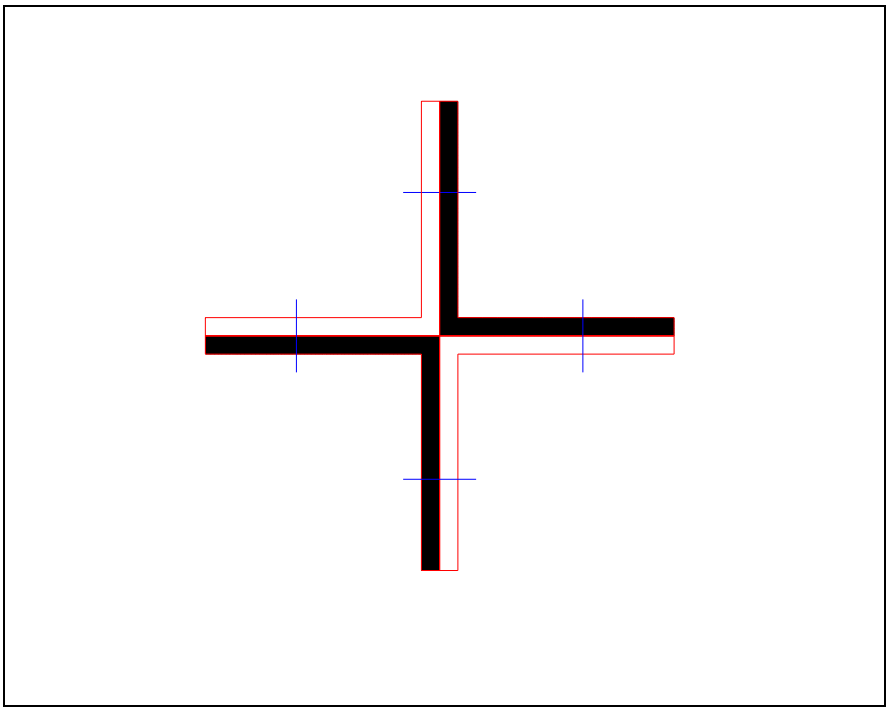


Figura # 36. Detalle Cruz

5.- MANUAL DEL USUARIO

5.- MANUAL DEL USUARIO

Para llevar a cabo la instalación del programa es necesario llamar al archivo ejecutable "Setup.exe", el cual abrirá el AutoCAD y automáticamente instalará la barra de herramientas "Perfiles de Acero". En el caso de que la instalación no finalice de una manera exitosa, el usuario podrá instalar la barra de herramientas de una manera manual siguiendo las siguientes instrucciones:

1. Crear una carpeta en C:\Archivo de programas\Acero.

2. Copiar todos los archivos desde el disco de instalación hasta C:\Archivo de programas\Acero.

3. Abrir AutoCAD 2000.

4. Una vez en AutoCAD 2000 llamar a la siguiente instrucción Tool>Load Aplication>Contents>Adds.

5. Seleccionar C:\Archivo de programas\Acero\ Acero.dvb y añadirlo.

6. Llamar a la instrucción Tool>Macros>Macros y correr la rutina C:\Archivo de programas\Acero\Barra_de_Herramienta, instrucción con la cual se creará la barra de herramientas y con la cual finalizará la instalación del programa. Ver Figura # 37.

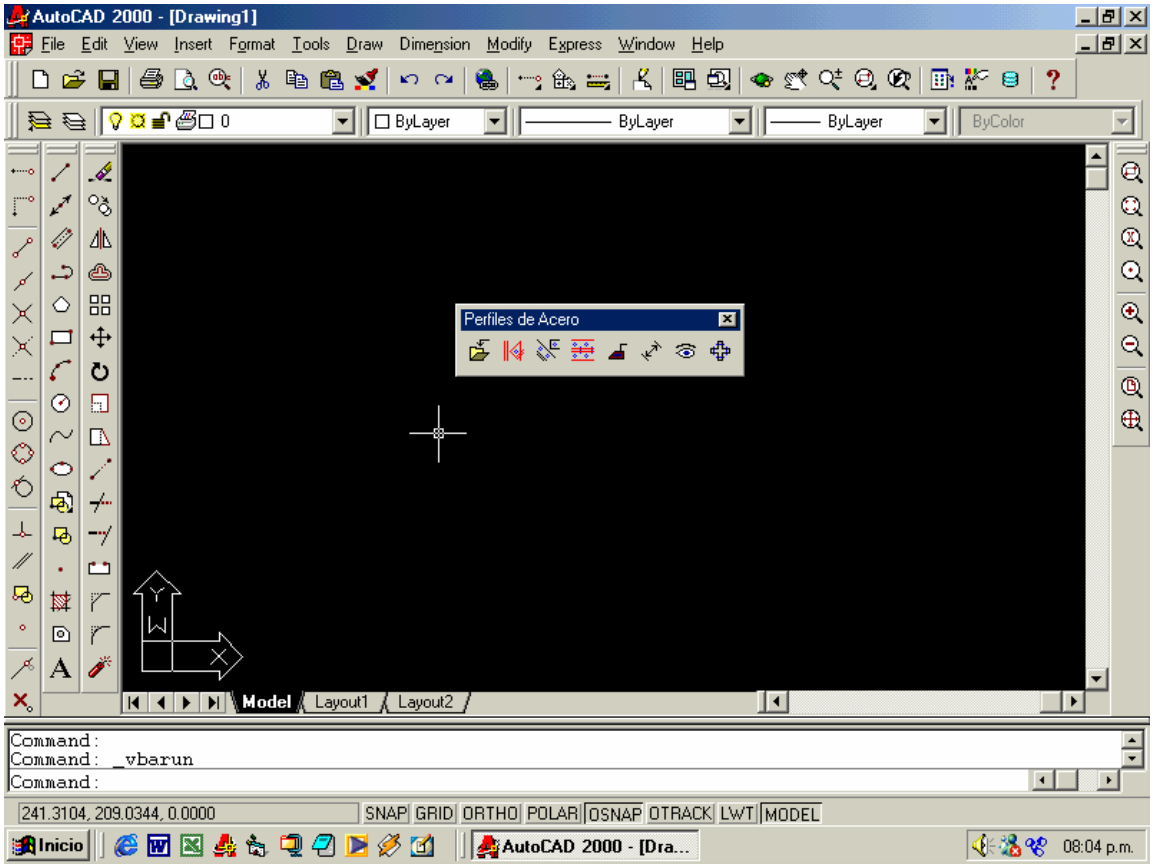


Figura # 37. Barra de Herramientas.

A continuación se muestran las instrucciones a seguir para el correcto funcionamiento de cada una de las rutinas que comforman la barra de herramientas "Perfiles de Acero" .

5.2- Meter Datos:

1- Se deben definir las características del Perfil llenando la forma "Introducción de Datos".

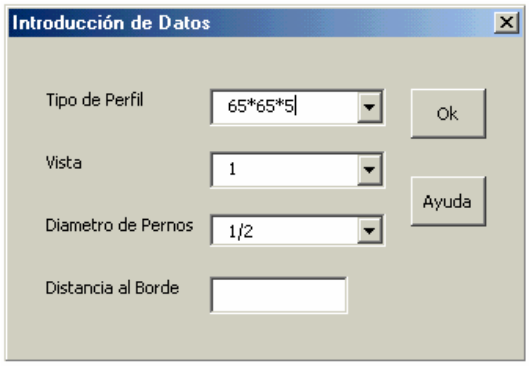


Figura # 38. Introducción de Datos

2- Al apretar el botón de Ayuda, se mostrará una forma que explica los distintos tipos de vistas.

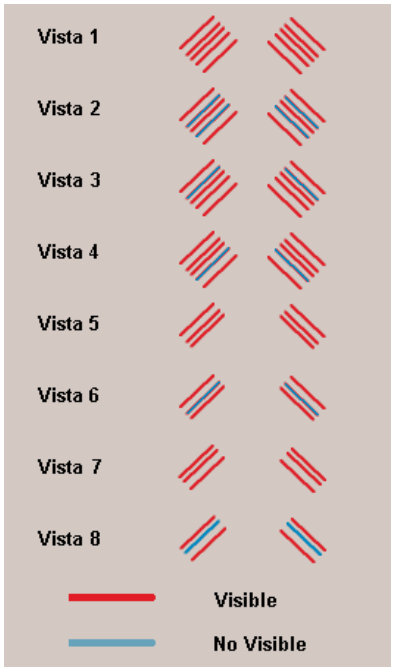


Figura # 39. Distintos Tipos de Vistas

3- La distancia al borde (dB) será la comprendida entre la línea de pernos (LPer) y el comienzo del ala del perfil.

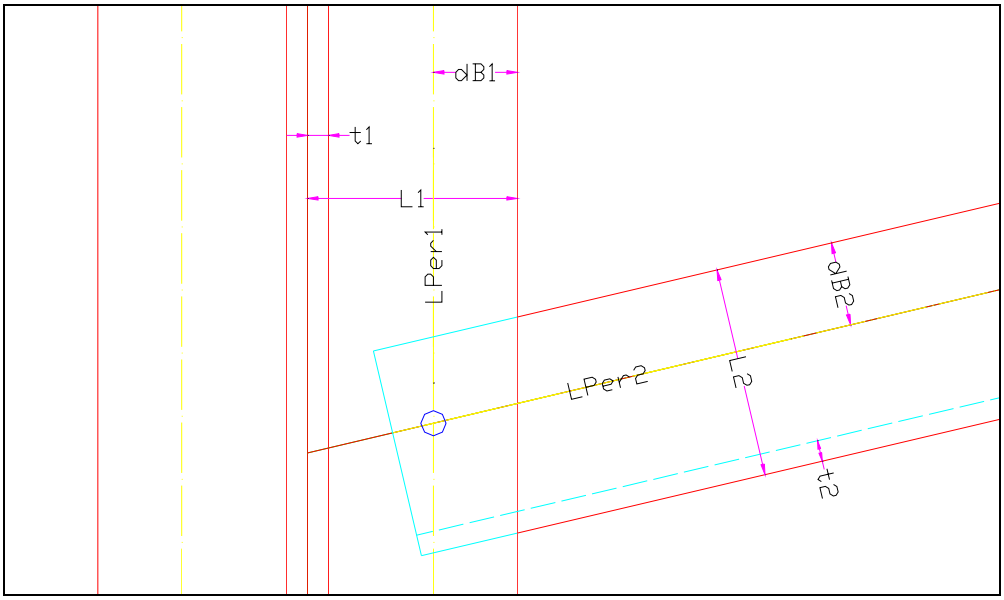


Figura # 40. Distancia al Borde

4- Seleccionar la línea que define al perfil

5- La línea que define el perfil se tornará de color verde.

5.3- Unión de Perfiles:

1- Seleccionar el perfil base (Perfil 1).

2- Seleccionar el perfil que se va a conectar (Perfil 2). Ver Figura #41.

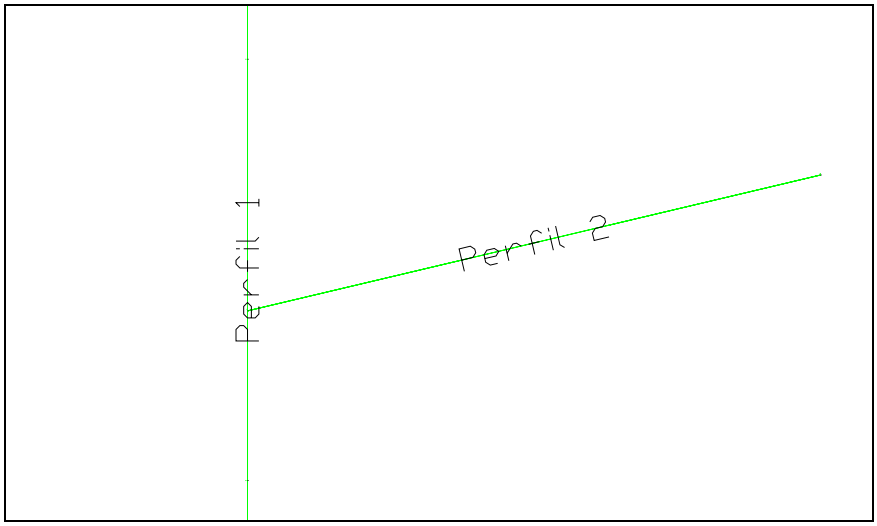


Figura # 41. Selección de Perfiles

3- Definir la distancia (d) en milímetros a partir del perno que va a tener el perfil que se va a conectar, esta distancia no debe ser mayor que la distancia existente entre los perfiles (dmax).

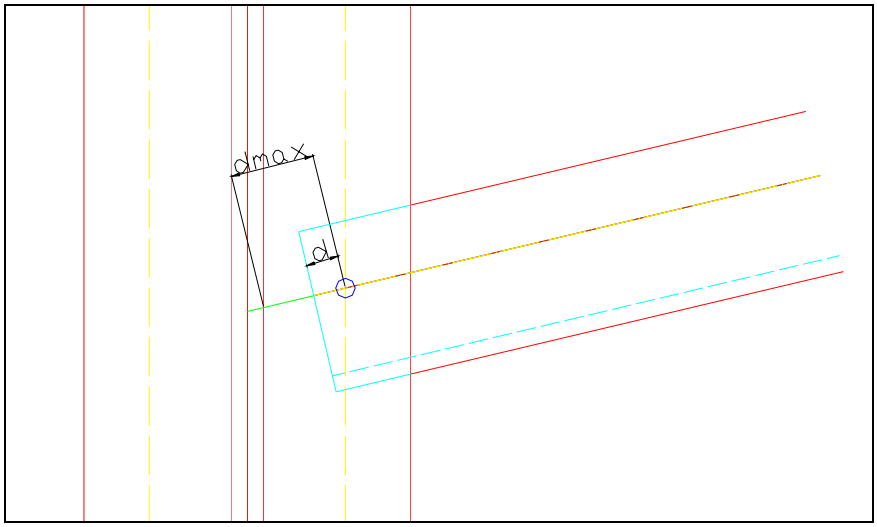


Figura # 42. Definición de la distancia d

5.4- Unión Para Planchas:

1. Seleccionar el perfil base (Perfil 1).
2. Seleccionar el perfil que se va a separar (Perfil 2). Ver Figura # 41.
3. Definir el número de pernos que tendrá el Perfil 2.
4. Definir la distancia entre los pernos (dP).
5. Definir la distancia entre el borde y el perno mas cercano al borde del Perfil 2(dB).
6. Definir la distancia perpendicular al Perfil 1 (disPer) que separará al Perfil 1 del Perfil 2.

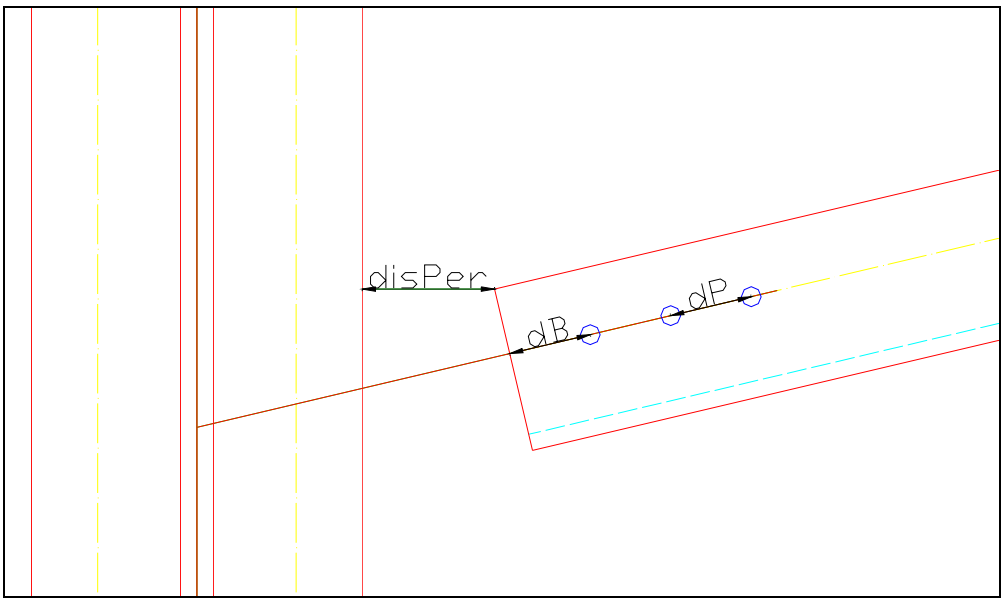


Figura # 43. Unión para Planchas

5.5- Separadores:

1- Seleccionar que tipo de separador se desea graficar.

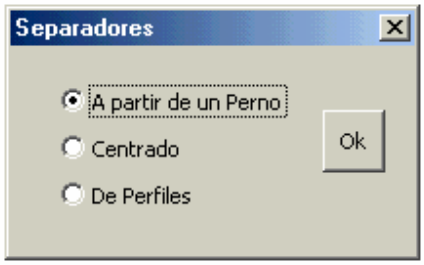


Figura # 44. Selección de Separadores

5.5.2- A partir de un Perno:

1- Seleccionar el perno (Perno) del cual se va a referenciar el separador.

2- Seleccionar el perfil (Perfil) en el cual se va a encontrar el separador.

3- Definir la ubicación del perno en el separador.

4- Definir el número de pernos por ala de perfil que va a tener el separador.

5- Definir la distancia entre los pernos (dP).

6- Definir la distancia entre el borde y el perno mas cercano al borde del Perfil (dB).

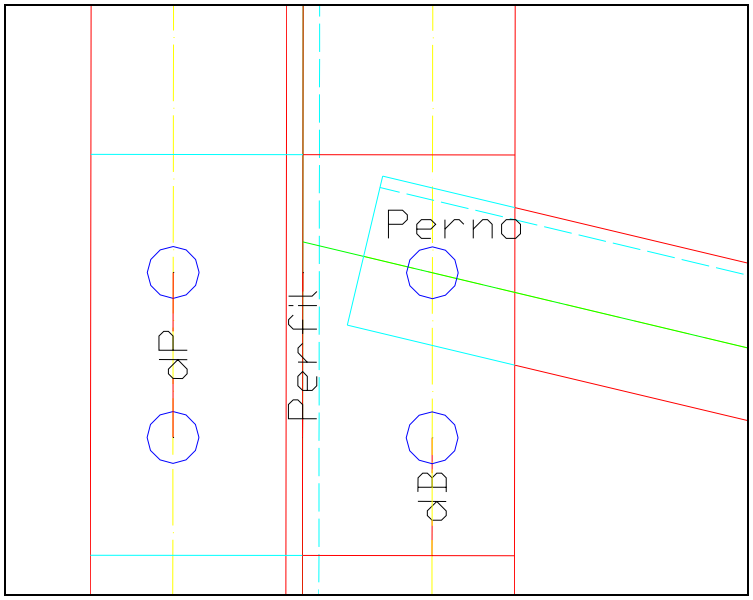


Figura # 45. Definición de distancias

7- Definir el punto en el cual se desea que se encuentre el detalle del separador.

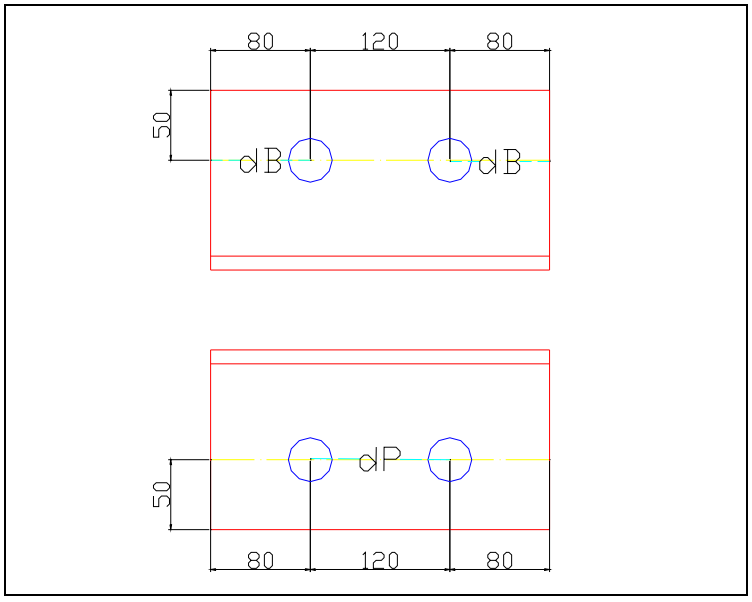


Figura # 46. Separador

5.5.3- Centrado:

1- Seleccionar los pernos (Perno 1 y Perno 2) entre los cuales se desea que vaya el separador.

2- Seleccionar el perfil (Perfil) en el cual se va a encontrar el separador.

