USERS

INCLUYE FOTOGRAFÍAS DE CADA INSTANCIA DE LOS PROYECTOS

ROBÓTICA AVANZADA

CONSTRUCCIÓN Y PROGRAMACIÓN DE ROBOTS

PROYECTOS REALES
GUANTE DE REALIDAD VIRTUAL
SONAR ROBÓTICO
ROSTRO ANIMATRÓNICO
¡Y MÁS!
★★★★★

- EL MICROCONTROLADOR PHIDGET
- ENSAMBLAJE DE BATERÍAS, MOTORES Y ENGRANAJES
- USO DE ELEMENTOS CASEROS PARA LA CONSTRUCCIÓN
- BRAZOS ROBÓTICOS, Y SISTEMA DE VISIÓN Y SONAR
- PROGRAMACIÓN DE LA INTERFAZ GRÁFICA DE CONTROL
- SISTEMAS MECÁNICOS PARA DEDOS, OJOS, CEJAS Y MANDÍBULA

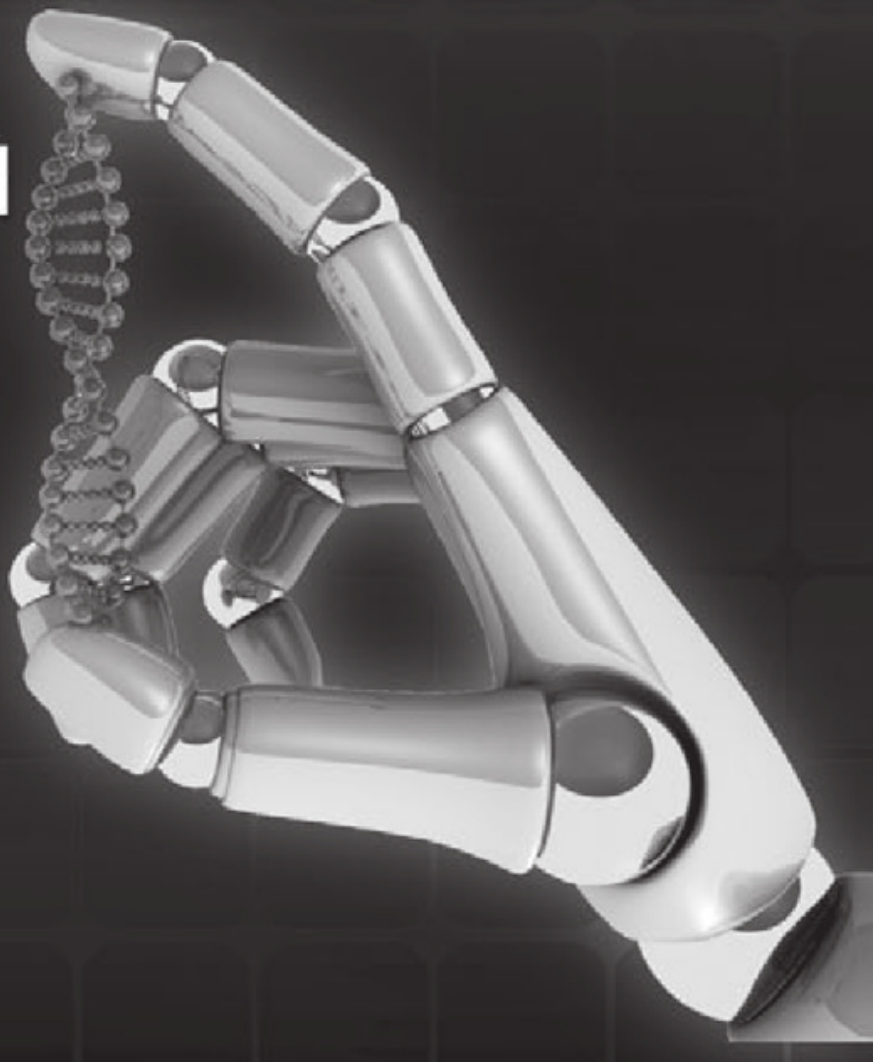
Nicolás Arrioja Landa Cosio

MANUALES USERS MANUALES USERS MANUALES USERS MANUALES USERS MA

¡APLIQUE TODO EL PODER DE C# A SUS CREACIONES!

ROBÓTICA AVANZADA

CONSTRUCCIÓN Y PROGRAMACIÓN DE ROBOTS



USERS

TÍTULO: Robótica avanzada
AUTOR: Nicolás Arrioja Landa Cosio
COLECCIÓN: Manuales USERS
FORMATO: 17 x 24 cm
PÁGINAS: 352

 Impreso en Argentina. Libro de edición argentina. Primera impresión realizada en Sevagraf, Costa Rica 5226, Grand Bourg, Malvinas Argentinas, Pcia. de Buenos Aires en enero de MMX.

ISBN 978-987-663-020-7

Landa Cosio, Nicolás Arrioja
Robótica avanzada. - 1a ed. - Banfield - Lomas de Zamora:
Gradi, 2010.
v. 179, 352 p. ; 24x17 cm. - (Manual users)

ISBN 978-987-663-020-7

1. Informática. I. Título
CDD 005.3

LÉALO ANTES GRATIS

EN NUESTRO SITIO PUEDE OBTENER, DE FORMA GRATUITA, UN CAPÍTULO DE CADA UNO DE LOS LIBROS

RedUSERS
COMUNIDAD DE TECNOLOGIA

redusers.com

Nuestros libros incluyen guías visuales, explicaciones paso a paso, recuadros complementarios, ejercicios, glosarios, atajos de teclado y todos los elementos necesarios para asegurar un aprendizaje exitoso y estar conectado con el mundo de la tecnología.

LLEGAMOS A TODO EL MUNDO VÍA OCA* Y DHL**

* SÓLO VÁLIDO EN LA REPÚBLICA ARGENTINA // ** VÁLIDO EN TODO EL MUNDO EXCEPTO ARGENTINA

usershop.redusers.com // usershop@redusers.com

Nicolás Arrioja Landa Cosio

Catedrático de **IUNIVERTECH** y de la Universidad Madero, elegido el mejor profesor de los ciclos 2006-2007 y 2007-2008, se dedica a la investigación, consulta y capacitación en áreas relacionadas con la realidad virtual, la visualización científica y los videojuegos. En 1997, desarrolló el primer videojuego de realidad virtual inmersiva en Latinoamérica, conocido como **VRaptor**. También, desarrolló el primer lenguaje de programación para realidad virtual en Latinoamérica **CND- VR**, que es usado en el medio académico para enseñar, de forma sencilla, los conceptos de programación de gráficas 3D. Ha sido catedrático en algunas de las universidades más importantes de México durante más de diez años. En esas instituciones, enseñó desde las bases de programación en lenguaje C hasta Inteligencia Artificial. Ha otorgado más de 25 conferencias relacionadas con el desarrollo de la realidad virtual y los videojuegos en diversas universidades de México. Tiene una patente referida a interfaces cinemáticas para videojuegos. Diseñó el plan y el programa de estudios para la Maestría en Realidad Virtual y Video Juegos en IUNIVERTECH. Es autor de los libros **DirectX, Inteligencia Artificial** y **Curso de programación C#**, de esta misma editorial. En la actualidad, realiza una investigación sobre Inteligencia Artificial no-Algorítmica.

Dedicatoria

A mi tío Luis Gerardo Ignacio Arrioja Landa y del Villar.

Agradecimientos

A todas las personas que han hecho posible este libro. A Chester, de Phidgets Inc en Calgary. Al equipo de la editorial: Diego Spaciuk, Nicolás Kestelboim y todos aquellos que colaboraron. A María Eugenia Pérez Duarte por su apoyo y su amistad, a mis alumnos y compañeros de trabajo. Al equipo de Audio-Animatronics, de *Walt Disney Imagineering*, por la inspiración cuando era niño.

PRÓLOGO

Desde su creación en 1972, por Kenneth L. Thompson, Brian Kernighan y Dennis M. Ritchie, el lenguaje de programación C ayudó de manera potencial al desarrollo de sistemas operativos. Con el devenir del tiempo, C se convirtió en el lenguaje más popular para el diseño de software de sistemas. Años más tarde, surge el lenguaje C# (*C Sharp* en inglés) como un lenguaje más sencillo que, en la actualidad, se ha convertido en el más utilizado, potente, flexible y robusto. Como se sabe, este lenguaje ha tomado lo mejor de las características de los lenguajes preexistentes, como Visual Basic, Java y C++.

El presente libro del Dr. Arrioja Landa Cosio se une al libro del mismo autor, publicado con anterioridad por esta casa editorial, denominado *Curso de Programación C#*, el cual ha sido ampliamente aceptado por los usuarios de este lenguaje de programación. Son bien conocidas, en el ambiente latinoamericano, las habilidades didáctico-pedagógicas del autor en la escritura de libros. El presente texto cumple esta función al trabajar por proyectos, permitiendo de esta forma, al lector adolescente o joven, programar sus propios robots desde su PC a través de un lenguaje conocido o fácil de aprender como lo es C#. Uno de los sueños más desafiantes de las ciencias computacionales, en palabras del autor, es "crear dispositivos que interactúen con el mundo real y sean controlados por la lógica de nuestros programas". A través de este libro, miles de personas —expertos o principiantes— podrán acceder a la programación de robots, desde sus PCs, de manera rápida y fácil. En resumen, este texto le permitirá construir su robot de manera ágil y sencilla, ingresando de esta forma en el mundo de la robótica y de la inteligencia artificial.

Es un gusto, como Director Académico de la Universidad Madero de Puebla, México, presentar a la comunidad hispanohablante el cuarto libro de nuestro profesor, destacando su talento y su autoridad para escribir libros sobre lenguajes de programación. En nuestra Universidad, el Dr. Arrioja Landa Cosio ha sido reconocido, en dos ocasiones consecutivas, como el mejor docente de licenciatura. En este sentido, no dudamos de que el presente trabajo contribuirá a la labor y al compromiso educativos que tenemos las universidades latinoamericanas y españolas, y asimismo a la aplicación del lenguaje C# en la creación de robots.

M.C. Sergio H. Díaz Martínez
Director Académico
Universidad Madero (UMAD)
Puebla, México

EL LIBRO DE UN VISTAZO

Este libro nos introduce a la creación y programación de robots mediante el lenguaje C#, a través de diferentes proyectos sencillos de realizar. Veamos una breve descripción de los temas que abordamos en cada uno de los capítulos.

Capítulo 1

LA INTERFAZ DEL PROYECTO

En este capítulo conoceremos cómo utilizar la PC para poder controlar robots. Aprenderemos sobre el puerto USB y los diferentes tipos de Phidgets que nos permiten controlar distintos dispositivos y leer sensores.

Capítulo 2

LA PC EN CONEXIÓN CON EL MUNDO EXTERIOR

Usando los Phidgets y el lenguaje de programación C#, procederemos a controlar dispositivos de salida y a leer valores de sensores. Crearemos una aplicación base que nos servirá para el resto de los proyectos del libro.

Capítulo 3

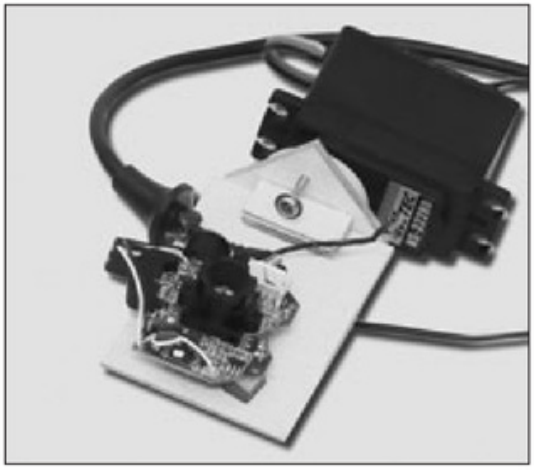
LAS HERRAMIENTAS DEL PROYECTO

Conoceremos los distintos componentes que necesitamos para crear los robots. Aprenderemos sobre los distintos tipos de baterías. Además, veremos la forma en la que funcionan los motores eléctricos y los sistemas de engranes. Por último, programaremos una aplicación que controle a los servos.

Capítulo 4

ROBOT CON VISIÓN POR COMPUTADORA

En este capítulo construiremos un sistema robótico que tiene visión por computadora. Por medio de los servos, podremos modificar la dirección hacia donde ve la cámara web y será capaz de seguir a los distintos objetos que entran en su campo de visión.



Capítulo 5

DEDO ROBÓTICO Y GUANTE VR

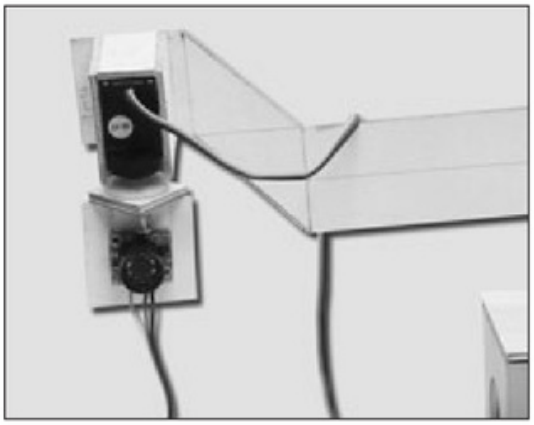
Crearemos un dedo robótico que es capaz de flexionarse usando un servo. El dedo será controlado por medio de un guante del mismo estilo que los usados en realidad virtual, el cual, por medio de un sensor, mandará la información a la aplicación.

Capítulo 6

SONAR ROBÓTICO

Usando un sistema de sonar, podemos detectar la distancia hasta un objeto; al mover el sonar con un robot, podemos obtener en

nuestra computadora las distancias a los diferentes objetos que se encuentran a su alrededor. Incluso, es posible hacer muestreos estereoscópicos con este sistema.



Capítulo 7

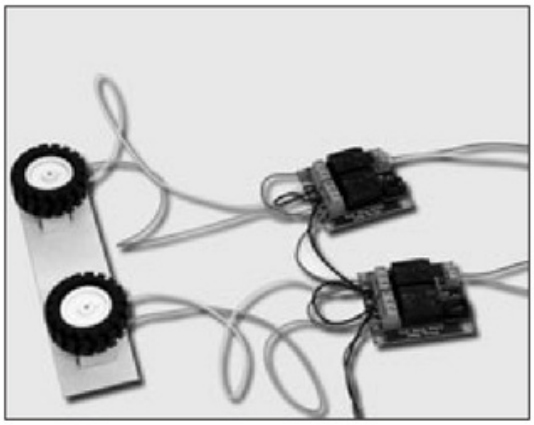
ROBOT ANIMATRÓNICO

En este capítulo comenzaremos un proyecto divertido para crear nuestro propio robot animatrónico: un rostro que representará la animación que nosotros programamos. Podremos dar movimiento a todo el rostro del robot desde nuestro equipo. Tanto el ojo, como la ceja y la boca estarán bajo nuestro control.

Capítulo 8

PLATAFORMA ROBÓTICA

Construiremos una plataforma robótica para nuestra computadora laptop. Esto nos permite tener robots autónomos que no necesitan estar fijos a un lugar en particular. También, aprenderemos cómo controlar motores eléctricos usando la computadora.



Servicios al lector

En este apartado encontraremos un listado de sitios útiles para mantenernos informados y un índice que nos ayudará a encontrar los términos más importantes de esta obra.

! INFORMACIÓN COMPLEMENTARIA

A lo largo de este manual, encontrará una serie de recuadros que le brindarán información complementaria: curiosidades, trucos, ideas y consejos sobre los temas tratados.
Cada recuadro está identificado con uno de los siguientes iconos:

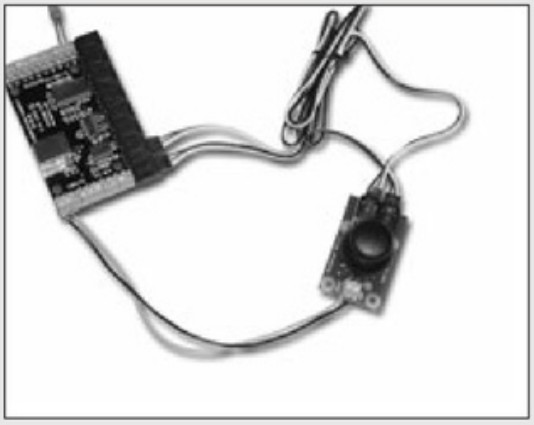
{} CURIOSIDADES E IDEAS

* ATENCIÓN

III DATOS ÚTILES Y NOVEDADES

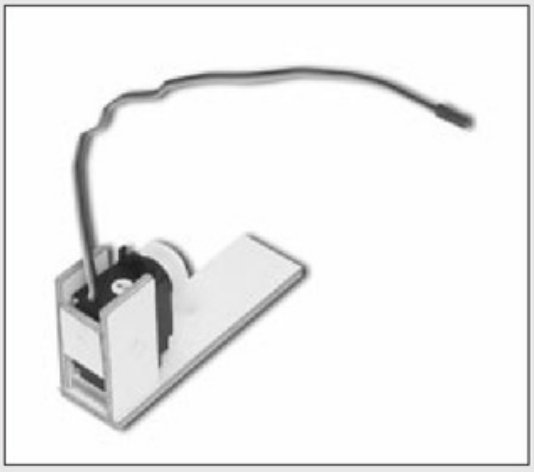
▶ SITIOS WEB

CONTENIDO



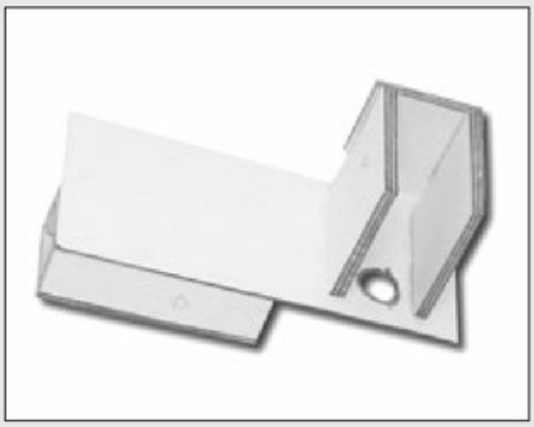
Capítulo 4

ROBOT CON VISIÓN POR COMPUTADORA



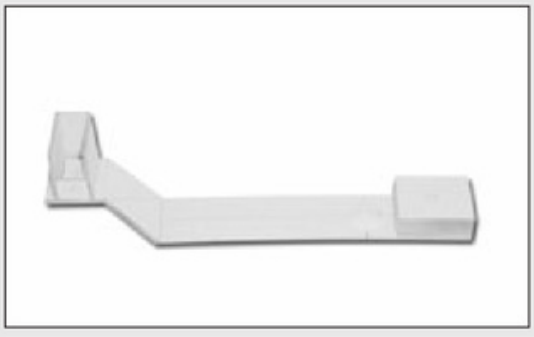
Capítulo 5

DEDO ROBÓTICO Y GUANTE VR

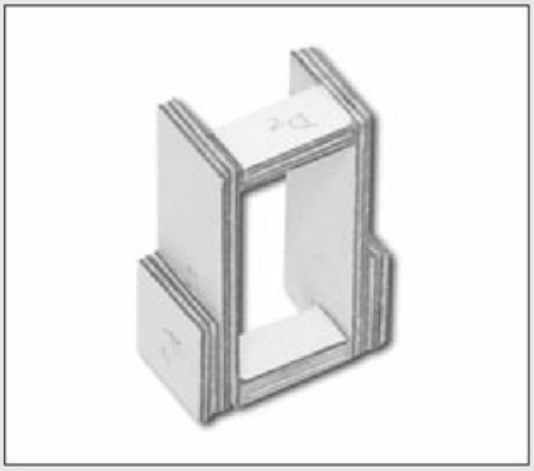


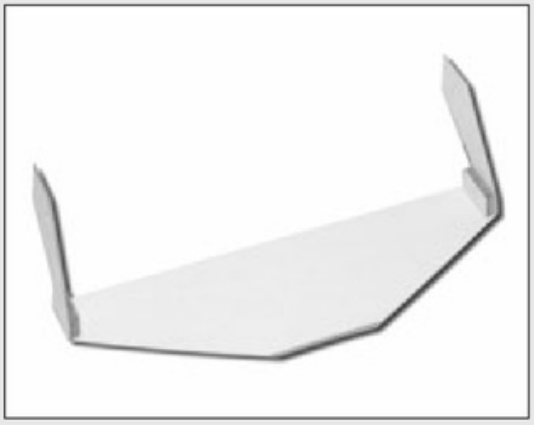
Capítulo 6

SONAR ROBÓTICO



Capítulo 7





Capítulo 8

Servicios al lector

INTRODUCCIÓN

Cuando se presentó la idea de hacer un libro que tocara el tema de los robots, quisimos que, también, fuera un desafío. En el mercado, existen muchos libros que hablan sobre los robots. La gran mayoría de ellos tocan los mismos temas y, al final, terminan con el proyecto del clásico robot que sigue la línea trazada en el piso.

