*Universidad Tecnológica Nacional*
*Facultad Regional Villa María*
*Departamento de Electrónica*
*Cátedra Trabajo Final de Grado*

# Perforadora Automática para PCB

Autor:

Buzzetti Gonzalo Ezequiel

## 2020

**Acreditación:**

**Fecha: Junio 2020**

**Comité Evaluador**

**Presidente:** Apellido y Nombres

**1º Vocal:** Apellido y Nombres

**2º Vocal:** Apellido y Nombres

# Dedicatorias

*A mis padres.*

*A mis hermanos.*

*Este trabajo se lo dedico a mi familia. Aquellos que siempre estuvieron allí apoyándome en mis estudios. En el comienzo de mi carrera surgieron muchos problemas y de no ser ellos posiblemente hoy no estaría aquí. Para ello, muchas gracias.*

# Agradecimientos

Agradezco a todos aquellos que aportaron su granito de arena para que yo pudiera lograr este proyecto. Gracias a José, Nilda y Alejandro, amigos de la familia.

# Memoria Descriptiva

*Este trabajo consistió en el diseño, desarrollo y construcción de una perforadora automática para PCB, cuyas perforaciones son necesarias para el montaje de componentes electrónicos trough-hole. Dicho prototipo es, en esencia, un router CNC, lo que se significa que las perforaciones se realizan en las ubicaciones indicadas por las coordenadas de los ejes X e Y. Éstas, su vez, son generadas por los CAD's o software de diseño de PCB con un formato especificado. La forma de implementación que se escogió para llevar el proceso de perforado es mediante el envío de tales coordenadas desde una PC hacia la placa de control de la máquina. Es por ello que se diseñó el circuito de control de la perforadora para que se encargue de establecer la comunicación con la PC, la lectura de datos y su interpretación y el accionamiento correspondiente de los ejes de la máquina.*
*Además, se implementó un software de PC para que el usuario interactúe con la perforadora y de esta manera sea fácil su utilización.*

# Índice

# Introducción

## Análisis del problema

Una placa posee en mayor o menor medida componentes con encapsulado through-hole, lo que significa que para montarlos a la misma es necesaria realizar perforaciones en el sustrato. Esta es una tarea que puede ejecutarse manualmente, logrando resultados con calidad aceptable. Sin embargo, sucede que, si la PCB posee demasiados componentes de este tipo, la cantidad de agujeros a realizar es elevada, lo que conlleva a un trabajo repetitivo y tedioso por parte de quien lo realiza.

Además, se le suma otro problema: la rotura de las mechas o brocas. Las mechas o brocas utilizadas mayormente poseen un diámetro de entre 0.5mm y 1mm. Al ser éstas tan delgadas si el mini torno o herramienta que se utilice no se posiciona perfectamente vertical se corre el riesgo de que se quiebren. No obstante, utilizando un soporte para mini torno o taladro de banco se logra colocar la herramienta perfectamente vertical, logrando que las perforaciones también lo estén y el montaje de los componentes electrónicos sea fácil de realizar. Sin embargo, esto no deja de ser un trabajo manual que depende de la habilidad de quien lo realiza.

## Análisis de sistemas existentes

Las perforadoras automáticas ya existen desde hace tiempo en la industria. De hecho, son los Router CNC o control numérico computarizado. No solo perforan, sino que también pueden programarse para fresar, cortar, grabar, etc. También existen en el mercado Router CNC para hobbystas y pequeños desarrolladores que pueden realizar mismas funciones, obviamente en piezas más pequeñas.

Este proyecto implica desarrollar una maquina similar a las que se encuentran en mercado cuya función sea la de perforar tarjetas electrónicas o PCBs.

Realizar este proyecto conlleva varias ventajas, a saber:

- Integración de muchos conocimientos adquiridos no solo de electrónica, sino también de programación y automatización.
- Está diseñado específicamente para perforar, en lo que se refiere a firmware y software.
- Su construcción no es compleja. Los materiales utilizados son muy fáciles de conseguir y los pasos para ensamblar la maquina son simples.
- Cuando se inició este proyecto los router CNC empezaban a tener popularidad. Se debió principalmente a las impresoras 3D. Hoy en día es muy fácil encontrarlos en el mercado. Sin embargo, la maquina puede servir como perfectamente como impresora 3D, fresadora, etc. Solo se cambia el mini torno por la herramienta apropiada y realizar un cambio de firmware, quedando completamente funcional.

## Descripción de las actividades del proyecto

Básicamente las actividades a realizar en el proyecto son las siguientes, en orden cronológico:

- Diseño de la estructura de la perforadora: se comienza primero analizando la forma constructiva de la perforadora, es decir, el tamaño, materiales, forma, etc., como así también facilidad de construcción, costo, etc. En base a ello se elige el diseño que mejor se adapte a los requisitos anteriores.
- Construcción: siguiendo el diseño optado se procede a la adquisición de los materiales y su posterior construcción.
- Diseño y construcción del circuito de control: la perforadora necesita un circuito electrónico que controle el accionamiento de los motores de los tres ejes. Es por ello que debe diseñarse e implementarse el mismo. Además, si se quiere que la perforadora sea controlada por una PC, este debe poder recibir los datos de dicha PC y transformarlo en movimientos de los motores.

- Desarrollo del firmware: el circuito de control se basa en un microcontrolador, por lo que necesita de un firmware o programa para funcionar. Debe poder comunicarse con una PC y llevar acabo los accionamientos de los ejes de la perforadora.
- Desarrollo de la aplicación de PC: se necesita escribir un código para realizar un programa para el cual controle la perforadora.
- Pruebas: una vez que se realiza todo lo anterior se debe probar el prototipo para ver si todo funciona según lo esperado y ver los resultados.

# Objetivos

## Objetivos generales

El objetivo de este proyecto es la realización de una máquina totalmente automática cuya tarea sea la de realizar perforaciones para placas de circuito impreso para el posterior montaje de componentes y circuitos integrados through-hole.

El desempeño de la misma debe ser tal que la placa PCB a la que se va a perforar tenga una gran precisión y permita montar todo tipo de componentes through-hole. También debe ser fácil de utilizar y ser lo suficientemente rápida para que el usuario puede producir PCB's de forma sencilla y rápida.

## Objetivos particulares

Los objetivos particulares para este proyecto son los siguientes:

- Implementar una perforadora totalmente funcional, estructuralmente robusta, con materiales fáciles de adquirir y con el costo más bajo posible.
- Como la idea es que dicha perforadora se controla por medio de una PC, el segundo objetivo es diseñar una placa de control con un microcontrolador capaz de comunicarse con la PC y controlar los motores con los que está constituido la máquina.
- Desarrollar un firmware lo más eficiente posible, en el que se implemente un protocolo de comunicación actualmente en uso (USB).
- Desarrollar un programa de PC para que el usuario opere la máquina de la forma más fácil posible. Por ello se optó por desarrollar un programa con interfaz gráfica (ventana) para que sea mucho más intuitivo.

# Diseño del Proyecto

## Revisión de requerimientos y parámetros de operación

A continuación, se enumeran y comentan qué características debe poseer la perforadora automática y sus parámetros de funcionamiento.
En primer lugar, su construcción no debe ser compleja y con materiales fáciles de conseguir. Además, como se utilizará para el perforado de placas no es necesario que sea de gran tamaño. Las PCB que se realizan en general tienen un tamaño promedio de 15cmx15cm, siendo estas medidas las que se tendrán en cuenta para la mesa de trabajo de la perforadora. Todo lo anterior conlleva a que sea económica al fabricarla.

Con respecto a los parámetros de operación son los siguientes:

1. Debe poseer un software amigable y fácil de utilizar,
2. Debe ser totalmente automática. Una vez que la PCB se encuentre en la mesa de trabajo y se posicione el cero de pieza, el usuario con darle un clic a un botón ya debe iniciar el proceso.
3. El software será el encargado de detectar la conexión de la perforadora automática al PC, sin que el usuario intervenga.
4. Deberá tener una precisión aceptable, con bajo margen de error.

## Selección de componentes y dispositivos

A continuación, se listan los componentes electrónicos que se utilizaron para la fabricación de las placas de circuito impreso en este proyecto, como así también las herramientas e instrumentos necesario para el funcionamiento de las mismas. En el Anexo II se detallan los materiales que fueron utilizados para la construcción de la perforadora junto con el paso a paso para su confección.

| MATERIALES PARA LA CONSTRUCCION DE LA PLACA DE CONTROL | CANTIDAD |
|---|---|
| **RECORTE PLACA EPOXY 80mmx90mm** | 1 |
| **PIC18F14K50 CON ZÓCALO** | 1 |
| **DRIVER POLOLU A4988** | 3 |
| **CAPACITOR POLARIZADO 100uF 35V** | 3 |
| **CAPACITOR POLARIZADO 47uF 25V** | 2 |
| **CAPACITOR CERAMICO 0,1uF** | 2 |
| **CAPACITOR CERAMICO 33pF** | 2 |
| **CRISTAL 12MHz** | 1 |
| **RESISTENCIA 10K ¼W** | 4 |
| **RESISTENCIA 1K ¼W** | 1 |
| **LED 3MM** | 1 |
| **CONECTOR USB TIPO 90º** | 1 |
| **CONECTOR BARREL JACK 5.5mmx2.5mm** | 1 |
| **CONECTOR BARREL JACK 5.5mmx2.1mm** | 1 |
| **TIRA DE PINES MACHO CORTOS 90º 1x4** | 3 |
| **TIRA DE PINES MACHO CORTOS RECTOS 1x8** | 1 |
| **TIRA DE PINES HEMBRA RECTO 1x6** | 1 |

***Tabla 1:*** *Lista con todos los componentes electrónicos para el montaje de la placa de control*

| MATERIALES PARA LA CONSTRUCCION DE SENSORES | CANTIDAD |
|---|---|
| **RECORTE PLACA EPOXY 1mmx5mm** | 4 |
| **FINALES DE CARRERA 1 AMPER PARA PCB** | 4 |
| **TIRA DE PINES MACHO CORTOS RECTOS 1x2** | 4 |

***Tabla 2:*** *Materiales para la construcción de los finales de carro.*

| MISCELÁNEAS | CANTIDAD |
|---|---|
| **ANILLOS DE FERRITE PARA CABLE** | VARIOS |
| **CABLE 26 AWG, VARIOS COLORES** | CANTIDAD NECESARIA |
| **FUENTE DE ALIMENTACION 12V 5A** | 1 |
| **CABLE USB CON FICHA TIPO B** | 1 |
| **PROGRAMADOR PICKIT 2 O SUPERIOR** | 1 |
| **PC CON CONEXIÓN USB, WINDOWS 7 O SUPERIOR** | |

***Tabla 3:*** *Componentes y elementos necesarios adicionales para el montaje de la perforadora.*

## Diagrama en bloques del dispositivo.

A continuación se muestra un diagrama de bloques del dispositivo a implementar:

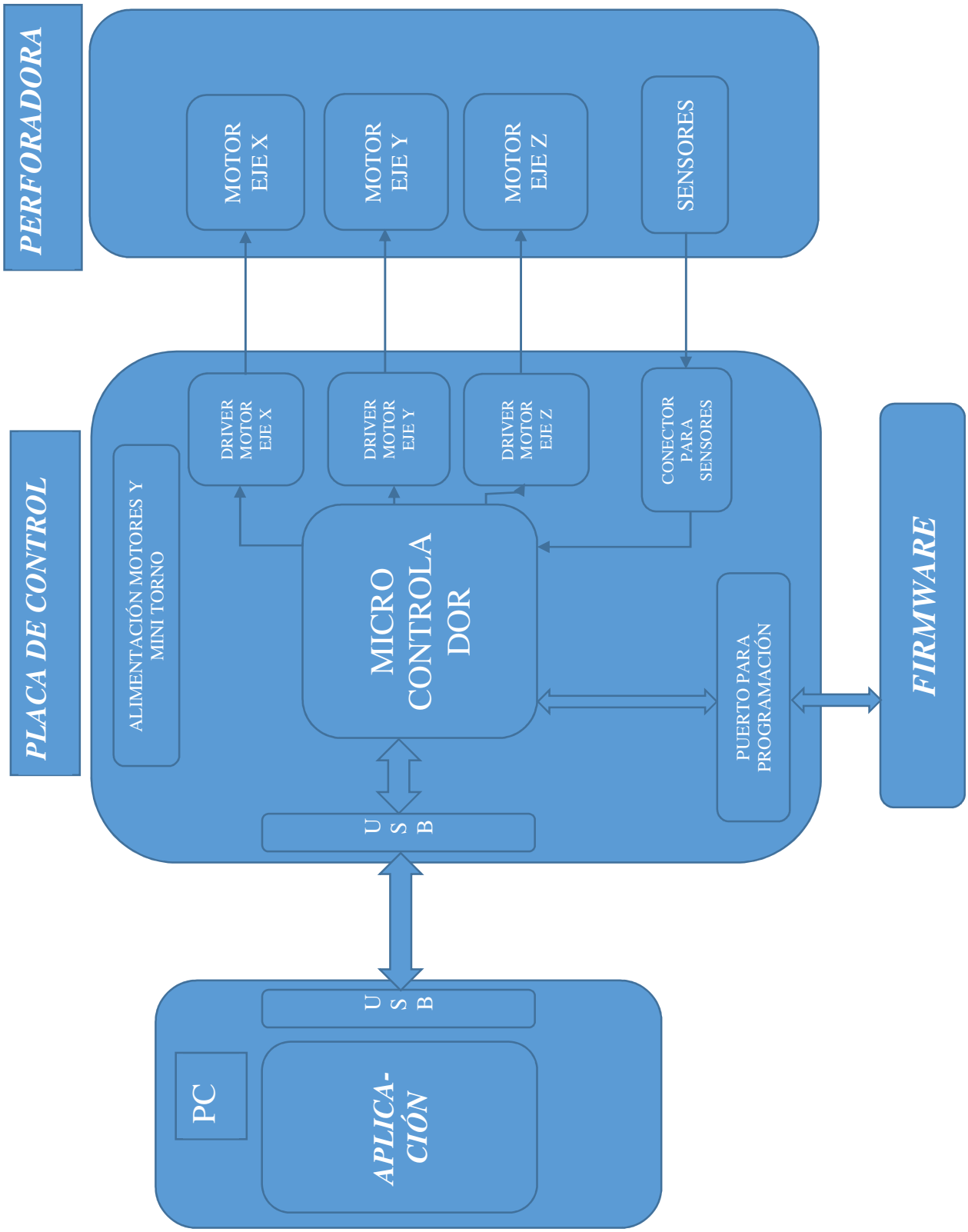



**FIG. Nº 1** *Diagrama en bloque del dispositivo*

De acuerdo a la figura anterior la implementación de prototipo está conformado básicamente por cuatro partes interrelacionadas entre sí. Ellas son: la estructura en sí misma junto a los motores y sensores necesarios para su funcionamiento. La segunda de ellas es la placa o circuito de control, encargada nada más y nada menos que ejecutar de forma exacta los desplazamientos de los ejes de la perforadora de acuerdo a los datos recibidos desde una PC. La tercera, intrínsecamente relaciona con la anterior, es el código del microcontrolador o firmware al que hay que dotar a este CI para realizar la labor pretendida. Por último se encuentra el software para PC desarrollado específicamente para manipular la perforadora por parte del usuario. Todos estos puntos mencionados anteriormente se desarrollan y profundizan a continuación.

## Descripción de cada una de las partes

De acuerdo con el apartado anterior el proyecto consta, básicamente, de cuatro partes: la construcción de la perforadora, el diseño y elaboración de la placa de control y sensores, la programación del microcontrolador o firmware y el desarrollo de la aplicación para PC para el control de la perforadora automática. A continuación se explican cada una de ellas.

## Construcción de la perforadora

Solo se mostrará la imagen final en software de diseño. Los planos para la fabricación de las piezas en MDF se encuentran en el Anexo I y los materiales y los pasos para su montaje en el Anexo II.

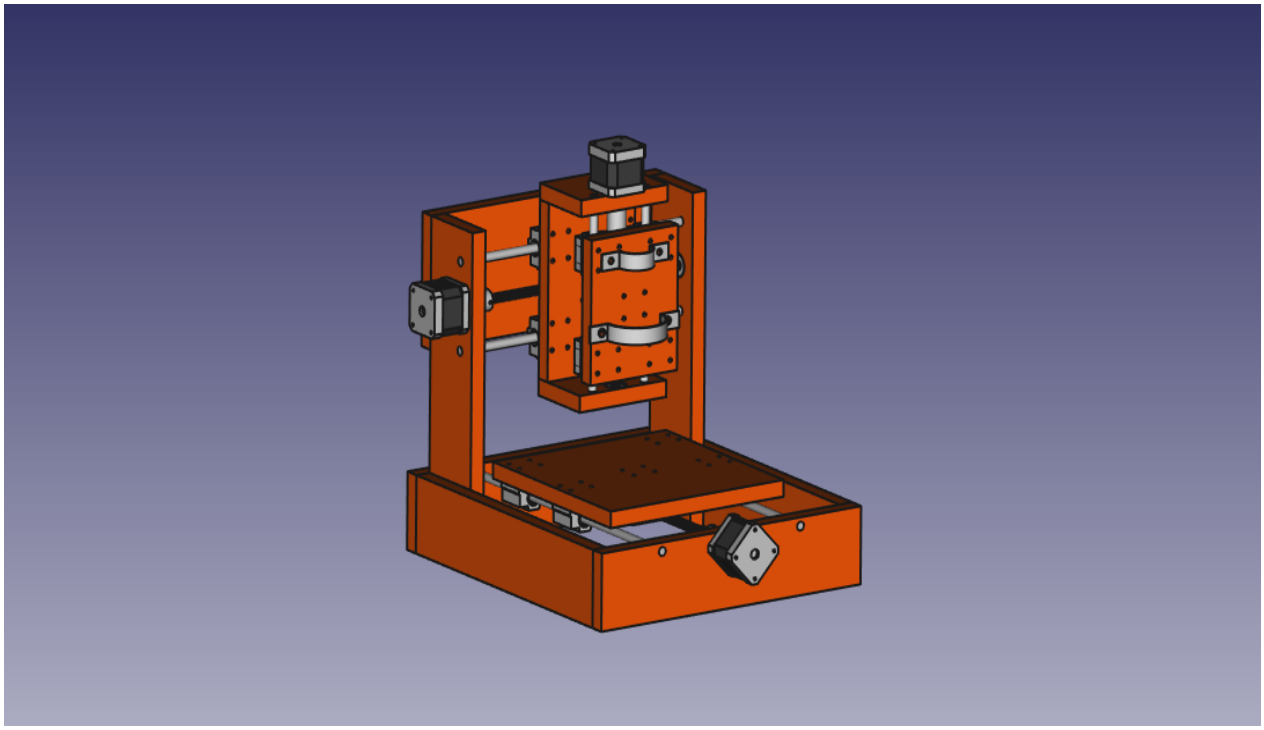


**FIG. Nº 2** *Ilustración en CAD de diseño de la perforadora acabada.*

Funcionamiento de la máquina

Cómo el giro del eje de los motores se traduce en un desplazamiento lineal es muy sencillo: Primero se necesita una tuerca con la misma rosca que la varilla del eje como la que se muestra en la Fig. Nº 3. Cabe destacar que el paso de la rosca de ambos es de 4X2mm por vuelta rosca tipo ACME.

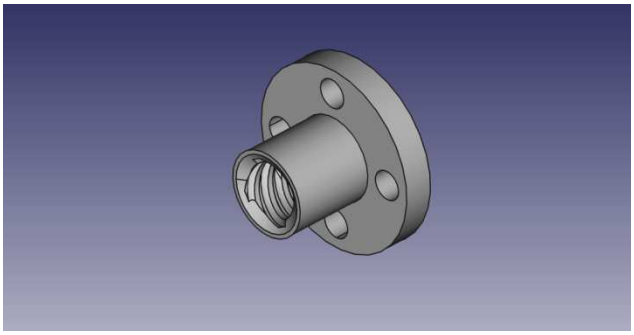


**FIG. Nº** *3 Imagen ilustrativa de tuerca paso 8 mm helicoidal.*

Luego se atornilla con la camisa utilizando los tornillos m3:

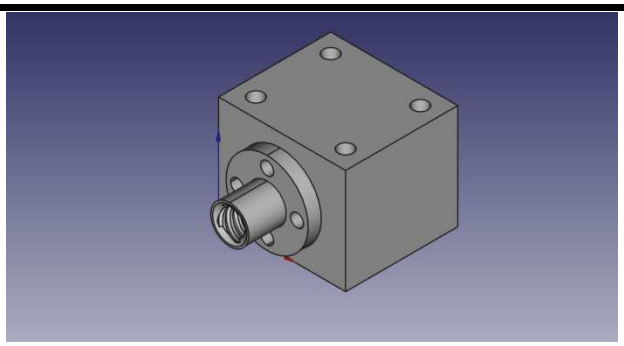


**FIG. Nº 4** *Tuerca acoplada a la camisa.*

El conjunto tuerca-bloque se enrosca en la varilla de 8mm y la pieza de madera del eje correspondiente se atornilla con tornillos m4 al dicho bloque, como se muestra en la figura Nº 5. De esta forma al girar el motor también girará la varilla roscada. Esto producirá un desplazamiento del conjunto tuerca-camisa y de la pieza de madera a la que sujeta en línea recta (la pieza a moverse estará además guiada por un par de varillas aceradas).

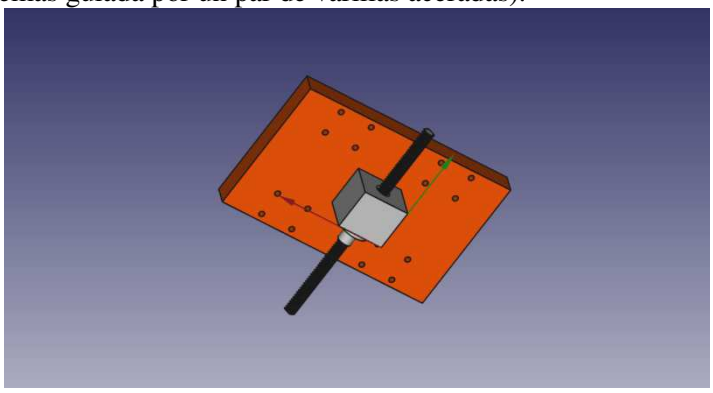


**FIG. Nº 5** *Conjunto tuerca-camisa acoplado a la pieza móvil.*

Para evitar pérdidas de precisión en el desplazamiento de los carros de los ejes 'X' e 'Y' se optó por tuercas anti-holgura o anti-backlash en lugar de una tuerca común como la mostrada en la figura 6.

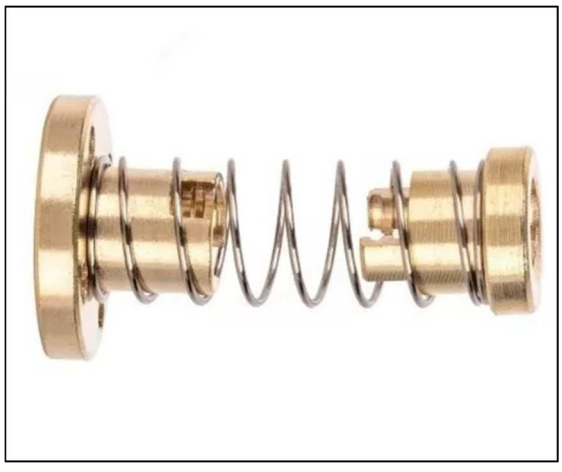


**FIG. Nº 6** *Tuerca "anti-backlash".*

Se sabe ahora cómo se traduce un movimiento rotatorio en uno lineal, pero no se sabe cuál es la relación entre ambos. Es decir, cuánto tiene que girar el eje del motor para desplazar el eje correspondiente la perforadora una cierta longitud. Es por ello que se explicará a continuación.
Primero se debe saber el paso que posee la varilla roscada. Ésta es de 4 hilos 2mm por paso, lo

que significa que recorre 8 mm por vuelta. Por otro lado los motores paso a paso son de 1,8º por paso, deduciéndose que necesitan 200 pasos para un giro completo.

$$Pasos\ por\ giro = \frac{360º}{angulo\ por\ paso} = \frac{360º}{1.8º} = 200 \qquad \textit{(Ecu. 1)}$$

Pero a su vez los drivers que controlan los motores del eje X e Y dividen cada paso del motor en 16,

$$Angulo\ por\ paso\ con\ driver\ = \frac{1.8º}{16} = 0.1125º \qquad \textit{(Ecu. 2)}$$

lo que significa que dará un giro completo cuando se apliquen 3200 pasos a los motores.

$$Pasos\ por\ giro\ con\ driver = \frac{360º}{angulo\ por\ paso\ con\ driver} = \frac{360º}{0.1125º} = 3200 \qquad \textit{(Ecu. 3)}$$

Para el eje Z no se dividen los pasos, ya que no se necesitan precisión, ni conocer la distancia que se desplaza (se explicará más adelante la razón de esto).
Entonces, resumiendo: la varilla avanza 8 mm por vuelta y los motores necesitan dar 3200 pasos para dar un giro. Con una regla de tres simple directa se calcula cuánto se desplaza el carro por cada paso del motor, y esto es igual a 0.0025mm.

$$1\ paso\ de\ motor = \frac{paso\ rosca}{pasos\ por\ giro\ con\ driver} = \frac{8mm}{3200} = 0.0025\ mm \qquad \textit{(Ecu. 4)}$$

Esta regla de cálculo servirá cuando se escriba el código del firmware del microcontrolador y se tenga que calcular la cantidad de pasos que se deban aplicar a los drivers de los motores.

## Diseño de la placa de control

Antes de entrar en el diseño del circuito electrónico que controlará la perforadora, se comentarán algunos criterios de diseño.
En primer lugar, se hablará de la elección del chip principal de la placa de control. Básicamente se puede optar por un microprocesador o un microcontrolador. Como los microcontroladores ya poseen memorias ROM, RAM y periféricos integrados se optó por este último. Dentro de las diferentes marcas se eligió los microcontroladores de 8 bits PIC de la empresa Microchip. Las razones fueron porque cuenta una gran variedad de microcontroladores de 8 bits, existe una vasta cantidad de información en internet, incluyendo blogs de aficiones, blogs propios de la empresa, y hasta páginas web oficiales con información y técnicas de programación. Posee su propio compilador en C gratuito llamado XC8 y el entorno de desarrollo denominado MPLABX IDE. Por último, y no menos importante, posee una librería de aplicaciones o MLA (Microchip Applications Library) la cual es un conjunto de archivos como código fuente y librerías, stacks de protocolos y frameworks básicos que brinda la empresa para que cualquier usuario pueda utilizarlos en sus proyectos. Una de esas librerías incluye funciones y macros para utilizarlas en proyectos que incluyan conectividad USB y emulación del puerto serie, que es el que se basa este proyecto, ya que el microcontrolador escogido posee modulo USB integrado, dotando a la placa de control con conexión USB.
El microcontrolador escogido es el PIC18F14K50, que entre sus características más importantes se destacan: 20 pines con 14 puertos de entrada/salida, 16kbytes de memoria ROM, 256 bytes de

RAM y módulo USB incorporado. Al integrar un módulo USB, se evita tener que colocar uno independiente que sirva como interfaz, lográndose un diseño de placa más sencillo. Si bien el manejo de dicho modulo no es sencillo, como se dijo anteriormente Microchip provee a sus usuarios de ejemplos de aplicación, librerías de cabeceras y demás archivos para facilitar el desarrollo de firmwares.

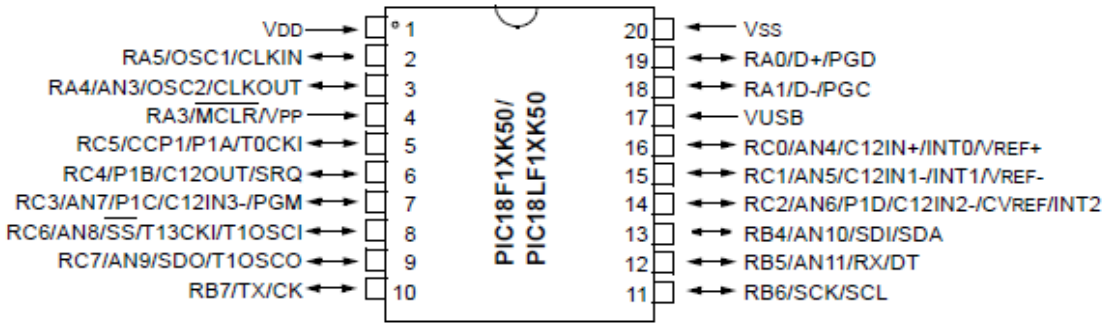



**FIG. Nº 7** *Configuración de pines del microcontrolador PIC18F14K50[6].*

Diagrama esquemático del circuito:

A continuación, en la fig. Nº 8, se muestra el diagrama esquemático del circuito. El software utilizado para el diseño y fabricación del mismo es el KICAD versión 4.0.1 estable. Este software también genera un archivo ".drl", que es un archivo con códigos G y M que muestras las coordenadas de las perforaciones. Este archivo es sumamente importante, ya que a partir de este es que la perforadora automática realiza su tarea y es por ello que se explicará más adelante.

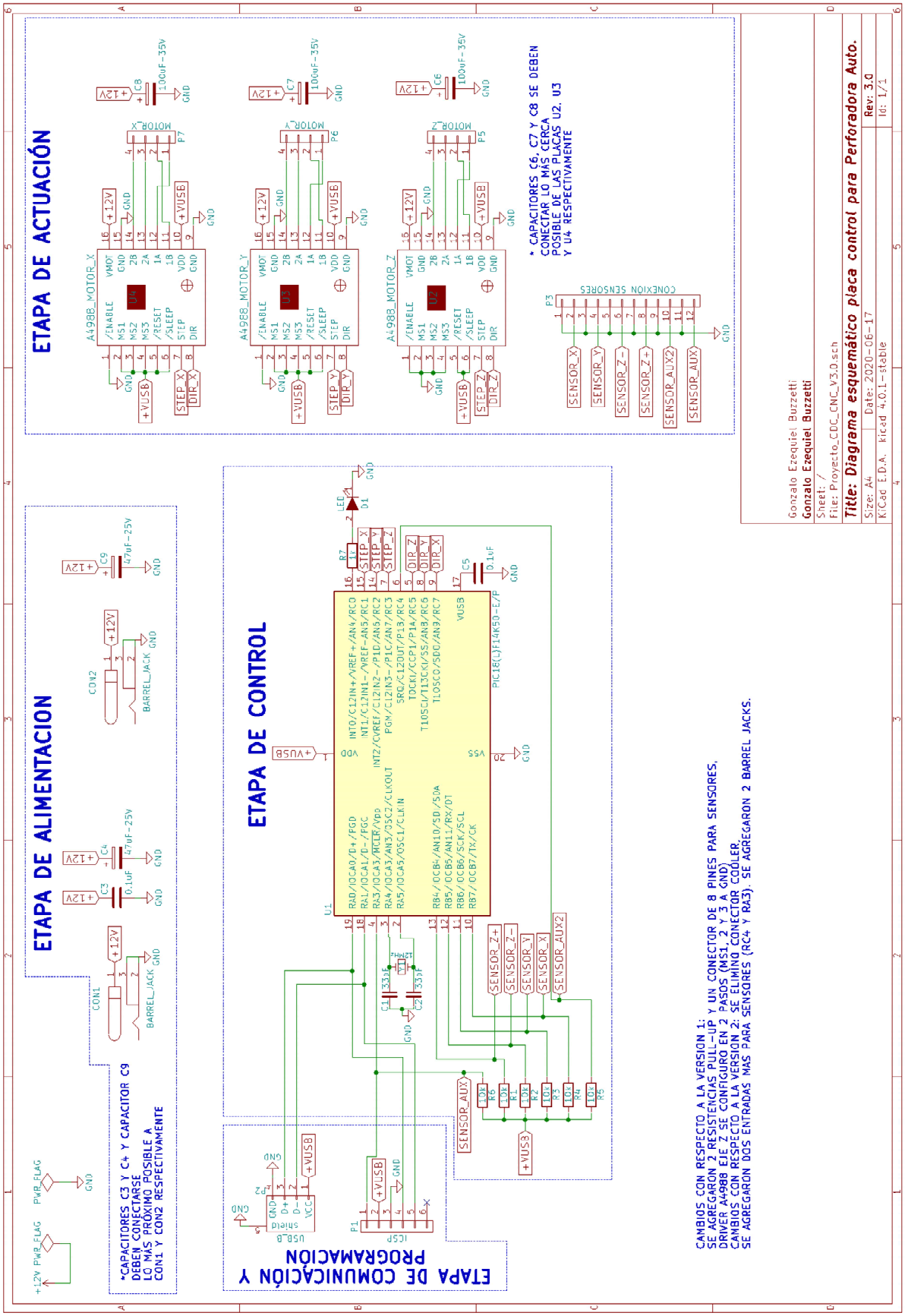



**FIG. Nº 8** *Diagrama esquemático de la placa de control de la perforadora.*

La parte principal de la placa es el microcontrolador PIC1814K50 (U1 en la fig. Nº 9), quien es el responsable de lograr la comunicación con la aplicación de la PC y controlar los avances de los tres carros de la perforadora durante el perforado. Se procederá a describir los parámetros de diseño.

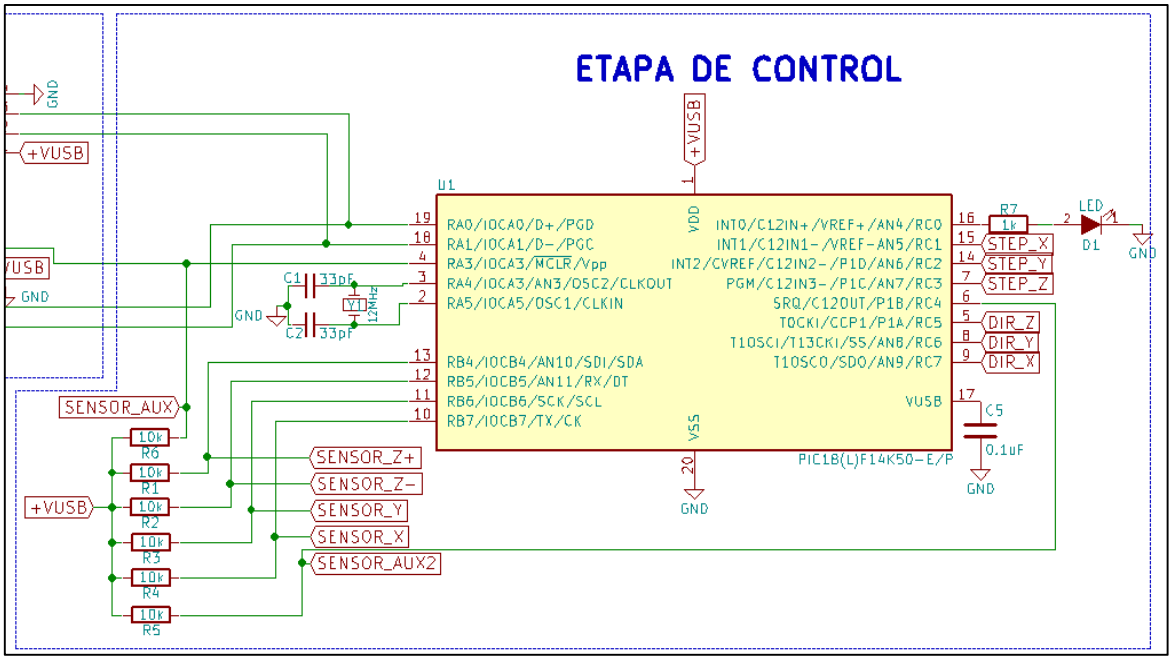



**FIG. Nº 9** *La placa de control está basada en el PIC18F14K50.*

## Alimentación

El microcontrolador opera con 5V mientras que los motores lo hacen con 12V. Dado que el puerto USB ya suministra 5V (con unos 500 mA máx.), se alimentó el micro, los drivers A4988 y resto de componentes desde el puerto USB (P2) y los bobinados de los motores con una fuente separada de 12V. Una observación muy importante es que al ser fuentes separadas se deben unir sus GND, así poseen la misma referencia.