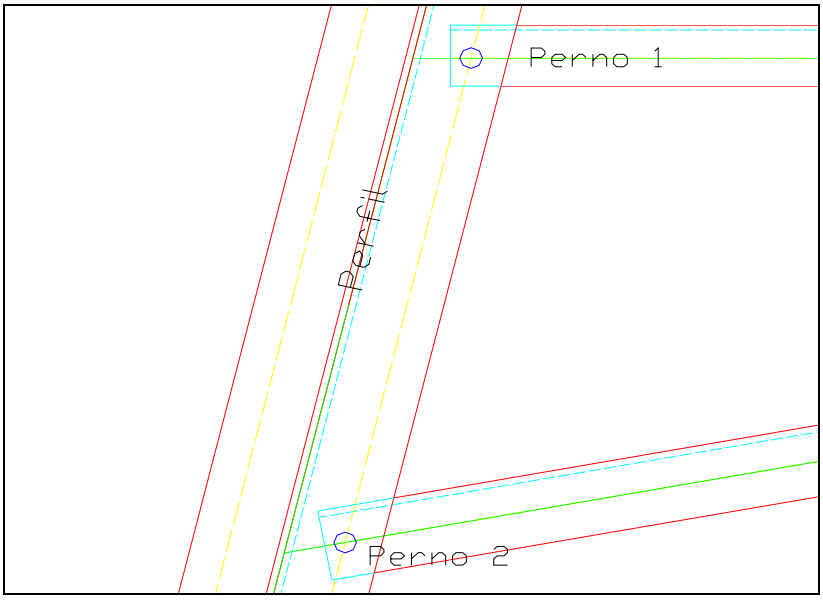


Figura # 47. Perfil y Pernos

3- Definir el número de pernos por ala de perfil que va a tener el separador.

4- Definir la distancia entre los pernos (dP).

5- Definir la distancia entre el borde y el perno mas cercano al borde del Perfil (dB).

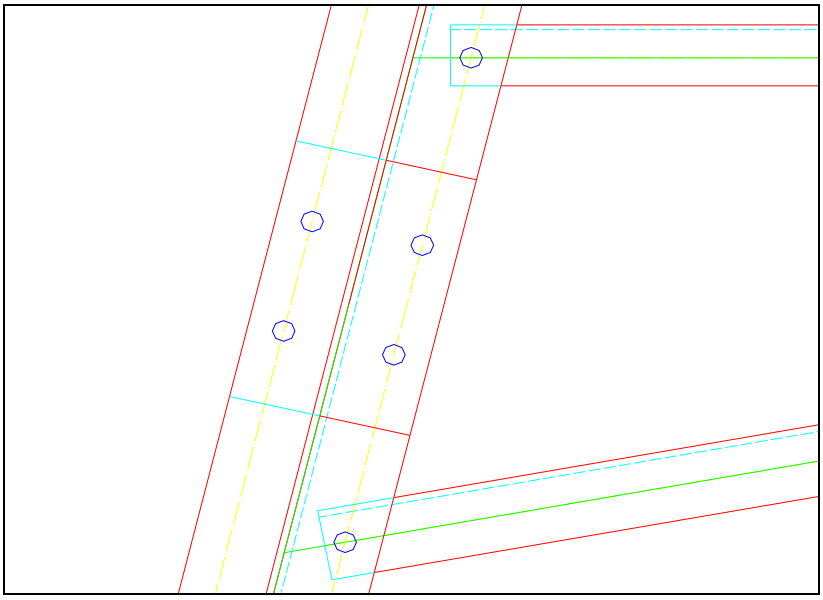


Figura # 48. Separador Centrado

6- Escoger el punto en el cual se desea que se encuentre el detalle

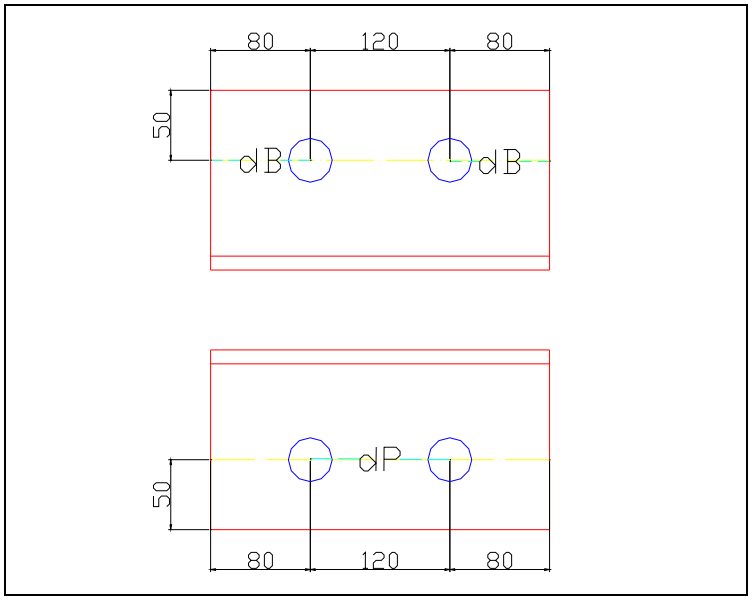


Figura # 49. Detalle Separador Centrado

5.5.4- De Perfiles:

1- Seleccionar los pernos (Perno 1 y Perno 2) entre los cuales se desea que vaya el separador.

2- Seleccionar el perfil (Perfil) en el cual se va a encontrar el separador .

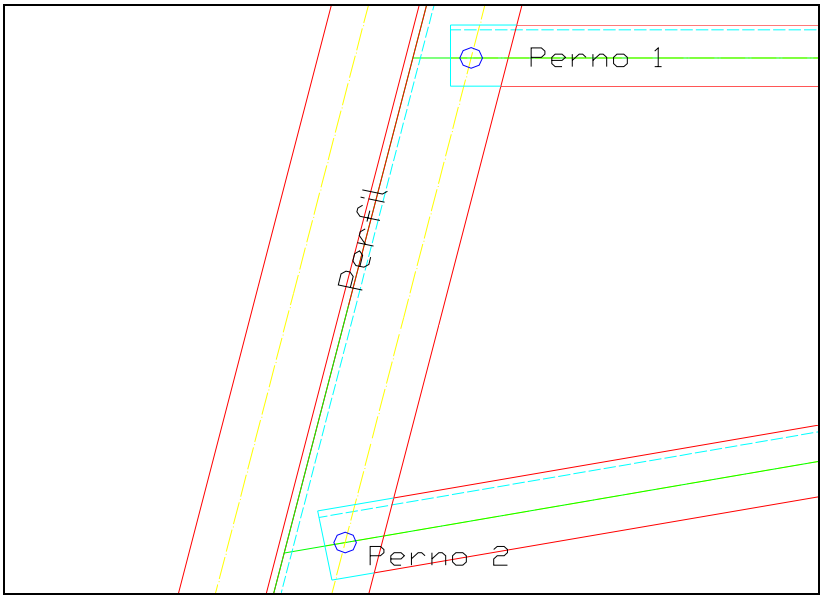


Figura # 50. Selección de Perfiles para Separadores

3- Escoger el punto en el cual se desea que se encuentre el detalle.

4- Definir si se desea que los pernos verticales esten alineados con los horizontales.

5- Definir la distancia (dS) que separa a los perfiles.

6- Definir el numero de pernos por ala de perfil que va a tener el separador.

7- Definir la distancia entre los pernos (dP).

8- Definir la distancia entre el borde y el perno mas cercano al borde del Perfil (dB).

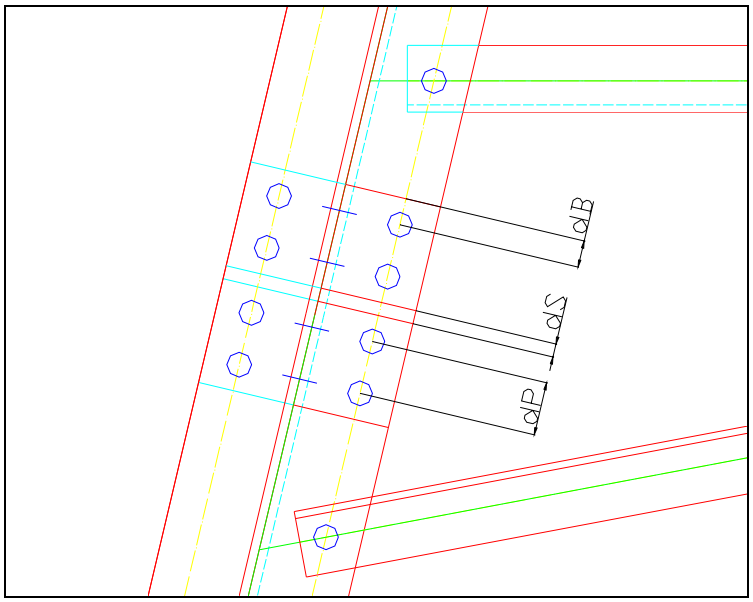


Figura # 51. Detalles del Separador

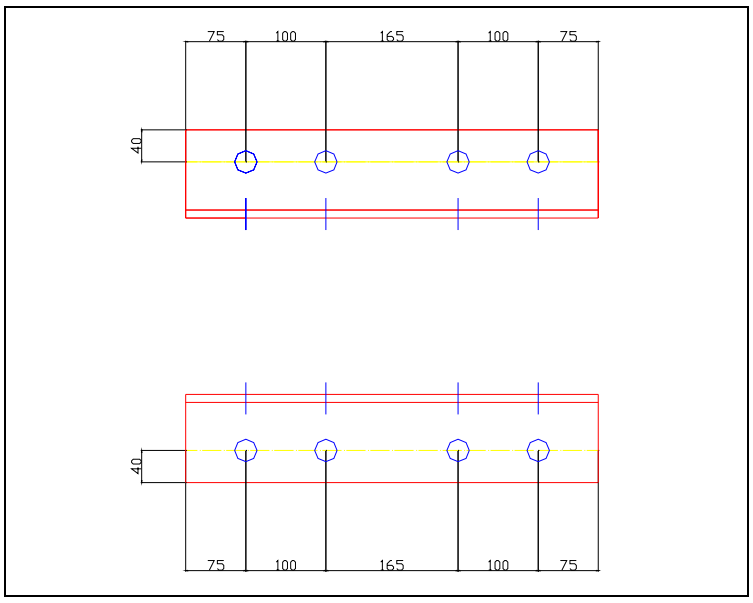


Figura # 52. Detalles

5.6- Planchas:

1- Seleccionar de manera antihoraria los pernos que definen la forma de la plancha.

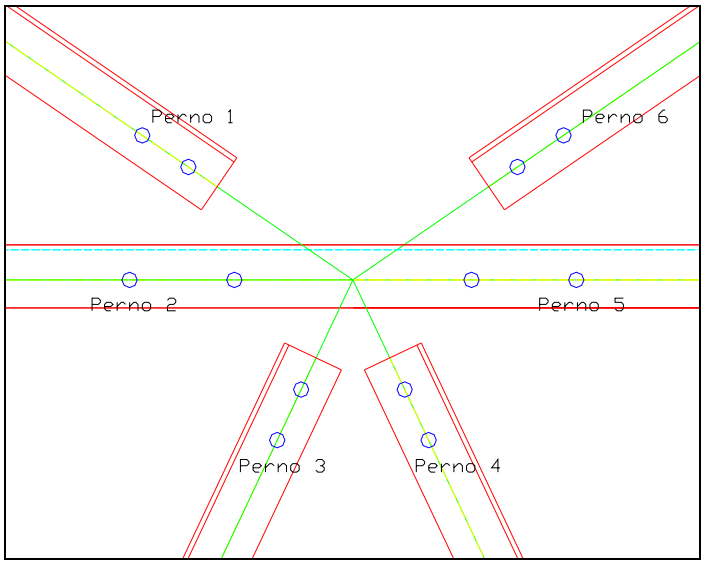


Figura # 53. Selección de Pernos para Planchas

2- Definir la mínima distancia que debe existir entre el borde de la plancha y los pernos.

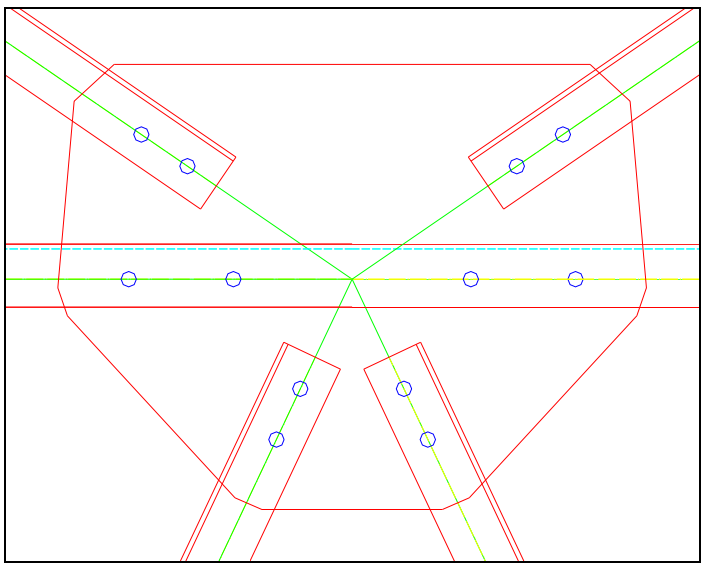


Figura # 54. Definición de Distancias para Planchas

5.7- Visibilidad:

1- Seleccionar las líneas externas del perfil que cortan a la plancha (LPerfil).

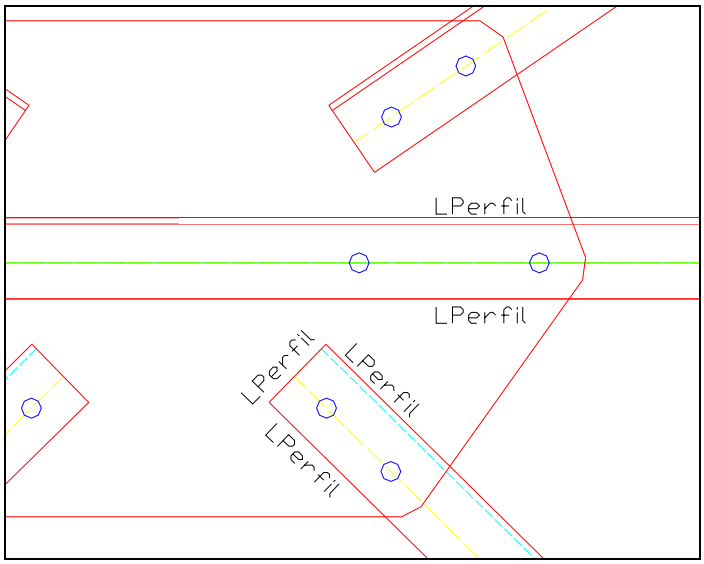


Figura # 55. Selección de Líneas Externas de Perfiles

2- Seleccionar las líneas de la plancha que se encuentran dentro del perfil (LPlancha).

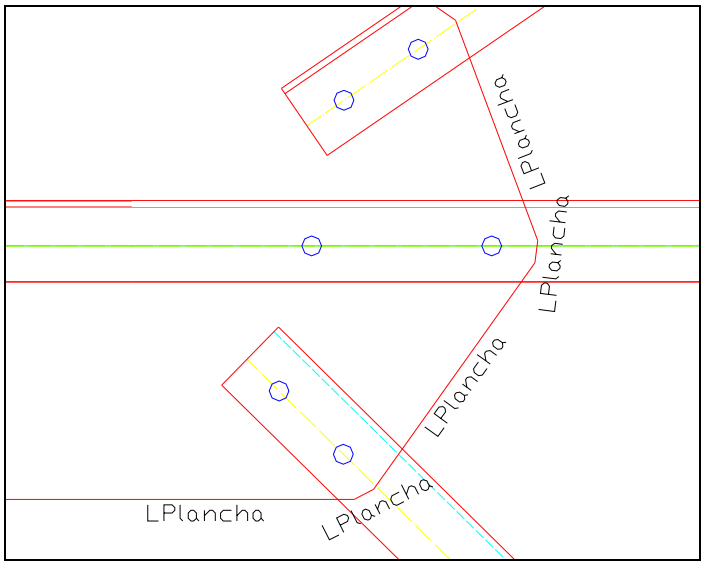


Figura # 56. Selección de Líneas Internas

3- Definir cuales son las líneas que no ven.

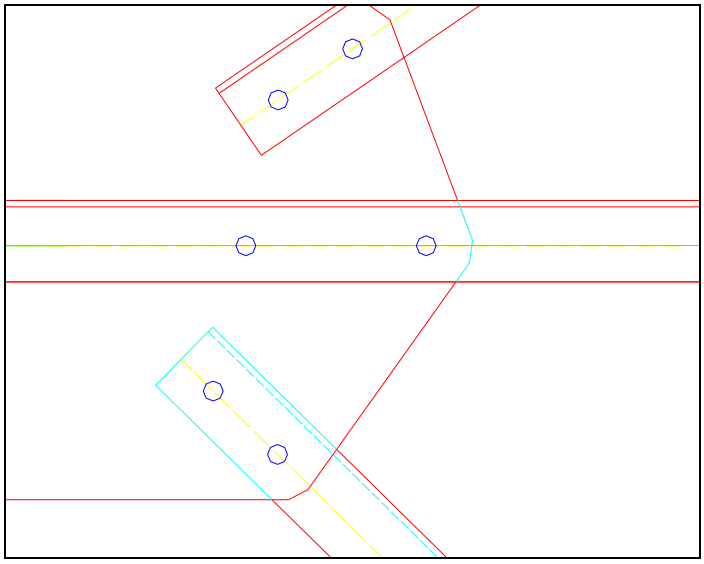


Figura # 57. Visibilidad

5.8- Dimensionar:

1- Seleccionar de la línea de pernos que se desea acotar.

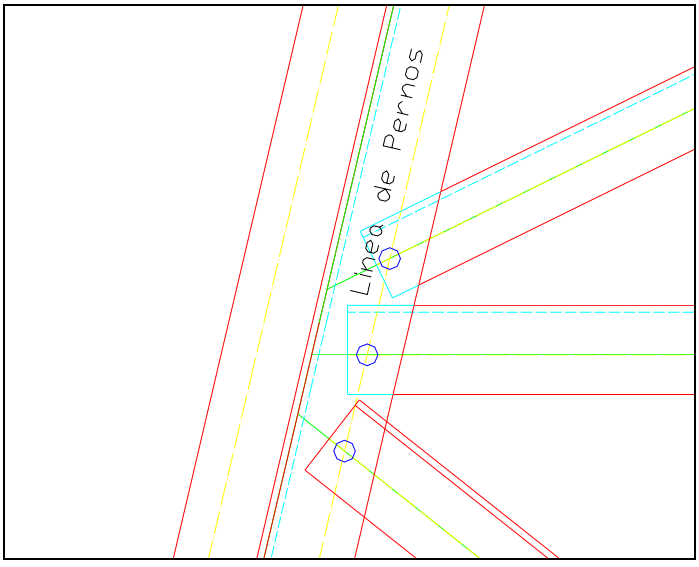


Figura # 58. Selección de Distancias a Acotar

2- Seleccionar el punto en el cual se desea que se encuentre el acotamiento.

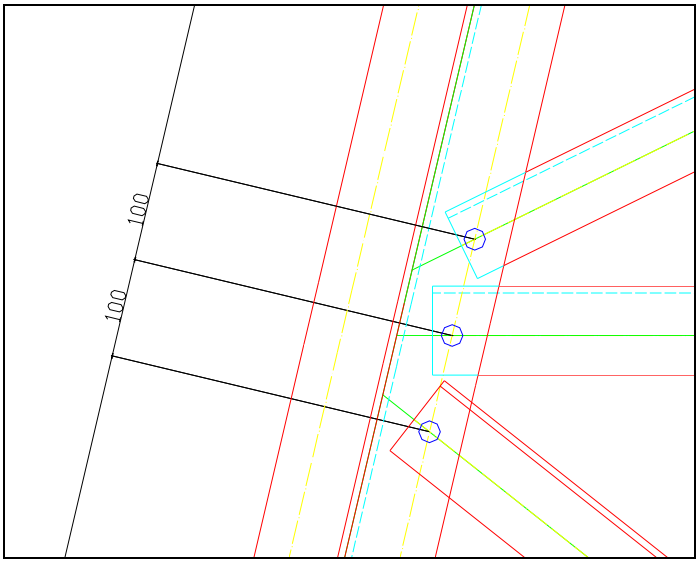


Figura # 59. Acotamiento

5.9- Detalle Cruz:

1- Seleccionar el perfil al cual se le desea hacer el detalle.

2- Escoger el punto en el se desea que se ubique el detalle.

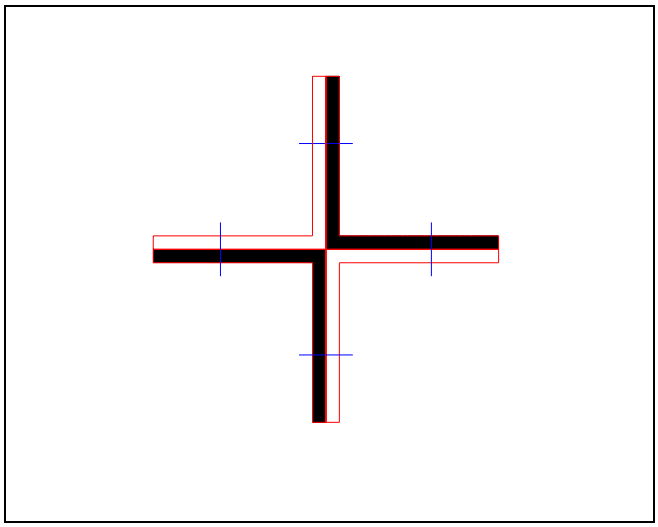


Figura # 60. Detalle Cruz

6.- LISTADO DEL PROGRAMA

6.2- MÓDULO1

Contiene las rutinas principales de "Perfiles de Acero".

