

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO



Facultad de Ingeniería



**INSTRUMENTACIÓN Y CONTROL PARA
LA OPERACIÓN REMOTA
DE LA PRÁCTICA
“TIRO PARABÓLICO Y CAÍDA LIBRE”**

TESIS

Que para obtener el título de

INGENIERO MECATRÓNICO

Presenta

BRUNO SENZIO-SAVINO BARZELLATO

Director de tesis

M. EN I. YUKIHIRO MINAMI KOYAMA

México, Distrito Federal a 1 de junio de 2011

RESUMEN

Ante el creciente cambio y modernización de la tecnología así como la creciente necesidad de controlar diferentes sistemas electrónicos de manera automática y remota, o teleoperación, se requiere innovar en los métodos de enseñanza.

Atendiendo a estas necesidades, se ha constituido el proyecto EN106204 del PAPIME con el nombre de “Creación de un laboratorio remoto accedido por medio de la Internet para la asignatura de Cinemática y Dinámica” cuyos objetivos se centran en diseño y construcción de experimentos capaces de ser controlados por vía remota, específicamente a través de la Internet, mejorando la calidad del aprendizaje y minimizando los costos de instrumentación, adecuando el equipamiento e infraestructura de los laboratorios de Cinemática y Dinámica de la Facultad de Ingeniería de la Universidad Nacional Autónoma de México.

En esta tesis se retoma el trabajo previo realizado para la práctica de “Tiro parabólico y caída libre”, a través del diseño e implementación de los instrumentos mecánicos y electrónicos de control, tomando en cuenta las diferentes alternativas consideradas para su construcción, destacando su funcionamiento así como sus principales ventajas y desventajas.

Además, se presenta la creación de una aplicación Web con arquitectura cliente-servidor capaz de transmitir video, de manera que pueda ponerse en marcha el sistema para ser accedido vía Internet. Dicha aplicación se comunica al sistema mediante el protocolo USB.

Finalmente, se resaltan los resultados obtenidos así como las recomendaciones para el uso efectivo del sistema.

CONTENIDO

RESUMEN

CAPÍTULO 1. INTRODUCCIÓN

1.1 DESCRIPCIÓN DEL PROBLEMA

1.2 JUSTIFICACIÓN

1.3 OBJETIVOS

1.4 ANTECEDENTES

1.4.1 DESCRIPCIÓN TEÓRICA DEL SISTEMA

1.4.2 DESCRIPCIÓN DEL SUBSISTEMA DE TIRO (CAÑÓN)

1.4.3 DESCRIPCIÓN DEL SUBSISTEMA DEL SOLTADOR

1.4.4 DESCRIPCIÓN DEL SUBSISTEMA DE ELEVACIÓN Y REPARTICIÓN

1.5 NECESIDADES PARA LA INSTRUMENTACIÓN Y EL CONTROL

1.5.1 SUBSISTEMA DE TIRO

1.5.2 SUBSISTEMA DEL SOLTADOR

1.5.3 SUBSISTEMA DE ELEVACIÓN Y REPARTICIÓN

1.5.4 SISTEMA DE TIRO PARABÓLICO Y CAÍDA LIBRE

CAPÍTULO 2. ANÁLISIS PARA LA INSTRUMENTACIÓN Y EL CONTROL

2.1 CONTROL DE ENTRADAS

2.1.1 SENsores

2.1.1.1 SENsores de CONTACTO

2.1.1.2 SENsores SIN CONTACTO

2.1.2 RESOLUCIÓN ANGULAR

2.1.2.1 ENCODER ROTATORIO

2.1.2.2 ENGRANES

2.1.2.3 POTENCIÓMETRO

2.1.3 TENSIÓN DE LA CUERDA

2.1.3.1 SENSOR DE CORRIENTE

2.1.3.2 CONTEO DE VUELTAS

2.1.4 TRATAMIENTO DE SEÑALES

2.2 CONTROL DE ACTUADORES

2.2.1 TRANSISTORES

2.2.2 RELEVADORES

2.2.3 PUENTE H

2.2.4 CONTROL DE UN MOTOR PASO A PASO

2.3 PROCESAMIENTO DE SEÑALES Y COMUNICACIÓN DEL SISTEMA

- 2.3.1 PROCESAMIENTO DE SEÑALES**
 - 2.3.1.1 MICROCONTROLADORES**
 - 2.3.1.2 MICROPROCESADORES**
 - 2.3.1.3 CONTROLADORES LÓGICOS PROGRAMABLES**

- 2.3.2 COMUNICACIÓN DEL SISTEMA**

- 2.4 ALIMENTACIÓN Y REGULACIÓN DE VOLTAJE**

- 2.4.1 FUENTES DE ALIMENTACIÓN**

- 2.4.2 CIRCUITOS PARA LA REGULACIÓN DE VOLTAJE**

CAPÍTULO 3. DISEÑO DE LA INSTRUMENTACIÓN Y EL CONTROL

- 3.1 SUBSISTEMA DE TIRO**

- 3.1.1 DETECCIÓN DE ENTRADAS Y CONTROL DE ACTUADORES**

- 3.1.2 TENSIÓN DE LA CUERDA**

- 3.1.3 AMPLIFICACIÓN ANGULAR**

- 3.1.4 DETECCIÓN DEL TRINQUETE**

- 3.1.5 DETECCIÓN DE CARGA DEL PROYECTIL**

- 3.2 SUBSISTEMA DEL SOLTADOR**

- 3.2.1 DETECCIÓN DE ENTRADAS Y CONTROL DE ACTUADORES**

- 3.3 SUBSISTEMA DE ELEVACIÓN Y REPARTICIÓN**

- 3.4 SISTEMA DE TIRO PARABÓLICO Y CAÍDA LIBRE**

- 3.4.1 PROCESAMIENTO DE SEÑALES**

- 3.4.2 REGULACIÓN**

CAPÍTULO 4. DISEÑO DE LA INTERFAZ USUARIO-SERVIDOR

- 4.1 COMPATIBILIDAD CON WINDOWS XP**

- 4.2 COMUNICACIÓN DEL SISTEMA**

- 4.3 ARQUITECTURA DE RED DE LA INTERFAZ**

- 4.4 HERRAMIENTAS DE PROGRAMACIÓN**

- 4.5 APLICACIONES PARA LA TRANSMISIÓN DE VIDEO**

CAPÍTULO 5. SELECCIÓN Y ESPECIFICACIÓN

- 5.1 INSTRUMENTACIÓN MECÁNICA DEL SISTEMA**

- 5.1.1 TENSIÓN DE LA CUERDA**

- 5.1.1.1 CAMBIO DE CUERDA**

- 5.1.1.2 CONTEO DE VUELTAS**

- 5.1.1.3 DETECCIÓN DEL TRINQUETE**

- 5.1.2 AMPLIFICACIÓN ANGULAR**

- 5.1.3 DETECCIÓN DE PELOTA**

- 5.1.4 ROBUSTECIMIENTO DE ENTRADAS LATERALES**

- 5.1.5 BALANCEO DEL CAÑÓN**

5.1.6 MANUFACTURA DE LA UNIDAD DE CONTROL

5.2 CONTROL ELECTRÓNICO DEL SISTEMA

5.2.1 DISPOSICIÓN DE LOS ELEMENTOS DEL SISTEMA

5.2.2 EL MÓDULO EXTERNO DEL CAÑÓN (MEC)

5.2.3 EL CIRCUITO DE CONTROL

5.2.3.1 TRATAMIENTO DE LA SEÑAL DE ENTRADA

5.2.3.2 SEÑALES DE CONTROL

5.2.4 EL CIRCUITO DE POTENCIA

5.2.5 CABLEADO DEL SISTEMA

5.2.6 ALGORITMO DE CONTROL Y PROGRAMACIÓN DEL MICROCONTROLADOR

5.2.6.1 POSICIÓN DE HOME

5.2.6.2 ALGORITMO DE CONTROL

5.2.6.3 PROGRAMACIÓN DEL MICROCONTROLADOR

5.3 INTERFAZ DE USUARIO-SERVIDOR

5.3.1 IMPLEMENTACIÓN DE LA INTERFAZ

5.3.1.1 APLICACIÓN DE USUARIO

5.3.1.2 APLICACIÓN DE SERVIDOR

5.3.2 CONEXIÓN A INTERNET

5.3.3 TRANSMISIÓN DE VIDEO

5.3.4 MONTAJE DEL SERVIDOR DE VÍDEO

CAPÍTULO 6. RESULTADOS Y CONCLUSIONES

6.1 RESULTADOS DE LAS PRUEBAS

6.2 CONCLUSIONES

6.3 RECOMENDACIONES

APÉNDICE

A1. PLANOS DE CONSTRUCCIÓN

A2. DIAGRAMAS ELECTRÓNICOS ESQUEMÁTICOS

A3. LONGITUD DEL CABLEADO Y CONTEO DE CONECTORES

A4. PROGRAMA DEL MICROCONTROLADOR PIC18F4550

A5. PROGRAMA DE LA INTERFAZ USUARIO-SERVIDOR

BIBLIOGRAFÍA

CAPÍTULO 1 INTRODUCCIÓN

1.1 DESCRIPCIÓN DEL PROBLEMA

La formación del ingeniero a lo largo de su carrera no sería integral y completa si no se apoyara en la educación empírica, visual, que ayude a comprender los teoremas abstractos que se plasman en un pizarrón, atendiendo a los principios de la física y las matemáticas.

Es por tanto, indispensable, el uso de aulas de experimentación o laboratorios donde se corroboren dichos teoremas, aplicando el método científico para estructurar su investigación basada en la observación de la realidad, obteniendo los resultados de su práctica, de manera que se tenga un aprendizaje adicional al de la teoría.

Para la asignatura de Cinemática y Dinámica, impartida en la Facultad de Ingeniería de la Universidad Nacional Autónoma de México, resulta ser importante el uso de un laboratorio en el cual los alumnos puedan analizar de forma interactiva los diferentes fenómenos físicos abordados. No obstante, la impartición de la parte experimental de la misma en un laboratorio tradicional, presenta diferentes inconvenientes[2], entre los que destacan la falta de la infraestructura física y de equipamiento (dado el elevado costo del equipo de instrumentación), la restricción de los espacios debido a la masificación de

la educación en la creciente población de estudiantes de la Facultad (teniéndose por resultado la creación de equipos de trabajo numerosos que puedan disminuir la calidad del aprendizaje), la necesidad de supervisión de los grupos de trabajo así como la necesidad de la presencia física de los estudiantes en cada una de las prácticas.

1.2 JUSTIFICACIÓN

Es debido a los motivos antes citados que se constituyó el proyecto EN106204 del PAPIME (Programa de Apoyo a Proyectos para la Innovación y el Mejoramiento de la Enseñanza) cuyo nombre es “Creación de un laboratorio remoto accedido por medio de la Internet para la asignatura Cinemática y Dinámica”, el cual pretende la construcción de un laboratorio que pueda ser accedido vía Internet, por cualquier estudiante de la asignatura que cuente con acceso a la red, disminuyendo la saturación de grupos, permitiendo un mejor aprendizaje de los conceptos. Cabe destacar que el proyecto contempla el desarrollo de seis prácticas diferentes, cada una con un funcionamiento particular en cuanto a los dispositivos mecánicos y electrónicos que la componen, de manera que puedan ser administrados por un servidor y ser controlados por el usuario de manera remota.

El trabajo de la presente tesis hace referencia a la práctica de “Tiro parabólico y caída libre”, cuya descripción es la siguiente:

“La práctica de tiro parabólico y caída libre consiste en verificar las características cinemáticas de posición, rapidez y aceleración de un objeto que se impulsa con un disparador, el cual es susceptible de ajustarse de tal forma que tenga una inclinación determinada. Al mismo tiempo se coloca un segundo objeto en la línea de mira del disparador, al que se suelta justo cuando el primero abandona al disparador del tiro parabólico y su movimiento será una caída libre (sic). Los dos objetos forzosamente deben colisionar, independientemente de la rapidez que se imprima al tiro parabólico. Luego de efectuar el experimento varias veces y verificar este hecho, se obtienen los modelos matemáticos que relacionan a la posición con el tiempo de ambos objetos, a partir de los cuales se puede verificar el instante y la posición en las que ocurre dicha colisión”. [1]

1.3 OBJETIVOS

El sistema para llevar a cabo la práctica cuenta con un subsistema de tiro (cañón para pelotas de golf), cuyo prototipo fue diseñado en una tesis previa, un soltador encargado de la caída libre de otra pelota de golf, así como un subsistema de elevación encargado de alimentar las pelotas al cañón y al soltador una vez culminado el evento, de tal manera que se repartan al soltador y al sistema de recarga

respectivos alternadamente. Los objetivos del presente trabajo son la instrumentación del subsistema de tiro y del soltador en conjunto con el subsistema de elevación así como la creación de la aplicación con arquitectura cliente-servidor que permite su control de manera remota a través de Internet. La operación de los actuadores del sistema se realiza mediante una etapa de potencia que recibe señales de control mediante un microcontrolador.

En esta tesis se han desarrollado los siguientes puntos:

- ✚ Diseño e implementación de los instrumentos mecánicos para el control electrónico del subsistema de tiro.
- ✚ Diseño e implementación de la interfaz electrónica para el control de sensores y actuadores.
- ✚ Diseño e implementación de la interfaz de usuario-servidor para el control del sistema de manera remota a través de la Internet.

Los alcances específicos fueron:

- ✚ Implementación de los elementos mecánicos necesarios para el control electrónico del subsistema de tiro.
- ✚ Construcción de la interfaz electrónica y del programa para la comunicación del sistema con el servidor.
- ✚ Creación de una aplicación Web del tipo cliente-servidor que permite la comunicación del usuario con el sistema así como la transmisión de vídeo con un retraso mínimo de dos segundos.

Dada la descripción de los alcances de la tesis, cabe mencionar de manera breve los conceptos tratados a lo largo de este documento.

En el primer capítulo se presentan las características del problema, así como el establecimiento de los objetivos y los alcances a lograr, mostrando finalmente los antecedentes tanto teóricos como del sistema de la práctica.

En el segundo capítulo se realiza una revisión de las diferentes alternativas que permiten la resolución de los objetivos, destacando sus propiedades y características, incluyendo su principio de funcionamiento.

En el tercer capítulo se muestran las comparaciones entre las diferentes alternativas para las cuestiones mecánicas y electrónicas, partiendo de la metodología de diseño, y tomando en cuenta sus ventajas y desventajas.

En el cuarto capítulo se describe el diseño y la implementación de la aplicación Web, incluyendo la transmisión de vídeo por medio de una cámara Web. Se analiza las alternativas sobre la arquitectura para la interfaz de usuario.

En el quinto capítulo, se destaca la toma de decisiones así como la construcción de los elementos mecánicos y de la interfaz de control electrónica y su operación mediante la aplicación. Se vincula el desarrollo de los mismos a la interfaz cliente-servidor.

Finalmente, el sexto capítulo muestra los resultados obtenidos a partir de las pruebas realizadas, junto con las recomendaciones para el uso efectivo y modificaciones posteriores del sistema, y las sugerencias para la implementación del sistema de seguridad y configuraciones previas para restringir el acceso al servidor.

Es importante destacar que no se abordaron los temas de diseño y la implementación del sistema de recarga del cañón, de diseño y construcción de un sistema de seguridad en caso de fallo de los sensores *actuales* así como de implementación de un acceso seguro al servidor.

1.4 ANTECEDENTES

1.4.1 Descripción teórica del sistema

Con base en la Figura 1.1 y tomando la ecuación vectorial para el tiro parabólico:

$$\vec{R} = \vec{R}_o + \vec{v}_o t + \frac{1}{2} \vec{a} t^2 \quad (1)$$

Dadas las posiciones del sistema como se muestra en dicha Figura (donde y_1 esa la altura entre la pelota del cañón al suelo y y_2 la altura de la pelota del soltador al suelo, se tienen las siguientes ecuaciones de tiro parabólico que seguirá la primera pelota:

$$R_x = x = v_o t \cos \theta \quad (2)$$

$$R_y = h = y_1 + v_o t \sin \theta - \frac{1}{2} g t^2 \quad (3)$$

Mientras tanto, la ecuación de caída libre para la segunda pelota es la siguiente:

$$h = y_2 - \frac{1}{2}gt^2 \quad (4)$$

Con base en la descripción previa y de la práctica, en la cual, se igualan (3) y (4) y se despeja la variable t , se tiene la siguiente ecuación para garantizar una colisión:

$$\tan \theta = \frac{y_2 - y_1}{x} \quad (5)$$

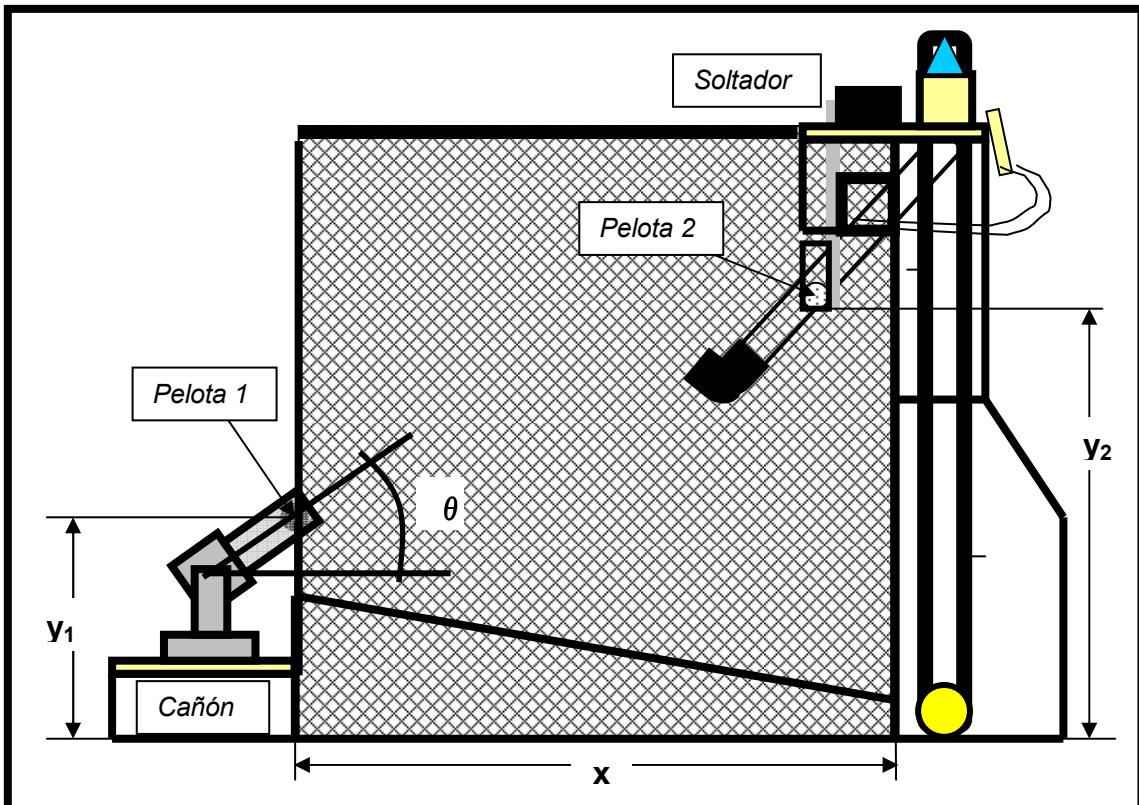


Figura 1.1 Ilustración del sistema.

Resulta por tanto, indispensable, que se pueda observar el soltador desde el cañón, de manera que se ha instalado una cámara Web que permita el posicionamiento de los elementos, a través del control de diferentes mecanismos con los que cuenta cada subsistema. El usuario podrá manipular de manera remota tanto el soltador como el cañón, de la forma que pueda lograr la colisión de las dos pelotas; por lo tanto, deberán estar resueltos los dispositivos que permitan el control. A continuación se describen cada uno de los subsistemas.

1.4.2 Descripción del subsistema de tiro (cañón)

El subsistema de tiro, mostrado en la Figura 1.2, se compone de un cañón, el cual a su vez contiene tres motores de corriente directa que le permiten controlar el movimiento lateral, angular y de tensión de la cuerda así como un solenoide para el disparo.



Figura 1.2 El subsistema de tiro.

Mientras que el motor lateral le permite pasar al cañón entre las posiciones de recarga y de tiro a través del giro de un tornillo sinfín, el motor angular controla el ángulo de disparo y el motor de tensión determina la potencia de tiro por medio de la tensión de una cuerda interna; tan pronto el cañón se encuentre en la posición de tiro, el usuario podrá ajustar estas dos variables para alcanzar el objetivo del experimento, ayudándose de un solenoide(trinquete) que permita el tiro del proyectil, una vez definidas las variables. Los actuadores se muestran en la Figura 1.3.

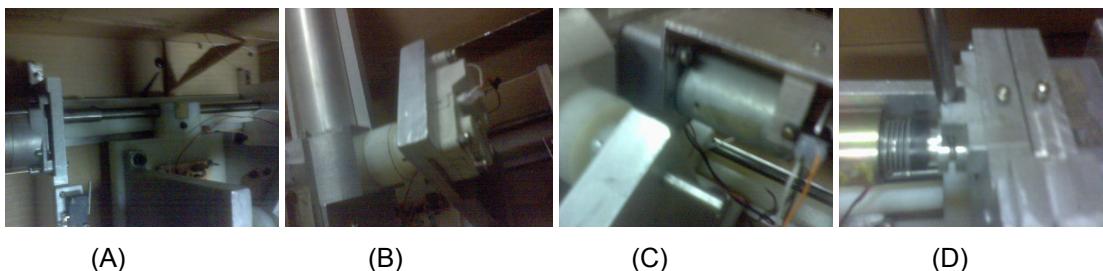


Figura 1.3 Actuadores del subsistema de tiro.

(A. tornillo sinfín, B. motor angular, C. motor de tensión, D. trinquete)

1.4.3 Descripción del subsistema del soltador

El soltador se compone de dos sensores laterales, un motor de pulsos y un solenoide.

El motor de pulsos se encarga de mover con precisión el soltador (con precisión) a su posición de carga(para recibir la pelota proveniente del subsistema de elevación), así como de manipular su movimiento (variación de altura) dentro de los límites laterales o de altura. El mecanismo adaptado al solenoide permite “soltar” la pelota de manera vertical para reproducir el fenómeno de caída libre. Cabe mencionar que cuando el soltador se posiciona para ser cargado, abre una pequeña escotilla que

permite el paso una pelota de golf contenida previamente en el riel correspondiente. En la Figura 1.4 se muestra más a detalle el sistema.

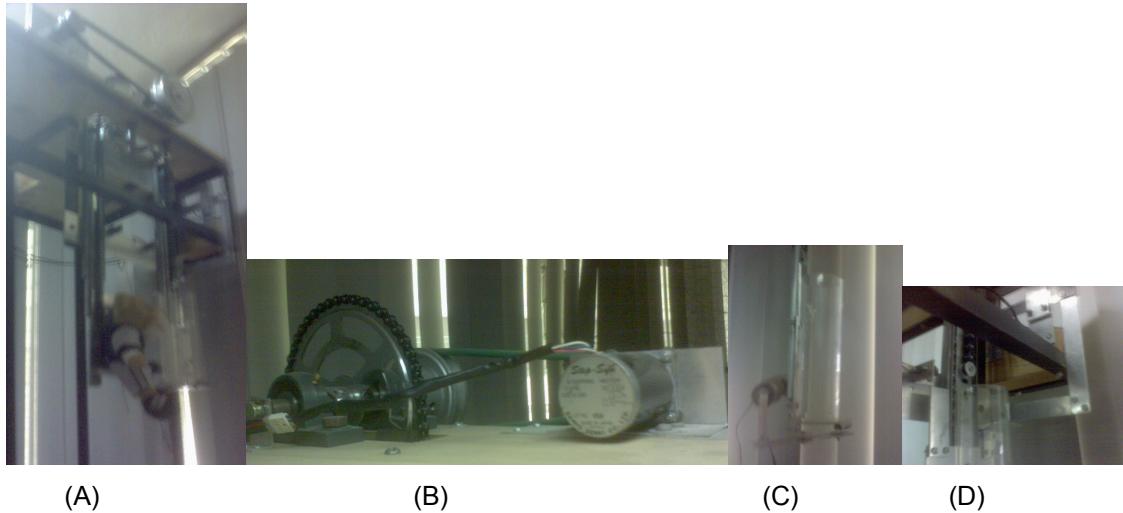


Figura 1.4 El subsistema del soltador.

(A. soltador c/contrapeso, B. motor del soltador, C. soltador, D. escotilla de carga)

1.4.4 Descripción del subsistema de elevación y repartición

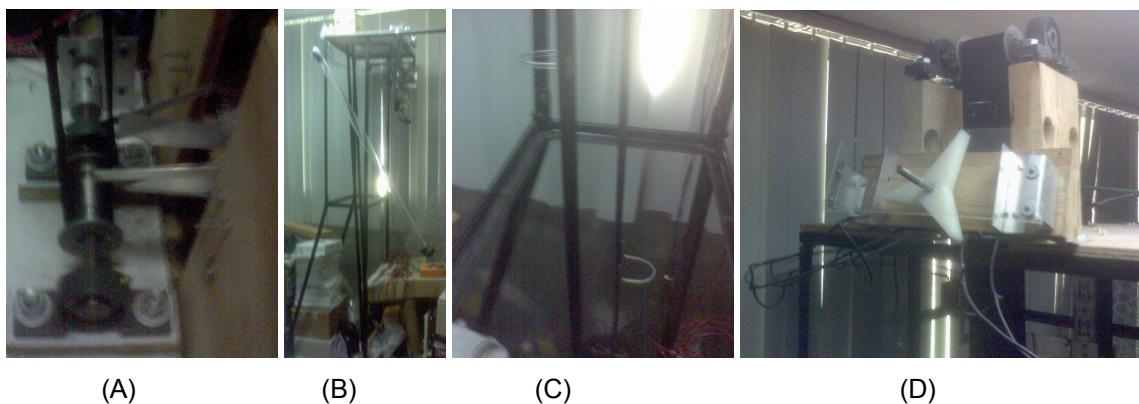


Figura 1.5 El subsistema de elevación y repartición.

(A. motor de la banda y contenedor, B. banda, C. cucharillas, D. estrella binaria y rieles)

Una vez realizado el experimento, es necesario elevar las pelotas de golf contenidas en el plano inclinado, de tal forma que sean repartidas tanto para la recarga del soltador como del cañón. Para llevar esto a cabo, se utiliza una banda operada por un motor, misma que contiene dos “cucharillas” que recogen las pelotas de golf almacenadas al final del plano. Cuando la pelota alcanza el final de la banda, esta caerá a un lado de la estrella binaria, de tal forma que caiga ya sea del lado del riel que la conduce al soltador, del lado del tubo que la llevará al sistema de recarga del cañón.

1.5 NECESIDADES PARA LA INSTRUMENTACIÓN Y EL CONTROL

Para tener un control adecuado de cada uno de los actuadores y conocer el estado en que se encuentra el experimento, es necesario implementar una interfaz que permita integrar a todos los subsistemas así como la construcción de elementos mecánicos para cada sistema en particular. A continuación se enuncian las necesidades particulares por subsistema.

1.5.1 Subsistema de tiro

➊ Tensión de la cuerda

Para que la potencia de tiro sea efectiva en el rango angular establecido ^[1], se deberá activar el motor de tensión hasta un punto determinado, de tal forma que el usuario pueda determinar la potencia de tiro, brindando un mayor reto al momento de realizar el experimento. No obstante, el alcance de ese punto máximo debe de ser soportado por la cuerda (debe resistir lo suficiente como para no romperse).

➋ Detección del trinquete

Cuando comienza el giro del motor encargado de tensar la cuerda, éste hace girar un trinquete basado en el *clutch* de un automóvil. Dicho trinquete gira un determinado número de veces hasta que se encuentra con sus topes mecánicos, y es en ese mismo instante que comienza a tensarse la cuerda. Para tener un control adecuado sobre el valor de la potencia de tiro se requiere conocer el instante en que inició la tensión de la cuerda, es decir, detectar cuando el trinquete comienza su función.

➌ Detección de pelota en la boca del cañón

Esta etapa es indispensable, ya que sin ella el cañón no podría colocarse en su posición de tiro, debe saberse que el disparador se encuentra cargado mediante la detección de la presencia de la pelota o proyectil, además de que dicho sensado permite conocer el momento exacto en el que la pelota abandona el cañón cuando se dispara, de tal forma que se indique al solenoide del soltador que también debe ser accionado.

➍ Robustecer la colocación de entradas para la detección de la posición

Aunque el cañón ya contaba con dos contactos encargados de detectar la posición del cañón (tiro o recarga), no están colocados de una manera robusta, lo que podía conducir a un fallo posterior o incluso interferencia mecánica al no detenerse el tornillo sinfín. Por lo tanto, debe implementarse una manera robusta de fijarlos al cañón.

Resolución angular

Previamente se había establecido la resolución angular ($\theta < 0.229^\circ$ ^[1]) para garantizar la colisión si se ajusta correctamente el ángulo de tiro del cañón, pero esta medida con en base en un objeto de 150 mm. de diámetro en caída libre. Por motivos de simplificación en el sistema de elevación, dicho objeto fue cambiado por otra pelota de golf de 42.67 mm. de diámetro, lo cual, modifica el valor de la resolución. Atendiendo a los cálculos previos ^[1] y con base en la ecuación (5), se obtiene el nuevo valor para la resolución.

Sabiendo que existe una diferencia de alturas entre el proyectil y la pelota en caída libre, se tiene la siguiente expresión:

$$\Delta h = x \tan \theta \pm \delta \quad (6)$$

El valor máximo de la diferencia δ aceptada es igual a la mitad de la suma de los radios del proyectil y el objetivo ^[1] teniendo por resultado $\delta = 21.34$ mm. Por lo tanto, para determinar el valor de la resolución y conociendo previamente el valor de $x=2500\text{mm}$, se sustituye el valor de la altura por δ de tal manera que:

$$\begin{aligned} \theta &\leq \arctan\left(\frac{\delta}{x}\right) \leq \arctan\left(\frac{21.34}{2500}\right) \\ \therefore \theta &\leq 0.489^\circ \end{aligned}$$

Con base en aspectos prácticos, el nuevo valor de resolución puede redondearse para tener $\theta \leq 0.50^\circ$. Es por tanto imprescindible contar con un instrumento capaz de medir esa resolución para asegurar la colisión en caso de que el experimento se configure de la manera correcta.

Detección de señales y control de los actuadores

Finalmente, el subsistema de tiro necesita comunicarse con el servidor, de tal manera que se pueda conocer el estado en que se encuentra, para estar coordinado con los demás subsistemas. Además, es necesario controlar a los actuadores mencionados, entre los que se encuentran motores de 12 y 24 V.

1.5.2 Subsistema del soltador

Este subsistema cuenta con un motor de pulsos, cuyo movimiento es ciertamente más preciso respecto a un motor de corriente directa. Dicho motor requiere una alimentación de al menos 2.6 V, con una intensidad de corriente de 3.3 A. Además, se debe controlar el soltador con un solenoide de 12V, y poder moverse dentro de los límites establecidos, para lo cual, al igual que para el subsistema de tiro, será necesaria la implementación de control electrónico adicional, estableciendo de la misma forma su comunicación con el servidor.

1.5.3 Subsistema de elevación y repartición

Deberá activarse el motor de la banda (12 V. a más de 2 A.) de elevación sólo durante el tiempo efectivo del experimento, para ahorrar energía y preservar la duración del sistema.

1.5.4 Sistema de tiro parabólico y caída libre

Establecimiento de una posición inicial o HOME

Al igual que en cualquier sistema, se debe elegir una posición inicial para todo el conjunto, tal que se emplee la menor energía posible y permita un diagnóstico del estado del sistema por parte del servidor. Así mismo, conviene que en dicha posición se mantenga la mayoría de los elementos mecánicos del sistema en reposo o que les mantenga en uso, a fin de mantener su duración.

Implementación de una interfaz electrónica y de usuario

Se debe contar con una interfaz electrónica capaz de detectar el estado de los subsistemas, y que pueda comunicarse con el servidor al mismo tiempo, de tal forma que se permita la operación de los actuadores a través de la interfaz de usuario.

Además, el funcionamiento del dispositivo deberá ser comprendido sin mayor problema, lo que incita a la creación de una interfaz interactiva con el usuario, didáctica y que permita el aprendizaje y aprovechamiento del experimento.

Facilidad de mantenimiento y uso

Es importante que cuando el sistema requiera de mantenimiento, este pueda ser ensamblado y desensamblado con facilidad, de manera que permita el cambio de piezas y preservar el estado físico de todo el sistema. Además, deberá permitir el fácil diagnosticar fallas en caso de que existan.

En el siguiente capítulo se presentan las diferentes alternativas propuestas para el diseño de la instrumentación y control del sistema.

CAPÍTULO 2 ANÁLISIS PARA LA INSTRUMENTACIÓN Y EL CONTROL

Para resolver las diferentes necesidades de la instrumentación y el control del sistema, es importante tomar en cuenta una serie de opciones válidas para la resolución de las mismas, siguiendo la metodología de diseño. A continuación se describen las características de los elementos a considerar en la búsqueda de soluciones.

2.1 CONTROL DE ENTRADAS

Para conocer el estado del sistema, es necesario saber la posición de cada uno de los dispositivos así como el conocimiento de algunas variables tales como la tensión de la cuerda, por lo cual, se hace evidente el uso de sensores y transductores. Además, es necesario instrumentar la amplificación angular del cañón a la resolución descrita, tomándose como referencia una serie de dispositivos que permiten la obtención del valor necesario para la operación.

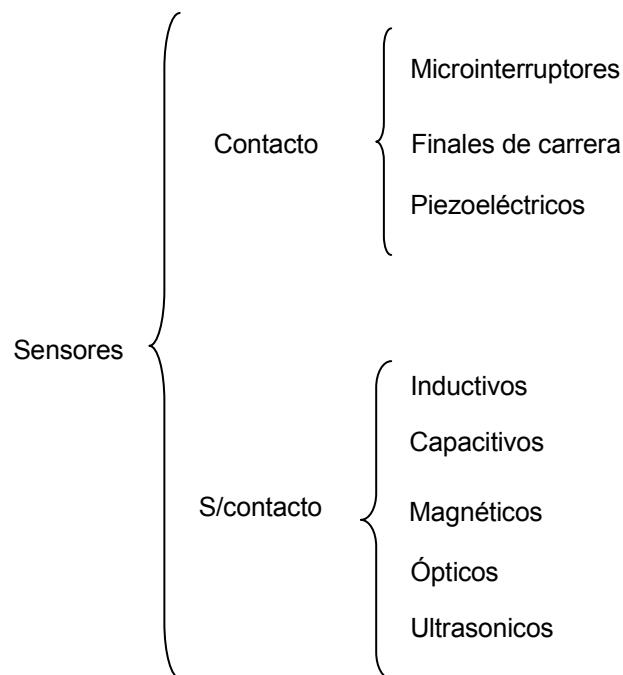
2.1.1 SENSORES

Se definen como elementos que proveen información sobre el estado de un sistema y se clasifican en dos grupos [3]:

- A) Medición o detección de la presencia o movimiento de cuerpos u objetos
- B) Medición de cambios en variables físicas.

Para la selección de un sensor, se deben tener en cuenta los siguientes aspectos:

- Voltaje de alimentación
- Tipo de salida
- Alcance
- Material
- Tamaño
- Velocidad
- Seguridad
- Costo.



Tal como se describe en el diagrama anterior, los sensores pueden ser de contacto o sin contacto, lo cual determina su principio de funcionamiento. A continuación se describe cada clase, así como algunas comparativas en cuanto a sus condiciones de operación.

2.1.1.1 SENSORES DE CONTACTO

Por su estado inicial, un contacto puede ser de dos tipos:

- 1) *Normalmente Abierto (NA)*. Significa que cuando está en reposo, no permitirá el paso de una corriente eléctrica a través de él.
- 2) *Normalmente Cerrado (NC)*. Sólo cuando está en reposo, permitirá el paso de una corriente a través de él.

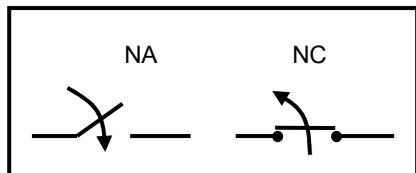


Figura 2.1 Sensores de contacto.

MICROINTERRUPTORES

Están diseñados de tal forma que la conmutación de los contactos ocurra en un punto específico del desplazamiento, independientemente de la velocidad de accionamiento, con la finalidad de evitar que se produzcan arcos eléctricos. La Figura 2.2 muestra su principio de funcionamiento.

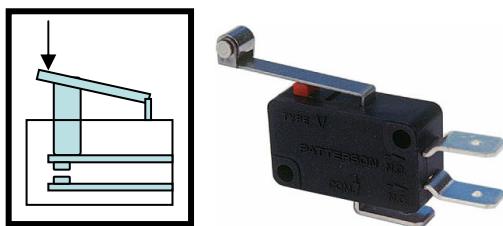


Figura 2.2 Microinterruptor de palanca con rodillo.

SENSORES DE FINAL DE CARRERA

Detectan límites de un movimiento. Se utilizan comúnmente para ubicar la posición de actuadores de manera que se pueda controlar su accionamiento

SENSORES PIEZOELÉCTRICOS

Utilizan un cristal denominado piezoeléctrico, el cual se coloca entre dos placas de electrodos. Cuando se aplica una fuerza a las placas, se produce una diferencia de potencial en la superficie del cristal (efecto piezoeléctrico). Son utilizados como transductores de bajo costo para la realización de medidas dinámicas así como fonocaptores.

2.1.1.2 SENSORES SIN CONTACTO

SENSORES INDUCTIVOS

Tienen una bobina con núcleo de ferrita, un oscilador, un circuito detector y una salida de estado sólido. El oscilador crea un campo electromagnético de alta frecuencia, que irradia de la bobina al frente del sensor, centrado en el eje de la misma. El núcleo de la ferrita envuelve y dirige el campo electromagnético hacia el frente. Sólo detectan materiales como el hierro, acero, cobre, latón, níquel. La Figura 2.3 muestra un sensor inductivo comercial.



Figura 2.3 Sensor inductivo.

SENSORES CAPACITIVOS

La capacitancia, C , depende del valor de la constante dieléctrica, ϵ , del área de traslape, A , entre el dieléctrico y electródos, y de separación, d , entre los mismos:

$$C = \frac{\epsilon A}{d}$$

Los sensores capacitivos detectan la variación del valor de la capacitancia cuando un objeto se encuentra entre los electrodos, con la finalidad de proporcionar un señal que indica la presencia de un material en particular, o bien, desplazamiento.

SENSORES MAGNÉTICOS

Sólo reaccionan ante la presencia de campos magnéticos y pueden ser de tres tipos:

- 1) Magnético-inductivos
- 2) Efecto Reed (Reed-switches)
- 3) Efecto Hall

La Figura 2.4 muestra algunos ejemplos comerciales de estos sensores.



Figura 2.4 Sensores magnéticos [20] (Reed switch y efecto Hall).

SENSORES ÓPTICOS

Su funcionamiento se basa en un transductor que convierte la incidencia de un rayo de luz en una señal eléctrica. Se clasifican en sensores reflectivos y de barrera, como se muestra en la Figura 2.5.

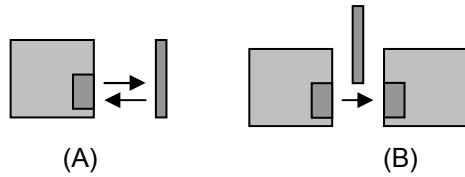


Figura 2.5 Tipos de sensores ópticos (A reflectivos B de barrera).

Para los sensores reflectivos debe tomarse en cuenta que existe una zona ciega en las proximidades, donde no será posible detectar objetos. Se utilizan principalmente en dispositivos de seguridad, de conteo y para la lectura de encoders. La Figura 2.6 muestra el H21A1 y el QRD114, dos sensores ópticos de barrera y retroreflectivo, respectivamente.