La placa posee 2 conectores Barrel Jack (CON1 y CON2): el primero utilizado para conectar la fuente de alimentación de 12V para los motores paso a paso y el mini torno, y el segundo como salida para conectar el mini torno a la placa y energizarlo (nótese que tienen el mismo conexionado y son intercambiables). Los capacitores C3 y C4 se ubican cerca del conector de alimentación del mini torno para filtrado y desacople. De igual manera, C9 proporciona un filtrado extra de la fuente de alimentación y se ubica en cercanías del conector de entrada de alimentación. Los motores paso a paso están calibrados para funcionar con 1A mientras que el mini torno de corriente continua tiene un consumo de 850 mA (especificado por el fabricante). Entonces sabiendo que se debe alimentar 3 motores paso a paso con 1 Ampere y un motor DC con 0.85 A, se deduce que el consumo máximo es de 3.8 A (unos 45.6W). Comercialmente se encuentran fuentes de 48W, que es la que mejor se adapta. Sin embargo, se escogió una de 60W (5 Amperios máx.) ya que los 3 motores paso a paso se pueden calibrar, en caso de ser necesario, para que produzcan un mayor torque y por ende es ineludible aumentar su consumo.

## Comunicación con la PC usuario

La perforadora recibe las coordenadas de los agujeros a través del software de PC. El medio de comunicación o transmisión de datos que se utilizó fue el protocolo USB, debido que es el estándar de comunicación más habitual que posee un ordenador actual. Es por ello que se escogió un microcontrolador con módulos USB incorporado. El conector USB tipo B (P2 en el diagrama) es la vía por donde se comunica la placa de control y la PC. Como puede observarse en el

esquemático, y según se dijo anteriormente, la alimentación del mismo y demás componentes en la placa es suministrada por dicho puerto. Solo son necesarios dos pines de E/S del micro que se cablean directamente al conector (pines 18 y 19).

Cuestiones sobre USB:
No se explicará el protocolo USB, ya que no es motivo de este trabajo, pero si se abordará algunos temas sobre cómo se ha configurado el modulo USB del microcontrolador PIC. Para información más detallada sobre este protocolo se deja en bibliografía los links correspondientes.
Conector: el conector utilizado es el tipo B, que de acuerdo a la topología de este BUS es el indicado para los dispositivos.
Alimentación: de acuerdo a las especificaciones del USB IF (Foro de Implementadores de Bus USB), un dispositivo USB puede ser alimentado externamente (Self powered) o por el bus (bus Powered), o incluso una combinación de ambos denominados híbridos (esto es muy utilizado en dispositivos portátiles). El bus es el que proveerá la energía la placa de control, por lo que se configuró como Bus Powered.
Velocidad: éste microcontrolador, PIC18F14K50, puede operar en modo velocidad baja (slow speed) o velocidad completa (full speed). Para mayor rendimiento en la transferencia de datos desde el PC a la perforadora y viceversa se configuró en velocidad completa, utilizando las resistencias pull-up internas.
VID&PID: cada dispositivo USB tiene un par de números de 16 bits que lo identifican del resto. Estos valores son el Identificador de dispositivo (PID) y el identificador de fabricante (VID). Microchip provee estos dos valores para utilizarlos en nuestros proyectos y se encuentran en el descriptor de dispositivo.
Clase: el dispositivo USB que se implementa aquí en este proyecto es en realidad un emulador de puerto serie o también llamado COM Virtual por lo que se encuentra dentro de la clase CDC (Clase de dispositivos de comunicación).
Descriptores: Los descriptores son un conjunto de datos almacenados en el dispositivo y que sirven para que, una vez conectado al BUS, el Host "aprenda" sobre él. Cuando se dice que aprenda sobre él se refiere a que conozca sus atributos, requerimientos de energía (Bus o Self Powered), VID & PID, tipos de transferencia que soporta (masiva, interrupción, etc.), tamaño de los endpoints, etc., para cargar el driver apropiado.
Este conjunto de datos está organizado en tablas dentro de lo que se conoce como una jerarquía de descriptores. Algunos de ellos son comunes a todos los dispositivos, mientras que otros son específicos para una función en particular. En el apartado *Firmware*, se explican todos los descriptores utilizados en este proyecto y su configuración.

## Control de motores

El control de motores paso a paso bipolares para los 3 ejes es llevado a cabo utilizando solamente 6 puertos de E/S del PIC como salida (dos por cada motor). Es posible gracias a los drivers A4988 de Pololu (fig. Nº 10).

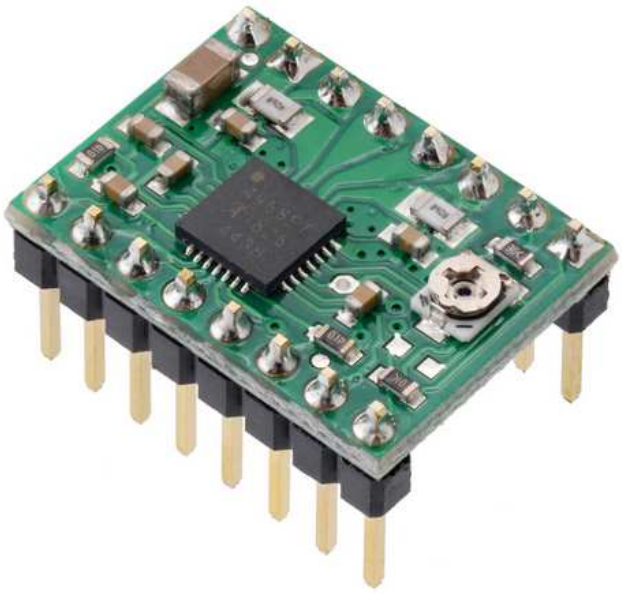


**FIG. Nº 10** *Imagen del driver A4988 de Pololu [9].*

Este driver en realidad es una pequeña PCB fabricada por la empresa POLOLU la cual contiene el chip A4988 diseñado por Allegro MicroSystems y cada uno de ellos es capaz de controlar un solo motor paso a paso bipolar. Cuenta con un preset para ajustar la intensidad de corriente a la que trabajaran las bobinas del motor (de 0 a 2 A) y así aumentar o disminuir el torque o par del mismo. También dicho driver tiene la posibilidad de dividir la cantidad de pasos y lograr mayor precisión (realizar microstepping). Se explica a continuación su implementación en este proyecto.

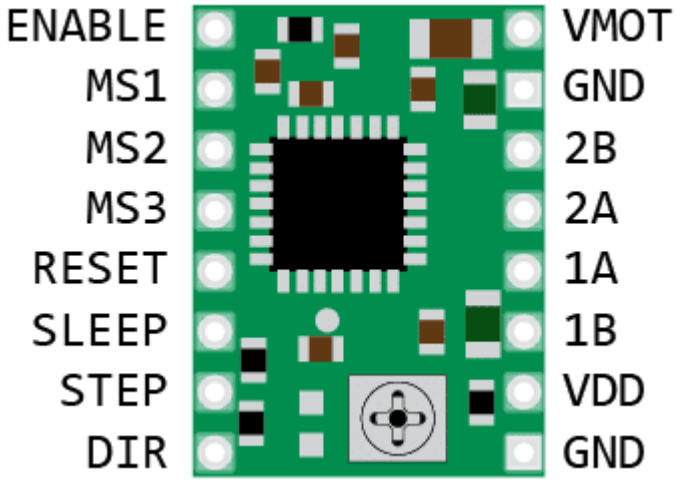



**FIG. Nº 11** *Pinout del driver A4988 [10 ].*

Como puede apreciarse en la fig. 11, los drivers poseen 8 entradas digitales para el total control del motor paso a paso (lado izq. de la figura). Las entradas Enable están conectadas a masa, es decir, los drivers se encuentran siempre activos, mientras que los pines Reset y Sleep están a tensión o '1' lógico, lo que significa que nunca entran en estado de reinicio ni de bajo consumo.
Por otra parte, los terminales STEP y DIR son los que se conectan al microcontrolador: por el pin STEP se aplican pulsos para lograr que el motor correspondiente gire mientras que por el pin DIR se le indica al motor en qué dirección debe hacerlo.
Se determinó que los pines RC1 y RC7, RC2 y RC6 y RC3 y RC5 del PIC controlen los pines STEP y DIR de los drivers de los motores del eje 'X', del eje 'Y' y del eje 'Z', respectivamente.

También posee los terminales MS1, MS2 y MS3, a través de los cuales se logra realizar micropasos o microstepping. De acuerdo con la combinación de valores lógicos de estas entradas será el modo de división de los pasos del motor.

| MS1 | MS2 | MS3 | Microstep Resolution | Excitation Mode |
|---|---|---|---|---|
| L | L | L | Full Step | 2 Phase |
| H | L | L | Half Step | 1-2 Phase |
| L | H | L | Quarter Step | W1-2 Phase |
| H | H | L | Eighth Step | 2W1-2 Phase |
| H | H | H | Sixteenth Step | 4W1-2 Phase |

***Tabla 4*** *Modos de micropasos del driver A4988[7].*

De acuerdo a la tabla anterior se pueden dividir los pasos de los motores en 1 (no existe división), 2, 4, 8 y 16 micropasos. Como se explicó anteriormente en el funcionamiento de la máquina, a los motores que accionan los ejes 'X' e 'Y' se les dividieron los pasos por 16, lo que resulta en colocar los pines MSx a '1' lógico. Por otra parte, para el motor del eje Z, al no necesitar precisión en el desplazamiento del minitorno, no se le aplica la técnica de micropasos y se conectan dichos pines a GND o '0' lógico.

Por último, los pines VDD y GND son de alimentación (5V y 0V, respectivamente), VMOT es por donde se energizan las bobinas de cada motor, que corresponden a 12V y 1A, 1B, 2A y 2B son los terminales de conexión del motor paso a paso.

Los motores a utilizar son bipolares de 1,8º tipo NEMA 17 con alto torque (5Kgf por cm) y una corriente nominal de 1.68A [11]. De acuerdo a pruebas realizadas, con una corriente de 1 A se logra un torque lo suficientemente grande para desplazar los carros de los ejes de la perforadora. Es por ello que se calibraron los drivers para suministrar mencionada corriente a los motores según se explica a continuación.

El chip (no la placa) A4988 posee entre sus 28 pines, un pin llamado Vref por el cual se le debe aplicar una tensión determinada para que el driver suministre la corriente deseada de acuerdo con la siguiente formula:

$$V_{ref} = 8 \cdot I_{max} \cdot R_{cs} \qquad \textit{(Ecu. 5)}$$

Donde:

Vref: es la tensión de referencia que se ajusta con el preset que posee el driver.

Imax: es la corriente que se le desea suministrar al motor.

Rcs: es un par de resistencias de sensado de corriente (shunts), incluidas en la placa fabricada por Pololu y de un valor de 0.1Ω.

Entonces, sabiendo la corriente máxima a la que se quiere programar y la resistencia de sensado, se calcula la tensión necesaria Vref. Utilizando la ecuación 5 resulta un valor de 0.8V. Por ende, con la ayuda de un voltímetro se ajusta el preset hasta dejarlo en 0.8V.

Lo anterior es válido para los drivers de los motores del eje 'X' e 'Y', pero no así para el driver del motor del eje 'Z' por una razón: este último no aplica la técnica de micropasos y por ende la fórmula que se utiliza es la siguiente:

$$I_{max} = \sqrt{I_{coil1} + I_{coil2}} \qquad \textit{(Ecu. 6)}$$

En esta situación Icoil1 e Icoil2 valen 1A, que es la corriente requerida, pero al estar las dos bobinas energizadas al mismo tiempo la corriente resultante es la suma vectorial de ambas. Por ello el valor de Imax es de 1.4A (ecuación 6). Aplicando la ecuación 5 se obtiene Vref = 1.12V.

Para finalizar con la cuestión de los drivers, el fabricante recomienda colocar capacitores electrolíticos de al menos el doble de tensión de la que se alimentan los motores para absorber los picos de tensión que pudieran originarse durante el normal funcionamiento del mismo y evitar dañarlo. Dichos capacitores corresponden a C6, C7 y C8 en la fig. 12.

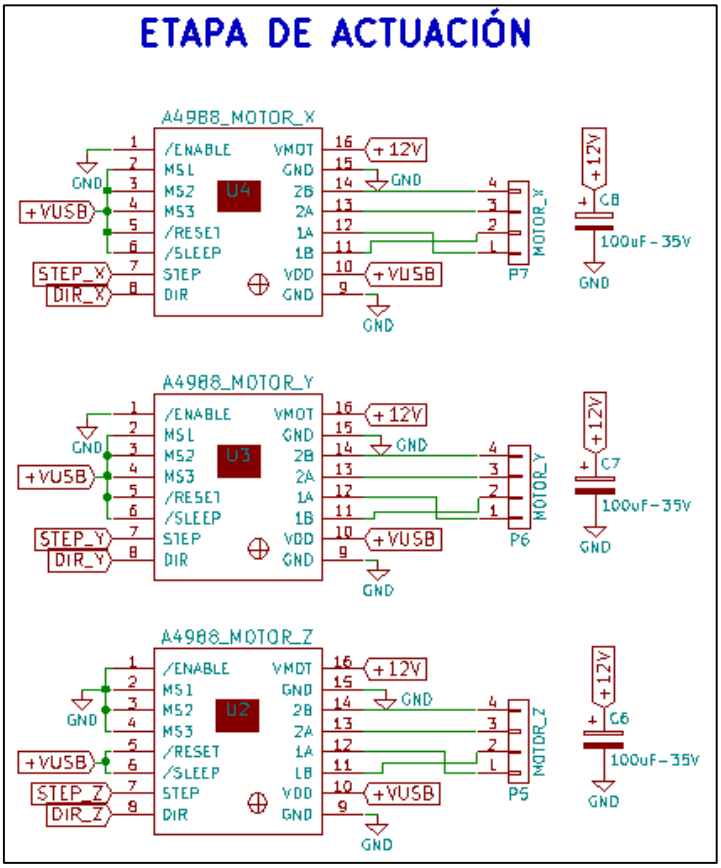



**FIG. Nº 12** *Detalle de los drivers A4988 en el circuito de control.*

## Indicador LED

Este LED D1 muestra el estado en el que se encuentra la comunicación entre la placa de control, específicamente el modulo USB, y el PC. Este indica si no se encuentra conectado al PC, si está conectado, pero todavía no se ha configurado o si ya se configuró. Se darán más detalles cuando se explique el firmware.

## Programación ICSP

El PCB también dispone de un conector para la programación del microcontrolador, así se evita tener que extraer el chip para reprogramar el firmware y, por ende, lograr mayor practicidad. Se puede observar en la fig. 8 como *P1*.

## Sensores

Para detectar el arribo de los carros en los ejes se colocaron sensores mecánicos. Estos no son más que unos finales de carrera montados en una PCB junto a un conector.

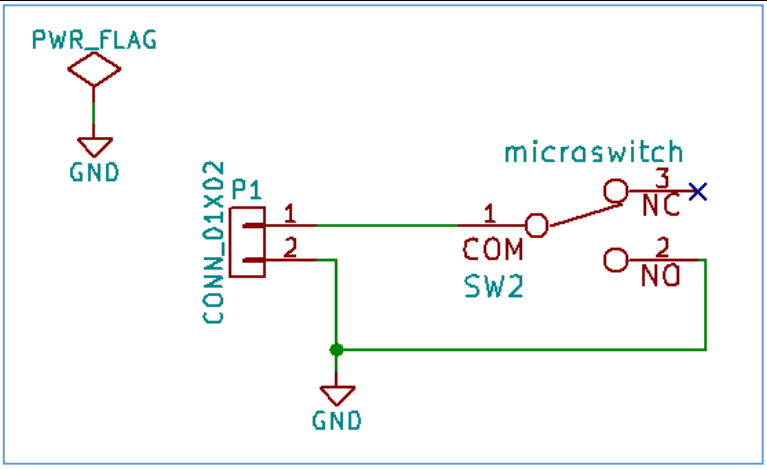



**FIG. Nº 13** *Diagrama esquemático del circuito sensor.*

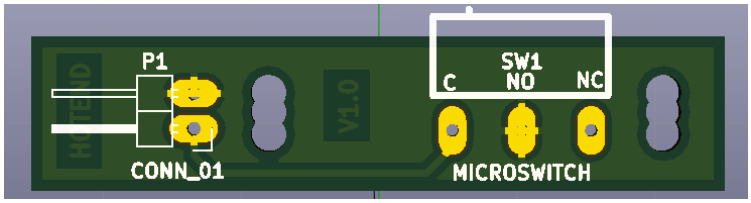



**FIG. Nº 14** *Diseño de la PCB de los sensores.*

Cuando el carro vertical o 'Z' que soporta el mini torno debe perforar, desciende hasta activar el sensor ubicado en la parte inferior. En ese momento comienza a ascender hasta que activa el sensor superior, culminando la perforación.
Por otro lado, en el arranque de la maquina o cuando ésta se reinicia por software, los carros correspondientes al eje 'X' e 'Y' se desplazan de manera tal hasta encontrarse con sus sensores respectivos, activándolos y deteniendo los carros. Por ende, se implementaron cuatro sensores mecánicos como los que se muestran en la figura anterior.
Estos sensores se conectan mediante cables a través del conector de 12 pines denominado P3 de la placa de control (ver fig. 8).

## Elaboración de la PCB

En las siguientes figuras se muestran imágenes del CAD de diseño de la placa de control. Las imágenes de la placa ya elaborada se mostrarán en más adelante en el apartado *Montaje y ensayo real del dispositivo.*

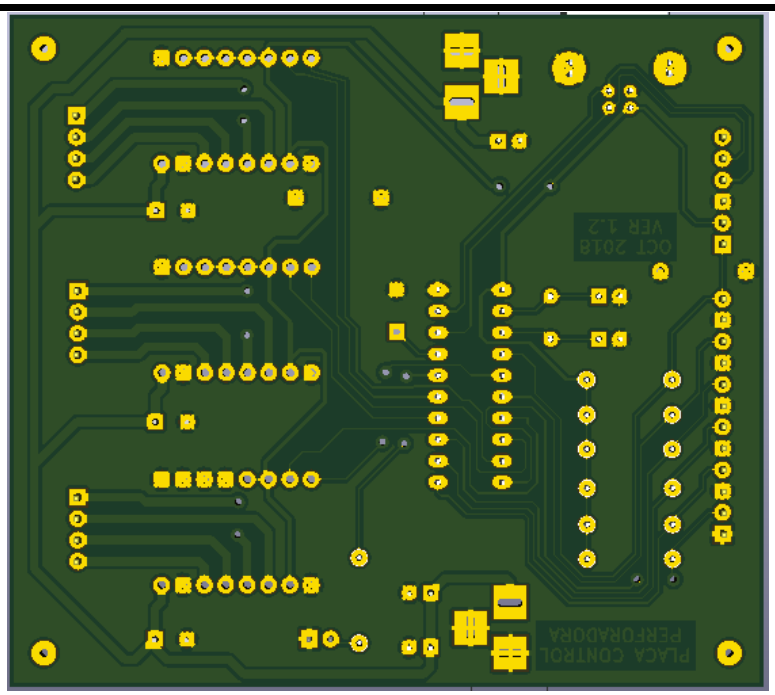



**FIG. Nº 15** *Imagen de la PCB de control del lado inferior.*

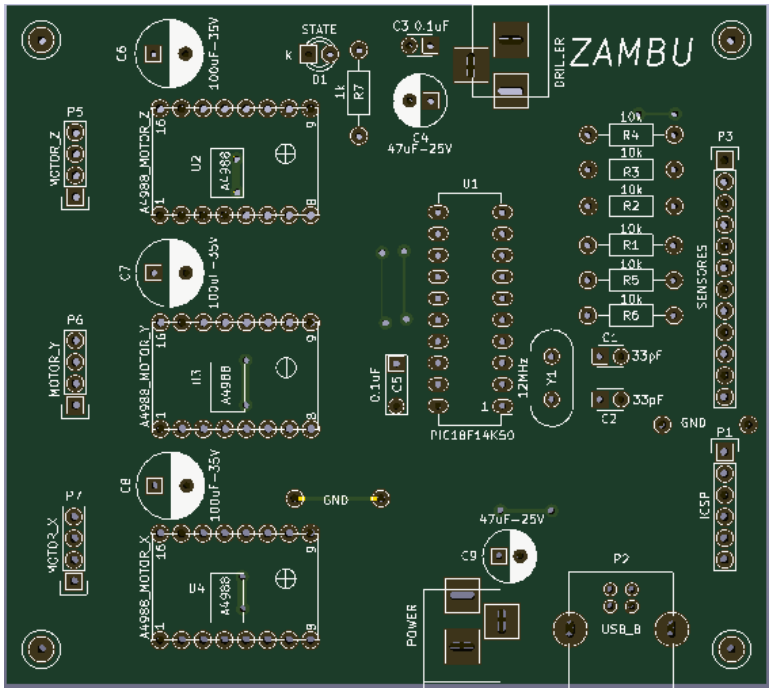



**FIG. Nº 16** *Imagen de la PCB de la misma placa, pero del lado de los componentes.*

# Firmware

El firmware es el código que se escribe para el microcontrolador PIC. Está escrito en C, en el entorno de desarrollo MPLAB IDE 3.15 y compilado con XC8 V1.37.
Se enumeran los diferentes archivos fuentes (.c) y cabecera (.h) utilizados en el proyecto:

| Archivos cabecera (.h) | Archivos fuentes (.c) |
|---|---|
| *app_device_cdc_basic.h* | *main.c* |
| *app_led_usb_status.h* | *app_device_cdc_basic.c* |
| *io_mapping.h* | *app_led_usb_status.c* |
| *leds.h* | *leds.c* |
| *power.h* | *usb_descriptors.c* |
| *system.h* | *system.c* |
| *system_config.h* | *usb_device.c* |
| *usb.h* | *usb_device_cdc.c* |
| *usb_ch9.h* | |
| *usb_common.h* | |
| *usb_config* | |
| *usb_device.h* | |
| *usb_device_cdc.h* | |
| *usb_device_local.h* | |
| *usb_hal.h* | |
| *usb_hal_pic18.h* | |

***Tabla 5** Achivos fuentes y cabecera que conforman el firmware del microcontrolador.*

Los archivos anteriores, a excepción de *main.c*, y *app_device_cdc_basic.h/.c,* corresponden a la Microchip Library for Applications –MLA (o Biblioteca para Aplicaciones de Microchip). Esta biblioteca es un conjunto de archivos (código fuentes, archivos de cabecera, stacks, drivers y demás documentos) que ofrece la compañía fabricante de los microcontroladores PIC para que los diferentes usuarios puedan utilizarlos en sus proyectos. Se pueden descargar gratuitamente en la página oficial de Microchip.

A continuación se muestra el diagrama de flujo del programa ejecutado por el microcontrolador y luego, a partir de éste, se explicará cómo funciona. En el Anexo III se detalla el código.

**Diagrama de flujo del firmware:**

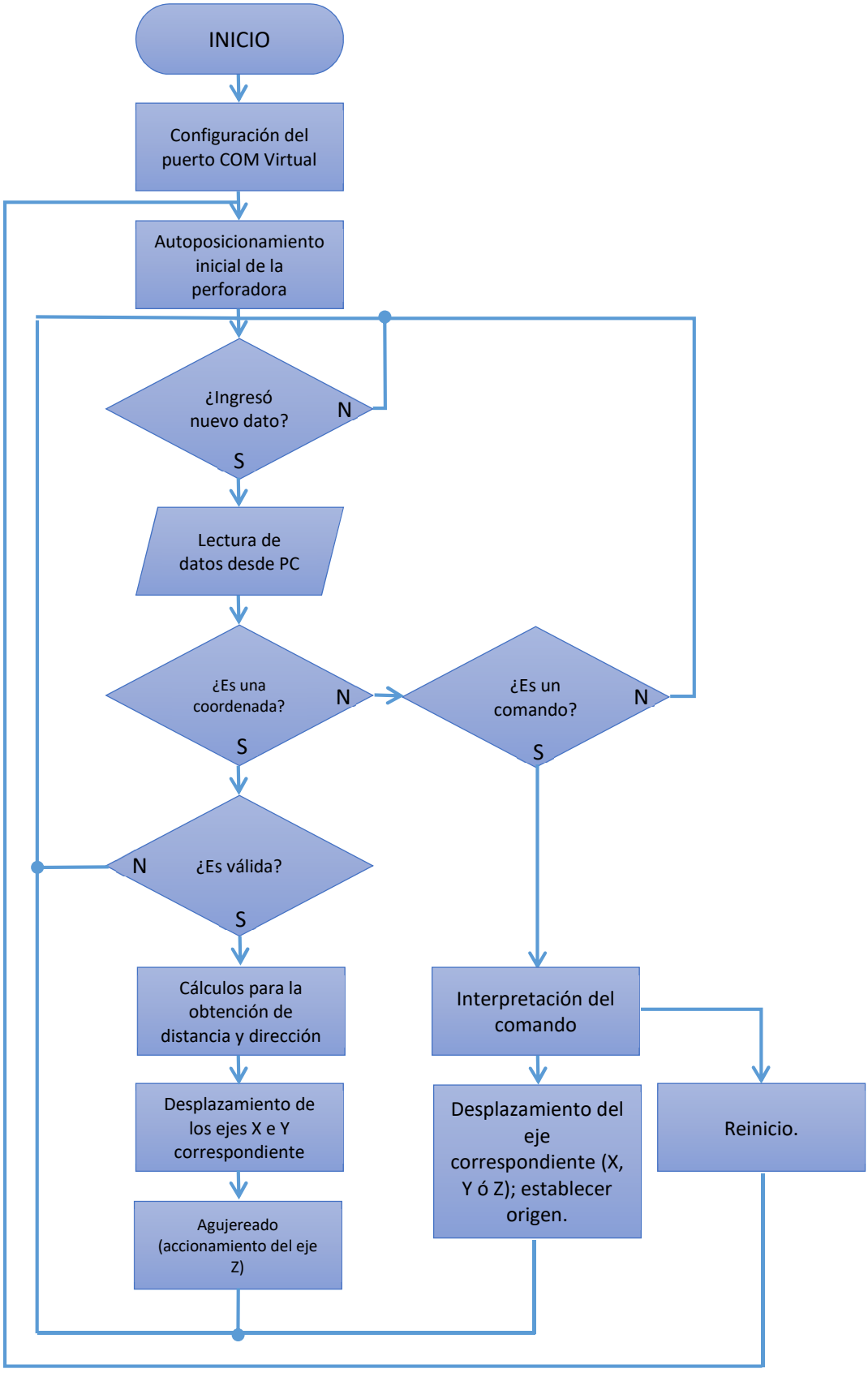



**FIG. Nº 17** *Diagrama de flujo del firmware.*

El programa inicia configurando los fusibles (ubicados en el archivo *system.c*), configurando los puertos del PIC que se utilizaran como E/S generales e inicializando y habilitando el modulo USB. Los puertos RC0 al RC7 (con excepción de RC4 que no se utiliza) se configuraron como salidas: RC0 corresponde al LED que indica el estado de la comunicación de la perforadora y la PC; RC1 a RC3 se utilizaron para proveer los pasos a los motores por medio de los drivers A4988; y RC5 a RC7 indican el sentido de giro de dichos motores.
Por otra parte, los puertos RB4 a RB7 se establecieron como entradas digitales ya que deben leer el estado de los sensores de fin de carro.

Luego de la inicialización de puertos y modulo USB comienza a funcionar la perforadora (en este punto ya tomó conexión con la aplicación de la PC). Antes que nada, los carros se desplazan a la posición de inicio, o sea, los carros se posicionan siempre en el mismo lugar para que luego el usuario los coloque en el área deseada. Los cuatro sensores implementados son los que se encargan de posicionar dichos carros en el lugar de reinicio.
A partir de allí es donde la placa de control queda a la espera de nuevos datos, datos que deben llegar de la aplicación. Cuando detecta nuevos datos en el buffer de entrada se ejecutan unos algoritmos para que los procese y actúe de acuerdo a la información que llegó.

Antes de pasar a explicar cómo se manipula la información arribada al PIC, se aclarará una cuestión. Los datos que pueden llegar desde la PC pueden ser datos de dos tipos: coordenadas, es decir, parte de código G que indica a la perforadora donde perforar; y comandos, que son datos específicos para mover alguno de los tres ejes en forma manual en cualquier dirección. Cada uno tiene preestablecido un formato. En el caso de las coordenadas tiene la siguiente forma: X001122Y004455, es decir, son 14 bytes que llegan desde la aplicación. Para los comandos, lo que se reciben, en cambio son unos caracteres especiales cuya incidencia se muestra en la siguiente tabla.

| Comando | Acción a realizar |
|---|---|
| carácter ‘f’ | Desplaza el carro eje X en sentido positivo una vez. |
| carácter ‘F’ | Desplaza el carro eje X en sentido positivo en forma continua. |
| carácter ‘b’ | Desplaza el carro eje X en sentido negativo una vez. |
| carácter ‘B’ | Desplaza el carro eje X en sentido negativo en forma continua. |
| carácter ‘l’ | Desplaza el carro eje Y en sentido positivo una vez. |
| carácter ‘L’ | Desplaza el carro eje Y en sentido positivo en forma continua. |
| carácter ‘r’ | Desplaza el carro eje Y en sentido negativo una vez. |
| carácter ‘R’ | Desplaza el carro eje Y en sentido negativo en forma continua. |
| carácter ‘u’ | Desplaza el carro eje Z en sentido positivo una vez. |
| carácter ‘d’ | Desplaza el carro eje Z en sentido negativo una vez. |
| carácter ‘o’ | Condición de reinicio. |

***Tabla 6*** *lista de comandos aceptados por la placa de control.*

Dicho esto, el firmware primero verifica si es una coordenada o un comando. Si llegase a ser una coordenada, verifica que sea válida, es decir, que tenga el formato adecuado. Si es así procede para calcular las distancias y direcciones a las que debe desplazarse los carros del eje ‘X’ e ‘Y’ y culminar con el accionamiento del eje Z para perforar en dicho punto.

Si es un comando comprueba cuál de todos es y realiza la acción correspondiente de acuerdo a la tabla anterior. En cualquier caso, si se recibe algún dato que no esté predefinido es ignorado, no ejecutándose acción alguna, y se regresa al inicio del programa a la espera de uno nuevo. Este ciclo se repite continuamente mientras la perforadora esté conectada a la PC.

Cuando se recibe una coordenada ésta es, en realidad, la ubicación del punto a perforar y que en base a ella se calcula la distancia y dirección que deben desplazarse los ejes 'X' e 'Y'. Antes de proceder a explicarlo se hablará sobre el formato de las mismas para poder comprender mejor como se calculan dichas distancias.

Como se dijo anteriormente, el software que se utilizó fue KICAD. Este, dentro de las muchas características que posee, brinda la posibilidad de generar un archivo de perforado, llamado *.drl. A continuación se muestra un ejemplo del mismo.

```
M48
;DRILL file {KiCad 4.0.1-stable} date 03/02/2016 17:15:26
;FORMAT={3:3/ absolute / metric / keep zeros}
FMAT,2
METRIC,TZ
T1C0.762
T2C0.800
T3C0.800
T4C0.900
T5C1.001
T6C1.016
T7C1.270
%
G90
G05
M71
T1
X055372Y019478
X055372Y017478
X055372Y015478
X055372Y013478
X055372Y011478
X055372Y009478
T2
X014605Y022352
X014605Y019812
X014605Y017272
X014605Y014732
X014605Y012192
X014605Y009652
X014605Y007112
X014605Y004572
```

**FIG. Nº 18** *Parte del archivo .drl generado por el CAD de diseño de PCB.*

Observando la imagen anterior es fácil ver que se trata de un programa en código G o G-code. Se encuentran las funciones misceláneas (M48, M71, etc), preparatorias (G90, G05) y las de numero de herramienta (T1, T2, T3, etc.). Pero lo que interesa aquí son las funciones X e Y. Lo más importante a destacar es el formato que poseen las coordenadas.
Las coordenadas para el agujereado tienen la siguiente forma: "X112233Y334455" (sin comillas), y se interpreta de la siguiente manera: el valor de cada eje está en milímetros y posee tres dígitos que corresponde a las centenas, decenas y unidades y otros 3 que corresponden a las décimas, centésimas y milésimas de milímetro, respectivamente. Entonces siguiendo el ejemplo anterior la posición del agujero en el eje X seria a 112,233 milímetros del punto de referencia. Mientras tanto, la del eje Y seria a 334,455 milímetros.

Entonces, lo que el algoritmo de cálculo debe realizar es tomar el número de las centenas y multiplicarlo por 100, luego tomar las decenas y multiplicarlo por 10 y las unidades quedan iguales. Luego continúa tomando el digito correspondiente a las décimas y lo multiplica por 0.1. De idéntica forma procede con los últimos dos dígitos, multiplicándolos por 0.01 y 0.001. Cada valor obtenido se va acumulando en una variable y se obtiene un número real que corresponde a la distancia a desplazarse.

Utilizando la ecuación 4 antes descripta se calculan los pulsos necesarios a aplicar a los motores paso a paso y que estos giren moviendo el carro.

Con respecto a la dirección del desplazamiento, es decir, si en sentido positivo o negativo, se toma el valor acumulado anteriormente y se lo compara con el obtenido de la nueva coordenada (si es mayor o menor al anterior). De este modo se obtiene la dirección.

Por último, al llegar al punto solicitado se procede a activar el carro del eje Z, donde está montado el mini torno, el cual desciende hasta llegar al sensor inferior y en dicho descenso perfora la placa y luego cambia de sentido hasta llegar a la parte superior.

La placa de control posee, según se ha visto, un diodo LED (D1) que se utiliza para mostrar el estado en el que se encuentra la perforadora. Dichos estados se muestran en la tabla siguiente y es el firmware el encargado de actualizar su estado.

| Estado del LED | Estado de la perforadora |
|---|---|
| **Apagado** | No energizada o no conectada a PC |
| **Intermitente (lento)** | Conectada a PC, pero no se encuentra lista para funcionar (se encuentra en proceso de enumeración) |
| **Intermitente (rápido)** | Conectada al PC y listo para funcionar |

***Tabla 7*** *Estados de la perforadora según LED indicador.*

## Descriptores

Como se mencionó anteriormente, los descriptores se organizan como un conjunto jerárquico como se ilustra en la siguiente imagen. A continuación se detallan los mismos, es decir, los valores que se les asignó a cada campo en cada descriptor para este proyecto.