El MÓDULO1 es de opción explícita, contiene las siguientes variables globales:

```
Public acadDoc              As AcadDocument
Public acadApp              As AcadApplication
Public mospace              As AcadModelSpace
Public flagi                As Boolean
Public Const direcciones    As String = "c:\Archivos de programa\Acero\"
```

Y está constituido por las siguientes rutinas y funciones:

Nombre	Sub Tipo_de_lineas()
Descripción	Define los distintos tipos de líneas que se van a utilizar en la realización del plano.
Variables de Entrada	`No tiene`

Código	`Dim entry  As AcadLineType`
	`Dim found  As Boolean`
	`found = False`
	`For Each entry In acadDoc.Linetypes`
	`If StrComp(entry.Name, "acad_iso04w100", 1) = 0 Then`
	`found = True`
	`Exit For`
	`End If`
	`Next`
	`If Not (found) Then acadDoc.Linetypes.Load`
	`"acad_iso04w100", "acad.lin"`
	`found = False`
	`For Each entry In acadDoc.Linetypes`
	`If StrComp(entry.Name, "acad_iso02w100", 1) = 0 Then`
	`found = True`
	`Exit For`
	`End If`
	`Next`
	`If Not (found) Then acadDoc.Linetypes.Load`
	`"acad_iso02w100", "acad.lin"`

Nombre	Sub ArregloSE_de_2_lineas()
Descripción	Ordena las líneas de tal manera que el EndPoint de la linea2 será el punto de la linea2 que está mas cercano a la linea1.
Variables de Entrada	`Linea1 As AcadLine` `Linea2 As AcadLine`

Código	`Dim d1 As Double`
	`Dim d2 As Double`
	`Dim p As Variant`
	`Dim po(0 To 2) As Double`
	`Dim pf(0 To 2) As Double`
	`p = linea2.IntersectWith(Linea1, acExtendBoth)`
	`d1 = ((p(0) - linea2.StartPoint(0)) ^ 2 + (p(1) - linea2.StartPoint(1)) ^ 2 + (p(2) - linea2.StartPoint(2)) ^ 2) ^ 0.5`
	`d2 = ((p(0) - linea2.EndPoint(0)) ^ 2 + (p(1) - linea2.EndPoint(1)) ^ 2 + (p(2) - linea2.EndPoint(2)) ^ 2) ^ 0.5`
	`If d1 <= d2 Then`
	`linea2.Rotate linea2.StartPoint, 3.14159265358`
	`linea2.Move linea2.EndPoint, linea2.StartPoint`
	`End If`
	`End Sub`
	`Sub arregloSE_de_linea_con_punto(linea As AcadLine, p() As Double)`
	`Dim d1 As Double`
	`Dim d2 As Double`
	`Dim po(0 To 2) As Double`
	`Dim pf(0 To 2) As Double`
	`d1 = ((p(0) - linea.StartPoint(0)) ^ 2 + (p(1) - linea.StartPoint(1)) ^ 2 + (p(2) - linea.StartPoint(2)) ^ 2) ^ 0.5`
	`d2 = ((p(0) - linea.EndPoint(0)) ^ 2 + (p(1) - linea.EndPoint(1)) ^ 2 + (p(2) - linea.EndPoint(2)) ^ 2) ^ 0.5`
	`If d1 <= d2 Then`
	`po(0) = linea.EndPoint(0)`
	`po(1) = linea.EndPoint(1)`
	`po(2) = linea.EndPoint(2)`
	`pf(0) = linea.StartPoint(0)`
	`pf(1) = linea.StartPoint(1)`
	`pf(2) = linea.StartPoint(2)`
	`linea.Delete`
	`Set linea = acadDoc.ModelSpace.AddLine(po, pf)`
	`End If`

Nombre	Sub Extiende_linea_una_distancia()
Descripción	Crea una línea nueva, con las mismas características que la anterior, pero mas larga una distancia d .
Variables de Entrada	`linea As AcadLine, d As Double`

Código	`Dim angulo As Double`
	`Dim po(0 To 2) As Double`
	`Dim pf(0 To 2) As Double`
	`angulo = linea.Angle`
	`po(0) = linea.StartPoint(0)`
	`po(1) = linea.StartPoint(1)`
	`pf(0) = Cos(angulo) * (linea.Length + d) + po(0)`
	`pf(1) = Sin(angulo) * (linea.Length + d) + po(1)`
	`linea.Delete`
	`Set linea = acadDoc.ModelSpace.AddLine(po, pf)`

Nombre	Sub Extiende_linea_a_linea()
Descripción	Permite dibujar una línea auxiliar que se encuentra extendida hasta la intersección con otra línea.
Variables de Entrada	`L1 As AcadLine, L2 As AcadLine` `punto() As Double`

Código	`Dim p As Variant`
	`Dim po(0 To 2) As Double`
	`p = L2.IntersectWith(L1, acExtendThisEntity)`
	`punto(0) = p(0): punto(1) = p(1): punto(2) = p(2)`
	`Call arregloSE_de_2_lineas(L1, L2)`
	`po(0) = L2.StartPoint(0): po(1) = L2.StartPoint(1):`
	`po(2) = L2.StartPoint(2)`
	`L2.Delete`
	`Set L2 = acadDoc.ModelSpace.AddLine(po, punto)`

Nombre	Sub Conv_de_var_en_linea()
Descripción	Convierte una variable tipo Variant en una variable tipo AcadLine.
Variables de Entrada	`va As AcadLine, L As AcadLine`

Código	`Set L = acadDoc.ModelSpace.AddLine(va.StartPoint,`
	`va.EndPoint)`
	`va.Delete`

Nombre	Sub distancias_possible()
Descripción	Calcula la distancia máxima que puede tener una conexión de 2 perfiles angulares sin que éstos se solapen.
Variables de Entrada	`linea2 As AcadLine, perno() As Double, L11 As AcadLine, L12 As AcadLine, L21 As AcadLine, L22 As AcadLine, LP As AcadLine, distpos As Double, radio As Double`

Código	`Dim lp1 As AcadLine`
	`Dim lp2 As AcadLine`
	`Dim d As Double`
	`Dim d1 As Double`
	`Dim d2 As Double`
	`Dim pa As Variant`
	`Dim pb As Variant`
	`Dim Message As String`
	`pa = L12.IntersectWith(L21, acExtendBoth)`
	`pb = L12.IntersectWith(L22, acExtendBoth)`
	`Set lp1 = LP.Copy`
	`Set lp2 = LP.Copy`
	`lp1.Move LP.EndPoint, pa`
	`lp2.Move LP.EndPoint, pb`
	`pa = linea2.IntersectWith(lp1, acExtendBoth)`
	`pb = linea2.IntersectWith(lp2, acExtendBoth)`
	`d1 = ((perno(0) - pa(0)) ^ 2 + (perno(1) - pa(1)) ^ 2 + (perno(2) - pa(2)) ^ 2) ^ 0.5`
	`d2 = ((perno(0) - pb(0)) ^ 2 + (perno(1) - pb(1)) ^ 2 + (perno(2) - pb(2)) ^ 2) ^ 0.5`
	`If d2 > d1 Then d = d1`
	`If d1 >= d2 Then d = d2`
	`lp1.Delete`
	`lp2.Delete`
	`If d <= Fix(radio + 1) Then`
	`MsgBox "No Cabe el Perfil, Borre"`
	`Exit Sub`
	`End If`
	`Message = "Distancia entre " & Fix(radio + 1) & " y " & Fix(d)`
	`distpos = InputBox(Message, "Distancia Posible")`
	`Do While Fix(d) < distpos Or distpos <= Fix(radio + 1)`
	`distpos = InputBox(Message, "Distancia Posible")`
	`Loop`

Nombre	Sub Saca_datos()
Descripción	Permite extraer del plano las características de cada perfil que se va a utilizar, que ya fueron definidas en cada Línea de Cg.
Variables de Entrada	`linea As AcadLine, L As Double, t As Double, dv As Double, vista As Double, radio As Double, perfil As String`

Código	`Dim ent As AcadEntity`
	`Dim lay1 As AcadLayer`
	`Dim lay2 As AcadLayer`
	`Dim lay3 As AcadLayer`
	`Dim xdataOut As Variant`
	`Dim xtypeOut As Variant`
	`Dim tipo As String`
	`Dim texto As String`
	`Dim nombre As String`
	`Set lay1 = acadDoc.Layers.Add("Perfiles")`
	`lay1.Color = acRed`
	`Set lay2 = acadDoc.Layers.Add("Linea de Pernos")`
	`lay2.Color = acYellow`
	`Set lay3 = acadDoc.Layers.Add("0")`
	`Repite:`
	`lay1.LayerOn = False`
	`lay2.LayerOn = False`
	`lay3.LayerOn = False`
	`texto = "Seleccione el Perfil " & perfil`
	`Call Test_GetEntity(ent, texto)`
	`Call prende_layers`
	`If Lresult = 2 Then Exit Sub`
	`If ent.ObjectName <> "AcDbLine" Then GoTo Repite`
	`Set linea = ent`
	`linea.GetXData "", xtypeOut, xdataOut`
	`tipo = xdataOut(1)`
	`dv = xdataOut(3)`
	`vista = xdataOut(5)`
	`radio = xdataOut(6)`
	`tipo = "x" + tipo`
	`Open direcciones + "datos.txt" For Input As #1`
	`Do While Not EOF(1)`
	`Input #1, nombre`
	`Input #1, L`
	`Input #1, t`
	`If nombre = tipo Then Exit Do`
	`Loop`
	`Close #1`

Nombre	Sub recorta_linea_1()
Descripción	Calcula y dibuja una línea auxiliar (con las mismas características que la linea1) que está comprendida entre los puntos medios de las próximas conexiones.
Variables de Entrada	`Linea1 As AcadLine, linea2 As AcadLine, linean As AcadLine`

Código	`Dim linean1 As AcadLine`
	`Dim linean2 As AcadLine`
	`Dim p(0 To 2) As Double`
	`Dim pdef1(0 To 2) As Double`
	`Dim pdef2(0 To 2) As Double`
	`Dim pn1(0 To 2) As Double`
	`Dim pn2(0 To 2) As Double`
	`Dim di As Double`
	`Dim ddef As Double`
	`Dim entry As Variant`
	`Dim pi As Variant`
	`Dim pV As Variant`
	`Dim i As Integer`
	`Dim h1 As Integer`
	`Dim h2 As Integer`
	`Dim k As Integer`
	`Dim j As Integer`
	`h1 = 0`
	`pV = linea2.IntersectWith(Linea1, acExtendNone)`
	`p(0) = pV(0): p(1) = pV(1): p(2) = pV(2)`
	`Set linean1 = acadDoc.ModelSpace.AddLine(p, Linea1.StartPoint)`
	`Set linean2 = acadDoc.ModelSpace.AddLine(p, Linea1.EndPoint)`
	`linean1.Color = acBlue: linean2.Color = acYellow`
	`For Each entry In acadDoc.ModelSpace`
	`If entry.handle <> Linea1.handle And entry.handle <> linea2.handle And entry.handle <> linean1.handle And entry.handle <> linean2.handle And entry.Layer = "Linea de Cg" Then`
	`pi = linean1.IntersectWith(entry, acExtendNone)`
	`k = 0`
	`For i = LBound(pi) To UBound(pi)`
	`k = k + 1`
	`If k = 1 Then`
	`di = ((pi(0) - p(0)) ^ 2 + (pi(1) - p(1)) ^ 2 + (pi(2) - p(2)) ^ 2) ^ 0.5`
	`If h1 = 0 And di > 0.1 Then`
	`ddef = di : h1 = 1`
	`End If`
	`If di <= ddef And di > 0.1 Then`
	`ddef = di`
	`pdef1(0) = pi(0): pdef1(1) = pi(1): pdef1(2) = pi(2)`

```
End If
End If
Next
End If
Next
h2 = 0
For Each entry In acadDoc.ModelSpace
If entry.handle <> Linea1.handle And entry.handle <>
linea2.handle And entry.handle <> linean1.handle And
entry.handle <> linean2.handle And entry.Layer =
"Linea de Cg" Then
pi = linean2.IntersectWith(entry, acExtendNone)
k = 0
For i = LBound(pi) To UBound(pi)
k = k + 1
If k = 1 Then
di = ((pi(0) - p(0)) ^ 2 + (pi(1) - p(1)) ^ 2 +
(pi(2) - p(2)) ^ 2) ^ 0.5
If h2 = 0 And di > 0.1 Then
ddef = di : h2 = 1
End If
If di <= ddef And di > 0.1 Then
ddef = di
pdef2(0) = pi(0): pdef2(1) = pi(1): pdef2(2) = pi(2)
End If
End If
Next
End If
Next
If h1 = 1 Then
For j = 0 To 2
pn1(j) = (pdef1(j) + p(j)) / 2
Next j
End If
If h2 = 1 Then
For j = 0 To 2
pn2(j) = (pdef2(j) + p(j)) / 2
Next j
End If
If h1 = 0 Then
pn1(0) = linean1.EndPoint(0): pn1(1) =
linean1.EndPoint(1): pn1(2) = linean1.EndPoint(2)
End If
If h2 = 0 Then
pn2(0) = linean2.EndPoint(0): pn2(1) =
linean2.EndPoint(1): pn2(2) = linean2.EndPoint(2)
End If
Set linean = acadDoc.ModelSpace.AddLine(pn1, pn2)
linean.Color = acRed
linean1.Delete: linean2.Delete
```

Nombre	Sub recorta_linea_2()
Descripción	Calcula y dibuja una línea auxiliar (con las mismas características que la linea1) que está comprendida entre los puntos medios de las próximas conexiones.
Variables de Entrada	`Linea1 As AcadLine, linea2 As AcadLine, linean As AcadLine`

Código	`Dim p(0 To 2) As Double`
	`Dim pdef(0 To 2) As Double`
	`Dim pn(0 To 2) As Double`
	`Dim di As Double`
	`Dim ddef As Double`
	`Dim i As Integer`
	`Dim h As Integer`
	`Dim k As Integer`
	`Dim j As Integer`
	`Dim entry As Variant`
	`Dim pi As Variant`
	`Dim pV As Variant`
	`h = 0`
	`pV = linea2.IntersectWith(Linea1, acExtendNone)`
	`p(0) = pV(0): p(1) = pV(1): p(2) = pV(2)`
	`For Each entry In acadDoc.ModelSpace`
	`If entry.handle <> Linea1.handle And entry.handle <> linea2.handle And entry.Layer = "Linea de Cg" Then`
	`pi = linea2.IntersectWith(entry, acExtendNone):k = 0`
	`For i = LBound(pi) To UBound(pi)`
	`k = k + 1`
	`If k = 1 Then`
	`di = ((pi(0) - p(0)) ^ 2 + (pi(1) - p(1)) ^ 2 + (pi(2) - p(2)) ^ 2) ^ 0.5`
	`If h = 0 And di > 0.1 Then ddef = di : h = 1`
	`End If`
	`If di <= ddef And di > 0.1 Then ddef = di`
	`pdef(0) = pi(0): pdef(1) = pi(1): pdef(2) = pi(2)`
	`End If`
	`End If`
	`Next`
	`End If`
	`Next`
	`If h = 1 Then`
	`For j = 0 To 2`
	`pn(j) = (pdef(j) + p(j)) / 2`
	`Next j`
	`Set linean = acadDoc.ModelSpace.AddLine(pn, p)`
	`End If`
	`If h = 0 Then Set linean = linea2.Copy`
	`Linean.Color = acRed`

Nombre	Sub ArregloID()
Descripción	Permite dibujar una línea auxiliar que tiene el punto de inicio a la izquierda de la pantalla.
Variables de Entrada	`linea As AcadLine`

Código	`If linea.StartPoint(0) > linea.EndPoint(0) Then`
	`linea.Rotate linea.StartPoint, 3.14159265358`
	`linea.Move linea.EndPoint, linea.StartPoint`
	`End If`
	`If linea.StartPoint(0) = linea.EndPoint(0) Then`
	`If linea.StartPoint(1) > linea.EndPoint(1) Then`
	`linea.Rotate linea.StartPoint, 3.14159265358`
	`linea.Move linea.EndPoint, linea.StartPoint`
	`End If`
	`End If`

Nombre	Sub Calcula_distancia_perpendicular_entre_punto _y_linea2 ()
Descripción	Calcula la distancia perpendicular entre un punto y una línea.
Variables de Entrada	`linea As AcadLine, punto() As Double` `d As Double`

Código	`Dim pV As Variant`
	`Dim linead As AcadLine`
	`Dim p(0 To 2) As Double`
	`Set linead = linea.Copy`
	`linead.Move linead.StartPoint, punto()`
	`linead.Rotate punto(), -3.14159265358 / 2`
	`pV = linead.IntersectWith(linea, acExtendBoth)`
	`p(0) = pV(0): p(1) = pV(1): p(2) = pV(2)`
	`d = ((punto(0) - p(0)) ^ 2 + (punto(1) - p(1)) ^ 2 +`
	`(punto(2) - p(2)) ^ 2) ^ 0.5`
	`linead.Delete`

Nombre	Sub dibuja_perfil()
Descripción	Permite dibujar todas las líneas que definen el perfil, que el programa las conozca como variables y que defina cuales son las que se deben interceptar.
Variables de Entrada	`Linea1 As AcadLine, linea2 As AcadLine, vista1 As Double, L11 As AcadLine, L12 As AcadLine, L13 As AcadLine, Lineadepernos1 As AcadLine, L1 As Double, t1 As Double, dv1 As Double, L11i As AcadLine, L13i As AcadLine, L11s As AcadLine, L13s As AcadLine, Lineadepernos1i As AcadLine, Lineadepernos1s As AcadLine`

Código	`Dim v As Variant`
	`Dim vf As Integer`
	`Dim i As Integer`
	`If vista1 <= 4 Then`
	`v = Linea1.Offset(-L1)`
	`Call conv_de_var_en_linea(v(0), L11i)`
	`L11i.Layer = "Perfiles"`
	`v = Linea1.Offset(L1)`
	`Call conv_de_var_en_linea(v(0), L11s)`
	`L11s.Layer = "Perfiles"`
	`v = Linea1.Offset(-t1)`
	`Call conv_de_var_en_linea(v(0), L13i)`
	`L13i.Layer = "Perfiles"`
	`v = Linea1.Offset(t1)`
	`Call conv_de_var_en_linea(v(0), L13s)`
	`L13s.Layer = "Perfiles"`
	`v = Linea1.Offset(-L1 + dv1)`
	`Call conv_de_var_en_linea(v(0), Lineadepernos1i)`
	`Lineadepernos1i.Layer = "Linea de Pernos"`
	`v = Linea1.Offset(L1 - dv1)`
	`Call conv_de_var_en_linea(v(0), Lineadepernos1s)`
	`Lineadepernos1s.Layer = "Linea de Pernos"`
	`If vista1 = 2 Then`
	`L13i.Linetype = "acad_iso02w100"`
	`L13s.Linetype = "acad_iso02w100"`
	`L13i.Color = acCyan`
	`L13s.Color = acCyan`
	`End If`
	`If vista1 = 3 Then`
	`L13s.Linetype = "acad_iso02w100"`
	`L13s.Color = acCyan`
	`End If`
	`If vista1 = 4 Then`
	`L13i.Linetype = "acad_iso02w100"`
	`L13i.Color = acCyan`
	`End If`
	`Lineadepernos1s.Linetype = "acad_iso04w100"`
	`Lineadepernos1i.Linetype = "acad_iso04w100"`

```
v = linea2.IntersectWith(L11s, acExtendNone)
vf = 0
If VarType(v) <> vbEmpty Then
For i = LBound(v) To UBound(v)
vf = vf + 1
If vf = 1 Then
Set L11 = L11s.Copy
Set L12 = Linea1.Copy
Set L13 = L13s.Copy
Set Lineadepernos1 = Lineadepernos1s.Copy
L11s.Delete
L13s.Delete
Lineadepernos1s.Delete
End If
Next i
End If
v = linea2.IntersectWith(L11i, acExtendNone)
vf = 0
If VarType(v) <> vbEmpty Then
For i = LBound(v) To UBound(v)
vf = vf + 1
If vf = 1 Then
Set L11 = L11i.Copy
Set L12 = Linea1.Copy
Set L13 = L13i.Copy
Set Lineadepernos1 = Lineadepernos1i.Copy
L11i.Delete
L13i.Delete
Lineadepernos1i.Delete
End If
Next i
End If
End If
If vista1 = 5 Or vista1 = 6 Then
v = Linea1.Offset(dv1)
Call conv_de_var_en_linea(v(0), L11)
v = Linea1.Offset(dv1 - L1)
Call conv_de_var_en_linea(v(0), L12)
v = Linea1.Offset(dv1 - L1 + t1)
Call conv_de_var_en_linea(v(0), L13)
v = Linea1.Offset(dv1 - dv1)
Call conv_de_var_en_linea(v(0), Lineadepernos1)
If vista1 = 6 Then
L13.Linetype = "acad_iso02w100"
L13.Color = acCyan
End If
End If
If vista1 = 7 Or vista1 = 8 Then
v = Linea1.Offset(-dv1)
Call conv_de_var_en_linea(v(0), L11)
v = Linea1.Offset(-dv1 + L1)
Call conv_de_var_en_linea(v(0), L12)
```

	v = Linea1.Offset(-dv1 + L1 - t1)
	Call conv_de_var_en_linea(v(0), L13)
	v = Linea1.Offset(-dv1 + dv1)
	Call conv_de_var_en_linea(v(0), Lineadepernos1)
	If vista1 = 8 Then
	L13.Linetype = "acad_iso02w100"
	L13.Color = acCyan
	End If
	End If
	L11.Layer = "Perfiles"
	L12.Layer = "Perfiles"
	L13.Layer = "Perfiles"
	Lineadepernos1.Layer = "Linea de Pernos"
	Lineadepernos1.Linetype = "acad_iso04w100"