La idea original era llevar al lector hacia robots que fueran más interesantes, y que, al mismo tiempo, pudiera complementar los proyectos con lo aprendido en el libro Inteligencia Artificial, publicado en esta misma editorial. Sin embargo, hacer inteligencia artificial en un microcontrolador PIC, es complicado, salvo que seamos expertos en programación de PICs. Por eso, la mejor opción fue programar los robots en un lenguaje que resultara sencillo y fácil de aprender, como C#, usando una computadora personal. Esto nos brinda una gran flexibilidad y le permite entrar en el mundo de la robótica y de la animatrónica a una gran cantidad de personas que, de otra forma, no lo hubieran logrado. Es posible pensar que esta facilidad de programación viene con el costo de la autonomía de los robots que hagamos, pero, al final del libro, mostramos cómo realizar una plataforma móvil para computadoras Laptop, con lo que recuperamos la autonomía del robot con la flexibilidad y el poder de cómputo de este tipo de equipos.

La entrada y salida de señales se ve simplificada enormemente al usar los Phidgets. Estos pequeños circuitos nos permiten interactuar con facilidad con el mundo exterior, ya sea por medio de sensores, o mandando señales eléctricas para controlar dispositivos. A diferencia de otros proyectos que usan el puerto paralelo o serial de las computadoras, que ya se encuentran casi en extinción, el Phidget tiene la ventaja de usar el puerto USB.

Esperamos que el presente libro sirva de estímulo para una nueva generación de ingenieros en robótica y animatrónica, y que permita entrar en el mundo de los robots a todas aquellas personas que antes lo pensaban como algo imposible.

La interfaz del proyecto

Una de las actividades más interesantes relacionadas con las ciencias computacionales es la inteligencia artificial y la robótica. Poder crear dispositivos que interactúen con el mundo real y que sean controlados por la lógica de nuestros programas resulta uno de los desafíos que soñamos con enfrentar quienes nos interesamos en las tecnologías.

LA PC PARA CONTROLAR ROBOTS

Durante muchos años, si deseábamos comunicar la computadora con el exterior, teníamos tres opciones: usar el **puerto serial**, usar el **puerto paralelo** o construir nuestra propia **tarjeta de interfaz casera**.
El puerto paralelo era sencillo de utilizar, pero existían limitaciones respecto de la cantidad de **bits** que se podían usar para la salida y para la entrada de datos. El puerto serial permitía transferir más información, pero, por el hecho de ser serial, se complicaba el circuito necesario para la comunicación con el robot. Hacer nuestra propia tarjeta de interfaz permitía tener la cantidad necesaria de bits de entrada o de salida, e, incluso, leer valores **analógicos**, pero se necesitaban conocimientos avanzados de electrónica para lograrlo. En la actualidad, la mayor parte de las nuevas computadoras, en especial las laptops, ya no presentan puertos seriales ni paralelos. Es necesario que utilicemos nuevas opciones para nuestros proyectos de conectividad con el mundo real. La alternativa que podemos utilizar es el **puerto USB**. Si deseamos crear un robot autónomo, podemos hacerlo mediante el uso de un **microcontrolador**. Muchas veces, se utiliza el microcontrolador **PIC**. Lamentablemente, no todas las personas poseen los conocimientos necesarios para realizar esta programación en forma directa. En algunas ocasiones, la cantidad de lógica necesaria para la aplicación robótica puede exceder las capacidades del microcontrolador.

El uso de la PC para controlar el robot nos brinda varias ventajas. La primera de ellas es que contamos con un significativo poder de cómputo. Con una PC, es posible programar lógicas mucho más avanzadas e introducir, dentro de nuestros programas, los algoritmos de Inteligencia Artificial necesarios para que el robot se comporte de manera más inteligente. Además, la gran capacidad de memoria y almacenamiento de las PC nos permite, incluso, que el robot sea capaz de generar sus propios mapas y de modificarlos según cambien las condiciones de su entorno.
Otra ventaja de usar la PC consiste en la posibilidad de programar el robot en un lenguaje conocido y fácil de aprender, como **C#**, un lenguaje más sencillo de programar que el lenguaje ensamblador del PIC. Esto permite que miles de programadores puedan acceder a la programación de robots de manera muy

INFORMACIÓN SOBRE LOS PIC

Si deseamos conocer más sobre el PIC, podemos visitar el siguiente sitio web: **www.piclist.com/techref/piclist/index.htm**, que contiene enlaces a personas que están interesadas en el desarrollo de proyectos con este microcontrolador. La ventaja que nos otorga este microcontrolador es la independencia del robot respecto de la computadora personal.

simple y rápida. Contamos además con buenas herramientas de depuración para el programa. Con la PC, también es posible llevar a cabo simulaciones, o probar la aplicación, aunque el hardware todavía no se encuentre listo. El aumento en el poder de cómputo, y la disminución de peso de las laptops nos permitiría incorporar el robot dentro de una computadora. De esta manera, ahora es posible que cualquier persona con conocimientos de programación pueda acceder al desarrollo y a la creación de sus propios robots.

El puerto USB

Los proyectos que desarrollaremos en este libro serán controlados por medio del puerto USB. Dicho puerto nos facilitará mucho la utilización de los robots. Las siglas USB, en inglés, significan **Universal Serial Bus**, y en su traducción al español, **Bus Universal en serie**. Este bus fue diseñado para reemplazar a los puertos seriales y paralelos. Una de las razones para esto fue el incremento en la velocidad de transferencia de datos. También, se buscó tener una interfaz estándar que permitiera conectar diferentes dispositivos. Con el puerto USB, podemos conectar cámaras, impresoras, escáneres, controladores para juegos, etcétera. El puerto USB, además, nos permite llevar a cabo **hot swapping**, esto significa que podemos cambiar el dispositivo conectado sin tener que apagar la computadora. El uso de dispositivos **plug ´n´play** es otra característica que provee USB y que no estaba disponible en los puertos anteriores.

El origen y las versiones del USB

Al momento de la publicación de este libro, existen tres especificaciones de USB, cada una de ellas superior a la anterior. La primera especificación conocida como **USB 1.0** fue el resultado del trabajo en conjunto de muchas empresas. Éstas se unieron para crear no sólo las especificaciones, sino también el hardware y el software necesarios para que otros proveedores pudieran implementarlos en sus productos. Las empresas que colaboraron en este proyecto son: **Intel**, **Microsoft**, **IBM**, **Compaq**, **Digital** y **Northern Telecom**. Para enero de 1996, las especifica-

DESCARGA DE C# EXPRESS EDITION

Podemos trabajar con la versión profesional de **Visual Studio**, producido por Microsoft o, si lo deseamos, es posible descargar la versión Express de C#. Esta versión es gratuita y contiene todas las herramientas que necesitamos para este libro. Para descargarla, debemos dirigirnos a **www.microsoft.com/express/vcsharp/**.

ciones ya estaban listas. Esta primera versión permite dos velocidades de transferencia. La velocidad **baja** o **Low-Speed** es de 1.5 Mbits/s, y la velocidad **alta**, también conocida como **Full-Speed**, permite 12 Mbits/s.

Figura 1. *En esta figura, podemos observar el logotipo utilizado en la versión 1.0 del USB.*

Durante el año 2000, en el mes de abril, aparece la especificación **2.0** del USB. Esta especificación permite transferencias de información a velocidades mucho mayores que la primera versión. Con dicha especificación, se puede transferir hasta un máximo de 480 Mbit/s. Esta velocidad se conoce como **Hi-Speed** o **velocidad alta**.

Figura 2. *Aquí observamos el logotipo utilizado para indicar que el dispositivo puede utilizar el Hi-Speed en un puerto USB.*

III SOBRE LA ESPECIFICACIÓN USB 1.1

En septiembre de 1998, se liberó una nueva especificación conocida como USB 1.1. Ésta se realizó para poder corregir una serie de problemas que tenía la versión 1.0. Muchas de las primeras computadoras con puertos USB usaban esta especificación, al igual que los dispositivos de esa época.

La más reciente versión de USB existente es la **3.0**. Ésta fue lanzada el 17 de noviembre de 2008, y se espera que, en un futuro próximo, empiecen a aparecer dispositivos que utilicen esta nueva adaptación. La característica más importante de esta versión es el incremento en la velocidad. Esta versión alcanza una increíble velocidad de transferencia de hasta 5.0 Gbit/s, y se la conoce como **SuperSpeed** o **supervelocidad,** en idioma castellano.

Figura 3. *En esta figura, podemos observar el logotipo para la versión 3 del USB con SuperSpeed.*

Funcionamiento del USB

Ahora que ya conocemos las bases, podemos empezar a aprender cómo funciona el USB. Para ello, debemos imaginarnos una **topología de estrella**. En ésta, encontramos un elemento conocido como **host**. Además, los dispositivos que utilizaremos tienen que ser conectados por medio de los **hubs**. Entre los hubs existentes, encontraremos uno especial denominado **root hub** o **hub raíz**, que se ubica, justamente, en el interior del host.

Conectamos el dispositivo al hub que puede ser, por ejemplo, una cámara digital o un controlador de juegos. Cada dispositivo puede tener uno o varios elementos

III MODIFICACIONES EN USB 2.0

A partir de octubre de 2000 y hasta septiembre de 2007, han surgido correcciones, pero, en especial, adiciones al USB 2.0. Cada uno de estos cambios se lleva a cabo por medio de las **ECN** o **Engineering Change Notices**. Por lo general, estos cambios ayudan a tener versiones todavía más estables y con una mayor eficiencia.

lógicos, conocidos como **funciones del dispositivo**. Gracias a este tipo de elementos un mismo dispositivo realice varias funciones. Por ejemplo, en un dispositivo multifuncional, encontramos una función de dispositivo para escanear y una función de dispositivo para imprimir.

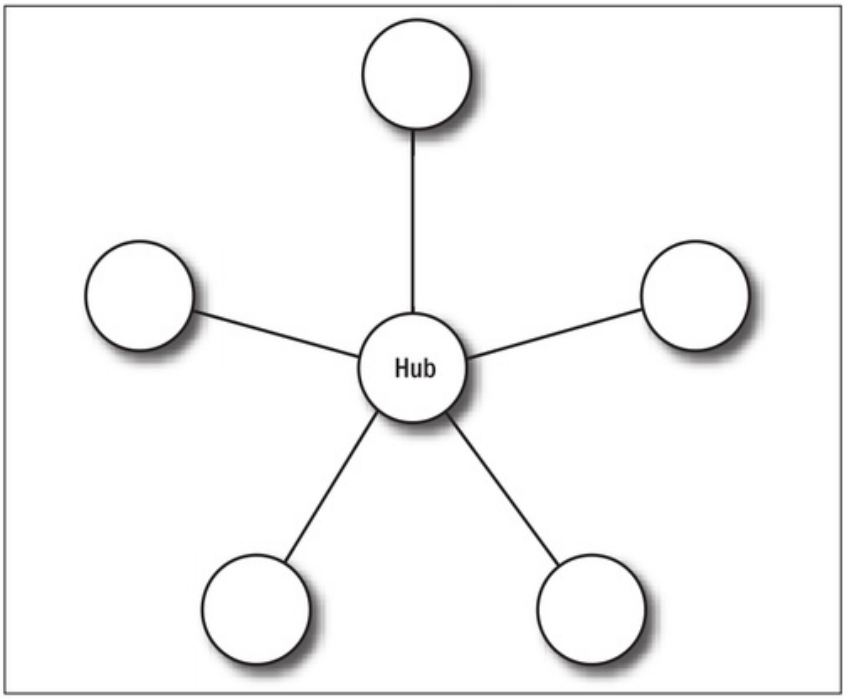


Figura 4. *Esta figura muestra una topología de estrella típica. El nodo central se conoce como hub.*

La comunicación entre el host y el dispositivo se lleva a cabo por medio de lo que se conoce como **pipe**, que es, simplemente, un canal lógico. En el dispositivo, los pipes llegan hasta otra entidad lógica que se conoce como **endpoint** o **punto final**. Los endpoints son importantes porque pueden transferir datos, pero sólo pueden hacerlo en una única dirección, conocida como **unidireccional**. La dirección puede ser hacia el dispositivo o desde el dispositivo. Cada uno de ellos puede tener un máximo de 32 pipes, 16 de los pipes son desde el host y 16 de ellos hacia el dispositivo. El encargado de controlar la transferencia de información es el host, pero, cuando se usa SuperSpeed, los dispositivos pueden pedir servicios al host también.

III LA TOPOLOGÍA DE ESTRELLA

Esta topología tiene un núcleo central conocido como hub, que se encarga de transferir los mensajes. Todos los demás dispositivos se conectan a él. Si pensamos el hub visualmente, veremos que se encuentra en el centro y los demás dispositivos se distribuyen de forma radial. Entonces su forma nos recuerda la estructura de una estrella.

Cuando conectamos un dispositivo al USB, lo primero que ocurre es un proceso conocido como **enumeración**. En la enumeración, se envía un **reset** al dispositivo y se reconoce la velocidad en la que éste puede operar. El dispositivo envía su información al host y, de esta forma, el host puede conocer qué tipo de dispositivo se ha conectado. Como puede haber muchos dispositivos conectados, es necesario tener una forma de poder reconocerlos; para esto, se les asigna una **dirección**. La dirección es única para cada dispositivo y consiste en 7 bits. El host carga los **drivers** necesarios para poder usar el dispositivo y llevar a cabo la comunicación, que se realiza por medio de **packets**. Existen diferentes tipos de packets: **handshake**, **token**, **data**, etcétera. Los packets son enviados por el host hacia un dispositivo. Para llegar, tienen que pasar por el root hub. Si existieran más hubs entre el root hub y el dispositivo, también deberán pasar por ellos. El dispositivo, además, puede contestar enviando packets de regreso al host.

Conectores USB

La información es enviada por medio de impulsos eléctricos que se transmiten por cables. En el extremo de los cables, encontraremos diferentes tipos de conectores. El USB define las diferencias entre éstos. En su origen, existían exclusivamente el conector de **Tipo A** y el conector de **Tipo B**. A medida que el USB fue evolucionando, se han agregado nuevos tipos de conectores. Es posible que algunos de los dispositivos USB que tenemos en casa o en la oficina muestren uno de estos tipos de conectores. El conector de Tipo A, también conocido como **USB-A**, es usado, en especial, en el host o en los hubs. Su forma recuerda a un rectángulo, y es el más común de los conectores. Lo encontramos con frecuencia en la computadora. El conector Tipo B o **USB-B** tiene forma cuadrada con dos esquinas cortadas. Este tipo de conectores se utilizaron en forma masiva en las impresoras.

Con el tiempo, muchos dispositivos empezaron a incorporar el USB, pero el conector tipo B era demasiado grande para, por ejemplo, las PDA y las cámaras digitales. Entonces, surgieron otros tipos de conectores. Entre ellos se encuentran: **Mini-A**, **Mini-B**, **Micro-AB** y **Micro-B**.

III LA RAZÓN DE A Y B

Una de las razones por las que se diseñaron un conector A y un conector B en el USB es para evitar que, de manera accidental, se creen circuitos como rizos, es decir un hub que se conecte a sí mismo. Al tener dos conectores diferentes, resulta más fácil identificar cómo se encuentran conectados entre sí; de esta forma, se evitan errores de conexión.

Figura 5. *Esta fotografía nos muestra diferentes tipos de conectores USB. Es posible conseguir* ***coples*** *que permitan convertir entre un tipo y otro.*

El microcontrolador Phidget

Nosotros recurriremos al USB para controlar nuestros proyectos robóticos, por lo que es necesario encontrar una forma sencilla de enviar información a través del puerto y recibir datos del exterior. El protocolo de comunicación del USB es complejo, y necesitaríamos crear nuestro propio hardware y software, lo cual insumiría mucho tiempo y no estaría al alcance de los conocimientos técnicos de todas las personas. Por esta razón, resulta oportuno buscar una forma sencilla de hacerlo. La solución más práctica consiste en recurrir a los **Phidgets**. Éstos son una serie de microcontroladores que se conectan a la computadora por medio del puerto USB. Estos microcontroladores pueden recibir y enviar información desde la computadora. Además, es posible programarlos en muchos lenguajes y utilizan librerías o bibliotecas que proveen funciones listas para su uso. El costo de los Phidgets es razonable si tomamos en cuenta todas las ventajas que nos reditúan.

LAS ESPECIFICACIONES DEL USB

Si queremos encontrar las especificaciones completas del USB, podemos visitar el siguiente sitio de Internet: **www.usb.org**. Conocer las especificaciones es importante si deseamos diseñar un dispositivo que se comunique con nuestra computadora personal por medio de este puerto.

La empresa Phidgets Inc. no sólo provee kits de interfaz, sino que, además, ofrece una completa serie de sensores, controladores de motor, RFID y demás accesorios. Los proyectos del libro se pueden desarrollar con cualquier tipo de microcontrolador, pero los programas de ejemplo y el hardware estarán basados en Phidgets. Ahora, debemos conocer el equipo con el que trabajaremos en los distintos capítulos del libro.

El kit de interfaz 8/8/8

Uno de nuestros elementos principales del microcontrolador Phidget es el **kit de interfaz 8/8/8**, que nos permitirá conectar la computadora con el exterior y, al mismo tiempo, recibir información desde él.

***Figura 6.** Esta figura nos muestra los contenidos del kit 8/8/8. En él se incluye el microcontrolador, el cable USB, hardware para montar y un instructivo.*

Este kit de interfaz nos proveerá de ocho **entradas analógicas**, ocho **entradas digitales** y ocho **salidas digitales**. También, cuenta con un puerto USB para poder conectarlo en forma directa a la computadora.

III DÓNDE CONSEGUIR LOS PHIDGETS

Los Phidgets se pueden comprar en línea y son enviados a todos los países por medio de UPS. También, es posible conseguirlos con diferentes distribuidores en diversas partes del mundo. El sitio web de la empresa es **www.phidgets.com**; aquí encontramos la tienda y los datos técnicos necesarios sobre todos los productos de la compañía.