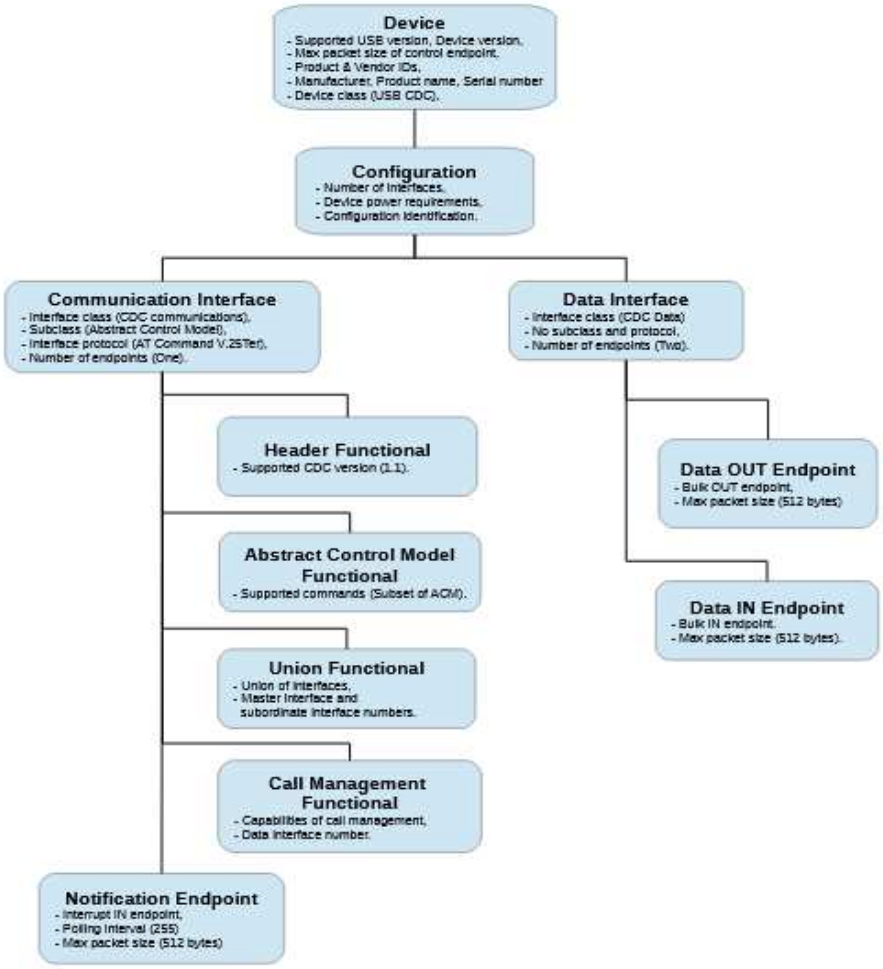



**FIG. Nº 19** *Jerarquía de descriptores utilizados para el firmware [8].*

Descriptor de dispositivo*: Brinda información como la versión de USB que utiliza, a qué clase de dispositivo pertenece, el VID&PID, los diferentes tipos de configuraciones y número de serie, entre otros.*

| Desplazamiento | Campo | Tamaño (Bytes) | Valor |
|---|---|---|---|
| **0** | bLength | 1 | 0x12 |
| **1** | bDescriptorType | 1 | 0x01 |
| **2** | bcdUSB | 2 | 0x0200 |
| **4** | bDeviceClass | 1 | 0x02 |
| **5** | bDeviceSubClass | 1 | 0x00 |
| **6** | bDeviceProtocol | 1 | 0x00 |
| **7** | bMaxPacketSize0 | 1 | 0x08 |
| **8** | idVendor | 2 | 0x04D8 |
| **10** | idProduct | 2 | 0x000A |
| **12** | bcdDevice | 2 | 0x0100 |
| **14** | iManufacturer | 1 | 0x01 |
| **15** | iProduct | 1 | 0x02 |
| **16** | iSerialNumber | 1 | 0x00 |
| **17** | bNumConfigurations | 1 | 0x01 |

***Tabla 8** Descriptor de Dispositivo y sus valores.*

*Descriptor de configuración:* brinda información sobre una configuración especifica del dispositivo (puede tener más de una), como por ejemplo número de interfaces, tipo de alimentación (BUS o SELF POWERED), corriente máxima necesaria, ente otras. En este proyecto el dispositivo posee una sola configuración.

| Desplazamiento | Campo | Tamaño (Bytes) | Valor |
|---|---|---|---|
| **0** | bLength | 1 | 0x09 |
| **1** | bDescriptorType | 1 | 0x02 |
| **2** | wTotalLength | 2 | 0x0043 |
| **4** | bNumInterfaces | 1 | 0x02 |
| **5** | bConfigurationValue | 1 | 0x01 |
| **6** | iConfiguration | 1 | 0x00 |
| **7** | bmAttributes | 1 | 0x80(BUS POWERED-NO REMOTE WEKAUP |
| **8** | bMaxPower | 1 | 200 (400mA) |

***Tabla 9*** *Valores del Descriptor de Configuración.*

*Descriptor de interfaz de comunicacion(1):* describe una interfaz dentro de cada configuración. Lo que más se destaca es la cantidad de endpoints que posee. También se declara la clase del dispositivo USB, que en este caso es CDC. Esto permite al Host cargar el driver apropiado para esa funcionalidad.

| Desplazamiento | Campo | Tamaño (Bytes) | Valor |
|---|---|---|---|
| **0** | bLength | 1 | 0x09 |
| **1** | bDescriptorType | 1 | 0x04 |
| **2** | bInterfaceNumber | 1 | 0x00 |
| **3** | bAlternateSetting | 1 | 0x00 |
| **4** | bNumEndpoints | 1 | 0x01 |
| **5** | bInterfaceClass | 1 | 0x02 (CDC Class) |
| **6** | bInterfaceSubClass | 1 | 0x02 (Abstract Control Model) |
| **7** | bInterfaceProtocol | 1 | 0x01 (V25ter, AT commands) |
| **8** | iInterface | 1 | 0x00 |

***Tabla 10*** *Descriptor de Interfaz de comunicación.*

En el caso de dispositivos CDC existe un descriptor especifico llamado Descriptor de Función (Functional Descriptor), que en realidad es un conjunto de otros descriptores (ver figura anterior). Estos son los siguientes: Descriptor de Función de Cabecera (*Header Functional*), Descriptor de Función de Control de Modelo Abstracto (*Abstract control model functional*), Descriptor de Función de Unión (*Union Functional*) y Descriptor de función de Administración de Llamada (*Call Management Functional*).

| Desplazamiento | Campo | Tamaño (Bytes) | Valor |
|---|---|---|---|
| **0** | bFunctionLength | 1 | 0x05 |
| **1** | bDescriptorType | 1 | 0x24 |
| **2** | bDescriptorSubType | 1 | 0x00 |
| **3** | bcdCDC | 2 | 0x10,0x01 |

***Tabla 11*** *Descriptor de Función de cabecera (Header Functional Descriptor).*

| Desplazamiento | Campo | Tamaño (Bytes) | Valor |
|---|---|---|---|
| **0** | bFunctionLength | 1 | 0x04 |
| **1** | bDescriptorType | 1 | 0x24 |
| **2** | bDescriptorSubType | 1 | 0x02 |
| **3** | bmCapabilities | 1 | 0x02 |

***Tabla 12*** *Descriptor de Función de Control de Modelo Abstracto (Abstract control model functional Descriptor)*

| Desplazamiento | Campo | Tamaño (Bytes) | Valor |
|---|---|---|---|
| **0** | bFunctionLength | 1 | 0x05 |
| **1** | bDescriptorType | 1 | 0x24 |
| **2** | bDescriptorSubType | 1 | 0x06 |
| **3** | bMasterInterface | 1 | 0x00 |
| **4** | bSlaveInterface0 | 1 | 0x01 |

***Tabla 13*** *Descriptor de Función de Unión (Union Functional Descriptor).*

| Desplazamiento | Campo | Tamaño (Bytes) | Valor |
|---|---|---|---|
| **0** | bFunctionLength | 1 | 0x05 |
| **1** | bDescriptorType | 1 | 0x24 |
| **2** | bDescriptorSubType | 1 | 0x01 |
| **3** | bmCapabilities | 1 | 0x00 |
| **4** | dDataInterface | 1 | 0x01 |

***Tabla 14*** *Descriptor de función de Administración de Llamada (Call Management Functional Descriptor).*

*Descriptores de Endpoint:* Un endpoint es una porción de memoria Ram del dispositivo en donde se almacenan los datos recibidos del Host (PC) o depositan para enviárselos al mismo. El Host debe saber la dirección de los mismos, el tipo de transferencia al que están configurados, el tamaño máximo del paquete que puede procesar, etc. Dicha información es la que está contenida en este descriptor.

| Desplazamiento | Campo | Tamaño (Bytes) | Valor |
|---|---|---|---|
| **0** | bLength | 1 | 0x07 |
| **1** | bDescriptorType | 1 | 0x05 |
| **2** | bEndpointAddress | 1 | 0x81 (Endpoint 1 IN) |
| **3** | bmAtributes | 1 | 0x03 (Interrupt Transfer) |
| **4** | wMaxPacketSize | 2 | 0x0008 |
| **6** | bInterval | 1 | 0x02 |

***Tabla 15*** *Descriptor de Endpoint 1:*

| Desplazamiento | Campo | Tamaño (Bytes) | Valor |
|---|---|---|---|
| **0** | bLength | 1 | 0x09 |
| **1** | bDescriptorType | 1 | 0x04 |
| **2** | bInterfaceNumber | 1 | 0x01 |
| **3** | bAlternateSetting | 1 | 0x00 |
| **4** | bNumEndpoints | 1 | 0x02 |
| **5** | bInterfaceClass | 1 | 0x0A (data Class) |
| **6** | bInterfaceSubClass | 1 | 0x00 |
| **7** | bInterfaceProtocol | 1 | 0x00 |
| **8** | iInterface | 1 | 0x00 |

***Tabla 16*** *Descriptor de interfaz de datos.*

| Desplazamiento | Campo | Tamaño (Bytes) | Valor |
|---|---|---|---|
| **0** | bLength | 1 | 0x07 |
| **1** | bDescriptorType | 1 | 0x05 |
| **2** | bEndpointAddress | 1 | 0x02 (Endpoint 2 OUT) |
| **3** | bmAtributes | 1 | 0x02 (Bulk Transfer) |
| **4** | wMaxPacketSize | 2 | 0x000E |
| **6** | bInterval | 1 | 0x00 |

***Tabla 17*** *Descriptores de Endpoint 2.*

Notar el tamaño máximo del paquete del EP2 OUT: 14 bytes (0x000E). esto está ligado al formato de las coordenadas, que según se analizó está compuesto por 14 bytes y por ello se configuró así.

| Desplazamiento | Campo | Tamaño (Bytes) | Valor |
|---|---|---|---|
| **0** | bLength | 1 | 0x07 |
| **1** | bDescriptorType | 1 | 0x05 |
| **2** | bEndpointAddress | 1 | 0x82 (Endpoint 2 IN) |
| **3** | bmAtributes | 1 | 0x02 (Bulk Transfer) |
| **4** | wMaxPacketSize | 2 | 0x0040 |
| **6** | bInterval | 1 | 0x00 |

***Tabla 18*** *Descriptores de Endpoint 2.*

Descriptores de string: son opcionales y solo brindan información sobre nombre dispositivo, fabricante, versión, etc.

| Desplazamiento | Campo | Tamaño (Bytes) | Valor |
|---|---|---|---|
| **0** | bLength | 1 | 0x04 |
| **1** | bDescriptorType | 1 | 0x03 |
| **2** | wLangID | 2 | 0x0409 |

***Tabla 19*** *Descriptor de string 0.*

| Desplazamiento | Campo | Tamaño (Bytes) | Valor |
|---|---|---|---|
| **0** | bLength | 1 | 0x34 |
| **1** | bDescriptorType | 1 | 0x03 |
| **2** | bString | 50 | "Microchip Technology Inc." |

***Tabla 20*** *Descriptor de string 1.*

| Desplazamiento | Campo | Tamaño (Bytes) | Valor |
|---|---|---|---|
| **0** | bLength | 1 | 0x34 |
| **1** | bDescriptorType | 1 | 0x03 |
| **2** | bString | 50 | "CDC RS-232 Emulation Demo" |

***Tabla 21*** *Descriptor de string 2.*

Drivers: Microchip provee el driver o controlador para el PC correspondiente y también el archivo .inf. Un driver o controlador de dispositivo es un software que permite que las aplicaciones (como la creada en este proyecto) puedan acceder a los dispositivos. El driver utilizado en este proyecto permite utilizar al puerto USB emulando un puerto COM serial. Microchip ofrece dicho driver llamado mchpcdc. Con ello se evita la ardua tarea que podría implicar desarrollar un driver para el dispositivo.

Una vez que el host detecta un nuevo dispositivo USB y aprende sobre el de acuerdo a la información contenida en los descriptores debe decidir que controlador le asigna al dispositivo. Windows utiliza los archivos .inf para ubicar el driver correspondiente. Un archivo .inf (archivo de información configuración de dispositivo o device-setup information file) es un archivo de texto que contiene información sobre uno o más dispositivos en una *clase de configuración*. No se entrará en detalle aquí sobre ello, pero resumidamente se puede decir que ciertos dispositivos son configurados de la misma manera por el sistema operativo cuando estos son conectados a una PC y por ello se dice que pertenecen a una determinada clase de configuración. El archivo .inf contiene precisamente esa información, le avisa al sistema operativo a qué clase pertenece y en base a ello lo configura. En este caso particular la clase a la que pertenece este dispositivo es la de Puertos o Ports.
Pero no es la única información que provee. Informa también que driver utilizar y el VID y PID, entre otros. Estos últimos, como se aclaró anteriormente, son números de 16 bits que definen unívocamente a un dispositivo y tienen que coincidir con el que se definió en el firmware anteriormente (en el Descriptor de Dispositivo) para que le asigne el driver apropiado.
Una aclaración importante es que en Windows 8 y en versiones anteriores es necesario el archivo .inf. Sin embargo, en la versión 10 de este Sistema Operativo, ya no lo es.

# Aplicación para PC

El control de la perforadora se realiza mediante una aplicación para PC. El lenguaje utilizado para escribir el código de programa fue el Visual C++ Express Edition 2008. Es un lenguaje orientado a objetos, y a pesar de ser la versión limitada, posee una gran cantidad de herramientas para llevarlo a cabo. A continuación se muestra una imagen de la misma.

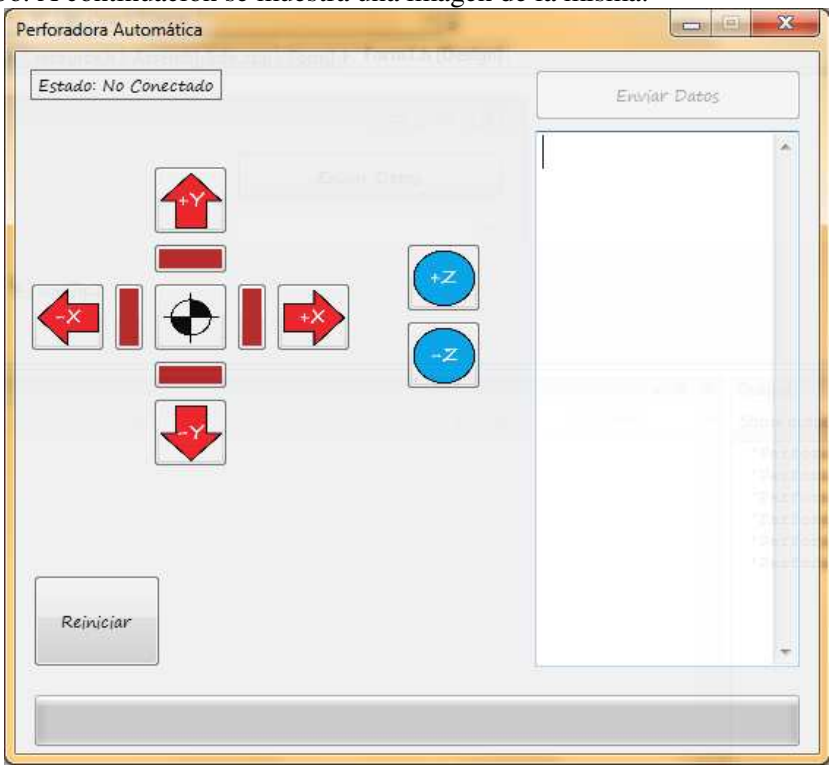



**FIG. Nº 20** *Captura de pantalla de la aplicación de PC.*

## Funcionamiento

Se explicarán las partes que conforman el formulario (ventana) de la aplicación. Luego se adentrará en el código. La aplicación posee 13 botones, de los cuales 10 son para movimiento manual de los carros 'X', 'Y' y 'Z', uno para establecer el origen de pieza, otro para reiniciar la aplicación y la perforadora y por último, un botón que envía las coordenadas de los agujeros hacia la placa.

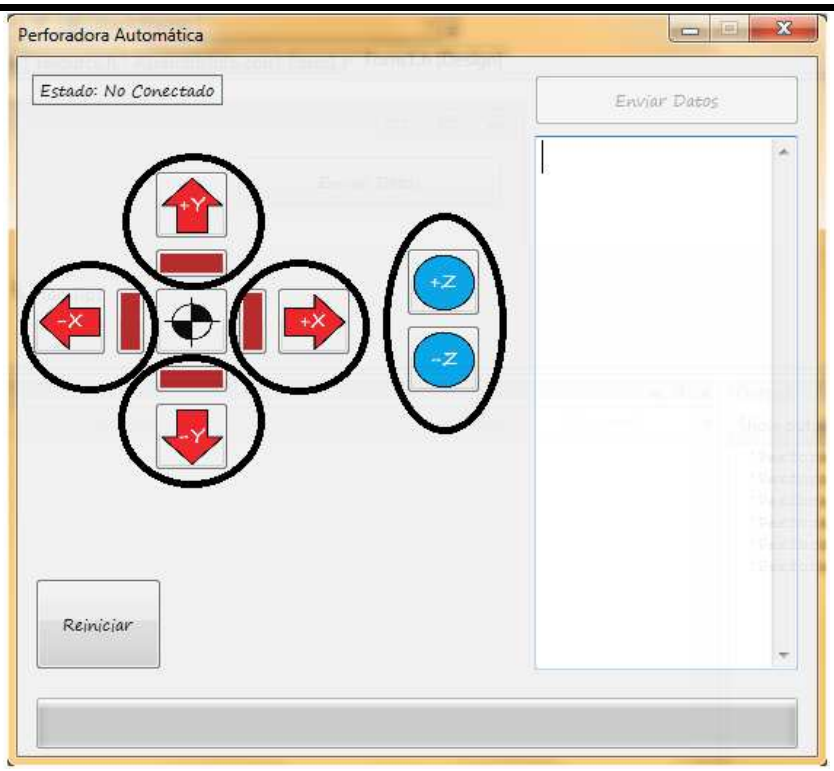



**FIG. Nº 21** *Botones para control manual de los ejes.*

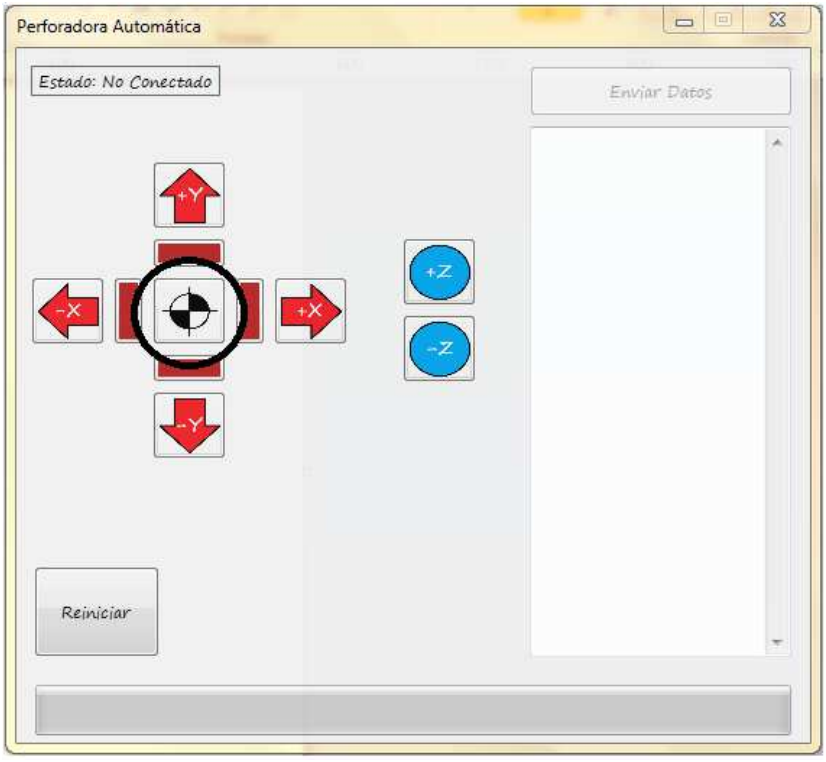



**FIG. Nº 22** *Botón establecimiento de origen.*

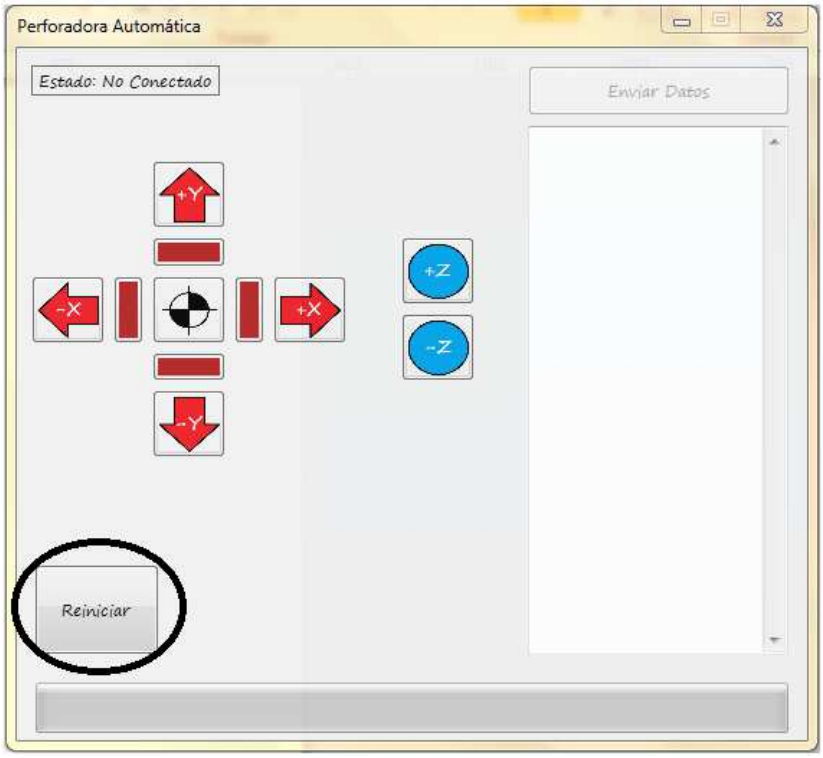



**FIG. Nº 23** *Botón de reinicio.*

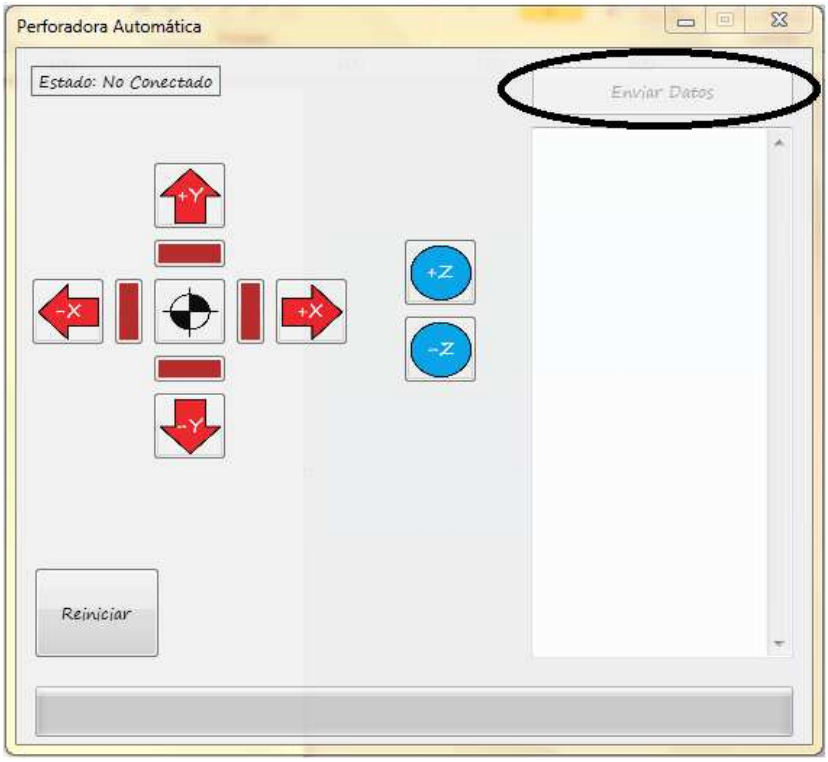



**FIG. Nº 24** *Botón para enviar el código G hacia la placa control. Debajo se encuentra la caja de texto.*

También posee una caja de texto, lugar donde se ingresa las coordenadas ubicada justo por debajo del botón *Enviar Datos.*

Como elementos indicadores existe una barra de estado en la parte inferior del formulario, que

indica el progreso del proceso de perforado y una etiqueta en la parte superior izquierda que indica si la perforadora se ha conectado al USB y, en caso de que así sea, a que puerto COM virtual.

El funcionamiento es muy sencillo e intuitivo. Se comienza conectando la placa de control de la Perforadora. Luego de unos segundos el programa le asigna un puerto COM Virtual y se habilita el botón *Enviar Datos* (notar que aparece desactivado en la imagen anterior cuando no se ha conectado). En este punto ya se puede accionar la máquina.
Lo primero que se hace es posicionar el taladro manualmente a la posición que se desee utilizando los botones que controlan los ejes. Para los ejes X e Y existe un par de botones para cada sentido de desplazamiento;

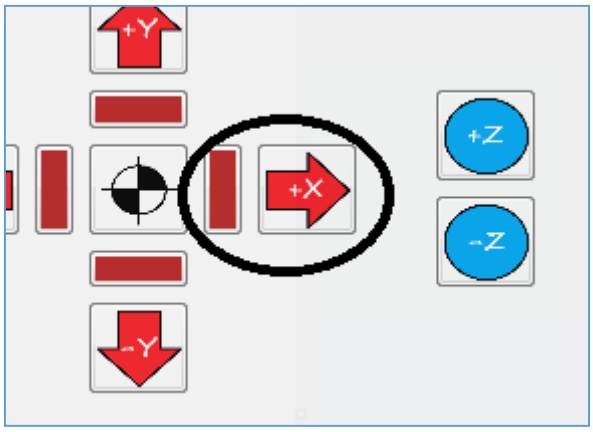



**FIG. Nº 25** *Par de botones para controlar un sentido de avance del eje X*

El botón más grande realiza un desplazamiento del eje correspondiente en forma continua y no se detendrá hasta que se oprima cualquier otro botón. Esto permite un avance automático del carro. Cuando se sitúa en la posición deseada se termina de ajustar con el botón pequeño, que también desplaza el carro, pero un paso por cada vez que se oprime. Para el eje Z es más sencillo: como en este eje no se necesita precisión del carro, ya que solo realiza un perforado, se implementaron 2 botones, para descender y ascender el mini torno (Ver tabla 7).
Una vez que se posiciona el taladro en lo que va a ser el origen se oprime el botón para establecer dicho origen. Lo que sucede aquí es que se envía una coordenada en la que el eje X y el eje Y están en cero, es decir, realmente se está enviando "X000000Y000000" (sin las comillas). Entonces a partir de allí se toma como referencia o cero de pieza ese punto.
Con esto ya se puede enviar el archivo drill generado por KICAD copiándolo y pegándolo en la caja de texto. El firmware, según lo explicado anteriormente, aceptará coordenadas válidas y desestimará los demás datos. En este punto, solo queda esperar a que la perforadora termine de agujerear y puede verse el estado a través la barra de progreso en la parte inferior de ventana de programa.

## Código

El programa cuenta con un temporizador, el cual lanza un evento cada 1 segundo y en dicho evento se ejecuta un método que verifica si algún dispositivo USB CDC fue conectado; si es el caso verifica el número de puerto y lo abre y también habilita el botón *Enviar Datos*. Sino detecta la placa de control de la Perforadora conectada a la PC cierra el puerto COM. En cualquiera de los dos casos se actualiza la etiqueta que se encuentra en la parte superior izquierda del formulario.

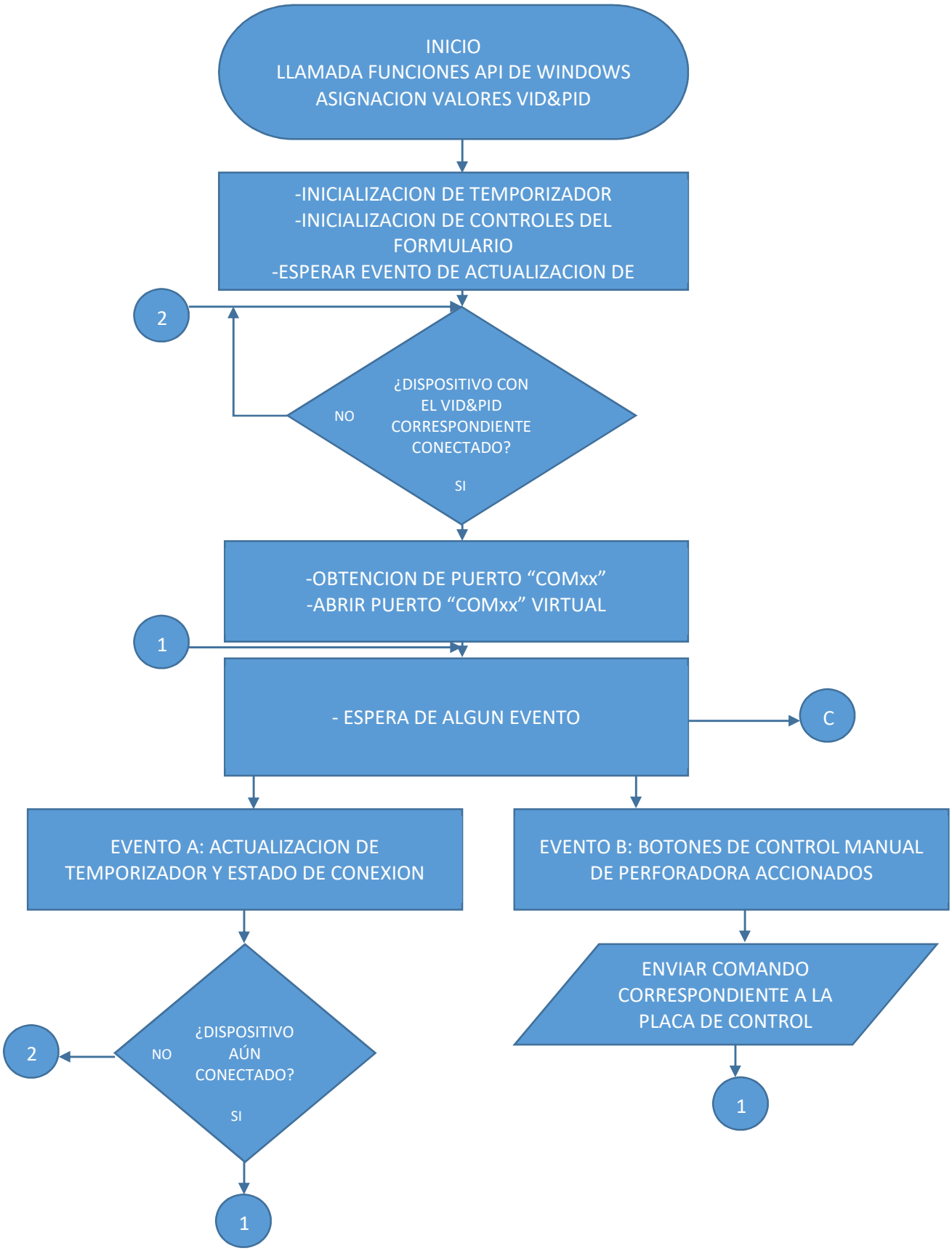

**FIG. Nº 26** *Etiqueta que indica el estado de conexión entre la PC y la perforadora.*

Dentro de este método se encuentra una función muy importante: USB_CheckIfPresentAndGetCOMString(). Esta funcion verifica si se encuentra un dispositivo USB CDC (CDC: Dispositivo de la clase de comunicación) actualmente conectado y que coincida con el PID e VID correspondiente (ver descriptor de dispositivo). Si hay coincidencia, devuelve el nombre del puerto COM virtual correspendiente.
El proceso de deteccion de la conexión de la placa control al puerto USB es el siguiente:
Se crea una cadena de caracteres o string llamada "HardwareID" con el VID y PID correspondiente al dispositivo que se quiere detectar. En este caso quedaría: “Vid_04d8&Pid_000a” y servirá luego para comparación. El siguiente paso es crear una lista de todos los dispositivos USB que esten conectados. Luego se recorre cada una de ellos y se extrae el Hardware ID y se compara con el del paso 1. Si hay coincidencia, se extrae el “FriendlyName”. El “FriendlyName” es una cadena de caracteres que aparece en una de las entradas en el Administrador de Dispositivos. En él se encuentra el numero de puerto COM. Se extra el string con el numero del puerto COM, que posee el siguiente formato “COMXX”, donde “XX” es el numero que le asigno el Sitema Operativo al puerto COM Virtual. Dicho string es el que devuelve la función y con conocimiento del puerto COM, ahora se lo puede abrir para comenzar la comunicación.

Lo anterior es un resumen de cómo funciona el algoritmo anterior. Cabe destacar que es mucho más complejo, ya que utiliza la información del Administrador de Dispositivos y del Registro del Sistema o *System Registry*. Por ejemplo, la información que necesita extraerse como el “FriendlyName” o “HardwareID” se encuentran el el Registro de Hardware.
Ademas se deben utilizar funciones de la API de windows, que estan escritas en C++. En el Anexo IV se deja el codigo completo.

## Diagrama de flujo de la aplicación

A continuación se muestra el diagrama de flujo de la aplicación explicada anteriormente.


INICIO
LLAMADA FUNCIONES API DE WINDOWS
ASIGNACION VALORES VID&PID
-INICIALIZACION DE TEMPORIZADOR
-INICIALIZACION DE CONTROLES DEL FORMULARIO
-ESPERAR EVENTO DE ACTUALIZACION DE
2
¿DISPOSITIVO CON EL VID&PID CORRESPONDIENTE CONECTADO?
NO
SI
-OBTENCION DE PUERTO “COMxx”
-ABRIR PUERTO “COMxx” VIRTUAL
1
- ESPERA DE ALGUN EVENTO
C
EVENTO A: ACTUALIZACION DE TEMPORIZADOR Y ESTADO DE CONEXION
EVENTO B: BOTONES DE CONTROL MANUAL DE PERFORADORA ACCIONADOS
¿DISPOSITIVO AÚN CONECTADO?
NO
SI
2
1
ENVIAR COMANDO CORRESPONDIENTE A LA PLACA DE CONTROL
1

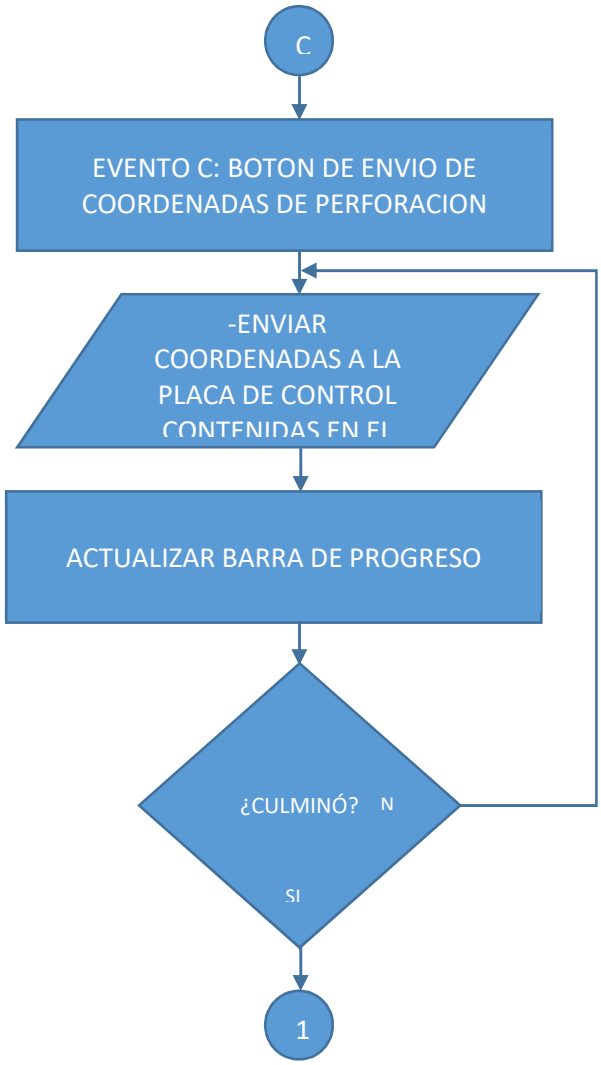



**FIG. Nº 27** *Diagrama de flujo del código de la aplicación*

# Evaluación Final del Sistema

## Preparación del prototipo con materiales finales

Para el armado del prototipo de perforadora se utilizaran los elementos y materiales mencionados en las tablas 1, 2 , 3 y 4. Se muestran imágenes de algunos de ellos.

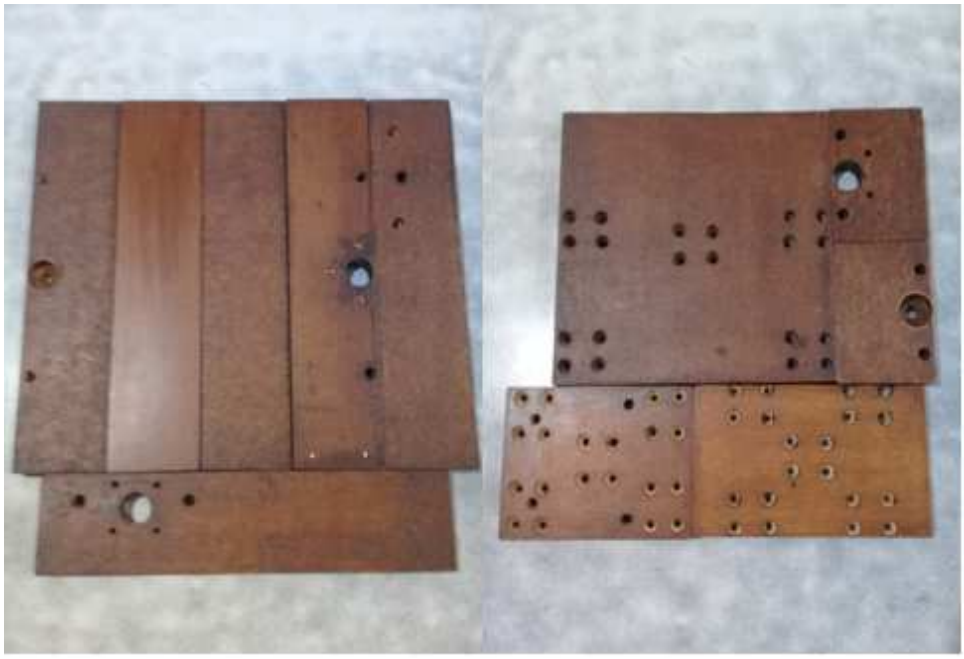


**FIG. Nº 28** *Piezas de MDF para la construcción de la perforadora.*

**FIG. Nº 29** *Imágenes de algunos de los materiales para el ensamble.*

## Montaje y ensayo real del prototipo

Con los materiales ya adquiridos se procede a realizar la perforadora, la placa y ensamblaje. Los detalles para la construcción de la perforadora se encuentran en el Anexo II.