Nombre	Sub Borra Perfil()
Descripción	Permite borrar las líneas que definen un perfil.
Variables de Entrada	L11 As AcadLine, L12 As AcadLine, L13 As AcadLine, Lineadepernos1 As AcadLine, L11i As AcadLine, L13i As AcadLine, L11s As AcadLine, L13s As AcadLine, Lineadepernos1i As AcadLine, Lineadepernos1s As AcadLine
Código	L11.Delete: L12.Delete: L13.Delete Lineadepernos1.Delete: L11i.Delete: L13i.Delete Lineadepernos1i.Delete: L11s.Delete: L13s.Delete Lineadepernos1s.Delete

Nombre	Sub Dibuja Pernos()
Descripción	Permite dibujar la distribución de pernos de un perfil que fue dada por el usuario.
Variables de Entrada	`lineadepernos As AcadLine, np As Integer, db As Integer, dP As Integer, radio As Double`

Código	`Dim angulo As Double`
	`Dim po(0 To 2) As Double`
	`Dim pf(0 To 2) As Double`
	`Dim i As Integer`
	`Dim LdB As AcadLine`
	`Dim LdP As AcadLine`
	`Dim perno As AcadCircle`
	`angulo = lineadepernos.Angle`
	`pf(0) = db`
	`Set LdB = acadDoc.ModelSpace.AddLine(po, pf)`
	`LdB.Move pf, lineadepernos.EndPoint`
	`LdB.Rotate lineadepernos.EndPoint, angulo`
	`pf(0) = dP`
	`Set LdP = acadDoc.ModelSpace.AddLine(po, pf)`
	`LdP.Move pf, lineadepernos.EndPoint`
	`LdP.Rotate lineadepernos.EndPoint, angulo`
	`LdP.Move LdB.EndPoint, LdB.StartPoint`
	`Set perno =`
	`acadDoc.ModelSpace.AddCircle(LdP.EndPoint, radio)`
	`perno.Layer = "Pernos"`
	`For i = 1 To np - 1`
	`Set perno =`
	`acadDoc.ModelSpace.AddCircle(LdP.StartPoint, radio)`
	`perno.Layer = "Pernos"`
	`LdP.Move LdP.EndPoint, LdP.StartPoint`
	`Next i`
	`LdP.Delete`
	`LdB.Delete`

Nombre	Sub Calcula_lineas_que_mandan2()
Descripción	Permite conocer cuales son las líneas que definen el perfil 2 que se van a interceptar primero con las líneas que definen el perfil 1.
Variables de Entrada	`L11 As AcadLine, L12 As AcadLine, L21 As AcadLine, L22 As AcadLine, L2m1 As AcadLine, L2m2 As AcadLine`

Código	`Dim pa1 As Variant`
	`Dim pa2 As Variant`
	`Dim pb1 As Variant`
	`Dim pb2 As Variant`
	`Dim d1 As Double`

```
Dim d2          As Double
pa1 = L21.IntersectWith(L11, acExtendBoth)
pa2 = L21.IntersectWith(L12, acExtendBoth)
pb1 = L22.IntersectWith(L11, acExtendBoth)
pb2 = L22.IntersectWith(L12, acExtendBoth)
d1=((pa1(0) - pb2(0))^ 2 + (pa1(1)- pb2(1)) ^ 2 +
(pa1(2) - pb2(2)) ^ 2) ^ 0.5
d2 = ((pa2(0) - pb1(0)) ^ 2 + (pa2(1) - pb1(1)) ^ 2 +
(pa2(2) - pb1(2)) ^ 2) ^ 0.5
If d2 >= d1 Then
Set L2m1 = L22.Copy
Set L2m2 = L21.Copy
End If
If d2 < d1 Then
Set L2m1 = L21.Copy
Set L2m2 = L22.Copy
End If
```

Nombre	Sub Calcula_lineas_que_mandan1()
Descripción	Permite conocer cuales son las líneas que definen el perfil 2 que se van a interceptar primero con las líneas que definen el perfil 1.
Variables de Entrada	`linea2 As AcadLine, L11 As AcadLine, L12 As AcadLine, L1m1 As AcadLine, L1m2 As AcadLine`
Código	

```
Dim p1          As Variant
Dim p2          As Variant
Dim vf          As Integer
Dim i           As Integer
p1 = L11.IntersectWith(linea2, acExtendThisEntity)
p2 = L12.IntersectWith(linea2, acExtendThisEntity)
vf = 0
If VarType(p2) <> vbEmpty Then
For i = LBound(p2) To UBound(p2)
vf = vf + 1
If vf = 1 Then Set L1m1 = L12.Copy
If vf = 1 Then Set L1m2 = L11.Copy
Next
End If
vf = 0
If VarType(p1) <> vbEmpty Then
For i = LBound(p1) To UBound(p1)
vf = vf + 1
If vf = 1 Then Set L1m1 = L11.Copy
If vf = 1 Then Set L1m2 = L12.Copy
Next
End If
```

Nombre	Sub Inicilizar()
Descripción	Define el documento en el cual se está trabajando.
Variables de Entrada	No tiene
Código	flagi = True Set acadApp = GetObject(, "AutoCAD.Application") Set acadDoc = acadApp.ActiveDocument Set mospace = acadDoc.ModelSpace

Nombre	Sub Dibuja_perfil_solo()
Descripción	Permite dibujar todas las líneas que definen el perfil.
Variables de Entrada	Linea1 As AcadLine, vista1 As Double, L1 As Double, t1 As Double, dv1 As Double, L11i As AcadLine, L13i As AcadLine, L11s As AcadLine, L13s As AcadLine, Lineadepernos1i As AcadLine, Lineadepernos1s As AcadLine

Código	Dim v As Variant
	v = Linea1.Offset(-L1)
	Call conv_de_var_en_linea(v(0), L11i)
	L11i.Layer = "Perfiles"
	v = Linea1.Offset(L1)
	Call conv_de_var_en_linea(v(0), L11s)
	L11s.Layer = "Perfiles"
	v = Linea1.Offset(-t1)
	Call conv_de_var_en_linea(v(0), L13i)
	L13i.Layer = "Perfiles"
	v = Linea1.Offset(t1)
	Call conv_de_var_en_linea(v(0), L13s)
	L13s.Layer = "Perfiles"
	v = Linea1.Offset(-L1 + dv1)
	Call conv_de_var_en_linea(v(0), Lineadepernos1i)
	Lineadepernos1i.Layer = "Linea de Pernos"
	v = Linea1.Offset(L1 - dv1)
	Call conv_de_var_en_linea(v(0), Lineadepernos1s)
	Lineadepernos1s.Layer = "Linea de Pernos"
	If vista1 = 2 Then
	L13i.Linetype = "acad_iso02w100"
	L13s.Linetype = "acad_iso02w100"
	End If
	If vista1 = 3 Then L13s.Linetype = "acad_iso02w100"
	If vista1 = 4 Then L13i.Linetype = "acad_iso02w100"
	Lineadepernos1s.Linetype = "acad_iso04w100"
	Lineadepernos1i.Linetype = "acad_iso04w100"

Nombre	Sub Ordena_puntos()
Descripción	Crea un matriz que contiene el centro de los pernos contenidos en el perfil indicado, de manera tal que sea posible poder orientar la acotación.
Variables de Entrada	`linea As AcadLine, radio As Double` `angulo As Double`

Código	`Dim lineaaux As AcadLine`
	`Dim per As AcadObject`
	`Dim perno As AcadCircle`
	`Dim vec(0 To 998) As Double`
	`Dim p(0 To 2) As Double`
	`Dim j As Double`
	`Dim i As Double`
	`Dim k As Double`
	`Dim n As Double`
	`Dim Aux As Double`
	`Dim v As Variant`
	`j = 0`
	`For Each per In acadDoc.ModelSpace`
	`If per.ObjectName = "AcDbCircle" Then`
	`Set perno = per.Copy`
	`v = linea.IntersectWith(perno, acExtendNone)`
	`i = 0`
	`If VarType(v) <> vbEmpty Then`
	`For k = LBound(v) To UBound(v)`
	`i = i + 1`
	`If i = 1 Then`
	`radio = perno.radius`
	`vec(j)=perno.center(0): vec(j + 1) = perno.center(1)`
	`vec(j + 2) = perno.center(2): j = j + 3`
	`per.Delete : perno.Delete`
	`End If`
	`Next k`
	`End If`
	`End If`
	`Next`
	`n = j - 3`
	`For j = 0 To 150`
	`For i = 0 To n - 3 Step 3`
	`If vec(i) > vec(i + 3) Then`
	`Aux = vec(i)`
	`vec(i) = vec(i + 3)`
	`vec(i + 3) = Aux`
	`Aux = vec(i + 1)`
	`vec(i + 1) = vec(i + 4)`
	`vec(i + 4) = Aux`
	`Aux = vec(i + 2)`
	`vec(i + 2) = vec(i + 5)`
	`vec(i + 5) = Aux`

	End If
	Next i
	Next j
	If linea.StartPoint(0) = linea.EndPoint(0) Then
	For j = 0 To 150
	For i = 0 To n - 3 Step 3
	If vec(i + 1) > vec(i + 4) Then
	Aux = vec(i)
	vec(i) = vec(i + 3)
	vec(i + 3) = Aux
	Aux = vec(i + 1)
	vec(i + 1) = vec(i + 4)
	vec(i + 4) = Aux
	Aux = vec(i + 2)
	vec(i + 2) = vec(i + 5)
	vec(i + 5) = Aux
	End If
	Next i
	Next j
	End If
	For i = 0 To n Step 3
	p(0) = vec(i)
	p(1) = vec(i + 1)
	p(2) = vec(i + 2)
	Set perno = acadDoc.ModelSpace.AddCircle(p, radio)
	perno.Layer = "Pernos"
	Next I

Nombre	Sub Limpia_Lineas()
Descripción	Elimina las líneas que se encuentran dibujadas 2 o mas veces.
Variables de Entrada	L As AcadLine
Código	Dim Lo As AcadObject Dim Ln As AcadLine Dim La As AcadLine Dim y As Integer Dim x As Integer y = 0 principio: Call arregloID(L) For Each Lo In acadDoc.ModelSpace x = 0 y = y + 1 If Lo.ObjectName = "AcDbLine" And Lo.ObjectID <> L.ObjectID And Lo.Color = L.Color Then Set Ln = Lo.Copy

```
Call arregloID(Ln)
If Abs(L.Angle - Ln.Angle) <= 0.0001 Then
If Abs(L.StartPoint(0) - Ln.StartPoint(0)) <= 0.0001
And Abs(L.StartPoint(1) - Ln.StartPoint(1)) <= 0.0001
And Abs(L.StartPoint(2) - Ln.StartPoint(2)) <= 0.0001
Then
If L.Length >= Ln.Length Then
Ln.Delete
Lo.Delete
GoTo principio
End If
If Ln.Length > L.Length Then
L.Delete
Set L = Ln.Copy
L.Layer = Ln.Layer
Ln.Delete
Lo.Delete
GoTo principio
End If
End If
If Abs(L.EndPoint(0) - Ln.EndPoint(0)) <= 0.0001 And
Abs(L.EndPoint(1) - Ln.EndPoint(1)) <= 0.0001 And
Abs(L.EndPoint(2) - Ln.EndPoint(2)) <= 0.0001 Then
If L.Length >= Ln.Length Then
Ln.Delete
Lo.Delete
GoTo principio
End If
If Ln.Length > L.Length Then
L.Delete
Set L = Ln.Copy
L.Layer = Ln.Layer
Ln.Delete
Lo.Delete
GoTo principio
End If
End If
If Abs(L.StartPoint(0) - Ln.EndPoint(0)) <= 0.0001
And Abs(L.StartPoint(1) - Ln.EndPoint(1)) <= 0.0001
And Abs(L.StartPoint(2) - Ln.EndPoint(2)) <= 0.0001
Then
Set La = acadDoc.ModelSpace.AddLine(Ln.StartPoint,
L.EndPoint)
La.Layer = L.Layer
L.Delete : Ln.Delete : Lo.Delete
Set L = La.Copy
L.Layer = La.Layer
La.Delete
GoTo principio
End If
If Abs(Ln.StartPoint(0) - L.EndPoint(0)) <= 0.0001
And Abs(Ln.StartPoint(1) - L.EndPoint(1)) <= 0.0001
```

```
And Abs(Ln.StartPoint(2) - L.EndPoint(2)) <= 0.0001
Then
Set La = acadDoc.ModelSpace.AddLine(L.StartPoint,
Ln.EndPoint)
La.Layer = L.Layer
L.Delete : Ln.Delete :  Lo.Delete
Set L = La.Copy
L.Layer = La.Layer
La.Delete
GoTo principio
End If
End If
x = 1
End If
If x = 1 Then
Ln.Delete
End If
Next
```

Nombre	Sub Limpia_pernos()
Descripción	Elimina los círculos (pernos) que se encuentran dibujados 2 o mas veces.
Variables de Entrada	No tiene
Código	

```
Dim v(0 To 10000)     As Long
Dim n                 As Double
Dim i                 As Double
Dim j                 As Double
Dim per1              As AcadObject
Dim per2              As AcadObject
Dim p1                As AcadCircle
Dim p2                As AcadCircle
Dim ch                As Integer
Dim k                 As Integer
n = 0
For Each per1 In acadDoc.ModelSpace
If per1.ObjectName = "AcDbCircle" Then
n = n + 1
v(n) = per1.ObjectID
End If
Next
i = 0
Do While i <> n
i = i + 1 :  j = i
ch = 0    :  n = n - ch
Set per1 = acadDoc.ObjectIdToObject(v(i))
```

	Set p1 = per1.Copy
	Do While j <> n
	j = j + 1
	Set per2 = acadDoc.ObjectIdToObject(v(j))
	Set p2 = per2.Copy
	If per1.ObjectID <> per2.ObjectID Then
	If Abs(p1.center(0) - p2.center(0)) <= 0.2 And
	Abs(p1.center(1) - p2.center(1)) <= 0.2 And
	Abs(p1.center(2) - p2.center(2)) <= 0.2 Then
	per2.Delete
	For k = j To n - 1
	v(k) = v(k + 1)
	Next k
	ch = ch + 1
	End If
	End If
	p2.Delete
	Loop
	p1.Delete
	Loop

Nombre	Sub Visivilidad()
Descripción	Permite la realización de la visibilidad de la conexión.
Variables de Entrada	vista2 As Double, L1m1 As AcadLine, L21 As AcadLine, L22 As AcadLine, LP As AcadLine
Código	Dim L1a As AcadLine Dim L1b As AcadLine Dim L1c As AcadLine Dim L2a1 As AcadLine Dim L2a2 As AcadLine Dim L2b1 As AcadLine Dim L2b2 As AcadLine Dim p1 As Variant Dim p2 As Variant Dim d1 As Double Dim d2 As Double Dim d3 As Double If (vista2 = 6) Or (vista2 = 8) Then Call arregloSE_de_2_lineas(L1m1, L21) Call arregloSE_de_2_lineas(L1m1, L22) p1 = L1m1.IntersectWith(L21, acExtendThisEntity) p2 = L1m1.IntersectWith(L22, acExtendThisEntity) d1 = ((L1m1.StartPoint(0) - p1(0)) ^ 2 + (L1m1.StartPoint(1) - p1(1)) ^ 2 + (L1m1.StartPoint(2) - p1(2)) ^ 2) ^ 0.5 d2 = ((L1m1.EndPoint(0) - p2(0)) ^ 2 +

	(L1m1.EndPoint(1) - p2(1)) ^ 2 + (L1m1.EndPoint(2) - p2(2)) ^ 2) ^ 0.5 d3 = ((p1(0) - p2(0)) ^ 2 + (p1(1) - p2(1)) ^ 2 + (p1(2) - p2(2)) ^ 2) ^ 0.5 Set L2a1 = acadDoc.ModelSpace.AddLine(L21.StartPoint, p1) Set L2a2 = acadDoc.ModelSpace.AddLine(p1, L21.EndPoint) Set L2b1 = acadDoc.ModelSpace.AddLine(L22.StartPoint, p2) Set L2b2 = acadDoc.ModelSpace.AddLine(p2, L22.EndPoint) L2a1.Layer = "Perfiles": L2a2.Layer = "Perfiles" L2b1.Layer = "Perfiles": L2b2.Layer = "Perfiles" L2a1.Color = acRed: L2b1.Color = acRed L2a2.Color = acCyan: L2b2.Color = acCyan LP.Color = acCyan : L21.Delete : L22.Delete End If

Nombre	Sub Detalle_de_separador()
Descripción	Dibuja el detalle de un separador.
Variables de Entrada	linea As AcadLine, np As Integer, db As Integer, dP As Integer, dv As Double, L As Double, t As Double, radio As Double
Código	Dim Ls As AcadLine Dim L1h As AcadLine Dim L2h As AcadLine Dim L3h As AcadLine Dim Lph As AcadLine Dim L1v As AcadLine Dim L2v As AcadLine Dim LdB As AcadLine Dim LdP As AcadLine Dim LacotH As AcadLine Dim LacotV As AcadLine Dim perno As AcadCircle Dim alin As AcadDimAligned Dim v As Variant Dim p(0 To 299) As Double Dim k As Double Dim dist As Double Dim po(0 To 2) As Double Dim pf(0 To 2) As Double Dim punto(0 To 2) As Double Dim texto As String Dim i As Integer Dim j As Integer texto = "Escoja el Punto donde Quiere el Detalle: "

```
Call Test_GetPoint(v, texto)
If Lresult = 2 Then Exit Sub
For i = 0 To 2
punto(i) = v(i)
Next i
po(0) = punto(0)
po(1) = punto(1) + L + 20
pf(0) = punto(0) + 10
pf(1) = punto(1) + L + 20
Set Ls = acadDoc.ModelSpace.AddLine(po, pf)
po(0) = punto(0): pf(0) = punto(0)
po(1) = punto(1): pf(1) = punto(1)
dist = 2 * db + (np - 1) * dP
pf(0) = punto(0) + dist
Set L1h = acadDoc.ModelSpace.AddLine(po, pf)
L1h.Layer = "Perfiles": L1h.Color = acRed
Set v = L1h.Mirror(Ls.StartPoint, Ls.EndPoint)
v = L1h.Offset(-20)
Call conv_de_var_en_linea(v(0), LacotH)
v = L1h.Offset(L)
Call conv_de_var_en_linea(v(0), L2h)
L2h.Layer = "Perfiles": L2h.Color = acRed
Set v = L2h.Mirror(Ls.StartPoint, Ls.EndPoint)
v = L1h.Offset(L - t)
Call conv_de_var_en_linea(v(0), L3h)
L3h.Layer = "Perfiles": L3h.Color = acRed
Set v = L3h.Mirror(Ls.StartPoint, Ls.EndPoint)
v = L1h.Offset(dv)
Call conv_de_var_en_linea(v(0), Lph)
Lph.Layer = "Linea de Pernos": Lph.Color = acYellow:
Lph.Linetype = "acad_iso04w100"
Set v = Lph.Mirror(Ls.StartPoint, Ls.EndPoint)
Set L1v = acadDoc.ModelSpace.AddLine(L1h.StartPoint,
L2h.StartPoint)
L1v.Layer = "Perfiles": L1v.Color = acRed
Set v = L1v.Mirror(Ls.StartPoint, Ls.EndPoint)
v = L1v.Offset(20)
Call conv_de_var_en_linea(v(0), LacotV)
Set L2v = acadDoc.ModelSpace.AddLine(L1h.EndPoint,
L2h.EndPoint)
L2v.Layer = "Perfiles": L2v.Color = acRed
Set v = L2v.Mirror(Ls.StartPoint, Ls.EndPoint)
pf(0) = punto(0) + db
Set LdB = acadDoc.ModelSpace.AddLine(po, pf)
LdB.Move po, Lph.StartPoint
pf(0) = punto(0) + dP
Set LdP = acadDoc.ModelSpace.AddLine(po, pf)
LdP.Move po, LdB.EndPoint
p(0) = Lph.StartPoint(0): p(1) = Lph.StartPoint(1):
p(2) = Lph.StartPoint(2)
k = 3
For i = 1 To np
```