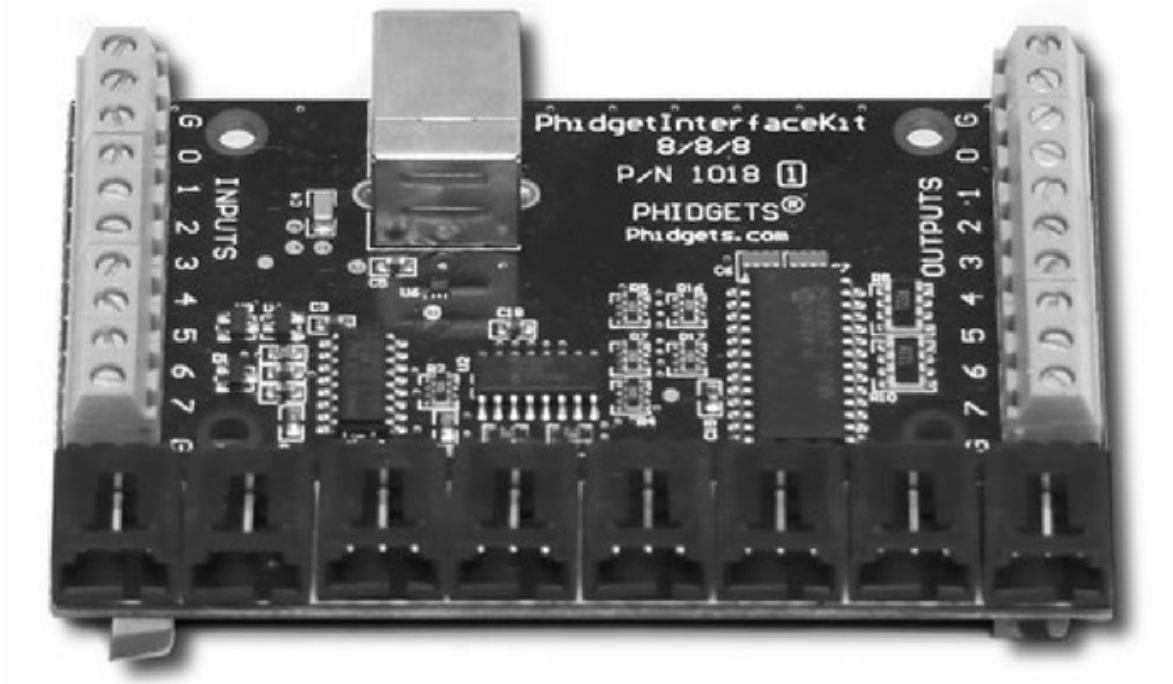


Figura 7. *En esta figura, observamos el microcontrolador. Los diferentes conectores corresponden a las entradas y a las salidas.*

Las entradas analógicas nos permiten realizar mediciones, en especial de sensores. Es posible que diseñemos nuestros propios sensores y que los conectemos. Cada una de las entradas provee **+5 VDC** para alimentar al sensor; también, una conexión a tierra y un retorno del sensor con el voltaje análogo a la medición realizada. La máxima corriente que podemos utilizar en una entrada analógica es de **400 mA**.

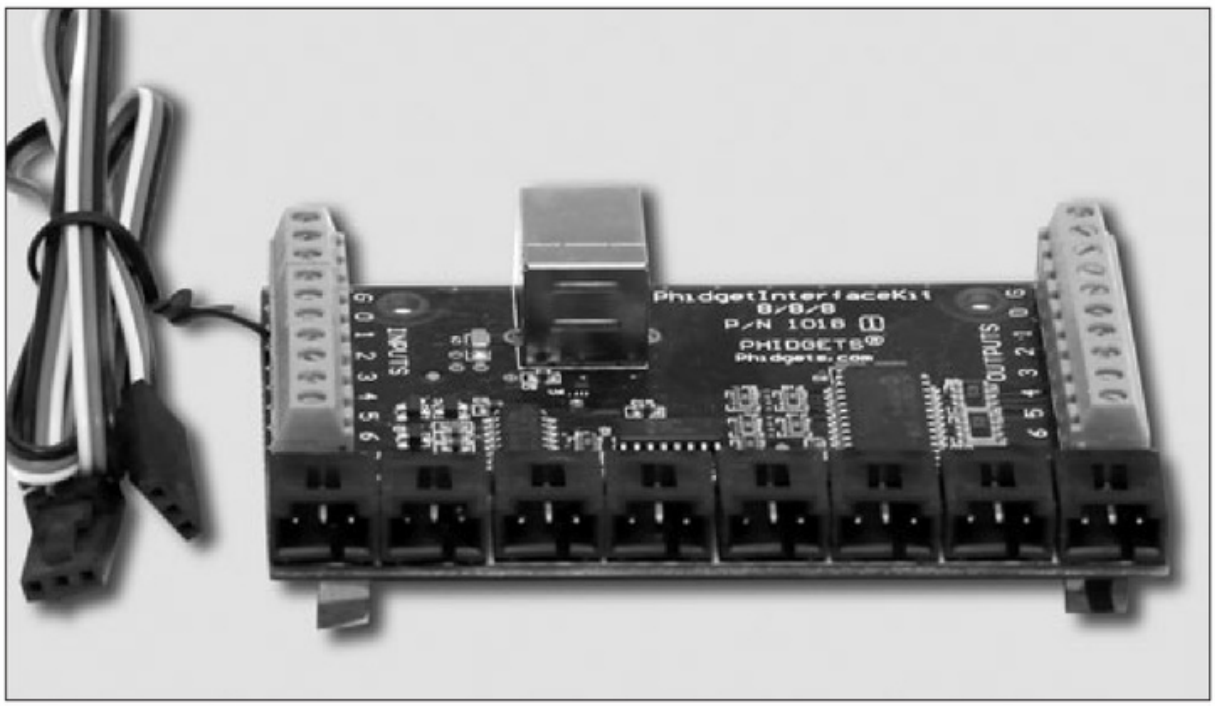


Figura 8. *En esta figura, podemos ver las entradas analógicas y un conector analógico. Estas entradas tienen forma cuadrada y tres pines cada una.*

Para su enganche, la entrada analógica usa un conector de tres pines. El cable de color negro es la tierra, el rojo corresponde al voltaje de alimentación de **+5VD**, y el blanco es el que lleva el voltaje regresado por el sensor. El puerto USB se encuentra en la parte superior del Phidget. Un extremo del cable se conectará a este puerto, y el otro, en la computadora.

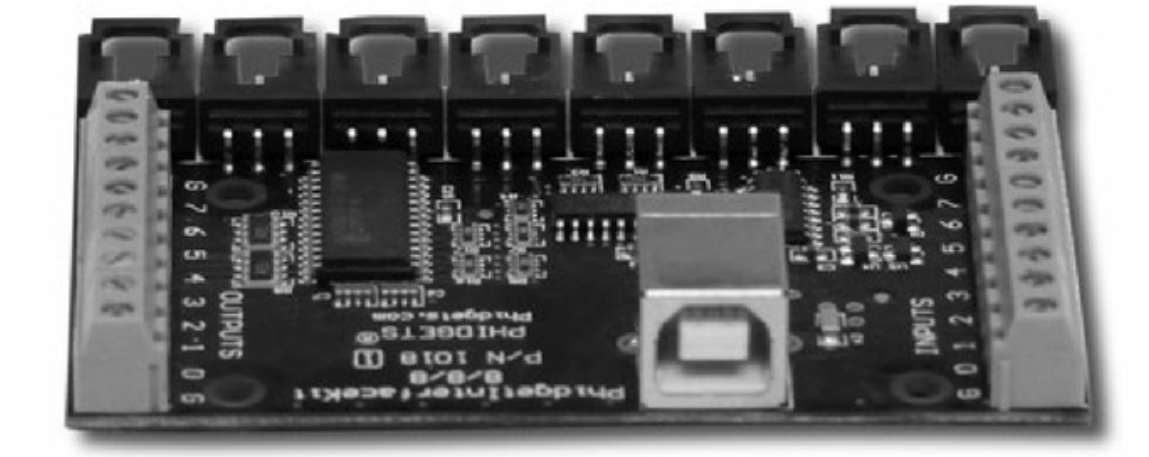


Figura 9. *Éste es el puerto USB que usaremos para conectar el Phidget a la computadora.*

Existen, también, entradas de tipo digital. Estas entradas nos brindan valores en el esquema de **Falso** o **Verdadero**. Es decir, encontramos o no voltaje en ellas. Entonces, a diferencia de las entradas analógicas, sólo nos ofrecen dos posibles valores. Estas entradas pueden ser utilizadas con switches, lógica digital o sensores que nos den el valor en términos digitales.

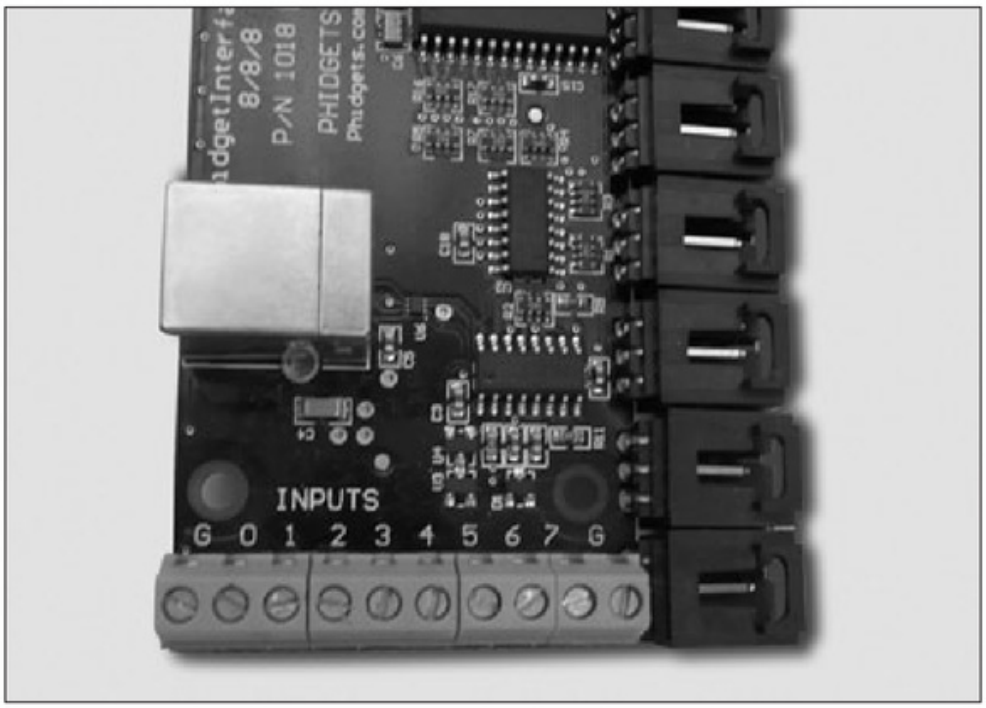


Figura 10. *Aquí podemos observar las entradas digitales que son las más cercanas al puerto USB. Como podemos ver en la figura, es necesario usar un destornillador con ellas.*

Las entradas digitales se encuentran a la izquierda del Phidget. Esto lo podemos observar en la **figura 10**. Están numeradas del 0 al 7 de izquierda a derecha. En ambos extremos, aparece un conector marcado como **G**. Este conector simboliza a la tierra. La **figura 11** nos muestra un ejemplo de cómo podemos utilizar una entrada digital. Tenemos un simple switch. Este switch, por ejemplo, podría ser un sensor de choque del robot. Vemos que una de las terminales del switch se conecta a la tierra, y la otra, a la entrada que nos interesa. Cuando la entrada no está conectada en forma directa a tierra, reportará el valor **Falso**, y, si se cierra el switch y queda conectada a tierra, obtendremos el valor **Verdadero**.

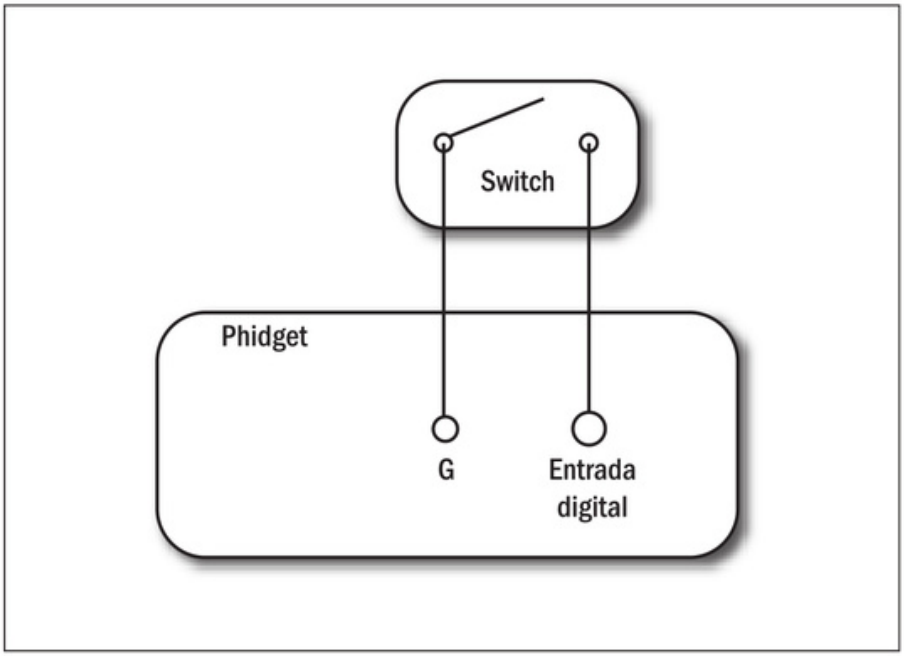


Figura 11. *Un circuito sencillo para la entrada digital. En este caso, se utiliza un switch.*

El 8/8/8 va a tener ocho salidas digitales. Estas salidas nos proveerán de un voltaje de +5VDC y se pueden utilizar para alimentar cualquier circuito que acepte señales **CMOS**, **transistores**, **LEDs** u otros componentes. Las salidas se encuentran del lado derecho del Phidget. Están numeradas del 7 al 0. La número 7 se encuentra a la izquierda, y la 0, a la derecha. En ambos extremos, tenemos conexiones para tierra.

MÚLTIPLES HUBS

Es posible conectar hubs USB a otros hubs para ampliar la cantidad de puertos disponibles. Se pueden tener hasta 127 dispositivos conectados a un host. La gran ventaja de esto es que no estamos limitados en la cantidad de Phidgets que podemos usar para nuestros proyectos robóticos.

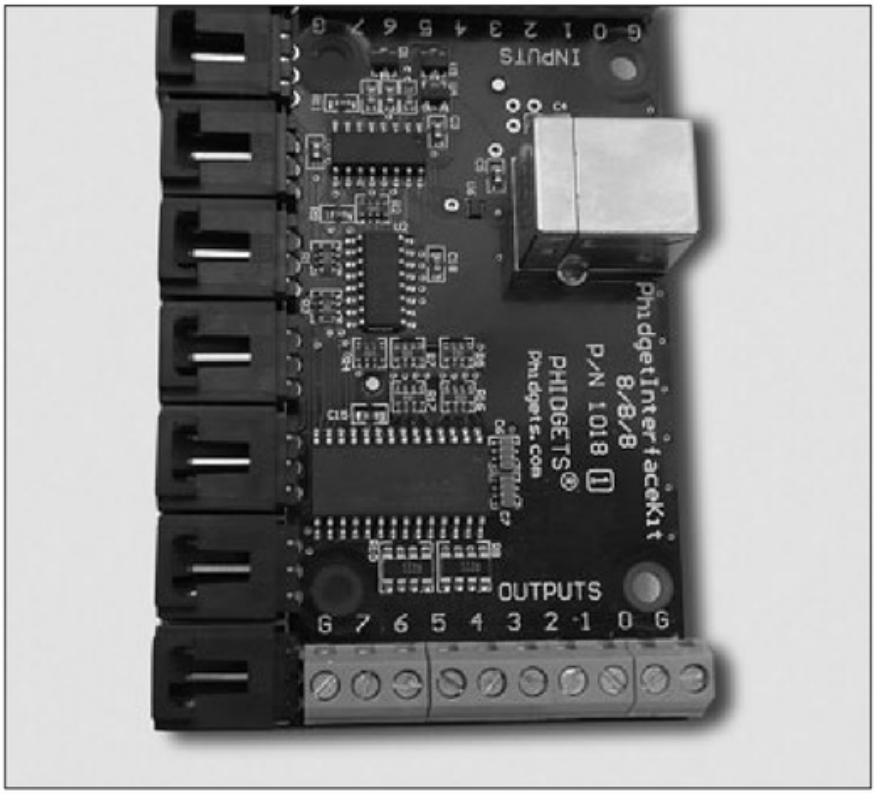


Figura 12. *Esta figura nos muestra las salidas digitales del Phidget.*

En la parte inferior del Phidget, encontramos una etiqueta que nos indica el número de serie. Este número es importante, ya que lo necesitaremos posteriormente para llevar a cabo identificaciones.

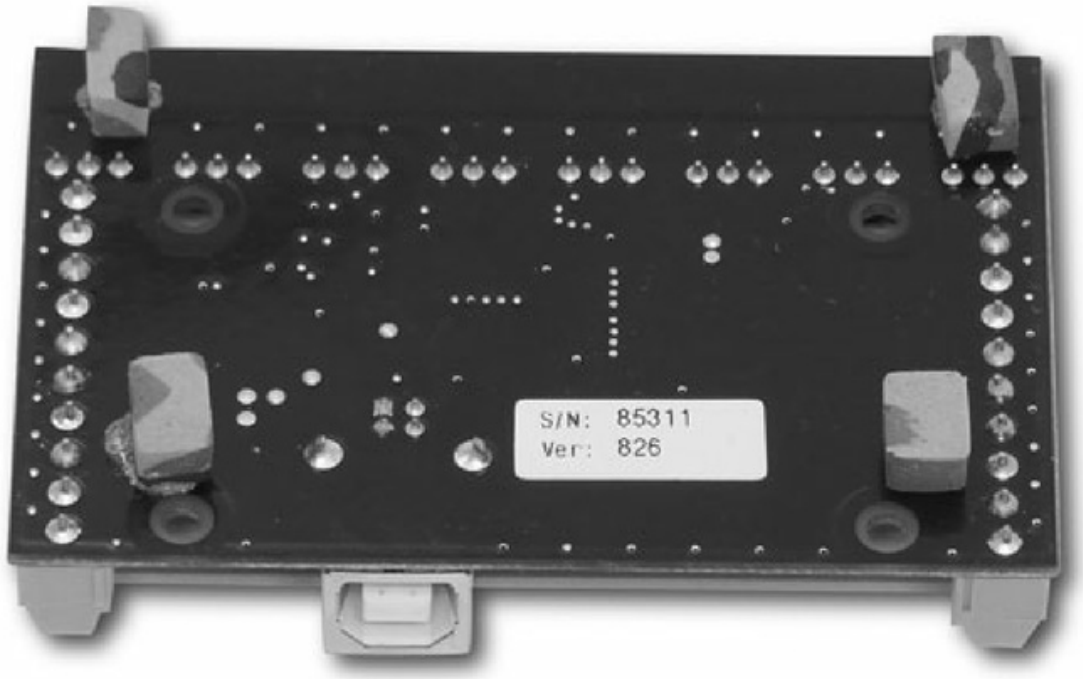


Figura 13. *El número de serie se encuentra en la parte inferior del circuito impreso del Phidget. Es fundamental no perderlo, pues lo vamos a necesitar en el programa que conecta el Phidget.*

El kit avanzado para servos cuatro-motores

Este kit es una combinación de varios productos, ya que contiene un microcontrolador que permite manejar hasta ocho **servos**, cuatro servos, cable USB para conectar a la computadora y una **fuente de poder**.

***Figura 14.** Éstos son los contenidos del kit avanzado para servos cuatro-motores.*

El microcontrolador es el modelo **PhidgetAdvancedServo 8-Motor**. Éste nos permite controlar la posición, velocidad y aceleración de hasta ocho motores servo. Tiene una resolución de 125 pasos por grado. Los servos pueden usar hasta 1.5 amperes de corriente. Este microcontrolador tiene diferentes conectores. En la parte inferior, encontramos el puerto USB; también, el conector para la fuente de poder, y dos conexiones para la tierra y el voltaje

III EL SERVO

El servo resulta de gran utilidad en aplicaciones robóticas, ya que se trata de un motor eléctrico al cual podemos configurar de forma precisa la posición o cantidad de giros. Esto nos permite crear fácilmente dispositivos donde el movimiento puede ser controlado, como ocurre con los brazos robóticos. Debemos cuidar la cantidad de carga que el servo puede usar y su rango de movimiento.