Figura 2.6 Sensores ópticos (izq. H21A1 der. QRD114).

SENSORES ULTRASÓNICOS

El sensor emite un pulso sónico que es reflejado de vuelta por cualquier objeto que esté en el cono sónico. El tiempo que toma al eco volver al sensor, es directamente proporcional a la distancia al objeto, debido a que la velocidad del sonido es constante. La Figura 2.7 muestra un sensor ultrasónico.



Figura 2.7 Sensor ultrasónico.

TABLAS COMPARATIVAS

En las tablas 2.1, 2.2 y 2.3 se muestra las condiciones de operación para los sensores sin contacto enunciados, entre las que se encuentran la distancia de sensado, materiales que son capaces de sensar así como la frecuencia de detección. En la Figura 2.8 se aprecian las distancias máximas de sensado para cada tipo de sensor.

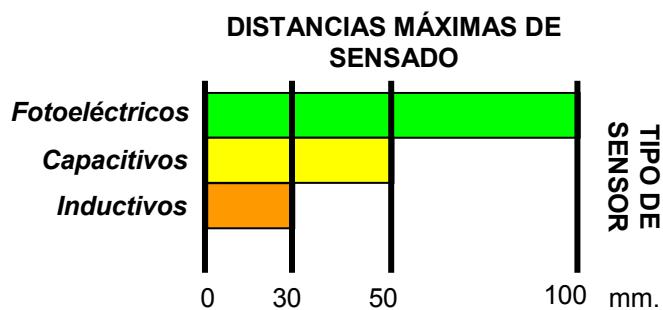


Figura 2.8 Distancia de sensado.

Tabla 2.1 Capacidad de los sensores para detectar materiales.

SENSOR / MATERIAL	Sólidos		Polvo		Líquido	
<i>Fotoeléctricos</i>	X	X			X	X
<i>Inductivos</i>		X		X		X
<i>Capacitivos</i>	X	X	X	X	X	X
<i>Metal (M), no metal(NM)</i>	NM	M	NM	M	NM	M

Tabla 2.2 Condiciones ambientales de operación.

<i>Fotoeléctricos</i>	X	X	X	
<i>Inductivos</i>		X	X	X
<i>Capacitivos</i>		X		X
	Calor	Humedad	Lluvia	Vibraciones

Tabla 2.3 Frecuencia de detección.

<i>Fotoeléctricos</i>	X	X	X	X
<i>Inductivos</i>	X	X	X	
<i>Capacitivos</i>	X	X		
<i>Frecuencia (Hz)</i>	0-10	10-30	30-1 k	1 k-5 k

2.1.2 RESOLUCIÓN ANGULAR

Para determinar el valor del ángulo respecto de la horizontal al cual se encuentra el eje del motor, el instrumento debe contar con una resolución de 0.5° , tal como se especificó en el capítulo anterior, lo cual se puede conseguir utilizando un encóder rotatorio de alta precisión, de al menos 1024 ranuras, o construyendo un instrumento que logre cuantificar el ángulo, pudiendo ser a través de engranes que amplifiquen dicho giro y pueda ser determinado por un transductor como el potenciómetro. A continuación se describe el principio de funcionamiento de cada uno de dichos dispositivos.

2.1.2.1 ENCÓDER ROTATORIO

Es un dispositivo electromecánico usado para convertir la posición angular de un eje a un código digital. Estos dispositivos se utilizan en robótica, en lentes fotográficas de última generación, en dispositivos de entrada de ordenador (tales como el ratón y el trackball), y en plataformas de radar rotatorias. Hay dos tipos principales: *absoluto* y *relativo*.

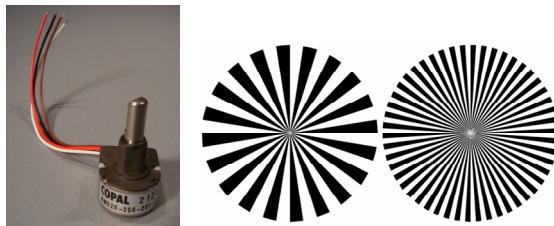


Figura 2.9 Encoder óptico (izq.) y discos de 40 y 100 divisiones (der.).

El tipo *absoluto* produce un código digital único para cada ángulo distinto del eje. Para construirlos, dependiendo de la resolución, se corta un patrón complejo como los de la Figura 2.9, en una hoja de metal y se pone en un disco aislador, que está fijado al eje. También se coloca una fila de contactos deslizantes a lo largo del radio del disco. Mientras que el disco rota con el eje, algunos de los contactos tocan el metal, mientras que otros caen en los huecos donde se ha cortado el metal. La hoja de metal está conectada con una fuente de corriente eléctrica, y cada contacto está conectado con un sensor eléctrico separado. Se diseña el patrón de metal de tal forma que cada posición posible del eje cree un código binario único en el cual algunos de los contactos estén conectados con la fuente de corriente (es decir encendido) y otros no (apagados). Este código se puede leer por un dispositivo controlador, tal como un microprocesador, para determinar el ángulo del eje.

2.1.2.1 ENGRANES

Es un mecanismo utilizado para transmitir potencia de un componente a otro dentro de un instrumento. Los engranajes están formados por dos ruedas dentadas, de las cuales la mayor se denomina “Corona” y la menor “Piñón” como se muestra en la Figura 2.10. Los engranes sirven para transmitir movimiento circular mediante el contacto de sus ruedas dentadas.



Figura 2.10 Engranes corona y piñón, respectivamente.

Para calcular la relación de transmisión, R, existente entre el piñón y la corona, se aplica la siguiente expresión:

$$R = \frac{N^{\circ} \text{dientes}_P}{N^{\circ} \text{dientes}_c}$$

Además, puede conocerse la relación de velocidades sabiendo que los engranes tienen la misma velocidad angular:

$$v_c = v_P$$

De lo cual se tiene que:

$$\omega_{corona} r_{corona} = \omega_{piñón} r_{piñón}$$

Finalmente, se obtiene la relación de velocidades:

$$\frac{\omega_{corona}}{\omega_{piñón}} = \frac{r_{corona}}{r_{piñón}}$$

Es importante destacar que se pueden implementar instrumentos basados en el mismo principio mediante el uso de poleas con banda.

2.1.2.3 POTENCIÓMETRO

El potenciómetro también se puede definir como una resistencia variable, cuyo valor se puede ajustar de manera lineal o giratoria, como el de la Figura 2.11. Existen dos tipos de potenciómetros: lineal y logarítmico. Mientras que en los potenciómetros lineales la variación o cambio en el giro es proporcional a su valor de resistencia, en los potenciómetros logarítmicos se tiene una equivalencia asimétrica respecto a su recorrido, formando una curva similar a la logarítmica.



Figura 2.11 Potenciómetro de perilla.

2.1.3 TENSIÓN DE LA CUERDA

Para detectar la tensión de la cuerda que determina la potencia de tiro, es necesario contar con algún instrumento que permita conocer la misma, lo cual puede conseguirse mediante un sensor de intensidad de corriente, o bien, mediante el conteo de las vueltas que da el motor encargado de producir la tensión. A continuación se describe el principio de funcionamiento de cada alternativa.

2.1.3.1 SENSOR DE CORRIENTE

A medida que se comienza a tensar la cuerda, el motor requiere de una mayor potencia para continuar su giro, lo cual requiere de una mayor intensidad de corriente, la cual puede ser medida a través de un sensor que indique el estado de la tensión. Este sensor emite una salida analógica, la cual puede ser leída para su procesamiento posterior. Para su implementación, puede utilizarse un sensor de efecto Hall.

2.1.3.2 CONTEO DE VUELTAS

Otra de las formas en que se puede conocer la tensión de la cuerda es mediante el conteo de las vueltas que realiza el motor que produce la tensión por medio de un encóder incremental, de tal forma que se puedan delimitar los diferentes estados de tensión con base en el número de vueltas. La implementación del encoder incremental se puede realizar mediante el uso de un sensor óptico.

2.1.4 TRATAMIENTO DE SEÑALES

Todos los sensores envían señales ya sean analógicas o digitales. De manera que para ser interpretadas, la mayoría de los circuitos integrados encargados del procesamiento de estos datos requieren valores dentro del margen de 5 V para ser reconocidas como “1” lógico; muchos sensores envían señales superiores a este valor, para lo cual será necesario el uso de circuitos capaces de establecer los valores dentro del límite permisible.

CIRCUITO INTEGRADO MAX232

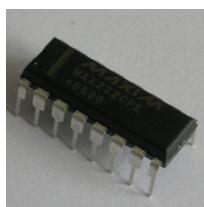


Figura 2.12 el circuito integrado MAX232.

El circuito integrado MAX232, mostrado en la Figura 2.12, es frecuentemente utilizado para convertir señales de un puerto serial RS-232 a señales de uso compatibles con circuitos con lógica de transistores, o TTL por sus siglas en inglés. El MAX232 es un receptor/transmisor doble que típicamente convierte las señales de recepción y transmisión, denominadas RX y TX respectivamente. El circuito convierte las señales en valores permisibles de voltaje para cualquier circuito lógico, entre 3 y 5 V.

2.2 CONTROL DE ACTUADORES

Para controlar los diferentes actuadores, cuyo voltaje de alimentación está entre los 12 y 24 V. a corrientes de hasta 4.5 A (contando los picos de corriente para algunos motores), se muestran una serie de dispositivos que permiten resolver el problema. Además de destacar componentes encargados del accionamiento de los actuadores, se requiere controlar la dirección de algunos motores, por lo cual se hace indispensable el uso o implementación de dispositivos que cumplan dicha función.

2.2.1 TRANSISTORES

Los transistores son dispositivos electrónicos que pueden utilizarse como interruptores para controlar el encendido y apagado de actuadores que trabajan con corriente directa. Al recibir una señal de control el transistor cierra un circuito eléctrico, y en ausencia de dicha señal lo abre.

Por su modo de conducción, los transistores pueden ser *PNP* y *NPN* (Figura 2.13).

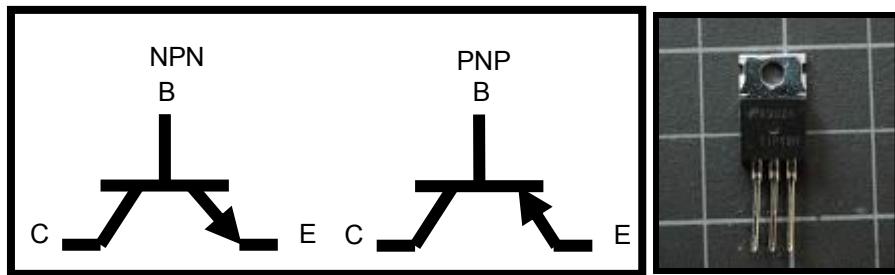


Figura 2.13 Tipos de transistores.

- *NPN*. El flujo de corriente va de colector a emisor (sink).
- *PNP*. El flujo de corriente va de emisor a colector (source).

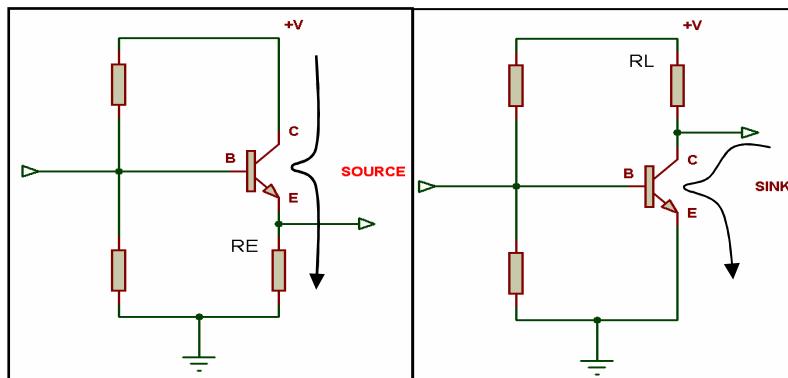


Figura 2.14 Configuraciones del transistor.

Como dispositivo de Entrada-Salida, o E/S de corriente, un transistor puede ser visto como un dispositivo unidireccional para alimentar una resistencia de carga. Como se muestra en la Figura 2.14, puede actuar ya sea de modo que suministre corriente (source), o bien, que reciba corriente (sink), pero no ambas.

Dependiendo de la carga del transistor y al momento de disipar energía, éstos pueden llegar a calentarse, por lo que es necesario que se utilicen con un disipador de calor.

TRANSISTORES DE EFECTO DE CAMPO (FET)

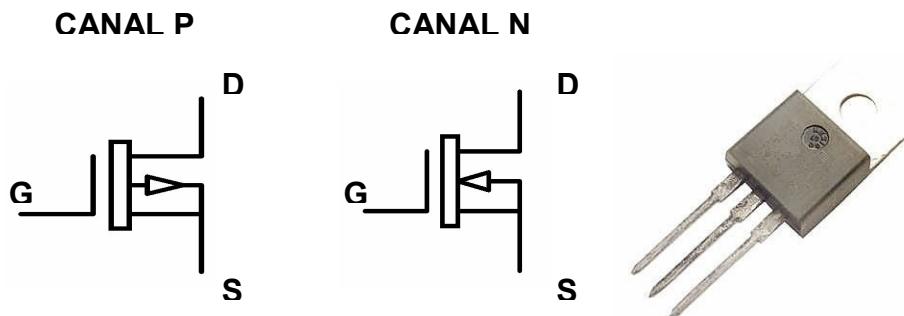


Figura 2.15 Tipos de MOSFET (izq.) y encapsulado (der.).

Es el transistor más utilizado en la industria microelectrónica por las grandes cantidades de corriente que llegan a soportar. Prácticamente la totalidad de los circuitos integrados de uso comercial están basados en transistores MOSFET.

Un transistor MOSFET consiste en un sustrato de material semiconductor dopado en el que, mediante técnicas de difusión de dopantes, se crean dos islas de tipo opuesto separadas por un área sobre la cual se hace crecer una capa de dieléctrico culminada por una capa de conductor. Los transistores MOSFET se dividen en dos tipos fundamentales dependiendo de cómo se haya realizado el dopaje:

- NMOS: sustrato de tipo *P* y difusiones de tipo *N*.
- PMOS: sustrato de tipo *N* y difusiones de tipo *P*.

Las áreas de difusión se denominan fuente (source), drenaje (drain) y el conductor entre ellos es la compuerta (gate).

El transistor MOSFET tiene tres estados de funcionamiento: corte, conducción lineal y saturación.

Las aplicaciones de MOSFET discretos más comunes son:

- Resistencia controlada por tensión.
- Circuitos de commutación de potencia (HEXFET, FREDFET, puente H, etc.)
- Mezcladores de frecuencia, con MOSFET de doble compuerta.

2.2.2 RELEVADORES

Interruptor de tipo electromagnético, formado por una bobina y al menos un contacto que será accionado por el campo magnético de la bobina cuando ésta se encuentra energizada. Aunque existen varios tipos de relevadores, su principio de operación es el mismo. En la Figura 2.16 se aprecia un diagrama del relevador así como el componente físico.

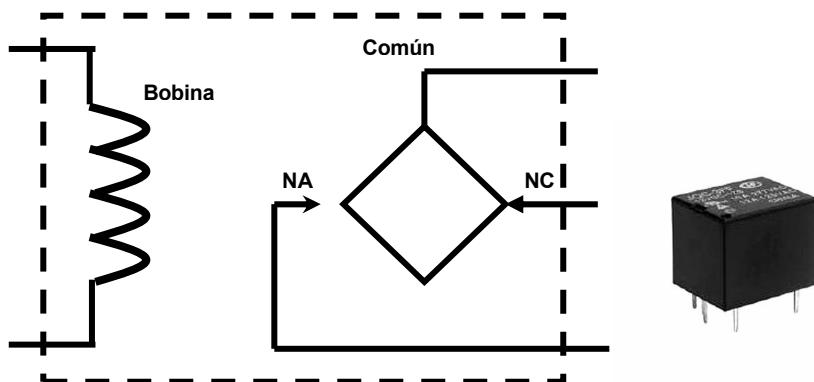


Figura 2.16 Diagrama de un relevador de un polo y dos tiros (izq.) y componente físico (der.).

Existen multitud de tipos distintos de relevadores, dependiendo del número de contactos, de la tensión admisible por los mismos, tipo de corriente de accionamiento, tiempo de activación y desactivación, etc. Cuando controlan grandes potencias se les llama contactores. Los relevadores se clasifican de la siguiente forma:

Relevador electromecánico

Relevador de estado sólido

Relevador de corriente alterna

Relevador de láminas.

2.2.3 PUENTE H

Es un circuito electrónico que permite a un motor eléctrico de corriente directa girar en ambos sentidos, *avance* y *retroceso*. Son ampliamente usados en robótica y como convertidores de

potencia. Los puentes H están disponibles como circuitos integrados, pero también pueden construirse a partir de componentes discretos.

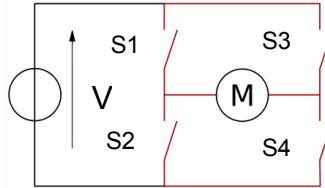


Figura 2.17 Estructura del puente H.

El término "puente H" proviene de la típica representación gráfica del circuito. Un puente H se construye con cuatro interruptores (mecánicos o mediante transistores). Atendiendo a la Figura 2.17, cuando los interruptores S1 y S4 están cerrados, S2 y S3 abiertos, se aplica una tensión positiva en el motor, haciéndolo girar en un sentido. Abriendo los interruptores S1 y S4, cerrando S2 y S3, el voltaje se invierte, permitiendo el giro en sentido inverso del motor.

El puente H se usa para invertir el giro de un motor, pero también puede usarse para frenarlo (de manera brusca), al hacer un corto entre los bornes del motor, o incluso puede usarse para permitir que el motor frene bajo su propia inercia, cuando desconectamos el motor de la fuente que lo alimenta. En la Tabla 2.4 se resumen las diferentes acciones.

Tabla 2.4 Estados del puente H (atendiendo a la Figura 2.17)

S1	S2	S3	S4	ESTADO
1	0	0	1	El motor gira en avance
0	1	1	0	El motor gira en retroceso
0	0	0	0	El motor se detiene bajo su inercia
0	1	0	1	El motor frena (<i>fast-stop</i>)

Lo más habitual en este tipo de circuitos es emplear interruptores de estado sólido (como transistores), puesto que sus tiempos de vida y frecuencias de conmutación son mucho más altos. En convertidores de potencia es impensable usar interruptores mecánicos, dado su bajo número de conmutaciones de vida útil y las altas frecuencias que se suelen emplear.

Además, los interruptores se acompañan de diodos (conectados a ellos en paralelo) que impiden a las corrientes circular en sentido inverso al previsto cada vez que se commute la tensión.

2.2.4 CONTROL DE UN MOTOR DE PULSOS

Dado que el actuador encargado del posicionamiento del soltador es un motor de pulsos, es importante destacar la manera de manipularlos [9]. En contraparte a un motor convencional de corriente directa, el control de un motor de pulsos no se realiza aplicando un voltaje en las bobinas. Estos motores cuentan con varias bobinas que, para producir el avance de un paso, necesitan ser alimentada en una secuencia determinada. De esta manera es que siguiendo una determinada secuencia se produce el giro del motor, y aplicando lógica inversa se consigue que el motor gire en el sentido opuesto. En caso de que la secuencia de alimentación sea incorrecta, el motor se moverá aleatoriamente sin obtener los resultados esperados. La secuencia normal para el control de este motor se muestra en la Figura 2.18



PASO	Bobina A	Bobina B	Bobina C	Bobina D	
1	ON	ON	OFF	OFF	
2	OFF	ON	ON	OFF	
3	OFF	OFF	ON	ON	
4	ON	OFF	OFF	ON	

ON – encendido, OFF - apagado

Figura 2.18 Motor de pulsos y la secuencia normal para su control.

2.3 PROCESAMIENTO DE SEÑALES Y COMUNICACIÓN DEL SISTEMA

Para poder controlar el sistema, es necesario procesar los datos de entrada y salida de los diferentes subsistemas, por lo que se requiere utilizar dispositivos electrónicos como los microcontroladores, microprocesadores o controladores lógicos programables. Además, dada la condición de operación remota, resulta necesario el uso de un protocolo de comunicación rápido, de manera que se conozca el estado del sistema en todo momento y se pueda transmitir y recibir información de manera estable.

2.3.1 PROCESAMIENTO DE SEÑALES

El procesamiento de señales debe efectuarse por medio de un dispositivo que reconozca voltajes lógicos TTL, donde el cero lógico equivale a 0 V y el uno lógico a 5 V. Además, dicho dispositivo debe ser capaz de establecer comunicación con el servidor. Es por ende indispensable, la implementación del control por medio de alguno de los dispositivos que se detallan a continuación.

2.3.1.1 MICROCONTROLADORES

Un microcontrolador es un circuito integrado o chip que incluye en su interior las tres unidades funcionales de una computadora: unidad central de procesamiento, memoria y unidades de E/S (entrada/salida). Son diseñados para reducir el costo económico y el consumo de energía de un sistema en particular.

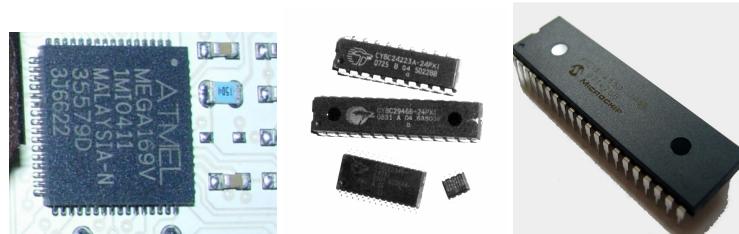


Figura 2.19 Microcontroladores Atmel, Cypress y Microchip [8].

Entre las marcas más comunes, presentadas en la Figura 2.19, se encuentran Atmel, Cypress, Intel y Microchip, estos últimos fabricantes de los microcontroladores PIC, o controladores de interfaz periférica.

2.3.1.2 MICROPROCESADORES

El microprocesador es un circuito integrado que contiene algunos o todos los elementos de hardware, y la unidad central de procesamiento, o CPU por sus siglas en inglés, que es un concepto lógico. Una CPU puede estar soportada por uno o varios microprocesadores, y un microprocesador puede soportar una o varias CPU. Un núcleo suele referirse a una porción del procesador que realiza todas las actividades de una CPU real.

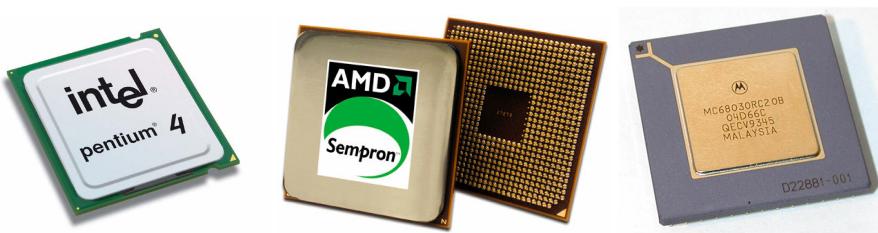


Figura 2.20 Microcontroladores Intel, AMD y Motorola.

Desde el punto de vista lógico, singular y funcional, el microprocesador está compuesto básicamente por: varios registros, una unidad de control, una unidad lógica-aritmética y dependiendo del procesador, puede contener una unidad de punto flotante. El microprocesador ejecuta instrucciones almacenadas como números binarios organizados secuencialmente en la memoria principal. Entre las marcas más comunes de microprocesadores se encuentran Intel, AMD y Motorola, mostradas en la Figura 2.20.

2.3.1.3 CONTROLADORES LÓGICOS PROGRAMABLES

Según la Asociación Nacional de Manufactureros Eléctricos, o *National Electrical Manufacturers Association* (NEMA) en inglés, se define a un controlador lógico programable, o PLC por sus siglas en inglés, como:

“Un aparato electrónico operado digitalmente, que usa una memoria programable para el almacenamiento interno de instrucciones para implementar funciones específicas, tales como lógica, secuenciación, registro y control de tiempos, conteo y operaciones aritméticas para controlar, a través de módulos de entrada/salida digitales (ON/OFF) o analógicos (1-5 V de corriente directa, 4-2 mA), varios tipos de máquinas o procesos”.



Figura 2.21 Controlador lógico programable.

Un PLC como el de la Figura 2.21, cuenta con una unidad central de procesamiento, una interfaz de entrada/salida, una fuente de poder y un dispositivo de programación. Las señales de entrada y salida pueden ser discretas, digitales y analógicas. El PLC cuenta con módulos especiales de entrada/salida como contadores de alta velocidad o contadores para encóders.

2.3.2 COMUNICACIÓN DEL SISTEMA

Si bien se ha establecido la necesidad de comunicar el estado del sistema a un servidor así como la recepción de comandos por parte del mismo, es importante describir los protocolos de comunicación que se pueden utilizar, de manera que se tenga una comunicación rápida y eficiente, atendiendo a los principios de operación del sistema. Deben considerarse factores como el formato de transmisión, la distancia del cableado, velocidad, así como los usos más frecuentes, variables importantes durante la construcción del sistema. La Tabla 2.5 muestra una comparativa entre los protocolos más populares [5]

Tabla 2.5 Comparativa entre protocolos de comunicación.

Protocolo	Formato	Distancia del cableado [ft]	Velocidad máxima [bits/s]	Uso típico
USB	asíncrono serial	16 (hasta 96 ft con 5 hubs)	1.5 M, 12 M, 480 M	Mouse, teclado, audio, impresoras
Ethernet	serial	1600	10 G	Comunicaciones de red.
IEEE-1394b (FireWire 800)	serial	300	3.2 G	Transmisión de video, almacenaje masivo
IEEE-488 (GPIB)	paralelo	60	16 M	Instrumentación
IrDA	asíncrono serial infrarrojo	6	3.4 M	Impresoras, computadoras de mano
I ² C	síncrono serial	18	2 M	Comunicación entre microcontroladores
Microwire	síncrono serial	10	31.5 k	Comunicación entre microcontroladores
Puerto paralelo	paralelo	10-30	8 M	Impresoras, escáneres, discos duros
RS-232 (EIA/TIA-232)	asíncrono serial	50-100	20 k (115 k con hardware)	Módem, mouse, instrumentación
RS-485 (TIA/EIA-485)	asíncrono serial	4000	10 M	Obtención de datos y control de sistemas
SPI	síncrono serial	10	2.1 M	Comunicación entre microcontroladores

2.4 ALIMENTACIÓN Y REGULACIÓN DE VOLTAJE

Existen diferentes formas para suministrar voltaje al sistema, tomando en cuenta que los dispositivos necesitan diversos valores de voltaje para su alimentación. Por tanto, se deberá implementar una etapa de regulación, la cual sea capaz de suministrar 5, 12 y 24 V ya que tanto actuadores como algunos circuitos integrados requieren tales valores para su funcionamiento. A continuación se mencionan los dispositivos que pueden cubrir esta necesidad.

2.4.1 FUENTES DE ALIMENTACIÓN

Estos elementos se encargan de transformar la alimentación alterna del tomacorriente a un valor de voltaje directo con una intensidad corriente determinada, dependiendo de su aplicación. Dichos dispositivos cuentan con un valor muy estable ya que presentan una buena regulación, de forma que se puede garantizar el valor de voltaje de salida. La Figura 2.22 muestra una fuente de computadora, la cual en algunos casos funciona cuando encuentra una carga o dispositivo al cual le suministre voltaje. El uso dos fuentes, de 12 y 24 V respectivamente, puede solucionar la necesidad de alimentación de la electrónica del sistema.



Figura 2.22 Fuente de alimentación para computadora.

2.4.2 TRANISTORES REGULADORES DE VOLTAJE

Para obtener la alimentación que requieren los circuitos integrados del sistema, se pueden considerar dos reguladores de voltaje, el L7805CT y el KA78R05 mostrados en la Figura 2.23, que suministran 5 V a la salida y con un valor de tensión estable. Estos reguladores soportan intensidades de corriente de hasta 1.5 A y requieren un disipador para funcionar adecuadamente.

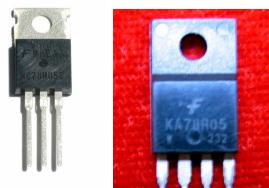


Figura 2.23 Reguladores L7805CT y KA78R05, respectivamente.

CAPÍTULO 3 DISEÑO DE LA INSTRUMENTACIÓN Y EL CONTROL

Para la toma de decisiones y la construcción de los instrumentos así como para el control del sistema, es importante establecer los elementos que se van a utilizar, atendiendo a sus ventajas respecto a los demás dispositivos, tales como su costo, facilidad de implementación, tiempo para su implementación y duración. En los siguientes apartados se destacan las comparaciones realizadas para la construcción del sistema.

3.1 SUBSISTEMA DE TIRO

Como se describió previamente, para el cañón es necesario definir una manera de interactuar con el sistema, además de desarrollar los instrumentos que permitirán medir la tensión de la cuerda y la resolución angular. A continuación se presentan las posibles soluciones para cada necesidad.

3.1.1 Detección de entradas y control de actuadores

El uso de sensores de bajo costo y buena resolución, y que además sean de fácil implementación permite la construcción del control electrónico para la detección de entradas provenientes de dichos sensores, que deberán transmitirse a la unidad de procesamiento. Entre las opciones más económicas se encuentran los sensores optoelectrónicos dada su producción en masa y uso en el ámbito académico. Además, no es necesaria la detección de un tipo de forma en particular, simplemente se debe detectar la presencia de un objeto.

Sin embargo, para el control de actuadores, se debe implementar una etapa de potencia, misma que recibirá señales digitales provenientes de la unidad de procesamiento. Dicho control de actuadores se puede construir mediante el uso de transistores, pero se debe de tomar en cuenta las condiciones de operación así como la corriente de operación de los dispositivos. En la Tabla 3.1 se realiza la comparación de las ventajas y desventajas entre el uso del transistor Darlington y el transistor MOSFET.

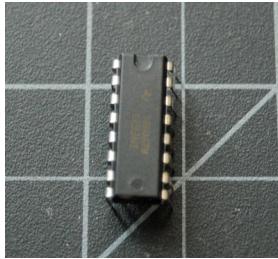
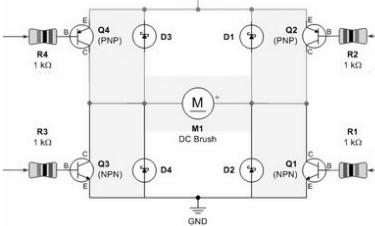
Tabla 3.1 Comparación de los diferentes tipos de transistor.

Componente	Ventajas	Desventajas
 Transistor Darlington	<ul style="list-style-type: none"> ✓ El valor de la ganancia de corriente es muy alto ✓ Pueden ser manipulados con facilidad 	<ul style="list-style-type: none"> ✗ Maneja valores de corriente de hasta 4 A ✗ Se pueden calentar mucho cuando se incrementa la corriente ✗ Alta tensión de saturación
 Transistor MOSFET [12]	<ul style="list-style-type: none"> ✓ Puede trabajar con valores de corriente muy altos ✓ Presentan poco calentamiento por disipación de energía ✓ Generan un menor nivel de ruido 	<ul style="list-style-type: none"> ✗ Tiende a autodestruirse ✗ Es susceptible a cargas estáticas ✗ Algunos presentan no linealidades ✗ Requiere una resistencia de 10 kΩ entre la compuerta y la fuente.

Cabe mencionar que para producir el movimiento lateral del cañón, se debe controlar la dirección del motor de corriente directa encargado de dicha acción. Esto se puede lograr mediante la el uso

del puente H descrito en el capítulo previo. Existen circuitos integrados como el L293D ó el L298N que incluyen dos puentes H cada uno, pero también se puede implementar un puente H mediante el uso del transistor Darlington o del transistor MOSFET. En la Tabla 3.2 se establece las comparativas entre el uso de alguno de los componentes mencionados.

Tabla 3.2 Comparación de los circuitos denominados puente H.

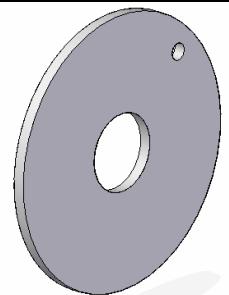
Dispositivo	Ventajas	Desventajas
 L293D	<ul style="list-style-type: none"> ✓ Alta inmunidad al ruido ✓ Sencillos de usar y colocar ✓ Contiene diodos de protección ✓ Soportan hasta 36V 	<ul style="list-style-type: none"> ✗ Soporta corrientes pico de hasta 1.2 A ✗ La corriente de salida puede ser de hasta 600mA ✗ Costo elevado
 L298N	<ul style="list-style-type: none"> ✓ Cuenta con sensor de corriente ✓ Alta inmunidad al ruido eléctrico ✓ Puede detectar señales lógicas a partir de 1.5V ✓ Soporta hasta 46V 	<ul style="list-style-type: none"> ✗ Soporta corrientes pico de máximo 4 A ✗ Forma física, ubicación en un circuito ✗ Se calientan demasiado
 Puente H con transistores [11]	<ul style="list-style-type: none"> ✓ Se puede diseñar de acuerdo a la necesidad ✓ Pueden soportar hasta 60 A (MOSFET) ✓ Fácil implementación 	<ul style="list-style-type: none"> ✗ Requiere de varios componentes para su implementación ✗ Costo elevado ✗ Susceptible al ruido ✗ Requiere un par con resistencias internas de fuente a descarga similares (MOSFET)

3.1.2 Tensión de la cuerda

Se había establecido previamente que se puede conocer la tensión de la cuerda a partir de la implementación de un sensor de corriente que indique el valor de la misma conforme se va tensando la cuerda, o bien, mediante la construcción de un encóder rotatorio incremental que

permita conocer el numero de vueltas que gira el motor de tensión, de manera que se compare la potencia de tiro con un numero de vueltas del motor determinado. Además, cabe destacar que es imprescindible seleccionar una cuerda que pueda soportar la tensión a la cual va a ser sometida. La Tabla 3.3 presenta las dos posibles soluciones para este requerimiento.

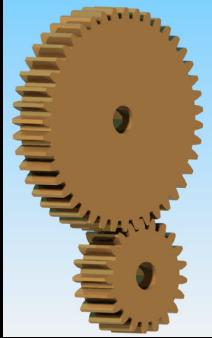
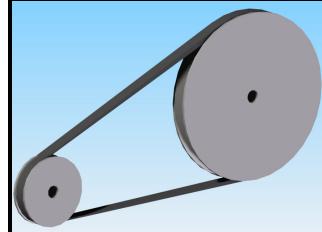
Tabla 3.3 Comparación de los instrumentos para la detección de la tensión de la cuerda.

Instrumento	Ventajas	Desventajas
 Sensor de corriente	<ul style="list-style-type: none"> ✓ Fácil implementación ✓ Resistente a las vibraciones mecánicas 	<ul style="list-style-type: none"> ✗ Baja precisión ✗ No tiene retroalimentación ✗ Requiere acondicionamiento ✗ Costo elevado ✗ Histéresis
 Conteo de vueltas	<ul style="list-style-type: none"> ✓ Precisión en el conteo ✓ Permite discretizar el valor de la potencia ✓ Fácil implementación del control 	<ul style="list-style-type: none"> ✗ Requiere manufactura ✗ Requiere un sensor para funcionar ✗ Necesita ajustarse al motor ✗ Susceptible a vibraciones mecánicas

3.1.3 Amplificación angular

La resolución angular puede obtenerse a través de una transmisión o mediante el uso de un encoder de alta precisión. Mientras que el uso de engranes o bandas requiere del ajuste de un transductor adicional que permita medir la señal analógica, el encoder sólo requiere ajustarse a la flecha encargada del giro del cañón. Las principales ventajas y desventajas de usar alguno de los mecanismos o dispositivos señalados se presentan en la Tabla 3.4.

Tabla 3.4 Comparación de los dispositivos para la amplificación angular.

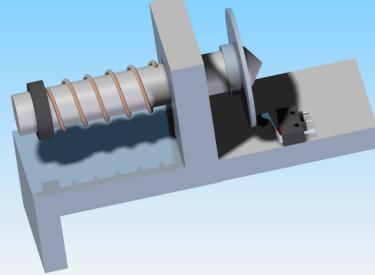
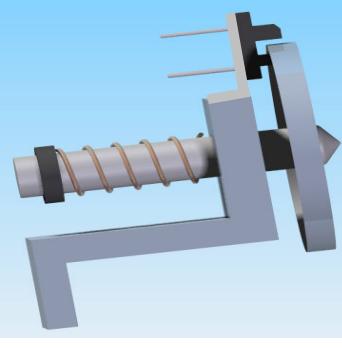
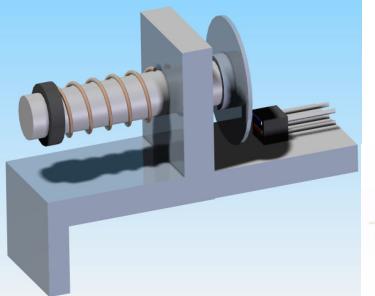
Instrumento	Ventajas	Desventajas
 Engranes (piñón y corona)	<ul style="list-style-type: none"> ✓ Se obtiene exactitud en la relación de transmisión ✓ Duración ✓ La transmisión es por rodadura pura ✓ Rotaciones de velocidad angular uniforme ✓ Se puede adaptar un sensor o transductor para la medición electrónica del giro 	<ul style="list-style-type: none"> ✗ Costo de manufactura ✗ Se deben posicionar de manera que no tengan juego ✗ Pueden ser ruidosos
 Poleas con banda	<ul style="list-style-type: none"> ✓ Transmisión poco ruidosa ✓ Fácil implementación ✓ Bajo costo de mantenimiento ✓ Se puede adaptar un sensor o transductor para la medición electrónica del giro 	<ul style="list-style-type: none"> ✗ Se pueden patinar si no están bien tensadas ✗ Pierden tensión con el uso, requieren tensarse periódicamente
 Encoder rotatorio incremental[22]	<ul style="list-style-type: none"> ✓ Alta precisión ✓ Fácil colocación ✓ Duración 	<ul style="list-style-type: none"> ✗ Costo muy elevado ✗ Requieren un buen algoritmo, como la modulación vectorial ✗ Algunos necesitan de un decodificador o software especial para su control

3.1.4 Detección del trinquete

Como se describió previamente, se debe detectar el momento en el que el trinquete comienza su función. Esto se puede conseguir detectando su proximidad, ya que cuando inicia la tensión, dicho trinquete se acerca ligeramente hacia el cañón. En la Figura 3.1 se observa la manera en que opera el trinquete.

La proximidad mencionada se puede medir a través de un microinterruptor con rodillo, o bien, con un sensor óptico, mismos que detectan a un disco colocado en la parte posterior del trinquete. Las diferentes soluciones y su comparación se exponen en la Tabla 3.5.

Tabla 3.5 Opciones para el mecanismo de detección del trinquete.