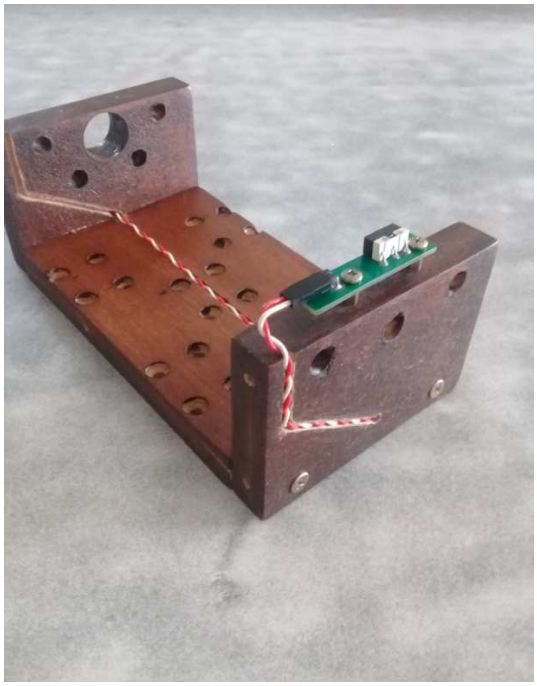


**FIG. Nº 30** *Ensamble del carro eje Y*

**FIG. Nº 31** *Motor y estructura del eje X.*

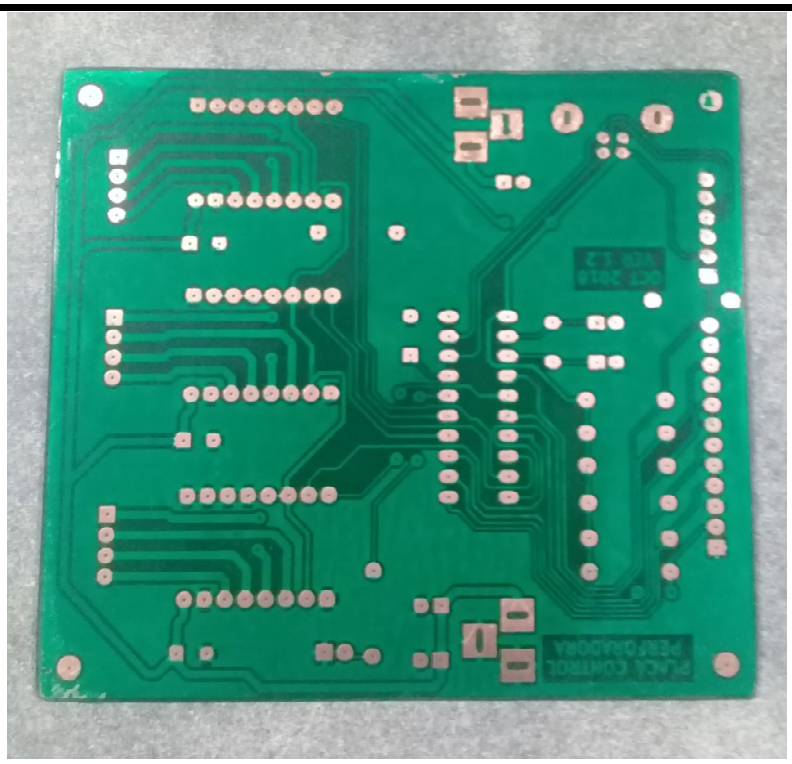


**FIG. Nº 32** *Fotografía de la PCB realizada (lado cobre).*

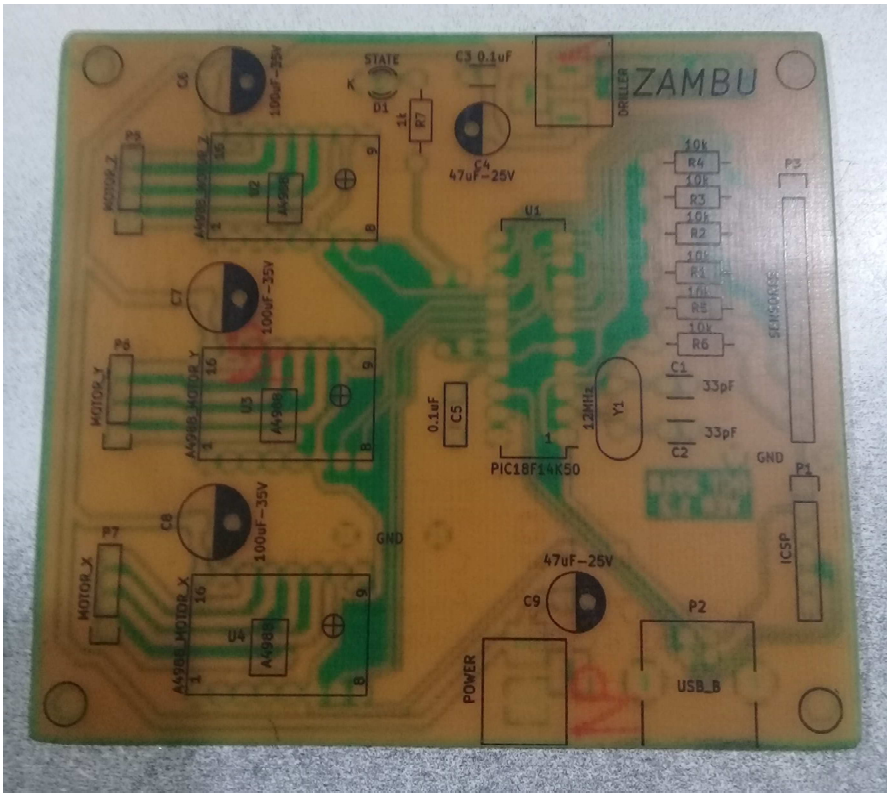



**FIG. Nº 33** *Fotografía de la PCB terminada (lado componentes).*

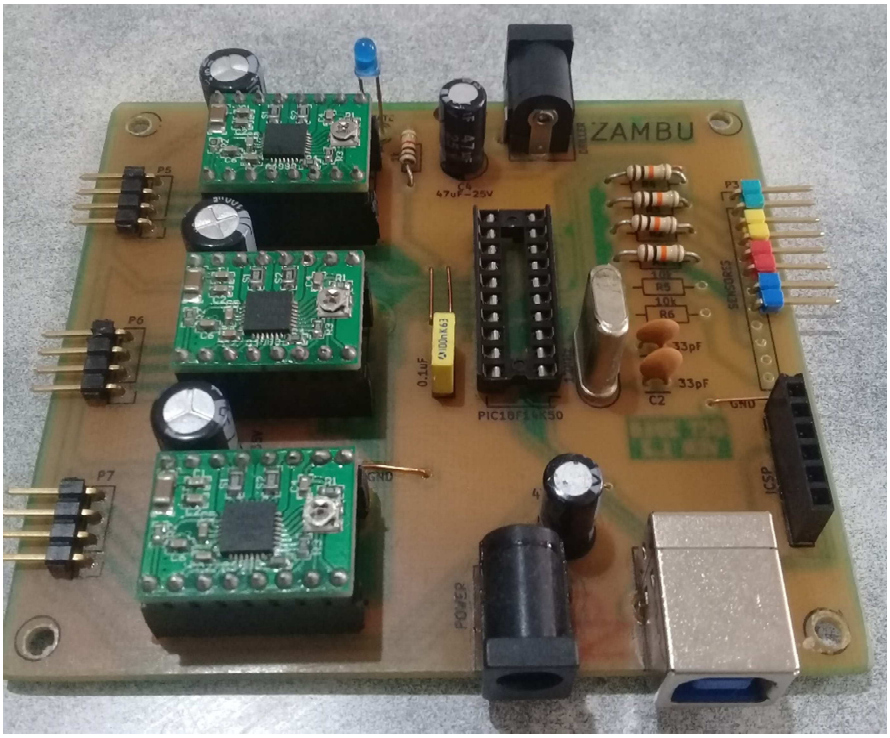



**FIG. Nº 34** *Imagen de la placa con sus componentes montados.*

## Análisis del sistema en campo

Para poner a prueba la perforadora se diseñó una placa del tipo pre-perforada (como las que se consiguen comercialmente). La placa diseñada en Kicad se muestra a continuación.

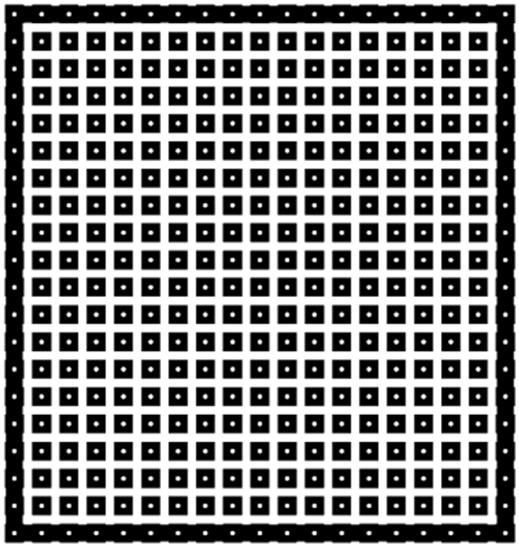


**FIG. Nº 35** *Diseño de una placa para prueba de la perforadora.*

El tamaño de la placa es de aproximadamente de 5cm x 5cm y tiene un total de 380 agujeros. Se utilizó una broca de 0.75 mm de diámetro. Los resultados obtenidos se muestran a continuación:

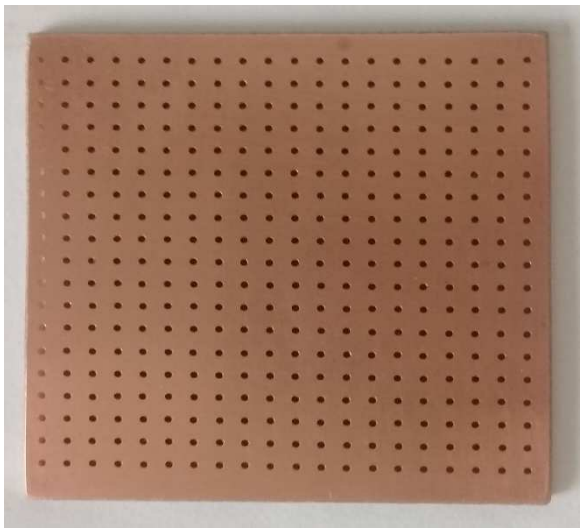


**FIG. Nº 36** *fotografía PCB perforada (lado cobre).*

Con la placa ya agujereada con la perforadora automática, se procedió a transferir la imagen del circuito utilizando el método de insolado, obteniendo el siguiente resultado:

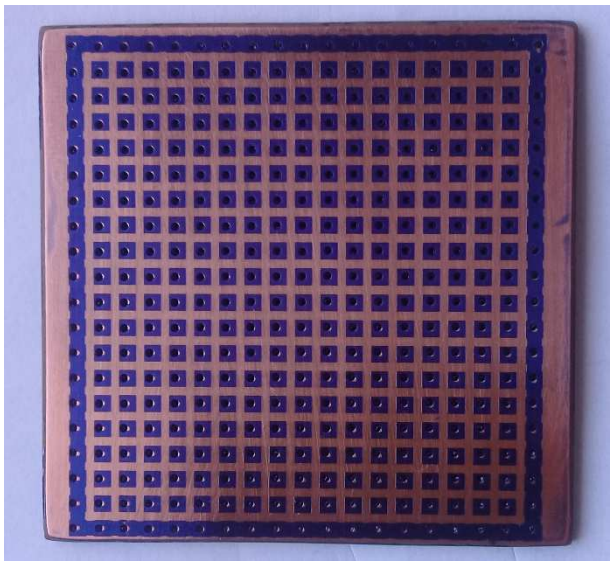


**FIG. Nº 37** *PCB con el proceso de revelado ya realizado.*

Luego se remueve el cobre y al final se quita el restante de película fotosensible, logrando el resultado que se muestra a continuación.

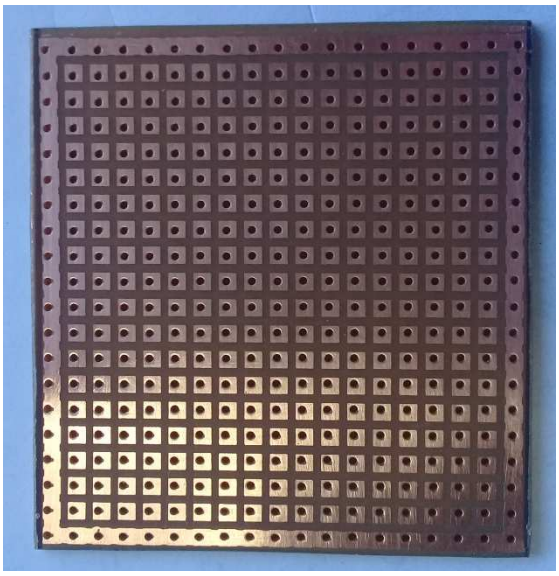


***FIG. Nº 38*** *Placa de prueba terminada.*

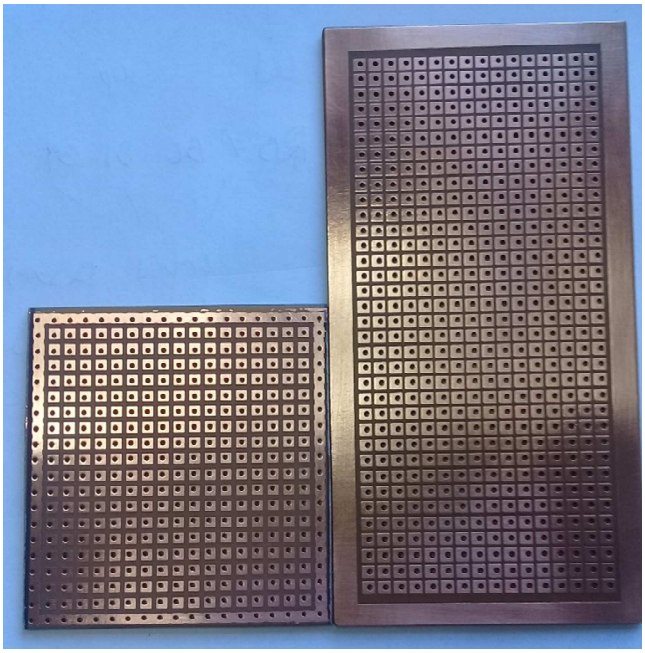


**FIG. Nº 39** *Comparación entre la PCB propia (izquierda) y una comercial (derecha).*

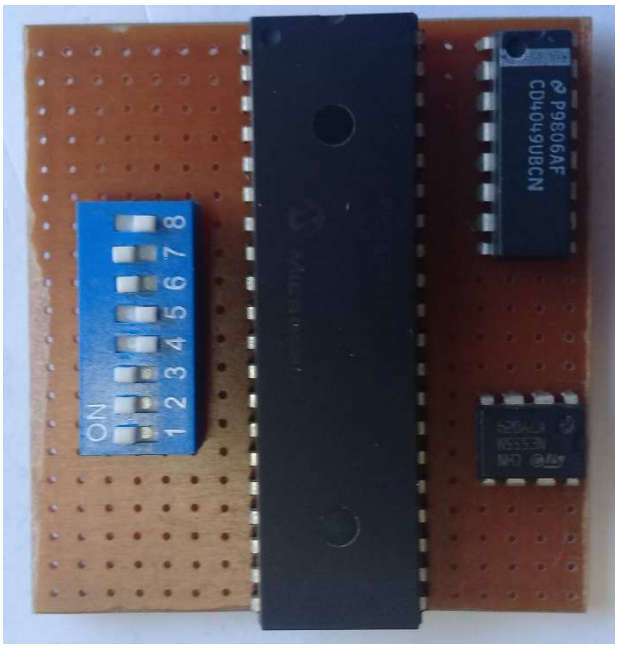



**FIG. Nº 40** *Montaje de algunos componentes sobre la placa de prueba anterior.*

## Presentación final del prototipo

Las imágenes del prototipo completamente acabado se muestran a continuación.

**FIG. Nº 41** *Prototipo terminado.*

**FIG. Nº 42** *Parte posterior, donde se apreciala placa de control y la fuente de alimentación.*

## Prestaciones de la perforadora

Se puede estimar la exactitud de los ejes 'X' e 'Y' si se tienen en cuenta los errores de los motores paso a paso en grados por paso. Las varillas roscadas no se tienen en consideración para el cálculo de exactitud del proceso de perforado debido a que se elimina el juego mecánico que poseen con tuercas anti-holgura (anti-backlash). Siguiendo con el análisis, de acuerdo a las especificaciones del fabricante, los motores paso a paso utilizados tienen una exactitud de 1.8º ± 5% [11]. De acuerdo a la ecuación 4, el avance por paso es de 0.0025 mm. Si se traslada el error producido por los motores se obtiene que el avance por paso es de 0.0025 mm ± 5%, o lo que es lo mismo, el desplazamiento estará comprendido entre 0.002375 mm y 0.002625 mm. Se observa que la tolerancia es de 0.25 milésimas de milímetro o 250 µm, un valor admisible.

No obstante, se puede lograr mayor exactitud cambiando las varillas roscadas de 8 mm por vuelta por varillas de 4 mm, 2 mm o incluso 1 mm por vuelta. De este modo, el desplazamiento por cada paso del motor se dividiría por dos, por cuatro o por ocho, respectivamente. Y, además, si se colocase un motor paso a paso de 0.9º/paso, se lograría dividir por dos nuevamente. Cabe tener en cuenta que al hacer lo anterior aumenta el tiempo de perforado y se debe tener en cuenta si se quiere rapidez en el proceso.

El tiempo que duro el perforado de la placa de prueba fue de unos 11 minutos aproximadamente, desde que se envió las coordenadas a la placa de control hasta que finalizó y se ubicó en la posición de inicio. La placa contaba con 380 agujeros en total, por lo que le llevó alrededor de 1.7 segundos por cada uno. Está claro que el tiempo dependerá del diseño de la placa además de la cantidad de agujeros, pero aun así se logra un muy buen tiempo. Si los pulsos del microcontrolador que activan los pasos de los motores se acortan en tiempo, se puede disminuir aún más el tiempo en caso de que sea necesario, siempre y cuando el ancho de los mismos sea mayor que el mínimo especificado por el driver, así no se pierden pasos y se comenten errores en los desplazamientos de los ejes.

# Conclusiones

Luego de poner en funcionamiento el prototipo de perforadora y verla en ejecución se pueden elaborar las siguientes conclusiones.
Uno de los requisitos de diseño para este proyecto era que el proceso de agujereado debía ser lo más preciso posible gracias a una resolución lograda de unos 250 µm en los ejes X e Y. Si se compara la placa de prueba producida aquí con una comercial se observa que los resultados son más que aceptables, con acabados muy similares.
Con respecto al control, la perforadora resultó muy fácil de operar: una vez conectada la fuente de alimentación y el cable USB al PC, la aplicación se encarga de abrir el puerto de comunicación correspondiente. El usuario no debe buscar ni asignar los puertos, sino que queda a cargo del programa facilitando y agilizando la comunicación USB. Al establecer la comunicación con el PC, los carros de los tres ejes se ubican en la posición de origen y lo que resta es colocar manualmente el mini torno en el lugar deseado y enviar el código G desde la aplicación a la placa de control.

Como conclusión final se puede afirmar que se logró cumplir con los objetivos planteados al inicio del proyecto.

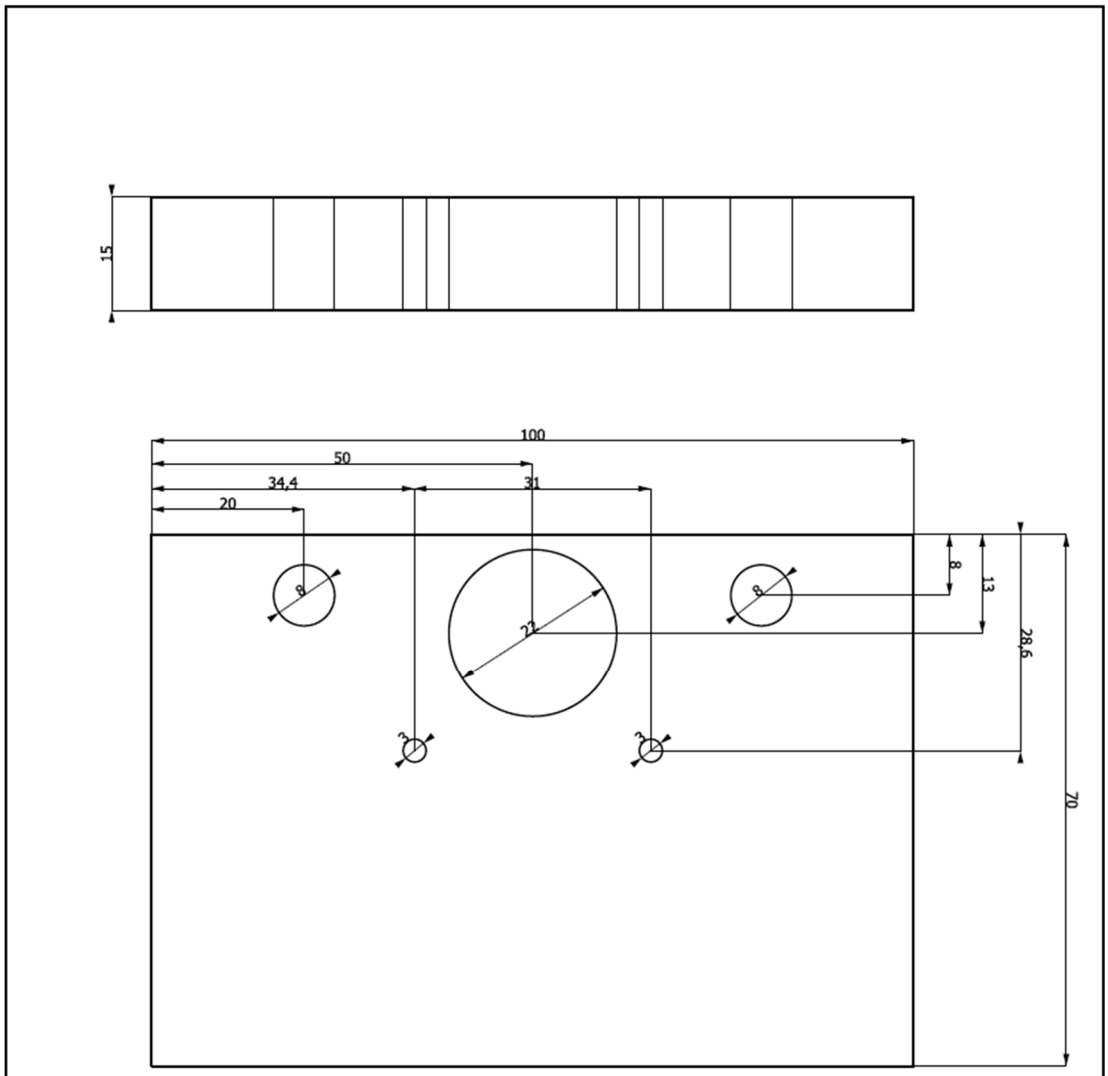
.

# Bibliografía

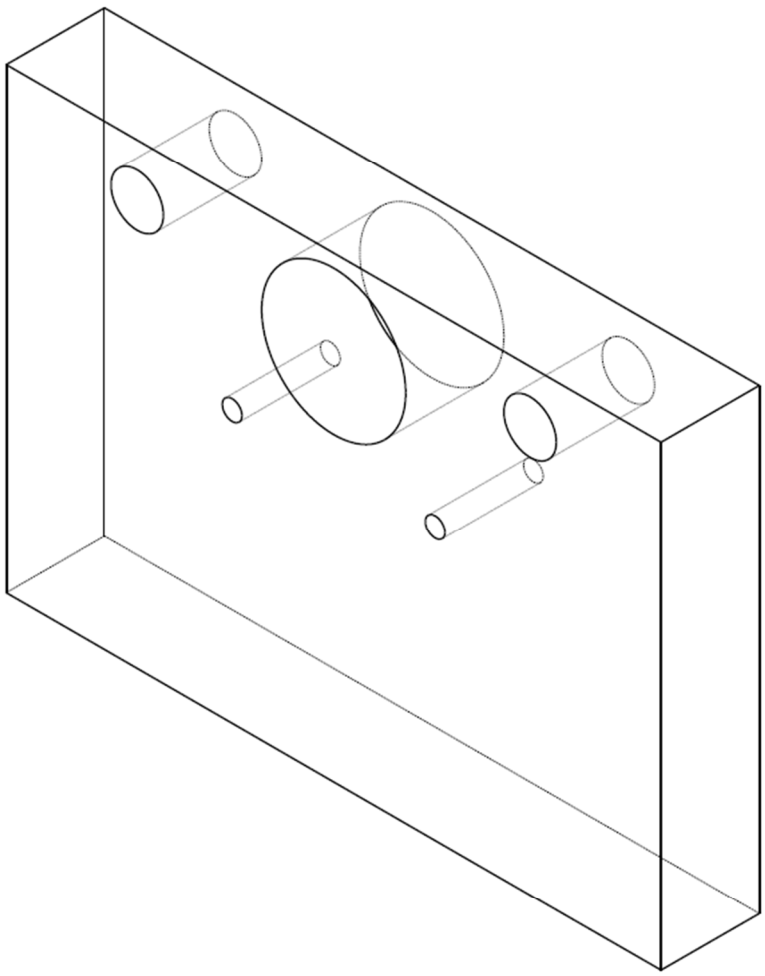
[1] Jan Axelson, *USB COMPLETE 5th Edition*; 5th Edition; Lakeview Research LLC, 2017.
[2] Jan Axelson, *SERIAL PORT COMPLETE 2nd Edition*; 2nd Edition; Lakeview Research LLC, 2015.
[3] Ceballos Francisco Javier, *Enciclopedia de Microsoft® Visual C#™*; 4ª Edición; Alfaomega-Ra-Ma, 2013.
[4] Orbegozo Arana Borja, Desarrollo de aplicaciones C# con Microsoft Visual Studio .NET Curso Práctico; 1ª Edición; Alfaomega,2015.
[5] John Allwork, C# 2008 and .NET Programming for Electronic Engineers; 1st Edition; elektor, 2009;
[6] Microchip, PIC18(L)F14K50 20-Pin USB Flash Microcontrollers with XLP Technology, Datasheet; (DS40001350F, 2015).
http://ww1.microchip.com/downloads/en/devicedoc/40001350f.pdf
[7] Allegro MicroSystems, A4988 DMOS Microstepping Driver with Translator And Overcurrent Protection, Datasheet; (4988-DS, REV. 5, 2014).
https://www.pololu.com/file/0J450/a4988_DMOS_microstepping_driver_with_translator.pdf
[8] XMOS®, Application Note AN00124 USB CDC Class as Virtual Serial Port;(XM006845, 2016)
https://www.xmos.com/download/AN00124:-USB-CDC-Class-as-Virtual-Serial-Port(2.0.2rc1).pdf
[9] Pololu Robotics & Electronics, Catalogue.
https://www.pololu.com/product/1182 (septiembre 2019).
[10] MarhGuides.com, How to control a stepper motor with A4988 driver and Arduino, Benne de Bakker (septiembre 2019).
https://www.makerguides.com/a4988-stepper-motor-driver-arduino-tutorial/
[11] SYC Semiconductores y components, Datasheet;
http://www.sycelectronica.com.ar/diseno/AR-NEMA17.pdf

# Anexo I

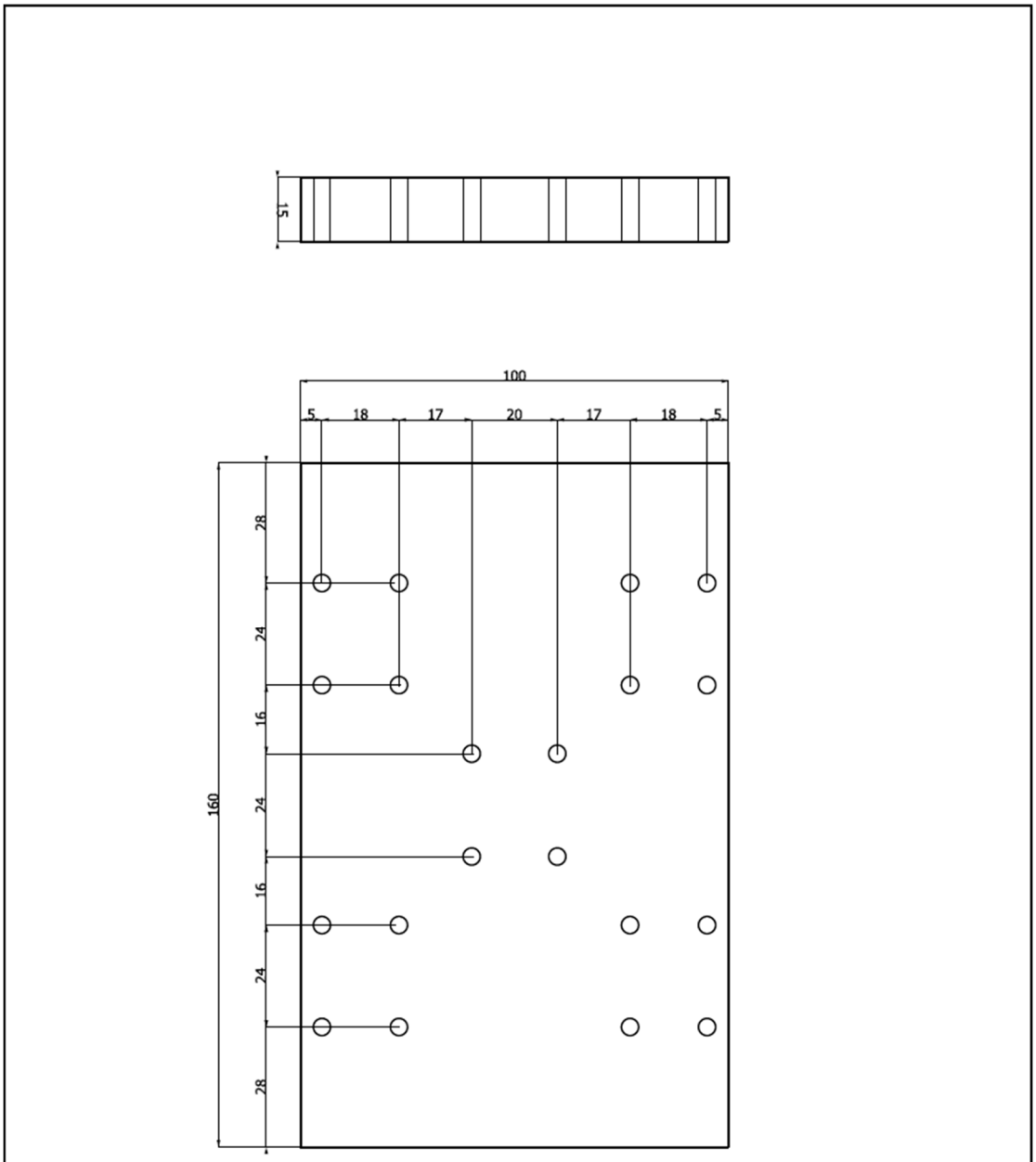
A continuación se muestran los planos de las piezas de madera MDF con sus dimensiones, necesarias para la construcción de la perforadora. Los dibujos fueron realizados utilizando el software Freecad versión 0.17.




| | | | |
|---|---|---|---|
| Created by: BUZZETTI, GONZALO E. | Title: PIEZAS MDF PERFORADORA AUTOMATICA | | |
| Supplementary information: | Size: A4 | Sheet: 1/2 | Scale: 1.25 |
| PIEZA Nº1-COTAS | Part number: 0001 | | |
| | Drawing number: 01 | | |
| Dimensiones en mm | Date: 02/09/19 | | Revision: REV 1.0 |

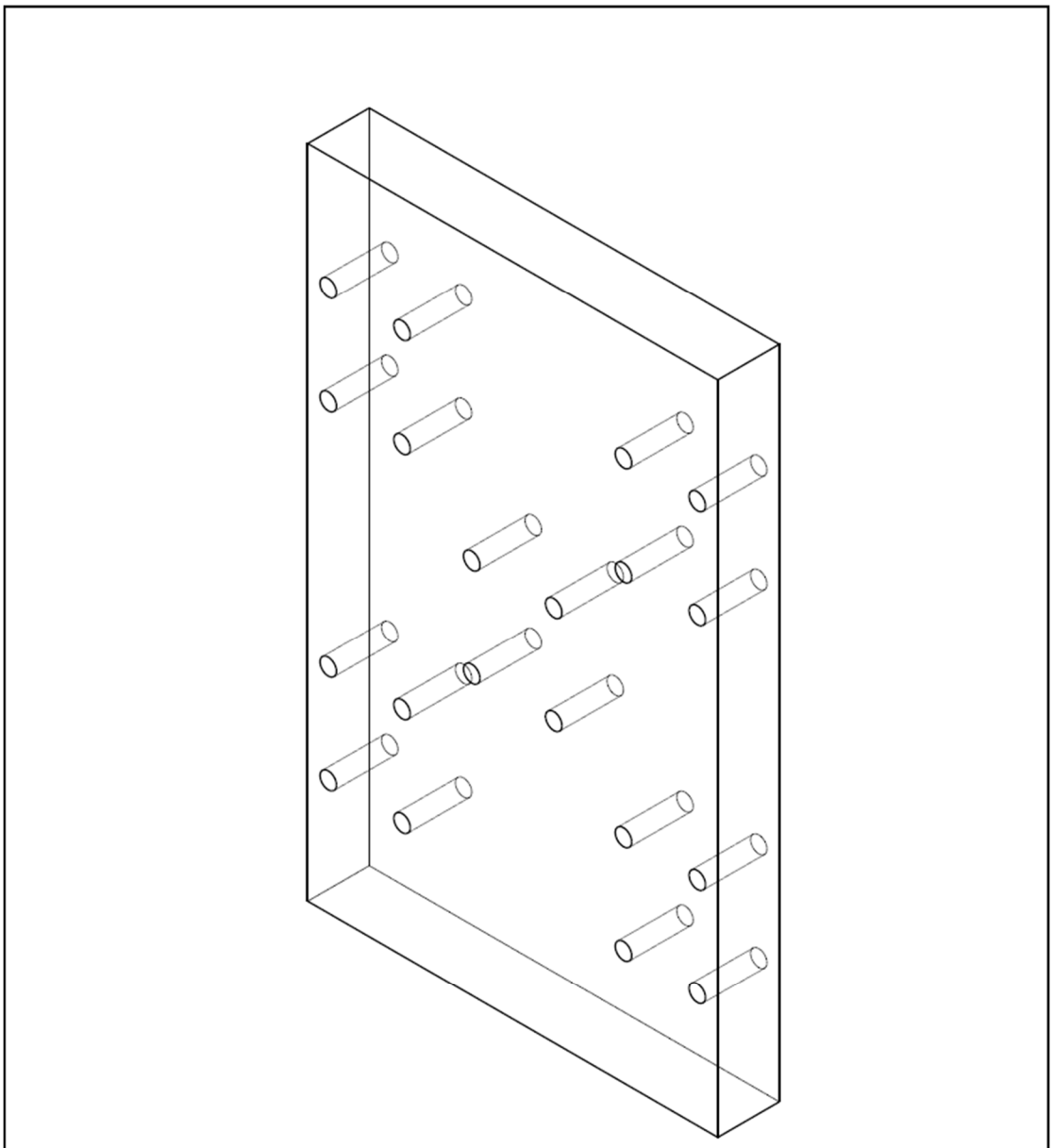


| | | | |
|---|---|---|---|
| Created by: BUZZETTI, GONZALO E. | Title: PIEZAS MDF PERFORADORA AUTOMATICA | | |
| Supplementary information: | Size: A4 | Sheet: 2/2 | Scale: 1.25 |
| PIEZA Nº1-PERSPECTIVA | Part number: 0001 | | |
| | Drawing number: 02 | | |
| Dimensiones en mm | Date: 02/09/19 | | Revision: REV 1.0 |