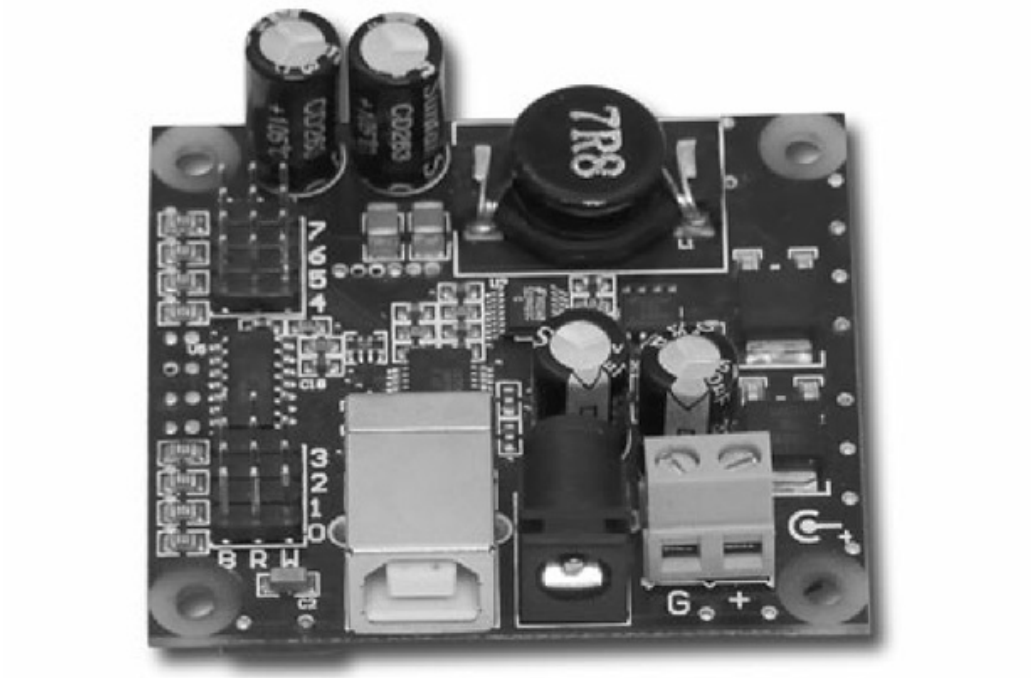


Figura 15. *Aquí podemos observar el microcontrolador y sus conexiones.*

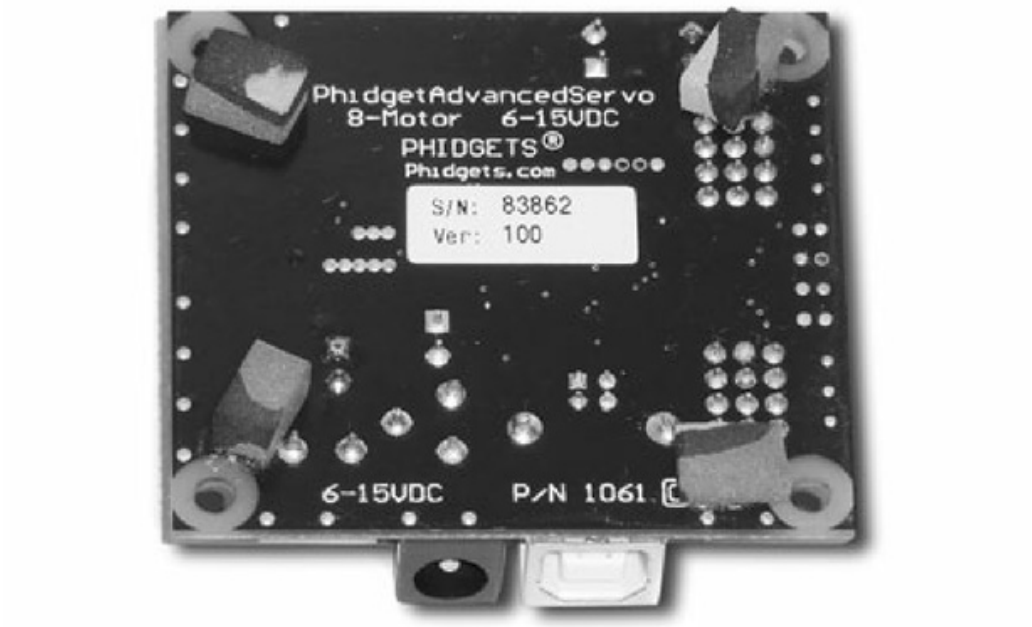


Figura 16. *En la parte inferior, encontramos el número de serie del controlador.*

III INFORMACIÓN TÉCNICA DE LOS DISPOSITIVOS

En la página de Phidgets, es posible encontrar documentos PDF con especificaciones técnicas de todos los productos. También nos ofrece documentación sobre el API y ejemplos de programación. Es recomendable leer las especificaciones técnicas antes de empezar a usar los dispositivos.

En el lado izquierdo del microcontrolador, tenemos los pines donde deben ser conectados los servos que serán manejados. Éstos se encuentran divididos en dos bloques de cuatro servos cada uno. En el bloque inferior, aparecen las conexiones para los servos del **0** al **3**. El servo **0**, en la parte baja, y el **3**, en la parte alta. El bloque superior es para los servos del **4** al **7**. El servo **4**, en la parte baja, y el **7**, en la parte alta.

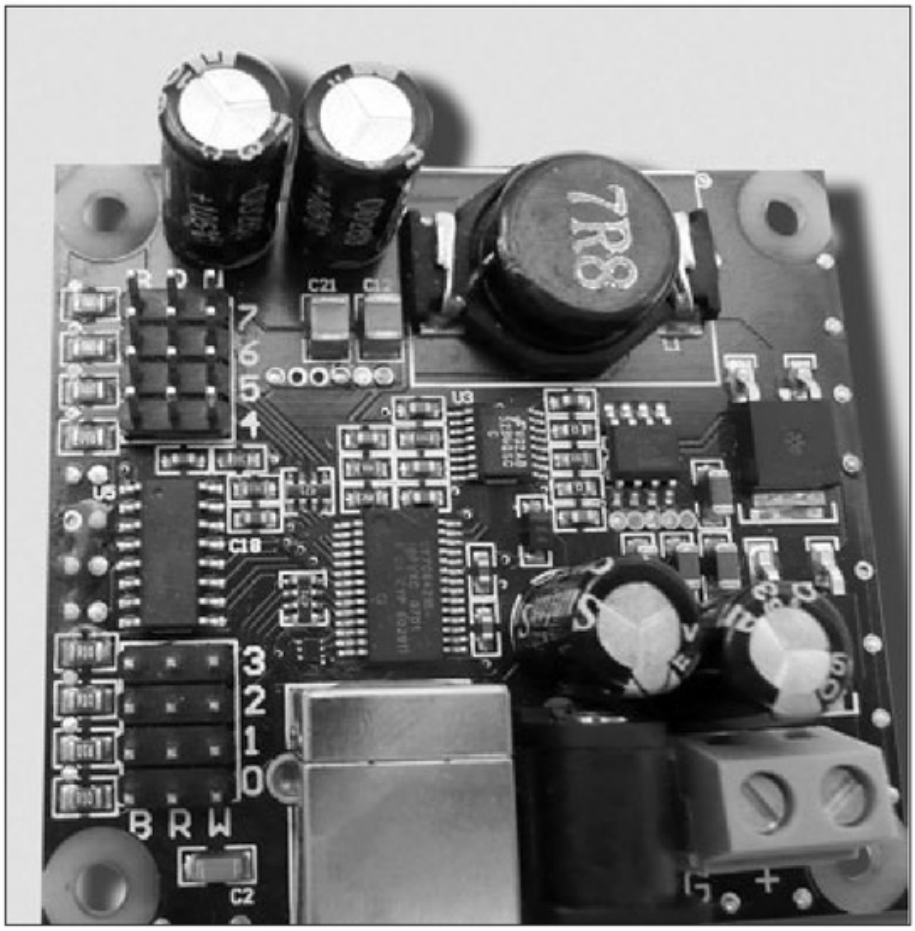


Figura 17. *En esta figura, podemos observar el lugar donde se conectarán los servos.*

Tarjeta con dos relevadores

Otro componente importante que vamos a utilizar es la tarjeta con dos **relevadores**. Con ella, controlaremos los motores eléctricos DC de nuestros proyectos. La tarjeta, desde luego, puede ser utilizada para una gran variedad de proyectos y controlar otros dispositivos, como electroimanes, solenoides e, incluso, lámparas incandescentes. Esta tarjeta contiene dos relevadores que funcionan como switches. Cada relevador es de tipo **SPDT**. La tarjeta se puede conectar con facilidad al 8/8/8 para controlar el estado de los relevadores por medio de las salidas digitales. En la parte superior de la tarjeta, encontramos los conectores para los relevadores. Existe un contacto para el común, y dos contactos que se encuentran de forma normal en estado abierto y cerrado, respectivamente. En la parte inferior, tenemos un conector de tres pines. Este conector debe dirigirse a una entrada analógica del 8/8/8 y es usado para

alimentar a la tarjeta. En la parte inferior y del lado derecho, encontramos otro conector que va a las salidas digitales del 8/8/8 y sirve para recibir la señal de control. Esta señal indica el estado en el que debe hallarse el relevador correspondiente.

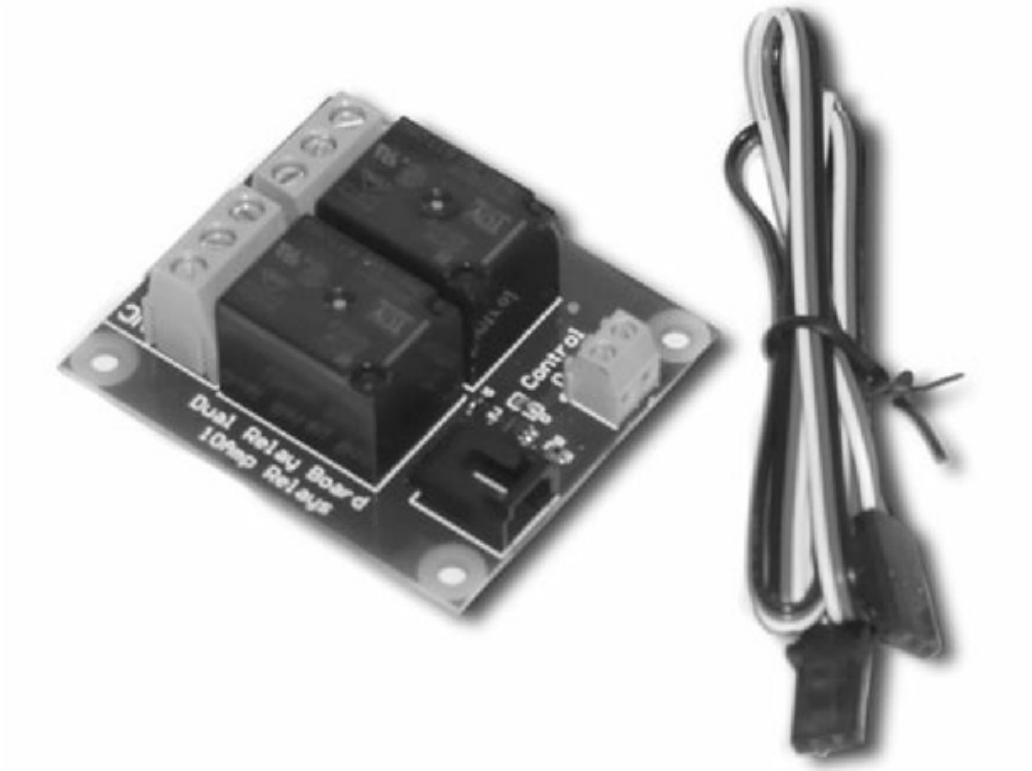


Figura 18. *En esta figura, vemos la tarjeta con dos relevadores y el cable que se utiliza para conectar su alimentación por medio de la entrada analógica del 8/8/8.*

Ahora, ya conocemos el hardware con el que trabajaremos. En el próximo capítulo, empezaremos a ocuparnos de manera específica de los programas necesarios para la comunicación con los microcontroladores.

... RESUMEN

En este capítulo establecimos que podemos utilizar la PC para controlar a nuestros robots. El puerto USB reemplaza, en la actualidad a los puertos seriales y paralelo. Luego, vimos que existen diferentes versiones del USB. Con el tiempo, aparecieron mejoras, en especial, en la velocidad de transferencia. El modelo de trabajo del USB requiere de un host y de hubs. El root hub existe dentro del host. Los dispositivos se conectan a los hubs. La comunicación se lleva a cabo por medio de pipes, y éstos llegan hasta el endpoint. El endpoint puede tener dirección de transferencia de los datos. Además, los Phidgets nos permiten conectar nuestro robot a la PC por medio del USB. También, existe un kit para poder controlar los servos. Se puede utilizar una tarjeta con relevadores para controlar dispositivos, como los motores.

ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1. ¿Qué puertos se pueden usar para conectar la PC a un robot?

2. ¿Por qué es mejor usar la PC que un PIC?

3. ¿Qué es el USB?

4. ¿Cuántas versiones hay de USB?

5. ¿Cuál es la velocidad de la versión más reciente?

6. ¿Qué es el host?

7. ¿Cuál es la función de los hubs?

8. ¿Qué es un pipe?

9. ¿Qué es un endpoint?

10. ¿Qué son los Phidgets?

11. ¿Cuál es el voltaje máximo de la entrada analógica del 8/8/8?

12. ¿Cuántos servos se pueden conectar al microcontrolador?

La PC en conexión con el mundo exterior

Ya conocemos los elementos básicos del puerto USB y de los Phidgets. Durante este capítulo, llevaremos a cabo la conexión de los Phidgets mediante el panel de control que provee la compañía, pero, más adelante, haremos nuestro propio software para conectarnos.

INSTALACIÓN DEL SOFTWARE

En primer lugar, debemos asegurarnos de que nuestro sistema operativo se encuentra totalmente actualizado con las últimas versiones del Framework de .NET

A continuación, debemos instalar el **Framework** de los Phidgets para Windows. Esta instalación se encargará de copiar a nuestra computadora todas las librerías necesarias para poder llevar a cabo la comunicación con los Phidgets; también, nos colocará una herramienta que nos permite verificar qué Phidgets han sido conectados y el estado en el que se encuentran.

Una vez descargado e instalado el Framework, notaremos que aparece un icono en la barra de tareas que tiene **Ph** en él. Este icono nos permite abrir el panel de control para los Phidgets. También, es posible hacer arrancar este panel de control, en forma manual, en el caso de que no se inicie automáticamente.

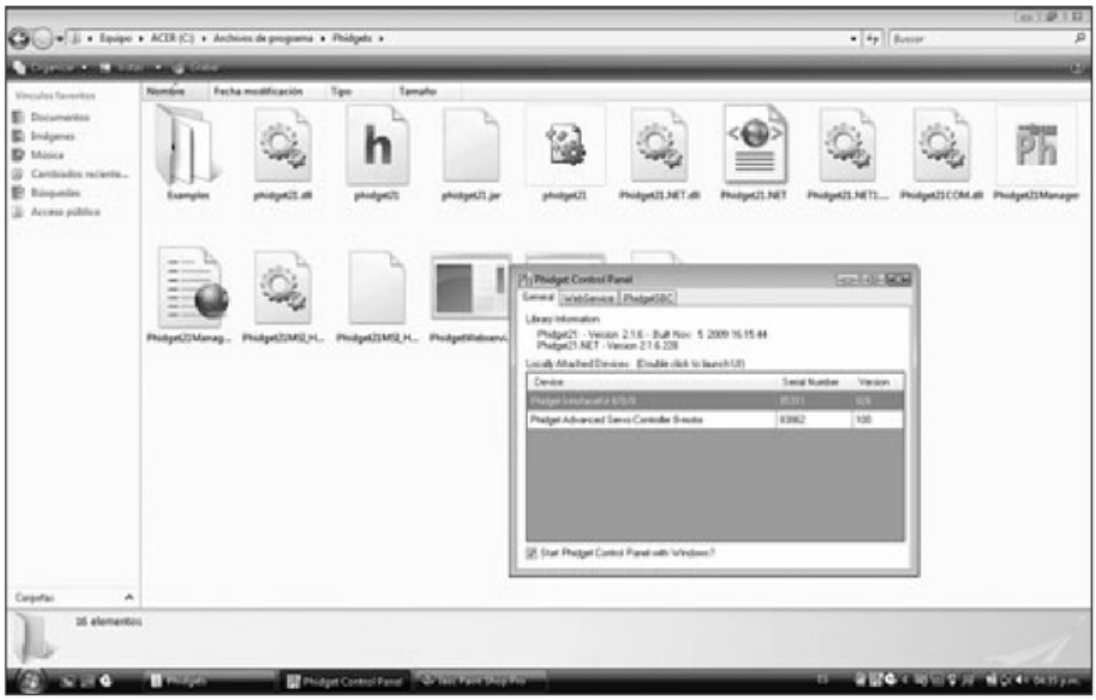


***Figura 1.** En esta figura, podemos observar el panel de control para los Phidgets.*

III ULTIMAS ACTUALIZACIONES

Para tener las últimas actualizaciones de Windows, hay que ejecutar Internet Explorer, luego, seleccionar el menú **Herramientas** y, por último, **Actualización de Windows**. Tener el sistema operativo actualizado nos permite trabajar con la última versión del Framework de .NET.

El panel de control para los Phidgets, que vemos en la **Figura 1**, tiene incorporadas dos pestañas. La primera de ellas nos permite controlar los Phidgets conectados a nuestra computadora. La segunda pestaña está relacionada con el **WebService** de Phidgets, tema que no abordaremos en este libro.

En la pestaña **General**, aparecen listados los Phidgets que se encuentren conectados a la computadora en el momento. Podemos observar el tipo de Phidget, el número de serie y la versión correspondiente. Por medio de esta lista, es posible ir, en forma directa, a una aplicación de control para el Phidget correspondiente. Sólo necesitamos efectuar un doble clic sobre el nombre.

***Figura 2.** Ésta es la aplicación de control para el 8/8/8.*

En la **figura 2**, observamos la aplicación de control para el 8/8/8. Esta ventana nos muestra la información general del Phidget. Podemos ver qué cantidad de entradas digitales y analógicas tiene el kit de interfaz, así como cuántas salidas digitales posee. Es posible, también, leer los valores de las entradas y enviar

DESCARGA DEL FRAMEWORK PARA PHIDGETS

Para descargar el Framework, es necesario ir al sitio web de Phidgets: **www.phidgets.com**. En el menú principal, encontraremos un enlace con el nombre de **Downloads**. Cuando seleccionamos nuestro sistema operativo, en la sección superior de la página resultante, se mostrará el enlace para llevar a cabo la descarga.

valores de salida. Esta aplicación es muy útil para probar nuestros circuitos sin tener que hacer un programa completo.

Prueba del 8/8/8

A continuación probaremos el dispositivo 8/8/8. Con el, leeremos una entrada digital, dos entradas analógicas y una salida digital. Los componentes que necesitaremos aparecen listados en la tabla que vemos a continuación.

COMPONENTE	CANTIDAD
Phidget Mini Joystick Sensor	1
LED	1
Destornillador pequeño	1
Cable negro: 15 cm	1
Cable verde o de otro color: 15 cm	1
Conectores para sensores	2

Tabla 1. *Necesitaremos estos componentes para la prueba.*

Figura 3. *Todos los componentes de la Tabla 1, sobre nuestra mesa de trabajo.*

III NÚMERO DE SERIE PERDIDO

El número de serie es importante para poder conectarnos a los Phidgets. Si perdemos la etiqueta que está pegada inicialmente, podemos encontrar este número en el panel de control. Lo mejor es anotar los números de serie en un lugar seguro para tenerlos como respaldo.

Empecemos por conectar el **LED** a la salida digital. Para facilitarnos el proceso, lo podemos hacer en la salida 7 o la salida 0, ya que éstas se encuentran colocadas junta a una conexión a tierra. Ayudándonos con el destornillador, debemos colocar las terminales del LED en los conectores correspondientes. Conectamos el cátodo del LED a la tierra, y la otra terminal, a la salida digital, tal como podemos observar en la siguiente figura.