Mecanismo	Ventajas	Desventajas
 Microswitch con rodillo	<ul style="list-style-type: none"> ✓ Fácil implementación mecánica ✓ No requiere circuito de control ✓ Permite el giro del disco 	<ul style="list-style-type: none"> ✗ Puede existir fricción con el sensor
 Sensor óptico de barrera	<ul style="list-style-type: none"> ✓ No presenta fricción con el sensor ✓ Se puede ajustar la distancia 	<ul style="list-style-type: none"> ✗ Requiere manufactura adicional para el disco ✗ Debe ajustarse de tal manera que no se produzca interferencia con el sensor ✗ Requiere circuito electrónico de control
 Sensor óptico de proximidad	<ul style="list-style-type: none"> ✓ No presenta fricción con el sensor ✓ Se puede ajustar la distancia 	<ul style="list-style-type: none"> ✗ Susceptible a vibraciones mecánicas ✗ Requiere circuito electrónico de control ✗ Debe calibrarse para que sea preciso ✗ Susceptible a cambios en la luz del ambiente

En la Figura 3.1 se observa la manera en que opera el trinquete

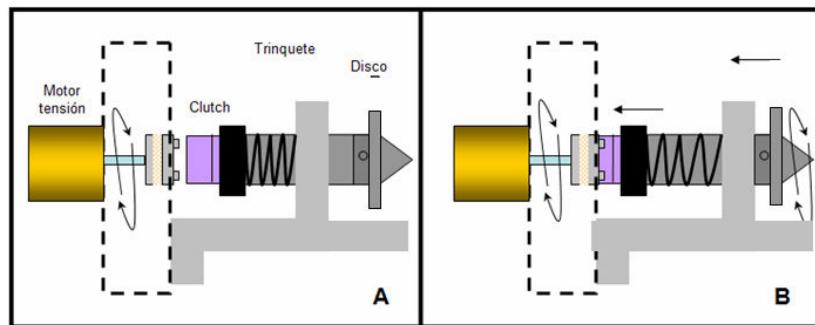
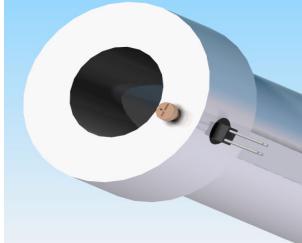
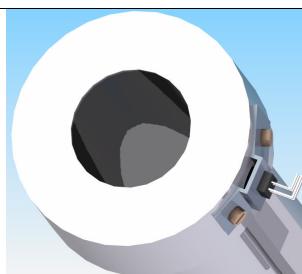
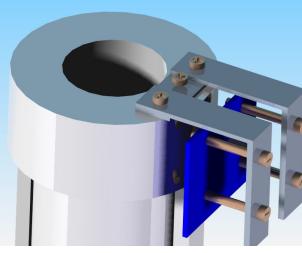
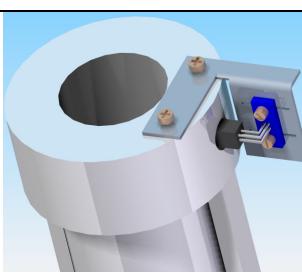


Figura 3.1 Detección del trinquete (A. previo a la tensión, B. inicio de la tensión).

3.1.5 Detección de carga del proyectil

Para detectar la presencia de del proyectil es conveniente utilizar un sensor óptico de proximidad como el QRD114. Sin embargo, la colocación de dicho sensor en la boca del cañón debe ser de tal manera que se pueda ajustar la distancia entre el sensor y el objeto a detectar además de que debe soportar las vibraciones mecánicas que se puedan producir por los movimientos laterales y angulares del cañón. En la Tabla 3.6 presentan las opciones para la construcción de dicho mecanismo.

Tabla 3.6 Comparación de los instrumentos para la amplificación angular.

Mecanismo	Ventajas	Desventajas
 Tornillo para fijar sensor	<ul style="list-style-type: none"> ✓ Fácil implementación mecánica ✓ Ajuste sencillo ✓ Requiere de sólo un elemento 	<ul style="list-style-type: none"> ✗ No se tiene precisión en la distancia de sensado ✗ Susceptible a vibraciones mecánicas ✗ Puede que no se ajuste bien el sensor
 Placa lateral	<ul style="list-style-type: none"> ✓ Fácil implementación mecánica ✓ Ajuste sencillo ✓ Pocos elementos para su construcción 	<ul style="list-style-type: none"> ✗ Puede llegar a interferir con la visión de la cámara ✗ Susceptible a vibraciones mecánicas
 Placa de movimiento	<ul style="list-style-type: none"> ✓ Su peso podría balancear el cañón ✓ Ajuste de distancia gradual y preciso 	<ul style="list-style-type: none"> ✗ Requiere mucha manufactura ✗ Requiere precisión para la colocación del sensor
 Lámina con rieles	<ul style="list-style-type: none"> ✓ Manufactura sencilla ✓ Fácil implementación ✓ Permite ajustar distancia de sensado 	<ul style="list-style-type: none"> ✗ Puede ser susceptible a vibraciones mecánicas ✗ Requiere precisión para la colocación del sensor

3.2 SUBSISTEMA DEL SOLTADOR

3.2.1 Detección de entradas y control de actuadores

Como se mencionó previamente, para el subsistema del soltador, debe destacarse que los sensores ópticos de final de carrera se encuentran ubicados a una altura de 2.5 m sobre el piso,

por tanto, la longitud del cableado genera un valor de resistencia más grande que puede afectar la señal de salida, es decir, que si tales sensores se alimentan con 5 V para proporcionar un valor lógico, la ubicación del dispositivo encargado del procesamiento de dicha señal no pueda reconocerla adecuadamente. Sin embargo, se puede resolver esta cuestión si se alimentan los sensores de final de carrera a una tensión mayor y se transforman a un valor de TTL mediante el uso de un circuito integrado intermedio, como el MAX232, descrito previamente.

En cuanto al control de actuadores, el solenoide del soltador se puede accionar mediante una etapa de potencia, tomando en cuenta la comparación mostrada en la Tabla 3.1. Sin embargo, como se había mencionado, el posicionamiento del soltador se realiza mediante la operación de un motor de pulsos, el cual requiere de un circuito electrónico de control adicional, similar el mostrado en la Figura 3.2. Las señales de control para dicho motor se pueden manipular mediante el uso de un circuito integrado de puente H doble, como el L293D ó L298N mostrados en la Tabla 3.2.

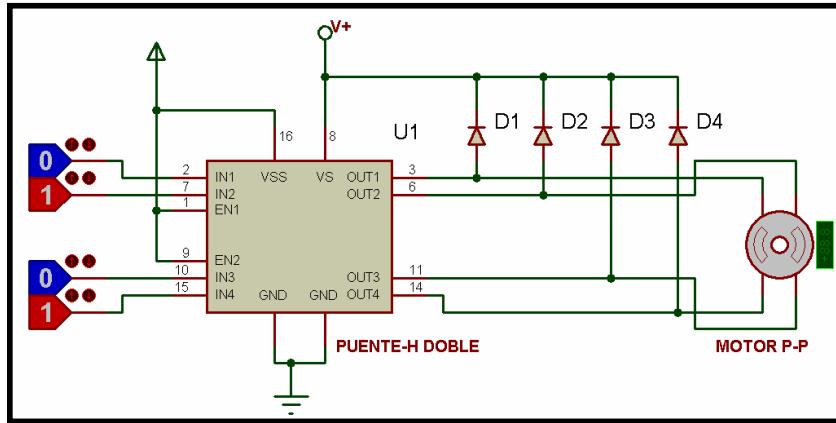


Figura 3.2 Circuito para el control de señales para un motor de pulsos con un L293D.

3.3 SUBSISTEMA DEL ELEVACIÓN Y REPARTICIÓN

Para controlar el accionamiento de la banda, se requiere del uso de una etapa de potencia, misma que se operará mediante una señal digital proveniente de la unidad de procesamiento. Dado que el motor de la banda es el mismo que se usa para el movimiento lateral del cañón del subsistema de tiro, se puede construir un circuito similar, tomando en cuenta los elementos de la Tabla 3.1.

3.4 SISTEMA DE TIRO PARABÓLICO Y CAÍDA LIBRE

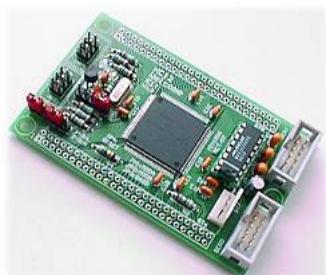
El sistema debe contar con un elemento capaz de recibir datos y manejar las salidas de acuerdo a las entradas. Además, debe contar con una etapa de regulación de voltaje encargada de suministrar el voltaje adecuado a todos los dispositivos que lo requieran.

No obstante, el sistema debe comunicarse de manera rápida y efectiva con el servidor. Para poder diseñar la interfaz de usuario es necesario establecer un protocolo de comunicación, concepto que se desarrolla en el capítulo 4. A continuación se establecen las comparativas para la selección de un dispositivo adecuado para el control electrónico de señales así como un circuito integrado que permita una alimentación de 5V estable.

3.4.1 Procesamiento de señales

Es necesario establecer una unidad de procesamiento que sea fácil de implementar, de bajo costo y que se pueda adaptar al diseño de los instrumentos de manera que pueda procesar su estado y pueda controlar los actuadores mediante su etapa de potencia. La comparación entre los dispositivos previamente expuestos se realiza en la Tabla 3.7.

Tabla 3.7 Comparación de los dispositivos de procesamiento de señales.

Dispositivo	Ventajas	Desventajas
 Microcontrolador PIC	<ul style="list-style-type: none"> ✓ Bajo costo de adquisición e implementación ✓ Existe mucha documentación sobre su uso ✓ Soporta programación en varios lenguajes ✓ Se puede dedicar parte de la memoria a guardar datos y programas auxiliares ✓ Cuenta con módulos de entrada analógico y PWM entre otros ✓ Se puede mejorar mediante C# 	<ul style="list-style-type: none"> ✗ Sensible a efectos electromagnéticos ✗ No resiste sobrevoltaje o sobrecarga
 Microprocesador	<ul style="list-style-type: none"> ✓ Rápida capacidad de procesamiento ✓ Se pueden programar mediante el uso de diferentes lenguajes ✓ Se puede dedicar parte de la memoria a guardar datos y programas auxiliares 	<ul style="list-style-type: none"> ✗ Costo elevado ✗ Por lo regular requieren una tarjeta de desarrollo ✗ Sensible a efectos electromagnéticos ✗ No resiste sobrevoltaje o sobrecarga
 PLC	<ul style="list-style-type: none"> ✓ Tamaño ✓ Buen control de procesos secuenciales ✓ Mantenimiento económico ✓ Existen módulos que permiten la programación por PC 	<ul style="list-style-type: none"> ✗ Costo elevado ✗ Limitación en los procesos internos que puede realizar ✗ Necesidad de cableado

3.4.2 Regulación de voltaje

La etapa de regulación de voltaje del sistema debe contener elementos que sean estables y que puedan cumplir eficientemente con esta necesidad [21]. La Tabla 3.8 compara las dos alternativas propuestas anteriormente.

Tabla 3.8 Ventajas y desventajas de los reguladores.

Regulador	Ventajas	Desventajas
 L7805CT	<ul style="list-style-type: none"> ✓ Bajo costo ✓ No necesita componentes adicionales para proporcionar una fuente estable de energía ✓ Fácil implementación 	<ul style="list-style-type: none"> ✗ El voltaje de entrada siempre tiene que ser más alto que el de salida ✗ No pueden proveer tanta energía a diseños que utilicen componentes discretos ✗ Soporta poca corriente ✗ Requiere disipador
 KA78R05	<ul style="list-style-type: none"> ✓ Cuenta con protección contra sobrevoltaje, sobrecarga y sobrecalentamiento ✓ Soporta voltajes de entrada un poco más altos ✓ Contiene un pin adicional de habilitación para su activación 	<ul style="list-style-type: none"> ✗ Costo ✗ Requiere disipador ✗ Soporta cargas máximo de 1A

Se han descrito las comparativas que ayudan en la toma de decisiones para la instrumentación y el control. Sin embargo, es necesario considerar las diferentes opciones para la programación de la interfaz que controla el sistema. En el siguiente capítulo se describirán los diferentes elementos que pueden utilizarse para la implementación de la interfaz de usuario.

CAPÍTULO 4.

DISEÑO DE LA INTERFAZ USUARIO-SERVIDOR

La interfaz de usuario se debe implementar de tal manera que permita tanto la comunicación remota con el servidor por parte del usuario vía Internet, como la transmisión de datos entre el sistema y el servidor, entre los que se encuentran la señal de video proveniente de una cámara Web montada sobre el cañón del subsistema de tiro. Dada esta necesidad, será importante que la comunicación con el sistema sea lo más rápida posible y sin fallos, para que no existan problemas al momento de la operación por parte del usuario. Es por tanto, indispensable, seleccionar una arquitectura de red para la interfaz, mediante el uso de aplicaciones que sean compatibles, al menos, con el sistema operativo Windows XP, de acuerdo con lo establecido en el proyecto EN106204 del PAPIME. A continuación se establecen las comparaciones entre los diferentes componentes de la interfaz de usuario-servidor.

4.1 COMPATIBILIDAD CON WINDOWS XP

Se debe programar una aplicación que sea compatible con las computadoras remotas que establezcan la comunicación con el servidor encargado de manipular el sistema. Dado que el objetivo del proyecto es la operación remota por parte de usuarios comunes, se necesita implementar la aplicación Web en Windows. En términos de compatibilidad, Windows XP es el sistema operativo más funcional, ya que una aplicación desarrollada para ejecutarse en dicho sistema puede también manejarse con versiones más avanzadas, como Windows Vista o el reciente Windows 7. Por tanto, el diseño de la interfaz de usuario se enfoca a la búsqueda de aplicaciones y herramientas que sean compatibles con Windows XP.

4.2 COMUNICACIÓN DEL SISTEMA

Ya que muchos de los protocolos descritos anteriormente no son compatibles con Windows XP y dado que algunos sólo se utilizan para la comunicación entre dispositivos que comparten una característica en particular, se puede establecer la comparativa entre tres de los protocolos más populares en cuanto a transmisión de datos se refiere. Dichas formas de comunicación se adaptan totalmente al sistema operativo y cuentan con diferentes ventajas y desventajas para su implementación, como se presenta en la Tabla 4.1.

Tabla 4.1 Comparación entre diferentes maneras de implementar la comunicación.

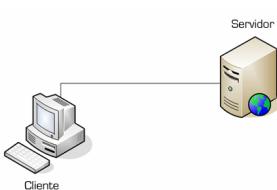
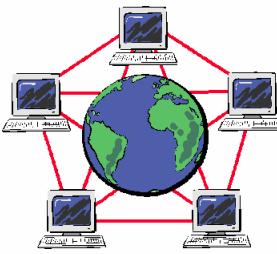
Protocolo	Ventajas	Desventajas
 Universal Serial Bus (USB)	<ul style="list-style-type: none"> ✓ Velocidad ✓ Configuración automática ✓ Tamaño de los conectores ✓ Puede proveer una alimentación de 5V ✓ Facilidad de conexión 	<ul style="list-style-type: none"> ✗ Longitud del cableado ✗ Costo del cableado ✗ Los dispositivos a controlarse requieren de un controlador, el cual los puede limitar a una plataforma en particular y actualización constante
 RS-232	<ul style="list-style-type: none"> ✓ Longitud del cableado ✓ Fácil implementación 	<ul style="list-style-type: none"> ✗ Velocidad de transmisión baja ✗ Susceptible a perder datos ✗ No es compatible con nuevas plataformas ✗ Tamaño
 Puerto paralelo	<ul style="list-style-type: none"> ✓ Velocidad de transmisión ✓ Fácil Implementación ✓ Longitud de cableado 	<ul style="list-style-type: none"> ✗ Necesidad de una tierra estable ✗ El protocolo está en desuso ✗ Tamaño ✗ No es compatible con nuevas plataformas

4.3 ARQUITECTURA DE RED DE LA INTERFAZ

El control del sistema se realiza mediante el servidor, mismo que recibe las peticiones por parte de un usuario remoto, de manera que la comunicación sea usuario con servidor y de servidor a sistema. La comunicación entre los primeros dos es lo que en cuestiones de redes se conoce como arquitectura de red de dos capas.

Para que se pueda manipular la transmisión de datos bajo este modelo, existen dos formas principales de implementar dicha red, ya sea haciendo que el usuario realice peticiones al servidor y la comunicación sea sólo en ese sentido, definida como arquitectura de cliente/servidor, o de forma tal que el servidor también pueda enviar peticiones al ordenador del usuario, lo que se define como red par a par, o P2P por sus siglas en inglés. En la Tabla 4.2 se describen las ventajas y desventajas de implementar alguna de estas dos arquitecturas para la interfaz de control.

Tabla 4.2 Comparación entre arquitecturas cliente/servidor.

Tipo de red	Ventaja	Desventaja
 <p>Cliente-Servidor</p>	<ul style="list-style-type: none"> ✓ Centralización del control ✓ Escalabilidad o que tanto el servidor como el cliente se pueden mejorar por separado ✓ Fácil mantenimiento 	<ul style="list-style-type: none"> ✗ Congestión por el tráfico cuando existen varios clientes ✗ Requiere que el servidor siempre esté en uso para poder funcionar ✗ El software y el hardware del servidor es determinante
 <p>Red par a par (P2P)</p>	<ul style="list-style-type: none"> ✓ Robustez al poderse encontrar los datos en caso de fallar un nodo ✓ La carga de datos está repartida entre usuarios ✓ Escalabilidad ✓ Habilita a todos los nodos como clientes simples 	<ul style="list-style-type: none"> ✗ El nodo puede ser inseguro ante la conexión de uno o varios clientes anónimos ✗ Se debe implementar un control para IP dinámica

4.4 HERRAMIENTAS DE PROGRAMACIÓN

Se cuentan con varias herramientas para la programación compatibles con Windows XP para la aplicación Web y que utilizan diferentes lenguajes, entre los que se encuentran Java, Basic ó C# por mencionar algunos de los más populares para la generación de interfaces. En la Tabla 4.3 se establecen las características de cuatro entornos de desarrollo que facilitan la programación de la interfaz de usuario.

Tabla 4.3 Diferentes herramientas para la generación de la interfaz de control.

Entorno de programación	Ventajas	Desventajas
 Netbeans	<ul style="list-style-type: none"> ✓ Multiplataforma ✓ Documentación ✓ Gratis ✓ Facilidad para la programación Web ✓ Editor de código sofisticado 	<ul style="list-style-type: none"> ✗ No posee interfaz para la creación de botones y similares ✗ Uso de recursos excesivos del sistema ✗ Se requieren complementos para reconocer diferentes lenguajes además de Java
 Visual Studio	<ul style="list-style-type: none"> ✓ Documentación ✓ Facilidad de uso ✓ Integra el diseño e implementación de formularios para Windows 	<ul style="list-style-type: none"> ✗ Requiere licencia para su uso ✗ Espacio en disco ✗ No es multiplataforma ✗ No soporta tratamiento de procesos como parte del lenguaje
 Eclipse	<ul style="list-style-type: none"> ✓ Entorno de desarrollo universal ✓ Documentación ✓ Facilidad para su instalación ✓ Gratis ✓ Multiplataforma 	<ul style="list-style-type: none"> ✗ Interfaz de usuario complicada ✗ Se requieren complementos para reconocer diferentes lenguajes además de Java

4.5 APLICACIONES PARA LA TRANSMISIÓN DE VIDEO

Puesto que la calibración del cañón debe ser visual durante la realización de los experimentos, se deben transmitir señales de video en tiempo real al usuario. Dicha transmisión se puede lograr mediante el montaje de un servidor virtual de video en la máquina destinada a ser el servidor. Cabe mencionar que la velocidad de subida de datos, o *upstream*, debe ser relativamente alta para evitar los retrasos por la gran cantidad de información que requiere el envío de video en tiempo real. Existen diferentes aplicaciones compatibles con Windows XP que permiten la comunicación de video. En la Tabla 4.4 se establece una comparativa entre los diferentes servicios que permiten la implementación del video en tiempo real para el sistema.

Tabla 4.4 Aplicaciones para la transmisión de video.

Aplicación	Ventajas	Desventajas
 Windows Media Server	<ul style="list-style-type: none"> ✓ Fácil implementación en Windows ✓ Amigable con el usuario ✓ Buena compresión de datos 	<ul style="list-style-type: none"> ✗ La transmisión sólo se puede realizar en sistemas operativos Windows ✗ Sólo soporta formatos compatibles con Windows ✗ Costo por el uso del servicio
 VideoLAN (VLC)	<ul style="list-style-type: none"> ✓ La transmisión se puede ser multiplataforma ✓ Montaje de video de fácil implementación ✓ Utiliza diferentes protocolos de transferencia ✓ Fácil implementación con C# ✓ Transmisión gratuita 	<ul style="list-style-type: none"> ✗ Requiere VLC del lado del cliente y del servidor ✗ Puede tener problemas con algunos formatos de video ✗ Carece de algunas características y opciones para la reproducción de video
	<ul style="list-style-type: none"> ✓ Se puede utilizar con varios reproductores de video ✓ Operación de 	<ul style="list-style-type: none"> ✗ La transmisión sólo se puede realizar en sistemas operativos Windows

	<ul style="list-style-type: none"> ✓ transmisión automatizada ✓ No requiere encoder, cambia los formatos de video al transmitir 	<ul style="list-style-type: none"> ✗ Sólo soporta formatos de video compatibles con Windows ✗ Costo por el uso del servicio
	<ul style="list-style-type: none"> ✓ Multiplataforma ✓ Soporta diferentes formatos de video ✓ Amigable con todos los tipos de usuario ✓ Gratuito 	<ul style="list-style-type: none"> ✗ Se tarda en cargar ✗ Necesita gran parte de la memoria del procesador

CAPÍTULO 5

SELECCIÓN Y ESPECIFICACIÓN

Una vez conocidas las diferentes herramientas y alternativas que permiten la implementación y el control para del sistema de tiro parabólico y caída libre, se pueden seleccionar aquéllas que sean más adecuadas desde el punto de vista ingenieril. En este capítulo se detalla la toma de decisiones y la manera en que se ha constituido el sistema, atendiendo por separado su mecánica, electrónica y programación.

5.1 INSTRUMENTACIÓN MECÁNICA DEL SISTEMA

El desarrollo de los diferentes dispositivos empleados para el funcionamiento del sistema se presenta a continuación. Para comprender más a detalle la manera en que se manufacturaron y construyeron los instrumentos, se pueden consultar los planos de cada una de las piezas fabricadas y modificadas que se muestran en el Apéndice A.1.

5.1.1 Tensión de la cuerda

5.1.1.1 Cambio de cuerda

El prototipo del cañón contaba con un hilo de caña de pescar, el cual se rompía en un determinado valor de la tensión, impidiendo ajustar la potencia de tiro, lo cual imposibilitaba el funcionamiento del subsistema de tiro. Debido a esta situación, se decidió cambiar dicho hilo por una cuerda utilizada en aeronáutica, como la que se muestra en la Figura 5.1. Una vez realizado este cambio, se volvieron a realizar pruebas, y en esta ocasión no se rompió la cuerda.



Figura 5.1 Cuerda utilizada para mover alerones de avión.

5.1.1.2 Conteo de vueltas

Se optó por utilizar un encóder rotatorio incremental consistente en un disco de aluminio con una sola perforación como el mostrado en la Figura 5.2, de tal manera que se puedan contar las vueltas del motor, y por ende la tensión de la cuerda, mediante la detección del paso de luz utilizando un optointerruptor H21A1.



Figura 5.2 Contador de vueltas.

Además de poderse construir de manera sencilla, el encóder incremental permite hacer discreto el valor de la tensión bajo un esquema de retroalimentación muy simple, como el que se presenta en la Figura 5.3, permitiendo tener un método muy estable para conocer la potencia de tiro, evitando una etapa de calibración más rigurosa; simplemente se puede hacer uso de una rutina de comparación, misma que se puede cambiar rápidamente, dependiendo de los ajustes que sufra la cuerda o el sistema, lo cual lo hace un instrumento de fácil mantenimiento.

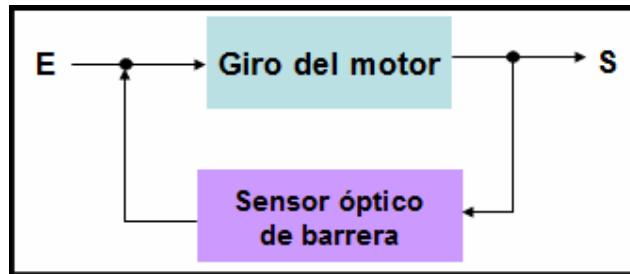


Figura 5.3 Esquema de control para el conteo de vueltas.

5.1.2 Amplificación angular

Dado el costo y la precisión que se requería, se decidió hacer uso de dos engranes que permitieran tener una resolución angular de 0.5° . Se acopló un potenciómetro al eje del piñón, de tal manera que el valor del ángulo sea proporcional a un voltaje determinado para su posterior procesamiento. En la Figura 5.5 se muestra el instrumento desarrollado para la medición del ángulo de giro del cañón.

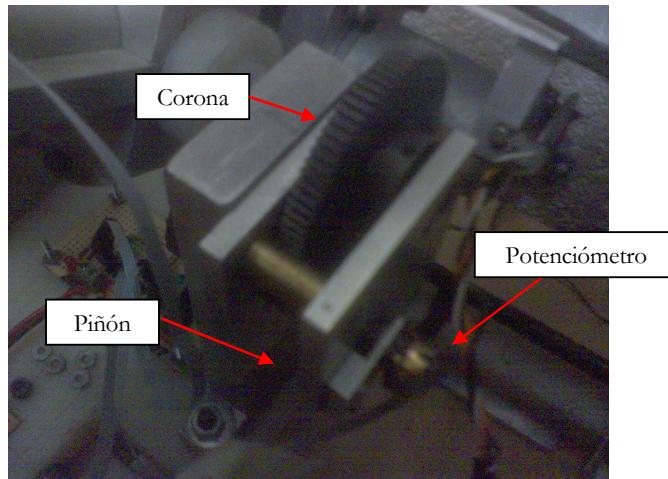


Figura 5.5 Mecanismo de amplificación angular.

5.1.3 Detección de pelota

Se fresó con un cortador de $1/8"$ una ranura con las dimensiones del sensor óptico QRD1114 en la boca del cañón para que dicho sensor entre de manera muy ajustada y no se pueda mover mientras se encuentre colocado. Dicho sensor se soporta mediante una lámina que también se atornilla en la boca del cañón, como se muestra en la Figura 5.6.



Figura 5.6 Ranura y sensor para la detección de la carga.

La manufactura se realizó en la parte superior de la boca del cañón, para que no se afectara la trayectoria del proyectil al momento de ser disparado, ya que la ranura podría ocasionar interferencia física. Además, la colocación de la placa no obstruye el campo visual de la cámara Web instalada en la parte superior del cañón.

5.1.4 Robustecimiento de las entradas laterales

Para fijar los sensores que determinan la posición de carga y tiro del cañón, se manufacturaron dos láminas simétricas entre sí, que permitieran el ajuste de dichos sensores a las piezas laterales del cañón. La manera en que se colocaron dichas piezas se muestra en la Figura 5.7.



**Figura 5.7 Sensores laterales del cañón
(sensor lateral izquierdo y derecho, respectivamente).**

5.1.5 Manufactura de la unidad de control

La unidad de control es la caja que contiene el sistema de control de entradas y salidas así como las fuentes encargadas de su alimentación y un ventilador para mantener a los circuitos a una temperatura adecuada. La disposición de dichos elementos se muestra en la Figura 5.9.

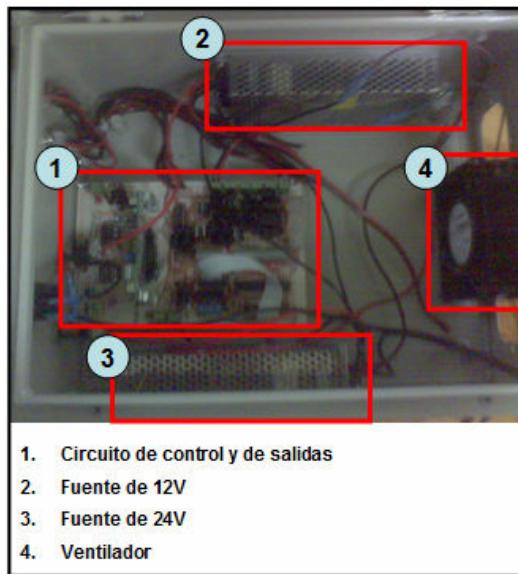


Figura 5.9 Disposición de elementos dentro de la unidad de control.

5.1.5.1 Colocación de conectores

Para manipular las conexiones de entrada, la unidad de control cuenta con un conector DB-9 que recibe las señales provenientes del cañón, y un orificio por el cual salen los cables que se conectaron a los sensores de final de carrera del soltador, como se observa en la Figura 5.10.



Figura 5.10 Conexiones externas de la unidad de control.

Las conexiones de salida se realizan mediante conectores universales macho de dos entradas para los motores y solenoides, y un conector de cinco entradas para el motor de pulsos. Además, la caja tiene colocados un interruptor ON/OFF para el encendido y apagado de todo el sistema y dos fusibles correspondientes a cada fuente de voltaje que se encuentra en el interior.

5.1.5.1 Posicionamiento de la unidad de control

Se analizaron diferentes posiciones para ubicar la caja en términos de cableado y facilidad de colocación. La ubicación óptima se presenta en la Figura 5.11, ya que las distancias de cableado para el subsistema de tiro y del soltador son menores, y la colocación de la caja se puede realizar atornillando la parte inferior a la armadura del sistema.



Figura 5.11 Posición de la unidad de control.

5.2 CONTROL ELECTRÓNICO DEL SISTEMA

Para el control de *actuadores* a partir de los sensores del sistema, se diseñó y construyó un circuito de control cuyas etapas se describen a continuación. Para analizar más a profundidad el funcionamiento de un circuito o fase en particular, se pueden consultar los diagramas electrónicos esquemáticos del Apéndice A.2.

5.2.1 Disposición de los elementos del sistema

El sistema de tiro parabólico y caída libre se encuentra organizado electrónicamente de la manera que se muestra en la Figura 5.12.

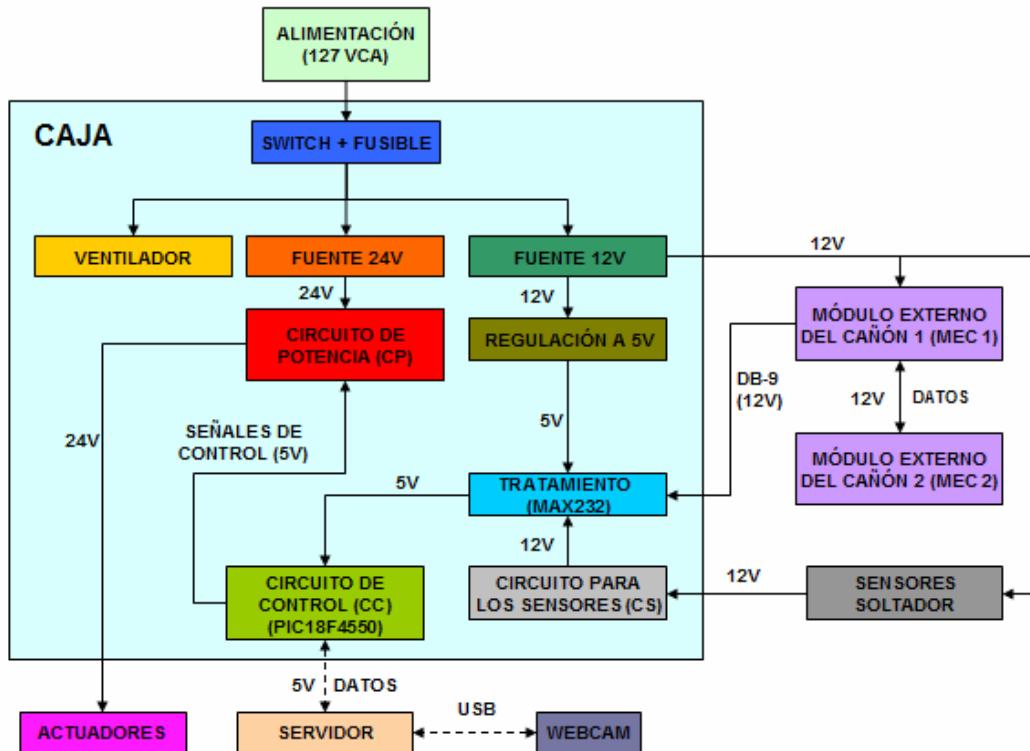


Figura 5.12 Disposición de elementos del sistema.

En general, el control electrónico se realizó mediante tres etapas: entrada, procesamiento y salida. Para el manejo de entradas se utilizaron los sensores de los subsistemas del soltador y del cañón, mismas que se tratan y se procesaron para que finalmente accionen un *actuador* en particular.

5.2.2 Módulo externo del cañón, MEC

Dada la posición de la caja que contiene a los circuitos de control y de potencia, descritos posteriormente, fue necesario construir el circuito de las entradas del subsistema de tiro por separado. Dicho circuito se denominó módulo externo del cañón, o MEC por sus siglas y se muestra en la Figura 5.13.



Figura 5.13 Módulos externos del cañón (izq. MEC1, der. MEC2).

El MEC consiste de dos unidades: la segunda unidad, o MEC2, contiene los circuitos de los sensores de detección del trinquete, de detección de presencia de la pelota, del amplificador angular y del encóder incremental rotatorio, con la finalidad de que las señales de respuesta de los sensores que se encuentran en el cañón, se concentren en un punto central. De dicho punto parten dichas señales a la primera unidad, o MEC1, que se encarga de recibir las señales de los dos sensores laterales del subsistema de tiro, y que no se encuentran específicamente en el cañón.

En total, el cañón cuenta con seis señales. Para enviar las señales del MEC2 a la unidad del control, se utilizaron conectores DB-9 y un cable de red, de manera que se tuvieran las seis señales y otras tres terminales de tierra que brindaran estabilidad en la transmisión de datos.

Finalmente, para evitar que se pudiera perder el valor de la señal de la respuesta por la longitud del cableado, los sensores del MEC se alimentaron con 12 V de tal manera que las señales llegaran inalteradas a la unidad de control. Es importante mencionar que los sensores de final de carrera del subsistema del soltador también se alimentaron a la misma tensión.

5.2.3 El circuito de control

Esta parte del circuito se encarga del procesamiento del sistema y de su comunicación con el servidor. Se eligió un microcontrolador PIC18F4550 como la unidad de procesamiento, dado su bajo costo y facilidad de implementación, además de que es portátil y permite la construcción de un circuito que pueda recibir mantenimiento con facilidad. Cabe mencionar que su programación es muy sencilla de realizar.

Este microcontrolador se encuentra colocado en una tarjeta diseñada especialmente para el mismo, como se presenta en la Figura 5.14. El colocar este dispositivo en dicha tarjeta permite que la unidad de control se ajuste al circuito a manera de ranura de memoria de acceso aleatorio en una computadora ordinaria.



Figura 5.14 Tarjeta de desarrollo UPRSYS v0308 con PIC18F4550 y conexión USB.

Además, la selección de este microcontrolador en particular tiene que ver con el uso del protocolo USB, mismo que fue seleccionado entre los protocolos descritos en el capítulo anterior por su velocidad de transmisión, adaptabilidad a las computadoras actuales, así como la capacidad de suministrar el voltaje necesario para el funcionamiento del microcontrolador, lo cual simplificó la implementación de una etapa de regulación para el mismo. Cabe mencionar que se contó con un controlador previamente diseñado [4] para Windows XP, que permitió el uso del USB con el microcontrolador PIC18F4550 [8] mediante la programación de interfaces con C#, que se detallará posteriormente.

5.2.3.1 Tratamiento de la señal de entrada

Para utilizar el microcontrolador de acuerdo con las condiciones de operación especificadas por el fabricante, es necesario tratar las señales de entrada del subsistema del soltador y de tiro. Como se describió previamente, los sensores de dichos subsistemas se alimentan con 12 V, voltaje que no es soportado en los puertos del microcontrolador, el cual requiere valores TTL para el procesamiento de señales lógicas. Por tanto, el empleo del circuito integrado MAX232 permite reducir el valor de la señal de 12 V a un valor TTL correspondiente al “0” lógico, ya que las salidas de dicho circuito son negadas. En caso contrario el integrado puede detectar hasta valores mínimos de 3.4 V. En la Figura 5.15 se muestra la ubicación física de los integrados en el circuito de control.

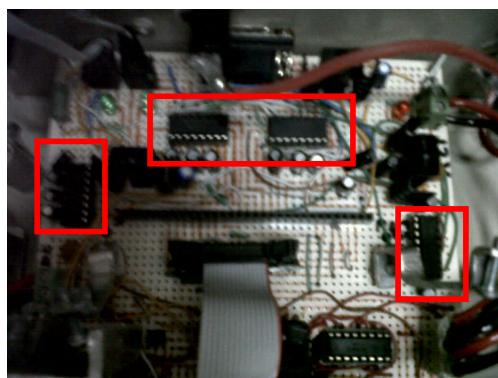


Figura 5.15 Tratamiento de entradas para el circuito de control.

Para alimentar los MAX232 se empleó una etapa de regulación consistente en un circuito KA78R05 cuya entrada se encuentra conectada a una resistencia de disipación que recibe 12 V de una fuente de alimentación. Cabe mencionar que por cuestiones de practicidad, se colocaron los circuitos encargados de alimentar y recibir las respuestas de los sensores del soltador cerca del circuito de control.

5.2.3.2 Señales de control

Es necesario crear una nomenclatura para las señales que el microcontrolador maneja. En la Tabla 5.1 se describen las abreviaturas para todas las señales de control y sus características.

Tabla 5.1 Descripción de las señales de control.

NOM.	DESCRIPCION	TIPO (E/S)	ORIGEN	DESTINO	ELEMENTO DE CONTROL
SSI	SENSOR SOLTADOR IZQ.	E	CS	CC	<i>H21A1</i>
SSD	SENSOR SOLTADOR DER.	E	CS	CC	<i>H21A1</i>
EC	ENCODER DEL CAÑÓN	E	MEC2	MEC1	<i>H21A1</i>
DP	DETECCIÓN DE PELOTA	E	MEC2	MEC1	<i>QRD114</i>
DT	DETECCIÓN DE TRINQUETE	E	MEC2	MEC1	<i>MICROSWITCH</i>
AA	AMPLIFICADOR ANGULAR	E	MEC2	MEC1	<i>POT-LINEAL 50KΩ</i>
ILI	INTERRUPTOR LATERAL IZQ.	E	MEC1	CC	<i>MICROSWITCH</i>
ILD	INTERRUPTOR LATERAL DER.	E	MEC1	CC	<i>MICROSWITCH</i>
ST	SOLENOIDE DEL TRINQUETE	S	CC	CP	<i>MOSFET CANAL N</i>
MLI	MOTOR MOV. LATERAL IZQ.	S	CC	CP	<i>P-H MOSFET</i>
MLD	MOTOR MOV. LATERAL DER.	S	CC	CP	<i>P-H MOSFET</i>
MAI	GIRO ANGULAR IZQ.	S	CC	CP	<i>L293D (OUT1)</i>
MAD	GIRO ANGULAR DER.	S	CC	CP	<i>L293D (OUT2)</i>
MS1	MOTOR SOLTADOR 1	S	CC	CP	<i>L293D (OUT1)</i>
MS2	MOTOR SOLTADOR 2	S	CC	CP	<i>L293D (OUT2)</i>
MS3	MOTOR SOLTADOR 3	S	CC	CP	<i>L293D (OUT3)</i>
MS4	MOTOR SOLTADOR 4	S	CC	CP	<i>L293D (OUT4)</i>
SS	SOLENOIDE DEL SOLTADOR	S	CC	CP	<i>MOSFET CANAL N</i>
MT	TENSIÓN DE CUERDA	S	CC	CP	<i>MOSFET CANAL N</i>
AB	ACCIONAMIENTO DE LA BANDA	S	CC	CP	<i>MOSFET CANAL N</i>

5.2.4 El circuito de potencia

El circuito de potencia está encargado de recibir las señales de salida del microcontrolador y accionar los *actuadores* dependiendo de aquéllas. Dicho circuito consiste de una etapa de prueba, etapa recepción y transmisión de señales, etapa de potencia y conexiones a los conectores de la unidad de control; su ubicación física se muestra en la Figura 5.16.

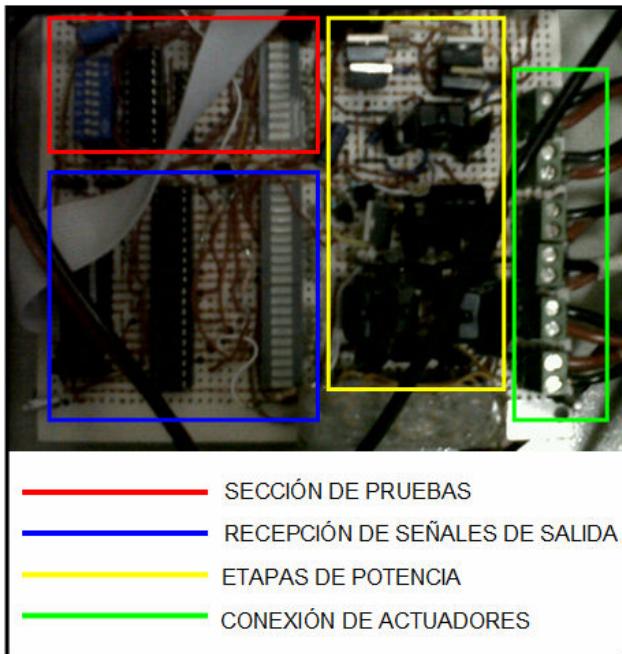


Figura 5.16 Componentes del circuito de potencia.