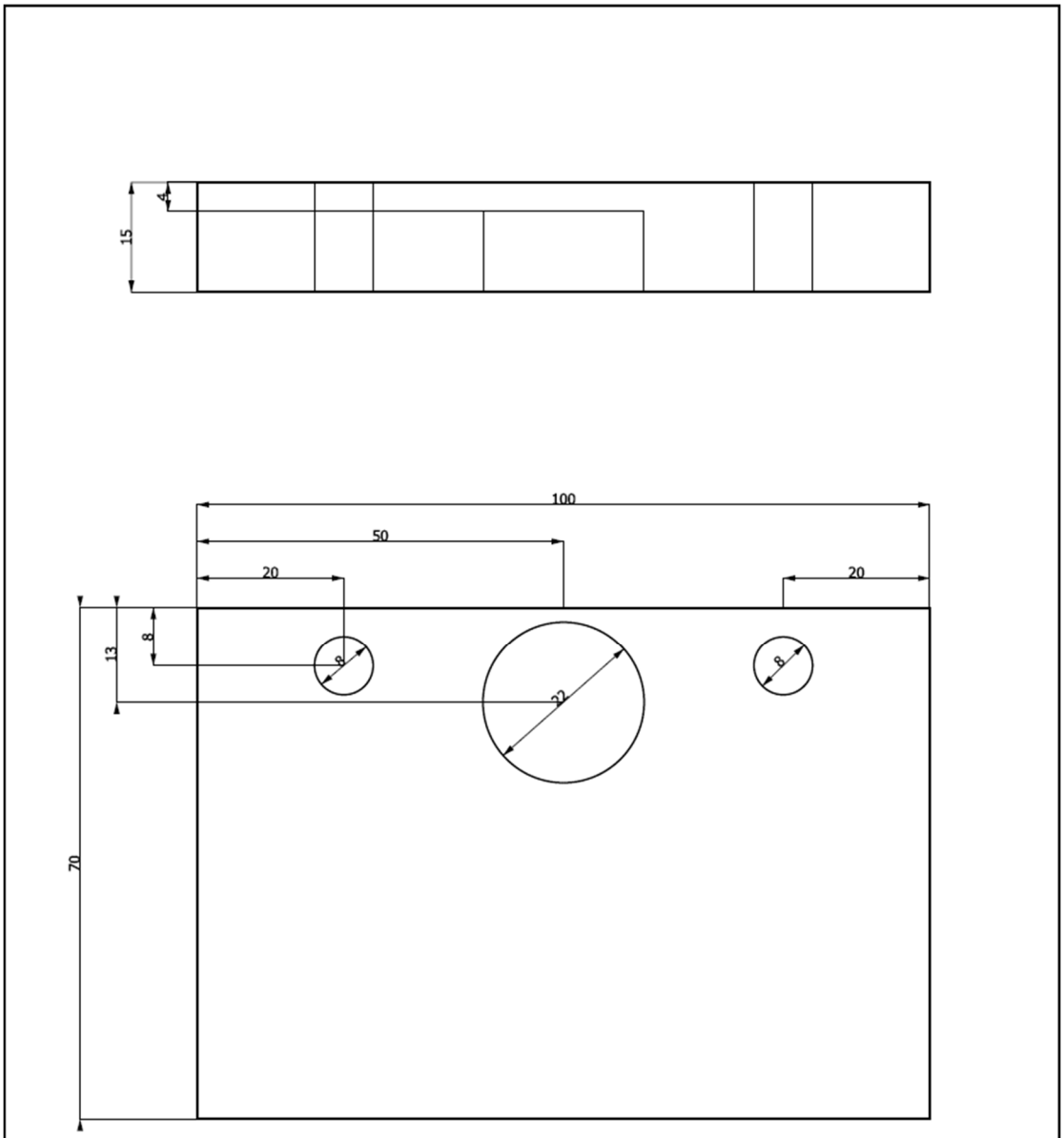



| | |
|---|---|
| Created by: BUZZETTI, GONZALO E. | Title: PIEZAS MDF PERFORADORA AUTOMATICA |

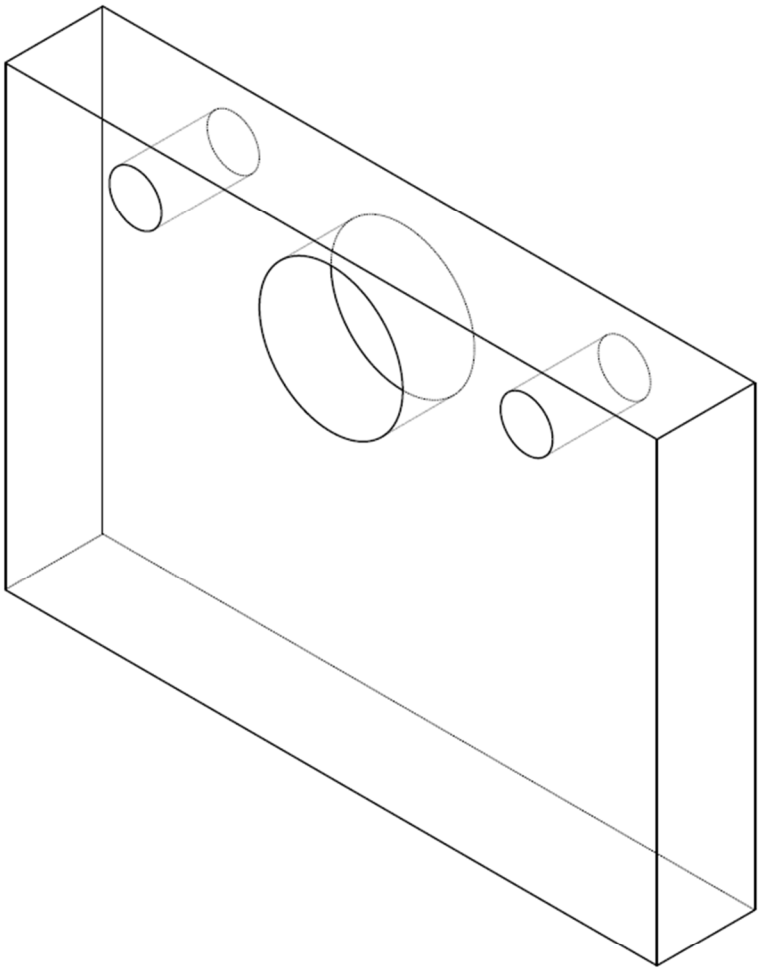
| | | | |
|---|---|---|---|
| Supplementary information: | Size: A4 | Sheet: 1/2 | Scale: 0,75 |
| PIEZA Nº2-COTAS | Part number: 0002 | | |
| | Drawing number: 03 | | |
| Diametros de los agujeros:4mm<br>Dimensiones en mm | Date: 02/09/19 | | Revision: REV 1.0 |



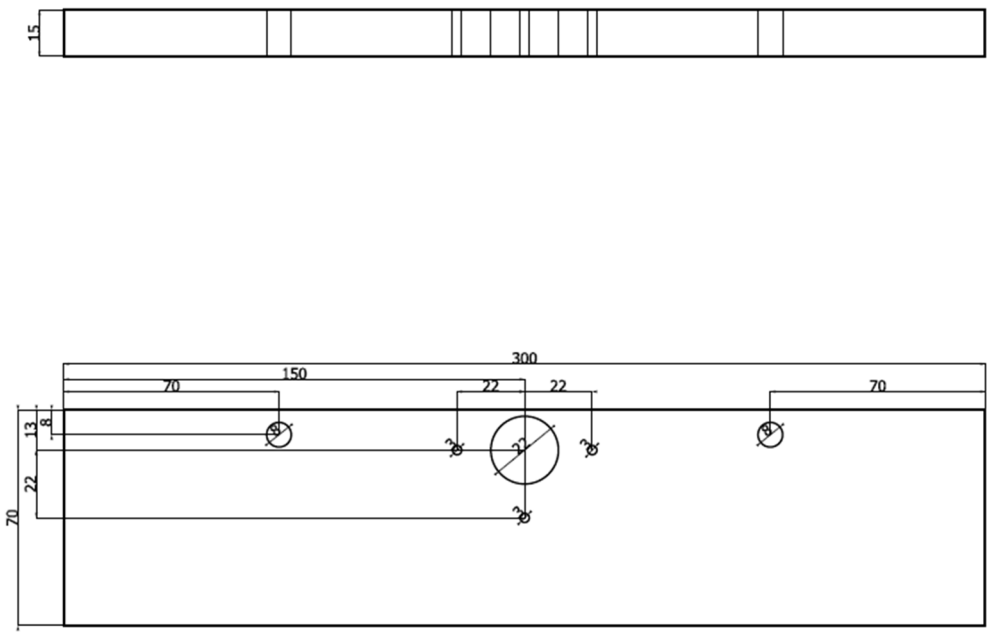
| | | | | |
|---|---|---|---|---|
| Created by: BUZZETTI, GONZALO E. | Title: PIEZAS MDF PERFORADORA AUTOMATICA | | | |
| Supplementary information: | Size: A4 | Sheet: 2/2 | Scale: 0,75 | |
| PIEZA Nº2-PERSPECTIVA | Part number: 0002 | | | |
| | Drawing number: 04 | | | |
| Dimensiones en mm | Date: 02/09/19 | | Revision: REV 1.0 | |



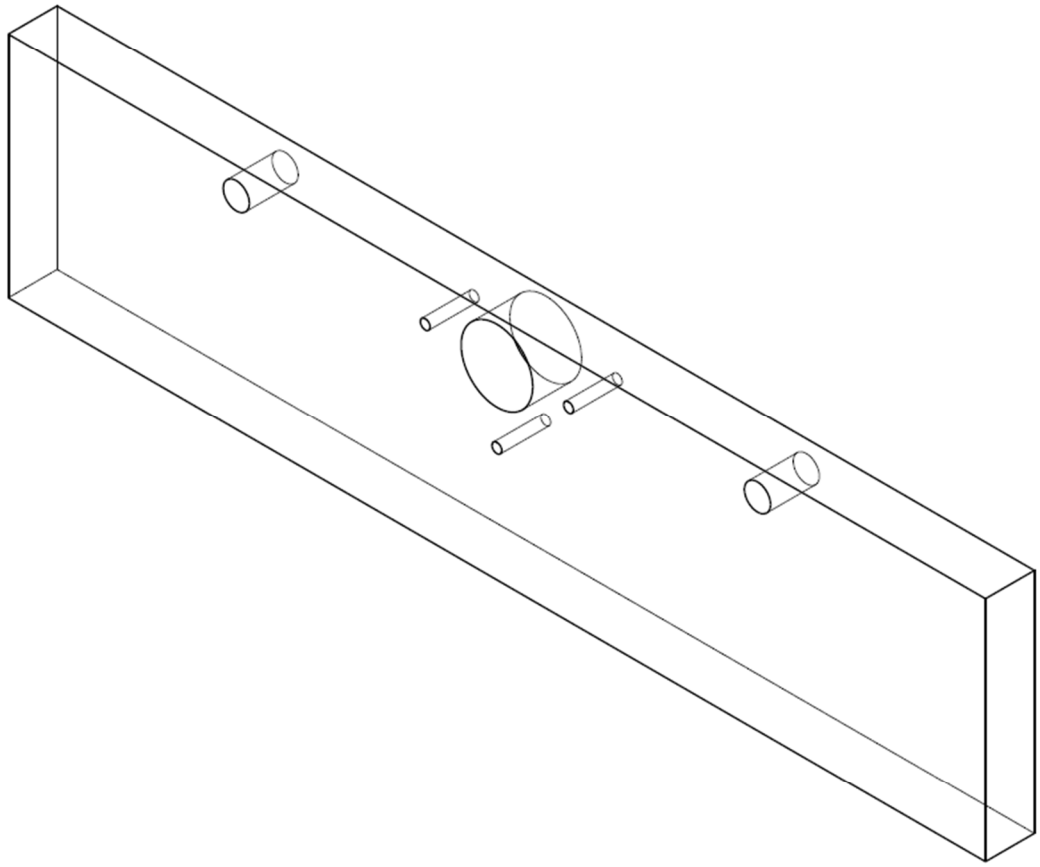

| Created by: BUZZETTI, GONZALO E. | Title: PIEZAS MDF PERFORADORA AUTOMATICA | | |
|---|---|---|---|
| Supplementary information: PIEZA Nº3-COTAS<br>Dimensiones en mm | Size: A4 | Sheet: 1/2 | Scale: 1.25 |
| | Part number: 0003<br>Drawing number: 05 | | |
| | Date: 02/09/19 | | Revision: REV 1.0 |



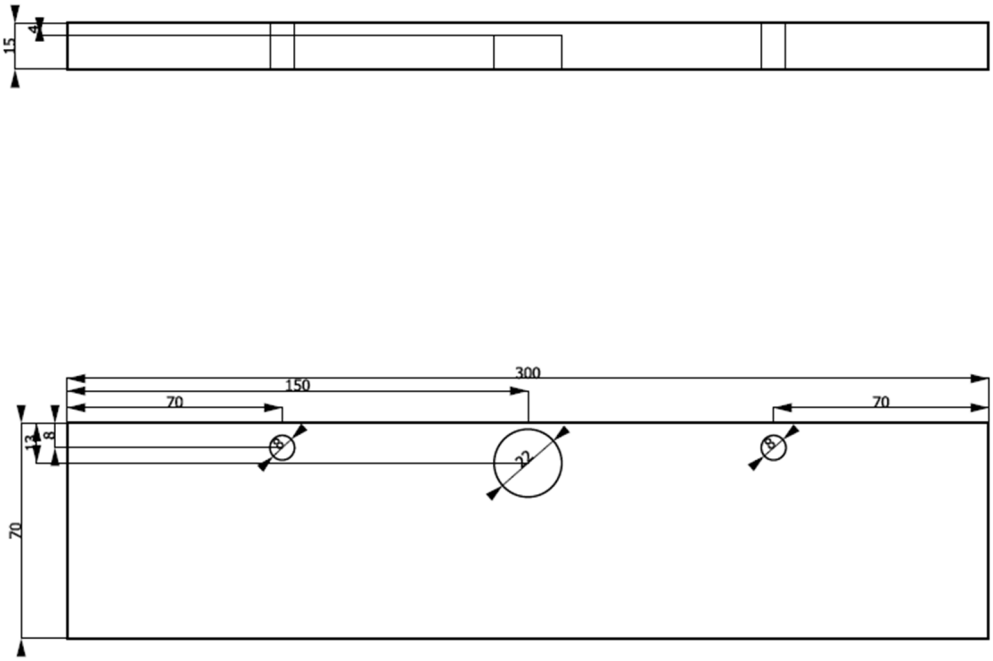
| | | | | |
|---|---|---|---|---|
| Created by: BUZZETTI, GONZALO E. | Title: PIEZAS MDF PERFORADORA AUTOMATICA | | | |
| Supplementary information: | Size: A4 | Sheet: 2/2 | Scale: 1.25 | |
| PIEZA Nº3-PERSPECTIVA | Part number: 0003 | | | |
| | Drawing number: 06 | | | |
| Dimensiones en mm | Date: 02/09/19 | | Revision: REV 1.0 | |



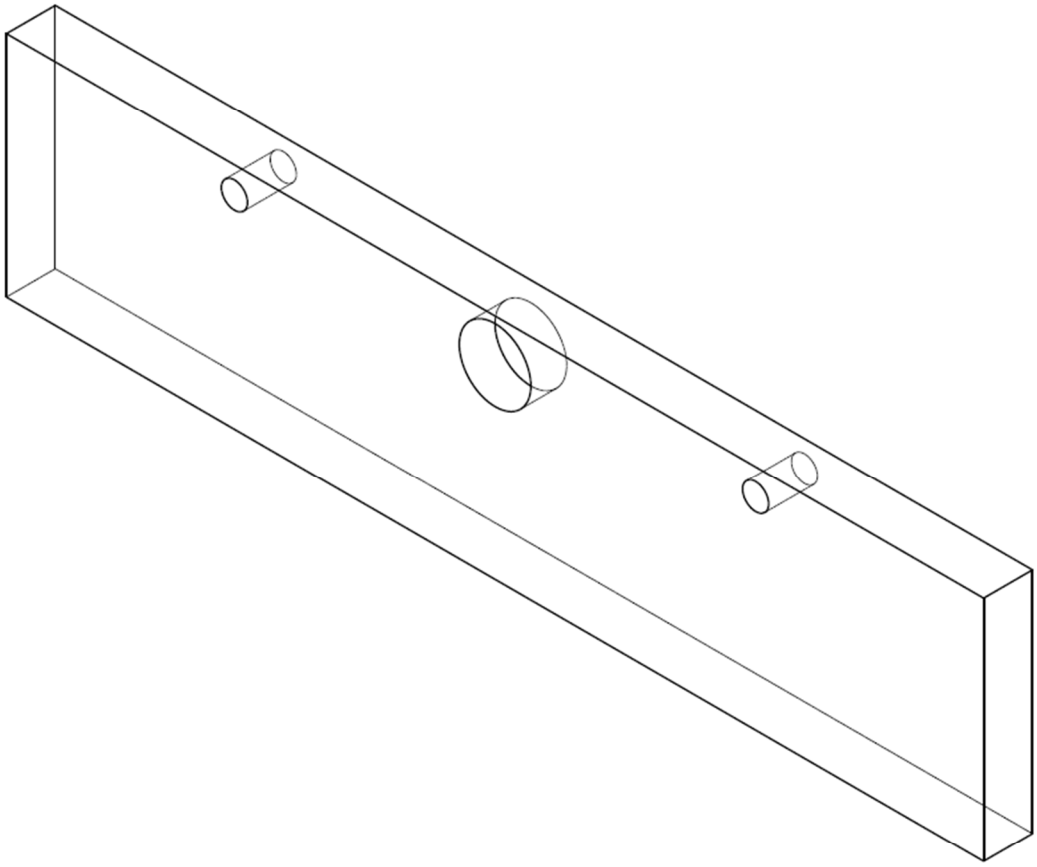

| Created by: BUZZETTI, GONZALO E. | Title: PIEZAS MDF PERFORADORA AUTOMATICA | | |
|---|---|---|---|
| Supplementary information: PIEZA Nº4-COTAS<br>Dimensiones en mm | Size: A4 | Sheet: 1/2 | Scale: 0,5 |
| | Part number: 0004<br>Drawing number: 07 | | |
| | Date: 02/09/19 | | Revision: REV 1.0 |



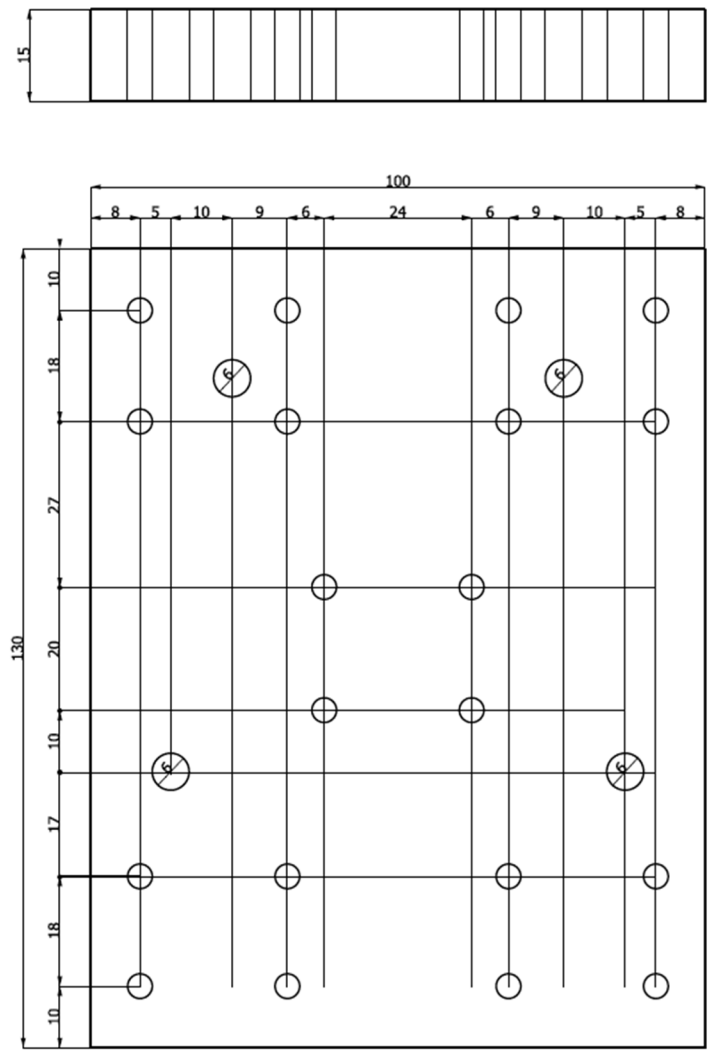
| | | | |
|---|---|---|---|
| Created by: BUZZETTI, GONZALO E. | Title: PIEZAS MDF PERFORADORA AUTOMATICA | | |
| Supplementary information: | Size: A4 | Sheet: 2/2 | Scale: 0,5 |
| PIEZA Nº4-PERSPECTIVA | Part number: 0004 | | |
| | Drawing number: 08 | | |
| Dimensiones en mm | Date: 02/09/19 | | Revision: REV 1.0 |



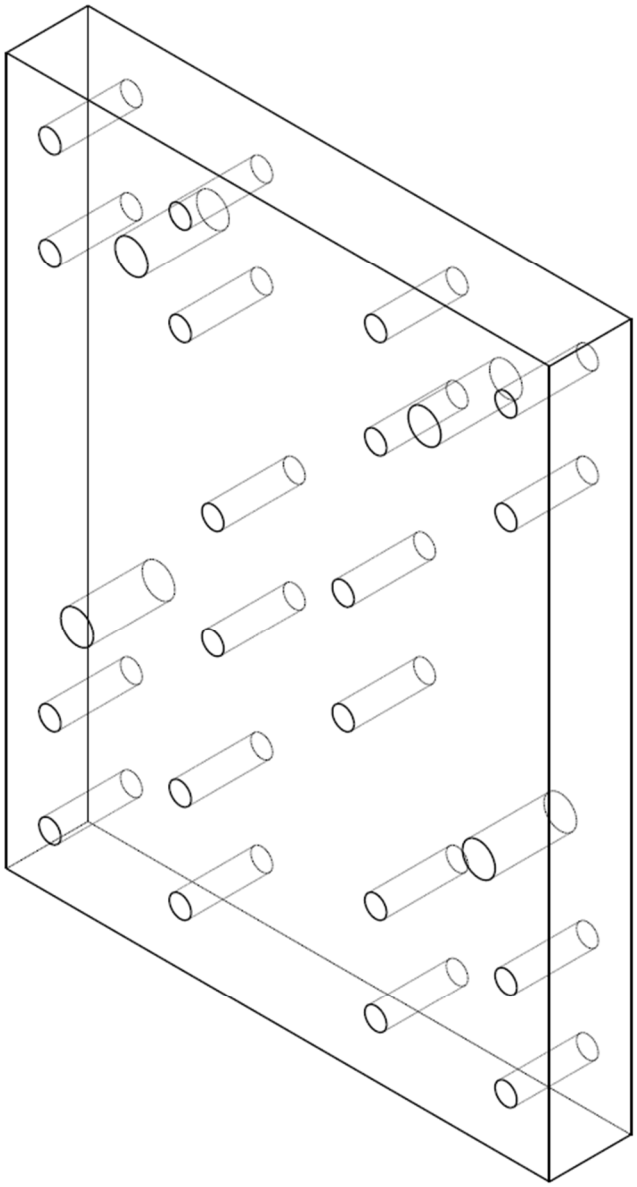

| Created by: BUZZETTI, GONZALO E. | Title: PIEZAS MDF PERFORADORA AUTOMATICA | | |
|---|---|---|---|
| Supplementary information: PIEZA Nº5-COTAS<br>Dimensiones en mm | Size: A4 | Sheet: 1/2 | Scale: 0,5 |
| | Part number: 0005<br>Drawing number: 09 | | |
| | Date: 02/09/19 | | Revision: REV 1.0 |



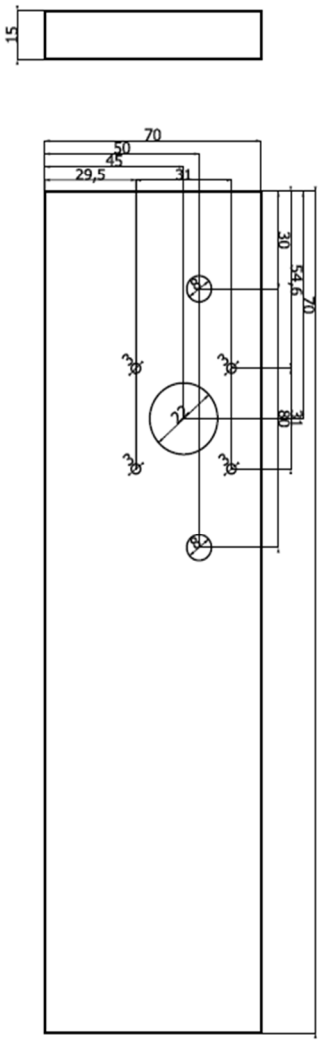
| | | | |
|---|---|---|---|
| Created by: BUZZETTI, GONZALO E. | Title: PIEZAS MDF PERFORADORA AUTOMATICA | | |
| Supplementary information: | Size: A4 | Sheet: 2/2 | Scale: 0,75 |
| PIEZA Nº5-PERSPECTIVA | Part number: 005 | | |
| | Drawing number: 10 | | |
| Dimensiones en mm | Date: 02/09/19 | | Revision: REV 1.0 |



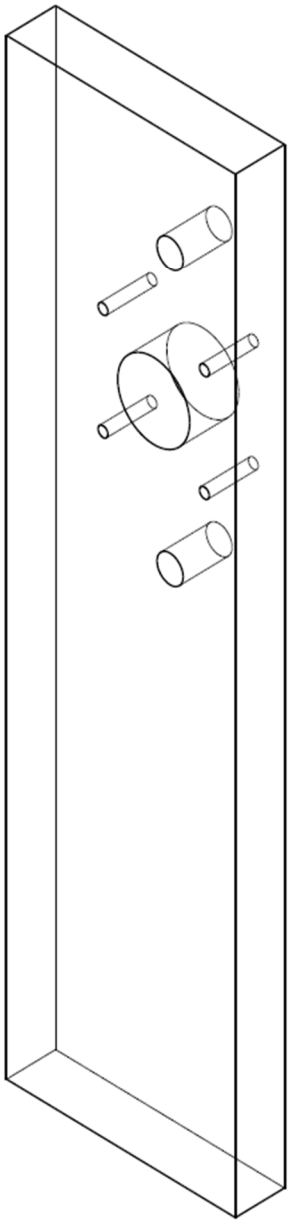

| | | | |
|---|---|---|---|
| Created by: BUZZETTI, GONZALO E. | Title: PIEZAS MDF PERFORADORA AUTOMATICA | | |
| Supplementary information: | Size: A4 | Sheet: 1/2 | Scale: 1 |
| PIEZA Nº6-COTAS<br>Agujeros no indicados: d=4mm<br>Dimensiones en mm | Part number: 0006<br>Drawing number: 11 | | |
| | Date: 02/09/19 | | Revision: REV 1.0 |



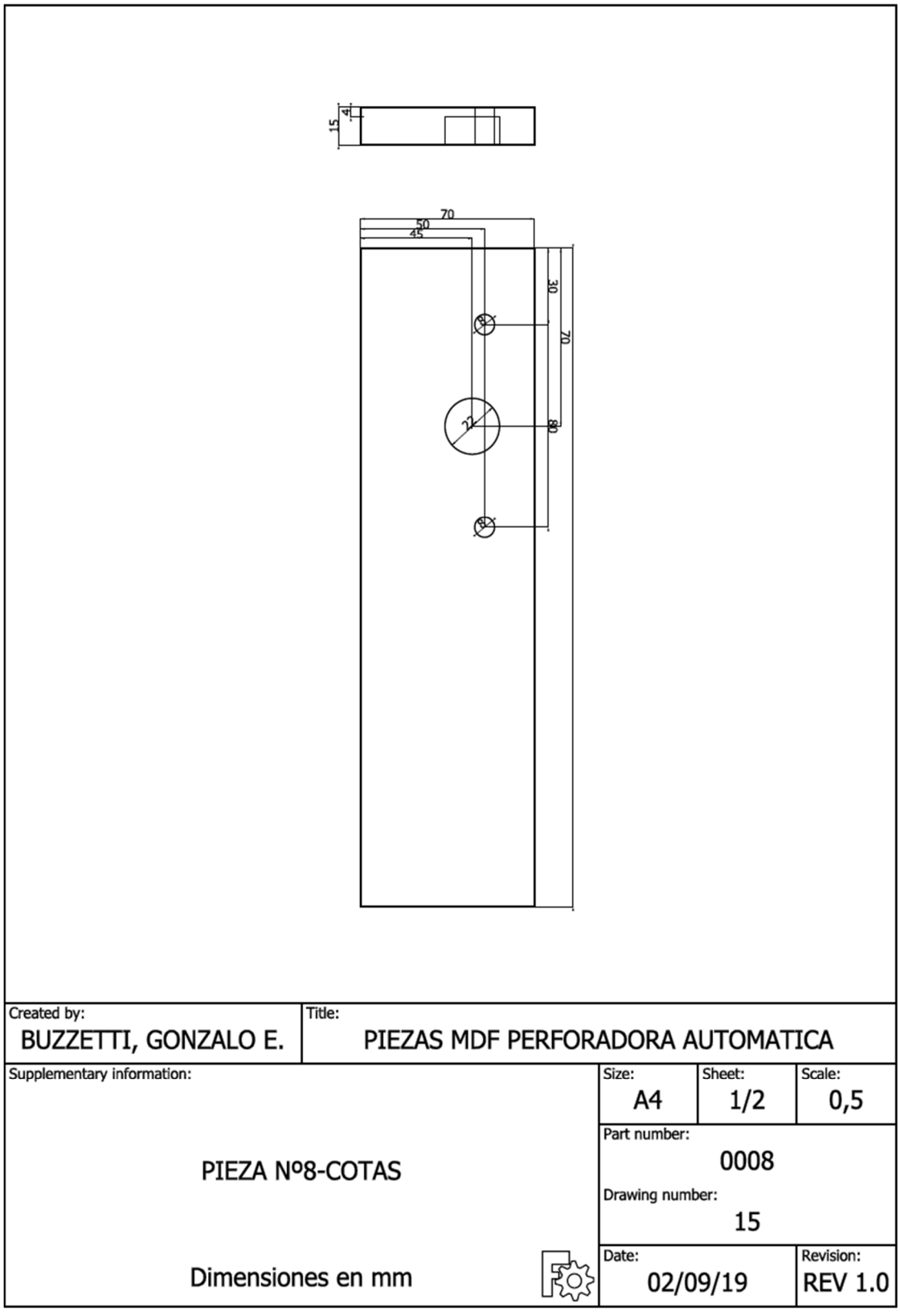
| | |
|---|---|
| Created by: BUZZETTI, GONZALO E. | Title: PIEZAS MDF PERFORADORA AUTOMATICA |
| Supplementary information: | Size: A4 / Sheet: 2/2 / Scale: 1,25 |
| PIEZA Nº6-PERSPECTIVA | Part number: 0006 / Drawing number: 12 |
| Dimensiones en mm | Date: 02/09/19 / Revision: REV 1.0 |



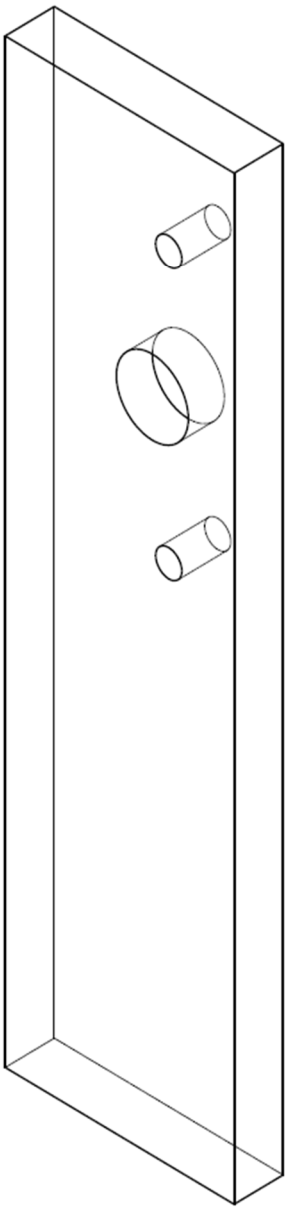

| | | | |
|---|---|---|---|
| Created by: BUZZETTI, GONZALO E. | Title: PIEZAS MDF PERFORADORA AUTOMATICA | | |
| Supplementary information: | Size: A4 | Sheet: 1/2 | Scale: 0,5 |
| PIEZA Nº7-COTAS | Part number: 0007 | | |
| | Drawing number: 13 | | |
| Dimensiones en mm | Date: 02/09/19 | | Revision: REV 1.0 |



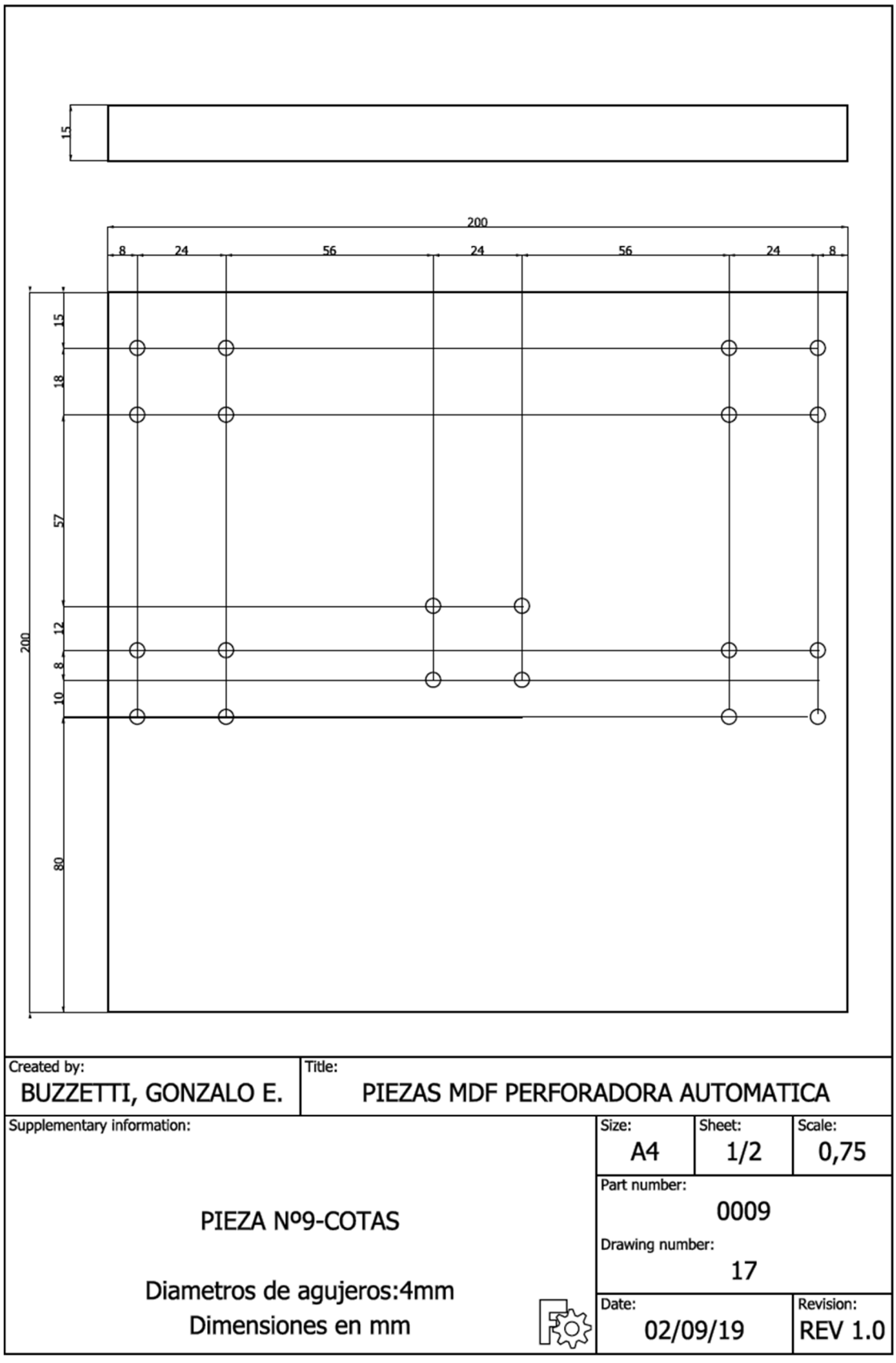
| | | | |
|---|---|---|---|
| Created by: BUZZETTI, GONZALO E. | Title: PIEZAS MDF PERFORADORA AUTOMATICA | | |
| Supplementary information: | Size: A4 | Sheet: 2/2 | Scale: 0,75 |
| PIEZA Nº7-PERSPECTIVA | Part number: 0007 | | |
| | Drawing number: 14 | | |
| Dimensiones en mm | Date: 02/09/19 | | Revision: REV 1.0 |