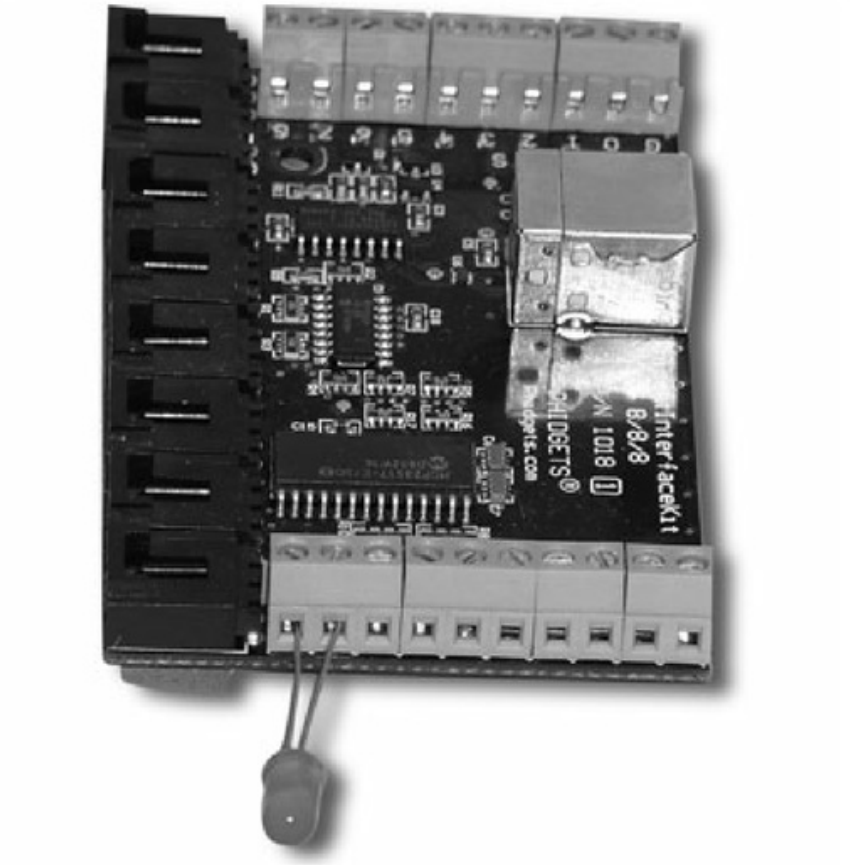


***Figura 4.** Ésta es la forma en la que el LED debe ser conectado a la salida digital.*

A continuación, conectamos la entrada digital. Para esto, usaremos los cables, y el destornillador, también, nos ayudará a realizar esta tarea. En los conectores para las entradas digitales, colocaremos el cable negro en la conexión a tierra. El cable verde deberá estar conectado a la entrada digital 0. Esto se observa en la siguiente figura.

III EL CÁTODO DEL LED

Para encontrar el **cátodo** del LED, debemos observar sus terminales. El cátodo tiene una terminal más corta, y se puede observar una muesca en la base del LED que también lo indica. Si el LED no prende, es posible que se encuentre conectado con la polaridad incorrecta, en cuyo caso solamente hay que intercambiar las terminales.

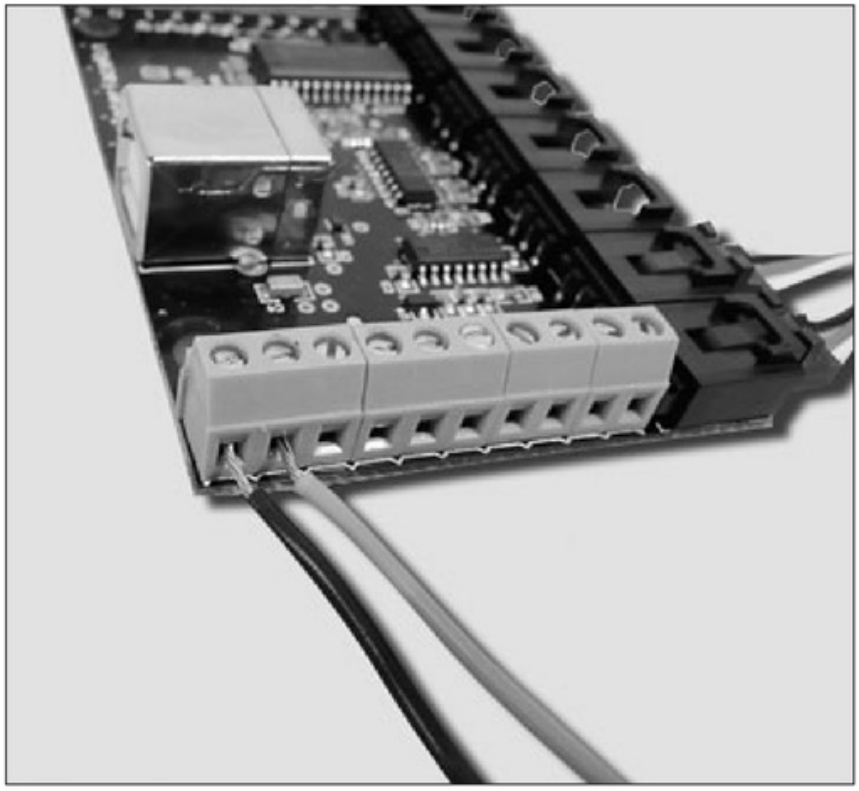


Figura 5. *Ésta es la forma en que los cables deben estar conectados.*

Los otros extremos de los cables se deben conectar a la salida digital del joystick. El cable negro se conecta a tierra, marcada como **GND**. El cable de color va a la terminal que dice **INPUT**. La **figura 6** nos lo muestra.

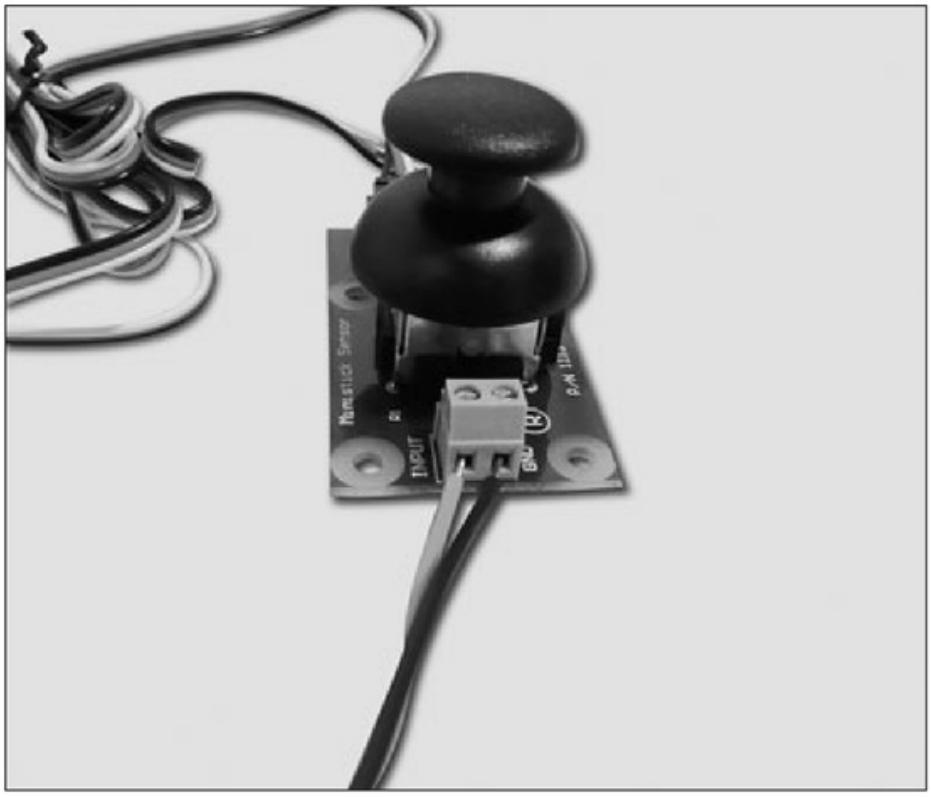


Figura 6. *De esta forma, los cables quedan conectados al joystick.*

Lo último que necesitamos es conectar las salidas analógicas del joystick a las entradas analógicas del 8/8/8. Esto es muy sencillo de hacer, ya que conectamos los dos circuitos por medio de los cables provistos en el kit. Las entradas analógicas que utilizaremos son la 0 y la 1. La siguiente figura nos muestra las conexiones completas.

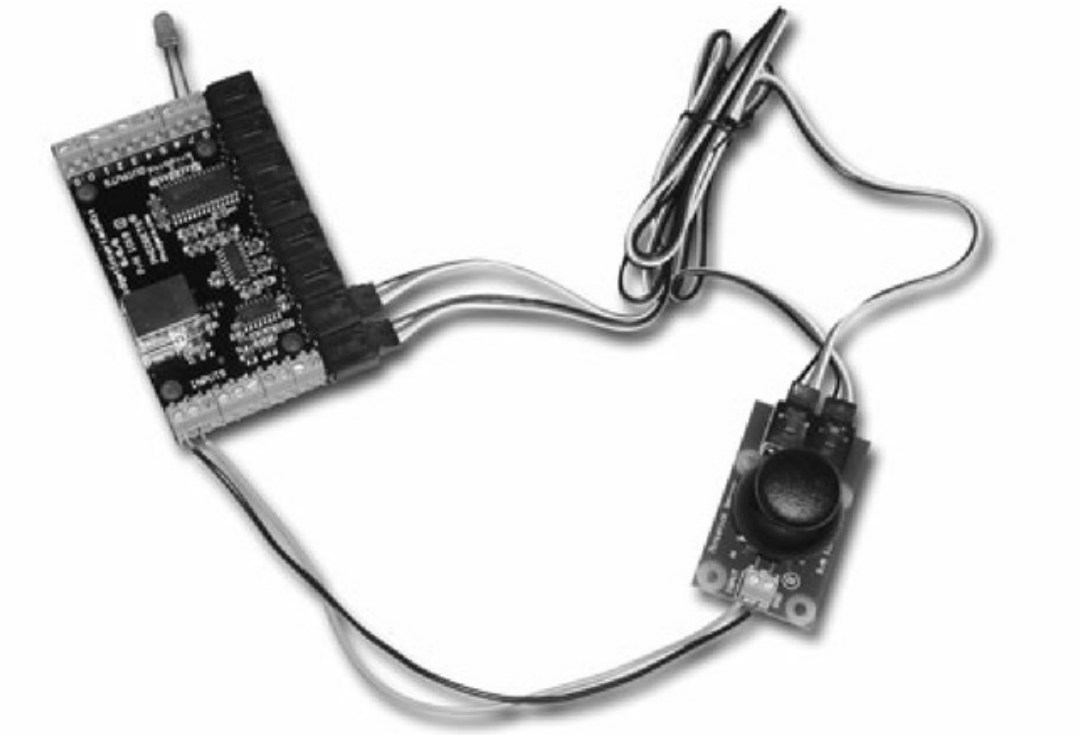


Figura 7. *El 8/8/8 y el joystick ya están conectados para proceder a la prueba.*

A continuación, realizamos la prueba. Para esto, activamos el panel de control de Phidgets y conectamos el 8/8/8 por medio del cable USB. Tan pronto como los controladores se instalen, de manera automática, aparecerá en pantalla el 8/8/8, más precisamente en la lista de Phidgets.

Ahora, realizamos un doble clic en el 8/8/8 listado, y aparecerá la aplicación de control. Al mover el joystick, los valores de las entradas analógicas cambian. Si se oprime el joystick, el valor de la entrada digital debe cambiar y mostrar el **CheckBox** marcado. Si marcamos el CheckBox de la salida digital, entonces el LED deberá iluminarse. Con esto, hemos probado que el 8/8/8 funciona en forma correcta.

Prueba del controlador de los servos

Al igual que con el 8/8/8, probaremos con facilidad el controlador de los servos. También, para esto, contamos con una pequeña aplicación de control en el panel de control de Phidgets. Con esta aplicación, es posible verificar el control sobre los servos. Lo primero que debemos hacer es indicar cuál es el servo sobre el que vamos a trabajar. Podemos cambiar la posición, la velocidad y la aceleración.

Hay dos marcadores para la posición. Uno indica la posición máxima del servo, y el otro, la mínima. Estos valores son importantes, ya que con ellos evitaremos que el servo trate de extender su movimiento más allá de sus límites, impidiendo que se fuerce el motor o el sistema mecánico.

Para probar el controlador de servos, necesitamos los componentes que aparecen en la tabla

COMPONENTE	CANTIDAD
1061 PhidgetAdvancedServo	1
Servo	1
Fuente de poder	1

***Tabla 2.** Éstos son los componentes necesarios para probar el controlador de servos.*

Conectar el servo es muy sencillo. Del servo, sale un conector que tiene tres cables: uno **negro**, otro **rojo** y uno **amarillo** o de otro color. El conector va de manera directa al controlador. En el lado izquierdo del controlador, se encuentran los pines donde colocaremos el conector del servo. Debemos ser cuidadosos, porque éste debe ser conectado con la polaridad correcta. En la parte inferior de los pines, observamos las letras **B**, **R** y **W**. Éstas nos ayudan a ubicar el conector con la polaridad correcta. El cable negro debe ir al pin marcado con **B**.

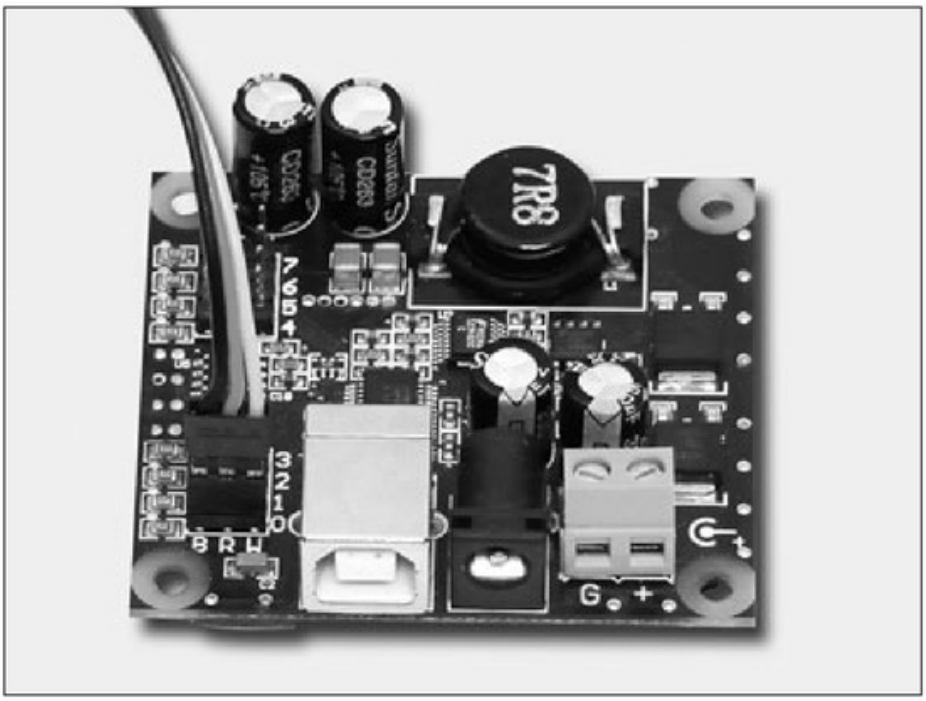


***Figura 8.** Aquí podemos observar cómo conectar el servo con la polaridad correcta.*

Una vez conectado el servo, el paso siguiente es vincular la fuente de poder. A partir de ahí, conectamos el cable USB. La computadora instalará, en forma automática,

el **driver** para el controlador de servos. En el panel de control de Phidgets, podemos verlo listado y, al hacer doble clic, iremos directamente a la aplicación de control donde podemos experimentar con el servo.

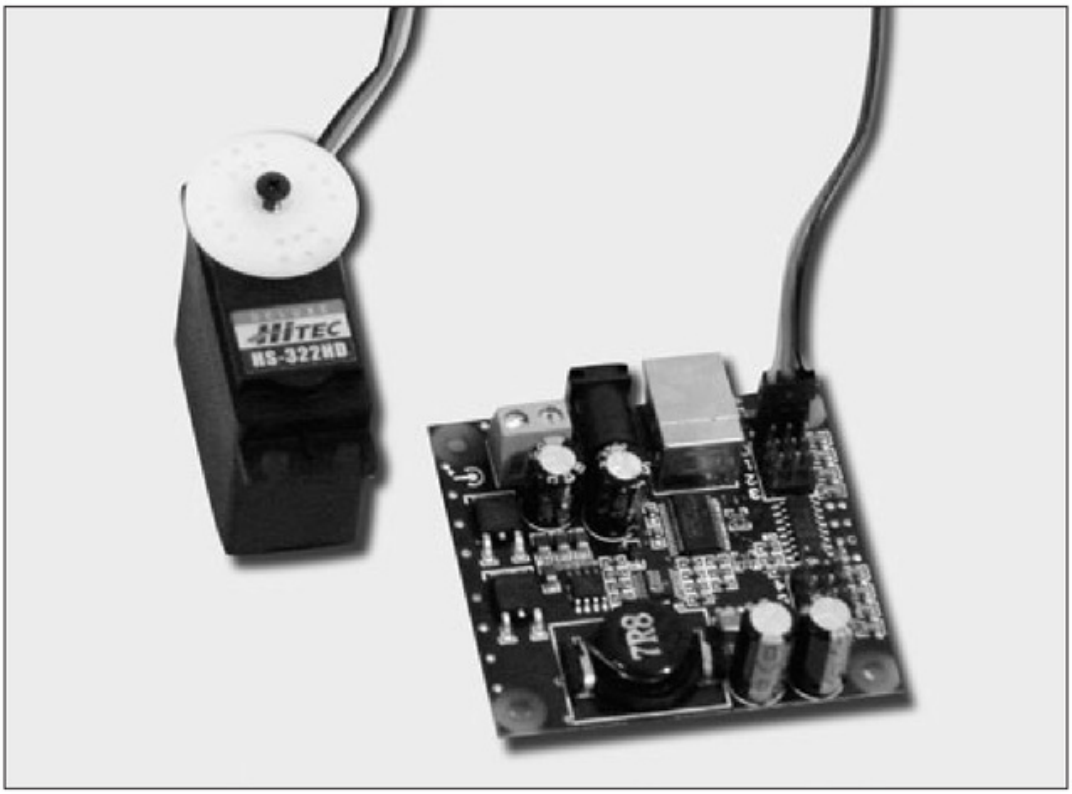


Figura 9. *En esta figura, observamos el controlador con un servo conectado a él.*

Ahora, simplemente, podemos experimentar con los servos, su posición y velocidad.

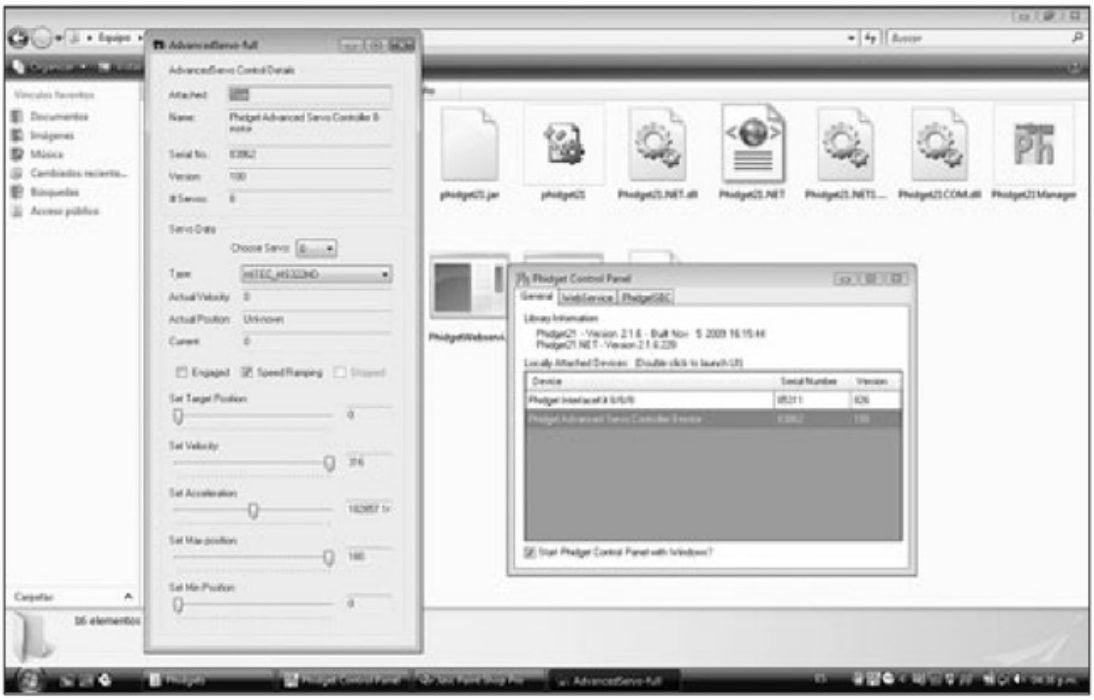


Figura 10. *La aplicación de control nos permite experimentar con los servos.*

Esto cubre las aplicaciones que nos brinda el propio panel de control, pero, en realidad, deberemos aprender a crear nuestra propia aplicación para controlar los diferentes dispositivos y, de esta forma, hacer que nuestra lógica controle el robot.