Si en algún momento se desea probar la etapa de potencia sin hacer uso del microcontrolador, se puede emplear la etapa de prueba, que cuenta con un DIPSWITCH que simula dichas señales. Esta parte del circuito también cuenta con leds indicadores de estado.

La etapa de recepción de señales de salida se compone de dos circuitos integrados 74LS245 cuya función es la de proteger a los dispositivos que interactúan con la etapa de potencia de posibles corrientes no deseadas o en caso de que dicha etapa fallara. Al igual que la etapa de pruebas, se cuenta con leds indicadores para conocer el estado de las salidas mientras se esté dando mantenimiento al sistema. Es importante mencionar que se utilizó una etapa de regulación similar a la del circuito de control para alimentar a los integrados 74LS245 así como algunos elementos de la etapa de potencia.

La etapa de potencia es la que activa los *actuadores* de todo el sistema. Dicha etapa se implementó en función de los requerimientos de control y condiciones de operación de los *actuadores*.

Para los solenoides del soltador y del cañón, así como el motor de tensión de la cuerda y de accionamiento de la banda, se empleó un arreglo consistente en un transistor MOSFET y un transistor bipolar de juntura BC548, como se puede apreciar en los diagramas esquemáticos del Apéndice A.2. Dicho arreglo permite la alimentación de dichos *actuadores* a 12 V mediante una fuente de alimentación.

Para el motor de giro angular, dado que no requiere mucha corriente para su operación se utilizó un puente H, circuito L293D que alimenta a dicho motor con 1 V y se puede manipular mediante dos señales de salida provenientes del microcontrolador. Para alimentar el circuito integrado del puente H, se utilizó el regulador que también alimenta a los integrados 74LS245 de la etapa de recepción de señales de salida.

El motor de pulsos requiere 3.3 V a 2.6 A para operar. Para manejarlo se emplea otro circuito integrado L293D. Este puente H recibe las señales de salida del microcontrolador y las salidas del integrado se dirigen a las bases de cuatro transistores TIP125 configurados como interruptores. Estos transistores se encuentran conectados cada uno a un voltaje de 3.3 V en el colector. Para poder suministrar esta tensión, se utilizaron cuatro reguladores KA78R33. El circuito funcionó de manera tal, que cada vez que se acciona una señal de salida, el L293D activa el transistor correspondiente para que se alimente dicha terminal del motor de pulsos.

El motor de movimiento lateral del cañón requirió 24 V a 4.2 A pico para funcionar. Dado que se requiere controlar el sentido de giro de dicho motor, y puesto que ni el L293D ni el L298N soportan dicha corriente, se implementó un puente H con transistores MOSFET.

Finalmente, las salidas de dicha etapa de potencia se conectaron a bornes dobles que permitieron la conexión de las terminales que se encuentran en la parte exterior de la caja de control.

5.2.5 Cableado del sistema

Para alimentar a los sensores y *actuadores* de los subsistemas, se utilizaron diferentes conectores en función de los dispositivos que alimentan. Para la conexión de los sensores se emplearon *headers* mientras que los *actuadores* emplearon conectores universales de acuerdo con el número de terminales que requirieron. En la Figura 5.17 se muestran los conectores empleados.



Figura 5.17 Conectores del sistema (izq. entradas, der. salidas).

Se utilizó cable plano para las conexiones con los sensores del soltador, mientras que todos los *actuadores* son conectados mediante cable dúplex calibre 18. Para conocer la cantidad total de

cada tipo cable que se utilizó para alambrar el sistema así como el total de conectores, incluyendo los de la unidad de control, se puede consultar el Apéndice A.3.

5.2.6 Algoritmo de control y programación del microcontrolador

5.2.6.1 Posición de HOME

Para programar adecuadamente al sistema, es necesario establecer una posición de inicio o HOME, la cual se define por tener ninguno o el menor número de elementos activos. Para el sistema de tiro parabólico y caída libre se estableció que la posición inicial del sistema será con el soltador y el cañón en sus posiciones de carga y el motor de la banda desactivado. Para determinar que el sistema se encuentre en el origen, se recurre a la detección de los estados de todos los sensores, siguiendo la nomenclatura expuesta previamente, como se muestra en la Tabla 5.2.

Tabla 5.2 Estado de los sensores en la posición de HOME.

SENSOR	ESTADO
SSI	0
SSD	1
EC	*
DP	1
DT	1
AA	Valor del ángulo de carga
ILI	1
ILD	0

5.2.6.2 Algoritmo de control

Una vez establecida la posición de inicio, se describe el funcionamiento del sistema, para lo cual se puede recurrir al diagrama de flujo que se muestra en el Figura 5.18

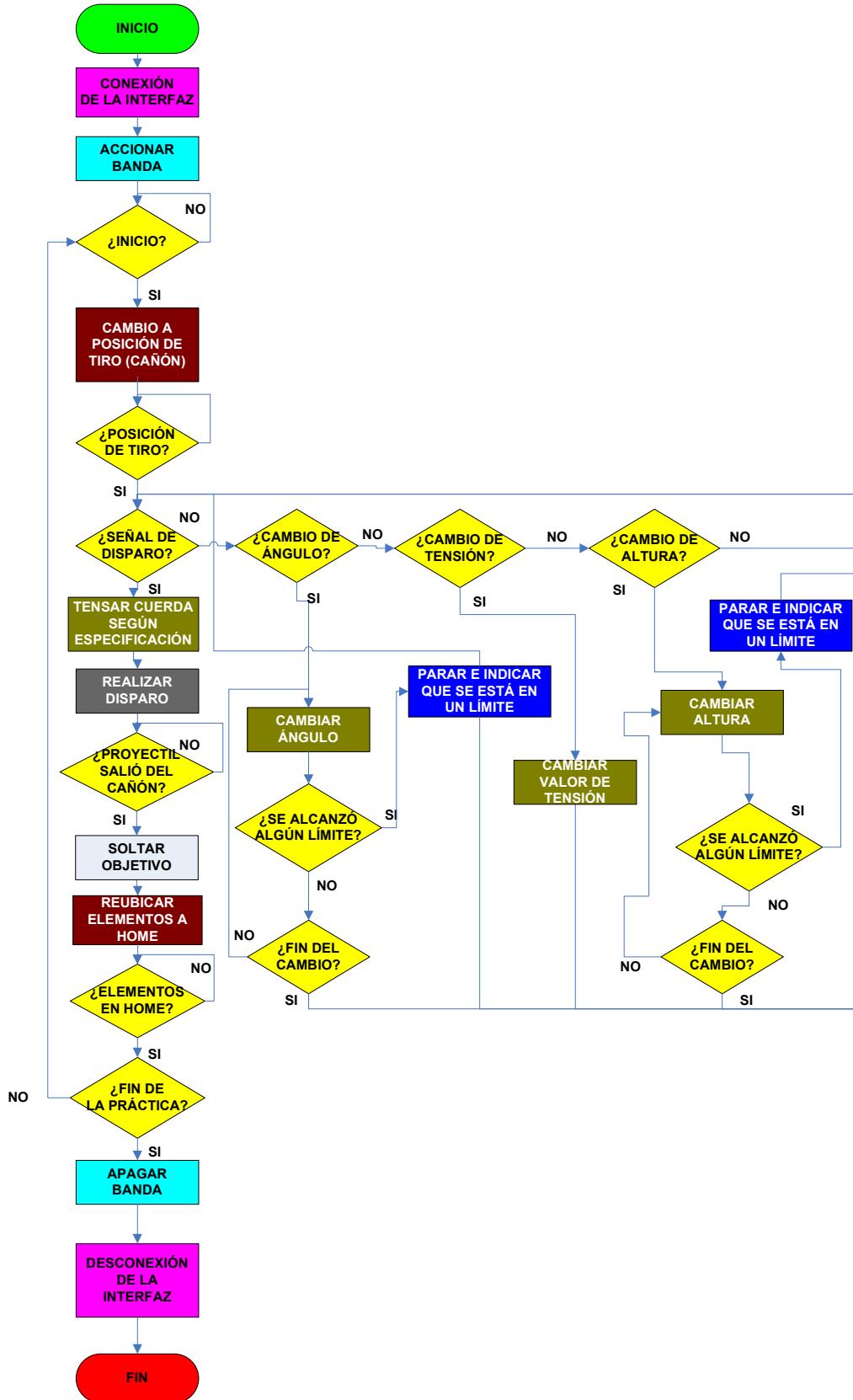


Figura 5.18 Diagrama del algoritmo de control del sistema.

Una vez que se haya inicializado el programa, se deberá realizar la conexión del sistema con el servidor. Una vez establecida la comunicación, se procede a accionar el motor de la banda de manera que éste se encuentre activo durante la realización de la práctica.

Posteriormente, se debe reconocer si el sistema está en su posición de HOME para proceder con el experimento. El sistema esperará una señal de inicio para cambiar el cañón a su posición de tiro y permitir el ajuste de las variables de tensión de la cuerda del cañón, ajuste de ángulo de giro y cambio en la altura del soltador. Cabe mencionar que el ángulo de tiro se puede modificar al mismo tiempo que la posición del soltador, con la ayuda de la cámara web montada en la base del cañón. Dado que no se puede visualizar el cambio en la tensión de la cuerda del cañón, el ajuste de dicha tensión se realizará una vez que se haya dado la señal de disparo.

Cuando se haya detectado que el proyectil ha salido de la boca del cañón, el soltador activará el solenoide encargado de hacer caer el objetivo. El sistema regresa a HOME y detectará si es el fin del experimento para desconectarse y apagar el motor de la banda. En caso contrario, reiniciará el proceso hasta que se termine el tiempo para realizar el experimento que será de quince minutos por persona.

5.2.6.3 Programación del microcontrolador

El microcontrolador se puede programar ya sea mediante el lenguaje ensamblador con las instrucciones y el compilador que ofrece el fabricante, o bien, mediante un lenguaje de alto nivel similar a C. Para el sistema se ha programado al microcontrolador con el lenguaje CCS C utilizando el compilador PCWH [6], cuyo entorno se muestra en la Figura 5.19.

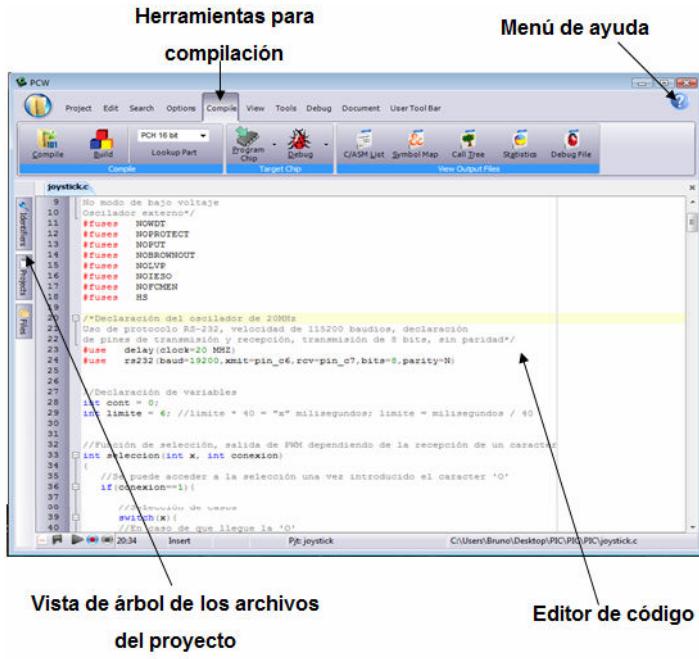


Figura 5.19 Compilador PCWH y lenguaje CCS C.

Dicho lenguaje permite configurar el microcontrolador mediante instrucciones y funciones de alto nivel similares a C. Además, la comunicación USB con el servidor es mucho más sencilla de establecer con este lenguaje. Para conocer el programa encargado de controlar al sistema, se puede consultar el Apéndice A.4, tomando como referencia la tabla de entradas y salidas en función de los puertos del microcontrolador y la nomenclatura de las señales previamente definida, como se establece en la Tabla 5.3.

Tabla 5.3 Tabla de entradas y salidas en función de los puertos del microcontrolador

Señal	Tipo	Ubicación
SSI	E	PB.4
SSD	E	PB.5
EC	E	PB.3
DP	E	PB.2
DT	E	PB.6
AA	E	PE.0
ILI	E	PB.0
ILD	E	PB.1
ST	S	PD.6
MLI	S	PD.0
MLD	S	PD.1

MAI	S	PD.2
MAD	S	PD.3
MS1	S	PA.0
MS2	S	PA.1
MS3	S	PA.2
MS4	S	PA.3
SS	S	PD.4
MT	S	PD.5
AB	S	PC.0

5.3 Interfaz usuario-servidor

El vínculo para la conexión entre el servidor y el sistema se establece mediante el protocolo USB como se ha mencionado anteriormente. El servidor abre el puerto de conexión TCP/IP, como se detallará posteriormente, para que pueda iniciar a recibir peticiones de una interfaz gráfica de usuario remoto, como se presenta en la Figura 5.20.

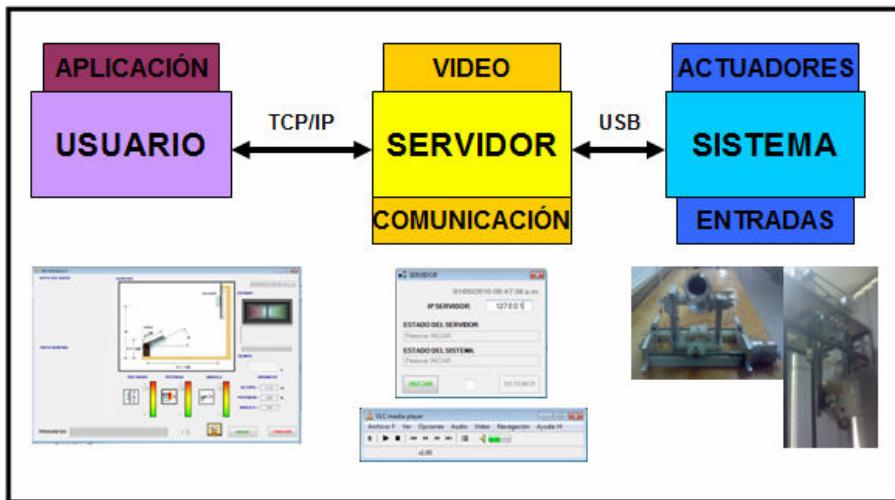


Figura 5.20 Funcionamiento de la interfaz usuario-servidor

5.3.1 Implementación de la interfaz

Para programar las interfaces que permiten la interacción gráfica con el usuario, tanto remoto como del servidor, se ha optado por utilizar el lenguaje C# dado que ya se cuenta con una forma sencilla de establecer la comunicación USB con el microcontrolador. Para desarrollar interfaces gráficas de usuario se utilizó el entorno de programación Microsoft Visual Studio 2005, mismo que genera aplicaciones totalmente compatibles para Windows XP y que facilita la programación

de dichas interfaces dada su rápida curva de aprendizaje. En la Figura 5.21 se presenta el entorno de programación así como la descripción de los componentes más importantes.

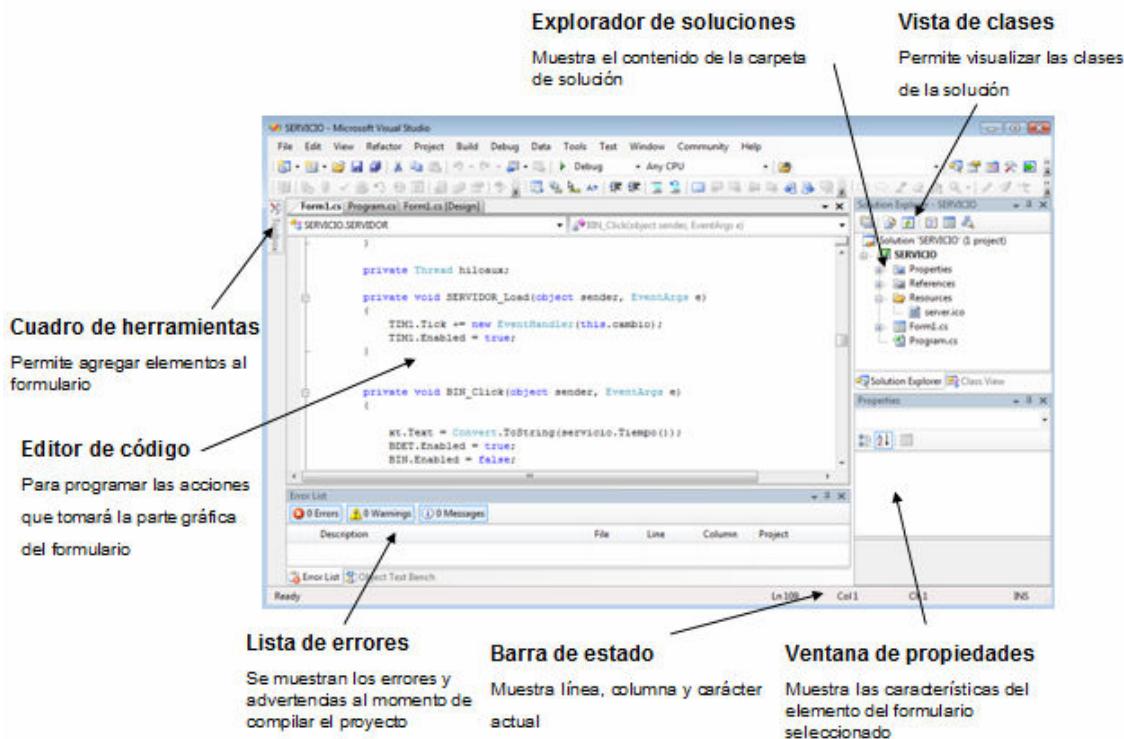


Figura 5.21 Ambiente de desarrollo de Microsoft Visual Studio.

El código utilizado para desarrollar toda la interfaz usuario-servidor, teniendo por separado la interfaz de usuario y la interfaz de servidor, se puede consultar a detalle en el Apéndice A.5.

5.3.1.1 Aplicación de usuario

La interfaz gráfica de usuario se encarga de proveer al usuario remoto con las señales de vídeo del sistema así como las indicaciones específicas del experimento. Además, le permite ajustar las diferentes variables inherentes al experimento, de manera que le permita interactuar con el mismo. A continuación se detallan cada uno de los componentes de la interfaz de acuerdo con la Figura 5.22.

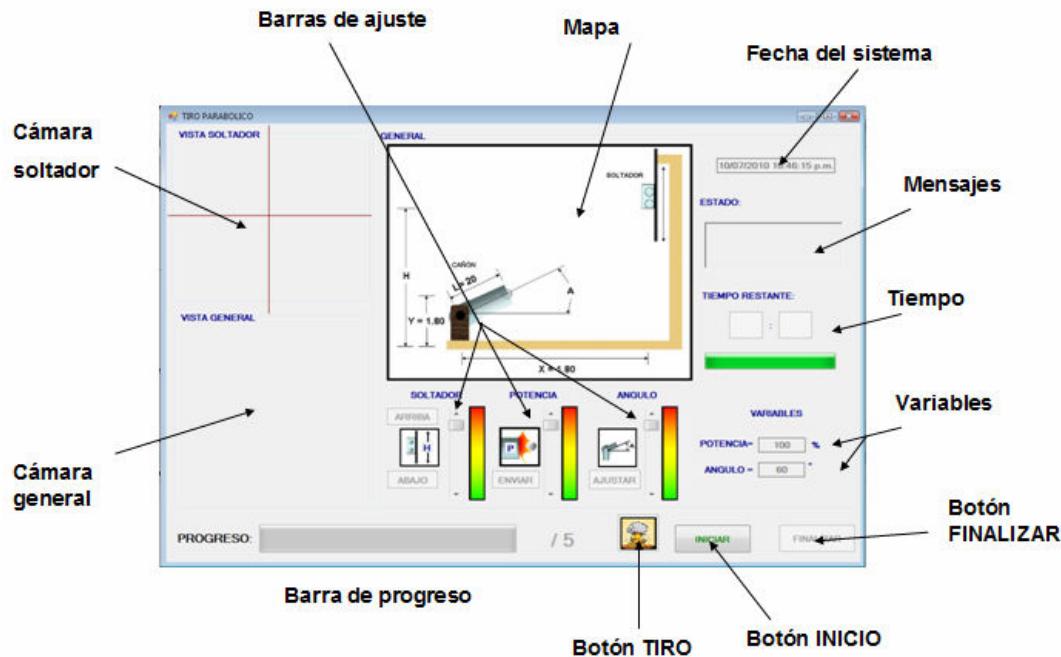


Figura 5.22 Interfaz gráfica de usuario.

- ❖ **Cámara de la mira del cañón.** Permite visualizar la señal de vídeo proveniente de la cámara web montada en la base del cañón.
- ❖ **Cámara general.** Muestra la señal de vídeo proveniente de la cámara IP montada en el laboratorio remoto y que presenta el sistema en general.
- ❖ **Barra de progreso.** Indica el número de lanzamiento en que se encuentra el usuario remoto.
- ❖ **Botón TIRO.** Se presionará dicho botón una vez que estén ajustadas las variables
- ❖ **Botón INICIO.** Se activa sólo al inicio del experimento e indica el inicio de la operación del sistema mediante la conexión con el servidor.
- ❖ **Botón FINALIZAR.** Se activa al final del experimento y permite la desconexión del servidor y cierre de la aplicación.
- ❖ **Variables.** Se indican los valores de la potencia y del ángulo de tiro del cañón, en función de los valores establecidos en las barras de ajuste.
- ❖ **Tiempo.** Muestra el tiempo restante para efectuar los lanzamientos, una vez iniciado el experimento.
- ❖ **Mensajes.** Indica el estado del sistema.
- ❖ **Fecha del sistema.** Indica la fecha de la realización del experimento.
- ❖ **Mapa.** Presenta la forma física del sistema así como etiquetas con los valores de las variables que se pueden ajustar.
- ❖ **Barras de ajuste.** Barras que pueden modificar los valores de altura del soltador, potencia y ángulo de giro del cañón dentro de los valores aceptables para el experimento.

Para comprender el detalle de funcionamiento de la aplicación, se puede observar la Figura 5.23, en el que se muestra un diagrama con los diferentes procesos que se realizan durante la ejecución e interacción con la interfaz.

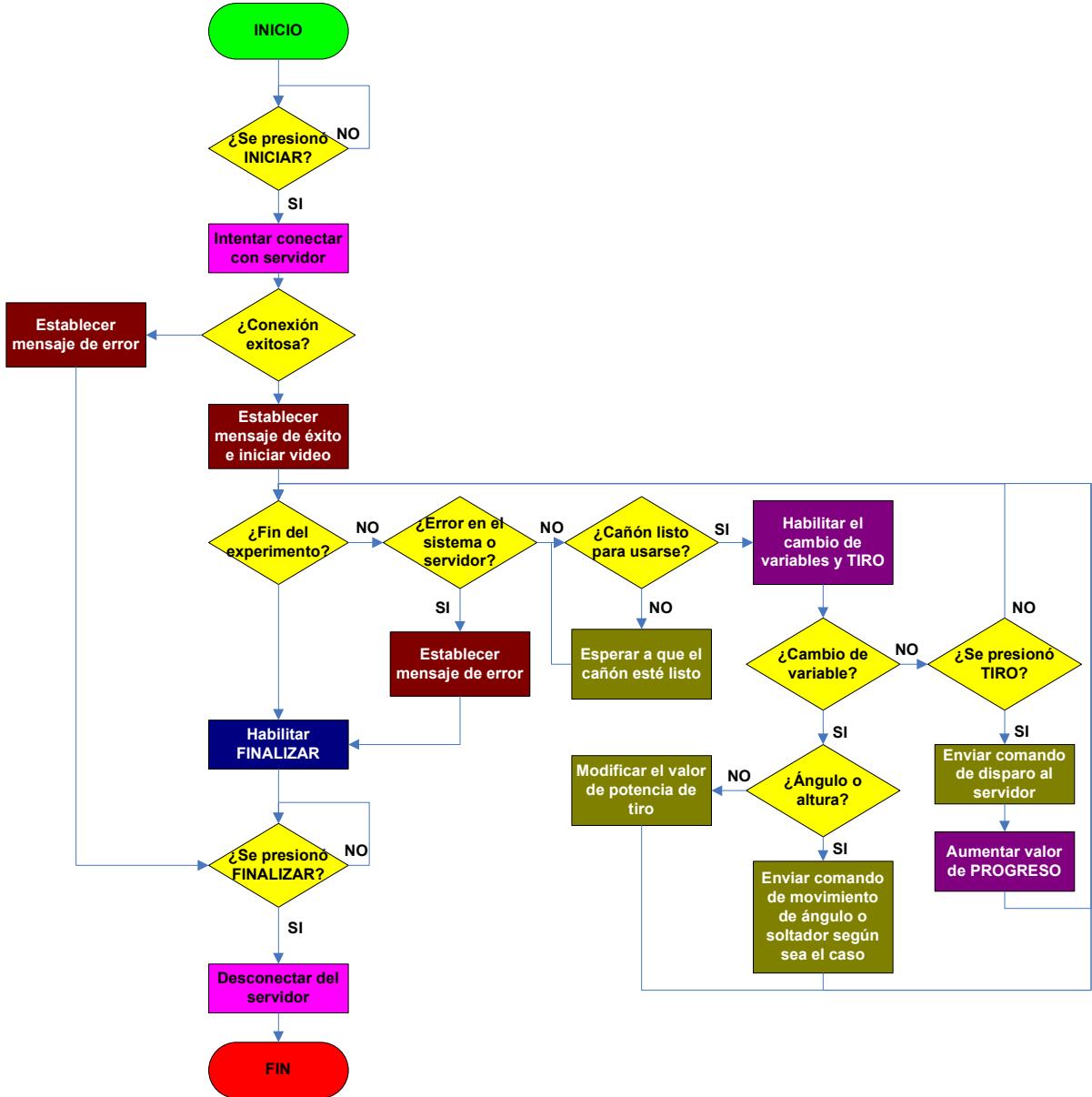


Figura 5.23 Diagrama de funcionamiento de la interfaz de usuario.

5.3.1.2 Aplicación de servidor

La interfaz de servidor se encarga de iniciar la conexión del sistema, así como establecer la comunicación a través de un socket mediante el puerto 8080 de la dirección IP que se especifique. Se puede ver el estado en que se encuentra el mismo así como el estado del

sistema, lo cual le puede indicar la posición en la que se encuentra y así enviar su respuesta a la aplicación de usuario. Atendiendo a la Figura 5.24 se describen los elementos de dicha interfaz.

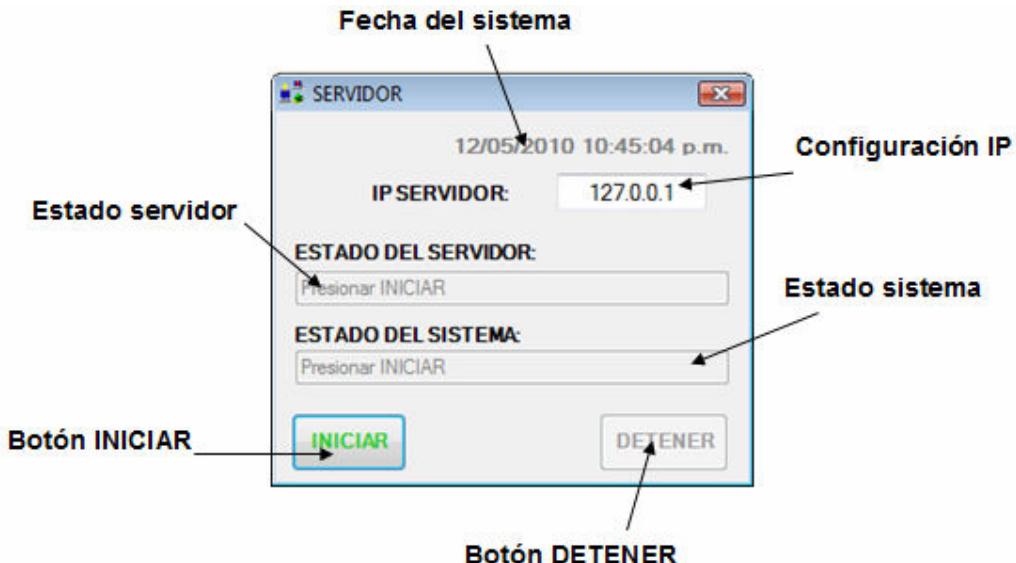


Figura 5.24 Interfaz gráfica de servidor.

- ❖ **Fecha del sistema.** Indica la fecha en que se encuentra operando el servidor.
- ❖ **Configuración IP.** Permite introducir la dirección IP del servidor, de manera que espere la conexión desde dicha IP por el puerto 8080.
- ❖ **Estado servidor.** Indica si se encuentra esperando una conexión del cliente, si se encuentra atendiendo a uno, o si el cliente se desconectó. Se podrá conocer el estado del servidor una vez que se haya iniciado el mismo.
- ❖ **Estado sistema.** Muestra si el sistema se encuentra activo, así como el estado o posición en el que se encuentra. En caso de algún error del sistema, se podrá conocer dicho error cuando sea enviado al servidor.
- ❖ **Botón INICIAR.** Permite iniciar el servidor por el puerto 8080 en la IP especificada.
- ❖ **Botón DETENER.** Finaliza el servidor en la IP introducida, lo que permite terminar todo tipo de conexión en caso de que existieran problemas con el sistema, o con el usuario remoto.

En la Figura 5.25 se muestra un diagrama en el que se indican los diferentes procesos que realiza el programa del servidor.

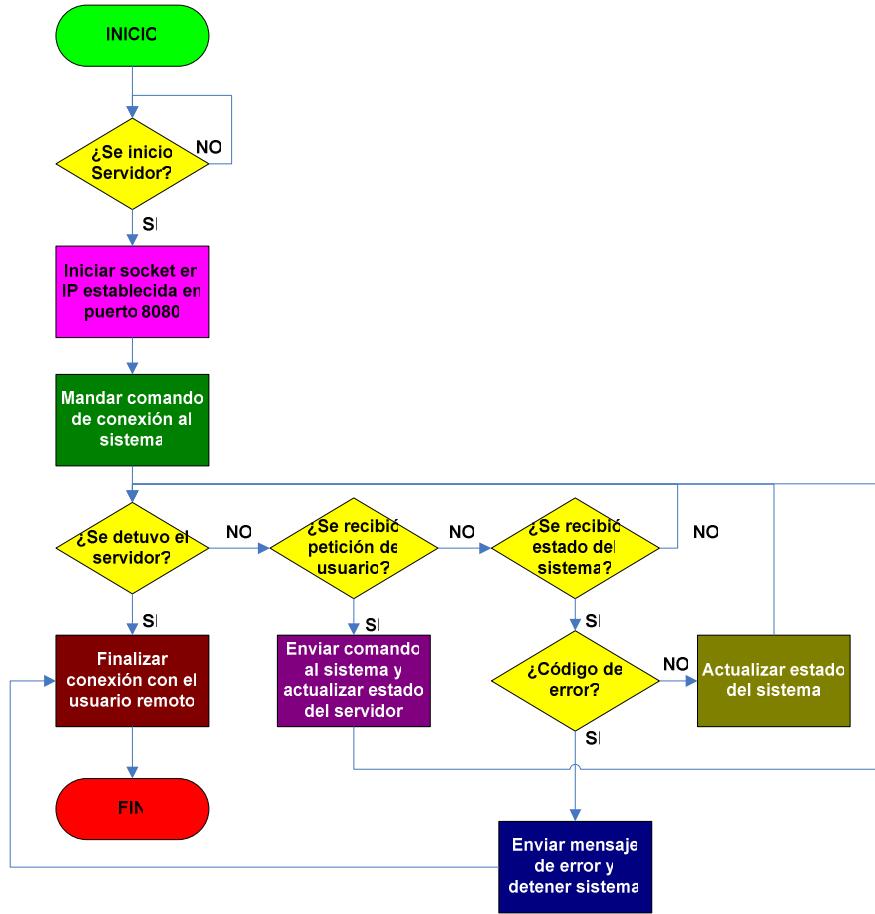


Figura 5.25 Diagrama de funcionamiento de la interfaz de servidor.

5.3.2 CONEXIÓN A INTERNET

Para que la interfaz de usuario se pueda conectar con el servidor y enviarle peticiones, se utilizan los *sockets* [13], como se ejemplifica en la Figura 5.26, es una conexión virtual a través de alguno de los puertos con los que cuenta una dirección IP, y que permite la transmisión y recepción de datos.

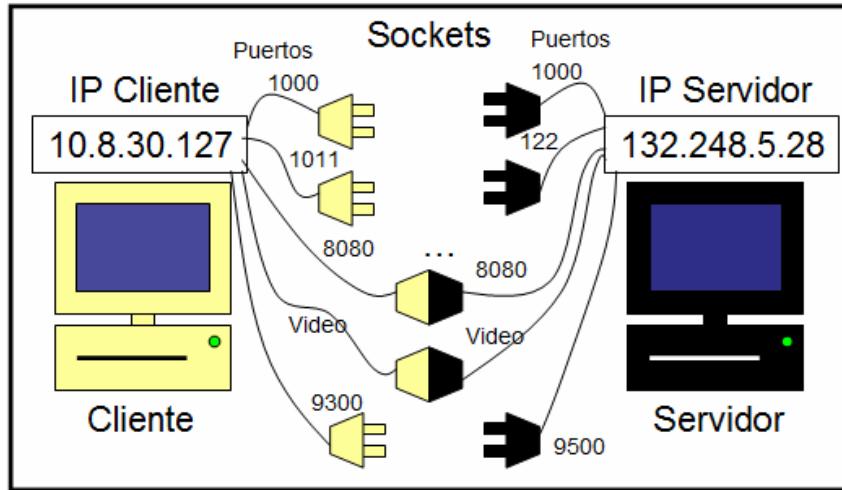


Figura 5.26 Funcionamiento de los sockets.

Para la interfaz de usuario-servidor se utiliza la arquitectura cliente-servidor, por sus condiciones de seguridad ya que sólo permite la operación del sistema de tiro parabólico y caída libre por parte del servidor, eliminando posibles intrusiones al mismo y que lo pudieran dañar. Dado que el programa de servidor se encarga de la comunicación con el sistema, se tiene sólo un programa y una forma de manipularlo; se restringe al usuario remoto a sólo poder enviar peticiones pero no recibirlas por parte del servidor.

5.3.3 Transmisión de vídeo

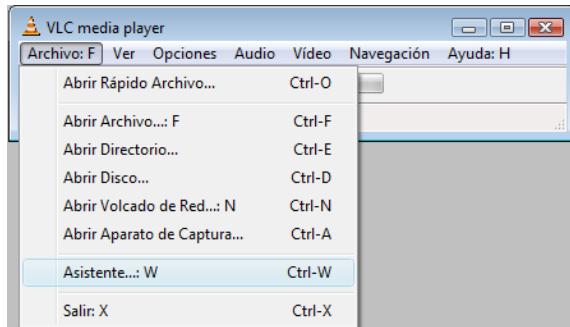
Para transmitir la señal de vídeo, se escogió la aplicación VLC en su versión 0.8.6c [18], dado que se cuenta con un *plugin* para C# y la manera de montar un servicio de vídeo con dicha aplicación es muy rápida y sencilla.

Es necesario que del lado del usuario se cuente con el *plugin* de la aplicación VLC [17] instalado de tal manera que se reciban las señales de vídeo. Esto se puede desarrollar incluyendo este controlador en la instalación de la aplicación de usuario, dado que la aplicación es de código abierto y permite realizar esta operación.

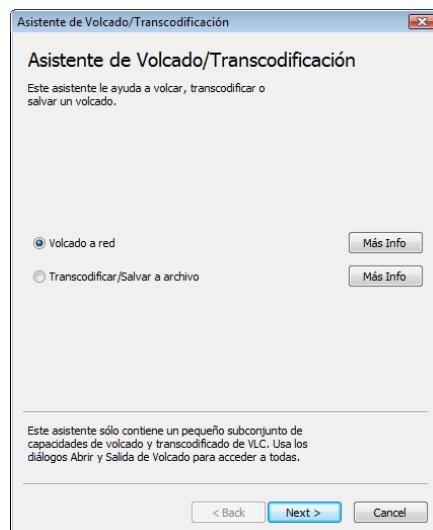
5.3.4 Montaje del servidor de vídeo

Para montar el servidor de vídeo [19], también se emplea la aplicación VLC pero del lado del servidor. Se debe escoger la opción de Volcado de red, y seguir las opciones de configuración conforme se explica a continuación.

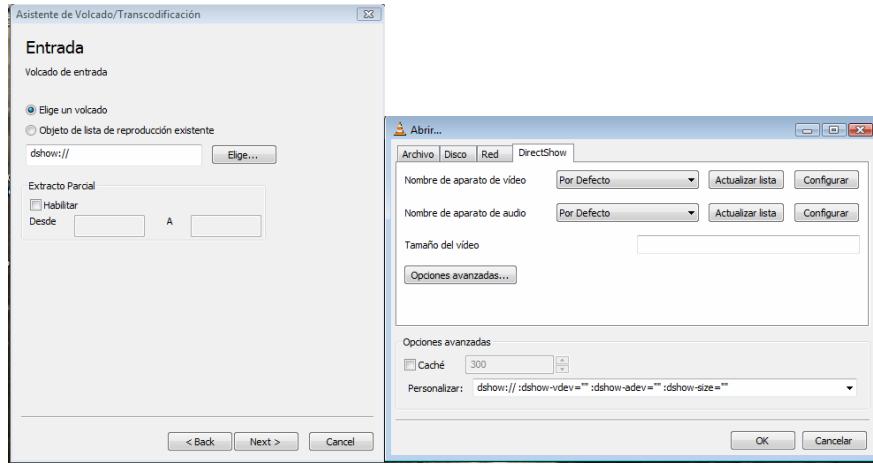
1. Seleccionar la opción de *Asistente del Menú*.



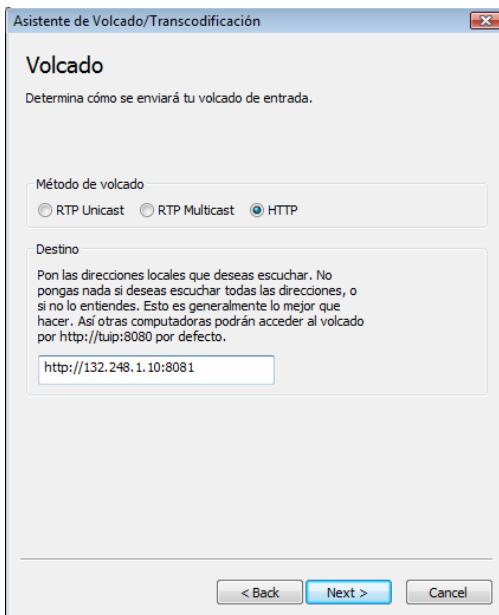
2. En la ventana del asistente, seleccionar la opción *Volcado de red* y escoger la opción de continuar.