	Set perno = acadDoc.ModelSpace.AddCircle(LdP.StartPoint, radio) p(k) = perno.center(0) p(k + 1) = perno.center(1) p(k + 2) = perno.center(2) k = k + 3 perno.Layer = "Pernos": perno.Color = acBlue Set v = perno.Mirror(Ls.StartPoint, Ls.EndPoint) LdP.Move LdP.StartPoint, LdP.EndPoint Next i p(k) = Lph.EndPoint(0): p(k + 1) = Lph.EndPoint(1): p(k + 2) = Lph.EndPoint(2) k = k + 3 For i = 0 To Fix(k / 3) - 2 po(0) = p(j): po(1) = p(j + 1): po(2) = p(j + 2) pf(0)= p(j + 3): pf(1) = p(j + 4): pf(2) = p(j + 5) j = j + 3 Set alin = acadDoc.ModelSpace.AddDimAligned(po, pf, LacotH.EndPoint) alin.TextHeight = 8 Set v = alin.Mirror(Ls.StartPoint, Ls.EndPoint) Next i Set alin = acadDoc.ModelSpace.AddDimAligned(L1h.StartPoint, Lph.StartPoint, LacotV.EndPoint) alin.TextHeight = 8 Set v = alin.Mirror(Ls.StartPoint, Ls.EndPoint) LacotH.Delete: LacotV.Delete Ls.Delete: LdB.Delete: LdP.Delete

Nombre	Sub prende_layers()
Descripción	Permite que el usuario pueda ver todos los objetos encontrados en plano.
Variables de Entrada	No tiene
Código	Dim lay As AcadLayer Set lay = acadDoc.Layers.Add("0") lay.LayerOn = True Set lay = acadDoc.Layers.Add("Perfiles") lay.LayerOn = True Set lay = acadDoc.Layers.Add("Pernos") lay.LayerOn = True Set lay = acadDoc.Layers.Add("Linea de Cg") lay.LayerOn = True Set lay = acadDoc.Layers.Add("Linea de Pernos") lay.LayerOn = True

Nombre	Sub visibilidad_lista()
Descripción	Cambia los colores de las líneas que por visibilidad no se deben ver en el plano.
Variables de Entrada	`ssetobj1 As AcadSelectionSet,` `ssetobj2 As AcadSelectionSet`

Código	`Dim L As AcadLine`
	`Dim L1 As AcadLine`
	`Dim L2 As AcadLine`
	`Dim d1 As Double`
	`Dim d2 As Double`
	`Dim d3 As Double`
	`Dim p(0 To 5) As Double`
	`Dim pn(0 To 2) As Double`
	`Dim pa(0 To 2) As Double`
	`Dim pb(0 To 2) As Double`
	`Dim po(0 To 2) As Double`
	`Dim pf(0 To 2) As Double`
	`Dim v As Variant`
	`Dim i As Integer`
	`Dim j As Integer`
	`Dim ch As Integer`
	`For Each L1 In ssetobj1`
	`i = 0`
	`For Each L2 In ssetobj2`
	`v = L1.IntersectWith(L2, acExtendNone)`
	`If VarType(v) <> vbEmpty Then`
	`For j = LBound(v) To UBound(v)`
	`p(i) = v(j)`
	`i = i + 1`
	`Next j`
	`End If`
	`Next`
	`i = i - 1`
	`If i = -1 Then`
	`Set L = L1.Copy`
	`L.Layer = "Perfiles"`
	`L.Color = acCyan`
	`ch = 1`
	`End If`
	`If i = 2 Then`
	`For j = 0 To 2`
	`pn(j) = p(j)`
	`Next j`
	`po(0) = L1.StartPoint(0)`
	`po(1) = L1.StartPoint(1)`
	`po(2) = L1.StartPoint(2)`
	`pf(0) = L1.EndPoint(0)`
	`pf(1) = L1.EndPoint(1)`
	`pf(2) = L1.EndPoint(2)`

```
For Each L2 In ssetobj1
If Abs(po(0) - L2.StartPoint(0)) <= 0.001 And
Abs(po(1) - L2.StartPoint(1)) <= 0.001 And Abs(po(2)
- L2.StartPoint(2)) <= 0.001 And L1.ObjectID <>
L2.ObjectID Then
Set L = acadDoc.ModelSpace.AddLine(po, pn)
L.Layer = "Perfiles":  L.Color = acCyan
Set L = acadDoc.ModelSpace.AddLine(pn, pf)
L.Layer = "Perfiles":  L.Color = acRed
ch = 1
End If
If Abs(po(0) - L2.EndPoint(0)) <= 0.001 And Abs(po(1)
- L2.EndPoint(1)) <= 0.001 And Abs(po(2) -
L2.EndPoint(2)) <= 0.001 And L1.ObjectID <>
L2.ObjectID Then
Set L = acadDoc.ModelSpace.AddLine(po, pn)
L.Layer = "Perfiles":  L.Color = acCyan
Set L = acadDoc.ModelSpace.AddLine(pn, pf)
L.Layer = "Perfiles":  L.Color = acRed
ch = 1
End If
If Abs(pf(0) - L2.StartPoint(0)) <= 0.001 And
Abs(pf(1) - L2.StartPoint(1)) <= 0.001 And Abs(pf(2)
- L2.StartPoint(2)) <= 0.001 And L1.ObjectID <>
L2.ObjectID Then
Set L = acadDoc.ModelSpace.AddLine(pf, pn)
L.Layer = "Perfiles":  L.Color = acCyan
Set L = acadDoc.ModelSpace.AddLine(pn, po)
L.Layer = "Perfiles":  L.Color = acRed
ch = 1
End If
If Abs(pf(0) - L2.EndPoint(0)) <= 0.001 And Abs(pf(1)
- L2.EndPoint(1)) <= 0.001 And Abs(pf(2) -
L2.EndPoint(2)) <= 0.001 And L1.ObjectID <>
L2.ObjectID Then
Set L = acadDoc.ModelSpace.AddLine(pf, pn)
L.Layer = "Perfiles":  L.Color = acCyan
Set L = acadDoc.ModelSpace.AddLine(pn, po)
L.Layer = "Perfiles":  L.Color = acRed
ch = 1
End If
Next
If ch = 0 Then
d1 = ((po(0) - pn(0)) ^ 2 + (po(1) - pn(1)) ^ 2 +
(po(2) - pn(2)) ^ 2) ^ 0.5
d2 = ((pn(0) - pf(0)) ^ 2 + (pn(1) - pf(1)) ^ 2 +
(pn(2) - pf(2)) ^ 2) ^ 0.5
If Abs(d1 + d2 - L1.Length <= 0.001) Then
Set L = acadDoc.ModelSpace.AddLine(po, pn)
L.Layer = "Perfiles":  L.Color = acRed
Set L = acadDoc.ModelSpace.AddLine(pn, pf)
L.Layer = "Perfiles":  L.Color = acRed
```

```
Else
Set L = acadDoc.ModelSpace.AddLine(po, pn)
L.Layer = "Perfiles":  L.Color = acRed
Set L = acadDoc.ModelSpace.AddLine(pn, pf)
L.Layer = "Perfiles":  L.Color = acRed
End If
End If
End If
If i = 5 Then
pa(0) = p(0): pa(1) = p(1): pa(2) = p(2)
pb(0) = p(3): pb(1) = p(4): pb(2) = p(5)
po(0) = L1.StartPoint(0)
po(1) = L1.StartPoint(1)
po(2) = L1.StartPoint(2)
pf(0) = L1.EndPoint(0)
pf(1) = L1.EndPoint(1)
pf(2) = L1.EndPoint(2)
d1 = ((po(0) - pa(0)) ^ 2 + (po(1) - pa(1)) ^ 2 +
(po(2) - pa(2)) ^ 2) ^ 0.5
d2 = ((pb(0) - pa(0)) ^ 2 + (pb(1) - pa(1)) ^ 2 +
(pb(2) - pa(2)) ^ 2) ^ 0.5
d3 = ((pf(0) - pb(0)) ^ 2 + (pf(1) - pb(1)) ^ 2 +
(pf(2) - pb(2)) ^ 2) ^ 0.5
If Abs(d1 + d2 + d3 - L1.Length <= 0.001) Then
Set L = acadDoc.ModelSpace.AddLine(po, pa)
L.Layer = "Perfiles":  L.Color = acRed
Set L = acadDoc.ModelSpace.AddLine(pa, pb)
L.Layer = "Perfiles":  L.Color = acCyan
Set L = acadDoc.ModelSpace.AddLine(pb, pf)
L.Layer = "Perfiles":  L.Color = acRed
Else
Set L = acadDoc.ModelSpace.AddLine(po, pb)
L.Layer = "Perfiles":  L.Color = acRed
Set L = acadDoc.ModelSpace.AddLine(pb, pa)
L.Layer = "Perfiles":  L.Color = acCyan
Set L = acadDoc.ModelSpace.AddLine(pa, pf)
L.Layer = "Perfiles":  L.Color = acRed
End If
End If
Next
For Each L In ssetobj1
L.Delete
Next
```

Nombre	Sub Mete_Datos_de_Lineas()
Descripción	Permite que el usuario defina las características de cada perfil (tipo de perfil, vista, diámetro de los pernos y distancia al borde).
Variables de Entrada	No tiene
Código	Dim ssetobj As AcadSelectionSet Dim linea As AcadLine Dim lay As AcadLayer Dim L As Double Dim t As Double Dim x As Double Dim dv As Double Dim vista As Double Dim radio As Double Dim tipo As String Call inicilizar UserForm1.Show Set lay = acadDoc.Layers.Add("Linea de Cg") lay.Color = acGreen tipo = UserForm1.ComboBox1.Text vista = Val(UserForm1.ComboBox2.Text) If UserForm1.ComboBox3 = "1/2" Then radio = (1 / 2) * 25.4 / 2 If UserForm1.ComboBox3 = "5/8" Then radio = (5 / 8) * 25.4 / 2 If UserForm1.ComboBox3 = "3/4" Then radio = (3 / 4) * 25.4 / 2 If UserForm1.ComboBox3 = "7/8" Then radio = (7 / 8) * 25.4 / 2 If UserForm1.ComboBox3 = "1" Then radio = (1) * 25.4 / 2 If UserForm1.ComboBox3 = "1 1/8" Then radio = (9 / 8) * 25.4 / 2 dv = Val(UserForm1.TextBox1.Text) Set ssetobj = acadDoc.SelectionSets.Add("TEST") ssetobj.SelectOnScreen For Each linea In ssetobj Dim DataType(0 To 9) As Integer Dim Data(0 To 9) As Variant Dim reals3(0 To 2) As Double Dim worldPos(0 To 2) As Double linea.Layer = "Linea de Cg" DataType(0) = 1001: Data(0) = "Test_Application" DataType(1) = 1000: Data(1) = tipo DataType(2) = 1003: Data(2) = "0" DataType(3) = 1040: Data(3) = dv DataType(4) = 1041: Data(4) = 1237324938 DataType(5) = 1070: Data(5) = vista DataType(6) = 1071: Data(6) = radio

```
DataType(7) = 1042: Data(7) = 10
reals3(0) = -2.95: reals3(1) = 100: reals3(2) = -20
DataType(8) = 1010: Data(8) = reals3
worldPos(0) = 4: worldPos(1) = 400.99999999:
worldPos(2) = 2.798989
DataType(9) = 1011: Data(9) = worldPos
linea.SetXData DataType, Data
linea.Color = acGreen
Next
ssetobj.Delete
```

Nombre	Sub Unir_Perfiles()
Descripción	Realiza los cortes necesarios para la conexión de perfiles angulares unidos mediante un perno.
Variables de Entrada	No tiene

Código

```
Dim lay1              As AcadLayer
Dim lay2              As AcadLayer
Dim lay3              As AcadLayer
Dim lay4              As AcadLayer
Dim Linea1            As AcadLine
Dim linea2            As AcadLine
Dim linea1v           As AcadLine
Dim linea2v           As AcadLine
Dim Lineadepernos1    As AcadLine
Dim Lineadepernos2    As AcadLine
Dim lineadepernos2x   As AcadLine
Dim L1m1              As AcadLine
Dim L1m2              As AcadLine
Dim L2m1              As AcadLine
Dim L2m2              As AcadLine
Dim L11               As AcadLine
Dim L12               As AcadLine
Dim L13               As AcadLine
Dim L21               As AcadLine
Dim L22               As AcadLine
Dim L23               As AcadLine
Dim Lineap            As AcadLine
Dim L11i              As AcadLine
Dim L11s              As AcadLine
Dim L13i              As AcadLine
Dim L13s              As AcadLine
Dim Lineadepernos1i   As AcadLine
Dim Lineadepernos1s   As AcadLine
Dim obj               As AcadObject
Dim r                 As AcadCircle
Dim perno(0 To 2)     As Double
```

```
Dim L1                As Double
Dim L2                As Double
Dim t1                As Double
Dim t2                As Double
Dim vista1            As Double
Dim vista2            As Double
Dim dv1               As Double
Dim dv2               As Double
Dim distpos           As Double
Dim radio             As Double
Dim radio1            As Double
Dim radio2            As Double
Dim nombre1           As String
Dim nombre2           As String
Dim p                 As Variant
Dim entry             As Variant
Dim v                 As Variant
Dim vf                As Integer
Dim i                 As Integer
Call inicilizar
acadDoc.StartUndoMark
On Error Resume Next
Retry:
Set lay1 = acadDoc.Layers.Add("Perfiles")
lay1.Color = acRed
Set lay2 = acadDoc.Layers.Add("Linea de Pernos")
lay2.Color = acYellow
Set lay3 = acadDoc.Layers.Add("0")
Set lay4 = acadDoc.Layers.Add("Pernos")
lay4.Color = acBlue
Call tipo_de_lineas
Call Saca_datos(linea1v, L1, t1, dv1, vista1, radio1,
"A")
If Lresult = 2 Then Exit Sub
Call Saca_datos(linea2v, L2, t2, dv2, vista2, radio2,
"B")
If Lresult = 2 Then Exit Sub
If radio1 = radio2 Then radio = radio1
If radio1 <> radio2 Then
MsgBox ("Los Pernos de Los Perfiles tiene distintos
Diametros")
Exit Sub
End If
Call recorta_linea_1(linea1v, linea2v, Linea1)
Call recorta_linea_2(linea1v, linea2v, linea2)
Call arregloID(Linea1)
Call arregloID(linea2)
If vista2 < 5 Or vista2 > 8 Then
MsgBox " Vista del Perfil a Unir Incorrecta (B)"
UserForm1.Caption = " Introduccion de Datos del
Perfil a Unir (B)"
Call Mete_Datos_de_Lineas
```

```
UserForm1.Caption = " Introduccion de Datos"
End If
Call dibuja_perfil(Linea1, linea2, vista1, L11, L12,
L13, Lineadepernos1, L1, t1, dv1, L11i, L13i, L11s,
L13s, Lineadepernos1i, Lineadepernos1s)
If vista2 = 7 Or vista2 = 8 Then
L2 = -L2
t2 = -t2
dv2 = -dv2
End If
v = linea2.Offset(dv2 - dv2)
Call conv_de_var_en_linea(v(0), Lineadepernos2)
v = linea2.Offset(dv2)
Call conv_de_var_en_linea(v(0), L21)
v = linea2.Offset(dv2 - L2)
Call conv_de_var_en_linea(v(0), L22)
v = linea2.Offset(dv2 - L2 + t2)
Call conv_de_var_en_linea(v(0), L23)
L21.Layer = "Perfiles": L22.Layer = "Perfiles":
L23.Layer = "Perfiles"
Lineadepernos2.Layer = "Linea de Pernos"
Call extiende_linea_a_linea(Lineadepernos1,
Lineadepernos2, perno())
Lineadepernos2.Layer = "Linea de Pernos"
Call arregloSE_de_linea_con_punto(L21, perno())
Call arregloSE_de_linea_con_punto(L22, perno())
L21.Layer = "Perfiles"
L22.Layer = "Perfiles"
Set Lineap = acadDoc.ModelSpace.AddLine(L21.EndPoint,
L22.EndPoint)
Lineap.Layer = "Perfiles"
v = Lineap.IntersectWith(Lineadepernos2,
acExtendOtherEntity)
Lineap.Move v, perno
Dim str As String
vf = 0
v = Lineap.IntersectWith(L13, acExtendNone)
If VarType(v) <> vbEmpty Then
For i = LBound(v) To UBound(v)
vf = vf + 1
If vf = 1 Then MsgBox "No Cabe el Perfil"
Exit Sub
Next
acadDoc.EndUndoMark
End If
Set r = acadDoc.ModelSpace.AddCircle(perno, radio)
r.Layer = "Pernos"
Call calcula_lineas_que_mandan1(Lineadepernos2, L11,
L13, L1m1, L1m2)
Call calcula_lineas_que_mandan2(L11, L13, L21, L22,
L2m1, L2m2)
Call distancias_posible(linea2, perno, L1m1, L1m2,
```

```
L2m1, L2m2, Lineap, distpos, radio)
Set lineadepernos2x = Lineadepernos2.Copy
Call extiende_linea_una_distancia(lineadepernos2x,
distpos)
linea2.Delete: L21.Delete: L22.Delete: L23.Delete
Call arregloID(lineadepernos2x)
v = lineadepernos2x.Offset(0)
Call conv_de_var_en_linea(v(0), linea2)
v = linea2.Offset(dv2)
Call conv_de_var_en_linea(v(0), L21)
v = linea2.Offset(dv2 - L2)
Call conv_de_var_en_linea(v(0), L22)
v = linea2.Offset(dv2 - L2 + t2)
Call conv_de_var_en_linea(v(0), L23)
If vista2 = 6 Or vista2 = 8 Then
L23.Linetype = "acad_iso02w100"
L23.Color = acCyan
End If
Call arregloSE_de_linea_con_punto(lineadepernos2x,
perno())
Lineap.Move perno, lineadepernos2x.EndPoint
Lineadepernos1.Layer = "Linea de Pernos"
lineadepernos2x.Layer = "Linea de Pernos"
Lineadepernos1.Linetype = "acad_iso04w100"
lineadepernos2x.Linetype = "acad_iso04w100"
L21.Layer = "Perfiles":  L22.Layer = "Perfiles":
L23.Layer = "Perfiles"
Call Visivilidad(vista2, L1m1, L21, L22, Lineap)
L1m1.Delete: L1m2.Delete: L2m1.Delete: L2m2.Delete
Linea1.Delete: linea2.Delete: Lineadepernos2.Delete
i = 0
For Each obj In acadDoc.ModelSpace
If obj.ObjectName = "AcDbCircle" And obj.Layer =
"Pernos" Then
Set r = obj.Copy
If Abs(r.center(0) - perno(0)) <= 0.001 And
(Abs(r.center(1) - perno(1)) <= 0.001) And
(Abs(r.center(2) - perno(2)) <= 0.001) Then
i = i + 1
If i = 2 Then
If vista1 < 5 Then
L12.Delete:   L11i.Delete
L11s.Delete:  L13i.Delete
L13s.Delete:  Lineadepernos1i.Delete
L11.Delete:   L12.Delete
L13.Delete:   Lineadepernos1s.Delete
End If
If vista1 > 4 Then
L12.Delete:   L11.Delete
L12.Delete:   L13.Delete
Lineadepernos1.Delete
End If
```

```
obj.Delete
End If
End If
r.Delete
End If
Next
```