CREACIÓN DE UN PROGRAMA PARA EL PHIDGET 8/8/8

Es tiempo de crear una aplicación base que, posteriormente, modificaremos según nuestras propias necesidades. La aplicación base llevará a cabo las acciones esenciales, como conectarse al Phidget, leer dos entradas analógicas, leer una entrada digital y mandar una salida digital.

Figura 11. *Esta versión de Phidget combina un 8/8/8 con una pantalla para poder mostrar mensajes o información.*

Creación de la interfaz de usuario

Empecemos por crear un proyecto de C#. El proyecto deberá ser una aplicación de ventana, a la cual llamaremos **BasePhidgets**. Nuestra aplicación tendrá una forma determinada, donde necesitamos colocar diversos controles. Podemos ver la interfaz de usuario que necesitamos en la siguiente figura.

III SELECCIÓN DEL SERVO PARA EL CONTROLADOR

Es importante no usar servos que excedan las capacidades del controlador. El modelo en particular que usamos puede manejar servos que consuman hasta 1.5 amperios de corriente. Debemos tener en cuenta que cada servo tiene sus especificaciones técnicas, las cuales pueden consultarse en el sitio web del fabricante.

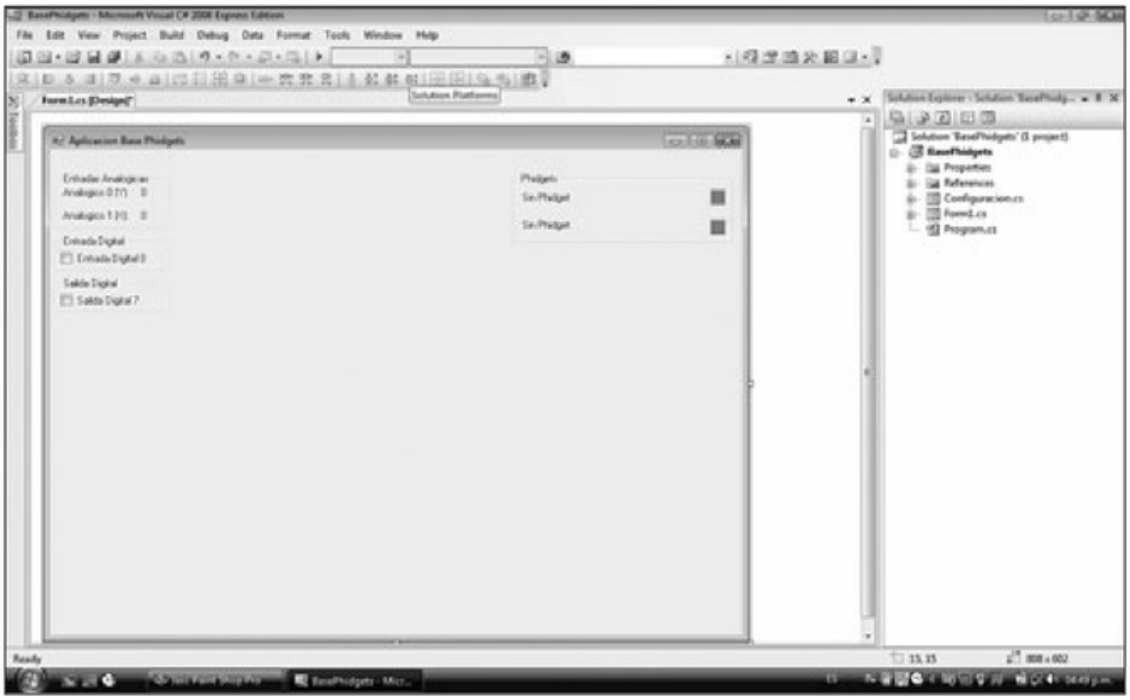


Figura 12. *Aquí vemos los controles y su ubicación para la aplicación base que deseamos crear.*

Lo primero que insertamos es una barra de menú o **MenuStrip**. En ésta, debemos ingresar dos menús. El primer menú es el **Archivo**, y el segundo, **Phidget**. Dentro del menú **Archivo**, colocamos un elemento de menú llamado **Salir** mientras que, en el menú **Phidget**, ubicamos un elemento de menú llamado **Configurar**. El menú **Salir**, simplemente, se encargará de cerrar la aplicación, y **Configurar** nos mostrará un cuadro de diálogo donde podremos ingresar el número de serie del Phidget. Más adelante, veremos cómo podemos utilizar el registro de Windows, para que el número de serie quede guardado y no sea necesario escribirlo cada vez que ejecutemos el programa.

Como deseamos conocer el momento en que el Phidget se encuentre conectado, en nuestra forma, insertaremos una zona donde podamos ver el estado del Phidget. Esto lo haremos mediante una etiqueta o **Label**, que nos indicará el nombre del Phidget conectado, y un cuadro de texto o **TextBox** al cual le cambiaremos el color de fondo según el estado de conexión. Si el Phidget no está conectado, observaremos

III SOBRE LOS NOMBRES DE LOS CONTROLES

Es importante que, cuando realicemos nuestro programa, ingresemos los mismos nombres que aparecen en el libro, esto con el fin de evitar confusiones. Con posterioridad, podremos hacer nuestros propios programas y seleccionar los nombres que consideremos adecuados.

el color rojo; si el Phidget se encuentra conectado aparecerá el fondo en color verde; y, si existiera algún problema, el color que mostrará será el amarillo. A la etiqueta, la llamamos **lblPhidget0** y le ponemos en su propiedad **Text** el valor **"Sin Phidget"**. Al cuadro de texto, le debemos dar el nombre **txtPhidget0**, su tamaño es de 20 por 20, y el color que ponemos en su propiedad **BackColor** es **Red**, de la colección de colores web. Podemos ingresar este TextBox como de sólo lectura.

Una vez que ingresamos estos controles, debemos encerrarlos en un **GroupBox**. En la propiedad **Text** del GroupBox, escribiremos **"Phidgets"**. Al GroupBox y sus contenidos, los ubicamos cerca de la esquina superior derecha.

Ahora, procedemos a colocar las etiquetas que nos mostrarán los valores de las entradas analógicas. En primer lugar, insertamos una etiqueta que sólo tenga el siguiente valor en su propiedad **Text**: **"Analógico 0 (Y):"**. A su derecha, insertamos otra etiqueta con el nombre **lblAnalogico0** e iniciamos su propiedad **Text** con **"0"**. En la parte inferior de estas etiquetas, hacemos algo similar. Colocamos una etiqueta que tenga el mensaje **"Analógico 1 (X):"** y, a su derecha, una etiqueta nueva con el nombre **lblAnalogico1**, con valor de **"0"** en su propiedad **Text**. A todas estas etiquetas, las encerramos dentro de un GroupBox con el título **"Entradas Analógicas"**, al que colocamos cerca de la esquina superior derecha de nuestra forma.

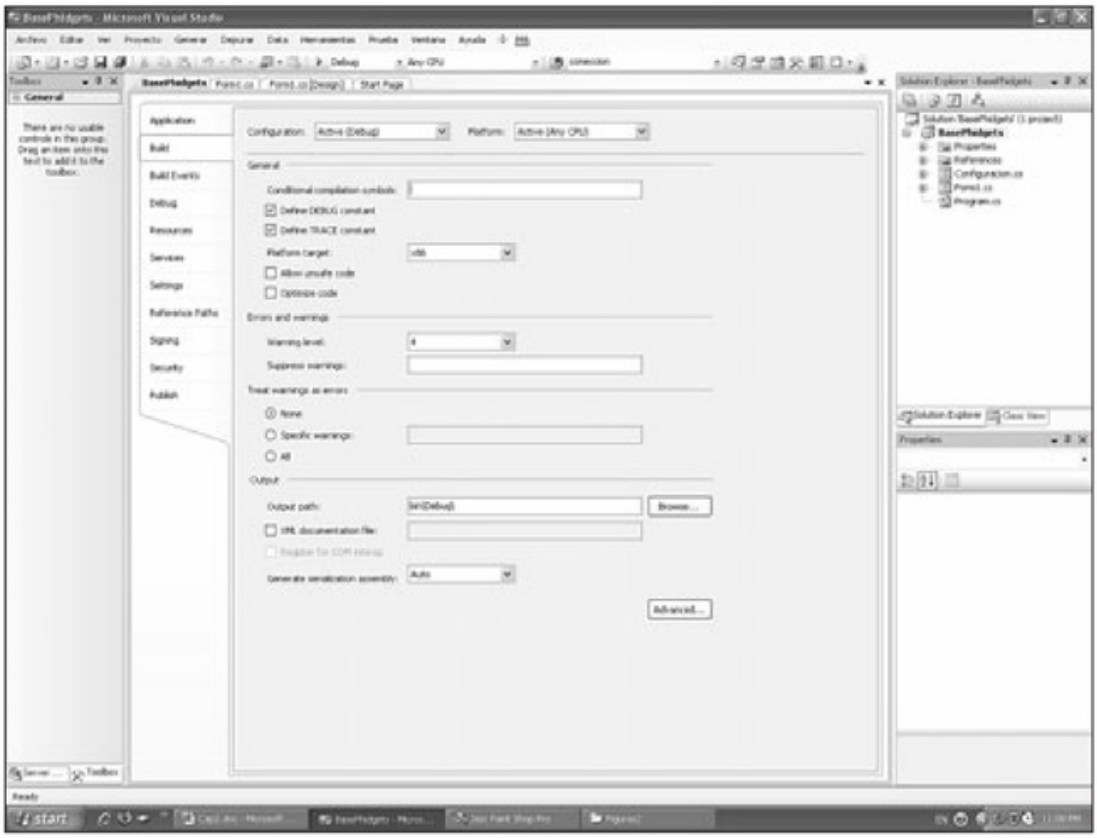


Figura 13. *Esta figura nos muestra el lugar dentro de las propiedades donde insertamos el proyecto para que trabaje en 32 bits.*

Continuemos ahora con la entrada digital. Para ésta, usaremos un control de tipo CheckBox. Cuando la entrada digital tenga un valor **Verdadero**, el CheckBox se mostrará marcado, en cambio, cuando sea un valor **Falso**, el CheckBox aparecerá sin marca. Insertamos entonces un CheckBox y le damos el nombre **chbDigital0**. En su propiedad **Text**, ponemos **"Entrada Digital 0"**. A este control, también lo encerramos en un GroupBox al cual le ponemos por título **Entrada Digital** y lo colocamos debajo del GroupBox de las entradas analógicas.

Para controlar nuestra salida digital, haremos uso de otro CheckBox. Si marcamos el CheckBox, entonces, en la salida digital 7 tendremos un valor **Verdadero**. En caso de que el CheckBox no esté marcado, la salida digital 7 tendrá un valor **Falso**. Para esto, insertamos un CheckBox con el nombre **chbSalida7** y, en su propiedad **Text**, colocamos el valor **"Salida Digital 7"**. Al igual que en los casos anteriores, lo encerramos en un GroupBox con el título **"Salida Digital"**. Ahora que ya tenemos completa nuestra forma principal, podemos comenzar a crear el cuadro de diálogo necesario para la configuración.

Creación del cuadro de configuración

Necesitamos crear un cuadro de diálogo en el cual podamos ingresar los datos de la configuración de nuestro Phidget. En realidad, no precisamos mucho, ya que lo único requerido es el número de serie del Phidget. En nuestro cuadro de diálogo, tendremos una etiqueta, un cuadro de texto y dos botones, tal y como podemos apreciar en la figura que aparece a continuación.

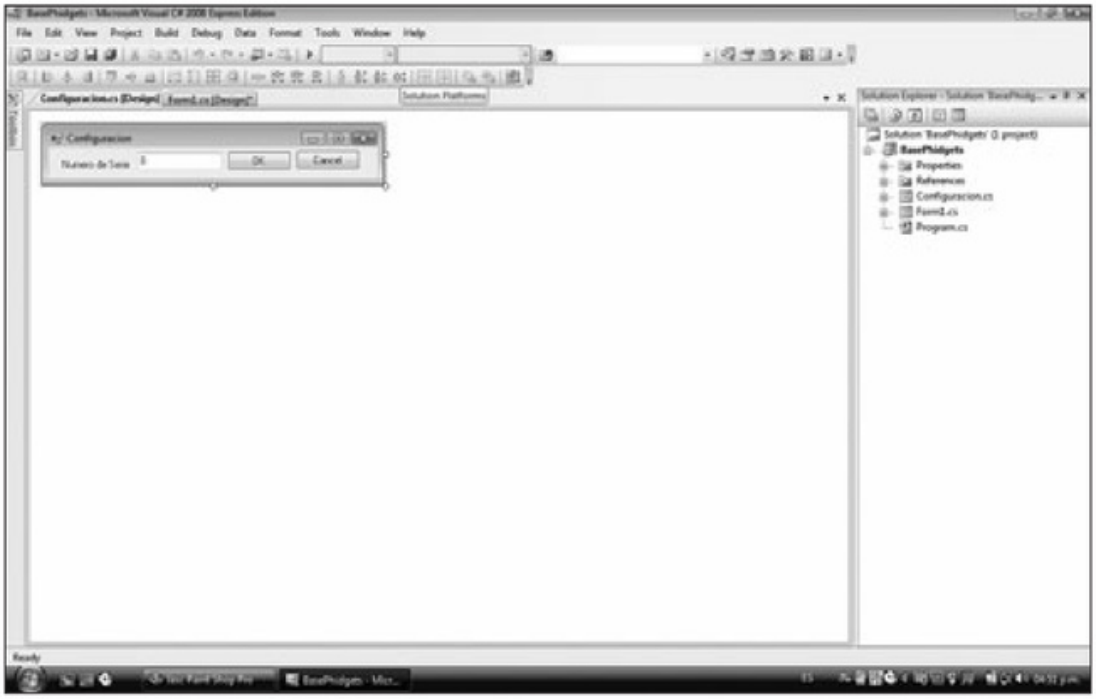


Figura 14. *Aquí podemos observar el diseño del cuadro de diálogo para la configuración del Phidget.*

Empecemos por adicionar al proyecto una nueva forma, a la que llamaremos **Configuración**. Visual Studio crea por nosotros una nueva clase llamada **Configuracion.cs**, gracias a la cual podemos ver la forma en el editor. A continuación, adicionamos una etiqueta que llevará en su propiedad **Text** el valor **"Número de serie"**. A la derecha de la etiqueta, insertamos un TextBox, el cual se llamará **txtNumeroSerie** y deberá contener **"0"** dentro de la propiedad **Text**. Luego, adicionaremos dos botones que servirán para que el cuadro de diálogo regrese un valor y sepamos cuál de ellos fue el presionado. El primer botón debe ubicarse a la derecha del TextBox. En su propiedad **Text**, ingresamos **"OK"** y, en la propiedad **DialogResult**, el valor **OK**. De forma similar, el otro botón lleva **"Cancel"** en la propiedad **Text**, y su propiedad de **DialogResult** debe tener el valor **Cancel**. El primer botón tiene por nombre **btnOK**, y el segundo, **btnCancel**.

Crear el código de la aplicación

Ya tenemos la interfaz de usuario terminada y podemos empezar a crear el código de nuestra aplicación. El cuadro de diálogo de la configuración es el más sencillo, por lo que comenzaremos desde aquí. Este cuadro de diálogo sólo necesita de un dato. Este dato consiste en el número de serie del Phidget y es de tipo entero. El dato deberá tener un acceso público, tal y como lo muestra el siguiente código.

```
// Variables necesarias
private int numeroSerie;     // Numero de serie para el phidget
```

Como será necesario enviar esta información a la forma principal y nuestra variable tiene un acceso privado, entonces, debemos crear una propiedad para ella.

```
// Propiedad para el numero de serie
public int NumeroSerie
{
    get
```

III CÓMO ADICIONAR UNA FORMA

Para adicionar una forma, en el explorador de soluciones debemos efectuar un clic con el botón derecho del mouse sobre el nombre de nuestro proyecto; luego, seleccionamos las opciones **Agregar** y **Forma**, de Windows. El ambiente de desarrollo agrega tanto la forma como el código necesario para ella, de manera automática.

```
        {
            return numeroSerie;
        }

        set
        {
            numeroSerie = value;
            txtNumeroSerie.Text = numeroSerie.ToString();
        }
    }
```

Debemos tener en cuenta que estamos aprovechando el set para actualizar el valor del TextBox en el caso de que recibamos un número de serie desde la forma principal. En ese momento, crearemos los handlers para los botones. En el caso específico del botón **OK**, lo que haremos será leer el valor del número de serie introducido por el usuario en el TextBox para ingresarlo en la variable interna. A continuación, el cuadro de diálogo se cerrará.

```
    private void btnOK_Click_1(object sender, EventArgs e)
    {
        // Obtenemos el numero de serie escrito en el text box
        numeroSerie = Convert.ToInt32(txtNumeroSerie.Text);

        // Cerramos el dialogo
        this.Close();
    }
```

El caso del botón **Cancel** es más sencillo, pues no debemos realizar ningún trabajo con él, sólo cerrar el cuadro de diálogo.

```
    private void btnCancel_Click(object sender, EventArgs e)
    {
        // Cerramos el dialogo
        this.Close();
    }
```

Esto es todo para la configuración; ya podemos empezar a escribir el código de la forma principal. Ahora, debemos ir al código de esta forma. Indicamos en la

parte superior los **namespace** que utilizaremos. Visual Studio se encargará de colocar los namespace más comunes por nosotros, pero debemos adicionar de manera manual los siguientes en nuestro código:

```
// Necesario para el registro
using Microsoft.Win32;

// Necesario para los Phidgets
using Phidgets;
using Phidgets.Events;

using System.Threading;
```

Debido a que haremos uso del registro de Windows para guardar la configuración de los Phidgets, necesitamos incluir **Microsoft.Win32**, el cual nos provee de todas las clases y métodos necesarios para lograrlo.

Para poder usar los Phidgets, necesitamos el namespace de los Phidgets, el cual contiene las clases principales que representan el hardware con el que nos conectaremos. La programación de los Phidgets tiene un esquema con orientación a eventos. Entonces, nuestro programa puede recibir eventos cuando ocurren sucesos como la conexión, desconexión del Phidget, cambio en el valor de un sensor, etcétera. Todo lo necesario para que podamos usar estos eventos está en el namespace **Phidgets.Events**. Más adelante, agregaremos las funciones que actuarán como handlers a dichos eventos. Para el caso de los servos, necesitaremos dar un tiempo de espera a la ejecución del programa mientras el servo adquiere su nueva posición. Para lograrlo, precisamos una función que se encuentra en el namespace de **System.Threading**. Nuestra clase principal será la forma de la aplicación, por lo que en ésta, ingresaremos todas las variables necesarias para trabajar. En primer lugar, debemos declarar algunas variables a nivel de clase, que podemos observar en el siguiente código.