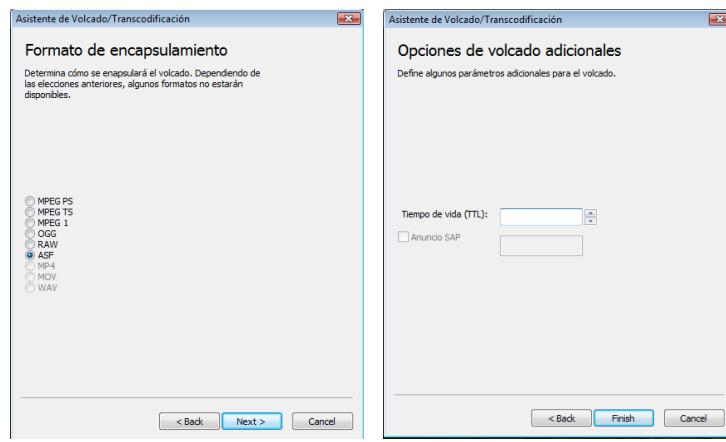
3. En la siguiente ventana, dar clic sobre el botón *Elegir*. En la ventana de *Abrir*, seleccionar el dispositivo de transmisión, la cámara web en este caso, en la pestaña *DirectShow*. Si sólo se cuenta con una cámara, puede dejarse la opción *Por Defecto*. Una vez seleccionado, dar clic en *OK*. Dar clic en *Siguiente* de la ventana restante.



4. En la siguiente sección, seleccionar la opción de *HTTP*, e indicar en el cuadro de texto la dirección IP del destino, que puede ser la IP de la aplicación de usuario o una red de transmisión de vídeo, indicando el puerto por el cual se enviará la señal. Una vez finalizado, dar clic en *Siguiente*.



5. En la siguiente sección, seleccionar el formato de encapsulado deseado y dar clic en *Siguiente*. Dejar el siguiente cuadro de texto en blanco y dar clic en *Finalizar*.



Tan pronto se haya terminado la configuración, el servidor de vídeo se habrá inicializado, y esperará una conexión por parte del usuario para transmitir la señal de la cámara web seleccionada. Las velocidades de transmisión y recepción estarán en función del *upstream* del servidor y del *downstream* del usuario.

CAPÍTULO 6.

RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

6.1 RESULTADOS DE LAS PRUEBAS

Durante las pruebas de las interfaces se obtuvo una transferencia de datos limpia, sin problemas, tanto entre el servidor y el sistema, como entre el usuario y el servidor. La transmisión de vídeo en tiempo real contuvo un retraso de al menos dos segundos, en el rango de una red de área local.

El control del sistema por parte del usuario se realizó sin problemas, alcanzando los objetivos fijados para el sistema. El manejo de entradas y salidas implementado funcionó y la comunicación entre dispositivos fue robusta a excepción del video que presenta un ligero retraso, el cual no facilita la operación remota del sistema. El uso del sistema queda restringido a las capacidades de *upstream*, o *velocidad para subir datos a la red*, del servidor y de *downstream*, o velocidad para descarga de datos, del usuario, mismas que produce los retrasos en las señales de video.

Se presentaron muchos problemas en cuanto al control del motor de pasos, ya que el transistor encargado de la regulación de voltaje se calentaba con mucha facilidad. Para resolver el problema se colocó un disipador más grande que permitió la operación del motor a pasos sin problemas. En caso de que este problema volviera a repetirse es recomendable sustituir el

transistor por uno de mayor capacidad de corriente de manera que se pueda operar el soltador en condiciones de diseño.

Además, se tuvieron problemas en el control del sistema mediante el puerto USB, ya que el controlador presenta problemas con sistemas superiores a Windows XP, lo que de cierta manera restringe el uso del sistema a servidores que cuenten con ese sistema operativo.

6.2 CONCLUSIONES

En general, la metodología de diseño aplicada para la instrumentación y el control del sistema de tiro parabólico y caída libre permitió el cumplimiento de los objetivos delimitados. Entre los principales aspectos que proporcionó la realización de este proyecto se encuentran el aprendizaje en el área de redes, la experiencia en el manejo de sistemas de video en tiempo real a través de Internet así como el control electrónico de actuadores de potencia. Asimismo, se logró conocer más conocimientos para la operación remota de sistemas mecatrónicos y su gran aplicación en el campo académico de la ingeniería.

Particularmente, la arquitectura de cliente/servidor permitió un control seguro de los sistemas mecatrónicos, puesto que sólo se limita a ser operado mediante el servidor con comandos o por parte del desarrollador, es decir, que aún cuando el servidor sea vulnerable, no se podrá tener el control del sistema si no se conoce su diseño. De la misma forma, la comunicación USB entre dispositivos es sumamente práctica y adaptable a las necesidades actuales, generando elementos novedosos y que ponen en uso conceptos dentro del estado del arte actual de la ingeniería mechatrónica.

Cabe mencionar que el retraso en las señales de video en tiempo real estará sometido a las condiciones del servicio de Internet, tanto del servidor como del usuario, y se espera a que dicho retraso sea cada vez menor con el avance del tiempo y la mejora de la infraestructura de las telecomunicaciones.

El uso del controlador previamente diseñado para el microcontrolador PIC18F4550 resultó muy útil para la rápida construcción del control electrónico del sistema. Cabe destacar que la programación de interfaces con C# para el manejo de sistemas mechatrónicos fue sencillo y la adaptación del *plugin* de video solucionó la manera en que se pueden transmitir este tipo de señales.

Se han alcanzado los objetivos planteados para el presente proyecto. Los instrumentos diseñados funcionaron como se había previsto y producen las señales electrónicas necesarias para el control de los actuadores. Dichos elementos se controlaron de manera sencilla con el uso del transistor MOSFET.

6.3 RECOMENDACIONES

Para mejorar las condiciones de operación y de mantenimiento posteriores a la realización de este trabajo, se tomaron en consideración una serie de recomendaciones que se describen a continuación:

- Cabe mencionar la importancia de la implementación de componentes adicionales que permitan que el sistema de tiro parabólico y caída libre sea seguro, puesto que en caso de que alguno de los sensores actuales fallara, se debe contar con elementos capaces de desactivar el sistema y enviar la señal de error al usuario.
- Para brindar mayor protección a los circuitos electrónicos físicos del subsistema de tiro, el uso de cajas o empaques para los mismos podría robustecer las conexiones actuales e indicar mayor información sobre las terminales y conectores que contienen.
- En caso de que la discretización del conteo de vueltas no resulte efectivo, se puede desarrollar un modelo matemático para la tensión de la cuerda. Dicho modelo podría basarse en una regresión lineal que atienda a la relación número de vueltas – alcance del tiro.
- Si llegara a presentarse algún problema con el movimiento lateral del cañón, se puede acoplar un rodamiento lineal en la guía del tornillo sinfin que produce dicho movimiento, evitando la interferencia mecánica de los elementos que permiten el cambio de posición del cañón.
- Es importante tomar en cuenta que con el transcurso del tiempo, será necesario actualizar el controlador diseñado, de manera que se adapte a los sistemas operativos más actuales, siempre y cuando se continúe utilizando el puerto USB.
- Para brindar seguridad en la obtención y descarga de la aplicación de usuario, la creación de un archivo de contraseñas y la vinculación del mismo con la interfaz de servidor podría ayudar a restringir el uso del sistema sólo a las personas que hayan sido designadas previamente.
- Si se deseara obtener una mejor transmisión de vídeo, se sugiere la operación de un servidor de vídeo y la transmisión de dichas señales a través de un sitio web, de manera que se puede controlar el sistema con las interfaces diseñadas y observar el vídeo mediante algún navegador compatible.
- Para poder acoplar el subsistema de recarga a la interfaz de control diseñada en el presente proyecto, es recomendable utilizar los puertos disponibles en el microcontrolador, y ajustar la detección de señales y el manejo de los posibles actuadores a los programas desarrollados.

ÁPENDICES

A.1 PLANOS DE CONSTRUCCIÓN

Tabla A1.1 Planos de las piezas para la instrumentación.

Piano	Pieza	No. de piezas
EC-1	Placa posicionamiento encóder	1
EC-2	Ángulo para soporte de sensor de encóder	1
EC-3	Disco c/perforación para encóder	1
EC-4	Sujetador del disco	1
PLD	Placa lateral derecha	1
PLI	Placa lateral izquierda	1
DT1	Soporte para vástago	1
DP	Boca del cañón c/ ranura para sensor	1
SPOT	Soporte para potenciómetro	1
COR	Corona para amplificador angular	1
PIN	Piñón para amplificador angular	1
ECOR	Eje para corona	1
EPIN	Eje para piñón	1
SENG	Separador de placas para engranaje	2

PSOP	Placas para soporte de engranaje	2
EC-E1	Encóder Incremental, disco (ensamble)	1
EC-E2	Encóder Incremental (ensamble)	1
SLD	Sensor lateral derecho (ensamble)	1
SLI	Sensor lateral izquierdo (ensamble)	1
ANG	Amplificador angular (ensamble)	1

A.2 DIAGRAMAS ELECTRÓNICOS ESQUEMÁTICOS

Tabla A.2.1 Requerimientos de material para el Módulo Externo del Cañón (MEC).

NOMENCLATURA	DESCRIPCIÓN	Cant.
R1-R3, R9-R13	Resistencia 1 kΩ ¼ W	11
R4	Resistencia 68 kΩ ¼ W	1
R6, R7	Resistencia 330 Ω ¼ W	2
R5, R8	Resistencia 10 kΩ ¼ W	2
U1	QRD1114	1
U2	H21A1	1
D1, D3	Led Rojo	2
D2, D4, D5	Led Verde	6
RV1	Potenciómetro lineal 50 kΩ	1
SW1, SW2	KW11-7-1	2
SW3	AVT3454	1

Tabla A.2.2 Requerimientos de material para el Circuito de Control (CC).

NOMENCLATURA	DESCRIPCIÓN	Cant.
R1, R4, R5	Resistencia 1 kΩ ¼ W	3
R2, R3	Resistencia 10 kΩ ¼ W	2
C1	Condensador electrolítico 0.1 µF	1
C2	Condensador electrolítico 47 µF	1
C3-C18	Condensador electrolítico 1 µF	8
U1, U2	H21A1	2
U3- U6	MAX232	2
U7	KA78R05	1
D1	LED Rojo	1

Tabla A.2.3 Requerimientos de material para el Circuito de Potencia (CP).

NOMENCLATURA	DESCRIPCIÓN	Cant.
R1-R8, R47, R48, R32, R33	Resistencia 1 kΩ ¼ W	10
R9-R16, R26,R27	Resistencia 330 Ω ¼ W	28
R17-R24,R28-R32	Resistencia 10 kΩ ¼ W	12
RDIS	Resistencia 6.8 Ω 3W	1
R25	Resistencia 560 Ω ¼ W	1
C1, C3	Condensador electrolítico 0.1 µF	3
C2, C4	Condensador electrolítico 47 µF	3
U1	KA78R05	1
U2-U4	74LS245	3
U5, U6	L293D	2
U9-U12	KA78R33	4
Q1-Q4, Q18, Q19	NTP60N06 (MOSFET canal N)	6
Q5-Q8, Q14,Q15	BC548 (TBJ)	6
Q9-Q13	TIP120 (Darlington)	4
Q16, Q17	IRF4905 (MOSFET-P)	2
D1	Diodo zener 6 V ½ W	1
D2	1N4973	1
D3-D6	1N4007	4
DSW1	Dipswitch 8	1

A3 LONGITUD DEL CABLEADO Y CONTEO DE CONECTORES

Tabla A.3.1 Cableado del sistema por secciones.

CABLE	ORIGEN	DESTINO	Longitud, en m	Cant.
Cable de red	MEC2	CC	2.50	1
Cable plano 4 entradas	CC	Soltador	2.75	2
Cable plano 6 entradas	MEC1	MEC2	0.30	1
Cable plano 14 entradas	CC	CC	0.30	1
Cable dúplex calibre 18	CC	Actuadores	2.50	8

Tabla A.3.2 Conectores del sistema por secciones.

ELEMENTO	UBICACIÓN	CONECTOR	Cant.
Sensor de fin de carrera izq.	Soltador	Header 4x1 hembra	2
Sensor de fin de carrera der.	Soltador	Header 4x1 hembra	2
Sensor de encóder incremental	Cañón	Header 4x1 hembra	2
Sensor de detección de pelota	Cañón	Header 4x1 hembra	2
Potenciómetro del amplificador	Cañón	Header 3x1 hembra	2
Sensor de detección del trinquete	Cañón	Header 2x1 hembra	2
Sensor lateral derecho	Cañón	Header 2x1 hembra	2
Sensor lateral izquierdo	Cañón	Header 2x1 hembra	2
Solenioide del soltador	Soltador	Conector universal hembra de dos	1

		entradas	
<i>Motor paso a paso</i>	Soltador	Conector universal hembra de cinco entradas	1
<i>Conexión del motor paso a paso</i>		Molex 6x1 macho	1
<i>Motor de ángulo de giro</i>	Cañón	Conector universal hembra de dos entradas	1
<i>Motor de movimiento lateral</i>	Cañón	Conector universal hembra de dos entradas	1
<i>Solenoide del trinquete</i>	Cañón	Conector universal hembra de dos entradas	1
<i>Motor de tensión de la cuerda</i>	Cañón	Conector universal hembra de dos entradas	1
<i>Accionamiento de la banda</i>	Banda	Conector universal hembra de dos entradas	1
<i>MEC2</i>	Cañón	Conector DB-9 hembra	2
		Conector DB-9 macho	1
		Header 6x1 macho	1
		Header 6x1 hembra	2
		Header 2x1 macho	2
		Borne de alimentación doble	1
<i>MEC1</i>	Cañón	Header 6x1 macho	1
		Header 4x1 macho	2
		Header 3x1 macho	1
		Header 2x1 macho	1
		Borne de alimentación doble	1
<i>Unidad de control</i>	Control	Conector universal macho de dos entradas	6
		Conector universal macho de cinco entradas	6
		Conector DB-9 macho	1
		Header 4x1 macho	2
		Header 31x1 hembra	1
		Borne de alimentación doble	9
		Molex 6x1 macho	1
		Header 8x2 hembra	2
		Conector para header 8x2 macho	2

Tabla A.3.4 Longitud total del cableado.

CABLE	Longitud, en m
Cable de red	2.50
Cable plano	6.10
Cable dúplex cal. 18	20.00

Tabla A.3.5 Total de conectores para el sistema.

CONECTOR	TOTAL
Header 8x2 hembra	2
Header 31x1 hembra	1
Header 6x1 hembra	2
Header 4x1 hembra	8
Header 3x1 hembra	1
Header 2x1 hembra	6
Header 8x2 macho	1
Header 6x1 macho	1
Header 3x1 macho	1
Header 2x1 macho	3
Molex 6x1 macho	2
Conector DB-9 macho	2
Conector DB-9 hembra	2
Conector universal hembra de dos entradas	6
Conector universal macho de dos entradas	2
Conector universal hembra de cinco entradas	1
Conector universal macho de cinco entradas	1
Borne de alimentación doble	9

A4 PROGRAMA DEL MICROCONTROLADOR PIC18F4550

```
//SE INCLUYE LAS BIBLIOTECAS PARA OPERAR EL PIC18F4550 CON CCS C
#include <18F4550.h>

//SE DECLARAN LAS PALABRAS DE CONFIGURACION
//NO WATCHDOG TIMER, NO MODO DE BAJO VOLTAJE...
#fuses HSPLL,NOWDT,NOPROTECT,LVP,NODEBUG,USBDIV,PLL5,CPUDIV1,VREGEN
//DECLARACION DE USO DE UNA FRECUENCIA DE 48MHz PARA EL OSCILADOR EXTERNO
//NOTA: OVERCLOCKING A UN OSCILADOR DE 20MHz
#use delay(clock=48000000)
```

```

//ESTA RUTINA PERMITE LA HABILITACION DEL MODO BOOTLOADER EN CASO DE SER NECESARIO
//REPROGRAMAR EL DISPOSITIVO
#define build(reset=0x800)
#define build(interrupt=0x808)
#define org 0x0000,0x07ff
void bootloader() {
    #asm
        nop
    #endasm
}

//CONFIGURACION DE TX/RX BULK
#define USB_HID_DEVICE FALSE //DESHABILITACION DE LAS DIRECTIVAS HID
#define USB_EP1_TX_ENABLE USB_ENABLE_BULK //INTERRUPCION DE TRANSFERENCIA BULK HABILITADA PARA EL EP1 (HOST)
#define USB_EP1_RX_ENABLE USB_ENABLE_BULK //INTERRUPCION DE RECEPCION BULK HABILITADA PARA EL EP1
#define USB_EP1_TX_SIZE 64 //TAMANO DEL BUFFER DE TRANSFERENCIA PARA EP1
#define USB_EP1_RX_SIZE 8 //TAMANO DEL BUFFER DE RECEPCION PARA EP1

//SE INCLUYEN LAS BIBLIOTECAS DE CCS C PARA EL USO DEL USB
#include <pic18_usb.h>
#include <PicUSB.h>
#include <usb.c>

//DEFINICION DE LOS DATOS DE RECEPCION
#define modo recibir[0] //MODO DE TRABAJO
#define dir recibir[1] //DIRECCION (ANGULO Y SOLTADOR)
#define pos recibir[2] //POSICION (ANGULO, SOLTADOR Y POTENCIA)
//DEFINICION DEL DATO DE ENVIO
#define estado manda[0] //ESTADO DEL SISTEMA

//DEFINICION DE LOS SENsoRES O ENTRADAS
#define ILD PIN_B0 //INTERRUPTOR LATERAL DERECHO
#define ILI PIN_B1 //INTERRUPTOR LATERAL IZQUIERDO
#define EC PIN_B2 //DETECCION DE PELOTA
#define DP PIN_B3 //ENCODER DEL CANON
#define SSI PIN_B4 //SENSOR SOLTADOR IZQUIERDO
#define SSD PIN_B5 //SENSOR SOLTADOR DERECHO
#define DT PIN_B6 //DETECCION DEL TRINQUETE
#define AA PIN_E0 //AMPLIFICADOR ANGULAR

//DEFINICION DE LAS SALIDAS
#define P1 PIN_A0 //BIT 1 MPAP SOLTADOR
#define P2 PIN_A1 //BIT 2 MPAP SOLTADOR
#define P3 PIN_A3 //BIT 3 MPAP SOLTADOR
#define P4 PIN_A4 //BIT 4 MPAP SOLTADOR
#define DIRS PIN_A2 //INDICADOR DIRECCION (0=ABAJO,1=ARRIBA)
#define OO PIN_A5 //INDICADOR ON/OFF

```

```

#define MLI PIN_D0 //MOTOR LATERAL IZQUIERDO
#define MLD PIN_D1 //MOTOR LATERAL DERECHO
#define MAI PIN_D2 //MOTOR ANGULO IZQUIERDA
#define MAD PIN_D3 //MOTOR ANGULO DERECHA
#define SS PIN_D4 //SOLENOIDE SOLTADOR
#define MT PIN_D5 //MOTOR DE TENSION
#define ST PIN_D6 //SOLENOIDE TRINQUETE
#define AB PIN_D7 //APAGADO BANDA

//DEFINICION DE LOS MODOS DEL SISTEMA
#define ENCENDIDO_BANDA 8
#define POSICION TIRO 1
#define AJUSTE_ANGULO 2
#define AJUSTE_SOLTADOR 3
#define AJUSTE_POTENCIA 4
#define TIRO 5
#define POSICION_CARGA 6
#define APAGADO_BANDA 7
#define SOP_TIRO 9

#define OK 0
#define MAL 1
#define TOPE 2 //TOPES (SOLTADOR)

int q; //VALOR DE LECTURA PARA EL CONVERTIDOR
//ACCESIBLE PARA TODAS LAS FUNCIONES

//DEFINICION DE LA FUNCION DE LECTURA ADC
void lectura(){
    set_adc_channel(5); //SE ESTABLECE EL CANAL DE LECTURA (PE.0)
    delay_us(20); //TIEMPO PARA PREPARAR LA CONVERSION
    q=read_adc(); //OBTENER EL VALOR ANALOGICO Y GUARDARLO
}

//DEFINICION DE LA FUNCION DE PETICION
int8 peticion(int8 m, int8 d, int8 p){
int exito=0;
int i=0;
//SE DETERMINARA EL MODO Y SE REALIZA LA ACCION
switch(m){

    case ENCENDIDO_BANDA:
        output_low(AB); //ENCENDER BANDA
        return OK; //REGRESAR ESTADO
        break; //SALIR

    case AJUSTE_ANGULO:
        lectura(); //REALIZAR LECTURA ADC
        if(q<d){ //SI EL VALOR ADC OBTENIDO ES MENOR AL REQUERIDO
            output_high(MAI); //ACTIVAR EL GIRO DEL MOTOR DE ANGULO A LA IZQUIERDA
            do{ //REALIZAR LA LECTURA ADC

```

```

lectura();
}while(q<d); //MIENTRAS NO SE HAYA ALCANZADO EL VALOR REQUERIDO
output_low(MAI); //DESACTIVAR EL MOTOR
return OK; //REGRESAR ESTADO CORRECTO
break; //SALIR DE RUTINA
}
else if(q>d){ //CASO CONTRARIO, SI EL VALOR ADC ES MAYOR AL REQUERIDO
output_high(MAD); //ACTIVAR EL GIRO DEL MOTOR DE ANGULO A LA DERECHA
do{ //REALIZAR LA LECTURA ADC
lectura();
}while(q>d); //MIENTRAS NO SE HAYA ALCANZADO EL VALOR REQUERIDO
output_low(MAD); //DEACTIVAR EL MOTOR
return OK; //REGRESAR ESTADO CORRECTO
break; //SALIR DE LA RUTINA
}
else if(q==d) //SI VALOR ADC ES IGUAL AL REQUERIDO
return OK; //NO HACER NADA Y REGRESAR
break; //SALIR DE RUTINA

//NOTA:LA RUTINA SE AJUSTA PARA QUE SE PASE UN POCO DEL VALOR REQUERIDO
//CON LA FINALIDAD DE NO CAER EN UN VALOR QUE NO SE PUDIERA IGUALAR

case AJUSTE_SOLTADOR:
//RUTINA DEL SOLTADOR
//DEPENDIENDO DEL VALOR DE LA DIRECCION, SE MOVERA EL MOTOR A PASOS
//DEL SOLTADOR POR UN NUMERO DE VECES INDICADO POR LA VARIABLE P
//SI SE ENCUENTRAN TOPES O FINALES DE CARRERA, SE SALDRA DE LA RUTINA
//INDICANDO QUE SE LLEGO A ALGUNO DE LOS MENCIONADOS
output_high(OO); //INDICADOR ON/OFF
if(d==0){
do{
if(!INPUT(SSD)){//SSD
exito=TOPE;
break;
}
//GIRANDO A LA IZQUIERDA
output_low(P1);
output_low(P3);
output_high(P4);
output_high(P2);
delay_ms(10);
output_low(P1);
output_low(P4);
output_high(P2);
output_high(P3);
delay_ms(10);
output_low(P4);
output_low(P2);
output_high(P1);
output_high(P3);
delay_ms(10);
}

```

```

        output_low(P2);
        output_low(P3);
        output_high(P4);
        output_high(P1);
        delay_ms(10);
        i++;
    }while(i!=p);
}
if(d==1){
    output_high(DIRS); //INDICADOR DE DIRECCION
    do{
        if(!INPUT(SSI)){ //SSI
            exito=TOPE;
            break;
        }
        //GIRANDO LA DERECHA
        output_low(P2);
        output_low(P3);
        output_high(P4);
        output_high(P1);
        delay_ms(10);
        output_low(P2);
        output_low(P4);
        output_high(P1);
        output_high(P3);
        delay_ms(10);
        output_low(P1);
        output_low(P4);
        output_high(P2);
        output_high(P3);
        delay_ms(10);
        output_low(P1);
        output_low(P3);
        output_high(P2);
        output_high(P4);
        delay_ms(10);
        i++;
    }while(i!=p);
}
//DESENERGIZANDO MPAP
output_low(P1);
output_low(P2);
output_low(P3);
output_low(P4);
output_low(OO); //APAGANDO INDICADORES
output_low(DIRS);
i=0;
if(exito==TOPE){
    exito=0;
    return TOPE;
    break;
}

```

```

}

return OK;
break;

case AJUSTE_POTENCIA:
output_low(MT); //ACTIVAR TENSION DE LA CUERDA
//while(INPUT(PIN_B4)){} //DT //ESPERAR AL SENSOR PARA INICIAR CONTEO
//RUTINA DE CONTEO CON EL ENCODER INCREMENTAL
while(!l!=p){ //NO SALIR DE LA RUTINA HASTA ALCANZAR
while(INPUT(EC)); //EC //EL NUMERO DE VUELTAS
    i++;
while(!INPUT(EC));
}
i=0; //REINICIAR EL CONTADOR DE VUELTAS
output_high(MT);
return OK; //REGRESAR EL ESTADO
break; //SALIR

case TIRO:
output_low(ST); //ACTIVAR SOLENOIDE TRINQUETE
//RUTINA PARA TIRO
//UNA VEZ ACTIVADO EL SOLENOIDE, ESPERARA A QUE SE
//ACTIVE EL SENSOR DP O BIEN ESPERA UN SEGUNDO SI NO LO HIZO
//SI SE ACTIVO EL SENSOR DP, ACTIVARA EL SOLENOIDE DEL SOLTADOR
//Y REGRESA UN ESTADO DE EXITO (1), EN CASO CONTRARIO ENVIARA EL MENSAJE
//DE FALLO (0)
while(!INPUT(DP)){} //DP

if(INPUT(DP)){
    output_low(SS);
    delay_ms(1000);
    output_high(ST);
    output_high(SS);
    exito = OK;
}
output_high(ST);
if(exito==OK){
    exito=0;
    return OK;
    break;
}
return MAL;
break; //SALIR

case SOP_TIRO:
output_high(MLD);
while(true){
    if(!INPUT(ILD))
    break;
}
output_low(MLD);

```

```

return OK;

case POSICION_CARGA:
output_high(ML1); //MOVER A POSICION DE CARGA
while(INPUT(PIN_B1)){
}
output_low(ML1); //UNA VEZ QUE LLEGO SE APAGA EL MOTOR
//return OK;      //REGRESAR ESTADO
delay_ms(1000);
do{
if(!INPUT(SSD)){ //SSI
    exito=TOPE;
    break;
}
//GIRANDO LA DERECHA
output_low(P1);
output_low(P3);
output_high(P4);
output_high(P2);
delay_ms(10);
output_low(P1);
output_low(P4);
output_high(P2);
output_high(P3);
delay_ms(10);
output_low(P4);
output_low(P2);
output_high(P1);
output_high(P3);
delay_ms(10);
output_low(P2);
output_low(P3);
output_high(P4);
output_high(P1);
delay_ms(10);
}while(true);
output_low(P1);
output_low(P2);
output_low(P3);
output_low(P4);
output_low(OO);
output_low(DIRS);
break;      //SALIR

case APAGADO_BANDA:
output_high(AB); //APAGAR BANDA
break;
}

}

//FUNCION PRINCIPAL

```

```

void main(void) {

    //DECLARACION DE VARIABLES
    int8 recibe[3];           //3 BYTES PARA RECIBIR
    int8 manda[1];            //1 BYTE PARA MANDAR
    int res;

    //CONFIGURACION CONVERTIDOR ANALOGICO/DIGITAL
    setup_adc_ports(AN0);
    setup_adc(ADC_CLOCK_INTERNAL);

    //CONFIGURACION DE PUERTOS
    //SALIDAS: PUERTO A, C Y D
    SET_TRIS_A(0X00);
    SET_TRIS_C(0x00);
    SET_TRIS_D(0x00);
    //HOME
    SET_TRIS_B(0xFF);
    output_a(0x00);
    output_low(MLI);
    output_low(MLD);
    output_low(MAI);
    output_low(MAD);
    output_high(SS);
    output_high(MT);
    output_high(ST);
    output_high(AB);

    ////DETECCION DEL SISTEMA////
    /*
    EL MICROCONTROLADOR VERIFICA QUE
    EL SISTEMA SE ENCUENTRE EN EL HOME ANTES
    DE INICIAR, DE LO CONTRARIO MANDARA UNA
    NOTIFICACION DE QUE NECESITA COLOCARSE
    EL SISTEMA EN LA POSICION DE HOME
    */

    //SE INICIA LA CONEXION USB
    usb_init();                //INICIALIZAR EL USB
    usb_task();                 //HABILITACION DEL PERIFERICO USB E INTERRUPCIONES
    usb_wait_for_enumeration(); //ESPERANDO SER ENUMERADO POR EL HOST

    /////CONFIGURACION INICIAL DEL SISTEMA
    //LAS SALIDAS SON NEGADAS DADA LA
    //CONFIGURACION DE LA ETAPA DE POTENCIA
    //ESTADO INICIAL DE LOS ACTUADORES
    //BANDA DESACTIVADA

    //CANON
}

```

```

//MOTOR ANGULO DESACTIVADO
//SOLENOIDE TRINQUETE DESACTIVADO
//MOTOR LATERAL DESACTIVADO
//MOTOR DE TENSION DESACTIVADO

//SOLTADOR
//MOTOR A PASOS SIN SENALES
//SOLENOIDE SOLTADOR DESACTIVADO
    output_a(0x00);
    output_low(MLI);
    output_low(MLD);
    output_low(MAI);
    output_low(MAD);
    output_high(SS);
    output_high(MT);
    output_high(ST);
    output_high(AB);

while (TRUE)
{
    if(usb enumerated())      //SE ESPERARAN DATOS SIEMPRE Y CUANDO EL DISPOSITIVO ESTE
CONFIGURADO POR EL HOST
    {
        if (usb_kbhit(1))    //INTERRUPCION POR RECEPCION DE DATOS EN EL PUERTO USB
        {
            usb_get_packet(1, recibe, 3); //RECIBIR LOS TRES BYTES POR PARTE DEL HOST
        }
    }
    //SE REALIZA UNA PETICION AL SISTEMA Y SE OBTIENE EL ESTADO
    manda[0] = peticion(modo, dir, pos);
    //LIMPIEZA DE VARIABLES
    recibe[0]=0;
    recibe[1]=0;
    recibe[2]=0;

}
}

```

A5 PROGRAMA DE LA INTERFAZ USUARIO-SERVIDOR

INTERFAZ DE SERVIDOR

Corrida de Program.cs

```

using System;
using System.Collections.Generic;
using System.Windows.Forms;
using System.Runtime.InteropServices; // Se agregaron estas 3
using PVOID = System.IntPtr;
using DWORD = System.UInt32;

```

```

namespace SERVICIO
{
    unsafe public class control
    {
        #region Definición de los Strings: EndPoint y VID_PID
        string vid_pid_norm = "vid_04d8&pid_0001";

        string out_pipe = "\\\'MCHP_EP1";
        string in_pipe = "\\\'MCHP_EP1";
        #endregion

        #region Funciones importadas de la DLL: USBDLL.dll
        [DllImport("mpusbapi.dll")]
        private static extern DWORD _MPUSBGetDLLVersion();
        [DllImport("mpusbapi.dll")]
        private static extern DWORD _MPUSBGetDeviceCount(string pVID_PID);
        [DllImport("mpusbapi.dll")]
        private static extern void* _MPUSBOpen(DWORD instance, string pVID_PID, string pEP, DWORD dwDir, DWORD
dwReserved);
        [DllImport("mpusbapi.dll")]
        private static extern DWORD _MPUSBRead(void* handle, void* pData, DWORD dwLen, DWORD* pLength,
DWORD dwMilliseconds);
        [DllImport("mpusbapi.dll")]
        private static extern DWORD _MPUSBWrite(void* handle, void* pData, DWORD dwLen, DWORD* pLength,
DWORD dwMilliseconds);
        [DllImport("mpusbapi.dll")]
        private static extern DWORD _MPUSBReadInt(void* handle, DWORD* pData, DWORD dwLen, DWORD* pLength,
DWORD dwMilliseconds);
        [DllImport("mpusbapi.dll")]
        private static extern bool _MPUSBClose(void* handle);
        #endregion

        void* myOutPipe;
        void* myInPipe;

        static void Main()
        {
            Application.EnableVisualStyles();
            Application.Run(new SERVIDOR());
        }

        public void OpenPipes()
        {
            DWORD selection = 0;

            myOutPipe = _MPUSBOpen(selection, vid_pid_norm, out_pipe, 0, 0);
            myInPipe = _MPUSBOpen(selection, vid_pid_norm, in_pipe, 1, 0);
        }

        public void ClosePipes()
        {
            _MPUSBClose(myOutPipe);
            _MPUSBClose(myInPipe);
        }

        private void SendPacket(byte* SendData, DWORD SendLength)
        {
            uint SendDelay = 1000;

            DWORD SentDataLength;

            OpenPipes();
            _MPUSBWrite(myOutPipe, (void*)SendData, SendLength, &SentDataLength, SendDelay);
            ClosePipes();
        }

        private uint SendReceivePacket(byte* SendData, DWORD SendLength, byte* ReceiveData,
DWORD* ReceiveLength, uint SendDelay, uint ReceiveDelay)
        {
            DWORD SentDataLength;
            DWORD ExpectedReceiveLength = *ReceiveLength;

            OpenPipes();
            if (_MPUSBWrite(myOutPipe, SendData, SendLength, &SentDataLength, SendDelay) > 0)
            {
                if (_MPUSBRead(myInPipe, ReceiveData, ExpectedReceiveLength,

```

```

        ReceiveLength, ReceiveDelay) > 0)
    {
        if (*ReceiveLength == ExpectedReceiveLength)
        {

            return 1; // Success!
        }
        else if (*ReceiveLength < ExpectedReceiveLength)
        {
            return 2; // Partially failed, incorrect receive length
        } //end if else
    }
}
ClosePipes();

return 0; // Operation Failed
}

private void ReceivePacket(byte* ReceiveData, DWORD* ReceiveLength)
{
    uint ReceiveDelay = 1000;

    DWORD ExpectedReceiveLength = *ReceiveLength;

    OpenPipes();
    _MPUSBRead(myInPipe, (void*)ReceiveData, ExpectedReceiveLength, ReceiveLength, ReceiveDelay);
    ClosePipes();
}
public void actual(int dato0, int dato1, int dato2)
{
    byte* send_buf = stackalloc byte[3];

    send_buf[0] = (byte)dato0;
    send_buf[1] = (byte)dato1;
    send_buf[2] = (byte)dato2;
    SendPacket(send_buf, 3);
}
public uint ReadADC(byte adcnim)
{
    byte* send_buf = stackalloc byte[1];
    byte* receive_buf = stackalloc byte[1];
    DWORD RecvLength = 1;
    send_buf[0] = adcnim;           // coloca un 3 o 4 en send_buf, que corresponde a la opcion de ADC en el PIC
    SendPacket(send_buf, 1);       // llama función para envío de datos
    ReceivePacket(receive_buf, &RecvLength); //llama a la función para recibir la lectura del ADC del PIC
    return (uint)receive_buf[0];    //Devuelve la lectura del ADC
}

public uint ReadSen()
{
    byte* send_buf = stackalloc byte[1];
    byte* receive_buf = stackalloc byte[1];
    DWORD RecvLength = 1;
    send_buf[0] = 5;               // coloca un 5 en send_buf, que corresponde a la opcion de LECTURA DE SENSORES
en el PIC
    SendPacket(send_buf, 1);       // llama función para envío de datos
    ReceivePacket(receive_buf, &RecvLength); //llama a la función para recibir la lectura del ADC del PIC
    return (uint)receive_buf[0];    //Devuelve la lectura de los sensores
}
public uint finaliza()
{
    byte* receive_buf = stackalloc byte[1];
    DWORD RecvLength = 1;
    ReceivePacket(receive_buf, &RecvLength);
    return (uint)receive_buf[0];    //Devuelve la lectura de los sensores
}
public uint Tiempo()
{
    uint dato = 0;

    byte* receive_buf = stackalloc byte[2];
    DWORD RecvLength = 2;
    ReceivePacket(receive_buf, &RecvLength);
}

```

```

        dato = receive_buf[0];
        return dato;
    }
}

```

CORRIDA DEL FORMULARIO

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
using System.Threading;

namespace SERVICIO
{
    public partial class SERVIDOR : Form
    {
        control servicio = new control();
        int n = 0, s=0;
        string message;
        char[] sep = { ';' };
        bool activado = true;
        public SERVIDOR()
        {
            InitializeComponent();
        }

        private void conexionhilo()
        {
            while (activado)
            {
                byte[] data = new byte[1024];
                IPEndPoint ipep = new IPEndPoint(IPAddress.Parse(IPTB.Text), 8081);
                Socket newsock = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
                Socket client;
                string welcome;
                int sent;

                try
                {
                    newsock.Bind(ipep);
                }

                catch (SocketException)
                {
                    n = 1;
                    Respuesta();
                }
                n = 2;
                Respuesta();

                newsock.Listen(1);

                client = newsock.Accept();

                IPEndPoint newclient = (IPEndPoint)client.RemoteEndPoint;

                n = 3;
                Respuesta();

                welcome = "BIENVENIDO";
            }
        }
    }
}

```

```

data = Encoding.ASCII.GetBytes(welcome);
sent = SendData(client, data);
for (int m = 0; m < 2; m++)
{
    if (message == "exit")
        break;
    try
    {
        data = ReceiveData(client);
        message = Encoding.ASCII.GetString(data);
        n = 4;
        Respuesta();
        m--;
    }
    catch (SocketException)
    {
        n = 5;
        Respuesta();
    }
}
n = 6;
Respuesta();

//client.Shutdown(SocketShutdown.Both);
//client.Close();

//newsock.Shutdown(SocketShutdown.Both);
//newsock.Close();

n = 5;
Respuesta();

s++;
if (s == 5)
    s = 0;
}
}

private Thread hiloaux;

private void SERVIDOR_Load(object sender, EventArgs e)
{
    TIM1.Tick += new EventHandler(this.cambio);
    TIM1.Enabled = true;
}

private void BIN_Click(object sender, EventArgs e)
{
    //xt.Text = Convert.ToString(servicio.Tiempo());
    BDET.Enabled = true;
    BIN.Enabled = false;
    IPTB.Enabled = false;

    ThreadStart hiloxd = new ThreadStart(conexionhilo);
    hiloaux = new Thread(hiloxd);

    hiloaux.Start();
}

private void cambio(object sender, EventArgs e)
{
    FTB.Text = DateTime.Now.ToString();
}

private void BDET_Click(object sender, EventArgs e)
{
    activado = false;
    BDET.Enabled = false;
    BIN.Enabled = true;
}

}

```

```

private static byte[] ReceiveData(Socket sock)
{
    int totalRecv = 0; //la cantidad total recibida
    int recv; //la cantidad recibida individualmente
    byte[] dataSize = new byte[4]; //este sera el tamaño del mensaje en byte[>
    int size; //Este sera el tamaño en entero
    int dataLeft = 0; //Esto nos indica cuanta informacion nos falta recibir

    recv = sock.Receive(dataSize, 0, 4, SocketFlags.None);
    size = BitConverter.ToInt32(dataSize, 0);
    dataLeft = size;
    byte[] data = new byte[size];

    while (totalRecv < size)
    {
        recv = sock.Receive(data, totalRecv, dataLeft, SocketFlags.None);
        if (recv == 0)
        {
            data = Encoding.ASCII.GetBytes("exit");
            break;
        }
        totalRecv += recv;
        dataLeft -= recv;
    }
    return data;
}

private static int SendData(Socket sock, byte[] data)
{
    int total = 0;
    int size = data.Length;
    int dataLeft = size;
    int sent;

    byte[] dataSize = new byte[4];
    dataSize = BitConverter.GetBytes(size);
    sent = sock.Send(dataSize);

    while (total < size)
    {
        sent = sock.Send(data, total, dataLeft, SocketFlags.None);
        total += sent;
        dataLeft -= sent;
    }
    return total; //regresamos el total que enviamos
}

private void Respuesta()
{
    if (InvokeRequired)
    {
        MethodInvoker method = new MethodInvoker(Respuesta);

        Invoke(method);

        return;
    }

    switch(n){
    case 1:
        ESTB.Text = "Al parecer otra instancia del servidor ya esta corriendo...";
        break;
    case 2:
        ESTB.Text = "Esperando cliente...";
        break;
    case 3:
        ESTB.Text = "Conectado con cliente";
        break;
    case 4:
        string[] arr = message.Split(sep);
        switch(arr[0]){
        case "EBANDA":

```