Nombre	Sub Union_para_Planchas()
Descripción	Realiza los cortes necesarios para la conexión de perfiles angulares unidos mediante una plancha pero sin dibujar la plancha.
Variables de Entrada	No tiene
Código	Dim linea1v As AcadLine Dim linea2v As AcadLine Dim Linea1 As AcadLine Dim linea2 As AcadLine Dim linea22 As AcadLine Dim La As AcadLine Dim Lb As AcadLine Dim LC As AcadLine Dim L1m1 As AcadLine Dim L1m2 As AcadLine Dim L2m1 As AcadLine Dim L2m2 As AcadLine Dim p(0 To 2) As Double Dim po(0 To 2) As Double Dim pf(0 To 2) As Double Dim np As Integer Dim db As Integer Dim dPer As Integer Dim dP As Integer Dim L1 As Double Dim t1 As Double Dim dv1 As Double Dim vista1 As Double Dim radio1 As Double Dim L2 As Double Dim t2 As Double Dim dv2 As Double Dim vista2 As Double Dim radio2 As Double Dim L11 As AcadLine Dim L12 As AcadLine Dim L13 As AcadLine Dim Lineadepernos1 As AcadLine

```
Dim L11i                    As AcadLine
Dim L13i                    As AcadLine
Dim Lineadepernos1i        As AcadLine
Dim L11s                    As AcadLine
Dim L13s                    As AcadLine
Dim Lineadepernos1s        As AcadLine
Dim L21                     As AcadLine
Dim L22                     As AcadLine
Dim L23                     As AcadLine
Dim Lineadepernos2         As AcadLine
Dim L21i                    As AcadLine
Dim L23i                    As AcadLine
Dim Lineadepernos2i        As AcadLine
Dim L21s                    As AcadLine
Dim L23s                    As AcadLine
Dim Lineadepernos2s        As AcadLine
Dim LP                      As AcadLine
Dim lp1                     As AcadLine
Dim lp2                     As AcadLine
Dim texto                   As String
Dim lay1                    As AcadLayer
Dim lay2                    As AcadLayer
Dim lay4                    As AcadLayer
Dim lay5                    As AcadLayer
Dim angulo                  As Double
Dim v                       As Variant
Dim i                       As Integer
Call inicilizar
Set lay1 = acadDoc.Layers.Add("Perfiles")
lay1.Color = acRed
Set lay2 = acadDoc.Layers.Add("Linea de Pernos")
lay2.Color = acYellow
Set lay4 = acadDoc.Layers.Add("0")
Set lay5 = acadDoc.Layers.Add("Pernos")
lay5.Color = acBlue
Call tipo_de_lineas
On Error Resume Next
Retry:
Call Saca_datos(linea1v, L1, t1, dv1, vista1, radio1,
"A")
If Lresult = 2 Then Exit Sub
Call Saca_datos(linea2v, L2, t2, dv2, vista2, radio2,
"B")
If Lresult = 2 Then Exit Sub
texto = "Numero de Pernos por Perfil (B): "
Call TextGetNumber(np, texto)
If Lresult = 2 Then Exit Sub
texto = "Distancia entre Pernos (B): "
Call TextGetNumber(dP, texto)
If Lresult = 2 Then Exit Sub
texto = "Distancia entre el borde de la plancha y el
perno (B): "
```

```
Call TextGetNumber(db, texto)
If Lresult = 2 Then Exit Sub
texto = "Distancia entre el perfil A y B
(Perpendicular al A : "
Call TextGetNumber(dPer, texto)
If Lresult = 2 Then Exit Sub
Call recorta_linea_1(linea1v, linea2v, Linea1)
Call recorta_linea_2(linea1v, linea2v, linea2)
Call arregloID(Linea1)
Call arregloID(linea2)
Call dibuja_perfil(Linea1, linea2, vista1, L11, L12,
L13, Lineadepernos1, L1, t1, dv1, L11i, L13i, L11s,
L13s, Lineadepernos1i, Lineadepernos1s)
If vista1 < 5 Then
Set L1m1 = L11.Copy
Set L1m2 = L12.Copy
End If
If vista1 > 4 Then
Call calcula_lineas_que_mandan1(linea2v, L11, L12,
L1m1, L1m2)
End If
pf(0) = dPer
Set La = acadDoc.ModelSpace.AddLine(po, pf)
Set Lb = acadDoc.ModelSpace.AddLine(L1m1.StartPoint,
L1m2.StartPoint)
angulo = Lb.Angle
La.Move pf, Lb.StartPoint
La.Rotate Lb.StartPoint, angulo
Set LC = L1m1.Copy
LC.Move L1m1.StartPoint, La.StartPoint
Call dibuja_perfil(linea2, Linea1, vista2, L21, L22,
L23, Lineadepernos2, L2, t2, dv2, L21i, L23i, L21s,
L23s, Lineadepernos2i, Lineadepernos2s)
Call calcula_lineas_que_mandan2(L1m1, L1m2, L21, L22,
L2m1, L2m2)
Call arregloSE_de_2_lineas(Linea1, linea2)
Call arregloSE_de_2_lineas(Linea1, L21)
Call arregloSE_de_2_lineas(Linea1, L22)
Call arregloSE_de_2_lineas(Linea1, L23)
Call arregloSE_de_2_lineas(Linea1, Lineadepernos2)
Set LP = acadDoc.ModelSpace.AddLine(L2m1.StartPoint,
L2m2.StartPoint)
v = L2m1.IntersectWith(LC, acExtendBoth)
For i = 0 To 2
p(i) = v(i)
Next i
LP.Move L2m1.StartPoint, p
Call Borra_Perfil(L21, L22, L23, Lineadepernos2,
L21i, L23i, L21s, L23s, Lineadepernos2i,
Lineadepernos2s)
v = linea2.IntersectWith(LP, acExtendBoth)
Set linea22 =
```

	acadDoc.ModelSpace.AddLine(linea2.StartPoint, v)
	Call arregloID(linea22)
	Call dibuja_perfil(linea22, Linea1, vista2, L21, L22, L23, Lineadepernos2, L2, t2, dv2, L21i, L23i, L21s, L23s, Lineadepernos2i, Lineadepernos2s)
	If vista2 < 5 Then
	LP.Delete
	Call arregloSE_de_2_lineas(Linea1, Lineadepernos2s)
	Call Dibuja_Pernos(Lineadepernos2s, np, db, dP, radio2)
	Call arregloSE_de_2_lineas(Linea1, Lineadepernos2i)
	Call Dibuja_Pernos(Lineadepernos2i, np, db, dP, radio2)
	Set LP = acadDoc.ModelSpace.AddLine(L21s.EndPoint, L21i.EndPoint)
	End If
	If vista2 >= 5 Then
	Call arregloSE_de_2_lineas(Linea1, Lineadepernos2)
	Call Dibuja_Pernos(Lineadepernos2, np, db, dP, radio2)
	End If
	LP.Layer = "Perfiles"
	La.Delete: Lb.Delete: LC.Delete
	Linea1.Delete: linea2.Delete: linea22.Delete
	L1m1.Delete: L1m2.Delete: L2m1.Delete: L2m2.Delete

Nombre	Sub Separadores()
Descripción	Muestra la forma para que el usuario defina que tipo de separador quiere dibujar.
Variables de Entrada	No tiene
Código	Call inicilizar UserForm2.Show If UserForm2.OptionButton1 Then Call Separador_a_partir_de_un_Perno If UserForm2.OptionButton2 Then Call Separador_Centrado If UserForm2.OptionButton3 Then Call Separador_de_perfiles If Lresult = 2 Then Exit Sub

Nombre	Sub Separador_a_partir_de_un_Perno()
Descripción	Dibuja un separador a partir de una conexión ya realizada (mediante el módulo Unir_Perfiles) .
Variables de Entrada	No tiene
Código	Dim perno As AcadCircle Dim pernoAux As AcadCircle Dim linea As AcadLine Dim L11i As AcadLine Dim L11s As AcadLine Dim L13i As AcadLine Dim L13s As AcadLine Dim Lineadepernos1i As AcadLine Dim Lineadepernos1s As AcadLine Dim LdP As AcadLine Dim lp1 As AcadLine Dim lp2 As AcadLine Dim LpA1 As AcadLine Dim LpB1 As AcadLine Dim LpA2 As AcadLine Dim LpB2 As AcadLine Dim LdB As AcadLine Dim v As Variant Dim po(0 To 2) As Double Dim pf(0 To 2) As Double Dim L As Double Dim t As Double Dim x As Double Dim dv As Double Dim vista As Double Dim radio As Double Dim dP As Integer Dim db As Integer Dim pos As Integer Dim np As Integer Dim NpD As Integer Dim NpI As Integer Dim lay1 As AcadLayer Dim lay2 As AcadLayer Dim lay3 As AcadLayer Dim lay4 As AcadLayer Dim lay5 As AcadLayer Dim texto As String Dim vf As Integer Dim i As Integer Call inicilizar Set lay1 = acadDoc.Layers.Add("Perfiles") lay1.Color = acRed Set lay2 = acadDoc.Layers.Add("Linea de Pernos")

```
lay2.Color = acYellow
Set lay3 = acadDoc.Layers.Add("Linea de Cg")
Set lay4 = acadDoc.Layers.Add("0")
Set lay5 = acadDoc.Layers.Add("Pernos")
lay5.Color = acBlue
On Error Resume Next
Retry:
Repite:
lay1.LayerOn = False: lay2.LayerOn = False
lay3.LayerOn = False: lay4.LayerOn = False
texto = "Seleccione el Perno : "
Call Test_GetEntity(perno, texto)
Call prende_layers
If Lresult = 2 Then Exit Sub
If perno.ObjectName <> "AcDbCircle" Then GoTo Repite
Call Saca_datos(linea, L, t, dv, vista, radio, "")
If Lresult = 2 Then Exit Sub
If vista > 4 Then
MsgBox "Vista Incorrecta"
Exit Sub
End If
Call arregloID(linea)
Do While pos < 1 Or pos > 3
texto= "Posicion del Perno <1-Izq, 2-Cen, 3-Der>: "
Call TextGetNumber(pos, texto)
If Lresult = 2 Then Exit Sub
Loop
If pos = 1 Or pos = 3 Then
texto = "Numero de Pernos por Ala: "
Call TextGetNumber(np, texto)
If Lresult = 2 Then Exit Sub
End If
If pos = 2 Then
np = 2
Do While np / 2 = Fix(np / 2)
texto = "Numero de Pernos por Ala: "
Call TextGetNumber(np, texto)
If Lresult = 2 Then Exit Sub
If np/2 = Fix(np / 2) Then MsgBox ("Numero Impar")
Loop
End If
texto = "Distancia entre Pernos : "
Call TextGetNumber(dP, texto)
If Lresult = 2 Then Exit Sub
texto = "Distancia al Borde : "
Call TextGetNumber(db, texto)
If Lresult = 2 Then Exit Sub
Call dibuja_perfil_solo(linea, vista, L, t, dv, L11i,
L13i, L11s, L13s, Lineadepernos1i, Lineadepernos1s)
Set pernoAux = perno.Mirror(linea.StartPoint,
linea.EndPoint)
Set lp1 = acadDoc.ModelSpace.AddLine(L11i.StartPoint,
```

```
linea.StartPoint)
Set lp2 =
acadDoc.ModelSpace.AddLine(linea.StartPoint,
L11s.StartPoint)
lp1.Layer = "Perfiles": lp2.Layer = "Perfiles"
vf = 0
v = Lineadepernos1i.IntersectWith(perno,
acExtendNone)
If VarType(v) <> vbEmpty Then
For i = LBound(v) To UBound(v)
vf = 1
v = Lineadepernos1i.IntersectWith(lp1, acExtendNone)
Next
End If
If vf = 0 Then v = Lineadepernos1s.IntersectWith(lp2,
acExtendNone)
If L13i.Linetype <> "ACAD_ISO02W100" Then lp1.Color =
acYellow
If L13s.Linetype <> "ACAD_ISO02W100" Then lp2.Color =
acYellow
If pos = 1 Then NpD = np
If pos = 2 Then
NpD = Fix(np / 2) + 1
NpI = Fix(np / 2) + 1
End If
If pos = 3 Then NpI = np
pf(0) = dP
Set LdP = acadDoc.ModelSpace.AddLine(po, pf)
LdP.Move po, perno.center
LdP.Rotate LdP.StartPoint, linea.Angle
For i = 2 To NpD
Set pernoAux =
acadDoc.ModelSpace.AddCircle(LdP.EndPoint,
perno.radius)
pernoAux.Layer = "Pernos"
LdP.Move LdP.StartPoint, LdP.EndPoint
Set pernoAux = pernoAux.Mirror(linea.StartPoint,
linea.EndPoint)
Next
pf(0) = db
Set LdB = acadDoc.ModelSpace.AddLine(po, pf)
If pos <> 3 Then
Set pernoAux = pernoAux.Mirror(linea.StartPoint,
linea.EndPoint)
LdB.Move po, pernoAux.center
pernoAux.Delete
End If
If pos = 3 Then LdB.Move po, perno.center
LdB.Rotate LdB.StartPoint, linea.Angle
Set LpA1 = lp1.Copy
Set LpB1 = lp2.Copy
LpA1.Move v, LdB.EndPoint
```

```
LpB1.Move v, LdB.EndPoint
LdP.Delete
LdB.Delete
pf(0) = dP
Set LdP = acadDoc.ModelSpace.AddLine(po, pf)
LdP.Move po, perno.center
LdP.Rotate LdP.StartPoint, linea.Angle
LdP.Move LdP.EndPoint, LdP.StartPoint
For i = 2 To NpI
LdP.Move LdP.EndPoint, LdP.StartPoint
Set pernoAux =
acadDoc.ModelSpace.AddCircle(LdP.EndPoint,
perno.radius)
pernoAux.Layer = "Pernos"
Set pernoAux = pernoAux.Mirror(linea.StartPoint,
linea.EndPoint)
Next
pf(0) = db
Set LdB = acadDoc.ModelSpace.AddLine(po, pf)
If pos <> 1 Then
Set pernoAux = pernoAux.Mirror(linea.StartPoint,
linea.EndPoint)
LdB.Move po, pernoAux.center
pernoAux.Delete
End If
If pos = 1 Then LdB.Move po, perno.center
LdB.Rotate LdB.StartPoint, linea.Angle
LdB.Move LdB.EndPoint, LdB.StartPoint
Set LpA2 = lp1.Copy
Set LpB2 = lp2.Copy
LpA2.Move v, LdB.StartPoint
LpB2.Move v, LdB.StartPoint
If vista = 1 Then
LpA1.Color = acCyan
LpA2.Color = acCyan
LpB1.Color = acCyan
LpB2.Color = acCyan
End If
If vista = 2 Then
LpA1.Color = acCyan:  LpA2.Color = acCyan
End If
If vista = 4 Then
LpB1.Color = acCyan:  LpB2.Color = acCyan
End If
Call detalle_de_separador(linea, np, db, dP, dv, L,
t, radio)
LdP.Delete: lp1.Delete: lp2.Delete
L11i.Delete: L11s.Delete: L13i.Delete: L13s.Delete
Lineadepernos1i.Delete: Lineadepernos1s.Delete
```

Nombre	Sub Separador_Centrado()
Descripción	Dibuja un separador ubicado en el medio de dos conexiones ya realizadas (los perfiles no están separados).
Variables de Entrada	No tiene
Código	Dim perno1 As AcadEntity Dim perno2 As AcadEntity Dim linea As AcadEntity Dim LP As AcadLine Dim lpa As AcadLine Dim lpb As AcadLine Dim lp1 As AcadLine Dim lp2 As AcadLine Dim L11i As AcadLine Dim L11s As AcadLine Dim L13i As AcadLine Dim L13s As AcadLine Dim Lineadepernos1i As AcadLine Dim Lineadepernos1s As AcadLine Dim LdP As AcadLine Dim LdB As AcadLine Dim lay1 As AcadLayer Dim lay2 As AcadLayer Dim lay3 As AcadLayer Dim lay4 As AcadLayer Dim lay5 As AcadLayer Dim v As Variant Dim p1 As Variant Dim p2 As Variant Dim po(0 To 2) As Double Dim pm(0 To 2) As Double Dim pf(0 To 2) As Double Dim L As Double Dim t As Double Dim dv As Double Dim vista As Double Dim radio As Double Dim db As Integer Dim dP As Integer Dim i As Integer Dim np As Integer Dim texto As String Dim n As Double Dim k As Double Call inicilizar Set lay1 = acadDoc.Layers.Add("Perfiles") lay1.Color = acRed Set lay2 = acadDoc.Layers.Add("Linea de Pernos") lay2.Color = acYellow

```
Set lay3 = acadDoc.Layers.Add("Linea de Cg")
Set lay4 = acadDoc.Layers.Add("0")
Set lay5 = acadDoc.Layers.Add("Pernos")
lay5.Color = acBlue
On Error Resume Next
Retry:
Repite:
lay1.LayerOn = False: lay2.LayerOn = False
lay3.LayerOn = False: lay4.LayerOn = False
texto = "Selecciona el Perno A :"
Call Test_GetEntity(perno1, texto)
Call prende_layers
If Lresult = 2 Then Exit Sub
If perno1.ObjectName <> "AcDbCircle" Then GoTo Repite
Repite2:
lay1.LayerOn = False: lay2.LayerOn = False
lay3.LayerOn = False: lay4.LayerOn = False
texto = "Selecciona el Perno B :"
Call Test_GetEntity(perno2, texto)
Call prende_layers
If Lresult = 2 Then Exit Sub
If perno2.ObjectName <> "AcDbCircle" Then GoTo
Repite2
Call Saca_datos(linea, L, t, dv, vista, radio, "")
If Lresult = 2 Then Exit Sub
If vista > 4 Then
MsgBox "Vista Incorrecta"
Exit Sub
End If
texto = "Numero de Pernos por Ala de Perfil :"
Call TextGetNumber(np, texto)
If Lresult = 2 Then Exit Sub
texto = "Distancia entre Pernos : "
Call TextGetNumber(dP, texto)
If Lresult = 2 Then Exit Sub
texto = "Distancia al Borde : "
Call TextGetNumber(db, texto)
If Lresult = 2 Then Exit Sub
Call arregloID(linea)
Call dibuja_perfil_solo(linea, vista, L, t, dv, L11i,
L13i, L11s, L13s, Lineadepernos1i, Lineadepernos1s)
Set lp1 = acadDoc.ModelSpace.AddLine(L11i.StartPoint,
L11s.StartPoint)
Set lp2 = acadDoc.ModelSpace.AddLine(L11i.StartPoint,
L11s.StartPoint)
lp1.Move L11i.StartPoint, perno1.center
lp2.Move L11i.StartPoint, perno2.center
p1 = lp1.IntersectWith(linea, acExtendThisEntity)
p2 = lp2.IntersectWith(linea, acExtendThisEntity)
lp1.Delete: lp2.Delete
For i = 0 To 2
pm(i) = (p1(i) + p2(i)) / 2
```

```
Next i
Set LP = acadDoc.ModelSpace.AddLine(L11i.StartPoint,
L11s.StartPoint)
LP.Move linea.StartPoint, pm
Call arregloID(LP)
p1 = LP.IntersectWith(Lineadepernos1i, acExtendNone)
If np / 2 = Fix(np / 2) Then
pf(0) = dP / 2
Set LdB = acadDoc.ModelSpace.AddLine(po, pf)
LdB.Move po, p1
LdB.Rotate LdB.StartPoint, linea.Angle
Set perno1 =
acadDoc.ModelSpace.AddCircle(LdB.EndPoint, radio)
perno1.Layer = "Pernos"
Set perno2 = perno1.Mirror(linea.StartPoint,
linea.EndPoint)
Set v = perno2.Mirror(LP.StartPoint, LP.EndPoint)
Set v = perno1.Mirror(LP.StartPoint, LP.EndPoint)
pf(0) = dP
Set LdP = acadDoc.ModelSpace.AddLine(po, pf)
LdP.Move po, LdB.EndPoint
LdB.Delete
LdP.Rotate LdP.StartPoint, linea.Angle
n = np / 2 - 1
k = dP / 2 + n * dP + db
End If
If np / 2 <> Fix(np / 2) Then
Set perno1 = acadDoc.ModelSpace.AddCircle(p1, radio)
perno1.Layer = "Pernos"
Set perno2 = perno1.Mirror(linea.StartPoint,
linea.EndPoint)
Set v = perno2.Mirror(LP.StartPoint, LP.EndPoint)
Set v = perno1.Mirror(LP.StartPoint, LP.EndPoint)
pf(0) = dP
Set LdP = acadDoc.ModelSpace.AddLine(po, pf)
LdP.Move po, p1
LdP.Rotate p1, linea.Angle
n = Fix(np / 2)
k = n * dP + db
End If
For i = 1 To n
Set perno1 =
acadDoc.ModelSpace.AddCircle(LdP.EndPoint, radio)
perno1.Layer = "Pernos"
Set perno2 = perno1.Mirror(linea.StartPoint,
linea.EndPoint)
Set v = perno2.Mirror(LP.StartPoint, LP.EndPoint)
Set v = perno1.Mirror(LP.StartPoint, LP.EndPoint)
LdP.Move LdP.StartPoint, LdP.EndPoint
Next i
Set lpa = acadDoc.ModelSpace.AddLine(L11i.StartPoint,
linea.StartPoint)
```

```
p1 = lpa.IntersectWith(Lineadepernos1i, acExtendNone)
p2 = LP.IntersectWith(Lineadepernos1i, acExtendNone)
lpa.Move p1, p2
Set lpb =
acadDoc.ModelSpace.AddLine(linea.StartPoint,
L11s.StartPoint)
p1 = lpb.IntersectWith(Lineadepernos1s, acExtendNone)
p2 = LP.IntersectWith(Lineadepernos1s, acExtendNone)
lpb.Move p1, p2
lpa.Layer = "Perfiles": lpb.Layer = "Perfiles"
If vista = "1" Then
lpa.Color = acCyan
lpb.Color = acCyan
End If
If vista = "2" Then
lpa.Color = acRed
lpb.Color = acRed
End If
If vista = "3" Then
lpa.Color = acCyan
lpb.Color = acRed
End If
If vista = "4" Then
lpa.Color = acRed
lpb.Color = acCyan
End If
v = lpa.Offset(k)
Call conv_de_var_en_linea(v(0), lp1)
lp1.Layer = lpa.Layer
lp1.Color = lpa.Color
Set v = lp1.Mirror(LP.StartPoint, LP.EndPoint)
v = lpb.Offset(k)
Call conv_de_var_en_linea(v(0), lp1)
lp1.Layer = lpb.Layer
lp1.Color = lpb.Color
Set v = lp1.Mirror(LP.StartPoint, LP.EndPoint)
lpa.Delete: lpb.Delete
LP.Delete: LdB.Delete: LdP.Delete
L11i.Delete: L11s.Delete: L13i.Delete: L13s.Delete
Lineadepernos1s.Delete: Lineadepernos1i.Delete
Call detalle_de_separador(linea, np, db, dP, dv, L,
t, radio)
```