III EL REGISTRO DE WINDOWS

El **registro de Windows** es una base de datos donde se almacena información de la configuración e instalación de los programas que existen en nuestra computadora. Los programas tienen la opción de modificarlo si necesitan guardar datos de configuración que se mantengan entre sus diferentes ejecuciones.

Figura 15. *También podemos utilizar para nuestros proyectos sensores que pueden medir el pH de una substancia, como el que se muestra en la fotografía.*

```
// Datos necesarios para la aplicacion base

private int numeroSerie;          // Numero de serie del phidget
private bool conectado;           // Para saber si hay conexion
private InterfaceKit iKit;        // Interfaz al kit
```

La primera variable es de tipo entero y la usaremos para guardar el valor del número de serie del Phidget; este número es necesario para poder conectarse. Como realizar operaciones sobre el Phidget cuando éste no se encuentra conectado nos puede llevar a tener problemas con la ejecución del programa, usaremos una variable llamada **conectado** para estar al tanto del estado de la conexión. La próxima variable es de tipo **InterfaceKit**. Esta clase es provista por la biblioteca de Phidgets y representa el hardware de un kit de interfaz, como el 8/8/8. A la variable, la llamaremos **iKit** y, por medio del objeto que luego ubicaremos por instancia, nos podremos comunicar con el hardware. En el constructor de la forma, debemos agregar ciertos códigos.

```
public Form1()
 {
     InitializeComponent();

     // Inicializamos la aplicacion

     // Empezamos como si no estuvieramos conectados
     conectado = false;


     // Colocamos un valor de default para el numero de serie
     numeroSerie = 0;
```

```
            // Verificamos si existe la llave en el registro
            RegistryKey sfwKey = Registry.LocalMachine.OpenSubKey
("Software");
            RegistryKey redKey = sfwKey.OpenSubKey("RedUsers");

            // Verificamos si existe
            if (redKey == null) // la llave no existe
            {

                // Creamos la llave
                sfwKey = Registry.LocalMachine.OpenSubKey("Software",
true);
                redKey = sfwKey.CreateSubKey("RedUsers");

                RegistryKey robotKey = redKey.CreateSubKey("Robots");

                // Colocamos un valor de default
                robotKey.SetValue("NumeroSerie0", (object)numeroSerie);

            }
            else // la llave existe
            {
                RegistryKey robotKey = redKey.OpenSubKey("Robots");

                // Obtenemos el numero de serie guardado
                numeroSerie = (int)robotKey.GetValue("NumeroSerie0");
            }

        }
```

Lo que realiza este código es leer la información de configuración desde el registro de Windows. Pero veámoslo más a fondo. Por seguridad, empezamos ingresando el valor **false** en la variable conectado. Como aún no sabemos el número de serie, inicializamos la variable **numeroSerie** con el valor de **0**. En este momento, empezaremos a trabajar con el registro. Tenemos una clase llamada **RegistryKey**: por medio de esta clase podemos realizar algunas operaciones sobre el registro. Creamos una instancia de esta clase que se llama **sfwKey** y abrimos en nuestra máquina local, es decir, en nuestra computadora, la llave llamada **Software**. En esta llave del registro, suele salvarse información de configuración de los programas. Dentro de esta llave, intentaremos encontrar una subllave llamada **RedUsers**. Ésta es la subllave donde luego colocaremos nuestra

información de configuración. El método **OpenSubKey()** regresa el valor **null** si la llave no existe. Esto puede ocurrir la primera vez que ejecutamos el programa, ya que no existiría previamente ninguna información de configuración. Si la llave existe, significa que ya debe haber información de configuración, y **OpenSubKey()** nos da una referencia a la subllave deseada. El método **OpenSubKey()** requiere de un parámetro que es el nombre de la llave o subllave que deseamos abrir.

Es importante notar que, la primera vez que la usamos, lo hacemos sobre el registro de la máquina actual y, la segunda vez, sobre la llave que obtuvimos con la primera. Veamos el caso en el cual la llave no existe. En primer lugar, debemos crear entonces la subllave que corresponde. En este caso, abrimos otra vez **Software**, pero con una versión sobrecargada de **OpenSubKey()**. En esta versión, colocamos un segundo parámetro que lleva el valor **true**. Esto indica que, también, vamos a poder escribir sobre el registro y no solamente leerlo. Una vez que se encuentra abierto **Software**, procedemos a crear la subllave que se llama **RedUsers**. Después de creada por el método **CreateSubKey()**, recibimos una referencia a esa llave en la variable **redKey**. El método **CreateSubKey()** requiere sólo de un parámetro que es el nombre con el cual creamos la subllave. La subllave **RedUsers** puede guardar configuraciones de varios proyectos, por lo que crearemos una nueva subllave llamada **Robots**, que será particular para este proyecto. Dentro de **Robots**, podremos ingresar los valores que deseemos.

Figura 16. *Éste es un sensor de distancia que podemos usar con los Phidgets. El sensor utiliza rayos infrarrojos.*

Ahora, podemos colocar el valor. En este caso, únicamente necesitamos de un valor que es el número de serie. Para ingresar un valor dentro de una llave o subllave, invocamos el método **SetValue()** de la llave. Este método necesita de dos parámetros; el primer parámetro es una cadena que contiene el nombre del valor, y el segundo pará-

metro es el valor por ingresar. Nuestro valor se llamará **NumeroSerie0** e insertaremos el contenido de la variable **numeroSerie**. Si el programa ya se hubiera ejecutado con anterioridad, entonces, la llave ya existiría. En este caso, se ejecutaría el código contenido en **else**. Este código es más sencillo, ya que sólo debemos leer el valor guardado. La primera línea de código abre la subllave **Robots** y, a partir de ella, usamos el método **GetValue()** para leer el valor guardado en el registro. El método **GetValue()** necesita de un parámetro que es una cadena que contiene el nombre del valor por leer. Con ayuda del método, leemos entonces el valor **NumeroSerie0**. **GetValue()** regresa un valor de tipo **object**, por lo que es necesario hacer un **type cast** a **int**; esto se debe a que **numeroSerie** es de tipo entero. Con esto, finaliza el constructor de la forma. Lo siguiente que haremos es el código para el menú **Salir**. Debemos crear su handler y colocar el siguiente código.

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

El código es muy sencillo, ya que únicamente debemos cerrar la forma.

A continuación, crearemos el handler para el menú **Configurar**. Cuando se seleccione este menú, deberá aparecer el cuadro de diálogo para la configuración. Si el usuario presiona el botón **OK**, entonces, debemos actualizar el registro con la nueva configuración. En caso de que el usuario pulsara **Cancel**, no hacemos nada. El código del handler para este menú se muestra a continuación.

```
        private void configurarToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            Configuracion dlgConfig = new Configuracion();
            dlgConfig.NumeroSerie = numeroSerie;

            if (dlgConfig.ShowDialog() == DialogResult.OK)
            {
                // Escribimos el nuevo valor
                numeroSerie = dlgConfig.NumeroSerie;

                // Abrimos la llave
                RegistryKey sfwKey =
```

```
Registry.LocalMachine.OpenSubKey("Software", true);
                RegistryKey redKey = sfwKey.CreateSubKey("RedUsers");

                RegistryKey robotKey = redKey.CreateSubKey("Robots");

                // Colocamos el nuevo valor
                robotKey.SetValue("NumeroSerieO", (object)numeroSerie);

            }
        }
```

Dentro de este handler, lo primero que hacemos es crear una instancia del cuadro de diálogo. A esta instancia, la llamamos **dlgConfig**. Como el cuadro de diálogo por sí mismo no conoce el valor del número de serie, aprovechando la propiedad **NumeroSerie** que hemos programado antes, le pasamos el valor contenido en la variable de la forma **numeroSerie**. Así, cuando el cuadro de diálogo se muestre, podrá colocar en su TextBox el valor correcto. Con la instancia de la clase del cuadro de diálogo, nos crea el objeto, pero no lo muestra. Para mostrarlo, tenemos que hacer uso del método **ShowDialog()**. Este método muestra el cuadro de diálogo para que podamos trabajar con él. El método regresará un valor de tipo **DialogResult**. Este valor depende del botón que se haya oprimido, en nuestro caso **OK** o **Cancel**; por eso cuando creamos la forma del cuadro de diálogo, le dimos un valor de retorno a los botones. Solamente nos interesa llevar a cabo una acción cuando el usuario haya presionado el botón **OK** en el cuadro de diálogo, por eso, ingresamos un if comparando el valor regresado por **ShowDialog()** y **DialogResult.OK**. Lo que haremos si el usuario presiona **OK** es actualizar el valor del número de serie, tanto en la variable como en el registro. Para actualizar el valor en la variable, simplemente asignamos el valor por medio de la propiedad **NumeroSerie**. Ahora que tenemos el valor en la variable, escribimos este nuevo valor en el registro usando la metodología que ya conocemos. Debemos crear un handler para el evento **Load** de la forma. En este handler nos conectaremos y configuraremos el Phidget. El código queda como se muestra a continuación.

III EL EVENTO LOAD

Durante el ciclo de vida de una forma, se generan diferentes eventos, uno de estos eventos es el conocido como evento **Load** que se genera cuando la forma se está cargando. Es un buen lugar para llevar a cabo inicializaciones de los controles u objetos que forman parte de ella.

```
        private void Form1_Load(object sender, EventArgs e)
        {
            // Inicializamos el objeto InterfaceKit
            try
            {
                // Instanciamos el objeto
                iKit = new InterfaceKit();

                // Colocamos los handlers relacionados con la conexion
                iKit.Attach += new AttachEventHandler(iKit_Conectar
Handler);
                iKit.Detach += new DetachEventHandler(iKit_Desconectar
Handler);

                // Colocamos el handler relacionado con el error
                iKit.Error += new ErrorEventHandler(iKit_ErrorHandler);

                // Colocamos el handler para las entradas digitales
                iKit.InputChange += new InputChangeEventHandler
(iKit_DigitalHandle);

                // Colocamos el handler para las entradas analogicas
                iKit.SensorChange +=
new SensorChangeEventHandler(iKit_AnalogoHandle);

                // Abrimos para conexión
                iKit.open(numeroSerie);


            } // fin de try
            catch (PhidgetException ex)
            {
                // Enviamos mensaje con el error
                lblPhidget0.Text = ex.Description;

                // Indicamos con amarillo que hay problemas
                txtPhidget0.BackColor = System.Drawing.Color.Yellow;
            }

        }
```

Algunas veces, es probable que tengamos problemas al intentar crear la conexión al Phidget, y esto puede llevar a que nuestro programa deje de funcionar. Para evitarlo, haremos uso de **try/catch**. Dentro de try, ingresaremos el código con riesgo y en catch, únicamente, el mensaje del error correspondiente que pudiera haber sido generado. Empecemos a colocar código en el try. La primera línea que tenemos es la puesta en instancia del objeto **InterfaceKit**. Con anterioridad, sólo habíamos creado su variable, pero, ahora, la estamos ingresando en una instancia. Este objeto representa a nuestro kit de interfaz y, por medio de él, podremos leer las entradas y mandar información a las salidas, así como llevar un control sobre todo el kit.

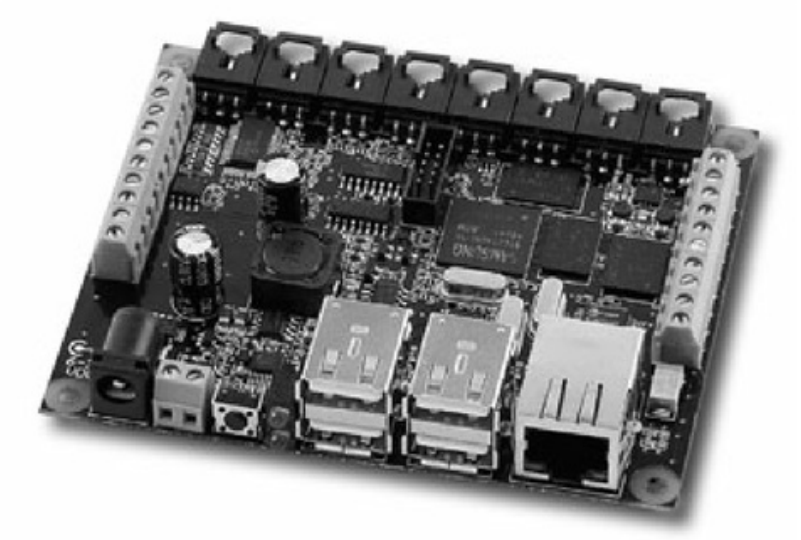


Figura 17. *Este es el PhidgetSBC, es más avanzado ya que puede operar sin la necesidad de una computadora.*

Como sabemos, los Phidgets trabajan en un modelo orientado a eventos. Cada dispositivo Phidget puede generar diferentes eventos y, si nos interesa o conviene a nuestra aplicación, podemos generar handlers para ellos. Como la biblioteca de Phidgets no puede saber con anticipación el nombre de la función que va a actuar como handler, nosotros tenemos que registrarla de tal forma que, cuando se genere el evento, el handler se invoque en forma automática. Para este programa, nos interesan varios eventos: el de conexión, el de desconexión, el de error, el de cambio de la entrada analógica y el de cambio de la entrada digital. Para cada uno de ellos, colocaremos una función. Primero, tenemos los handlers relacionados con los eventos de conexión. Ingresamos la función **iKit_ConectarHandler** para el evento **Attach**, es decir, el de la conexión. Luego, para el evento de la desconexión, es decir **Detach**, agregamos la función **iKit_DesconectarHandler**. Posteriormente, crearemos el código de dichas funciones. Para el evento de error, asignamos la función handler **iKit_ErrorHandler**. Cuando cambia el valor de una entrada digital, se genera el evento **InputChange,** y le asignaremos como handler la función **iKit_DigitalHandle**. De igual forma, cuando cambia el valor de una entrada analógica, se genera un evento, en este caso **SensorChange**. La función que usaremos

como handler será **iKit_AnalogoHandle**. Una vez que los handlers han sido asignados, podemos llevar a cabo la conexión o apertura del Phidget. Para esto, usaremos el método **open()**. En la versión que emplearemos de dicho método, ingresaremos como parámetro el número de serie. Esto termina la sección del try, que esperamos sea exitoso en la gran mayoría de los casos.

Si llegara a ocurrir algún problema en la conexión hacia el Phidget, entonces, el código del catch se ejecuta. Nuestro catch recibirá como parámetro un objeto de tipo **PhidgetException**. Esto es útil ya que, por medio de este objeto, podemos obtener una descripción del tipo de problema que se tiene. En el código del catch, simplemente, ingresamos la descripción del error en la etiqueta que se encuentra en nuestra forma e insertamos, en color amarillo, el color de fondo del TextBox que también se encuentra en nuestra forma. Ingresemos ahora los handlers que referenciamos en el código anterior. Empecemos por el relacionado con **Attach**, es decir, la conexión.

```
// Handler de la conexión
public void iKit_ConectarHandler(object sender, AttachEventArgs e)
{
    // Indicamos que hay conexión
    conectado = true;

    // Enviamos mensaje de la conexión
    lblPhidget0.Text = e.Device.Name;

    // Indicamos con color verde que hay conexión
    txtPhidget0.BackColor = System.Drawing.Color.Lime;

}
```

El handler tiene un acceso público, no regresa ningún valor. El nombre es el que ingresamos antes: **iKit_ConectarHandler**. Como parámetros, necesitamos, en primer lugar,

III LA RAZÓN DEL NÚMERO DE SERIE

El número de serie es necesario, ya que podemos conectar varios Phidgets a la computadora, incluso varios del mismo modelo. Por esta razón, para especificar cuál es el que deseamos abrir y tener más control, utilizamos el número de serie para referenciarlo.

un objeto de tipo **object** llamado **sender** y, luego, un objeto llamado **e** de tipo **AttachEventArgs**. Dentro del objeto **e**, debe aparecer la información relacionada con el evento. Esta información es muy útil para poder llevar a cabo la lógica de nuestro programa. El handler realiza tres funciones, la primera es ingresar el valor **true** en la variable conectado. Luego, obtenemos el nombre del dispositivo conectado por medio de la propiedad **Name**, que se encuentra en **e.Device**. Este nombre es asignado a la etiqueta del Phidget. Esto permitirá que la forma muestre el nombre del dispositivo cuando el programa se esté ejecutando. Por último, insertamos un color verde en el fondo del TextBox para indicar de manera visual la conexión. El código para el handler de la desconexión es:

```
// Handler de desconexion
public void iKit_DesconectarHandler(object sender, DetachEventArgs e)
{
    // Indicamos que hay desconexion
    conectado = false;

    // Quitamos mensaje
    lblPhidget0.Text = "Sin phidget";

    // Indicamos con color rojo que no hay conexión
    txtPhidget0.BackColor = System.Drawing.Color.Red;

}
```

Este handler es invocado cuando el Phidget es desconectado. Lo que hacemos primero es ingresar el valor **false** en la variable conectado; esto, con el fin de que no se realicen operaciones sobre un Phidget que ya no se encuentra conectado al sistema. La etiqueta se actualiza, e ingresamos un mensaje para indicar que no tenemos el Phidget. Por último, se inserta el color rojo en el fondo del TextBox. Es importante notar que la información del evento se encuentra en el objeto **e**, que es de tipo **DetachEventArgs**. De forma similar, tenemos el handler para el caso de que surja un evento de error.

{} PHIDGETS CONTROLADOS DE MANERA REMOTA

Es posible controlar los Phidgets en forma remota a través de Internet. Para esto, debemos hacer uso del WebService que provee el Framework de Phidgets. De esta manera, dicho servicio nos permite controlar un robot que contenga Phidgets de forma remota por Internet.

```
// Handler del error
public void iKit_ErrorHandler(object sender, ErrorEventArgs e)
{
    // Indicamos que hay desconexion
    conectado = false;

    // Mandamos mensaje de error
    lblPhidget0.Text = e.Description;

    // Indicamos con color amarillo que hay problemas
    txtPhidget0.BackColor = System.Drawing.Color.Yellow;

}
```

El esquema de trabajo es muy similar a los otros dos. Para el error, lo primero es ingresar el valor **false** en la variable conectado. La información sobre el error se encuentra en el objeto **e**. Para este handler, el objeto **e** es de tipo **ErrorEventArgs**. Si deseamos tener una descripción del error, la podemos encontrar en la propiedad **Description**, la cual usamos para ingresar la información en la etiqueta. Como se trata de un error, insertaremos el color de fondo del TextBox en amarillo.

Esto finaliza los handlers que, de alguna manera, están relacionados con la conexión del Phidget; sin embargo, tenemos otros eventos que también necesitan de un handler. Por ejemplo, si cambia una entrada digital, deseamos que nuestro programa pueda llevar a cabo un proceso en respuesta a ese cambio. El handler **iKit_DigitalHandler** será invocado cuando una entrada digital cambie de valor. El handler tiene dos parámetros, pero nos interesa el segundo, que es de tipo **InputChangeEventArgs**, ya que éste contiene la información sobre el evento. El código del handler se muestra a continuación.

```
public void iKit_DigitalHandle(object sender, InputChangeEventArgs e)
{
    // Verificamos que estemos conectados
    if (conectado)
    {
        // Verificamos que sea la entrada digital 0
        if (e.Index == 0)
        {
            // Verificamos el valor de la entrada
            if (e.Value == true)
```

```
                    chbDigital0.Checked = true;
                else
                    chbDigital0.Checked = false;
            }
        }
    }
```

Como este handler llevará a cabo procesos con el Phidget, lo primero que necesitamos hacer es verificar si se encuentra conectado o no. En el caso de que tengamos la conexión, entonces, procedemos. Dentro del objeto **e**, tenemos una propiedad llamada **Index**. Esta propiedad contiene el índice de la entrada digital que ha cambiado. Aunque en este ejemplo no es estrictamente necesario, ingresamos un if para verificar si fue la entrada digital 0 la que sufrió el cambio. Debemos recordar que, en nuestro hardware, colocamos el switch del joystick precisamente en la entrada digital 0. En el objeto **e**, tenemos la propiedad **Value**. Esta propiedad tiene el valor **true** o **false**, dependiendo del valor de la entrada digital que generó el evento. Aquí, usamos el valor para ingresar el CheckBox en el estado correspondiente.