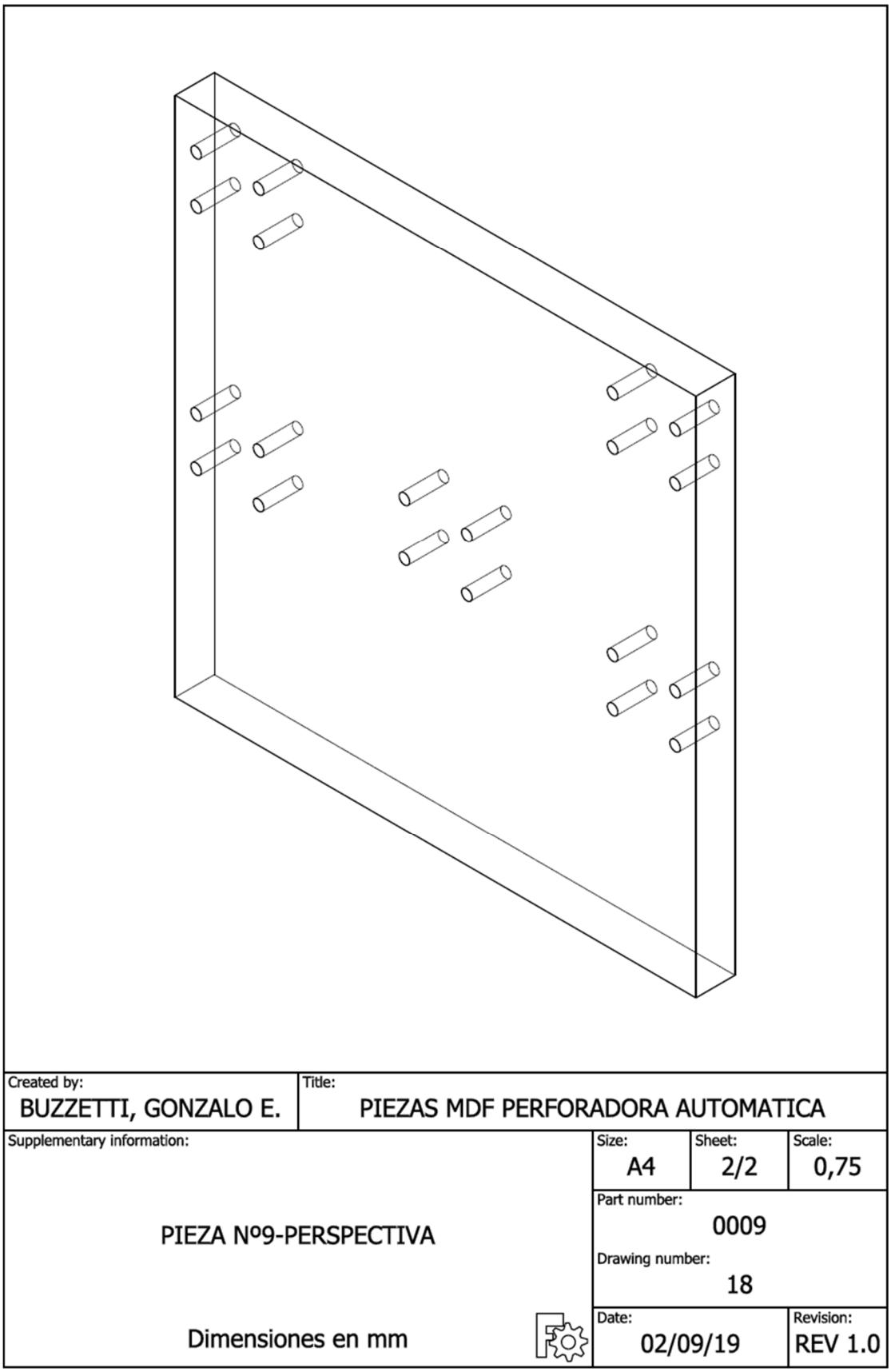


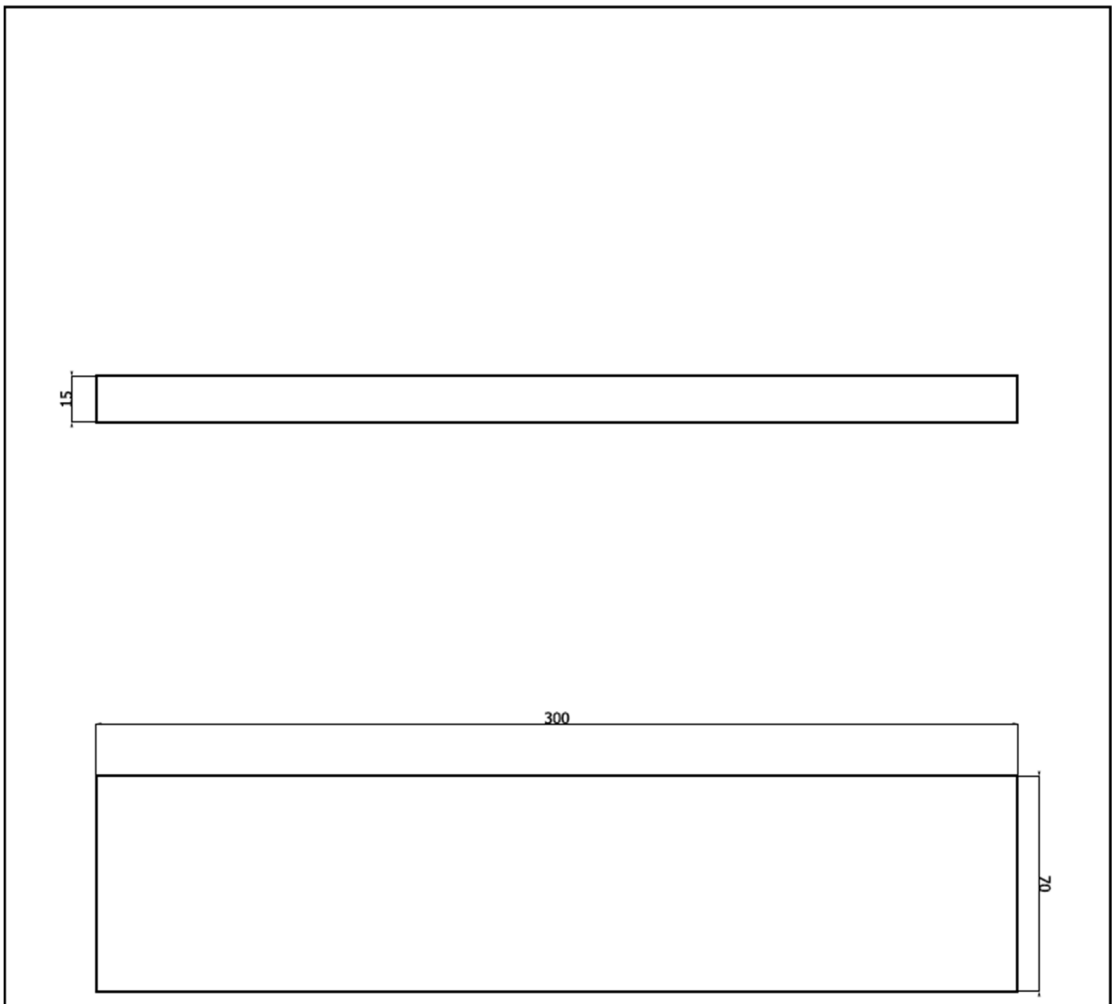

| | | | |
|---|---|---|---|
| Created by: BUZZETTI, GONZALO E. | Title: PIEZAS MDF PERFORADORA AUTOMATICA | | |
| Supplementary information: PIEZA Nº8-COTAS<br>Dimensiones en mm | Size: A4 | Sheet: 1/2 | Scale: 0,5 |
| | Part number: 0008<br>Drawing number: 15 | | |
| | Date: 02/09/19 | | Revision: REV 1.0 |



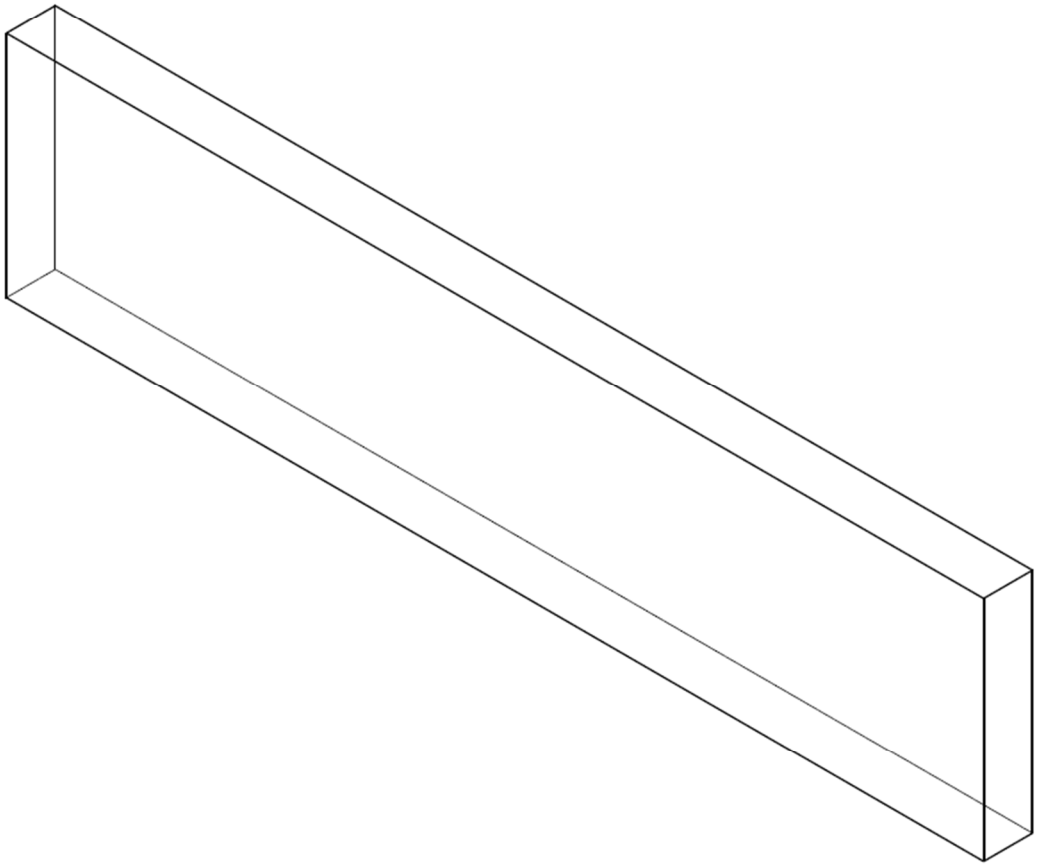
| | | | |
|---|---|---|---|
| Created by: BUZZETTI, GONZALO E. | Title: PIEZAS MDF PERFORADORA AUTOMATICA | | |
| Supplementary information: | Size: A4 | Sheet: 2/2 | Scale: 0,75 |
| PIEZA Nº8-PERSPECTIVA | Part number: 0008 | | |
| | Drawing number: 16 | | |
| Dimensiones en mm | Date: 02/09/19 | | Revision: REV 1.0 |


15
200
8
24
56
24
56
24
8
15
18
57
12
8
10
200
80
Created by:
BUZZETTI, GONZALO E.
Title:
PIEZAS MDF PERFORADORA AUTOMATICA
Supplementary information:
PIEZA Nº9-COTAS
Diametros de agujeros:4mm
Dimensiones en mm
Size:
A4
Sheet:
1/2
Scale:
0,75
Part number:
0009
Drawing number:
17
Date:
02/09/19
Revision:
REV 1.0

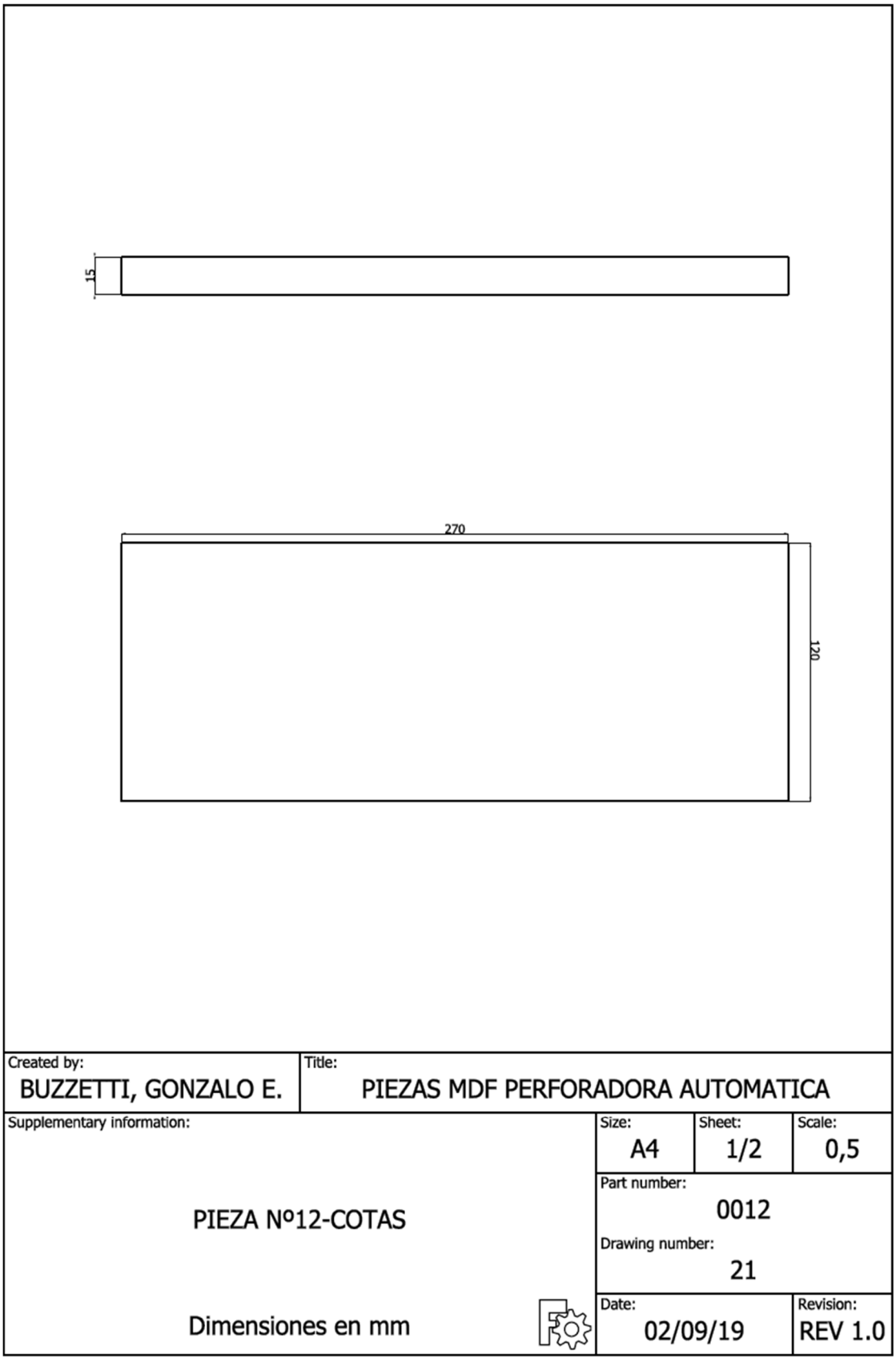

Created by:
BUZZETTI, GONZALO E.
Title:
PIEZAS MDF PERFORADORA AUTOMATICA
Supplementary information:
PIEZA Nº9-PERSPECTIVA
Dimensiones en mm
Size:
A4
Sheet:
2/2
Scale:
0,75
Part number:
0009
Drawing number:
18
Date:
02/09/19
Revision:
REV 1.0

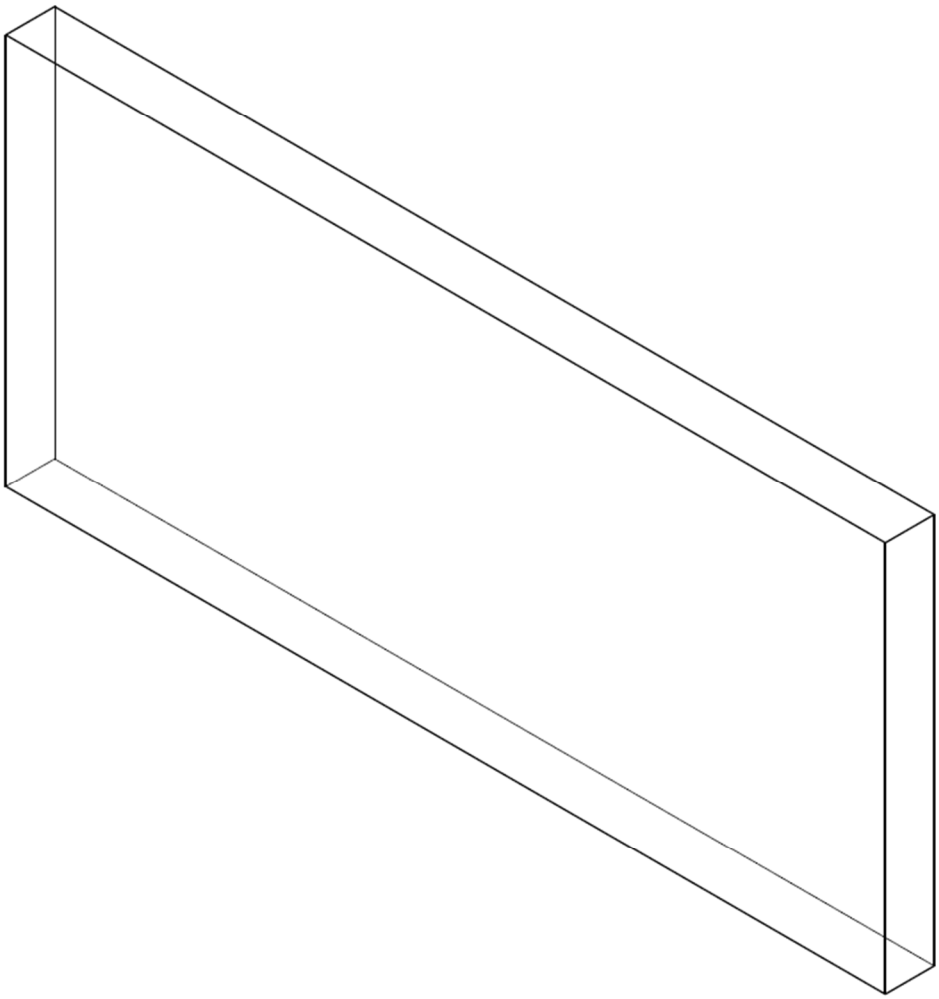



| | | | |
|---|---|---|---|
| Created by: BUZZETTI, GONZALO E. | Title: PIEZAS MDF PERFORADORA AUTOMATICA | | |
| Supplementary information: | Size: A4 | Sheet: 1/2 | Scale: 0,5 |
| PIEZAS Nº10 Y Nº11-COTAS | Part number: 0010 | | |
| | Drawing number: 19 | | |
| Dimensiones en mm | Date: 02/09/19 | | Revision: REV 1.0 |



| | | | |
|---|---|---|---|
| Created by: BUZZETTI, GONZALO E. | Title: PIEZAS MDF PERFORADORA AUTOMATICA | | |
| Supplementary information: | Size: A4 | Sheet: 2/2 | Scale: 0,75 |
| PIEZAS Nº10 Y Nº11-PERSPECTIVA | Part number: 0010 | | |
| | Drawing number: 20 | | |
| Dimensiones en mm | Date: 02/09/19 | | Revision: REV 1.0 |


15
270
120
Created by:
BUZZETTI, GONZALO E.
Title:
PIEZAS MDF PERFORADORA AUTOMATICA
Supplementary information:
PIEZA Nº12-COTAS
Dimensiones en mm
Size:
A4
Sheet:
1/2
Scale:
0,5
Part number:
0012
Drawing number:
21
Date:
02/09/19
Revision:
REV 1.0



| | | | |
|---|---|---|---|
| Created by: BUZZETTI, GONZALO E. | Title: PIEZAS MDF PERFORADORA AUTOMATICA | | |
| Supplementary information: | Size: A4 | Sheet: 2/2 | Scale: 0,75 |
| PIEZA Nº12-PERSPECTIVA | Part number: 0012 | | |
| | Drawing number: 22 | | |
| Dimensiones en mm | Date: 02/09/19 | | Revision: REV 1.0 |

# Anexo II

**Lista de materiales para la construcción de la perforadora.**

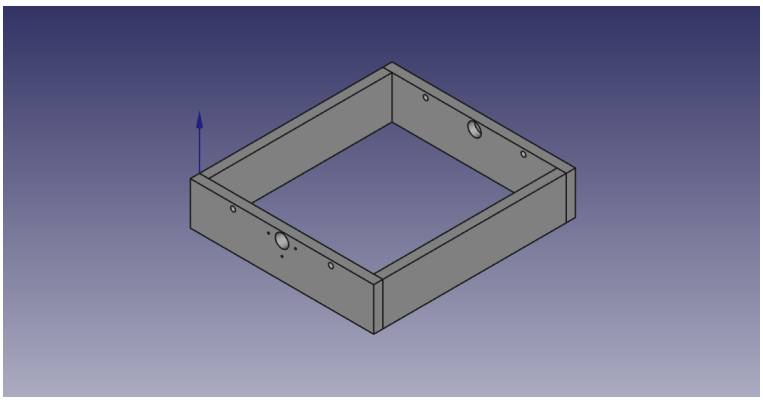
| MATERIALES PARA LA CONSTRUCCION | CANTIDAD |
|---|---|
| **RECORTES MDF (en mm) [1]** | |
| **RECORTE 200x200x15 (pieza Nº9)** | 1 |
| **RECORTE 300x70x15 (piezas Nº 4, 5, 7, 8, 10 y 11)** | 6 |
| **RECORTE 100x130x15 (pieza Nº 6)** | 1 |
| **RECORTE 160x100x15 (pieza Nº 2)** | 1 |
| **RECORTE 100x70x15 (piezas Nº 1 y Nº3)** | 2 |
| **RECORTE 120x270x15 (pieza Nº 12)** | 1 |
| **VARILLAS (diametros y largo en mm)** | |
| **VARILLA ROSCADA THSL 4 HILOS 8 x200** | 1 |
| **VARILLA ROSCADA THSL 4 HILOS 8 x250** | 1 |
| **VARILLA ROSCADA THSL 4 HILOS 8 x330** | 1 |
| **VARILLA ACERADA RECTIFICADA 8 x330** | 2 |
| **VARILLA ACERADA RECTIFICADA 8 x270** | 2 |
| **VARILLA ACERADA RECTIFICADA 8 x200** | 2 |
| **RODAMIENTOS** | |
| **RODAMIENTO LINEAL 8mm CON CAMISA** | 12 |
| **RODAMIENTO 608** | 3 |
| **TUERCAS** | |
| **TUERCA THSL 4 HILOS x 2mm CON BLOQUE** | 1 |
| **TUERCA THSL 4 HILOS x 2mm ANTI-BACKLASH CON BLOQUE** | 2 |
| **MOTORES** | |
| **MOTOR PASO A PASO TIPO NEMA 17 1,8º 4,4Kgf** | 3 |
| **ACOPLE FLEXIBLE DE 5mm A 8mm** | 3 |
| **ABRAZADERAS** | |
| **ABRAZADERA TIPO OMEGA 1¼ pulg** | 1 |
| **ABRAZADERA TIPO OMEGA 1 pulg** | 1 |
| **TORNILLOS** | |
| **TORNILLOS M3 x20mm** | 21 |
| **TORNILLOS M4 x20mm** | 60 |
| **TORNILLOS PARA MADERA x30mm** | 30 |
| ***1 Ver Anexo I para la ubicación y tamaño de los orificios a realizar en los recortes MDF.*** | |

***Lista 1*** *Listado de materiales para construcción de perforadora.*

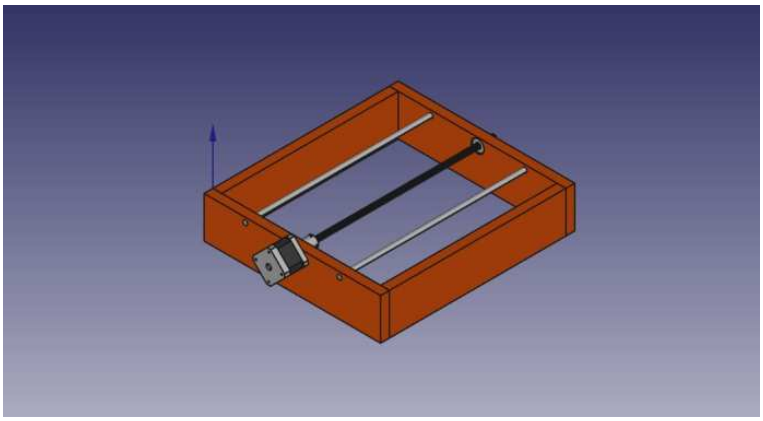
**Construcción de la perforadora**

En este apartado se explica paso a paso como montar la estructura de la perforadora de acuerdo a las piezas del Anexo I. Para hacer sencillo e intuitivo se utilizarán imágenes. Se comenzará con el eje X, correspondiente a la mesa de trabajo.

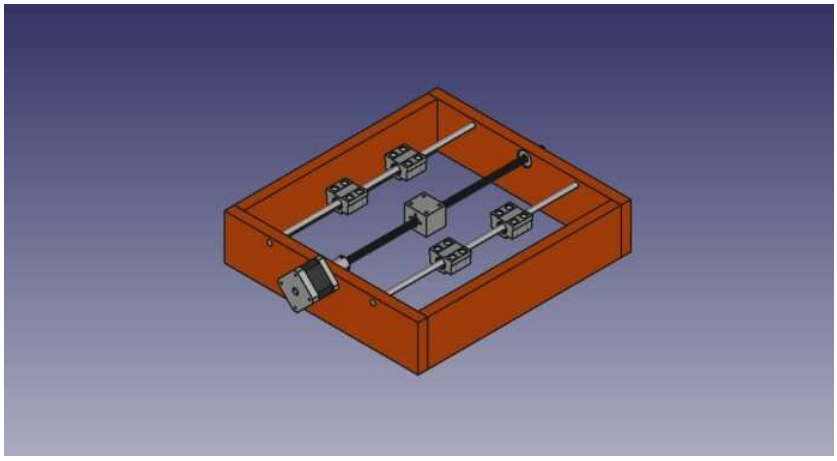
*Eje 'X':*
Con las piezas número 4, 5, 10 y 11 se monta la base de la perforadora



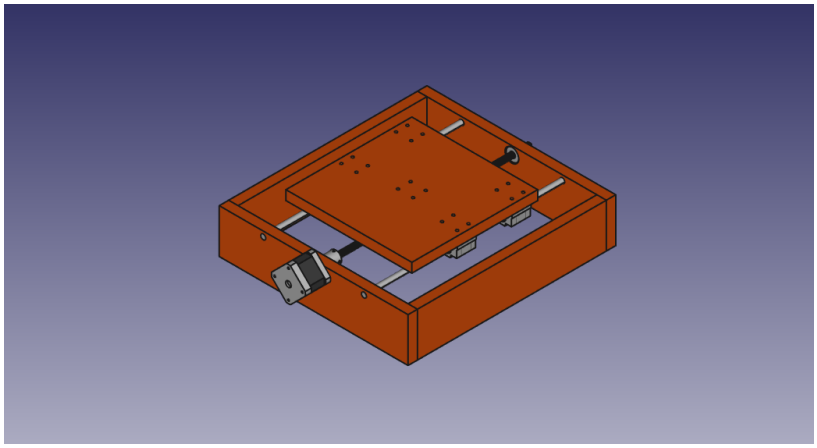
Se le agregan las varas aceradas de 8mm, la varilla roscada 8mm, el rodamiento 608 y el motor PAP con el acople.



Luego se monta la cama de fresado, que corresponde a la pieza Nº 9,con los respectivos rodamientos lineales y bloque para tuerca

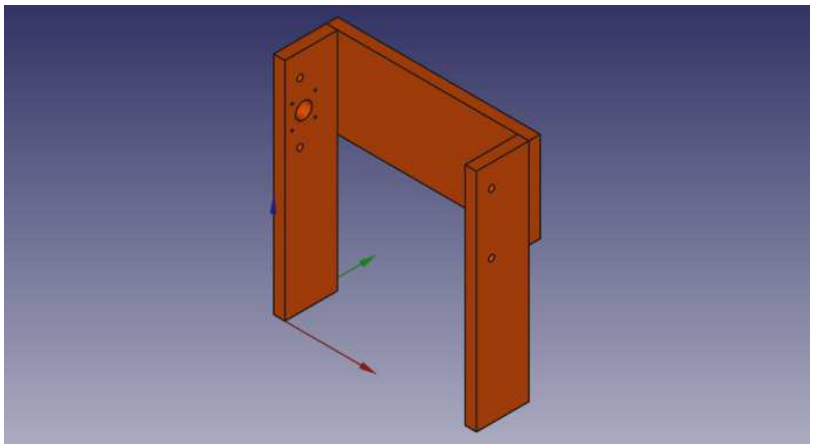


Se obtiene el eje 'X' completo según se muestra a continuación:

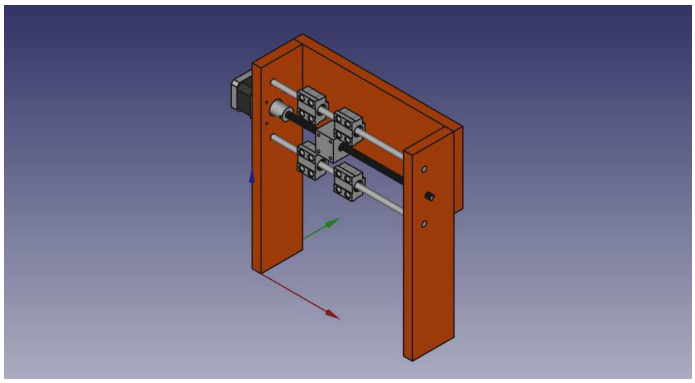


*Eje 'Y'*

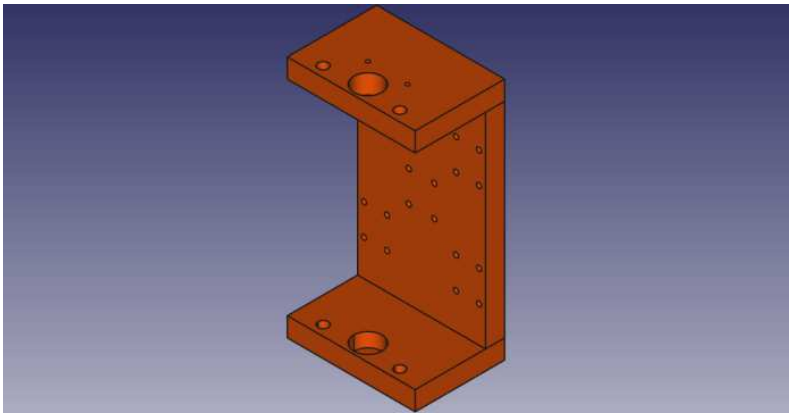
Con las piezas Nº 7, 8 y 12 se arma la siguiente estructura:



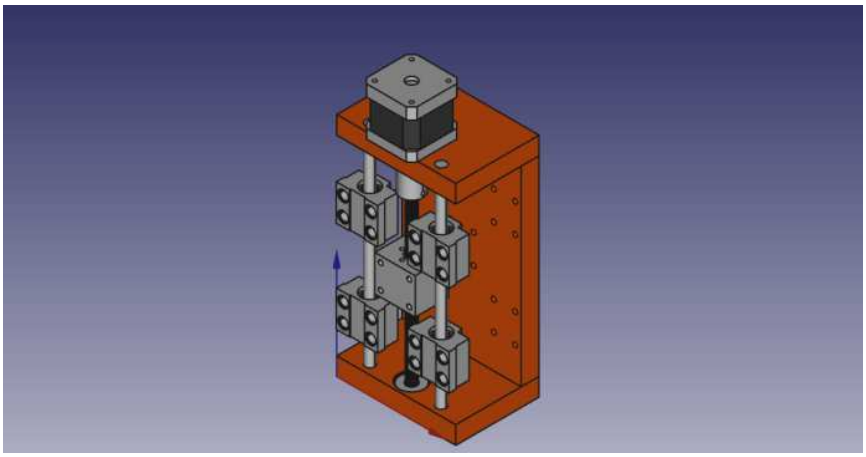
Se le agregan las varas, la varilla roscada, motor, rodamientos, etc. Debe quedar como en la siguiente imagen:



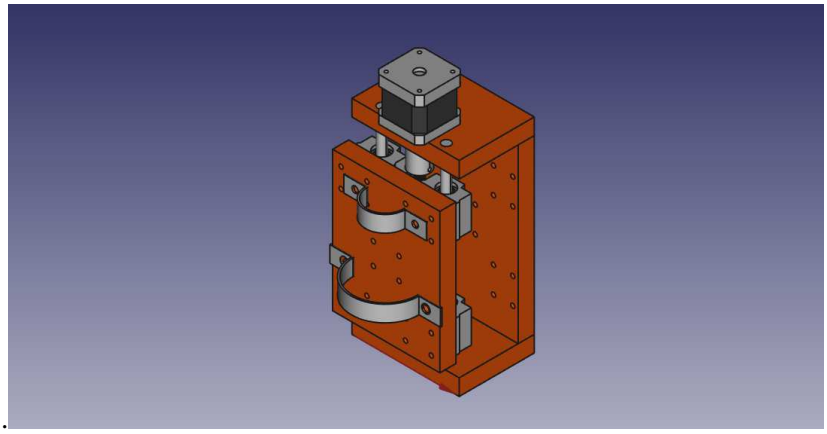
Para el carro del eje Y se parte las piezas número 1, 2, y 3 y se ensamblan como se ilustra en la siguiente imagen. Luego se une a la estructura del eje Y.



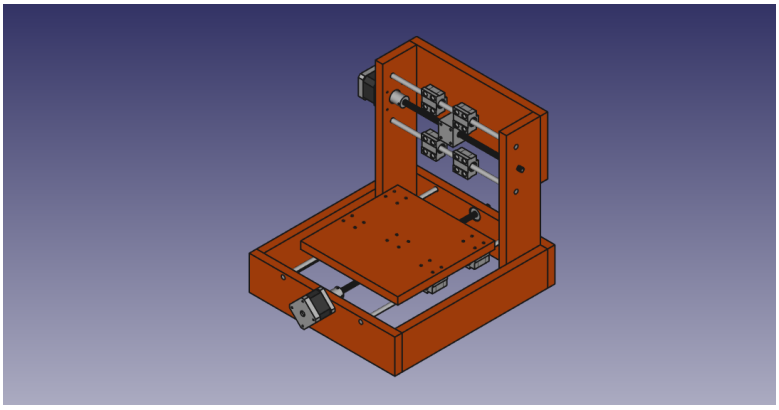
*Eje 'Z'*
El carro de éste eje se monta sobre el del eje 'Y', es por ello que se colocan las varillas, la rosca, el motor y los rodamientos según se muestra a continuación.