Nombre	Sub Separador_de_perfiles()
Descripción	Dibuja un separador ubicado en el medio de dos conexiones ya realizadas (los perfiles están separados y unidos por el separador).
Variables de Entrada	No Tiene
Código	Dim lay1 As AcadLayer Dim lay2 As AcadLayer Dim lay3 As AcadLayer Dim lay4 As AcadLayer Dim lay5 As AcadLayer Dim perno1 As AcadEntity Dim perno2 As AcadEntity Dim linea As AcadEntity Dim lp1 As AcadLine Dim lp2 As AcadLine Dim L11i As AcadLine Dim L11s As AcadLine Dim L13i As AcadLine Dim L13s As AcadLine Dim Lineadepernos1i As AcadLine Dim Lineadepernos1s As AcadLine Dim Lpernos As AcadLine Dim LdP As AcadLine Dim LsP As AcadLine Dim LacotH As AcadLine Dim LacotV As AcadLine Dim Mirr As AcadLine Dim Ln As AcadLine Dim LdB1 As AcadLine Dim LdB2 As AcadLine Dim lpa As AcadLine Dim lpb As AcadLine Dim alin As AcadDimAligned Dim punto(0 To 2) As Double Dim po(0 To 2) As Double Dim pf(0 To 2) As Double Dim pm(0 To 2) As Double Dim L As Double Dim t As Double Dim dv As Double Dim vista As Double Dim radio As Double Dim SP As Integer Dim np As Integer Dim dP As Integer Dim db As Integer Dim inter As String Dim v As Variant Dim p1 As Variant

```
Dim p2                      As Variant
Dim i                       As Integer
Dim texto                   As String
Dim x                       As Integer
Call inicilizar
Set lay1 = acadDoc.Layers.Add("Perfiles")
lay1.Color = acRed
Set lay2 = acadDoc.Layers.Add("Linea de Pernos")
lay2.Color = acYellow
Set lay3 = acadDoc.Layers.Add("Linea de Cg")
Set lay4 = acadDoc.Layers.Add("0")
Set lay5 = acadDoc.Layers.Add("Pernos")
lay5.Color = acBlue
On Error Resume Next
Retry:
Repite:
lay1.LayerOn = False: lay2.LayerOn = False
lay3.LayerOn = False: lay4.LayerOn = False
texto = "Selecciona el Perno A :"
Call Test_GetEntity(perno1, texto)
Call prende_layers
If Lresult = 2 Then Exit Sub
If perno1.ObjectName <> "AcDbCircle" Then GoTo Repite
Repite2:
lay1.LayerOn = False: lay2.LayerOn = False
lay3.LayerOn = False: lay4.LayerOn = False
texto = "Selecciona el Perno B :"
Call Test_GetEntity(perno2, texto)
Call prende_layers
If Lresult = 2 Then Exit Sub
If perno2.ObjectName <> "AcDbCircle" Then GoTo
Repite2
Call Saca_datos(linea, L, t, dv, vista, radio, "")
If Lresult = 2 Then Exit Sub
If vista > 4 Then
MsgBox "Vista Incorrecta"
Exit Sub
End If
texto = "Escoja el Punto donde Quiere el Detalle: "
Call Test_GetPoint(v, texto)
If Lresult = 2 Then Exit Sub
For i = 0 To 2
punto(i) = v(i)
Next i
x = 0
Do While x = 0
inter = acadDoc.Utility.GetString(False, "Pernos
Intercalados (S/N) :")
inter = UCase(inter)
If inter = "S" Or inter = "N" Then x = 1
Loop
texto = "Separacion de los Perfiles :"
```

```
Call TextGetNumber(SP, texto)
If Lresult = 2 Then Exit Sub
x = 0
Do While x = 0
texto = "Numero de Pernos por Ala de Perfil :"
Call TextGetNumber(np, texto)
If Lresult = 2 Then Exit Sub
If (np / 2 = Fix(np / 2)) Then x = 1
If (np / 2 <> Fix(np / 2)) Then MsgBox ("El Numero de
Pernos debe ser Par")
Loop
texto = "Distancia entre Pernos : "
Call TextGetNumber(dP, texto)
If Lresult = 2 Then Exit Sub
texto = "Distancia al Borde : "
Call TextGetNumber(db, texto)
If Lresult = 2 Then Exit Sub
ReDim LPer(1 To 5) As AcadLine
ReDim pernoV(0 To 2 * np) As AcadLine
ReDim perno(1 To 2 * np) As AcadCircle
Call arregloID(linea)
Call dibuja_perfil_solo(linea, vista, L, t, dv, L11i,
L13i, L11s, L13s, Lineadepernos1i, Lineadepernos1s)
Set lp1 = acadDoc.ModelSpace.AddLine(L11i.StartPoint,
L11s.StartPoint)
Set lp2 = acadDoc.ModelSpace.AddLine(L11i.StartPoint,
L11s.StartPoint)
lp1.Move L11i.StartPoint, perno1.center
lp2.Move L11i.StartPoint, perno2.center
p1 = lp1.IntersectWith(linea, acExtendThisEntity)
p2 = lp2.IntersectWith(linea, acExtendThisEntity)
lp1.Delete: lp2.Delete
For i = 0 To 2
pm(i) = (p1(i) + p2(i)) / 2
Next i
po(0) = punto(0): po(1) = punto(1) - 2 * t + t / 2
pf(0) = punto(0): pf(1) = punto(1) + 2 * t + t / 2
Set pernoV(0) = acadDoc.ModelSpace.AddLine(po, pf)
po(0) = punto(0): po(1) = punto(1): pf(1) = punto(1)
If inter = "N" Then pf(0) = punto(0) + 4 * db + 2 *
(np - 1) * dP + SP
If inter = "S" Then pf(0) = punto(0) + 4 * db + 2 *
(np - 1) * dP + SP + dP
Set LPer(1) = acadDoc.ModelSpace.AddLine(po, pf)
v = LPer(1).Offset(L)
Call conv_de_var_en_linea(v(0), LPer(2))
v = LPer(1).Offset(t)
Call conv_de_var_en_linea(v(0), LPer(3))
Set LPer(4) =
acadDoc.ModelSpace.AddLine(LPer(1).StartPoint,
LPer(2).StartPoint)
Set LPer(5) =
```

```
acadDoc.ModelSpace.AddLine(LPer(1).EndPoint,
LPer(2).EndPoint)
v = LPer(1).Offset(L - dv)
Call conv_de_var_en_linea(v(0), Lpernos)
pf(0) = punto(0) + db
Set LdB1 = acadDoc.ModelSpace.AddLine(po, pf)
LdB1.Move LdB1.StartPoint, Lpernos.StartPoint
If inter = "S" Then pf(0) = punto(0) + db + dP / 2
Set LdB2 = acadDoc.ModelSpace.AddLine(po, pf)
pf(0) = punto(0) + dP
Set LdP = acadDoc.ModelSpace.AddLine(po, pf)
Set perno(1) =
acadDoc.ModelSpace.AddCircle(LdB1.EndPoint, radio)
Set pernoV(1) = pernoV(0).Copy
pernoV(1).Move LdB2.StartPoint, LdB2.EndPoint
For i = 2 To np
Set perno(i) = perno(i - 1).Copy
perno(i).Move LdP.StartPoint, LdP.EndPoint
Set pernoV(i) = pernoV(i - 1).Copy
pernoV(i).Move LdP.StartPoint, LdP.EndPoint
Next i
po(0) = punto(0): po(1) = punto(1): pf(1) = punto(1)
If inter = "N" Then pf(0) = punto(0) + (4 * db + 2 *
(np - 1) * dP) / 2 + SP
If inter = "S" Then pf(0) = punto(0) + (4 * db + 2 *
(np - 1) * dP + dP) / 2 + SP
Set Ln = acadDoc.ModelSpace.AddLine(po, pf)
For i = 1 To np
perno(i).Layer = "Pernos": pernoV(i).Layer = "Pernos"
perno(i).Color = acBlue:  pernoV(i).Color = acBlue
Set perno(np + i) = perno(i).Copy
Set pernoV(np + i) = pernoV(i).Copy
perno(np + i).Move Ln.StartPoint, Ln.EndPoint
pernoV(np + i).Move Ln.StartPoint, Ln.EndPoint
Next i
For i = 1 To 5
LPer(i).Layer = "Perfiles": LPer(i).Color = acRed
Next i
Lpernos.Layer = "Linea de Pernos": Lpernos.Color =
acYellow
Lpernos.Linetype = "acad_iso04w100"
v = LPer(1).Offset(-L)
Call conv_de_var_en_linea(v(0), Mirr)
v = LPer(1).Offset(2 * L)
Call conv_de_var_en_linea(v(0), LacotH)
Set alin =
acadDoc.ModelSpace.AddDimAligned(Lpernos.StartPoint,
perno(1).center, LacotH.EndPoint)
alin.TextHeight = 8
Set v = alin.Mirror(Mirr.StartPoint, Mirr.EndPoint)
Set v = Lpernos.Mirror(Mirr.StartPoint,
Mirr.EndPoint)
```

```
For i = 1 To 2 * np - 1
Set alin =
acadDoc.ModelSpace.AddDimAligned(perno(i).center,
perno(i + 1).center, LacotH.EndPoint)
alin.TextHeight = 8
Set v = alin.Mirror(Mirr.StartPoint, Mirr.EndPoint)
Next i
Set alin = acadDoc.ModelSpace.AddDimAligned(perno(2 *
np).center, Lpernos.EndPoint, LacotH.EndPoint)
alin.TextHeight = 8
Set v = alin.Mirror(Mirr.StartPoint, Mirr.EndPoint)
v = LPer(4).Offset(L / 2)
Call conv_de_var_en_linea(v(0), LacotV)
Set alin =
acadDoc.ModelSpace.AddDimAligned(Lpernos.StartPoint,
LPer(4).EndPoint, LacotV.EndPoint)
alin.TextHeight = 8
Set v = alin.Mirror(Mirr.StartPoint, Mirr.EndPoint)
For i = 1 To 2 * np
Set v = pernoV(i).Mirror(Mirr.StartPoint,
Mirr.EndPoint)
Set v = perno(i).Mirror(Mirr.StartPoint,
Mirr.EndPoint)
Next i
For i = 1 To 5
Set v = LPer(i).Mirror(Mirr.StartPoint,
Mirr.EndPoint)
Next i
LacotH.Delete: LacotV.Delete: Ln.Delete
LdB1.Delete: LdB2.Delete: LdP.Delete
Mirr.Delete: pernoV(0).Delete
po(0) = (LPer(1).StartPoint(0) + LPer(1).EndPoint(0))
/ 2
po(1) = (LPer(1).StartPoint(1) + LPer(1).EndPoint(1))
/ 2
For i = 1 To 2 * np
Set v = perno(i).Copy
v.Move po, pm
v.Rotate pm, linea.Angle
Set v = v.Mirror(linea.StartPoint, linea.EndPoint)
Next i
po(0) = (LPer(1).StartPoint(0) + LPer(1).EndPoint(0))
/ 2
po(1) = (LPer(1).StartPoint(1) + LPer(1).EndPoint(1))
/ 2 + t / 2
For i = 1 To 2 * np
Set v = pernoV(i).Copy
v.Move po, pm
v.Rotate pm, linea.Angle
Next i
Set lpa =
acadDoc.ModelSpace.AddLine(linea.StartPoint,
```

```
L11i.StartPoint)
lpa.Move lpa.StartPoint, pm
Set lpb =
acadDoc.ModelSpace.AddLine(linea.StartPoint,
L11s.StartPoint)
lpb.Move lpb.StartPoint, pm
lpa.Layer = "Perfiles"
lpb.Layer = "Perfiles"
If vista = "1" Then
lpa.Color = acCyan:    lpb.Color = acCyan
End If
If vista = "2" Then
lpa.Color = acRed:    lpb.Color = acRed
End If
If vista = "3" Then
lpa.Color = acCyan:    lpb.Color = acRed
End If
If vista = "4" Then
lpa.Color = acRed:    lpb.Color = acCyan
End If
If inter = "N" Then v = lpa.Offset(2 * db + SP / 2 +
(np - 1) * dP)
If inter = "S" Then v = lpa.Offset(2 * db + SP / 2 +
(np - 1) * dP + dP / 2)
Call conv_de_var_en_linea(v(0), lp1)
lp1.Layer = lpa.Layer
lp1.Color = lpa.Color
Set v = lp1.Mirror(lpa.StartPoint, lpa.EndPoint)
If inter = "N" Then v = lpb.Offset(2 * db + SP / 2 +
(np - 1) * dP)
If inter = "S" Then v = lpb.Offset(2 * db + SP / 2 +
(np - 1) * dP + dP / 2)
Call conv_de_var_en_linea(v(0), lp2)
lp2.Layer = lpb.Layer
lp2.Color = lpb.Color
Set v = lp2.Mirror(lpb.StartPoint, lpb.EndPoint)
v = lpa.Offset(SP / 2)
Call conv_de_var_en_linea(v(0), lp1)
lp1.Layer = lpa.Layer
lp1.Color = lpa.Color
Set v = lp1.Mirror(lpa.StartPoint, lpa.EndPoint)
v = lpb.Offset(SP / 2)
Call conv_de_var_en_linea(v(0), lp2)
lp2.Layer = lpb.Layer
lp2.Color = lpb.Color
Set v = lp2.Mirror(lpb.StartPoint, lpb.EndPoint)
lpa.Delete:  lpb.Delete
L11i.Delete: L11i.Delete
L13s.Delete: L13s.Delete
Lineadepernos1i.Delete: Lineadepernos1s.Delete
```

Nombre	Sub Dibuja_plancha()
Descripción	Permite dibujar alrededor de los pernos seleccionados una plancha que cumple con los requerimientos del usuario.
Variables de Entrada	No Tiene

Código	Dim obj As AcadEntity
	Dim ssetobj(0 To 100) As AcadCircle
	Dim punto As AcadCircle
	Dim lineas As AcadPolyline
	Dim L As AcadPolyline
	Dim a As AcadLine
	Dim b As AcadLine
	Dim LdC As AcadLine
	Dim LdN As AcadLine
	Dim Aux As AcadLine
	Dim v As Variant
	Dim explodedObjects As Variant
	Dim pn As Variant
	Dim numero As Integer
	Dim db As Integer
	Dim n As Integer
	Dim x As Integer
	Dim ver As Integer
	Dim k As Integer
	Dim j As Integer
	Dim i As Integer
	Dim index As Integer
	Dim texto As String
	Dim Esquina As String
	Dim pi As Double
	Dim angulo1 As Double
	Dim angulo2 As Double
	Dim angulo3 As Double
	Call inicilizar
	n = 0
	MsgBox ("Escoja los pernos en forma AntiHoraria")
	Repite:
	texto = "Selecciona el Perno " & n + 1 & " : "
	Call Test_GetEntity(obj, texto)
	If Lresult = 2 Then Exit Sub
	Debug.Print n
	If Lresult <> 9 Then
	If obj.ObjectName = "AcDbCircle" Then
	Set ssetobj(n) = obj
	obj.Highlight (True)
	n = n + 1
	Else
	If n = -1 Then n = 0
	End If

```
GoTo Repite
End If
Set ssetobj(n) = ssetobj(0)
For i = 0 To n
ssetobj(i).Highlight (False)
Next i
If n <= 1 Then Exit Sub
texto = "Distancia al Borde de la Plancha :"
Call TextGetNumber(db, texto)
If Lresult = 2 Or Lresult = 9 Then Exit Sub
x = 0
Do While x = 0
Esquina = acadDoc.Utility.GetString(False, "Esquinas
Picadas (S/N) :")
Esquina = UCase(Esquina)
If Esquina = "S" Or Esquina = "N" Then x = 1
Loop
j = 0
ReDim p(0 To 3 * n + 2) As Double
For i = 0 To n
p(j) = ssetobj(i).center(0)
p(j + 1) = ssetobj(i).center(1)
p(j + 2) = ssetobj(i).center(2)
j = j + 3
Next i
numero = 3 * n + 2
Set lineas = acadDoc.ModelSpace.AddPolyline(p)
v = lineas.Offset(db)
Set L = v(0).Copy
pn = L.Coordinates
v(0).Delete: lineas.Delete
If Esquina = "N" Then
L.Layer = "Perfiles": L.Color = acRed
explodedObjects = L.Explode
L.Delete
End If
If Esquina = "S" Then
Dim pa(0 To 2)     As Double
Dim pb(0 To 2)     As Double
Dim po(0 To 2)     As Double
Dim pf(0 To 2)     As Double
Dim ldef(0 To 500) As Double
j = 0
k = 0
For index = 0 To n - 1
pa(0)= p(j):   pa(1)= p(j + 1):   pa(2) = p(j + 2)
pb(0)= pn(j):  pb(1)= pn(j + 1):  pb(2) = pn(j + 2)
Set Aux = acadDoc.ModelSpace.AddLine(pa, pb)
pf(0) = db
Set LdN = acadDoc.ModelSpace.AddLine(po, pf)
LdN.Move po, pa
LdN.Rotate pa, Aux.Angle
```

```
pf(0) = 0:    pf(1) = db
Set LdC = acadDoc.ModelSpace.AddLine(po, pf)
po(1) = db
LdC.Move po, LdN.EndPoint
LdC.Rotate LdN.EndPoint, Aux.Angle
v = LdC.IntersectWith(L, acExtendThisEntity)
If VarType(v) = vbEmpty Then
ldef(k) = pn(j)
ldef(k + 1) = pn(j + 1)
ldef(k + 2) = pn(j + 2)
k = k + 3
End If
If VarType(v) <> vbEmpty Then
ver = 0
For i = LBound(v) To UBound(v)
ver = ver + 1
Next i
If ver <> 6 Then
ldef(k) = pn(j)
ldef(k + 1) = pn(j + 1)
ldef(k + 2) = pn(j + 2)
k = k + 3
End If
If ver = 6 Then
po(0) = v(0):  po(1) = v(1):    po(2) = v(2)
pf(0) = v(3):  pf(1) = v(4):    pf(2) = v(5)
Set a = acadDoc.ModelSpace.AddLine(pa, po)
Set b = acadDoc.ModelSpace.AddLine(pa, pf)
pi = 3.14159265358979
angulo1 = a.Angle
angulo2 = b.Angle
angulo3 = (angulo2 - angulo1) * 180 / pi
If angulo3 > 180 Then
ldef(k) = v(3):          ldef(k + 1) = v(4)
ldef(k + 2) = v(5):      ldef(k + 3) = v(0)
ldef(k + 4) = v(1):      ldef(k + 5) = v(2)
k = k + 6
End If
If angulo3 < 0 And angulo3 > -180 Then
ldef(k) = v(3):          ldef(k + 1) = v(4)
ldef(k + 2) = v(5):      ldef(k + 3) = v(0)
ldef(k + 4) = v(1):      ldef(k + 5) = v(2)
k = k + 6
End If
If angulo3 < -180 Then
ldef(k) = v(0):          ldef(k + 1) = v(1)
ldef(k + 2) = v(2):      ldef(k + 3) = v(3)
ldef(k + 4) = v(4):      ldef(k + 5) = v(5)
k = k + 6
End If
If angulo3 <= 180 And angulo3 >= 0 Then
ldef(k) = v(0):          ldef(k + 1) = v(1)
```

```
ldef(k + 2) = v(2):     ldef(k + 3) = v(3)
ldef(k + 4) = v(4):     ldef(k + 5) = v(5)
k = k + 6
End If
a.Delete
b.Delete
End If
End If
j = j + 3
po(0) = 0: po(1) = 0: po(2) = 0
pf(0) = 0: pf(1) = 0: pf(2) = 0
LdC.Delete: LdN.Delete: Aux.Delete
Next index
ldef(k) = ldef(0)
ldef(k + 1) = ldef(1)
ldef(k + 2) = ldef(2)
ReDim ld(0 To k + 2) As Double
For i = 0 To k + 2
ld(i) = ldef(i)
Next i
Set lineas = acadDoc.ModelSpace.AddPolyline(ld)
L.Delete
lineas.Layer = "Perfiles"
lineas.Color = acRed
explodedObjects = lineas.Explode
lineas.Delete
End If
```