```

ESIS.Text = message;
//servicio.actual(8, 0, 0);
break;
case "PTIRO":
ESIS.Text = message;
//servicio.actual(9, 0, 0);
break;
case "PCARGA":
ESIS.Text = message;
//servicio.actual(6, 0, 0);
break;
case "ANG":
ESIS.Text = "ANGULO = " + arr[1];
//servicio.actual(2, Convert.ToInt16(textBox1.Text), 0);
break;
case "POT":
ESIS.Text = "POTENCIA = " + arr[1] + " %";
//servicio.actual(4, 0, bar.Value*100);
break;
case "SIZQ":
ESIS.Text = "SOLTADOR ARRIBA " + arr[1];
// x.actual(3, 0, bar.Value);
break;
case "SDER":
ESIS.Text = "SOLTADOR ABAJO " + arr[1];
// servicio.actual(3, 1, bar.Value);
break;
case "BOOM":
ESIS.Text = message;
//servicio.actual(5, 0, 0);
break;
case "ABANDA":
ESIS.Text = message;
//servicio.actual(7, 0, 0);
break;
default:
ESTB.Text = message;
break;
}
break;
case 5:
ESTB.Text = "No se encontro conexión con el cliente";
break;
case 6:
ESIS.Text = "Pase por aqui";
ESTB.Text = "Se ha desconectado el cliente, esperando conexión... ";
break;
}
}
}

private void SERVIDOR_Unload(object sender, EventArgs e)
{
this.Close();
hiloaux.Abort();
}

}
}

```

INTERFAZ DE USUARIO

CORRIDA DE Program.cs

```

using System; // bibliotecas de clases
using System.Collections.Generic;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
using System.Text;

namespace TIRO_PARABOLICO
{
unsafe public class control

```

```

{
    [STAThread]
    static void Main(string[] args)
    {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new Form1());
    }
}

```

CORRIDA DEL FORMULARIO

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Threading;
using System.Net;
using System.Net.Sockets;
using System.IO;
using Microsoft.Win32;
using System.Runtime.InteropServices;

namespace TIRO_PARABOLICO
{
    public partial class Form1 : Form
    {
        int progreso, tiempo,minutos, n=0,s=0, tcarga=0;
        control datos = new control();
        byte[] data = new byte[1024];
        bool carga, ftiempo;
        string message;
        char[] sep = {','};
        public Form1()
        {
            InitializeComponent();
        }

        public class LibVlc : IDisposable
        {

            #region public enums
            public enum Error
            {
                Success = -0,
                NoMem = -1,
                Thread = -2,
                Timeout = -3,
                NoMod = -10,
                NoObj = -20,
                BadObj = -21,
                NoVar = -30,
                BadVar = -31,
                Exit = -255,
                Generic = -666,
                Execption = -998,
                NoInit = -999
            };
            enum Mode
            {
                Insert = 0x01,
                Replace = 0x02,
                Append = 0x04,
                Go = 0x08,
                CheckInsert = 0x10
            };
            enum Pos
            {
                End = -666
            };
            #endregion
            #region public structs

```

```

[StructLayout(LayoutKind.Explicit)]
public struct vlc_value_t
{
    [FieldOffset(0)]
    public Int32 i_int;
    [FieldOffset(0)]
    public Int32 b_bool;
    [FieldOffset(0)]
    public float f_float;
    [FieldOffset(0)]
    public IntPtr psz_string;
    [FieldOffset(0)]
    public IntPtr p_address;
    [FieldOffset(0)]
    public IntPtr p_object;
    [FieldOffset(0)]
    public IntPtr p_list;
    [FieldOffset(0)]
    public Int64 i_time;
    [FieldOffset(0)]
    public IntPtr psz_name;
    [FieldOffset(4)]
    public Int32 i_object_id;
}
#endifregion
#region libvlc api
[DllImport("libvlc")]
static extern int VLC_Create();
[DllImport("libvlc")]
static extern Error VLC_Init(int iVLC, int Argc, string[] Argv);
[DllImport("libvlc")]
static extern Error VLC_AddIntf(int iVLC, string Name, bool Block, bool Play);
[DllImport("libvlc")]
static extern Error VLC_Die(int iVLC);
[DllImport("libvlc")]
static extern string VLC_Error();
[DllImport("libvlc")]
static extern string VLC_Version();
[DllImport("libvlc")]
static extern Error VLC_CleanUp(int iVLC);
[DllImport("libvlc")]
static extern Error VLC_Destroy(int iVLC);
[DllImport("libvlc")]
static extern Error VLC_AddTarget(int iVLC, string Target, string[] Options, int OptionsCount, int Mode, int Pos);
[DllImport("libvlc")]
static extern Error VLC_Play(int iVLC);
[DllImport("libvlc")]
static extern Error VLC_Pause(int iVLC);
[DllImport("libvlc")]
static extern Error VLC_Stop(int iVLC);
[DllImport("libvlc")]
static extern bool VLC_IsPlaying(int iVLC);
[DllImport("libvlc")]
static extern float VLC_PositionGet(int iVLC);
[DllImport("libvlc")]
static extern float VLC_PositionSet(int iVLC, float Pos);
[DllImport("libvlc")]
static extern int VLC_TimeGet(int iVLC);
[DllImport("libvlc")]
static extern Error VLC_TimeSet(int iVLC, int Seconds, bool Relative);
[DllImport("libvlc")]
static extern int VLC_LengthGet(int iVLC);
[DllImport("libvlc")]
static extern float VLC_SpeedFaster(int iVLC);
[DllImport("libvlc")]
static extern float VLC_SpeedSlower(int iVLC);
[DllImport("libvlc")]
static extern int VLC_PlaylistIndex(int iVLC);
[DllImport("libvlc")]
static extern int VLC_PlaylistNumberOfItems(int iVLC);
[DllImport("libvlc")]
static extern Error VLC_PlaylistNext(int iVLC);
[DllImport("libvlc")]
static extern Error VLC_PlaylistPrev(int iVLC);
[DllImport("libvlc")]
static extern Error VLC_PlaylistClear(int iVLC);

```

```

[DllImport("libvlc")]
static extern int VLC_VolumeSet(int iVLC, int Volume);
[DllImport("libvlc")]
static extern int VLC_VolumeGet(int iVLC);
[DllImport("libvlc")]
static extern Error VLC_VolumeMute(int iVLC);
[DllImport("libvlc")]
static extern Error VLC_FullScreen(int iVLC);
[DllImport("libvlc")]
static extern Error VLC_VariableType(int iVLC, string Name, ref int iType);
[DllImport("libvlc")]
static extern Error VLC_VariableSet(int iVLC, string Name, vlc_value_t value);
[DllImport("libvlc")]
static extern Error VLC_VariableGet(int iVLC, string Name, ref vlc_value_t value);
[DllImport("libvlc")]
static extern string VLC_Error(int i_err);
#endregion
#region local members
private int m_iVlcHandle = -1;
private Control m_wndOutput = null;
private string m_strVlcInstallDir = "";
private string m_strLastError = "";
#endregion
public LibVlc()
{
    m_strVlcInstallDir = QueryVlcInstallPath();
}
#region IDisposable Members
public void Dispose()
{
    if (m_iVlcHandle != -1)
    {
        try
        {
            VLC_CleanUp(m_iVlcHandle);
            VLC_Destroy(m_iVlcHandle);
            VideoOutput = null;
        }
        catch {}
    }
    m_iVlcHandle = -1;
}
#endregion
#region PUBLIC PROPERTIES
public string VlcInstallDir
{
    get { return m_strVlcInstallDir; }
    set { m_strVlcInstallDir = value; }
}
public bool IsInitialized
{
    get { return (m_iVlcHandle != -1); }
}
public Control VideoOutput
{
    get { return m_wndOutput; }
    set
    {
        // clear old window
        if (m_wndOutput != null)
        {
            m_wndOutput.Resize -= new EventHandler(wndOutput_Resize);
            m_wndOutput = null;
            if (m_iVlcHandle != -1)
                SetVariable("drawable", 0);
        }
        // set new
        m_wndOutput = value;
        if (m_wndOutput != null)
        {
            if (m_iVlcHandle != -1)
                SetVariable("drawable", m_wndOutput.Handle.ToInt32());
            m_wndOutput.Resize += new EventHandler(wndOutput_Resize);
            wndOutput_Resize(null, null);
        }
    }
}

```

```

        }
    public string LastError
    {
        get { return m_strLastError; }
    }
    public bool IsPlaying
    {
        get
        {
            if (m_iVlcHandle == -1)
            {
                m_strLastError = "LibVlc is not initialized";
                return false;
            }
            try
            {
                return VLC_IsPlaying(m_iVlcHandle);
            }
            catch (Exception ex)
            {
                m_strLastError = ex.Message;
                return false;
            }
        }
    }
    public int LengthGet
    {
        get
        {
            if (m_iVlcHandle == -1)
            {
                m_strLastError = "LibVlc is not initialized";
                return -1;
            }
            try
            {
                return VLC_LengthGet(m_iVlcHandle);
            }
            catch (Exception ex)
            {
                m_strLastError = ex.Message;
                return -1;
            }
        }
    }
    public int TimeGet
    {
        get
        {
            if (m_iVlcHandle == -1)
            {
                m_strLastError = "LibVlc is not initialized";
                return -1;
            }
            try
            {
                return VLC_TimeGet(m_iVlcHandle);
            }
            catch (Exception ex)
            {
                m_strLastError = ex.Message;
                return -1;
            }
        }
    }
    public float PositionGet
    {
        get
        {
            if (m_iVlcHandle == -1)
            {
                m_strLastError = "LibVlc is not initialized";
                return -1;
            }
            try
            {

```

```

        return VLC_PositionGet(m_iVlcHandle);
    }
    catch (Exception ex)
    {
        m_strLastError = ex.Message;
        return -1;
    }
}
public int VolumeGet
{
    get
    {
        if (m_iVlcHandle == -1)
        {
            m_strLastError = "LibVlc is not initialized";
            return -1;
        }
        try
        {
            return VLC_VolumeGet(m_iVlcHandle);
        }
        catch (Exception ex)
        {
            m_strLastError = ex.Message;
            return -1;
        }
    }
}
public bool Fullscreen
{
    get
    {
        int iIsFullScreen = 0;
        if (GetVariable("fullscreen", ref iIsFullScreen) == Error.Success)
            if (iIsFullScreen != 0)
                return true;
            return false;
    }
    set
    {
        int iFullScreen = value ? 1 : 0;
        SetVariable("fullscreen", iFullScreen);
    }
}
#endregion

#region PUBLIC METHODS

public bool Initialize()
{// check if already initialized
    if (m_iVlcHandle != -1)
        return true;

    // try init
    try
    {
        // create instance
        m_iVlcHandle = VLC_Create();
        if (m_iVlcHandle < 0)
        {
            m_strLastError = "Failed to create VLC instance";
            return false;
        }

        // make init options
        string[] strInitOptions = { "vlc",
        "--no-one-instance",
        "--no-loop",
        "--no-drop-late-frames",
        "--disable-screensaver"};
        if (m_strVlcInstallDir.Length > 0)
            strInitOptions[0] = m_strVlcInstallDir + @"\vlc";
        // init libvlc
        Error errVlcLib = VLC_Init(m_iVlcHandle, strInitOptions.Length, strInitOptions);
    }
}

```

```

        if (errVlcLib != Error.Success)
        {
            VLC_Destroy(m_iVlcHandle);
            m_strLastError = "Failed to initialise VLC";
            m_iVlcHandle = -1;
            return false;
        }
    }
    catch
    {
        m_strLastError = "Could not find libvlc";
        return false;
    }

    // check output window
    if (m_wndOutput != null)
    {
        SetVariable("drawable", m_wndOutput.Handle.ToInt32());
        wndOutput_Resize(null, null);
    }

    // OK
    return true;
}
public Error AddTarget(string Target)
{
    if (m_iVlcHandle == -1)
    {
        m_strLastError = "LibVlc is not initialized";
        return Error.NoInit;
    }
    Error enmErr = Error.Success;
    try
    {
        enmErr = VLC_AddTarget(m_iVlcHandle,
                               Target,
                               null,
                               0,
                               (int)Mode.Append,
                               (int)Pos.End);
    }
    catch (Exception ex)
    {
        m_strLastError = ex.Message;
        return Error.Exception;
    }
    if ((int)enmErr < 0)
    {
        m_strLastError = VLC_Error((int)enmErr);
        return enmErr;
    }
    // OK
    return Error.Success;
}
public Error AddTarget(string Target, string[] Options)
{
    if (m_iVlcHandle == -1)
    {
        m_strLastError = "LibVlc is not initialized";
        return Error.NoInit;
    }
    // check options
    int iOptionsCount = 0;
    if (Options != null)
        iOptionsCount = Options.Length;
    // add
    Error enmErr = Error.Success;
    try
    {
        enmErr = VLC_AddTarget(m_iVlcHandle,
                               Target,
                               Options,
                               iOptionsCount,
                               (int)Mode.Append,
                               (int)Pos.End);
    }
}

```

```

        catch (Exception ex)
    {
        m_strLastError = ex.Message;
        return Error.Execution;
    }
    if ((int)enmErr < 0)
    {
        m_strLastError = VLC_Error((int)enmErr);
        return enmErr;
    }
    // OK
    return Error.Success;
}
public Error Play()
{
    if (m_iVlcHandle == -1)
    {
        m_strLastError = "LibVlc is not initialized";
        return Error.NoInit;
    }
    Error enmErr = Error.Success;
    try
    {
        enmErr = VLC_Play(m_iVlcHandle);
    }
    catch (Exception ex)
    {
        m_strLastError = ex.Message;
        return Error.Execution;
    }
    if ((int)enmErr < 0)
    {
        m_strLastError = VLC_Error((int)enmErr);
        return enmErr;
    }
    // OK
    return Error.Success;
}
public Error Pause()
{
    if (m_iVlcHandle == -1)
    {
        m_strLastError = "LibVlc is not initialized";
        return Error.NoInit;
    }
    Error enmErr = Error.Success;
    try
    {
        enmErr = VLC_Pause(m_iVlcHandle);
    }
    catch (Exception ex)
    {
        m_strLastError = ex.Message;
        return Error.Execution;
    }
    if ((int)enmErr < 0)
    {
        m_strLastError = VLC_Error((int)enmErr);
        return enmErr;
    }
    // OK
    return Error.Success;
}
public Error Stop()
{
    if (m_iVlcHandle == -1)
    {
        m_strLastError = "LibVlc is not initialized";
        return Error.NoInit;
    }
    Error enmErr = Error.Success;
    try
    {
        enmErr = VLC_Stop(m_iVlcHandle);
    }
    catch (Exception ex)

```

```

        {
            m_strLastError = ex.Message;
            return Error.Exception;
        }
        if ((int)enmErr < 0)
        {
            m_strLastError = VLC_Error((int)enmErr);
            return enmErr;
        }
        // OK
        return Error.Success;
    }
    public float SpeedFaster()
    {
        if (m_iVlcHandle == -1)
        {
            m_strLastError = "LibVlc is not initialized";
            return -1;
        }
        //Error enmErr = Error.Success;
        try
        {
            return VLC_SpeedFaster(m_iVlcHandle);
        }
        catch (Exception ex)
        {
            m_strLastError = ex.Message;
            return -1;
        }
    }
    public float SpeedSlower()
    {
        if (m_iVlcHandle == -1)
        {
            m_strLastError = "LibVlc is not initialized";
            return -1;
        }
        //Error enmErr = Error.Success;
        try
        {
            return VLC_SpeedSlower(m_iVlcHandle);
        }
        catch (Exception ex)
        {
            m_strLastError = ex.Message;
            return -1;
        }
    }
    public Error PlaylistNext()
    {
        if (m_iVlcHandle == -1)
        {
            m_strLastError = "LibVlc is not initialized";
            return Error.NoInit;
        }
        Error enmErr = Error.Success;
        try
        {
            enmErr = VLC_PlaylistNext(m_iVlcHandle);
        }
        catch (Exception ex)
        {
            m_strLastError = ex.Message;
            return Error.Exception;
        }
        if ((int)enmErr < 0)
        {
            m_strLastError = VLC_Error((int)enmErr);
            return enmErr;
        }
        // OK
        return Error.Success;
    }
    public Error PlaylistPrevious()
    {

```

```

        if (m_iVlcHandle == -1)
    {
        m_strLastError = "LibVlc is not initialized";
        return Error.NoInit;
    }
    Error enmErr = Error.Success;
    try
    {
        enmErr = VLC_PlaylistPrev(m_iVlcHandle);
    }
    catch (Exception ex)
    {
        m_strLastError = ex.Message;
        return Error.Execution;
    }
    if ((int)enmErr < 0)
    {
        m_strLastError = VLC_Error((int)enmErr);
        return enmErr;
    }
    // OK
    return Error.Success;
}

public Error PlaylistClear()
{
    if (m_iVlcHandle == -1)
    {
        m_strLastError = "LibVlc is not initialized";
        return Error.NoInit;
    }
    Error enmErr = Error.Success;
    try
    {
        enmErr = VLC_PlaylistClear(m_iVlcHandle);
    }
    catch (Exception ex)
    {
        m_strLastError = ex.Message;
        return Error.Execution;
    }
    if ((int)enmErr < 0)
    {
        m_strLastError = VLC_Error((int)enmErr);
        return enmErr;
    }
    // OK
    return Error.Success;
}
public Error TimeSet(int newPosition, bool bRelative)
{
    if (m_iVlcHandle == -1)
    {
        m_strLastError = "LibVlc is not initialized";
        return Error.NoInit;
    }
    Error enmErr = Error.Success;
    try
    {
        enmErr = VLC_TimeSet(m_iVlcHandle, newPosition, bRelative);
    }
    catch (Exception ex)
    {
        m_strLastError = ex.Message;
        return Error.Execution;
    }
    if ((int)enmErr < 0)
    {
        m_strLastError = VLC_Error((int)enmErr);
        return enmErr;
    }
    // OK
    return Error.Success;
}
public float PositionSet(float newPosition)
{

```

```

    if (m_iVlcHandle == -1)
    {
        m_strLastError = "LibVlc is not initialized";
        return -1;
    }
    try
    {
        return VLC_PositionSet(m_iVlcHandle, newPosition);
    }
    catch (Exception ex)
    {
        m_strLastError = ex.Message;
        return -1;
    }
}
public int VolumeSet(int newVolume)
{
    if (m_iVlcHandle == -1)
    {
        m_strLastError = "LibVlc is not initialized";
        return -1;
    }
    //Error enmErr = Error.Success;
    try
    {
        return VLC_VolumeSet(m_iVlcHandle, newVolume);
    }
    catch (Exception ex)
    {
        m_strLastError = ex.Message;
        return -1;
    }
}
public Error VolumeMute()
{
    if (m_iVlcHandle == -1)
    {
        m_strLastError = "LibVlc is not initialized";
        return Error.NoInit;
    }
    Error enmErr = Error.Success;
    try
    {
        enmErr = VLC_VolumeMute(m_iVlcHandle);
    }
    catch (Exception ex)
    {
        m_strLastError = ex.Message;
        return Error.Exception;
    }
    if ((int)enmErr < 0)
    {
        m_strLastError = VLC_Error((int)enmErr);
        return enmErr;
    }
    // OK
    return Error.Success;
}
public Error SetVariable(string strName, Int32 Value)
{
    if (m_iVlcHandle == -1)
    {
        m_strLastError = "LibVlc is not initialized";
        return Error.NoInit;
    }
    Error enmErr = Error.Success;
    try
    {
        // create vlc value
        vlc_value_t val = new vlc_value_t();
        val.i_int = Value;
        // set variable
        enmErr = VLC_VariableSet(m_iVlcHandle, strName, val);
    }
    catch (Exception ex)
    {

```

```

        m_strLastError = ex.Message;
        return Error.Exception;
    }
    if ((int)enmErr < 0)
    {
        m_strLastError = VLC_Error((int)enmErr);
        return enmErr;
    }
    // OK
    return Error.Success;
}
public Error GetVariable(string strName, ref int Value)
{
    if (m_iVlcHandle == -1)
    {
        m_strLastError = "LibVlc is not initialized";
        return Error.NoInit;
    }

    Error enmErr = Error.Success;
    try
    {
        // create vlc value
        vlc_value_t val = new vlc_value_t();
        // set variable
        enmErr = VLC_VariableGet(m_iVlcHandle, strName, ref val);
        Value = val.i_int;
    }
    catch (Exception ex)
    {
        m_strLastError = ex.Message;
        return Error.Exception;
    }
    if ((int)enmErr < 0)
    {
        m_strLastError = VLC_Error((int)enmErr);
        return enmErr;
    }
    // OK
    return Error.Success;
}

public Error ToggleFullscreen()
{
    if (m_iVlcHandle == -1)
    {
        m_strLastError = "LibVlc is not initialized";
        return Error.NoInit;
    }
    Error enmErr = Error.Success;
    try
    {
        enmErr = VLC_FullScreen(m_iVlcHandle);
    }
    catch (Exception ex)
    {
        m_strLastError = ex.Message;
        return Error.Exception;
    }
    if ((int)enmErr < 0)
    {
        m_strLastError = VLC_Error((int)enmErr);
        return enmErr;
    }
    // OK
    return Error.Success;
}

public Error PressKey(string strKey)
{
    if (m_iVlcHandle == -1)
    {
        m_strLastError = "LibVlc is not initialized";
        return Error.NoInit;
    }
    Error enmErr = Error.Success;

```

```

        try
    {
        // create vlc value
        vlc_value_t valKey = new vlc_value_t();
        // get variable
        enmErr = VLC_VariableGet(m_iVlcHandle, strKey, ref valKey);
        if (enmErr == Error.Success)
        {// set pressed
            enmErr = VLC_VariableSet(m_iVlcHandle, "key-pressed", valKey);
        }
    }
    catch (Exception ex)
    {
        m_strLastError = ex.Message;
        return Error.Exception;
    }
    if ((int)enmErr < 0)
    {
        m_strLastError = VLC_Error((int)enmErr);
        return enmErr;
    }
    // OK
    return Error.Success;
}
#endregion

#region PRIVATE METHODS

private string QueryVlcInstallPath()
{
    // open registry
    RegistryKey regkeyVlcInstallPathKey =
Registry.LocalMachine.OpenSubKey(@"SOFTWARE\VideoLAN\VLC");
    if (regkeyVlcInstallPathKey == null)
        return "";
    return (string)regkeyVlcInstallPathKey.GetValue("InstallDir", "");
}
#endregion
#region EVENT METHODS

void wndOutput_Resize(object sender, EventArgs e)
{
    if (m_iVlcHandle != -1)
    {
        SetVariable("conf::width", m_wndOutput.ClientRectangle.Width);
        SetVariable("conf::height", m_wndOutput.ClientRectangle.Height);
    }
}

#endregion

}

private void Form1_Load(object sender, EventArgs e)
{
    progreso = 1;
    tiempo = 60;
    minutos = 15;
    PB2.Value = 0;
    TTB.Enabled = false;
    TIM1.Tick += new EventHandler(this.OnTimerTick);
    PTB.Text = Convert.ToString(100-PTSB.Value * 10);
    ATB.Text = Convert.ToString(60 - PTSB.Value*.5);
    TTTB.Text = DateTime.Now.ToString();
    TIM2.Tick += new EventHandler(this.cambio);
    message = "";
    BP1.Value = 960;
}

private void TIRO_Click(object sender, EventArgs e)

```

```

{
    message = "BOOM";
    ETB.Text = "Sistema en carga, favor de esperar...";
    carga = true;
    BFIN.Enabled = false;
    ARRISOL.Enabled = false;
    ABASOL.Enabled = false;
    ANGCAN.Enabled = false;
    POTCAN.Enabled = false;
    TIRO.Enabled = false;
}

private void PT_Click(object sender, EventArgs e)
{
}

private void BINICIAR_Click(object sender, EventArgs e)
{
    BFIN.Enabled = true;
    ARRISOL.Enabled = true;
    ABASOL.Enabled = true;
    ANGCAN.Enabled = true;
    POTCAN.Enabled = true;

    BINICIAR.Enabled = false;

    PB2.Value = PB2.Value + 1;
    PRTB.Text = Convert.ToString(progreso++);
    TIM1.Enabled = true ;
    tiempo = 60;
    minutos = 14;

    message = "EBANDA";

    ThreadStart hiloxd = new ThreadStart(conexionhilo);
    hiloaux = new Thread(hiloxd);
    hiloaux.Start();

    try
    {
        LibVlc vlc = new LibVlc();
        vlc.Initialize();
        vlc.VideoOutput = V1GB;
        vlc.PlaylistClear();
        string[] options = { ":sout=#duplicate{dst=display,dst=std{access=file,mux=asf,dst=\"C:\\\\MOV000006.avi\"}}" };
        vlc.AddTarget("http://192.168.1.64:8080", options);
        //LISTA DE CANALES
        //mms://live1.wm.skynews.servecast.net/skynews_wmlz_live300k
        //mms://VODSTR.cjmbc.co.kr/LiveTV
        //http://192.168.1.68:8080 "http://10.4.9.233:8080"
        vlc.Play();
    }
    catch (Exception e1)
    {
        MessageBox.Show(e1.ToString());
    }
}
}

private void OnTimerTick(object sender, EventArgs e)
{
    if (minutos < 10)
        MTTB.Text = "0" + Convert.ToString(minutos);

    else
    {
        MTTB.Text = Convert.ToString(minutos);
    }

    tiempo--;
    if (tiempo == 0)
    {

        TTB.Text = Convert.ToString(tiempo);
        tiempo = 60;
    }
}

```

```

        }

        if (tiempo < 10)
        {
            TTB.Text = "0" + Convert.ToString(tiempo);
        }
        else
        {
            if (tiempo == 60)
            {
                TTB.Text = "00";
                minutos--;
                if (minutos == -1)
                {
                    message = "BOOM";
                    carga = true;
                    BFIN.Enabled = false;
                    ARRISOL.Enabled = false;
                    ABASOL.Enabled = false;
                    ANGCAN.Enabled = false;
                    POTCAN.Enabled = false;
                    TIRO.Enabled = false;
                    ETB.Text = "¡Se acabó el tiempo! Esperar a que el sistema regrese";
                    ftiempo = true;
                }
            }
            else
            {
                TTB.Text = Convert.ToString(tiempo);
            }
        }
        BP1.Value--;
    }

    if (carga == true)
    {
        tcarga++;

        if (tcarga == 2)
            message = "PCARGA";

        if (tcarga == 4)//30
        {
            if (ftiempo == true)
            {
                TIM1.Enabled = false;
                TTB.Text = "...";
                MTTB.Text = "...";
                ETB.Text = "El experimento terminó. Presione FINALIZAR";
                BFIN.Enabled = true;
            }
            else
            {
                message = "PTIRO";
                ETB.Text = "Pasando a la posición de disparo...";
            }
        }

        if (tcarga == 6)//60
        {
            PRTB.Text = Convert.ToString(progreso++);
            ETB.Text = "¡Listo! El sistema ya puede utilizarse";

            tcarga = 0;
            carga = false;

            if (PB2.Value < 5)
            {
                ARRISOL.Enabled = true;
                ABASOL.Enabled = true;
                ANGCAN.Enabled = true;
            }
        }
    }
}

```

```

        POTCAN.Enabled = true;
        PB2.Value++;
    }
    else{
        TIM1.Enabled = false;
        TTB.Text = "___";
        MTTB.Text = "___";
        ETB.Text = "Se ha completado el experimento. Presione FINALIZAR";
        BFIN.Enabled = true;
    }
}
}

private void cambio(object sender, EventArgs e)
{
    TTTB.Text = DateTime.Now.ToString();
}

private void BFIN_Click(object sender, EventArgs e)
{
    message = "ABANDA";
    this.Close();
}

private void PTSB_Scroll(object sender, ScrollEventArgs e)
{
    PTB.Text = Convert.ToString(100-PTSB.Value*10);
}

private void ASB_Scroll(object sender, ScrollEventArgs e)
{
    ATB.Text = Convert.ToString(60 - ASB.Value*.5);
}

private void HSB_Scroll(object sender, ScrollEventArgs e)
{
}

private Thread hiloaux;

//Rutinas del socket
private void conexionhilo()
{
    otra:
    int sent;
    IPEndPoint ipep = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 8081);
    Socket server = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

    try
    {
        server.Connect(ipep);
    }
    catch (SocketException)
    {
        n = 2;
        Respuesta();
        return;
    }

    try
    {
        data = ReceiveData(server);
        n = 3;
        Respuesta();
    }
    catch (SocketException)
    {
        n = 4;
        Respuesta();
    }
}

```

```

        return;
    }

    while (true)
    {
        if (message == "exit")
            break;
        try
        {
            sent = SendData(server, Encoding.ASCII.GetBytes(message));
        }
        catch (SocketException)
        {//Si hay excepcion nos salimos pq no hay conexion
            n = 5;
            Respuesta();
            return;
        }

        n = 6;
        Respuesta();
        s++;
        if (s == 5)
            s = 0;
        //server.Shutdown(SocketShutdown.Both);
        //server.Close();
        goto otra;
    }

    private static int SendData(Socket sock, byte[] data)
    {
        int total = 0; //Este es la cantidad enviada del mensaje
        int size = data.Length; // Este es el tamaño del mensaje
        int dataLeft = size; //Este es la parte del mensaje que no se ha enviado
        int sent; //Esto es lo que se envio en esta sesion

        //este sera el tamaño del mensaje que explique arriba
        byte[] dataSize = new byte[4];
        dataSize = BitConverter.GetBytes(size); //obtenemos los bytes de 'size'
        sent = sock.Send(dataSize); //enviamos el tamaño primero, un byte[> de 4

        //ahora enviaremos el mensaje....
        while (total < size)
        {
            //La funcion Send nos devuelve un entero de cuanta cantidad logro enviar
            //Entonces mientras no se haya enviado todo el mensaje este ciclo se
            //ejecuta
            sent = sock.Send(data, total, dataLeft, SocketFlags.None);
            total += sent;
            dataLeft -= sent;
        }
        return total; //regresamos el total que enviamos
    }

    private static byte[] ReceiveData(Socket sock)
    {
        int totalRecv = 0; //la cantidad total recibida
        int recv; //la cantidad recibida individualmente
        byte[] dataSize = new byte[4]; //este sera el tamaño del mensaje en byte[>
        int size; //Este sera el tamaño en entero
        int dataLeft = 0; //Esto nos indica cuanta informacion nos falta recibir

        recv = sock.Receive(dataSize, 0, 4, SocketFlags.None); //recibimos el tamaño
        size = BitConverter.ToInt32(dataSize, 0); //lo pasamos a un entero
        dataLeft = size; //asigamos lo que nos falta recibir
        byte[] data = new byte[size]; //creamos un array de byte[> del tamaño del mensaje

        while (totalRecv < size)
        {
            //empezamos a recibir
            recv = sock.Receive(data, totalRecv, dataLeft, SocketFlags.None);
            if (recv == 0)
            {
                //cuando se reciba 0 es que el cliente se desconecto ...
                //entonces rompemos el ciclo.

```

```

        data = Encoding.ASCII.GetBytes("exit");
        break;
    }
    totalRecv += recv;
    dataLeft -= recv;
}
return data; //regresamos el byte[> que recibimos
}
private void Respuesta()
{
    if (InvokeRequired)
    {
        MethodInvoker method = new MethodInvoker(Respuesta);
        Invoke(method);
        return;
    }
    switch (n)
    {
        case 2:
            ETB.Text = "No se pudo conectar con el servidor. Presione FINALIZAR.";
            ARRISOL.Enabled = false;
            ABASOL.Enabled = false;
            ANGCAN.Enabled = false;
            POTCAN.Enabled = false;
            TIRO.Enabled = false;
            BFIN.Enabled = true;
            TIM1.Enabled = false;
            TTB.Text = "---";
            MTTB.Text = "---";
            break;
        case 3:
            ETB.Text = Encoding.ASCII.GetString(data);
            break;
        case 4:
            ETB.Text = "No se pudo conectar con el servidor. Presione FINALIZAR.";
            ARRISOL.Enabled = false;
            ABASOL.Enabled = false;
            ANGCAN.Enabled = false;
            POTCAN.Enabled = false;
            TIRO.Enabled = false;
            BFIN.Enabled = true;
            TIM1.Enabled = false;
            TTB.Text = "---";
            MTTB.Text = "---";
            break;
        case 5:
            ETB.Text = "Se perdió la conexión con servidor. Presione FINALIZAR.";
            ARRISOL.Enabled = false;
            ABASOL.Enabled = false;
            ANGCAN.Enabled = false;
            POTCAN.Enabled = false;
            TIRO.Enabled = false;
            BFIN.Enabled = true;
            ftiempo = true;
            TTB.Text = "---";
            MTTB.Text = "---";
            break;
        case 6:
            ETB.Text = "Desconectado del servidor. Presione FINALIZAR.";
            ARRISOL.Enabled = false;
            ABASOL.Enabled = false;
            ANGCAN.Enabled = false;
            POTCAN.Enabled = false;
            TIRO.Enabled = false;
            BFIN.Enabled = true;
            ftiempo = true;
            TTB.Text = "---";
            MTTB.Text = "---";
            break;
    }
}
private void ARRISOL_Click(object sender, EventArgs e)

```

```

{
    message = "SIZQ," + Convert.ToString(HSB.Value);
    ETB.Text = "Subir soltador...";
}

private void ABASOL_Click(object sender, EventArgs e)
{
    message = "SDER," + Convert.ToString(HSB.Value);
    ETB.Text = "Bajar soltador...";
}

private void POTCAN_Click(object sender, EventArgs e)
{
    ARRISOL.Enabled = false;
    ABASOL.Enabled = false;
    ANGCAN.Enabled = false;
    POTCAN.Enabled = false;
    TIRO.Enabled = true;
    message = "POT," + Convert.ToString(PTSB.Value);

    ETB.Text = "La potencia se ha ajustado al " + PTB.Text + "%, presionar el botón de TIRO";
}

private void ANGCAN_Click(object sender, EventArgs e)
{
    message = "ANG," + ATB.Text;

    ETB.Text = "Se ajusta el cañón a un ángulo de " + ATB.Text + "°";
}

}

```

REFERENCIAS

- [1] **García Ordaz, Ragnar Ulises**, *Diseño y construcción de un lanzaproyectiles automático para la práctica de tiro parabólico.*, Tesis de licenciatura, Facultad de Ingeniería, UNAM, México, 2008.
- [2] **Hernández Delgado, Armando y Cuevas Domínguez, Verónica Alejandra**, *Instrumentación y control para la operación remota de la práctica de trabajo y energía*, Tesis de licenciatura, Facultad de Ingeniería, UNAM, México, 2009.
- [3] **Creus Solé, Antonio**, *Instrumentación Industrial*, Ed. Alfaomega, México 2003.
- [4] **Monroy Cano, Abraham Israel**, *Desarrollo de una interfaz USB para el control remoto de sistemas electromecánicos*, Tesis de licenciatura, Facultad de Ingeniería, UNAM, México 2008.
- [5] **Axelson, Jan**, *USB Complete: Everything you need to control custom USB peripherals*, ISBN13 978-1-931448-03-1, Lakeview Research LLC, Estados Unidos, 2005, pp. 3-10.
- [6] **García Breijo, Eduardo**, *Compilador C CCS y simulador PROTEUS para Microcontroladores PIC*, Ed. Alfaomega, México 2008.
- [7] http://picmania.garcia-cuervo.net/usb_1_bulktransfers.php, Transferencia bulk para PIC18F4550, consulta en línea en febrero 2010.
- [8] <http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en010300>, Especificaciones del PIC18F4550, consulta en línea en febrero 2010
- [9] <http://www.monografias.com/trabajos17/motor-paso-a-paso/motor-paso-a-paso.shtml>, Funcionamiento del motor PAP, consulta en línea en febrero 2010.
- [10] http://www.sonelec-musique.com/logiciels_proteus_lib_en.html, Librerías para PROTEUS, consulta en línea en febrero 2010.
- [11] <http://www.ladelec.com/practicas/circuitos-analogos/94-puente-de-h-con-mosfet-complementarios.html>, Esquemático del puente H con MOSFET, consulta en línea en febrero 2010.
- [12] <http://datasheetz.com/ONSEMI/NTP60N06-D/NTP60N06-D.html>, Hoja de datos del MOSFET canal N NTP0N6, consulta en línea en febrero 2010.
- [13] http://foro.elhacker.net/net/tutorial_sockets_en_c-t165986.0.html,

Tutorial de sockets en C#, consulta en línea en febrero 2010.

[14] <http://www.codeproject.com/KB/audio-video/cameraviewer.aspx>,
Adaptación de cámara Web a interfaces gráficas, consulta en línea en marzo de 2010.

[15] <http://weblogs.asp.net/nleghari/articles/webcam.aspx>,
Adaptación de cámara Web a interfaces gráficas, consulta en línea en marzo de 2010.

[16] <http://www.news2news.com/vfp/?example=437&ver=vcs>,
Adaptación de cámara Web a interfaces gráficas, consulta en línea en marzo de 2010.

[17] <http://www.videolan.org/vlc/>,
Reproductor de video VLC, consulta en línea en marzo de 2010.

[18] <http://csharpmagics.blogspot.com/>,
Transmisión de video con VLC en C#, consulta en línea en marzo de 2010.

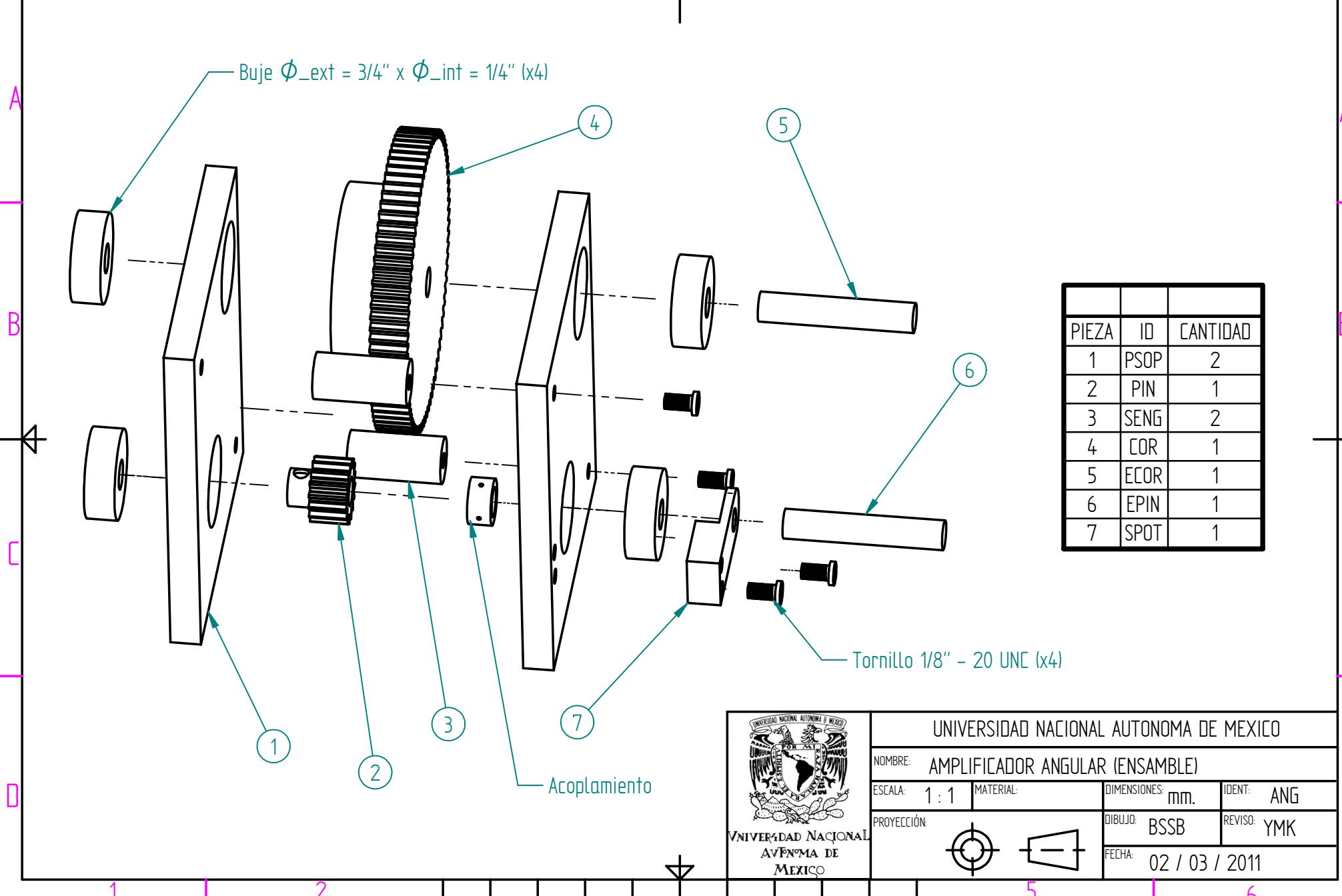
[19] <http://geraldnaveen.blogspot.com/2009/04/streaming-webcam-using-vlc.html>,
Montar servidor de video VLC, consulta en línea en marzo de 2010.