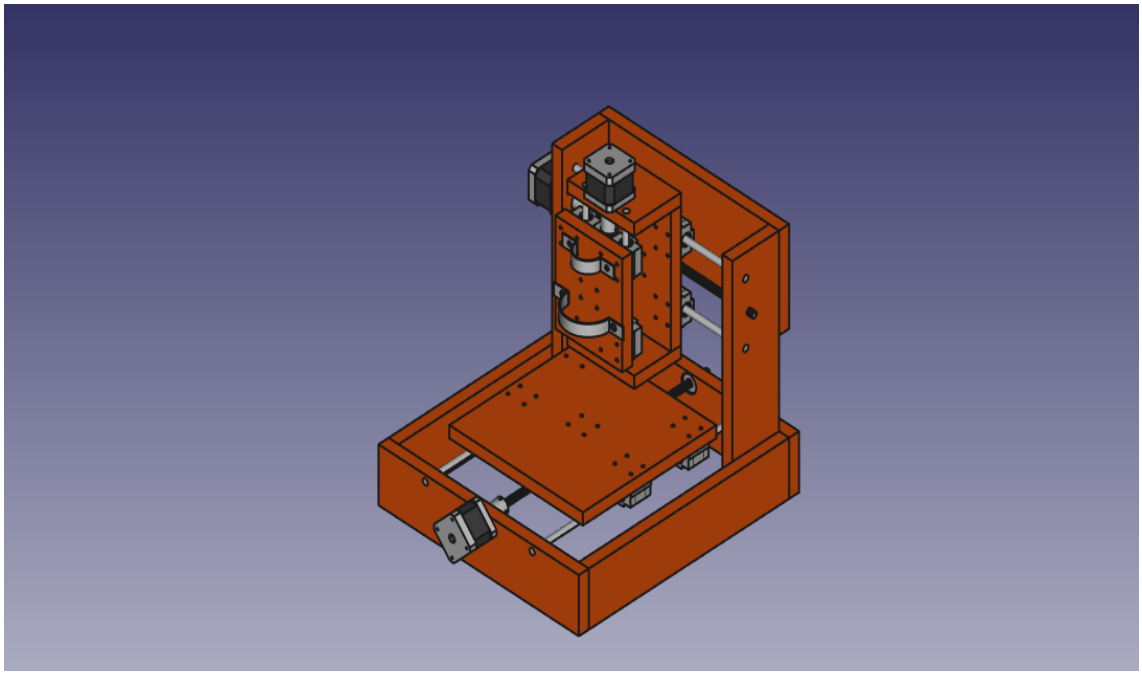
Por último se coge la piza Nº 6 que es la que sostendrá el minitorno a través de las abrazaderas se adhiere al carro.



Se fija la base y la estructura del eje Y.



La perforadora terminada queda como se muestra en la siguiente imagen:



# Anexo III

El código completo del firmware es muy extenso si se tienen en cuenta todos los archivos de cabecera (.h) y fuente (.c) que lo componen. Sin embargo, muchos de esos archivos son parte del protocolo USB de la MLA y por ende están allí para su análisis y utilización. Aquí solamente se mostrarán los códigos relacionados a la aplicación en sí, es decir, a la perforadora. Esto incluye el archivo *main.c* y *app_device_cdc_basic.c*..

*main.c*

```
/** INCLUDES *******************************************************/
#include "system.h"
#include "system_config.h"
#include <string.h>

#include "app_device_cdc_basic.h"
#include "app_led_usb_status.h"

#include "usb.h"
#include "usb_device.h"
#include "usb_device_cdc.h"
#define _XTAL_FREQ 48000000 //con 48 MHz, el argumento maximo para __delay_ms() es de 16
MAIN_RETURN main(void)
{

    ANSELH=0;               //Se configuran los pines E/S como digitales
    SYSTEM_Initialize(SYSTEM_STATE_USB_START);  //Habilitación del LED de estado
    //A continuación se configuran 3 puertos como salida. De ellos saldran pulsos
    //hacia los drivers A4988 que serviran para indicar los pasos de los motores.
    //Comenzarán con el estado logico '0'.

    //Pin en estado bajo ó '0' correspondiente a los pulsos driver motor 'X'
    TRISCbits.TRISC1=0;
    LATCbits.LATC1=0;
    //Pin en estado bajo ó '0' correspondiente a los pulsos driver motor 'Y'
    TRISCbits.TRISC2=0;
    LATCbits.LATC2=0;
    //Pin en estado bajo ó '0' correspondiente a los pulsos driver motor 'Z'
    TRISCbits.TRISC3=0;
    LATCbits.LATC3=0;
    //Similar a los puertos anteriores, se configuran 3 puertos adicionales
    //como salida hacia el Driver cuyo fin es indicar el sentido de giro de
    //cada motor. Se inicializan en estado alto ó '1' lógico.
    //Pin en estado alto ó '1' correspondiente al sentido de giro driver motor 'X'
    TRISCbits.TRISC7=0;
    LATCbits.LATC7=1;
    //Pin en estado alto ó '1' correspondiente al sentido de giro driver motor 'Y'
    TRISCbits.TRISC6=0;
    LATCbits.LATC6=1;
    //Pin en estado alto ó '1' correspondiente al sentido de giro driver motor 'Z'
    TRISCbits.TRISC5=0;
    LATCbits.LATC5=1;
    //Por ultimo, se toman 4 puertos para configurarlos como entradas (digitales)
    //que seran utilizados para los sensores de final de carro en los 3 ejes
    //(2 para el eje Z y los otros 2 para el eje X e Y).
    TRISBbits.TRISB4=1; //RB4 como entrada, sensor eje Z+
    TRISBbits.TRISB5=1; //RB5 como entrada, sensor eje Z-
    TRISBbits.TRISB6=1; //RB6 como entrada, sensor eje Y-
    TRISBbits.TRISB7=1; //RB7 como entrada, sensor eje X-
    //Se habilita el modulo USB en su estado por defecto.
    USBDeviceInit();
    //Verifica que exista tension VBUS en el puerto y, si esta conectado,
    //que le indique al host sobre ello.
    USBDeviceAttach();

    while(1)     //Ciclo continuo
    {
       SYSTEM_Tasks();  //Mantiene el sistema en ejecución
        //Se ejecuta la funcion USBDeviceTasks() periodicamente o en el
        //contexto de una interrupcion, dependiendo de como se defina.
        //Esta funcion mantiene en ejecucion la maquina de estado del stack USB
        #if defined(USB_POLLING)
            // Método de Interupción o Sondeo.  Si se utiliza el sondeo,
            // debe llamarse periodicamente la función USBDeviceTask().
            // El Host necesita que los paquetes de Setup se acepten y procesen
```

```
            // de manera oportuna, por eso debe llamarse esta funcion regularmente
            // cada 1.8ms o menos.
            // Si se opta por Interrupción, como es el caso, se encargarán
            // las subrutinas de interrupcion de dichas tareas.
        USBDeviceTasks();
        #endif
        //Si todavia no se ha configurado el dispositivo USB, se regresa al inicio
        //del ciclo while.
        if( USBGetDeviceState() < CONFIGURED_STATE )
        {
            //Se vuelve al inicio del ciclo while
            continue;
        }

        //Si se encuentra suspendido y no permite activacion remota no se puede
        //lograr comunicacion con el host, por lo que se vuelve al inicio.
        if( USBIsDeviceSuspended()== true )
        {
            //Se vuelve al inicio del ciclo while
            continue;
        }
        //Se ejecuta la tarea propia del dispositivo. Es la funcion mas
        //importante luego del main()
        APP_DeviceCDCBasicDemoTasks();
    }//fin del while(1)
}//fin del main()

//La siguiente funcion se atiende cuando existe una interupcion
//Sirve para detectar que tipo de evento a ocurrido y llevar la accion correspondiente
bool USER_USB_CALLBACK_EVENT_HANDLER(USB_EVENT event, void *pdata, uint16_t size)
{
    switch( (int) event )
    {
        case EVENT_TRANSFER:
            break;

        case EVENT_SOF:
            //El paquete SOF se utiliza como temporizador para el LED de estado
            APP_LEDUpdateUSBStatus();
            break;

        case EVENT_SUSPEND:
            //Si entra en modo suspension se apaga el led de estado
            APP_LEDUpdateUSBStatus();
            SYSTEM_Initialize(SYSTEM_STATE_USB_SUSPEND);
            break;

        case EVENT_RESUME:
            //El LED de estado destella mas lento.
            APP_LEDUpdateUSBStatus();
            SYSTEM_Initialize(SYSTEM_STATE_USB_RESUME);
            break;

        case EVENT_CONFIGURED:
            //Si se configuró el dispositivo, se reinicia el programa de la aplicacion
            APP_DeviceCDCBasicDemoInitialize();
            break;

        case EVENT_SET_DESCRIPTOR:
            break;

        case EVENT_EP0_REQUEST:
            USBCheckCDCRequest();
            break;

        case EVENT_BUS_ERROR:
            break;

        case EVENT_TRANSFER_TERMINATED:
            break;

        default:
            break;
    }
    return true;
}//fin de USER_USB_CALLBACK_EVENT_HANDLER()
```

*app_device_cdc_basic.c*

```
/** INCLUDES *******************************************************/
#include "system.h"
#include "usb.h"
#include "app_led_usb_status.h"
#include "app_device_cdc_basic.h"
#include "usb_config.h"
#define _XTAL_FREQ 48000000

/** VARIABLES ******************************************************/
static uint8_t readBuffer[CDC_DATA_OUT_EP_SIZE];
static uint8_t writeBuffer[CDC_DATA_IN_EP_SIZE];
static uint8_t eje[CDC_DATA_OUT_EP_SIZE];
static float j;
//esta variable se utiliza para identificar si se ha leido una nueva coordenada.
//Si nuevo_dato=0--> no se leyo nuevo dato. Si nuevo_dato=1--> se leyo nuevo dato
static unsigned char nuevo_dato=0;
//esta variable se utiliza para identificar si se ha leido un nuevo comando.
//(avanzar, retroceder,etc). Si: nuevo_comando=0-->no se recibió nuevo comando.
//nuevo_comando=1-->se recibio un nuevo comando.
static unsigned char nuevo_comando=0;
//flag para posicionamiento inicial
static unsigned char posicionamiento_inicial=1;

static float b=0;   //Se utiliza como acumulador

static float b_ant_x=0; //variable temporal para almacenar el valor de ultima
                        //coordenada del eje X.
static float b_ant_y=0; //variable temporal para almacenar el valor de ultima
                        //coordenada del eje Y.
static float diferencia_x;  //Para almacenar el valor en distancia que debe
                            //desplazarse en eje X.
static float diferencia_y;  //Para almacenar el valor en distancia que debe
                            //desplazarse en eje Y.
static unsigned char signo_x;   //utilizada para identificar sentido de desplazamiento en eje X
static unsigned char signo_y;   //utilizada para identificar sentido de desplazamiento en eje X
//En estas variables se almacenan el numero de pasos que debe aplicarse a los respectivos motores
static float pasosx;
static float pasosy;

//******************************************************************//
//Funcion que inicializa el puerto COM Virtual
void APP_DeviceCDCBasicDemoInitialize()
{
    CDCInitEP();

    line_coding.bCharFormat = 0;
    line_coding.bDataBits = 8;
    line_coding.bParityType = 0;
    line_coding.dwDTERate = 9600;
}
//******************************************************************//
//Mantiene la aplicacion en ejecucion
void APP_DeviceCDCBasicDemoTasks()
{
    //-------------------------------------------------------------------------//
    //aqui se autoposiciona el taladro
    if(posicionamiento_inicial==1)
    {
    //Aqui avanza hasta que el sensor 1 este a '0', es decir, se posicione en la parte superior
    while (PORTBbits.RB4==1)
     {
        if(PORTBbits.RB5==0 && PORTBbits.RB4==1)
        {
            LATCbits.LATC5=0;
        }
        LATCbits.LATC3^=1;
        __delay_ms(1);
     }
    //Aqui se posiciona el eje X
    LATCbits.LATC7=0;
    while(PORTBbits.RB7==1)
        {
        LATCbits.LATC1^=1;
        __delay_us(100);
        }
     //Aqui se posiciona el eje Y
    LATCbits.LATC6=0;
    while(PORTBbits.RB6==1)
        {
        LATCbits.LATC2^=1;
        __delay_us(100);
```

```
        }
    b_ant_x=0;       //Se tiene que borrar, porque sino al regresar a posicion inicial queda
    b_ant_y=0;       //guardado ese valor y tiene que ser 0 en ambas coordenadas.
    posicionamiento_inicial=0;
    }

    //------------------------------------------------------------------------//
    //Se verifica si existe una transmision de datos en progreso. Si no la hay, se procede a leer
    //el siguiente dato y procesarlo.
    if( USBUSARTIsTxTrfReady() == true)
    {
        uint16_t numBytesRead;
        numBytesRead = getsUSBUSART(readBuffer, sizeof(readBuffer));
        //Si el primer elemento del string es 'x' ó 'X', entonces es coordenada
        if((readBuffer[0]=='x') || (readBuffer[0]=='X'))
        {
            //Se asegura de que el dato leido sea de 14 bytes (debido al formato del mismo).
            //cualquier otra cantidad de bytes no se considera una coordenada.
                if(numBytesRead == CDC_DATA_OUT_EP_SIZE)
                {
                    nuevo_dato=1;
                }
        }
        //si el primer elemento del string es 'f', 'b', 'l', 'r', 'u' ó 'd', significa que es un
        //comando de calibracion. El carro se desplaza POR CADA nuevo comando recibido.
         if((readBuffer[0]=='f') || (readBuffer[0]=='b') ||(readBuffer[0]=='l') ||
                 (readBuffer[0]=='r') || (readBuffer[0]=='u') || (readBuffer[0]=='d')
                 || (readBuffer[0]=='o'))
            {
                 //se actualiza la variable nuevo_comando.
                if(numBytesRead == 1)
                {
                nuevo_comando = 1;
                }
            }
        //si el primer elemento del string es 'F', 'B', 'L', 'R', 'U' ó 'D', significa que es un
        //comando de calibracion. A diferencia de los comandos anteriores, el carro
correspondiente
        //CONTINUA desplazandose hasta que se reciba una nueva orden.
        if((readBuffer[0]=='F') || (readBuffer[0]=='B') || (readBuffer[0]=='L') ||
                (readBuffer[0]=='R'))
        {
            if(numBytesRead == 1)
                nuevo_comando ^= 1;
        }
    }
    //------------------------------------------------------------------------//
    //Esta seccion de codigo se procesan los datos recibidos, ya sean comandos o
    //coordenadas.
    CDCTxService();//Funcion que realiza las transacciones del dispositivo al Host

    //Si el dato que se recibió fue un comando...
    if(nuevo_comando==1)
    {
        //Se verifica solamente el primer byte del string, ya que solo se envia
        //uno solo desde el host.
        switch(readBuffer[0])
        {
            //En caso de 'f' ó 'F', avanza el carro del eje X en sentido negativo
            //ó -X hasta que llegue al final de carro.
            case 'f':
            case 'F':
            LATCbits.LATC7=0;
            if(PORTBbits.RB7==1)
                {
                LATCbits.LATC1=1;
                __delay_us(50);
                LATCbits.LATC1=0;
                __delay_us(50);
                }
            break;

            //En caso de 'b' ó 'B', avanza el carro del eje X en sentido positivo
            //ó +X.
            case 'b':
            case 'B':
            LATCbits.LATC7=1;
            LATCbits.LATC1=1;
            __delay_us(50);
            LATCbits.LATC1=0;
            __delay_us(50);
```

```
            break;

            //En caso de 'l' ó 'L', avanza el carro del eje Y en sentido negativo
            //ó -Y hasta que llegue al final de carro.
            case 'l':
            case 'L':
            LATCbits.LATC6=0;
            if(PORTBbits.RB6==1)
                {
                LATCbits.LATC2=1;
                __delay_us(50);
                LATCbits.LATC2=0;
                __delay_us(50);
                }
            break;

            //En caso de 'r' ó 'R', avanza el carro del eje Y en sentido positivo
            //ó +Y.
            case 'r':
            case 'R':
            LATCbits.LATC6=1;
            LATCbits.LATC2=1;
            __delay_us(50);
            LATCbits.LATC2=0;
            __delay_us(50);
            break;

            //En caso de 'u', avanza el carro del eje Z en sentido positivo
            //ó +Z hasta que llegue al final de carro.
            case 'u':
                LATCbits.LATC5=0;
            if(PORTBbits.RB4==1)
                {
                for(j=0;j<20;j++)
                    {
                    LATCbits.LATC3^=1;
                    __delay_ms(1);
                }
                }
            break;

            //En caso de 'd', avanza el carro del eje Z en sentido negativo
            //ó -Z hasta que llegue al final de carro.
            case 'd':           //en caso de que sea la 'd'
                LATCbits.LATC5=1;
                if(PORTBbits.RB5==1)
                {
                for(j=0;j<20;j++)
                    {
                    LATCbits.LATC3^=1;
                    __delay_us(800);
                    }
                }
            break;

            //En caso de que el comando recibido sea 'o', se setea la variable
            //posicionamiento_inicial, lo cual provocará que la perforadora vuelva
            //a su posicion de inicio.
            case 'o':
            posicionamiento_inicial=1;
            break;

            //En cualquier otro caso, que no realice ninguna accion.
            default: break;
        }
        //Al terminar de procesar el ultimo comando recibido se setea o borra
        //el flag correspondiente para el siguiente comando a recibir
        //Si fue un comando para avance continuo...
        if((readBuffer[0]=='F') || (readBuffer[0]=='B') || (readBuffer[0]=='L')
          || (readBuffer[0]=='R'))
        nuevo_comando=1;
        //Si fue un comando para avance por pasos...
        if((readBuffer[0]=='f') || (readBuffer[0]=='b') || (readBuffer[0]=='l')
          || (readBuffer[0]=='r') || (readBuffer[0]=='u') || (readBuffer[0]=='d')
          || (readBuffer[0]=='o'))
        nuevo_comando=0;
    }// fin if(nuevo_comando==1)


/*****************************************************************************************/
    //Se copia el dato del buffer leído a otra variable para no modificarlo
```

```
    strcpy(eje,readBuffer);
    //La coordenada que envia el host posee el siguiente formato:
    //X123456Y123456. Entonces se cambian los caracteres 'X' e 'Y' por
    //ceros. Si los 14 bytes son numeros quiere decir que es una coordenada
    //valida y se procesa. De lo contrario se descarta.
    eje[0]='0';
    eje[7]='0';
    for(char subindice=0;subindice<CDC_DATA_OUT_EP_SIZE;subindice++)
        {
          //Si todos los bytes corresponden a numeros en ascii continua...
          if(eje[subindice]>=0x30&&eje[subindice]<=0x39)
              continue;
          //Si no, no es una coordenada valida y se borra la bandera nuevo_dato
          else
              nuevo_dato=0;
        }
    //Si el dato que se recibió fue una coordenada...
    if(nuevo_dato==1)
    {
        //la conversion se inicia restandole al dato en ASCII el valor 30h
        //el resultado estará comprendido entre 0 y 9.
        //se lo multiplica por 100 para obtener los centenares
        //y de alli se acumula el valor en la variable b tipo float
        b=b+((eje[1]-0x30)*100);
        //de igual forma se procede con el resto de elementos del string
        //se obtiene las decenas y se acumulan en la variable b
        b=b+((eje[2]-0x30)*10);
        //se obtienen las unidades y se acumulan
        b=b+(eje[3]-0x30);
        //se obtienen las decimas y se acumulan
        b=b+((eje[4]-0x30)*0.1);
        //se obtienen las centesismas y se acumulan
        b=b+((eje[5]-0x30)*0.01);
        //se obtienen las milesimas y se acumula a b
        b=b+((eje[6]-0x30)*0.001);
        // en la variable float b ya esta disponible el valor en binario.
        //este es al valor actual, se debe restar al valor anterior para obtener  la
        //diferencia de distancia que se debe avanzar. Se hace b_ant_x - b = diferencia
        //se debe inicializar b_ant_x a cero para el comienzo del programa. De este modo,
        //si el primer valor que se lee es X000000 no se desplazara la CNC.

        //para saber el signo se compara b_ant_x con b:si b_ant_x es mayor o igual a b entonces
el
        //resultado o diferencia es positiva,sino, si b es mayor a b_ant_x
        if(b_ant_x>=b)
            {
            signo_x=0;   //El signo positivo esta repredentado por un '0'
            LATCbits.LATC7=0;
            }
        else
            {
            signo_x=1;     //El signo negativo esta representado por un '1'
            LATCbits.LATC7=1;
            }

        //Se obtiene lo que debe desplazarse la CNC en milimetros en el eje X.
        //Se pasa el valor actual de b a b_ant_x para el siguiente calculo.
        //se debe borrar el contenido de b porque sino se seguiria acumulando
        //con los valores posteriores.
        diferencia_x=b_ant_x-b;
        b_ant_x=b;
        b=0;

        //Ahora se convierten en decimal la distancia del eje Y. El proceso es
        //exactamente igual al del eje X
        b=b+((eje[8]-0x30)*100);
        b=b+((eje[9]-0x30)*10);
        b=b+(eje[10]-0x30);
        b=b+((eje[11]-0x30)*0.1);
        b=b+((eje[12]-0x30)*0.01);
        b=b+((eje[13]-0x30)*0.001);
        //Para saber el signo se realiza el mismo algoritmo que en el caso anterior
        //Si el valor anterior es mayor al actual, entonces la diferencia es positiva y
        //el carro de dicho avanzará en el eje positivo. De lo contrario avanzará
        //en sentido opuesto ó -Y.
        if(b_ant_y>=b)
            {
            signo_y=0;    //El signo positivo esta repredentado por un '0'
            LATCbits.LATC6=0;
            }
        else
```

```
        {
         signo_y=1;       //El signo negativo esta representado por un '1'
         LATCbits.LATC6=1;
        }
        //Se obtiene lo que debe desplazarse la CNC en milimetros en el eje Y.
        //Se pasa el valor actual de b a b_ant_y para el siguiente calculo.
        //se debe borrar el contenido de b porque sino se seguiria acumulando
        //con los valores posteriores.
        diferencia_y=b_ant_y-b;
        b_ant_y=b;
        b=0;
        //Con el valor obtenido de la diferencia de distancia y conociendo el signo se puede
        //calcular la cantidad de pasos que debe hacer el motor PAP para avnazar esa distancia y
        //en que sentido. Con ese valor de diferencia se calcula la cantidad de pasos que deben
        //aplicarse al motor para que avance dicha distancia. Primero se le quita el signo
negativo
        //si es que lo es. Con ese dato ya se sabe el sentido de giro del motor (PIN DIR del
Driver).
        //Luego con una regla de 3 simple directa se calculan los pasos.
        //Para el motor de la mesa, eje X, cada vuelta del motor corresponde a:
        //200 (que son los pasos del motor) por 16(que multiplica el driver A4988)=3200 pasos
        //y recorre 8 mm, que es el paso de la varilla roscada.
        // Entonces:
        //                            8mm------------3200 pasos
        //                       X= 0.0025mm-----------1 paso
        //Es decir, que para ese eje, por cada paso que haga se desplazara 0.0025mm
        //Por lo tanto, para  hacer desplazar la distancia requerida, se debe multiplicar el
valor
        //de diferencia por 3200 y a ese valor dividorlo por 8.
        if(signo_x==1)
            {
            diferencia_x=diferencia_x*(-1);     //para quitarle el signo
            }
        pasosx=(diferencia_x*3200)/8;

        //Se asegura que el taladro regrese a la posicion superior antes de desplazarse
        //en los ejes X e Y, asi se evita que se rompa la broca al moverse
        LATCbits.LATC5=0;
        while(PORTBbits.RB4==1)
            {
            LATCbits.LATC3=1;
            __delay_ms(1);
            LATCbits.LATC3=0;
            __delay_ms(1);
            }
        //El siguiente ciclo 'for' envia los pulsos correspondientes al driver
        //del eje X para gire el motor paso a paso y se posicione segun la coordenada recibida.
        for(j=0;j<pasosx;j++)
            {
            LATCbits.LATC1=1;
            __delay_us(10);
            LATCbits.LATC1=0;
            __delay_us(10);
            }
        //Para el motor del ejeY es identico:, se debe tomar el valor de diferencia,
        //multiplicarlo por 3200 y dividirlo por 8

        if(signo_y==1)
            {
            diferencia_y=diferencia_y*(-1);    //para quitarle el signo negativo
            }
        pasosy=(diferencia_y*3200)/8;
        //El siguiente ciclo 'for' envia los pulsos correspondientes al driver
        //del eje Y para gire el motor paso a paso y se posicione segun la coordenada recibida.
        for(j=0;j<pasosy;j++)
            {
            LATCbits.LATC2=1;
            __delay_us(10);
            LATCbits.LATC2=0;
            __delay_us(10);
            }
        //Hasta aqui se tiene el taladro en la posicion exacta en donde taladrar
        //Lo siguiente es accionar el eje de perforacion, es decir, el eje
        //Z y realizar el agujero en dicha coordenada.

        /*en esta ultima seccion de código se realiza la perforacion*/
        //Mientras el carro no active el sensor inferior que siga bajando...
        while(PORTBbits.RB5==1)
            {
            LATCbits.LATC5=1;
            LATCbits.LATC3=1;
```

```
            __delay_ms(2);
            LATCbits.LATC3=0;
            __delay_ms(2);
            }
        //Si el taladro llegó hasta el fin de carro se cambia el sentido de giro
        if(PORTBbits.RB5==0)LATCbits.LATC5=0;
        //Mientras el taladro no active ahora el sensor superior que siga subiendo
        while(PORTBbits.RB4==1)
            {
            LATCbits.LATC3=1;
            __delay_ms(1);
            LATCbits.LATC3=0;
            __delay_ms(1);
            }
        //Se ha procesado una coordana completamente. Se borra el flag correspondiente
        //para la llegada del proximo dato y posterior procesamiento.
        nuevo_dato=0;
    }//fin de if(nuevo_dato==1)
}//fin de void APP_DeviceCDCBasicDemoTasks()
```