Nombre	Sub Dimensionar()
Descripción	Permite acotar las líneas de pernos.
Variables de Entrada	No Tiene
Código	Dim Lineav As AcadLine Dim linea As AcadLine Dim Ln As AcadLine Dim Laux As AcadLine Dim LP As AcadLine Dim per As AcadObject Dim perno As AcadCircle Dim alin As AcadDimAligned Dim lay1 As AcadLayer Dim lay2 As AcadLayer Dim lay3 As AcadLayer Dim lay4 As AcadLayer Dim v As Variant Dim punto(0 To 2) As Double

```
Dim p(0 To 2)         As Double
Dim po(0 To 2)        As Double
Dim pm(0 To 2)        As Double
Dim pf(0 To 2)        As Double
Dim pi                As Double
Dim d                 As Double
Dim radio             As Double
Dim perfil            As String
Dim texto             As String
Dim i                 As Integer
Dim j                 As Integer
Dim k                 As Integer
Dim angulo1           As Double
Dim angulo2           As Double
Dim angulo3           As Double
On Error Resume Next
Retry:
Call inicilizar
Set lay1 = acadDoc.Layers.Add("0")
Set lay2 = acadDoc.Layers.Add("Perfiles")
Set lay3 = acadDoc.Layers.Add("Linea de Cg")
Set lay4 = acadDoc.Layers.Add("Pernos")
i = 0
pi = 3.14159265358979
Repite:
lay1.LayerOn = False: lay2.LayerOn = False
lay3.LayerOn = False: lay4.LayerOn = False
texto = "Selecciona la Linea de Pernos :"
Call Test_GetEntity(linea, texto)
Call prende_layers
If Lresult = 2 Or Lresult = 9 Then Exit Sub
If linea.ObjectName <> "AcDbLine" Then GoTo Repite
texto = "Punto :"
Call Test_GetPoint(v, texto)
If Lresult = 2 Or Lresult = 9 Then Exit Sub
punto(0) = v(0): punto(1) = v(1): punto(2) = v(2)
Call Limpia_pernos
Call Limpia_Lineas(linea)
linea.Linetype = "acad_iso04w100"
Call arregloID(linea)
Call
Calcula_distancia_perpendicular_entre_punto_y_linea2(linea,
punto(), d)
Set LP = acadDoc.ModelSpace.AddLine(linea.StartPoint,
punto)
angulo1 = linea.Angle
angulo2 = LP.Angle
angulo3 = (angulo2 - angulo1) * 180 / pi
If angulo3 > 180 Then d = -d
If angulo3 < 0 And angulo3 > -180 Then d = -d
LP.Delete
v = linea.Offset(d)
```

```
Call conv_de_var_en_linea(v(0), Laux)
Set LP = acadDoc.ModelSpace.AddLine(linea.StartPoint,
Laux.StartPoint)
po(0) = linea.StartPoint(0): po(1) = linea.StartPoint(1):
po(2) = linea.StartPoint(2)
Call ordena_puntos(linea, radio, LP.Angle)
For Each per In acadDoc.ModelSpace
If per.ObjectName = "AcDbCircle" Then
Set perno = per.Copy
v = linea.IntersectWith(perno, acExtendNone)
j = 0
If VarType(v) <> vbEmpty Then
For k = LBound(v) To UBound(v)
j = j + 1
If j = 1 Then
p(0) = perno.center(0)
p(1) = perno.center(1)
p(2) = perno.center(2)
pm(0) = (po(0) + p(0)) / 2
pm(1) = (po(1) + p(1)) / 2
pm(2) = (po(2) + p(2)) / 2
LP.Move LP.StartPoint, pm
If Abs(p(0) - pm(0)) > 1 Or Abs(p(1) - pm(1)) > 1 Then
Set alin = acadDoc.ModelSpace.AddDimAligned(po, p,
LP.EndPoint)
alin.TextHeight = 8
End If
po(0) = p(0): po(1) = p(1): po(2) = p(2)
End If
Next k
End If
perno.Delete
End If
Next
pf(0) = linea.EndPoint(0): pf(1) = linea.EndPoint(1): pf(2)
= linea.EndPoint(2)
pm(0) = (po(0) + pf(0)) / 2: pm(1) = (po(1) + pf(1)) / 2:
pm(2) = (po(2) + pf(2)) / 2
LP.Move LP.StartPoint, pm
Set alin = acadDoc.ModelSpace.AddDimAligned(po, pf,
LP.EndPoint)
alin.TextHeight = 8
perno.Delete: LP.Delete: Laux.Delete
```

Nombre	visivilidad_de_lineas
Descripción	Permite realizar la visibilidad manual de una conexión (recomendado para después de dibujar una plancha).
Variables de Entrada	No Tiene

Código	Dim ssetobj1 As AcadSelectionSet
	Dim ssetobj2 As AcadSelectionSet
	Dim gpCode(0) As Integer
	Dim dataValue(0) As Variant
	Dim groupCode As Variant
	Dim dataCode As Variant
	Dim linea As AcadLine
	Dim texto As String
	Dim y As Integer
	Dim n1 As Integer
	Dim n2 As Integer
	Dim L As AcadEntity
	Call inicilizar
	gpCode(0) = 0
	dataValue(0) = "Line"
	groupCode = gpCode
	dataCode = dataValue
	On Error Resume Next
	Retry:
	Set ssetobj1 = acadDoc.SelectionSets.Add("AyacaYo")
	MsgBox ("Escoja las Lineas del Perfil :")
	ssetobj1.SelectOnScreen groupCode, dataCode
	n1 = ssetobj1.count
	If n1 = 0 Then
	ssetobj1.Delete
	Exit Sub
	End If
	Set ssetobj2 = acadDoc.SelectionSets.Add("AyacaYo2")
	MsgBox ("Escoja las Lineas de la Plancha :")
	ssetobj2.SelectOnScreen groupCode, dataCode
	n2 = ssetobj2.count
	If n2 = 0 Then
	ssetobj1.Delete
	ssetobj2.Delete
	Exit Sub
	End If
	MsgBox ("¿ Cuales Lineas no se Ven ?")
	Repite:
	texto = "¿ Cuales No se Ven ?"
	Call Test_GetEntity(linea, texto)
	If Lresult = 2 Or Lresult = 9 Then Exit Sub
	If linea.ObjectName <> "AcDbLine" Then GoTo Repite
	For Each L In ssetobj1
	If L.ObjectID = linea.ObjectID Then y = 0

	Next For Each L In ssetobj2 If L.ObjectID = linea.ObjectID Then y = 1 Next If y = 0 Then Call visibilidad_lista(ssetobj1, ssetobj2) If y = 1 Then Call visibilidad_lista(ssetobj2, ssetobj1) ssetobj1.Delete ssetobj2.Delete

Nombre	Sub detalle_cruz()
Descripción	Permite realizar un detalle de la forma de los perfiles.
Variables de Entrada	No Tiene
Código	Dim po(0 To 2) As Double Dim pf(0 To 2) As Double Dim Ln(0 To 5) As AcadLine Dim perno As AcadLine Dim linea As AcadLine Dim hatchObj As AcadHatch Dim patternName As String Dim PatternType As Long Dim bAssociativity As Boolean Dim L As Double Dim t As Double Dim dv As Double Dim vista As Double Dim radio As Double Dim pi As Double Dim perfil As String Dim v As Variant Dim texto As String Dim i As Integer Dim j As Integer Call inicilizar Call Saca_datos(linea, L, t, dv, vista, radio, perfil) patternName = "ANSI31" PatternType = 0 bAssociativity = True texto = "Escoja el Punto donde Quiere el Detalle: " Call Test_GetPoint(v, texto) If Lresult = 2 Or Lresult = 9 Then Exit Sub For i = 0 To 2 po(i) = v(i)

```
Next i
Set hatchObj =
acadDoc.ModelSpace.AddHatch(PatternType, patternName,
bAssociativity)
pi = 3.14159265358979
pf(0) = po(0): pf(1) = po(1) + L
Set Ln(0) = acadDoc.ModelSpace.AddLine(po, pf)
pf(0) = po(0) + t: pf(1) = po(1) + L
Set Ln(1) =
acadDoc.ModelSpace.AddLine(Ln(0).EndPoint, pf)
pf(0) = po(0) + t: pf(1) = po(1) + t
Set Ln(2) =
acadDoc.ModelSpace.AddLine(Ln(1).EndPoint, pf)
pf(0) = po(0) + L: pf(1) = po(1) + t
Set Ln(3) =
acadDoc.ModelSpace.AddLine(Ln(2).EndPoint, pf)
pf(0) = po(0) + L: pf(1) = po(1)
Set Ln(4) =
acadDoc.ModelSpace.AddLine(Ln(3).EndPoint, pf)
pf(0) = po(0): pf(1) = po(1)
Set Ln(5) =
acadDoc.ModelSpace.AddLine(Ln(4).EndPoint, pf)
po(0) = po(0) + L - dv: po(1) = po(1) - 2 * t
pf(0) = po(0): pf(1) = po(1) + 4 * t
Set perno = acadDoc.ModelSpace.AddLine(po, pf)
For i = 0 To 5
Ln(i).Layer = "Perfiles": Ln(i).Color = acRed
Next i
perno.Layer = "Pernos": perno.Color = acBlue
hatchObj.AppendOuterLoop (Ln)
hatchObj.Evaluate
acadDoc.Regen True
For i = 0 To 2
po(i) = v(i)
Next i
For j = 0 To 2
For i = 0 To 5
Set Ln(i) = Ln(i).Copy: Ln(i).Rotate po, pi / 2
Next i
Set perno = perno.Copy: perno.Rotate po, pi / 2
If j = 1 Then hatchObj.AppendOuterLoop (Ln)
Next j
hatchObj.Evaluate
acadDoc.Regen True
```

6.3.- MÓDULO GETXX

Contiene rutinas relacionada con el manejo del mouse y del teclado, con la finalidad de filtrar la entrada de datos.

El Módulo GETXX fue realizado por Jacob Dinardi el 03/19/01, es de opción explícita, contiene las siguientes variables globales y privadas:

```
Private Declare Function GetAsyncKeyState Lib "user32" (ByVal vKey As Long) As Integer
Private Const VK_ESCAPE = &H1B
Private Const VK_LBUTTON = &H1
Private Const VK_RBUTTON = &H2
Private Const VK_RETURN = &HD
Public Const VK_SPACE = &H20
Private Const pi = 3.141592654
Public Const GETXX_SUCCESS = &H1
Public Const GETXX_ESCAPE = &H2
Public Const GETXX_RBUTTON = &H4
Public Const GETXX_RETURN = &H8
Public Const GETXX_KEYWORD = &H10
Public Const GETXX_TOOLBAR = &H20
Public Const GETXX_NOPICK = &H40
Public Const GETXX_ACADRETURN = GETXX_RBUTTON Or GETXX_RETURN
Public Const GETXX_FAILED = GETXX_ESCAPE Or GETXX_RBUTTON Or
GETXX_RETURN Or GETXX_NOPICK
Dim RC As Long, LC As Long, RET As Long, SP As Long, ESC As Long
Dim RetVal As Variant
Dim ErrNo As Integer
Dim bUseKeywords As Boolean
```

Y está constituido por las siguientes rutinas y funciones :

Nombre	Public Function GetEntity() As AcadEntity
Descripción	Retorna una variable de tipo AcadEntity dada por el usuario.
Variables de Entrada	`Lerror As Long, Optional Pt As Variant, Optional sPrompt As String`

Código	`On Error Resume Next`
	`Call Module1.inicilizar`
	`Dim RetVal As AcadEntity`
	`Do`
	`InitVars`
	`InitKeys`
	`acadDoc.Utility.GetEntity RetVal, Pt, sPrompt`
	`If Err Then`
	`Err.Clear`

	Lerror = GetXXErrType(RetVal, False, 1, sPrompt)
	If Lerror = GETXX_KEYWORD Then Exit Do
	Else
	Lerror = GETXX_SUCCESS
	Set GetEntity = RetVal
	End If
	If Lerror And GETXX_FAILED Then Exit Do
	Loop Until Lerror And GETXX_SUCCESS

Nombre	Public Function GetNumber() As Variant
Descripción	Retorna una variable tipo Integer dada por el usuario.
Variables de Entrada	Lerror As Long, bIntegerOnly As Boolean, Optional sPrompt As String, Optional sKeyWordList As String, Optional lBits As Long

Código	On Error Resume Next
	Call Module1.inicilizar
	Do
	InitVars
	InitKeys
	If sKeyWordList <> vbNullString Then
	acadDoc.Utility.InitializeUserInput lBits, sKeyWordList
	bUseKeywords = True
	End If
	If bIntegerOnly Then
	RetVal = acadDoc.Utility.GetInteger(sPrompt)
	Else
	RetVal = acadDoc.Utility.GetReal(sPrompt)
	End If
	If Err Then
	Err.Clear
	Lerror = GetXXErrType(RetVal, bUseKeywords, lBits, sPrompt)
	If Lerror = GETXX_KEYWORD Then
	GetNumber = RetVal
	Exit Do
	End If
	Else
	Lerror = GETXX_SUCCESS
	If bIntegerOnly Then
	GetNumber = CInt(RetVal)
	Else
	GetNumber = RetVal
	End If
	End If
	If Lerror And GETXX_FAILED Then Exit Do
	Loop Until Lerror And GETXX_SUCCESS

Nombre	Public Function GetPoint() As Variant
Descripción	Retorna una variable tipo AcadPoint dada por el usuario.
Variables de Entrada	`Lerror As Long, Optional Pt As Variant, Optional sPrompt As String, Optional sKeyWordList As String, Optional lBits As Long`

Código	`On Error Resume Next`
	`Call Module1.inicilizar`
	`Do`
	`InitVars`
	`InitKeys`
	`If sKeyWordList <> vbNullString Then`
	`acadDoc.Utility.InitializeUserInput lBits, sKeyWordList`
	`bUseKeywords = True`
	`End If`
	`If Not IsMissing(Pt) Then`
	`RetVal = acadDoc.Utility.GetPoint(Pt, sPrompt)`
	`Else`
	`RetVal = acadDoc.Utility.GetPoint(, sPrompt)`
	`End If`
	`If Err Then`
	`Err.Clear`
	`Lerror = GetXXErrType(RetVal, bUseKeywords, lBits, sPrompt)`
	`If Lerror = GETXX_KEYWORD Then`
	`GetPoint = RetVal`
	`Exit Do`
	`End If`
	`Else`
	`Lerror = GETXX_SUCCESS`
	`GetPoint = RetVal`
	`End If`
	`If Lerror And GETXX_FAILED Then Exit Do`
	`Loop Until Lerror And GETXX_SUCCESS`

Nombre	Public Function GetString() As Variant
Descripción	Retorna una variable tipo String dada por el usuario.
Variables de Entrada	`Lerror As Long, Optional bHasSpaces As Boolean, Optional sPrompt As String, Optional sKeyWordList As String, Optional lBits As Long`

Código	`On Error Resume Next`
	`Do`
	`InitVars`

	InitKeys
	If sKeyWordList <> vbNullString Then
	bUseKeywords = True
	ThisDrawing.Utility.InitializeUserInput lBits, sKeyWordList
	RetVal = ThisDrawing.Utility.GetKeyword(sPrompt)
	Else
	RetVal = ThisDrawing.Utility.GetString(bHasSpaces, sPrompt)
	End If
	If Err Then
	Err.Clear
	Lerror = GetXXErrType(RetVal, bUseKeywords, lBits, sPrompt)
	Else
	If bUseKeywords Then
	Lerror = GETXX_KEYWORD
	Else
	Lerror = GETXX_SUCCESS
	End If
	GetString = RetVal
	Exit Do
	End If
	If Lerror And GETXX_FAILED Then Exit Do
	Loop Until Lerror And GETXX_SUCCESS

Nombre	Private Sub InitVars()
Descripción	Inicializa a las variables.
Variables de Entrada	No Tiene
Código	ErrNo = 0 ThisDrawing.SetVariable "ERRNO", ErrNo bUseKeywords = False

Nombre	Private Sub InitKeys()
Descripción	Inicializa las constantes del mouse y del teclado.
Variables de Entrada	No Tiene
Código	GetAsyncKeyState (VK_LBUTTON) GetAsyncKeyState (VK_RBUTTON) GetAsyncKeyState (VK_RETURN) GetAsyncKeyState (VK_SPACE)

6.4.- MÓDULO MODTESTGETXX

Contiene rutinas relacionada con el manejo del mouse y del teclado, con la finalidad de filtrar la entrada de datos.

El Módulo MODTESTGETXX fue realizado por Jacob Dinardi el 03/19/01, es de opción explícita, contiene las siguientes variables globales y privadas:

```
Public Lresult As Long
Dim RetVal As Variant
Dim RetEnt As AcadEntity
Dim Origin(0 To 2)  As Double
```

Y está constituido por las siguientes rutinas y funciones :

Nombre	Sub Test_GetEntity()
Descripción	Pide una entidad y se repite hasta que la entidad sea suministrada o se apriete el botón derecho del Mouse o la tecla Esc sea presionada.
Variables de Entrada	`RetEnt As AcadEntity, texto As String`

Código	`Do`
	`Set RetEnt = GetEntity(Lresult, , texto)`
	`Loop Until Lresult And GETXX_SUCCESS Or Lresult And GETXX_ESCAPE Or Lresult And GETXX_RBUTTON`
	`Select Case Lresult`
	`Case GETXX_SUCCESS`
	`Case GETXX_ESCAPE`
	`Exit Sub`
	`Case GETXX_RETURN`
	`Case GETXX_RBUTTON`
	`Lresult = 9`
	`Exit Sub`
	`Case GETXX_KEYWORD`
	`End Select`

Nombre	Sub TextGetNumber()
Descripción	Pide un número entero y se repite hasta que el número sea suministrado o se apriete el botón derecho del Mouse o la tecla Esc sea presionada.
Variables de Entrada	`numero As Integer, texto As String`

Código	`RetVal = GetNumber(Lresult, True, texto)`
	`Select Case Lresult`
	`Case GETXX_SUCCESS`
	`numero = RetVal`
	`Case GETXX_ESCAPE`
	`Exit Sub`
	`Case GETXX_RETURN`
	`Case GETXX_RBUTTON`
	`Case GETXX_KEYWORD`
	`End Select`

Nombre	Sub Test_GetPoint()
Descripción	Pide un punto y se repite hasta que el punto sea suministrado o se apriete el botón derecho del Mouse o la tecla Esc sea presionada.
Variables de Entrada	`RetVal As Variant, texto As String`

Código	`RetVal = GetPoint(Lresult, , texto)`
	`Select Case Lresult`
	`Case GETXX_SUCCESS`
	`Case GETXX_ESCAPE`
	`Exit Sub`
	`Case GETXX_RETURN`
	`MsgBox "return"`
	`Case GETXX_RBUTTON`
	`MsgBox "rbutton"`
	`Case GETXX_KEYWORD`
	`MsgBox RetVal`
	`End Select`

7.- CONCLUSIONES

7.- CONCLUSIONES

El programa realizado aumenta la eficiencia técnica y económica en el dibujo de conexiones estructurales de acero utilizando el Sistema AutoCad. Además, constituye una herramienta muy importante para el Ingeniero Civil, ya que facilita el entendimiento de los planos de Taller a la vez que reduce el tiempo de elaboración de los mismos y agiliza cualquier cambio en el diseño.

A través del Sistema AutoCAD, se facilita la programación debido a que éste maneja instrucciones de graficación, con lo cual se permite que la elaboración del programa sea de una manera mas sencilla.

Es importante resaltar que al ver la forma tan rápida en que puede elaborarse un plano de Taller con la utilización de este programa, no podría percibirse el grado de dificultad y lo complejo del trabajo realizado para la elaboración del mismo, sobre todo por la diversidad de criterios de modelaje estructural de éstos y lo minucioso del detallado en el dibujo de los planos.

8.- RECOMENDACIONES

8.- RECOMENDACIONES

El presente Trabajo Especial de Grado permite el dibujo de algunos tipos de conexiones de perfiles de acero. Considerando que resulta imposible elaborar un programa que abarque todos los tipos de conexiones, en el tiempo estipulado para la conclusión del mismo, se hizo necesario el planteamiento de limitaciones. Por lo tanto, se recomienda continuar creando rutinas de mayor número de conexiones, así como también, la creación de una interfase de este programa de dibujo con otro de cálculo y diseño estructural.

9.- BIBLIOGRAFÍA

9.- BIBLIOGRAFÍA

1. RENGIFO, B.; **Diseño de un Generador de Datos Gráficos en Lenguaje Auto LISP para el Análisis de Estructuras Planas y-o Espaciales**, Trabajo Especial de Grado, UCV, Caracas, 1996.

2. BENTOLILA, R.; LANDER, P., **Módulo de Lenguaje AutoLISP para el Dibujo de Conexiones Estructurales de Acero en Ambiente AutoCAD**, Trabajo Especial de Grado, Universidad Metropolitana, Caracas, 1992.

3. ROE, A; **Using Visual Basic with AutoCAD 2000**, Delmar Publishers, 2nd Bk&Cd edition, ISBN: 0766820912, October 15, 2000.

4. GESNER, R.; SMITH, J., **Maximizing AutoCAD**, New Riders Publishing, Indiana, USA, 1992.

5. TAJADURA Z. Jose A, MANSO I. Begoña. y LOPEZ F. Javier, **Programación con AutoCAD**, Mc Graw Hill / Interamericana de España, S.A.U, Madrid España, 1999.

6. RAKER, D. And RICE, H., **Inside AutoCAD**, New Riders Publishing, U.S.A., 1990.

7. **Exploiting the Power of VBA in AutoCAD 2000**, Miller Freeman Books, ISBN: 0-87930-574-6, December 1999.

8. DIETMAR, R; **Mastering AutoCAD 2000 Objects**, ISBN: 078212562X, December 1999.

9. SUTPHIN, J; **AutoCAD 2000 VBA Programers Reference**, Wrox Press Inc, ISBN: 1861002564, April 1999.

10. http://www.lawebdelprogramador.com/cursos/autocad/doce_1.php

11. KRAMER, B. , **Accesing the X Data**, Programmer's toolbox, Cadence, 1994.

12. CVG SIDERÚRGICA DEL ORINOCO, **Perfiles Comerciales**, Puerto Ordaz, 1980.