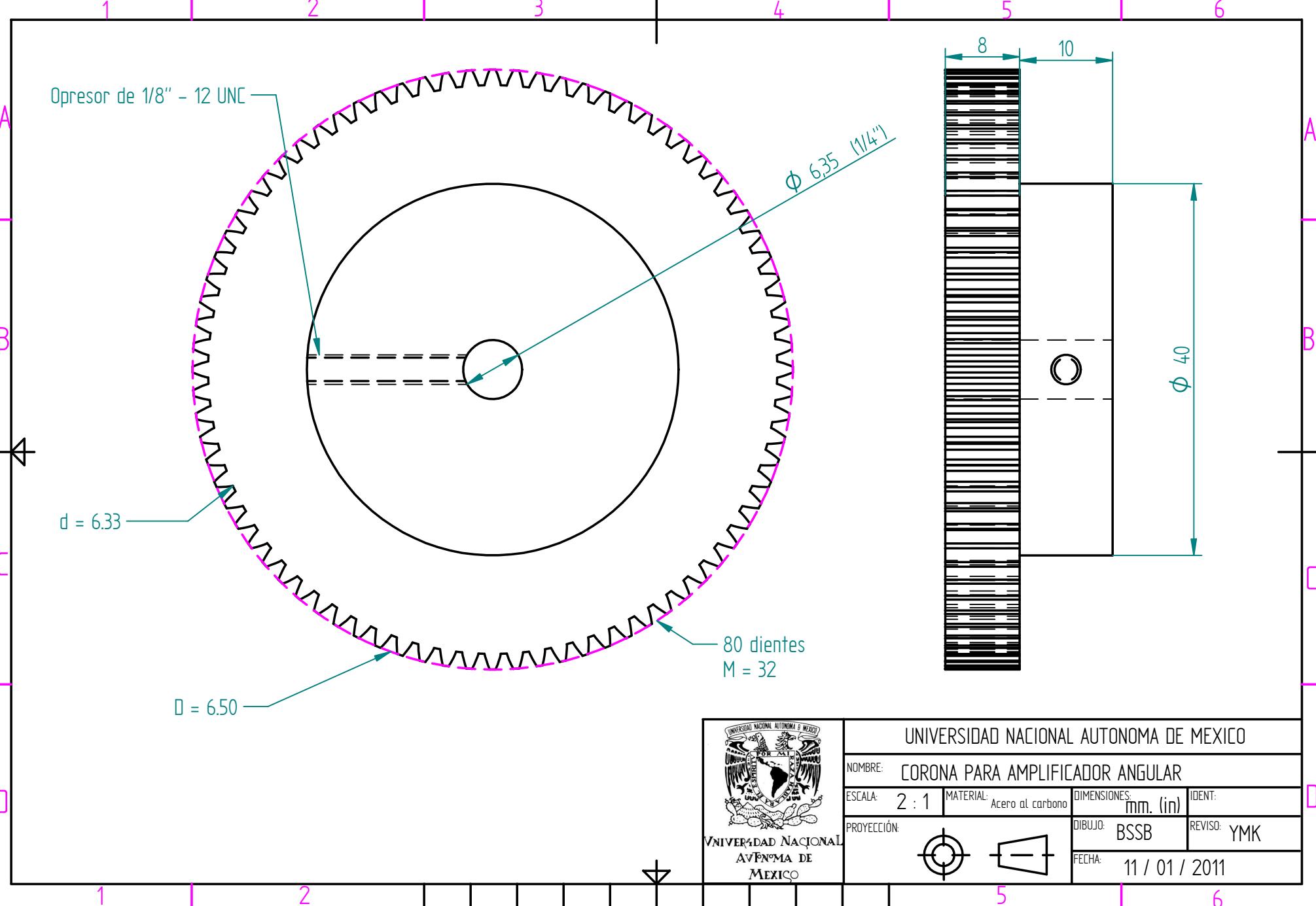
[20] <http://www.ingenia-cat.com/reference/learn/TEC.PAP.5379191030.pdf>,
Encoder incremental mediante sensores de efecto Hall, consulta en línea en marzo 2010.

[21] <http://www.datasheetcatalog.com/>,
Hojas de datos de los dispositivos electrónicos utilizados, consulta en línea en abril 2010.

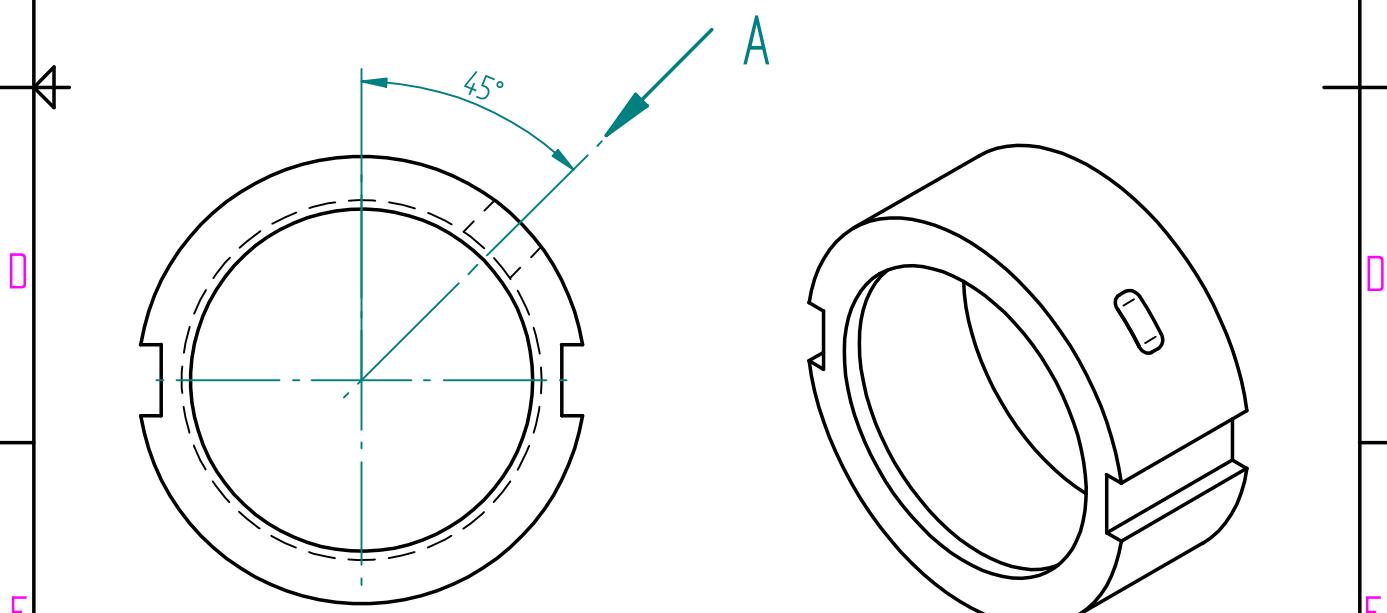
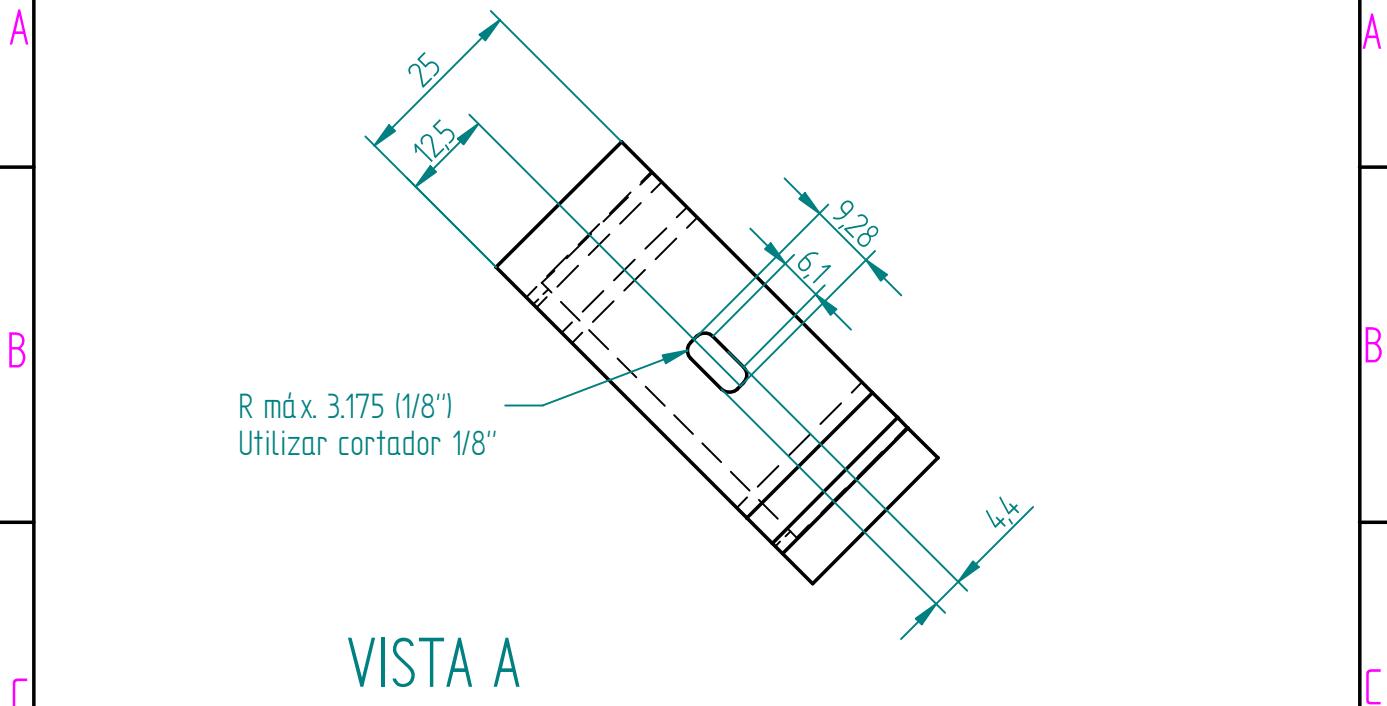
[22] <http://usdigital.com/>,
Encoders rotatorios incrementales, consulta en línea en abril 2010.

1 | 2 | 3 | 4 | 5 | 6





1 2 3 4



	UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO			
NOMBRE:	BOCA DEL CAÑON C/RANURA PARA SENSOR			
ESCALA:	1 : 1	MATERIAL:	Aluminio	DIMENSIONES:
PROYECCION:			m m (in)	IDENT: DP
			DIBUJO: BSSB	REVISIO: YMK
			FECHA: 09 / 02 / 2010	

1 2 3 4

1 2 3 4

A

B

C



D

E

F



UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

NOMBRE: SOPORTE PARA VASTAGO

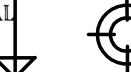
ESCALA: 1 : 1

MATERIAL: Nylomaq

DIMENSIONES: mm (in)

IDENT: DT1

PROYECCIÓN:

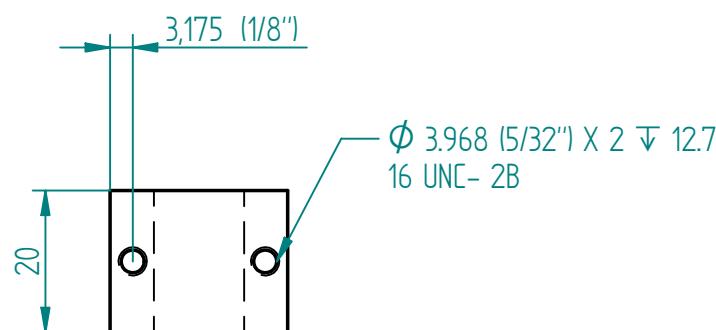
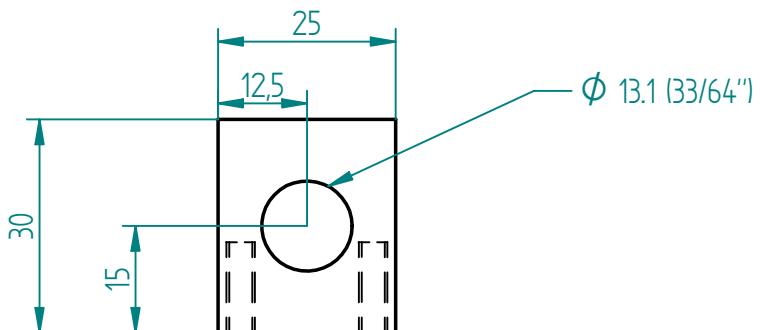


DIBUJO: BSSB

REVISÓ: YMK

FECHA:

09 / 02 / 2010



1

4

1

2

3

4

A

B

C

D

E

F

A

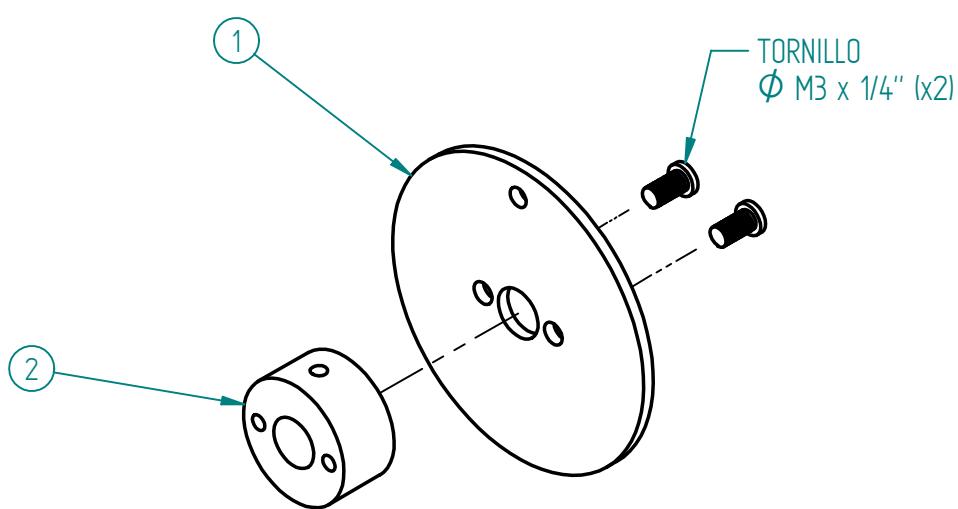
B

C

D

E

F



PIEZA	ID	CANTIDAD
1	EC-3	1
2	EC-4	1



UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

NOMBRE:

ENCODER INCREMENTAL (DISCO)

ESCALA:

1 : 1

MATERIAL:

DIMENSIONES:

mm (in)

IDENT:

EC-E1

PROYECCIÓN:

DIBUJO:

REVISÓ:

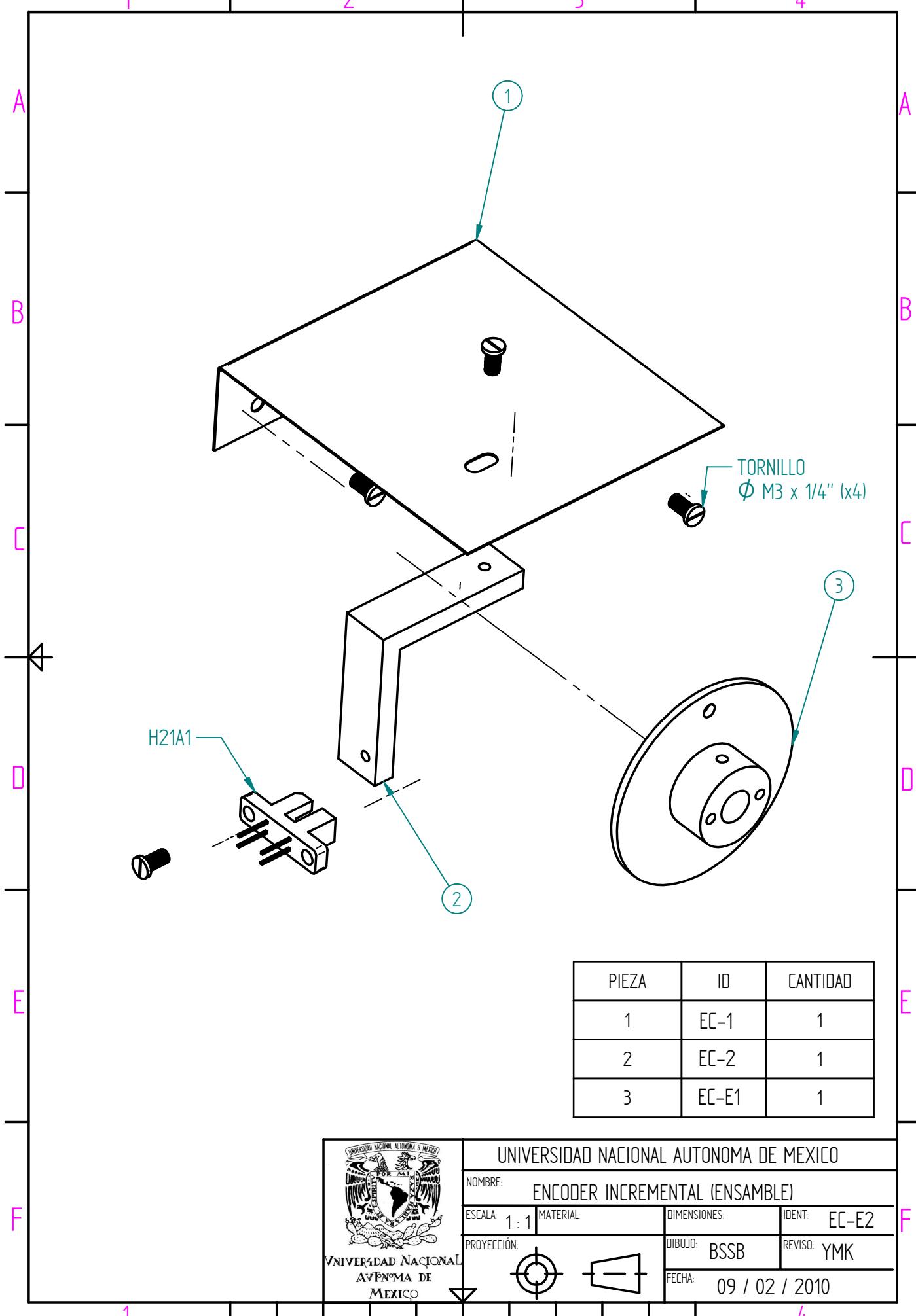
YMK

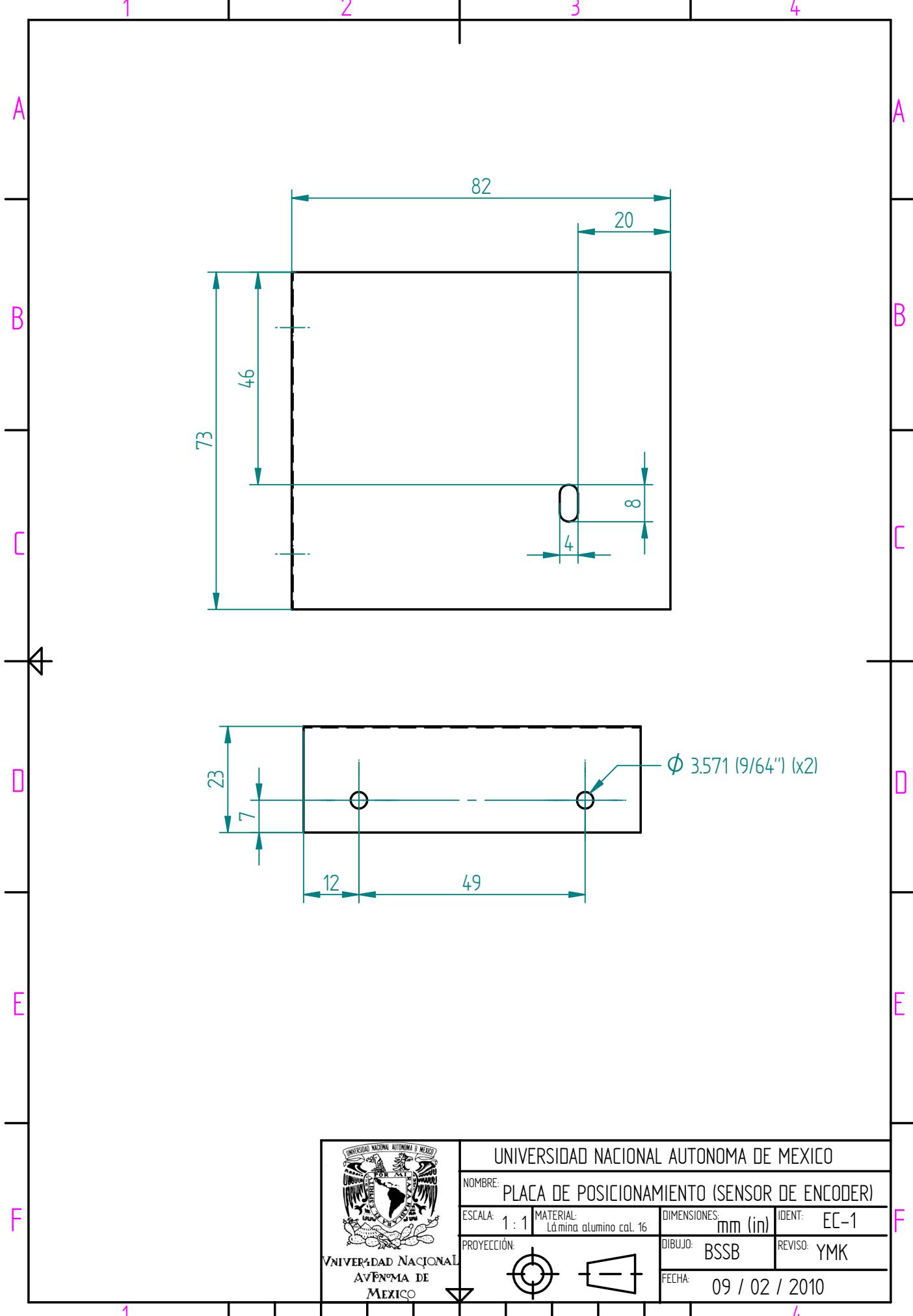
FECHA:

09 / 02 / 2010

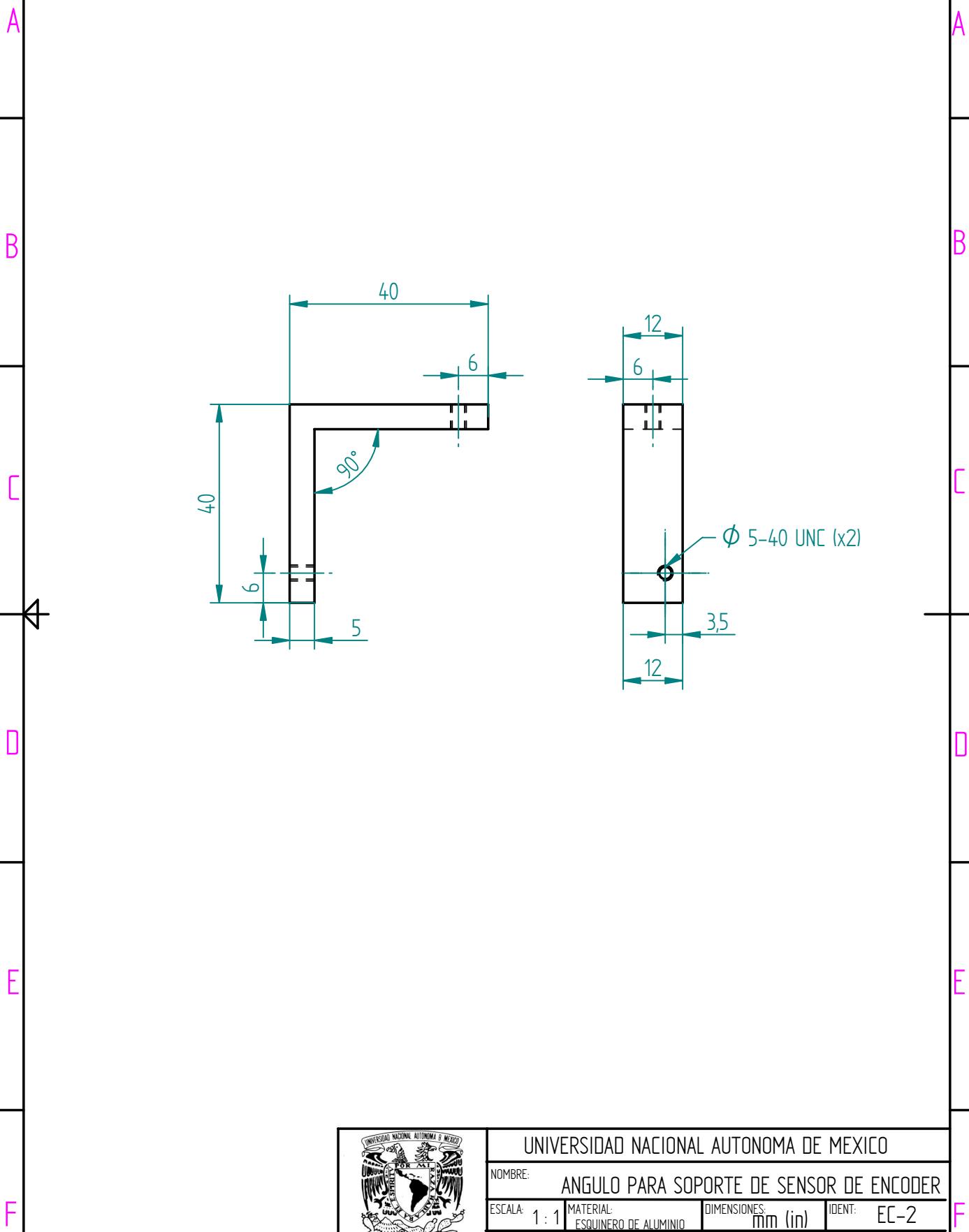
1

4





1 2 3 4

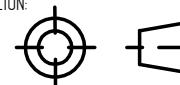


UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

NOMBRE: ANGULO PARA SOPORTE DE SENSOR DE ENCODER

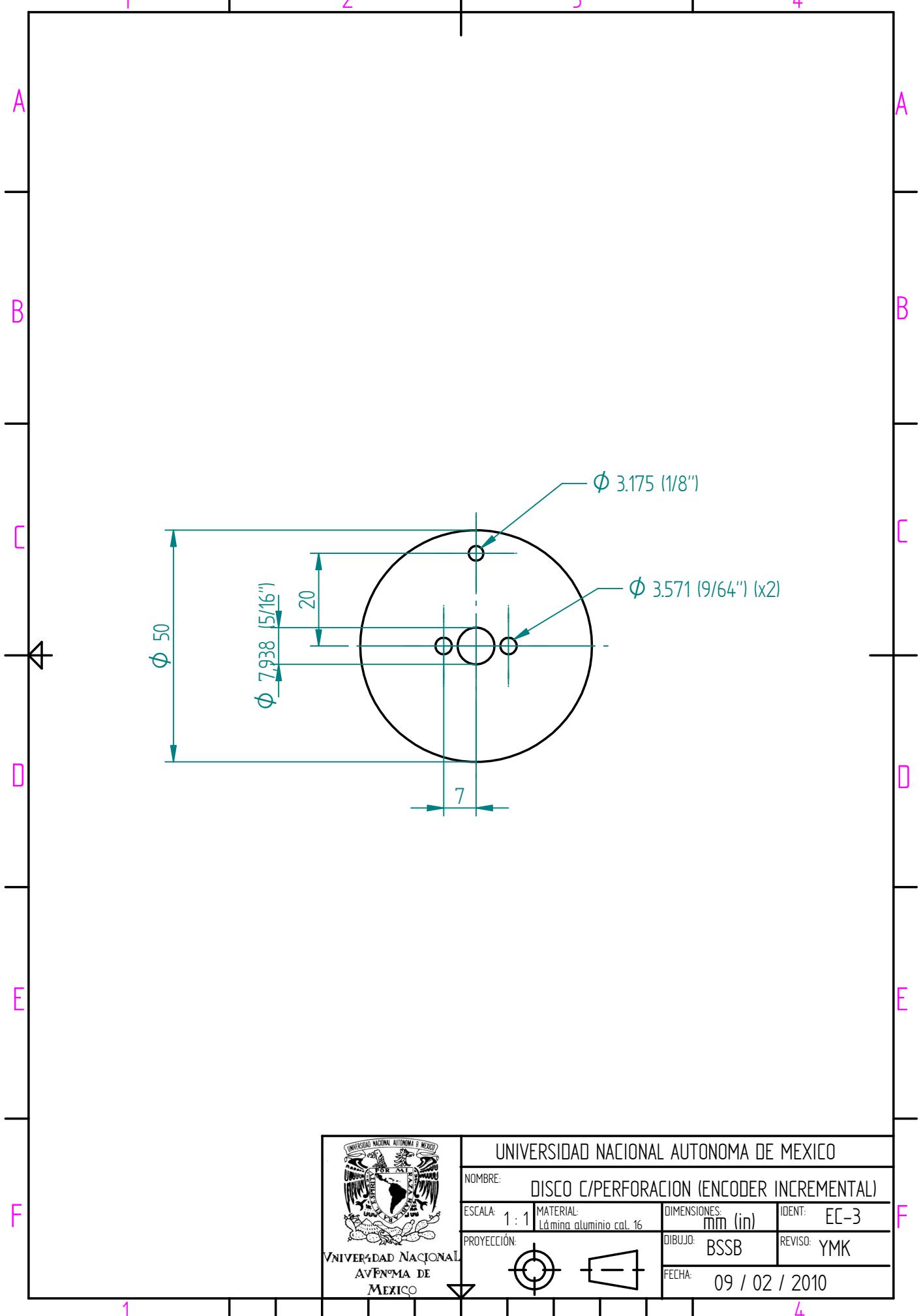
ESCALA: 1 : 1 MATERIAL: ESQUINERO DE ALUMINIO DIMENSIONES IDENT: EC-2

PROYECCION: DIBUJO: BSSB REVISIO: YMK



FECHA: 09 / 02 / 2010

1 2 3 4



1

2

3

4

A

B

C

D

E

F

A

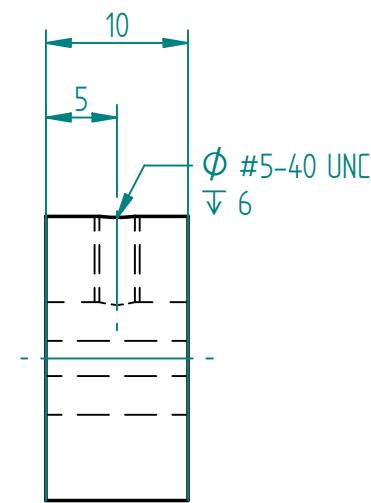
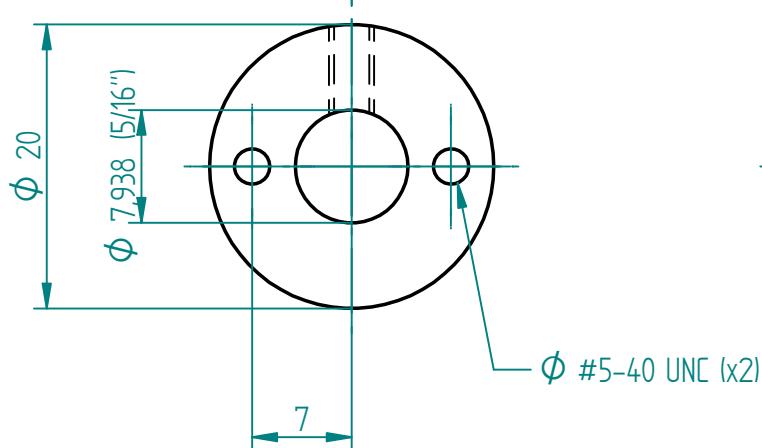
B

C

D

E

F



UNIVERSIDAD NACIONAL AUTÓNOMA DE MEXICO

NOMBRE:	DISCO C/PERFORACION (ENCODER INCREMENTAL)		
ESCALA:	2 : 1	MATERIAL:	Aluminio
PROYECCIÓN:		DIMENSIONES: mm (in)	IDENT.: EC-4
		DIBUJO: BSSB	REVISÓ: YMK
		FECHA:	09 / 02 / 2010

1

4

1

2

3

4

A

B

C

D

E

F

A

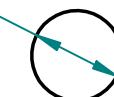
B

C

D

E

F

 $\phi 6.35$ 

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

NOMBRE:

EJE PARA CORONA

ESCALA:

2 : 1

MATERIAL:

Aluminio

DIMENSIONES:

mm (in)

IDENT:

ECOR

PROYECCIÓN:

DIBUJO: BSSB

REVISÓ: YMK



FECHA:

10 / 01 / 2011

1

4

1

2

3

4

A

B

C

D

E

F

A

B

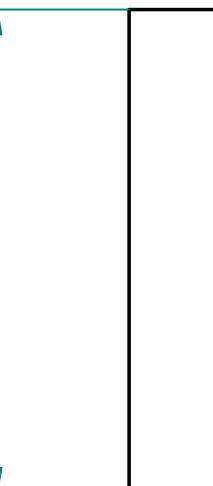
C

D

E

F

3/4

 $\phi 6,35$ 

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

NOMBRE: EJE PARA PIÑÓN

ESCALA: 2 : 1

MATERIAL: Aluminio

DIMENSIONES: mm (in)

IDENT: EPIN

PROYECCIÓN:

DIBUJO: BSSB

REVISÓ: YMK

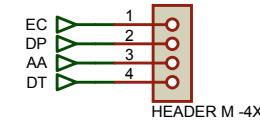
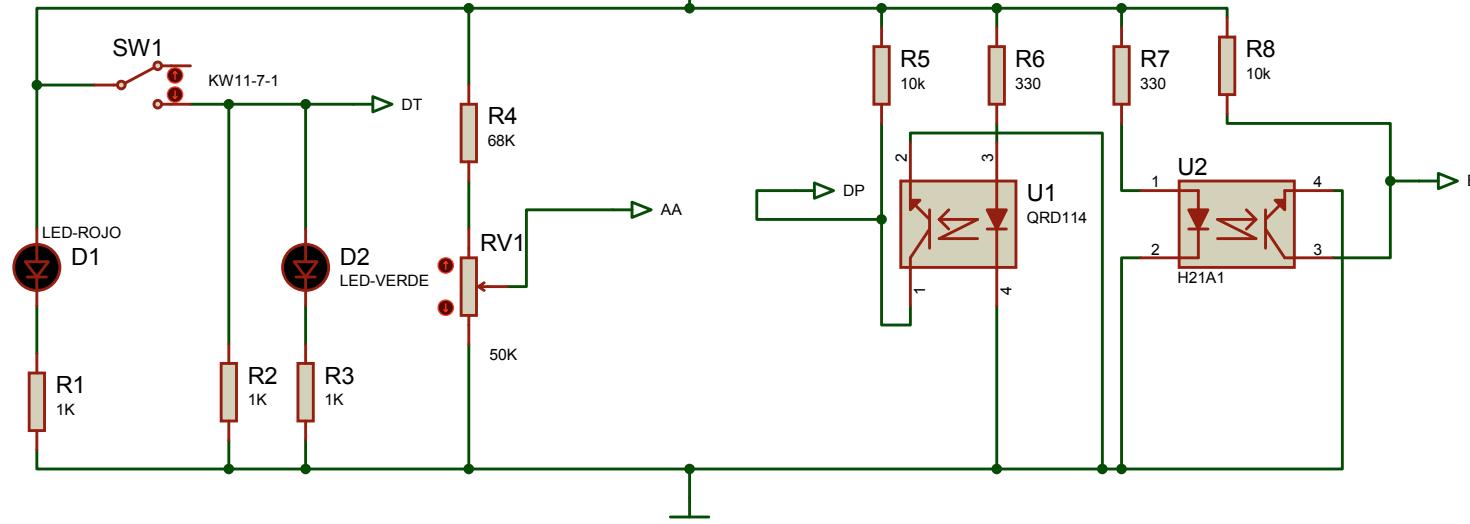


FECHA: 12 / 01 / 2011

1

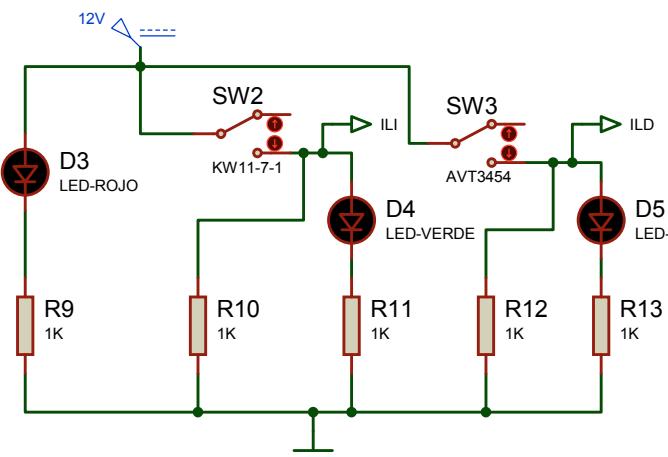
4

MEC-1

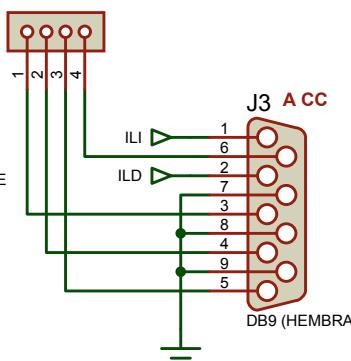


A MEC-1

MEC-2



J2 SEÑALES MEC-1
HEADER M - 4X1



DB9 (HEMBRA)

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

NOMBRE:

MÓDULO EXTERNO DEL CAÑÓN (MEC)

FECHA:

10/07/2010

DIBUJÓ:

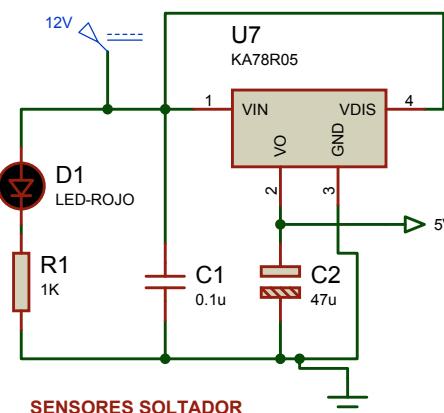
B S S B

REVISÓ: Y M K

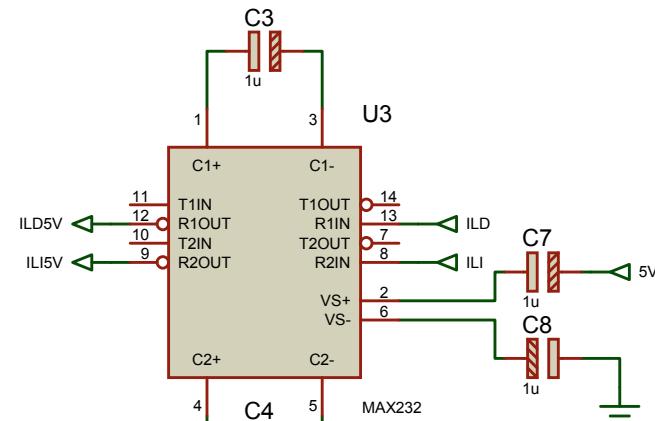
5

6

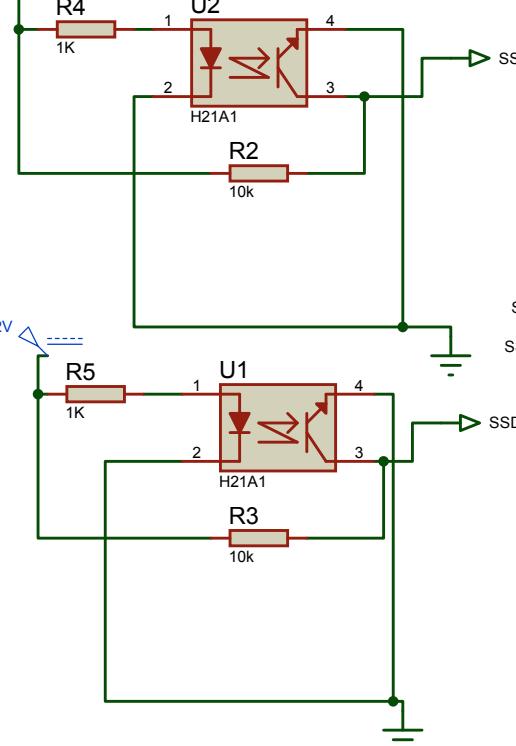
REGULACION A 5V



MODULACIÓN DE SEÑALES A 5V

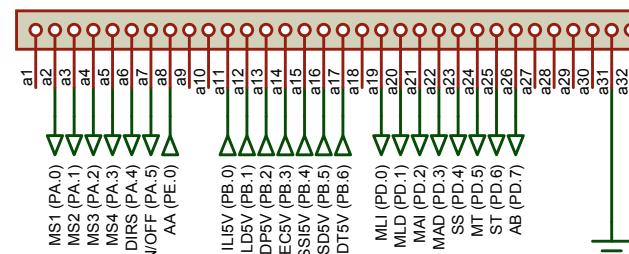


SENSORES SOLTADOR

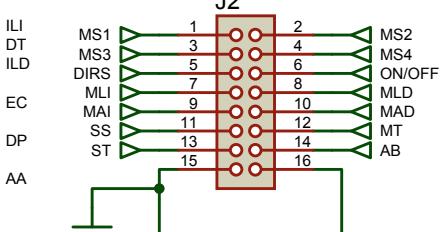


J3

HEADER H - 32X1 RANURA P/TARJETA V0308



CONECTOR 16 ENTRADAS



UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

NOMBRE:

CIRCUITO DE CONTROL (CC)

FECHA:

10/07/2010

DIBUJO:

B S S B

REVISÓ:

Y M K

2

3

4

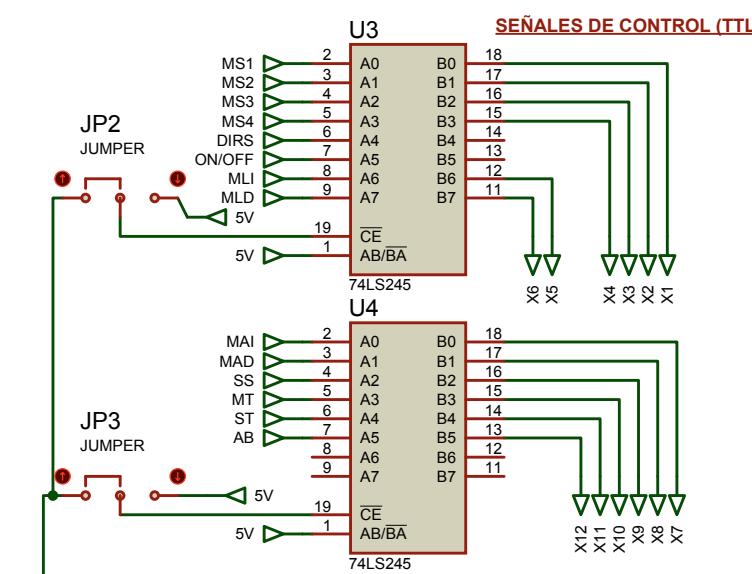
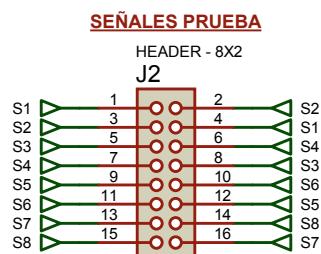
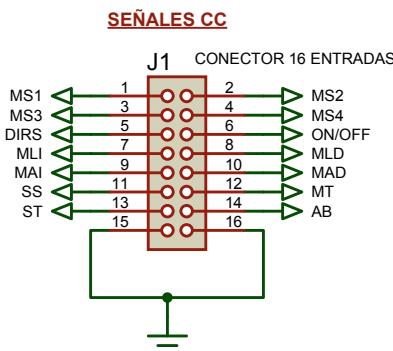
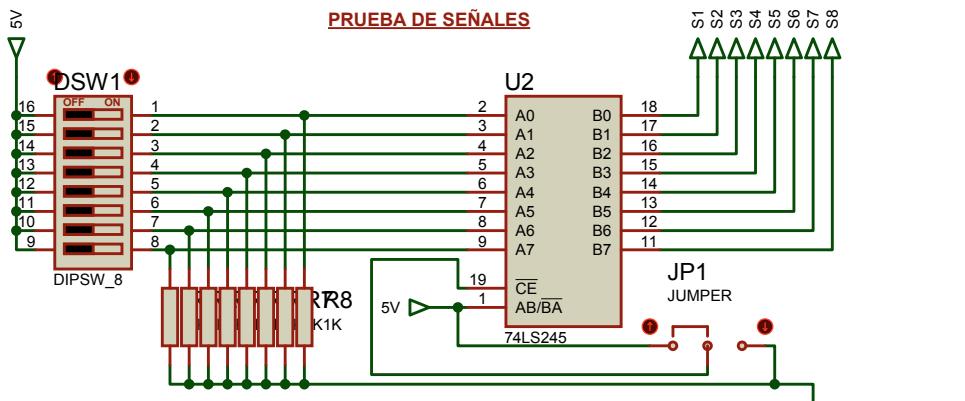
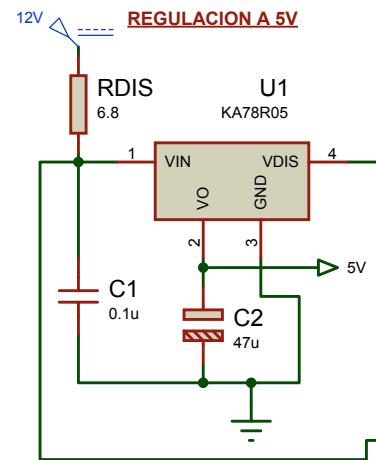
5

6

2

5

6



UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

NOMBRE:

CIRCUITO DE POTENCIA (CP) I/3

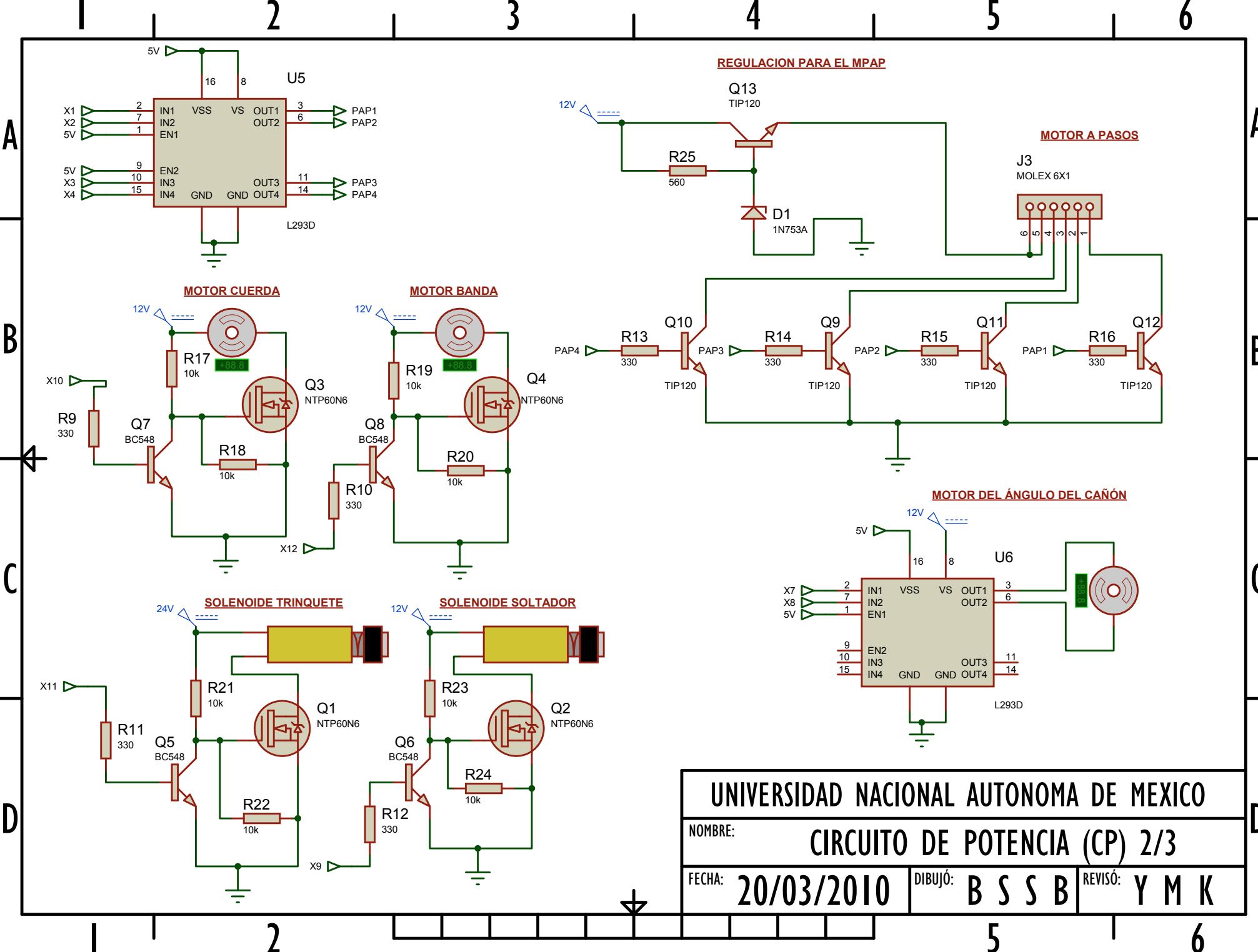
FECHA:

10/07/2010

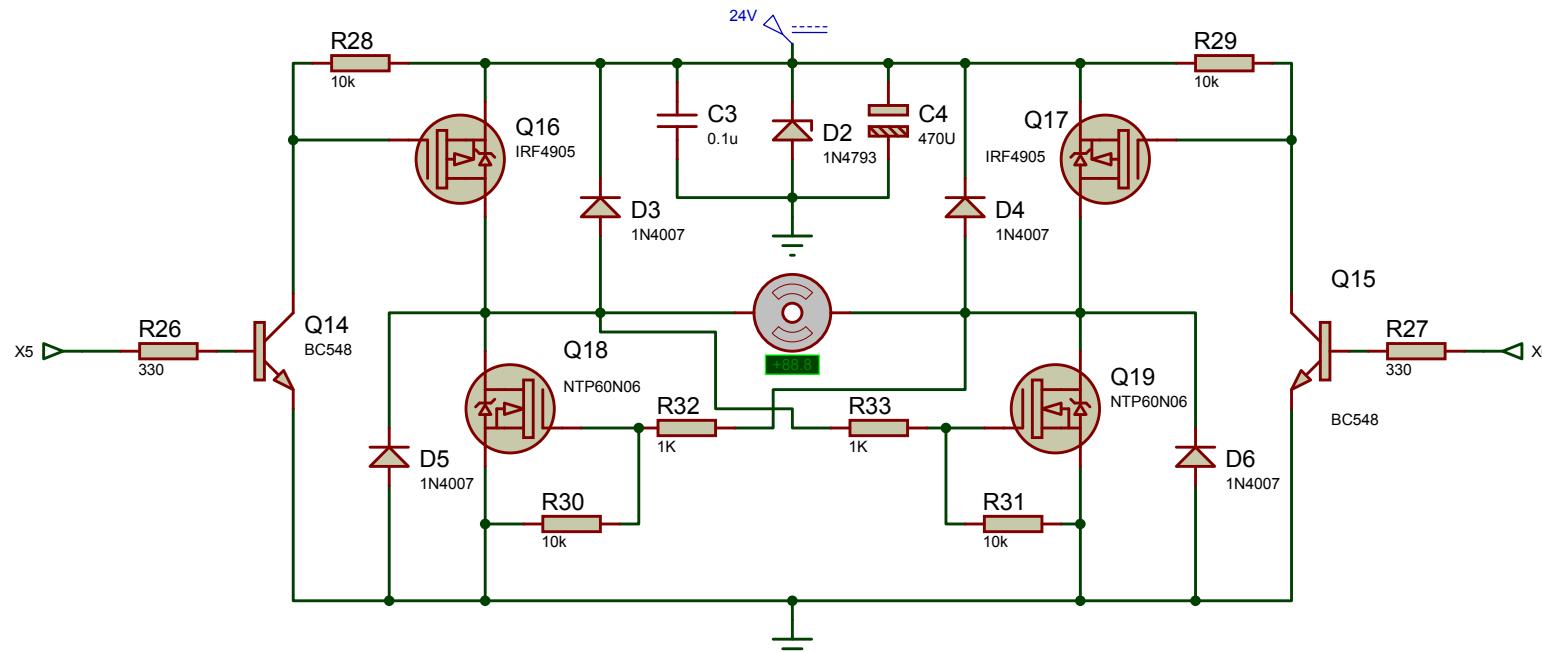
DIBUJO:

B S S B

REVISÓ: Y M K



PUENTE H MOSFET (MOTOR LATERAL DEL CAÑÓN)



UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

NOMBRE:

CIRCUITO DE POTENCIA (CP) 3/3

FECHA:

10/07/2010

DIBUJO:

B S S B

REVISÓ: Y M K

1

2

3

4

A

A

B

B

C

C

D

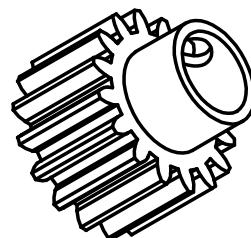
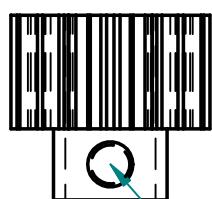
D

E

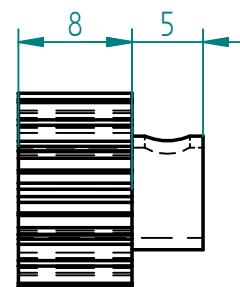
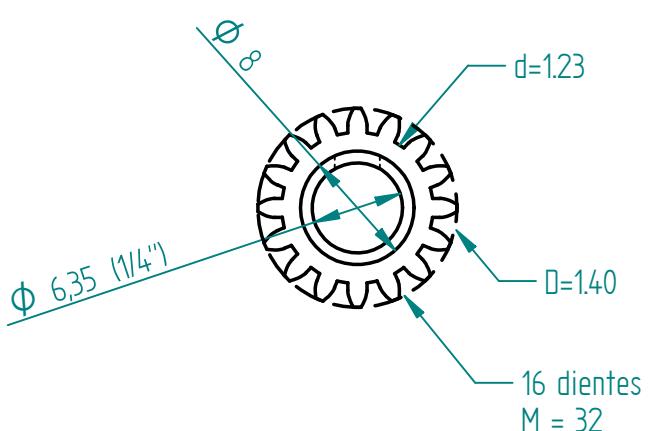
E

F

F



Opresor de 1/8" - 20 UNC

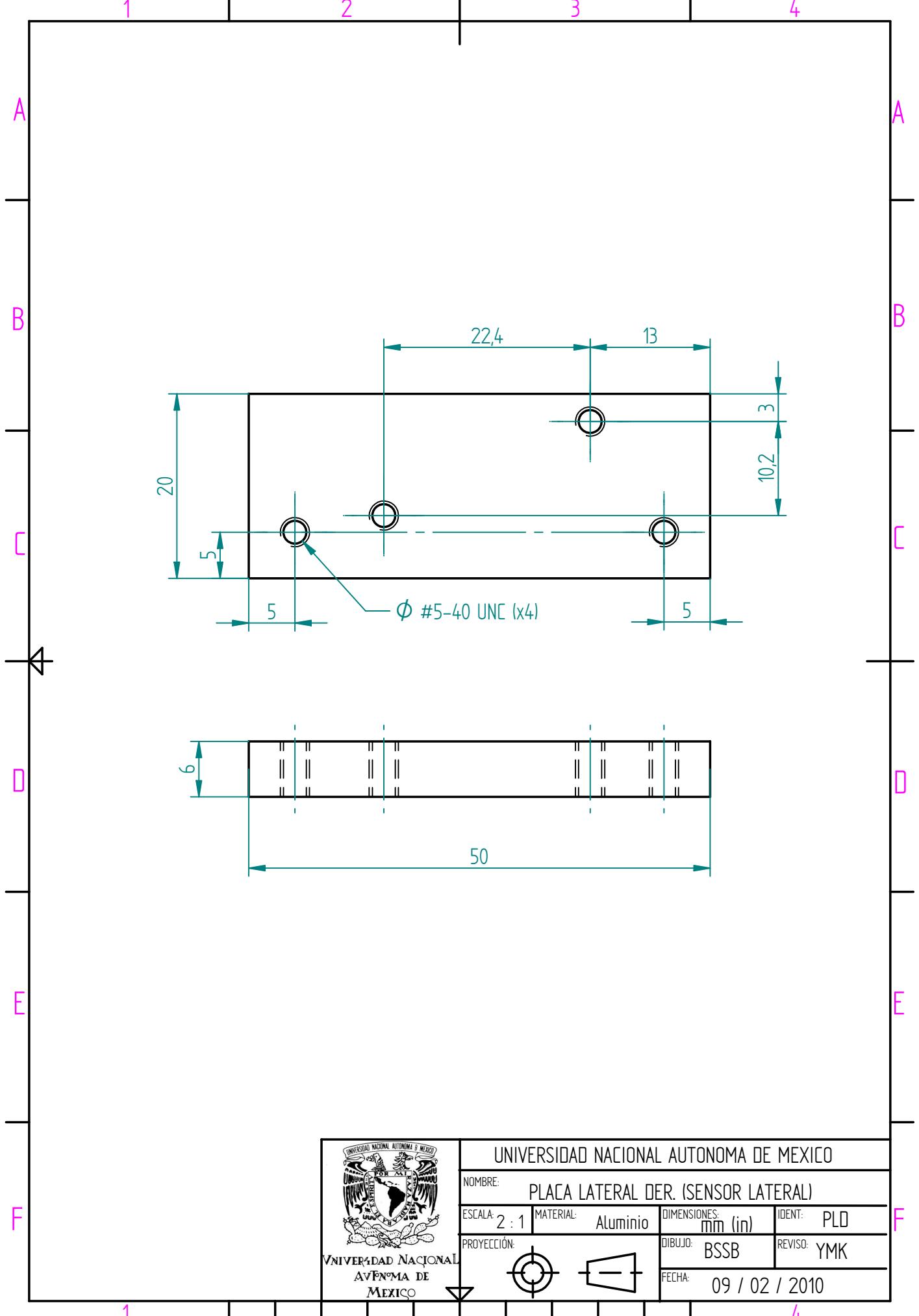


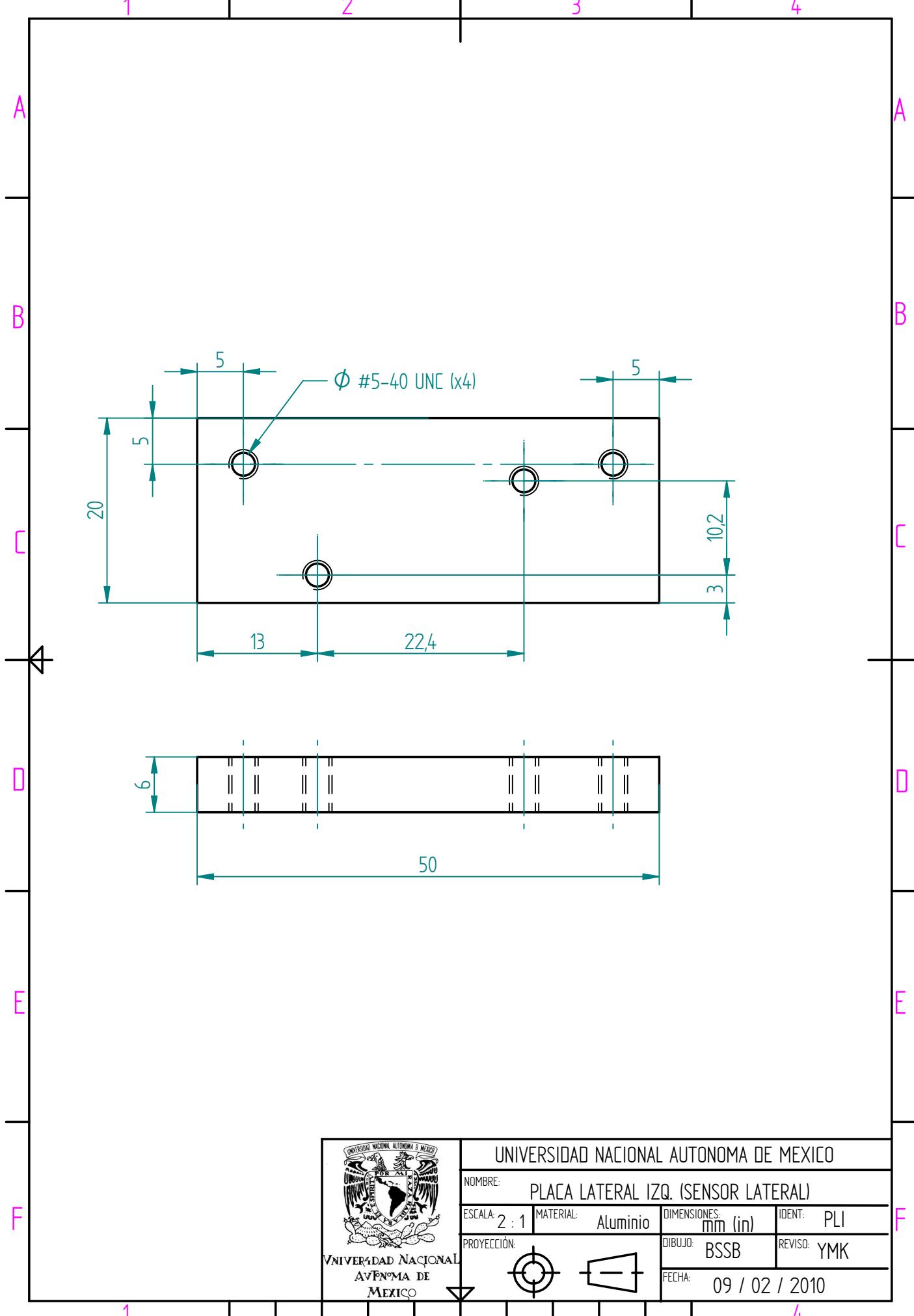
UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

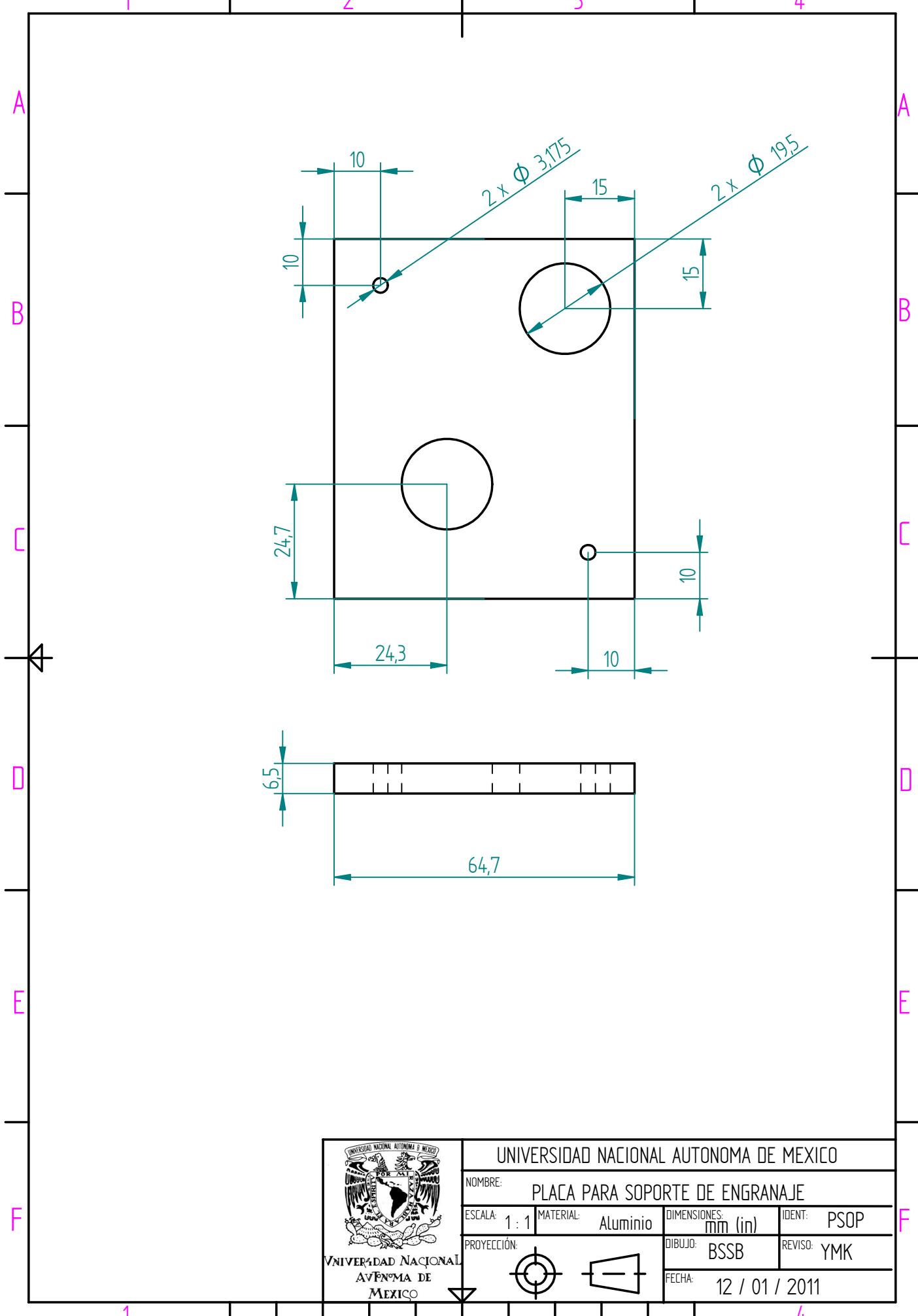
NOMBRE:	PIÑON PARA AMPLIFICADOR ANGULAR		
ESCALA:	2 : 1	MATERIAL:	Acero al carbono
PROYECCIÓN:		DIMENSIONES:	mm (in)
		IDENT.:	PIN
		DIBUJO:	BSSB
		REVISÓ:	YMK
		FECHA:	12 / 01 / 2011

1

4







1

2

3

4

A

B

C

D

E

F

A

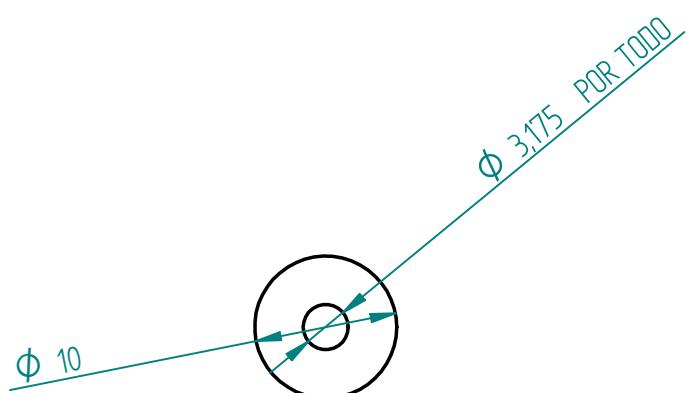
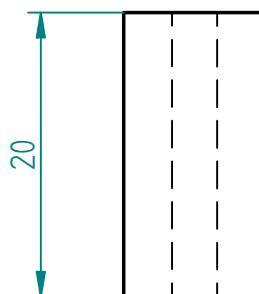
B

C

D

E

F



UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

NOMBRE: SEPARADOR DE PLACAS PARA ENGRANAJE

ESCALA: 2 : 1

MATERIAL: Aluminio

DIMENSIONES: mm (in)

IDENT: SENG

PROYECCION:



DIBUJO: BSSB

REVISIO: YMK

FECHA:

12 / 01 / 2011

1

4

1

2

3

4

A

A

B

B

C

C

D

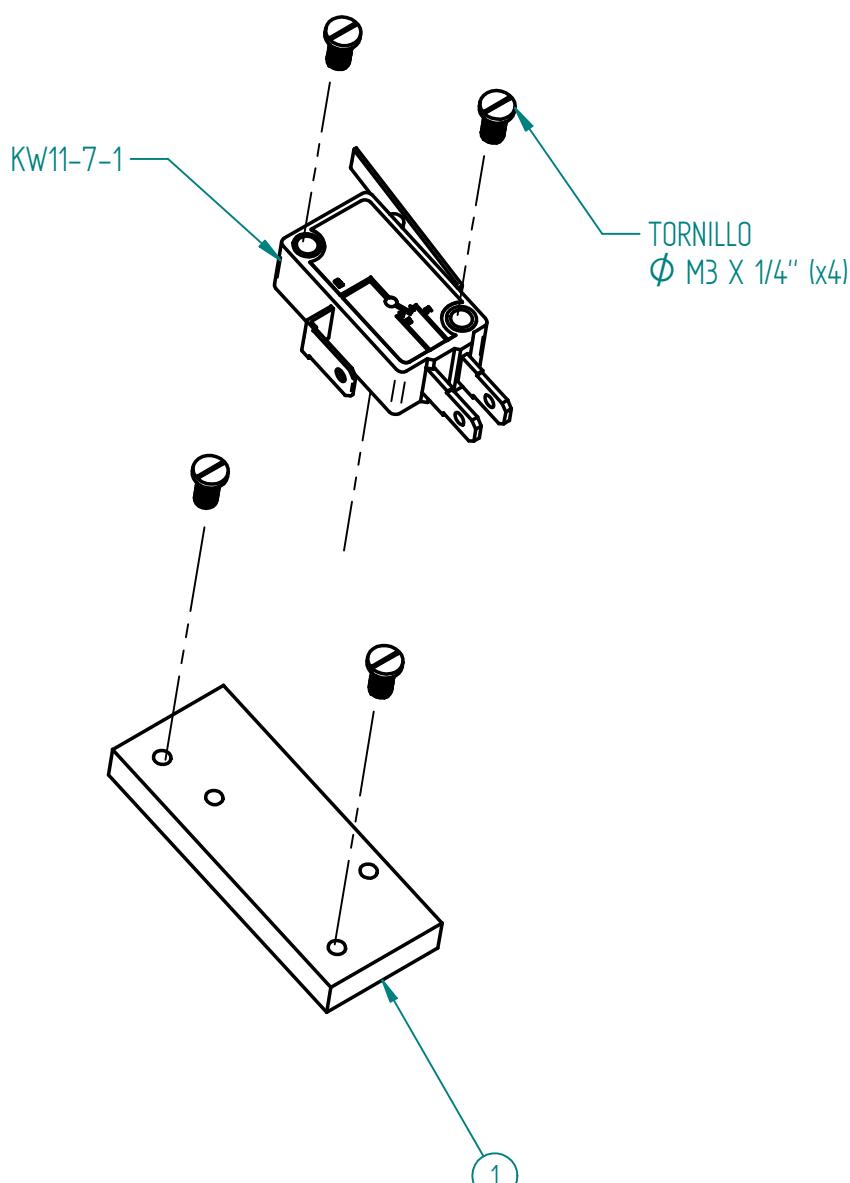
D

E

E

F

F



PIEZA	ID	CANTIDAD
1	PLD	1



UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

NOMBRE: SENSOR LATERAL DERECHO (ENSAMBLE)

ESCALA: 1 : 1

MATERIAL:

DIMENSIONES:

mm (in)

IDENT:

SLD

PROYECCIÓN:



DIBUJO:

BSSB

REVISÓ:

YMK

FECHA:

09 / 02 / 2010

1

4

1

2

3

4

A

B

C

D

E

F

A

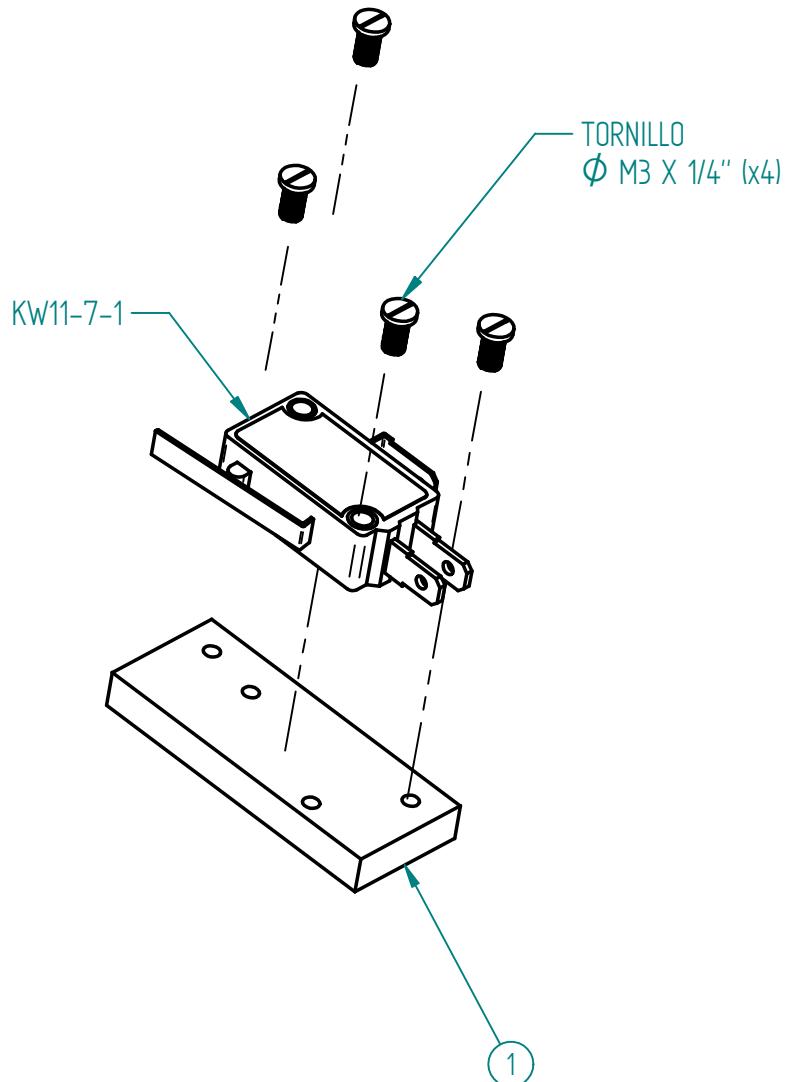
B

C

D

E

F



PIEZA	ID	CANTIDAD
1	PLI	1



UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

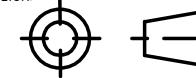
NOMBRE: SENSOR LATERAL IZQUIERDO (ENSAMBLE)

ESCALA: 1 : 1 MATERIAL:

DIMENSIONES: IDENT: SLI

PROYECCIÓN:

DIBUJO: BSSB REVISÓ: YMK

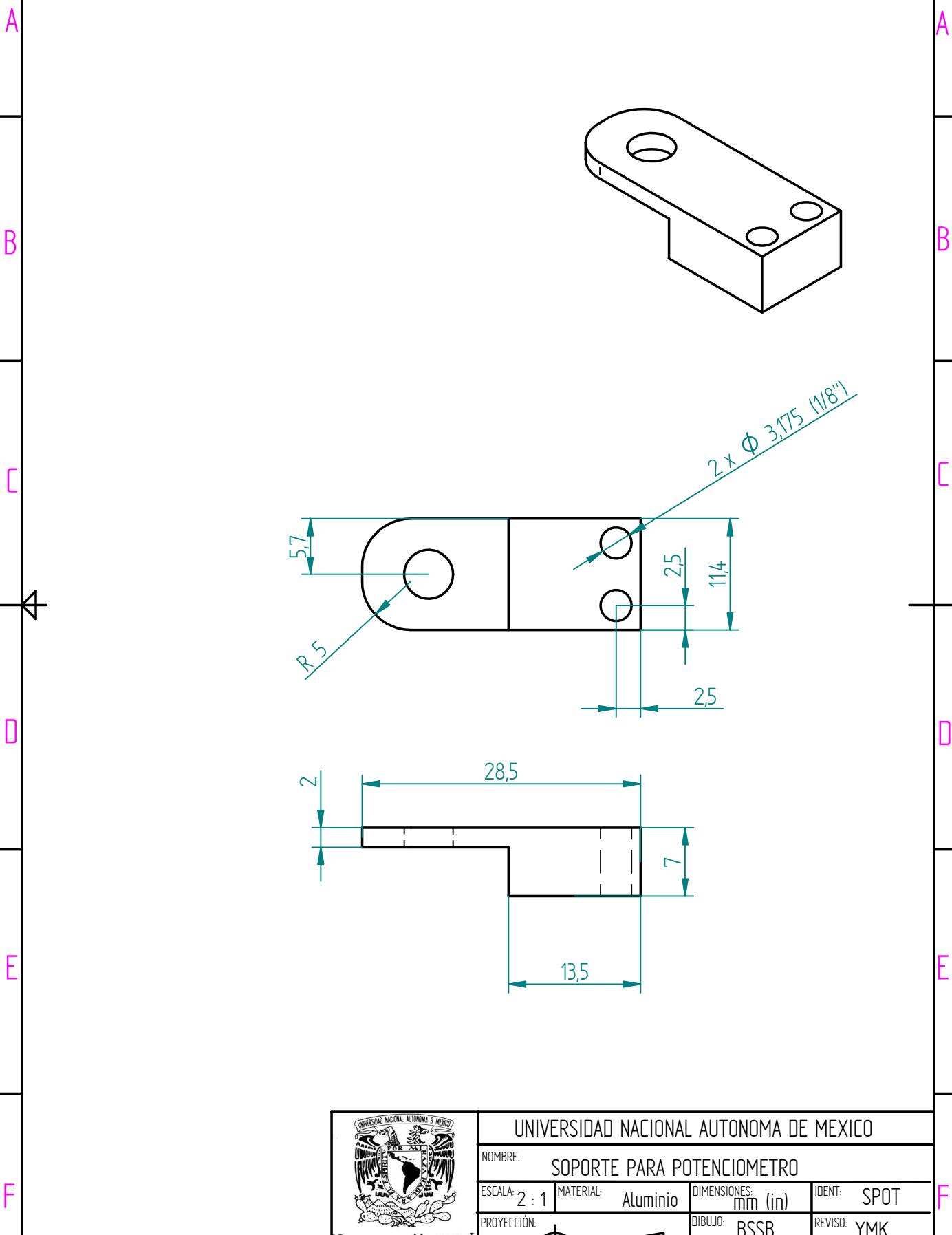


FECHA: 09 / 02 / 2010

1

4

1 2 3 4



1 2 3 4