# Anexo IV

A continuación se muestra el código de la aplicación creada en la PC. Se recuerda que fue programado en Visual C++ Express Edition 2008.

```
#pragma once


#define USB_VENDOR_ID     0x04D8   //ID del fabricante de 16 bits (ver descriptor del dispositivo)
#define USB_PRODUCT_ID    0x000A   //ID del producto de 16 bits (ver descriptor de dispositivo)
#define USB_DEVICE_INSTANCE       0       //Siempre '0', a menos que existan multiples instancias del
dispositivo
                                                                    //conectados al mismo tiempo. En ese
caso el indice de instancia comienza
                                                                    //en 0 y se va incrementando por cada
dispositivo conectado actualmente
                                                                    //al sistema( que coincidan con el VID y
PID)
#define USB_IS_DEVICE_COMPOSITE   false    //Usar 'false' si no es un dispositivo compuesto. De lo contrario
'true'
#define USB_COMPOSITE_INTERFACE_INDEX 0    //Irrelevante cuando no es un dispositivo compuesto

//-----------------------------------------------------------------------------------------------------//
#include <windows.h>      //Define tipos de datos y demas necesarias para usar las funciones setupapi.dll
#include <setupapi.h>     //Contiene definiciones de las funciones setupapi.dll

namespace Perforadora_Automatica {

        using namespace System;
        using namespace System::ComponentModel;
        using namespace System::Collections;
        using namespace System::Windows::Forms;
        using namespace System::Data;
        using namespace System::Drawing;
        using namespace System::Security::Permissions;
        using namespace Microsoft::Win32;
        using namespace System::Runtime::InteropServices; //Necesario para llamar las funciones API


        #ifdef UNICODE
        #define Seeifdef Unicode
        #else
        #define Seeifdef Ansi
        #endif
        //Devuelve un tipo HDEVINFO que es un conjunto de information de dispositivos( USB en nuestro caso)
        //Se necesita el HDEVINFO como parametro de entrada para otras funciones de SetupDixxx().
        [DllImport("setupapi.dll",CharSet=CharSet::Seeifdef,EntryPoint="SetupDiGetClassDevs",CallingConvent
ion=CallingConvention::Winapi)]
        extern "C" HDEVINFO SetupDiGetClassDevsUM(
                LPGUID ClassGuid,                       //Entrada:Aqui va el GUID (GUID de clase de
interfaz USB en este caso
                PCTSTR Enumerator,                      //Entrada: NULL. No importa en este caso
                HWND hwndParent,                  //Entrada: NULL. No importa en este caso
                DWORD Flags);                           //Entrada: las banderas filtran el resultado(por
ejemplo que busque
                                                                    //dispositivos dados por el
GUID de interfaz USB y que esten conectados

        //Devuelve una estructura de datos del tipo "PSP_DEVICE_INTERFACE_DATA" el cual contiene el GUID
        //de interfaz especifico( que es diferente al GUID de clase). Se necesita el GUID de interfaz
        //para encontrar la ruta del dispositivo (device path).
        [DllImport("setupapi.dll",CharSet= CharSet::Seeifdef,EntryPoint="SetupDiEnumDeviceInterfaces",
                CallingConvention=CallingConvention::Winapi)]
        extern "C" WINSETUPAPI BOOL WINAPI SetupDiEnumDeviceInterfacesUM(
                HDEVINFO DeviceInfoSet,                                                 //Entrada:
HDEVINFO obtenida de

                //SetupDiGetClassDevs()
                PSP_DEVINFO_DATA DeviceInfoData,                //Entrada: Opcional
                LPGUID InterfaceClassGuid,                                              //Entrada:GUID
de clase
                DWORD MemberIndex,
        //Entrada: "Indice" del dispositivo al que se

                //quiere obtener la ruta.
                PSP_DEVICE_INTERFACE_DATA DeviceInterfaceData);    //Salida:Esta funcion devuelve una
estructura de datos

                // del tipo SP_DEVICE_INTERFACE_DATA
        //SetupDiDestroyDeviceInfoList() libera la memoria borrando la lista de informacion de dispositivos
        //(el conjunto de informacion de dispositivos)
        [DllImport("Setupapi.dll",CharSet=CharSet::Seeifdef,EntryPoint="SetupDiDestroyDeviceInfoList",
                CallingConvention=CallingConvention::Winapi)]
```

```
        extern "C" WINSETUPAPI BOOL WINAPI SetupDiDestroyDeviceInfoListUM(
                HDEVINFO DeviceInfoSet);          //Entrada: Lista de informacion de dispositivo
                                                                        //que se desea borrar
de la RAM

        //SetupDiEnumDeviceInfo() devuelve una estructura del tipo "SP_DEVINFO_DATA", el cual se
        //necesita para la funcion SetupDiGetDeviceRegistryProperty()
        [DllImport("Setupapi.dll",CharSet=CharSet::Seeifdef,EntryPoint="SetupDiEnumDeviceInfo",
                CallingConvention=CallingConvention::Winapi)]
        extern "C" WINSETUPAPI BOOL WINAPI SetupDiEnumDeviceInfoUM(
                HDEVINFO DeviceInfoSet,
                DWORD MemberIndex,
                PSP_DEVINFO_DATA DeviceInfoData);

        //SetupDiGetDeviceRegistryProperty() devuelve el Hardware ID, el cual contiene el PID
        //y el VID
        [DllImport("Setupapi.dll",CharSet=CharSet::Seeifdef,EntryPoint="SetupDiGetDeviceRegistryProperty",
                CallingConvention=CallingConvention::Winapi)]
        extern "C" WINSETUPAPI BOOL WINAPI SetupDiGetDeviceRegistryPropertyUM(
                HDEVINFO DeviceInfoSet,
                PSP_DEVINFO_DATA DeviceInfoData,
                DWORD Property,
                PDWORD PropertyRegDataType,
                PBYTE PropertyBuffer,
                DWORD PropertyBufferSize,
                PDWORD RequiredSize);

        //-----------------------------------------------------------------
        //Variables Globales
        //-----------------------------------------------------------------
        BOOL DeviceAttachedState=FALSE;                 //Utilizado para detectar estado del hardware
        BOOL OldDeviceAttachedState=FALSE;              //Utilizado para detectar estado del hardware

        /// <summary>
        /// Summary for Form1
        ///
        /// WARNING: If you change the name of this class, you will need to change the
        ///          'Resource File Name' property for the managed resource compiler tool
        ///          associated with all .resx files this class depends on.  Otherwise,
        ///          the designers will not be able to interact properly with localized
        ///          resources associated with this form.
        /// </summary>
        public ref class Form1 : public System::Windows::Forms::Form
        {
/*      //Se crea un delegado (un puntero a un metodo) para tomar los datos leidos del puerto
        //Serie y escribirlos en una caja de texto, ya que la lectura de los datos del puerto
        //se realizan en un hilo secundario del programa
        delegate void SetTextCallBack(String^ text);*/

        public:
                Form1(void)
                {
                        InitializeComponent();
                }

        public:
                void EnviarlineasdeTexto()
                {
                //Se crea un array de Strings y en cada posicion se almacena el contenido de cada linea de
texto
                //de la caja de texto txtSendData
                array<String^>^ tempArray=gcnew array<String^>(txtSendData->Lines->Length);
                tempArray=txtSendData->Lines;
                ProgressBarDrill->Maximum=txtSendData->Lines->Length;
                static int counter=0;
                //Se realiza un bucle para escribir cada elemento del array(cada linea de txtSendData)
                //en el puerto COM
                for(int counter=0;counter<tempArray->Length;counter++)
                        {
                        ProgressBarDrill->Increment(1);
                        serialPort1->WriteLine(tempArray[counter]);
                    }
                }

        protected:
                /// <summary>
                /// Clean up any resources being used.
                /// </summary>
                ~Form1()
                {
```

```
                        if (components)
                        {
                                delete components;
                        }
                }
        private: System::Windows::Forms::TextBox^  txtSendData;
        private: System::Windows::Forms::ProgressBar^  ProgressBarDrill;
        private: System::IO::Ports::SerialPort^  serialPort1;
        private: System::Windows::Forms::Button^  btnSendData;
        private: System::Windows::Forms::Timer^  timer1;
        private: System::Windows::Forms::Label^  COMStatus_lbl;
        private: System::Windows::Forms::Button^  btnAxisZminus;
        private: System::Windows::Forms::Button^  btnAxisZplus;
        private: System::Windows::Forms::Button^  btnAxisXminus;
        private: System::Windows::Forms::Button^  btnAxisXplus;
        private: System::Windows::Forms::Button^  btnAxisYplus;
        private: System::Windows::Forms::Button^  btnAxisYminus;
        private: System::Windows::Forms::Button^  btnReset;
        private: System::Windows::Forms::Button^  btnOrigin;
        private: System::Windows::Forms::ToolTip^  toolTip1;
        private: System::Windows::Forms::Button^  btnAxisXplusfast;
        private: System::Windows::Forms::Button^  btnAxisXminusfast;
        private: System::Windows::Forms::Button^  btnAxisYplusfast;
        private: System::Windows::Forms::Button^  btnAxisYminusfast;



        private: System::ComponentModel::IContainer^  components;

        protected:

        protected:

        private:
                /// <summary>
                /// Required designer variable.
                /// </summary>


#pragma region Windows Form Designer generated code
                /// <summary>
                /// Required method for Designer support - do not modify
                /// the contents of this method with the code editor.
                /// </summary>
                void InitializeComponent(void)
                {
                        this->components = (gcnew System::ComponentModel::Container());
                        System::ComponentModel::ComponentResourceManager^  resources = (gcnew
System::ComponentModel::ComponentResourceManager(Form1::typeid));
                        this->btnAxisZplus = (gcnew System::Windows::Forms::Button());
                        this->txtSendData = (gcnew System::Windows::Forms::TextBox());
                        this->ProgressBarDrill = (gcnew System::Windows::Forms::ProgressBar());
                        this->serialPort1 = (gcnew System::IO::Ports::SerialPort(this->components));
                        this->btnSendData = (gcnew System::Windows::Forms::Button());
                        this->timer1 = (gcnew System::Windows::Forms::Timer(this->components));
                        this->COMStatus_lbl = (gcnew System::Windows::Forms::Label());
                        this->btnAxisZminus = (gcnew System::Windows::Forms::Button());
                        this->btnAxisXminus = (gcnew System::Windows::Forms::Button());
                        this->btnAxisXplus = (gcnew System::Windows::Forms::Button());
                        this->btnAxisYplus = (gcnew System::Windows::Forms::Button());
                        this->btnAxisYminus = (gcnew System::Windows::Forms::Button());
                        this->btnReset = (gcnew System::Windows::Forms::Button());
                        this->btnOrigin = (gcnew System::Windows::Forms::Button());
                        this->toolTip1 = (gcnew System::Windows::Forms::ToolTip(this->components));
                        this->btnAxisXplusfast = (gcnew System::Windows::Forms::Button());
                        this->btnAxisXminusfast = (gcnew System::Windows::Forms::Button());
                        this->btnAxisYplusfast = (gcnew System::Windows::Forms::Button());
                        this->btnAxisYminusfast = (gcnew System::Windows::Forms::Button());
                        this->SuspendLayout();
                        //
                        // btnAxisZplus
                        //
                        this->btnAxisZplus->BackgroundImage = (cli::safe_cast<System::Drawing::Image^
>(resources->GetObject(L"btnAxisZplus.BackgroundImage")));
                        this->btnAxisZplus->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Stretch;
                        this->btnAxisZplus->Cursor = System::Windows::Forms::Cursors::Hand;
                        this->btnAxisZplus->Font = (gcnew System::Drawing::Font(L"Segoe Print", 11,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                                static_cast<System::Byte>(0)));
```

```
                            this->btnAxisZplus->Location = System::Drawing::Point(267, 131);
                            this->btnAxisZplus->Name = L"btnAxisZplus";
                            this->btnAxisZplus->Size = System::Drawing::Size(51, 47);
                            this->btnAxisZplus->TabIndex = 4;
                            this->toolTip1->SetToolTip(this->btnAxisZplus, L"Avanza en eje Z");
                            this->btnAxisZplus->UseVisualStyleBackColor = true;
                            this->btnAxisZplus->Click += gcnew System::EventHandler(this,
&Form1::btnAxisZplus_Click);
                            //
                            // txtSendData
                            //
                            this->txtSendData->Font = (gcnew System::Drawing::Font(L"Segoe Print", 9.75F,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                                    static_cast<System::Byte>(0)));
                            this->txtSendData->Location = System::Drawing::Point(355, 54);
                            this->txtSendData->Multiline = true;
                            this->txtSendData->Name = L"txtSendData";
                            this->txtSendData->ScrollBars = System::Windows::Forms::ScrollBars::Vertical;
                            this->txtSendData->Size = System::Drawing::Size(181, 367);
                            this->txtSendData->TabIndex = 0;
                            //
                            // ProgressBarDrill
                            //
                            this->ProgressBarDrill->Location = System::Drawing::Point(13, 440);
                            this->ProgressBarDrill->Name = L"ProgressBarDrill";
                            this->ProgressBarDrill->Size = System::Drawing::Size(523, 35);
                            this->ProgressBarDrill->TabIndex = 1;
                            //
                            // btnSendData
                            //
                            this->btnSendData->Cursor = System::Windows::Forms::Cursors::Hand;
                            this->btnSendData->Enabled = false;
                            this->btnSendData->Location = System::Drawing::Point(355, 12);
                            this->btnSendData->Name = L"btnSendData";
                            this->btnSendData->Size = System::Drawing::Size(182, 35);
                            this->btnSendData->TabIndex = 2;
                            this->btnSendData->Text = L"Enviar Datos";
                            this->toolTip1->SetToolTip(this->btnSendData, L"Enviar coordenadas ");
                            this->btnSendData->UseVisualStyleBackColor = true;
                            this->btnSendData->Click += gcnew System::EventHandler(this,
&Form1::btnSendData_Click);
                            //
                            // timer1
                            //
                            this->timer1->Enabled = true;
                            this->timer1->Interval = 1000;
                            this->timer1->Tick += gcnew System::EventHandler(this, &Form1::timer1_Tick);
                            //
                            // COMStatus_lbl
                            //
                            this->COMStatus_lbl->AutoSize = true;
                            this->COMStatus_lbl->BorderStyle =
System::Windows::Forms::BorderStyle::FixedSingle;
                            this->COMStatus_lbl->Location = System::Drawing::Point(10, 12);
                            this->COMStatus_lbl->Name = L"COMStatus_lbl";
                            this->COMStatus_lbl->Size = System::Drawing::Size(135, 21);
                            this->COMStatus_lbl->TabIndex = 3;
                            this->COMStatus_lbl->Text = L"Estado: No Conectado ";
                            //
                            // btnAxisZminus
                            //
                            this->btnAxisZminus->BackgroundImage = (cli::safe_cast<System::Drawing::Image^
>(resources->GetObject(L"btnAxisZminus.BackgroundImage")));
                            this->btnAxisZminus->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Stretch;
                            this->btnAxisZminus->Cursor = System::Windows::Forms::Cursors::Hand;
                            this->btnAxisZminus->Font = (gcnew System::Drawing::Font(L"Segoe Print", 15.75F,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                                    static_cast<System::Byte>(0)));
                            this->btnAxisZminus->Location = System::Drawing::Point(267, 184);
                            this->btnAxisZminus->Name = L"btnAxisZminus";
                            this->btnAxisZminus->Size = System::Drawing::Size(51, 47);
                            this->btnAxisZminus->TabIndex = 5;
                            this->toolTip1->SetToolTip(this->btnAxisZminus, L"Retrocede en eje Z");
                            this->btnAxisZminus->UseVisualStyleBackColor = true;
                            this->btnAxisZminus->Click += gcnew System::EventHandler(this,
&Form1::btnAxisZminus_Click);
                            //
                            // btnAxisXminus
                            //
```

```
                        this->btnAxisXminus->BackColor = System::Drawing::Color::Firebrick;
                        this->btnAxisXminus->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Stretch;
                        this->btnAxisXminus->Cursor = System::Windows::Forms::Cursors::Hand;
                        this->btnAxisXminus->Font = (gcnew System::Drawing::Font(L"Segoe Print", 15.75F,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                                static_cast<System::Byte>(0)));
                        this->btnAxisXminus->Location = System::Drawing::Point(67, 158);
                        this->btnAxisXminus->Name = L"btnAxisXminus";
                        this->btnAxisXminus->Size = System::Drawing::Size(21, 47);
                        this->btnAxisXminus->TabIndex = 6;
                        this->toolTip1->SetToolTip(this->btnAxisXminus, L"Retrocede un paso en eje X");
                        this->btnAxisXminus->UseVisualStyleBackColor = false;
                        this->btnAxisXminus->Click += gcnew System::EventHandler(this,
&Form1::btnAxisXminus_Click);
                        //
                        // btnAxisXplus
                        //
                        this->btnAxisXplus->BackColor = System::Drawing::Color::Firebrick;
                        this->btnAxisXplus->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Stretch;
                        this->btnAxisXplus->Cursor = System::Windows::Forms::Cursors::Hand;
                        this->btnAxisXplus->Font = (gcnew System::Drawing::Font(L"Segoe Print", 15.75F,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                                static_cast<System::Byte>(0)));
                        this->btnAxisXplus->Location = System::Drawing::Point(151, 158);
                        this->btnAxisXplus->Name = L"btnAxisXplus";
                        this->btnAxisXplus->Size = System::Drawing::Size(21, 47);
                        this->btnAxisXplus->TabIndex = 7;
                        this->toolTip1->SetToolTip(this->btnAxisXplus, L"Avanza un paso en eje X");
                        this->btnAxisXplus->UseVisualStyleBackColor = false;
                        this->btnAxisXplus->Click += gcnew System::EventHandler(this,
&Form1::btnAxisXplus_Click);
                        //
                        // btnAxisYplus
                        //
                        this->btnAxisYplus->BackColor = System::Drawing::Color::Firebrick;
                        this->btnAxisYplus->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Stretch;
                        this->btnAxisYplus->Cursor = System::Windows::Forms::Cursors::Hand;
                        this->btnAxisYplus->FlatAppearance->BorderSize = 0;
                        this->btnAxisYplus->Font = (gcnew System::Drawing::Font(L"Segoe Print", 15.75F,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                                static_cast<System::Byte>(0)));
                        this->btnAxisYplus->Location = System::Drawing::Point(94, 131);
                        this->btnAxisYplus->Name = L"btnAxisYplus";
                        this->btnAxisYplus->Size = System::Drawing::Size(51, 21);
                        this->btnAxisYplus->TabIndex = 8;
                        this->toolTip1->SetToolTip(this->btnAxisYplus, L"Avanza un paso en eje Y");
                        this->btnAxisYplus->UseVisualStyleBackColor = false;
                        this->btnAxisYplus->Click += gcnew System::EventHandler(this,
&Form1::btnAxisYplus_Click);
                        //
                        // btnAxisYminus
                        //
                        this->btnAxisYminus->BackColor = System::Drawing::Color::Firebrick;
                        this->btnAxisYminus->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Stretch;
                        this->btnAxisYminus->Cursor = System::Windows::Forms::Cursors::Hand;
                        this->btnAxisYminus->Font = (gcnew System::Drawing::Font(L"Segoe Print", 15.75F,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                                static_cast<System::Byte>(0)));
                        this->btnAxisYminus->Location = System::Drawing::Point(94, 211);
                        this->btnAxisYminus->Name = L"btnAxisYminus";
                        this->btnAxisYminus->Size = System::Drawing::Size(51, 20);
                        this->btnAxisYminus->TabIndex = 9;
                        this->toolTip1->SetToolTip(this->btnAxisYminus, L"Retrocede un paso en eje Y");
                        this->btnAxisYminus->UseVisualStyleBackColor = false;
                        this->btnAxisYminus->Click += gcnew System::EventHandler(this,
&Form1::btnAxisYminus_Click);
                        //
                        // btnReset
                        //
                        this->btnReset->Cursor = System::Windows::Forms::Cursors::Hand;
                        this->btnReset->Location = System::Drawing::Point(12, 358);
                        this->btnReset->Name = L"btnReset";
                        this->btnReset->Size = System::Drawing::Size(87, 63);
                        this->btnReset->TabIndex = 10;
                        this->btnReset->Text = L"Reiniciar";
                        this->toolTip1->SetToolTip(this->btnReset, L"Volver carro a posición inicial");
```

```
                            this->btnReset->UseVisualStyleBackColor = true;
                            this->btnReset->Click += gcnew System::EventHandler(this, &Form1::btnReset_Click);
                            //
                            // btnOrigin
                            //
                            this->btnOrigin->BackgroundImage = (cli::safe_cast<System::Drawing::Image^
>(resources->GetObject(L"btnOrigin.BackgroundImage")));
                            this->btnOrigin->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Stretch;
                            this->btnOrigin->Cursor = System::Windows::Forms::Cursors::Hand;
                            this->btnOrigin->Location = System::Drawing::Point(94, 158);
                            this->btnOrigin->Name = L"btnOrigin";
                            this->btnOrigin->Size = System::Drawing::Size(51, 47);
                            this->btnOrigin->TabIndex = 11;
                            this->toolTip1->SetToolTip(this->btnOrigin, L"Establecer origen");
                            this->btnOrigin->UseVisualStyleBackColor = true;
                            this->btnOrigin->Click += gcnew System::EventHandler(this,
&Form1::btnOrigin_Click);
                            //
                            // btnAxisXplusfast
                            //
                            this->btnAxisXplusfast->BackgroundImage = (cli::safe_cast<System::Drawing::Image^
>(resources->GetObject(L"btnAxisXplusfast.BackgroundImage")));
                            this->btnAxisXplusfast->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Stretch;
                            this->btnAxisXplusfast->Location = System::Drawing::Point(178, 158);
                            this->btnAxisXplusfast->Name = L"btnAxisXplusfast";
                            this->btnAxisXplusfast->Size = System::Drawing::Size(51, 47);
                            this->btnAxisXplusfast->TabIndex = 12;
                            this->toolTip1->SetToolTip(this->btnAxisXplusfast, L"Avanza automaticamente en eje
X");
                            this->btnAxisXplusfast->UseVisualStyleBackColor = true;
                            this->btnAxisXplusfast->Click += gcnew System::EventHandler(this,
&Form1::btnAxisXplusfast_Click);
                            //
                            // btnAxisXminusfast
                            //
                            this->btnAxisXminusfast->BackgroundImage = (cli::safe_cast<System::Drawing::Image^
>(resources->GetObject(L"btnAxisXminusfast.BackgroundImage")));
                            this->btnAxisXminusfast->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Stretch;
                            this->btnAxisXminusfast->Location = System::Drawing::Point(10, 158);
                            this->btnAxisXminusfast->Name = L"btnAxisXminusfast";
                            this->btnAxisXminusfast->Size = System::Drawing::Size(51, 47);
                            this->btnAxisXminusfast->TabIndex = 13;
                            this->toolTip1->SetToolTip(this->btnAxisXminusfast, L"Retrocede automaticamente en
eje X");
                            this->btnAxisXminusfast->UseVisualStyleBackColor = true;
                            this->btnAxisXminusfast->Click += gcnew System::EventHandler(this,
&Form1::btnAxisXminusfast_Click);
                            //
                            // btnAxisYplusfast
                            //
                            this->btnAxisYplusfast->BackgroundImage = (cli::safe_cast<System::Drawing::Image^
>(resources->GetObject(L"btnAxisYplusfast.BackgroundImage")));
                            this->btnAxisYplusfast->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Stretch;
                            this->btnAxisYplusfast->Location = System::Drawing::Point(94, 78);
                            this->btnAxisYplusfast->Name = L"btnAxisYplusfast";
                            this->btnAxisYplusfast->Size = System::Drawing::Size(51, 47);
                            this->btnAxisYplusfast->TabIndex = 14;
                            this->toolTip1->SetToolTip(this->btnAxisYplusfast, L"Avanza automaticamente en eje
Y");
                            this->btnAxisYplusfast->UseVisualStyleBackColor = true;
                            this->btnAxisYplusfast->Click += gcnew System::EventHandler(this,
&Form1::btnAxisYplusfast_Click);
                            //
                            // btnAxisYminusfast
                            //
                            this->btnAxisYminusfast->BackgroundImage = (cli::safe_cast<System::Drawing::Image^
>(resources->GetObject(L"btnAxisYminusfast.BackgroundImage")));
                            this->btnAxisYminusfast->BackgroundImageLayout =
System::Windows::Forms::ImageLayout::Stretch;
                            this->btnAxisYminusfast->Location = System::Drawing::Point(94, 237);
                            this->btnAxisYminusfast->Name = L"btnAxisYminusfast";
                            this->btnAxisYminusfast->Size = System::Drawing::Size(51, 47);
                            this->btnAxisYminusfast->TabIndex = 15;
                            this->toolTip1->SetToolTip(this->btnAxisYminusfast, L"Retrocede automaticamente en
eje Y");
                            this->btnAxisYminusfast->UseVisualStyleBackColor = true;
```

```
                        this->btnAxisYminusfast->Click += gcnew System::EventHandler(this,
&Form1::btnAxisYminusfast_Click);
                        //
                        // Form1
                        //
                        this->AutoScaleDimensions = System::Drawing::SizeF(7, 19);
                        this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
                        this->BackColor = System::Drawing::SystemColors::Control;
                        this->BackgroundImageLayout = System::Windows::Forms::ImageLayout::Stretch;
                        this->ClientSize = System::Drawing::Size(549, 480);
                        this->Controls->Add(this->btnAxisYminusfast);
                        this->Controls->Add(this->btnAxisYplusfast);
                        this->Controls->Add(this->btnAxisXminusfast);
                        this->Controls->Add(this->btnAxisXplusfast);
                        this->Controls->Add(this->btnOrigin);
                        this->Controls->Add(this->btnReset);
                        this->Controls->Add(this->btnAxisYminus);
                        this->Controls->Add(this->btnAxisYplus);
                        this->Controls->Add(this->btnAxisXplus);
                        this->Controls->Add(this->btnAxisXminus);
                        this->Controls->Add(this->btnAxisZminus);
                        this->Controls->Add(this->btnAxisZplus);
                        this->Controls->Add(this->COMStatus_lbl);
                        this->Controls->Add(this->btnSendData);
                        this->Controls->Add(this->ProgressBarDrill);
                        this->Controls->Add(this->txtSendData);
                        this->Cursor = System::Windows::Forms::Cursors::Default;
                        this->Font = (gcnew System::Drawing::Font(L"Segoe Print", 8.25F,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
                                static_cast<System::Byte>(0)));
                        this->FormBorderStyle = System::Windows::Forms::FormBorderStyle::FixedDialog;
                        this->MaximizeBox = false;
                        this->Name = L"Form1";
                        this->Opacity = 0.95;
                        this->StartPosition = System::Windows::Forms::FormStartPosition::CenterScreen;
                        this->Text = L"Perforadora Automática";
                        this->ResumeLayout(false);
                        this->PerformLayout();

                }
#pragma endregion

        //-------------------------------------------------------------------------------------//
        //Funcion:
        //                System::Void timer1_Tick(System::Object^  sender, System::EventArgs^  e)
        //
        //                Esta funcion periodicamente verifica si un dispositivo USB CDC se conecto al
        //                puerto COM virtual. Si se conectó abre el puerto COM correspondiente, sino lo
cierra.
        //
        //Parametros:
        //                object Sender            -Sender del evento (este formulario)
        //                EventArgs e                      -Los argumentos del evento
        //
        //Return Values:
        //                Ninguno
        //
        //
        private: System::Void timer1_Tick(System::Object^  sender, System::EventArgs^  e)
                         {
                                 String^ COMNumberString="";      //Se define un string para almacenar el
nombre
                                                                                                 //del
puerto COM
                                 //Se llama a la siguiente funcion para que devuelva el nombre del puerto
COMXX
                                 //(por ejemplo COM5)
                                 COMNumberString=USB_CheckIfPresentAndGetCOMString(
                                         USB_VENDOR_ID,USB_PRODUCT_ID,USB_DEVICE_INSTANCE,
                                         USB_IS_DEVICE_COMPOSITE,USB_COMPOSITE_INTERFACE_INDEX);
                                 //Si no devuelve nada es porque no esta conectado
                                 if(COMNumberString=="")
                                 {
                                         DeviceAttachedState=FALSE;
                                         COMStatus_lbl->Text="Estado: No Conectado";
                                 }
                                 //Sino esta conectado
                                 else
                                 {
                                         DeviceAttachedState=TRUE;
```

```
                                }
                                //Verificar si pasa de "No conectado" a "Conectado"
                                //Si: no estaba conectado y ahora si...

if((OldDeviceAttachedState!=DeviceAttachedState)&&(DeviceAttachedState==TRUE))
                                {
                                        //Abrir el puerto con el nombre correspondiente
                                        //Se utiliza try-catch porque si existe una excepcion no se
                                        //produce ningun error en el programa.
                                        try
                                        {
                                                serialPort1->PortName=COMNumberString;
                                                serialPort1->Open();
                                                btnSendData->Enabled=TRUE;
                                                COMStatus_lbl->Text="Estado: Conectado como
"+COMNumberString;
                                        }
                                        catch(...)
                                        {
                                        //Cambiar los valores de los estados del puerto
                                        OldDeviceAttachedState=TRUE;
                                        DeviceAttachedState=FALSE;
                                        try
                                                {
                                                serialPort1->Close();
                                                }
                                        catch(...)
                                                {
                                                }
                                        COMStatus_lbl->Text="Estado: Conectado pero sin comunicacion.
Por favor desconecte, espere unos segundos y reintente.";
                                        }
                                }
                                //Verifica si pasa del estado "Conectado" a "No conectado"
                                //Si estaba conectado y ahora ya no...

if((OldDeviceAttachedState!=DeviceAttachedState)&&(DeviceAttachedState==FALSE))
                                {
                                        //Se desactiva el boton de envio, se limpian los buffers de del
puerto
                                        // y se cierra el puerto.
                                        btnSendData->Enabled=FALSE;
                                        try
                                        {
                                                serialPort1->DiscardInBuffer();
                                                serialPort1->DiscardOutBuffer();
                                        }
                                        catch(...){}
                                        try
                                        {
                                        serialPort1->Close();
                                        }
                                        catch(...){}
                                }
                                //Se actualiza el valor del estado del puerto
                                OldDeviceAttachedState=DeviceAttachedState;
                        }
        //------------------------------------------------------------------------------//
        //Funcion:
        //              String^  USB_CheckIfPresentAndGetCOMString(WORD USBVendorID,
        //              WORD USBProductID, DWORD deviceInstanceIndex, bool deviceIsComposite,
        //              BYTE compositeInterfaceIndex)
        //
        //              Esta funcion verifica si se encuentra un dispositivo USB CDC actualmente conectado
        //              y que coincida con el PID e VID correspondiente. Si hay coincidencia, devuelve el
nombre
        //              del puerto COM  virtual correspendiente.
        //Parametros:
        //              WORD USBVendorID: Numero de 16 bits a buscar (VID)
        //              WORD USBProductID: Numero de 16 bits a buscar (PID)
        //              DWORD DeviceInstanceIndex: Siempre '0', a menos que existan varios dispositivos
conectados
        //              al mismo tiempo
        //              bool deviceIsComposite: 'False' en este caso, ya que no es un dispositivo
compuesto. Sino, 'True'
        //              BYTE compositeInterfaceIndex: Irrelevante cuando deviceIsComposite es False.
        //Return Values:
        //              String^ que contiene el numero de puerto "COMXX". Si no existte coincidencia
devuelve "".
```

```
        String^ USB_CheckIfPresentAndGetCOMString(WORD USBVendorID, WORD USBProductID, DWORD
deviceInstanceIndex, bool deviceIsComposite, BYTE compositeInterfaceIndex)
{
                //Identificador unico global (GUID) para dispositivos de puerto COM.
                //Windows usa GUIDs para identificar cosas.  GUIDs son estructuras que contiene un
                //numero de 128-bits .
                GUID InterfaceClassGuid = {0xa5dcbf10, 0x6530, 0x11d2, 0x90, 0x1f, 0x00, 0xc0, 0x4f, 0xb9,
0x51, 0xed};
                //GUID_DEVINTERFACE_USB_DEVICE (comun para cualquier dispositivo USB) - utilizado para
encontrar el dispositivo USB
                DEVPROPKEY devPropKeyContainerID = {{0x8c7ed206, 0x3f8a, 0x4827, 0xb3, 0xab, 0xae, 0x9e,
0x1f, 0xae, 0xfc, 0x6c}, 2};
                //necesario para llamar la funcion SetupDiGetDeviceProperty() en dispositivos compuestos
solamente.

                HDEVINFO DeviceInfoTable = INVALID_HANDLE_VALUE;
                PSP_DEVICE_INTERFACE_DATA InterfaceDataStructure = new SP_DEVICE_INTERFACE_DATA;
                PSP_DEVICE_INTERFACE_DETAIL_DATA DetailedInterfaceDataStructure = new
SP_DEVICE_INTERFACE_DETAIL_DATA;
                SP_DEVINFO_DATA DevInfoData;
                DWORD InterfaceIndex = 0;
                DWORD StatusLastError = 0;
                DWORD dwRegType;
                DWORD dwRegSize;
                DWORD StructureSize = 0;
                PBYTE PropertyValueBuffer;
                bool MatchFound = false;
                DWORD ErrorStatus;
                BOOL BoolStatus = FALSE;
                DWORD LoopCounter = 0;
                String^ deviceIDToFind;
                String^ completeIdentifier;
                String^ FriendlyNameString = "";
                int IndexOfStartofCOMChars;
                int IndexOfEndOfCOMChar = 0;
                String^ COMString = "";
//              String^ registryPathString;
//              RegistryKey^ regKey;
//              RegistryKey^ regSubKey;
//              DWORD i;
                DWORD instanceCounter = 0;
//              String^ containerIDToCheck;
//              String^ parentDeviceGUIDString;
//              DEVPROPTYPE devPropType;
//              GUID containerIDGUID;


                //Se crea una cadena de caracteres llamada "HardwareID" .
                //El HardwareID tiene la siguiente forma: "Vid_04d8&Pid_000a".
                //Si el dispositivo es parte de un disopsitivo compuesto,
                //se le agrega ademas "&MI_xx" al final, donde "MI" significa "multiple interface" y "xx"
is un numero 00, 01, 02, etc.,
                //que representa el indice de interfaz
                deviceIDToFind = "Vid_" + USBVendorID.ToString("x4") + "&Pid_" +
USBProductID.ToString("x4");      //The "x4" format strings make the 16-bit numbers into hexadecimal (0-9 +
A-F) text strings.
                if(deviceIsCompositeIdentifier == true)
                {
                        completeIdentifier = deviceIDToFind + "&MI_" +
compositeInterfaceIndex.ToString("d2");   //Append &MI_xx to end of the string (ex:
"Vid_04d8&Pid_0057&MI_01").
                }
                else
                {
                        completeIdentifier = deviceIDToFind;
                }
                //convierte el string a minusculas, para mejor comparacion
                deviceIDToFind = deviceIDToFind->ToLowerInvariant();

                //Ahora se hace una lista de todos los dispositivos conectados(especificados por
"DIGCF_PRESENT"),
                //y especificados por el GUID de interfaz, es decir, dispositivos USB.
```

```
                DeviceInfoTable = INVALID_HANDLE_VALUE;
                DeviceInfoTable = SetupDiGetClassDevsUM(&InterfaceClassGuid, NULL, NULL, DIGCF_PRESENT |
DIGCF_DEVICEINTERFACE);   //Creates a device list in memory, kind of like looking at a cut down internal
version of the Windows device manager.
                ErrorStatus = GetLastError();
                if(DeviceInfoTable == INVALID_HANDLE_VALUE)
                {
                        //Fallo al obtener la lista.  No se puede hacer mas  nada. Se sale del programa
                        delete InterfaceDataStructure;
                        delete DetailedInterfaceDataStructure;
                        return "";
                }


                //Ahora se recorre la lista creada anteriormente.
                //Se verifica si alguno de los dispositos USB conectados coincide con nuestro VID y PID.
                while(true)
                {
                        InterfaceDataStructure->cbSize = sizeof(SP_DEVICE_INTERFACE_DATA);
                        if(SetupDiEnumDeviceInterfacesUM(DeviceInfoTable, NULL, &InterfaceClassGuid,
InterfaceIndex, InterfaceDataStructure))
                        {
                                ErrorStatus = GetLastError();
                                //Si se llego al final de la lista...
                                if(ErrorStatus == ERROR_NO_MORE_ITEMS)
                                {       //No se encontro el dispositivo.  No debe estar conectado.
                                        //Se borran los registros que ya no se necesitan.
                                        SetupDiDestroyDeviceInfoListUM(DeviceInfoTable);
                                        delete InterfaceDataStructure;
                                        delete DetailedInterfaceDataStructure;
                                        return "";
                                }
                        }
                        else    //Sino ocurrió algun tipo de error
                        {
                                ErrorStatus = GetLastError();
                                //Se borran los registros que ya no se necesitan.
                                SetupDiDestroyDeviceInfoListUM(DeviceInfoTable);
                                delete InterfaceDataStructure;
                                delete DetailedInterfaceDataStructure;
                                return "";
                        }

                        //Ahora se extrae el Hardware ID del registro.  El hardware ID contiene el VID y
PID,
                        //el cual luego se verifica si coincide con nuestro dispositivo o no.

                        //Se inicializa una estructura de datos del tipo SP_DEVINFO_DATA .
                        //Se necesita esta estructura para la funcion SetupDiGetDeviceRegistryProperty().
                        DevInfoData.cbSize = sizeof(SP_DEVINFO_DATA);
                        SetupDiEnumDeviceInfoUM(DeviceInfoTable, InterfaceIndex, &DevInfoData);

                        //Primero se consulta por el tamaño del hardware ID,
                        //asi se sabe el tamaño que deberá tener el buffer que almacenará la estructura.
                        SetupDiGetDeviceRegistryPropertyUM(DeviceInfoTable, &DevInfoData,
SPDRP_HARDWAREID, &dwRegType, NULL, 0, &dwRegSize);

                        //Se asigna un buffer para el hardware ID.
                        PropertyValueBuffer = (BYTE *) malloc (dwRegSize);
                        //Si es nulo, error. No se puede alojar en memoria.
                        if(PropertyValueBuffer == NULL)
                        {       //No se puede hacer mas nada, se sale.
                                //Se borran los registros que ya no se necesitan.
                                SetupDiDestroyDeviceInfoListUM(DeviceInfoTable);
                                delete InterfaceDataStructure;
                                delete DetailedInterfaceDataStructure;
                                return "";
                        }

                        //Se extrae el Hardware ID para el dispositivo actual al que estamos analizando.
PropertyValueBuffer se llena con un
                        //REG_MULTI_SZ (un arreglo de string que terminan con caracter nulo).  Para
encontrar al dispositivo,
```

```
                        //solo interesa el primer string en el buffer, que es el "device ID".
                        //El device ID es un string que contiene el VID y PID, por ejemplo con la
siguiente forma:
                        //"Vid_04d8&Pid_000a".
                        SetupDiGetDeviceRegistryPropertyUM(DeviceInfoTable, &DevInfoData,
SPDRP_HARDWAREID, &dwRegType, PropertyValueBuffer, dwRegSize, NULL);

                        //Ahora verificamos si el primer string del Hardware ID (DeviceID) coincide con el
de
                        //nuestro dispositivo USB.
                        #ifdef UNICODE
                        String^ DeviceIDFromRegistry = gcnew String((wchar_t *)PropertyValueBuffer);
                        #else
                        String^ DeviceIDFromRegistry = gcnew String((char *)PropertyValueBuffer);
                        #endif

                        free(PropertyValueBuffer);              //Ya no se necesita PropertyValueBufer,
se libera la memoria

                        //Se convierte en minusculas, para mejor comparacion.
                        DeviceIDFromRegistry = DeviceIDFromRegistry->ToLowerInvariant();
                        //Ahora se cheque si el HardwareID que se esta analizando contiene el correcto
VID/PID
                        MatchFound = DeviceIDFromRegistry->Contains(deviceIDToFind);
                        if(MatchFound == true)
                        {
                                //Se encontro un dispositivo que coincide con nuestro VID/PID.

                                //ahora verificamos si la instancia coincide con la instancia deseada del
usuariuo.
                                //Sino seguimos buscando hasta que se encuentre VID+PID+Instance Index.
                                //Como se definio que InstanceIndex=0, siempre va a coincidir en el primer
ciclo
                                if(instanceCounter == deviceInstanceIndex)
                                {
                                        //Se encontro el dispositivo.  Ahora se debe leer el
"FriendlyName".  El FriendlyName es un texto
                                        //que aparece en una de las entradas en el administrador de
dispositivos.  El friendlyname para el puerto COM virtual
                                        //de un texto del archivo .inf utilizado para instalar el
dispositivo, combinado con el numero de puerto COMX
                                        //asignado al dispositivo.  El numero de COMX tambien aparece en
la entrada del registro "PortName" del dispositivo.
                                        //Primero se se llama la funcion para determinar el tamaño
requerido para el buffer.
                                        SetupDiGetDeviceRegistryPropertyUM(DeviceInfoTable, &DevInfoData,
SPDRP_FRIENDLYNAME, &dwRegType, NULL, 0, &dwRegSize);
                                        //Se asigna un buffer para el hardware ID.
                                        PropertyValueBuffer = (BYTE *) malloc (dwRegSize);
                                        //No se puede hacer mas nada, se sale.
                                        //Se borran los registros que ya no se necesitan.
                                        if(PropertyValueBuffer == NULL)
                                        {
                                                SetupDiDestroyDeviceInfoListUM(DeviceInfoTable);
                                                delete InterfaceDataStructure;
                                                delete DetailedInterfaceDataStructure;
                                                return "";
                                        }

                                        //Se intenta recuperar el string FriendlyName que contiene el
numero de COMx.
                                        SetupDiGetDeviceRegistryPropertyUM(DeviceInfoTable, &DevInfoData,
SPDRP_FRIENDLYNAME, &dwRegType, PropertyValueBuffer, dwRegSize, NULL);

                                        //Se copia el string devuelto por la API en otro de tipo
administrado
                                        //para mejor comparacion.
                                        #ifdef UNICODE
                                                FriendlyNameString = gcnew String((wchar_t
*)PropertyValueBuffer);
                                        #else
                                                FriendlyNameString = gcnew String((char
*)PropertyValueBuffer);
```

```
                                        #endif
                                        free(PropertyValueBuffer);              //No se necesita mas
este buffer, se borra.


                                        //Se verifica si contiene el string COMx esperado y si existe, se
extrae y se dvuelve
                                        //para poder luego abrir el puerto COMx.
                                        if((FriendlyNameString->Contains("(COM")) && (FriendlyNameString-
>Contains(")")))
                                        {
                                                IndexOfStartofCOMChars = FriendlyNameString-
>IndexOf("COM");
                                                IndexOfEndOfCOMChar = FriendlyNameString->IndexOf(")");
                                                if((IndexOfEndOfCOMChar > IndexOfStartofCOMChars) &&
(IndexOfEndOfCOMChar <= FriendlyNameString->Length))
                                                {
                                                        //Se copia el valor del string "COMxx" desde la
variable FriendlyNameString a COMString.
                                                        //El string COMString tiene exactamente el
formato que se necesita para asignarle el
                                                        //numero de puerto COM.
                                                        COMString = FriendlyNameString-
>Substring(IndexOfStartofCOMChars, IndexOfEndOfCOMChar - IndexOfStartofCOMChars);
                                                }

                                                //Aqui posiblemente se pudo haber encontrado el numero
del puerto COM y se debe regresar.
                                                SetupDiDestroyDeviceInfoListUM(DeviceInfoTable);   //Se
borran los registros que ya no se necesitan.
                                                delete InterfaceDataStructure;
                                                delete DetailedInterfaceDataStructure;
                                                return COMString;         //Puede contener"" sino se pudo
encontrar el numero del puerto COM.
                                                                                        //De lo
contrario devolverá "COMxxx", donde xxx es el numero del puerto serie.
                                        }

                                }//if(instanceCounter == deviceInstanceIndex)
                                else
                                {
                                        //Aunque se encontró el dispositivo con el PID/VID, no se ha
podido abrir el puerto COM.
                                        //Puede deberse a que esta conectado en una instancia superior.
                                        instanceCounter++;
                                }
                        }//if(MatchFound == true)

                        //Si se llegó hasta aquí significa que se ha analizado un dispositivo de la lista
del administrador de dispositivos de Windows
                        //Ya que no se encontró el dispositivo con el VID/PID correcto, se pasa al
siguiente elemento de la lista.
                        InterfaceIndex++;

                        //Se sigue analizando la lista hasta que se encuentre el dspositivo con el VID/PID
o hasta que ya no queden elementos en la lista.
                        //Sin embargo, en caso de que ocurra algun error inesperado, se hace un
seguimiento del numero ciclos ejecutados.
                        //Si el numero de ciclos excede un numero muy grande, se sale del mismo para
prevenir un ciclo infinito.
                        LoopCounter++;
                        if(LoopCounter == 10000000)       //Diez millones de ciclos es mas que suficiente,
ya que en ninguna PC puede haber
                                                                                //esa cantidad de
dispositivos conectados.
                        {
                                SetupDiDestroyDeviceInfoListUM(DeviceInfoTable);   //Se borran los
registros que ya no se necesitan.
                                delete InterfaceDataStructure;
                                delete DetailedInterfaceDataStructure;
                                return "";
                        }
                }//end of while(true)
```

```
}

////////////////////////////////////////////////

private: System::Void btnSendData_Click(System::Object^  sender, System::EventArgs^  e)
                 {
                         try
                         {
                                 EnviarlineasdeTexto();
                         }
                         catch(...){}

                 }
private: System::Void btnAxisZplus_Click(System::Object^  sender, System::EventArgs^  e)
                 {
                         try
                         {
                                 serialPort1->Write("u");
                         }
                         catch(...){}

                 }
private: System::Void btnAxisZminus_Click(System::Object^  sender, System::EventArgs^  e)
                 {
                         try
                         {
                                 serialPort1->Write("d");
                         }
                         catch(...){}
                 }
private: System::Void btnAxisXminus_Click(System::Object^  sender, System::EventArgs^  e)
                 {
                          try
                         {
                                 serialPort1->Write("f");
                         }
                         catch(...){}

                 }
private: System::Void btnAxisXplus_Click(System::Object^  sender, System::EventArgs^  e)
                 {
                        try
                         {
                                 serialPort1->Write("b");
                         }
                         catch(...){}

                 }
private: System::Void btnAxisYplus_Click(System::Object^  sender, System::EventArgs^  e)
                 {
                        try
                         {
                                 serialPort1->Write("r");
                         }
                         catch(...){}
                 }
private: System::Void btnAxisYminus_Click(System::Object^  sender, System::EventArgs^  e)
                 {
                         try
                         {
                                 serialPort1->Write("l");
                         }
                         catch(...){}
                 }
private: System::Void btnOrigin_Click(System::Object^  sender, System::EventArgs^  e)
                 {
                         try
                         {
                                 serialPort1->Write("X000000Y000000");
                         }
                         catch(...){}
```

```
                }
private: System::Void btnReset_Click(System::Object^  sender, System::EventArgs^  e)
                {
                        txtSendData->Clear();
                        ProgressBarDrill->Value=0;
                        try
                        {
                                serialPort1->Write("o");
                        }
                        catch(...){}
                }
private: System::Void btnAxisXplusfast_Click(System::Object^  sender, System::EventArgs^  e)
                {
                        try
                        {
                                serialPort1->Write("B");
                        }
                        catch(...){}
                }
private: System::Void btnAxisXminusfast_Click(System::Object^  sender, System::EventArgs^  e)
                {
                       try
                        {
                                serialPort1->Write("F");
                        }
                        catch(...){}
                }
private: System::Void btnAxisYplusfast_Click(System::Object^  sender, System::EventArgs^  e)
                {
                        try
                        {
                                serialPort1->Write("R");
                        }
                        catch(...){}
                }
private: System::Void btnAxisYminusfast_Click(System::Object^  sender, System::EventArgs^  e)
                {
                        try
                        {
                                serialPort1->Write("L");
                        }
                        catch(...){}
                }
};
}
```