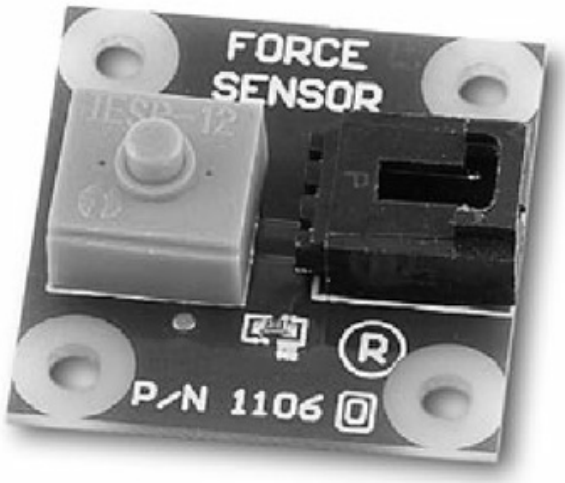


Figura 18. *El sensor que se muestra en esta fotografía permite obtener lecturas de fuerza aplicadas a él mismo.*

III EL MÁXIMO Y EL MÍNIMO DE MOVIMIENTO

Los valores máximos y mínimos de movimiento de los servos siempre pueden ser encontrados entre la documentación que provee el fabricante. En caso de no poder hallar esos datos, contamos con la posibilidad de determinar los valores a través de medios experimentales.

De manera similar, tenemos un handler para las entradas digitales, de tal forma que se invoque el handler cada vez que cambia el valor de alguna de las entradas digitales del Phidget. El código para este handler será:

```
public void iKit_AnalogoHandle(object sender, SensorChangeEventArgs e)
{
    // Verificamos que estemos conectados
    if (conectado)
    {
        // Verificamos que sea la entrada 0
        if (e.Index == 0)
        {
            // Colocamos el valor
            lblAnalogico0.Text = e.Value.ToString();
        }

        // Verificamos que sea la entrada 1

        if (e.Index == 1)
        {
            // Colocamos el valor
            lblAnalogico1.Text = e.Value.ToString();
        }

    }
}
```

La información sobre el evento se encuentra en el evento **e**, que en este caso es de tipo **SensorChangeEventArgs**. Primero, verificamos que nos encontramos conectados y, después, comprobamos cuál de las dos entradas digitales fue la que ocasionó el evento. Según cuál fue la entrada que lo hizo, se actualiza la etiqueta

III EL ÍNDICE DE OUTPUTS

Es importante notar que el índice de outputs se inicia en 0 y termina en 7. Esto lo tenemos que tener presente para evitar dar información a una salida que no corresponde o intentar dar información a una salida que no existe, como la 8. Si ingresamos un valor no válido, el programa fallará mientras se encuentre en ejecución.

correspondiente para mostrar el valor de la entrada digital. El valor de la entrada digital se encuentra en la propiedad **Value** de **e**.

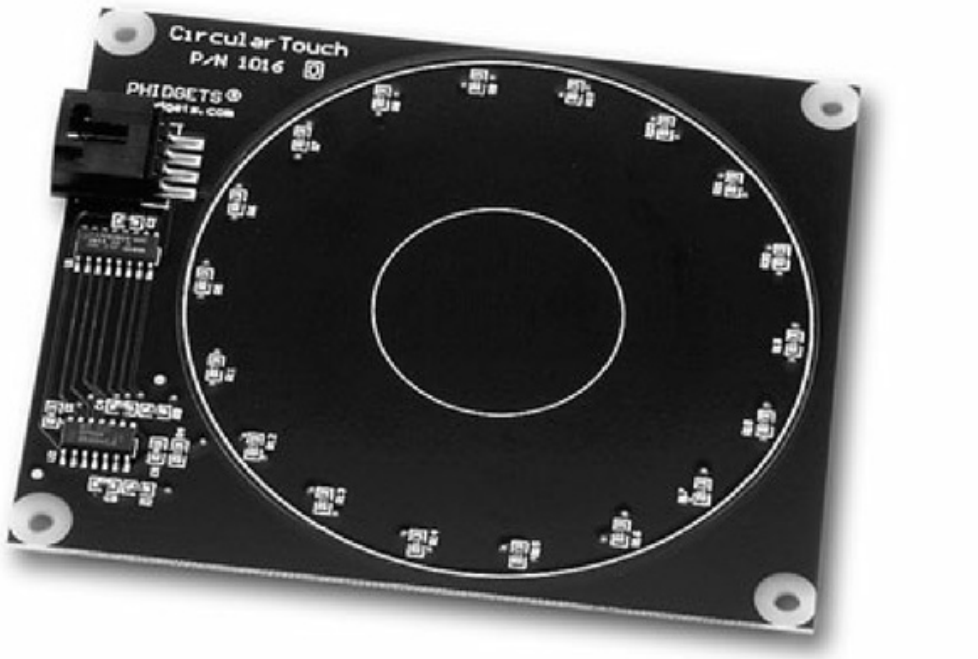


Figura 19. *Este dispositivo de apariencia extraña es un sensor de tacto. Puede determinar hasta 125 posiciones distintas en el círculo.*

Sin embargo, también podemos mandar señales al exterior por medio de las salidas digitales. Dentro de nuestra interfaz, insertamos un CheckBox que utilizaremos con este fin. Cuando el CheckBox se encuentre marcado, el LED se encenderá y, cuando no esté marcado, permanecerá apagado. Para poder llevar a cabo esta funcionalidad, creamos un handler para el evento **CheckedChanged** del CheckBox. El código queda de la siguiente manera.

```
private void chbSalida7_CheckedChanged(object sender, EventArgs e)
{
    // Verificamos que estemos conectados
    if (conectado)
    {
        // Verificamos el valor del checkbox
        if (chbSalida7.Checked == true)
            iKit.outputs[7] = true; // Prendemos el LED
        else
            iKit.outputs[7] = false; // Apagamos el LED
    }
}
```

Lo primero que hay que hacer es verificar si estamos conectados. Más adelante, estableceremos un valor a la salida de acuerdo con el estado del CheckBox. Las salidas en el Phidget se encuentran en un arreglo llamado **outputs** dentro del objeto **iKit**. El índice en el arreglo corresponde al índice de la salida a la cual le asignaremos el valor. En nuestro caso se trata del índice **7**, ya que en este caso es la salida 7 la que se encuentra conectada al LED.

Verificamos el estado del Checkbox y colocamos el valor **true** en la salida correspondiente para prender el LED o el valor **false** para apagarlo. Con esto, ya casi tenemos el programa completo, pero aún falta trabajo para terminarlo. Cuando el programa finaliza, debemos quitar los handlers que referenciamos a **iKit** y, después, cerrar el Phidget. El lugar apropiado para hacer esto es en el handler para el evento de la forma **FormClosing**.

```
        private void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {
            // Quitamos los handlers

            iKit.Attach -= new AttachEventHandler(iKit_ConectarHandler);
            iKit.Detach -= new DetachEventHandler(iKit_Desconectar
Handler);
            iKit.Error -= new ErrorEventHandler(iKit_ErrorHandler);

            iKit.InputChange -= new InputChangeEventHandler(iKit_Digital
Handle);
            iKit.SensorChange -= new SensorChangeEventHandler(iKit_Analo
goHandle);

            Application.DoEvents();

            // Cerramos el phidget
            iKit.close();

        }
```

El código no es complicado. Debemos invocar a **DoEvents()** para que todo evento de la forma, que haya quedado esperando, se lleve a cabo; luego, simplemente, cerramos el Phidget invocando al método **close()**. Ahora, es necesario incluir la biblioteca de Phidgets para poder compilar de manera correcta, la aplicación. Para esto, es necesario ir al menú **Proyecto**; luego, seleccionamos el elemento de menú

Adicionar Referencia. Entonces, aparece un cuadro de diálogo con varias pestañas. Seleccionamos la pestaña con el nombre **Browse**. Es esta pestaña, buscaremos el archivo **Phidget21.NET.dll**. Este archivo se encuentra en el directorio donde se instaló el Framework de Phidgets, por lo general, dentro de la carpeta **Archivos de Programa**. Una vez seleccionado, presionamos el botón **OK**, y la referencia se insertará en nuestro proyecto. Así, podremos compilar y ejecutar el programa.

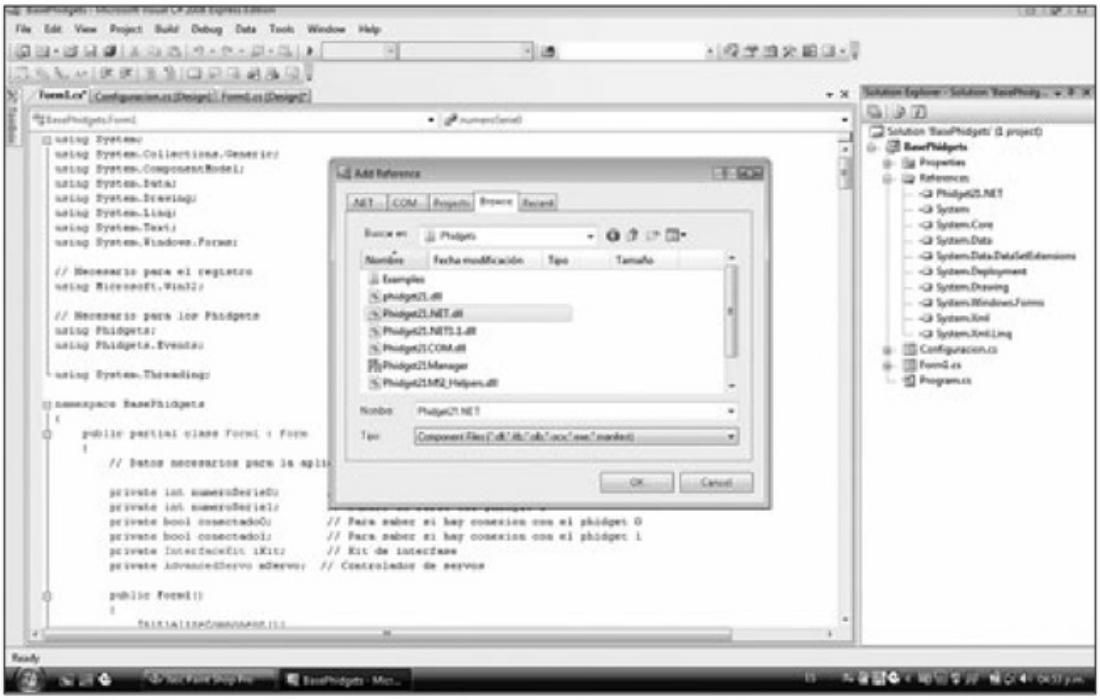


Figura 20. *Esta figura nos muestra cómo insertar la referencia para la librería de Phidgets.*

Con esto, establecimos las bases para controlar los Phidgets. En el próximo capítulo, no sólo aprenderemos más sobre otros componentes que usan los robots, sino también extenderemos el programa para poder controlar los servos.

... RESUMEN

En este capítulo aprendimos que, para poder utilizar los Phidgets, necesitamos instalar el Framework correspondiente. El panel de control de los Phidgets nos permite saber cuáles están conectados y provee de aplicaciones de control con las que los podemos probar. También, vimos que la clase que representa al 8/8/8 se conoce como InterfaceKit y pertenece al namespace de Phidgets. Los Phidgets tienen una orientación a eventos, por lo que debemos registrar los handlers que manejarán dichos eventos. El Phidget se conecta por medio del método open() y se desconecta con el método close(). El número de serie es muy importante para hacer la conexión. El objeto e de los eventos nos provee la información necesaria sobre el evento que ocurrió. Las salidas digitales se controlan con el arreglo de outputs.

ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1. ¿Por qué es importante el número de serie?
2. ¿Qué otros dispositivos podemos utilizar para una entrada analógica?
3. ¿Por qué necesitamos registrar los handlers?
4. ¿Qué tipos de eventos tenemos?
5. ¿Cómo adicionamos una referencia?
6. ¿Cómo sabemos el valor de una entrada analógica?
7. ¿Cómo agregamos el valor a una salida digital?

ACTIVIDADES

1. Utilice un timer para que el LED parpadee constantemente.
2. Adicione más LEDs a las salidas digitales.
3. Confeccione un programa que convierta un valor a sistema binario y lo muestre por medio de los LEDs.
4. Cree un programa que lea un valor en sistema binario por medio de las entradas digitales y lo muestre en la ventana.
5. Haga un programa que muestre un círculo en la ventana y que pueda ser controlado por el joystick.

Las herramientas del proyecto

Ya conocemos las bases para poder programar un Phidget. Sin embargo, el libro trata sobre robots, y eso hace necesario que conozcamos distintos componentes que pueden ser utilizados para su construcción. En este capítulo, aprenderemos sobre algunos de los componentes más utilizados en los robots.

LAS BATERÍAS

Los robots necesitan de energía para poder llevar a cabo sus actividades. Esta energía, por lo general, es provista por medio de la electricidad. Si bien es posible diseñar una fuente de poder y proveer la electricidad por medio de cables, esto resulta poco práctico. Si lo que deseamos es tener un robot con cierto grado de autonomía, es necesario pensar en otro mecanismo para proveer la electricidad. La forma más sencilla de hacerlo sería por medio de baterías, ya que éstas convierten la energía química en energía eléctrica y han sido utilizadas por muchos años. La batería se trata de un invento realizado en el año 1800 por **Alejandro Volta**.

Figura 1. *Este dibujo nos muestra a Alejandro Volta, inventor de la batería, nacido en el año 1745.*

Cuando trabajamos con baterías, es necesario conocer sus características, ya que éstas nos brindan información acerca de su comportamiento y de cómo usarlas de manera correcta. Las baterías se pueden distribuir en dos grandes clasificaciones principales: **primarias** y **secundarias**.

Baterías primarias

Las baterías primarias no pueden ser recargadas y, en su interior, incluyen productos químicos. Estos productos químicos efectúan reacciones que producen el flujo de electrones. Una vez que la batería no puede producir estas reacciones, decimos que está **exhausta**. Debido a los cambios químicos en su interior, no es posible recargarla.

Figura 2. *Ésta es una fotografía de una batería voltaica. Podemos observar los diferentes círculos de metal.*

Las baterías primarias se pueden desechar una vez que se han agotado. Cuando son nuevas, tienen su carga completa, pero ésta se pierde con el tiempo y con el uso. Estas baterías son fáciles de obtener y se pueden comprar en diferentes tipos de comercios, no necesariamente en tiendas especializadas. Podemos encontrar dos tipos de estas baterías: las de **zinc-carbón** y las **alcalinas**.

Las baterías de zinc-carbón tienen un **ánodo** construido con **zinc**. Algunos fabricantes aprovechan y confeccionan el contenedor de la batería con este mismo material. El ánodo es fabricado con **carbón** mezclado con **dióxido de manganeso**. El **cátodo** tiene la forma de un cilindro y va en el centro de la batería hasta la parte superior; se encuentra rodeado por el **electrólito**, y todo esto, contenido en un

NUNCA RECARGAR BATERÍAS PRIMARIAS

Sólo se deben recargar aquellas baterías que, en forma explícita, indiquen que es posible hacerlo. El intento de recargar baterías primarias, como las baterías alcalinas, puede terminar con la explosión de la batería, una circunstancia de mucho peligro que llega causar daño físico.

empaque metálico de zinc. El electrólito, en este tipo de baterías, se encuentra formado por **cloruro de zinc** y **cloruro de amonio**. En el siguiente diagrama, podemos observar cómo está formada la batería.

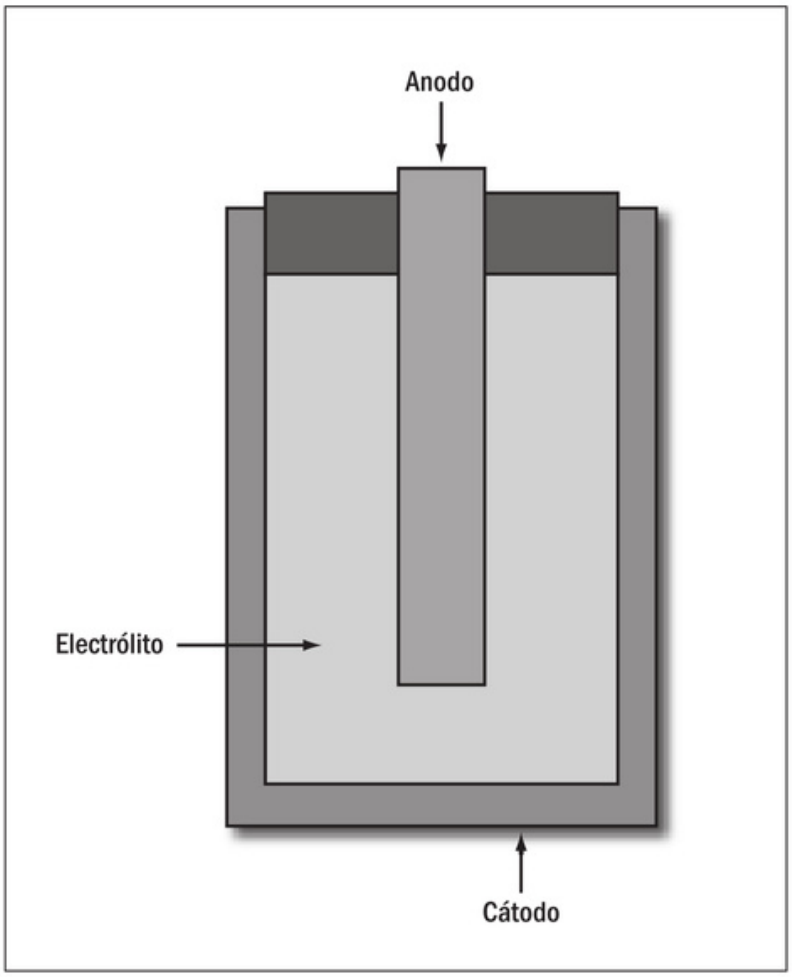


Figura 3. *Este diagrama nos muestra la construcción típica de una batería primaria.*

Las baterías alcalinas tienen mayor densidad de energía que las de zinc-carbón; también, conservan su carga por mayor tiempo cuando no están siendo utilizadas. Su nombre lo reciben debido al electrólito de tipo **alcalino** que usan. Por lo general, el electrólito es **hidróxido de potasio**.

Baterías secundarias

En las baterías secundarias, también existen productos químicos que, al reaccionar, producen la corriente eléctrica. Pero estas reacciones químicas pueden restaurarse al aplicárseles, a su vez, una corriente eléctrica; de esta manera, la batería es recargada. Estas baterías, por lo general, salen descargadas de fábrica, y es necesario cargarlas en forma completa antes de utilizarlas. La más común de ellas es la de **plomo-ácido**, a la que encontramos con frecuencia en los vehículos automotores. Estas baterías pueden

producir una buena corriente, aunque tienden a ser pesadas. Para muchas aplicaciones de robots autónomos grandes, resultan una buena opción. En estas baterías, el electrólito es líquido, pero se pueden obtener algunas que lo contengan en forma de gel.

Figura 4. *Aquí observamos baterías de diferentes tipos y tamaños.*

Existen otros tipos de baterías secundarias en las cuales aparecen diferentes electrólitos y son más pequeñas. Las podemos encontrar en los teléfonos celulares, PDAs y dispositivos de MP3. Uno de estos tipos es la batería de **níquel-cadmio**, que utiliza el **cadmio** como electrodos. Las baterías **NiCd** proveen un voltaje menor que sus similares primarias por lo que, a veces, podemos tener una baja en el desempeño de los equipos diseñados para usar baterías primarias; sin embargo, son buenas para proveer una gran cantidad de corriente cuando los motores eléctricos son arrancados. Otro tipo de batería secundaria es la de **Litio-Ion**, muy popular en la actualidad. En ésta se aprovecha el movimiento de los iones de **litio** entre el ánodo y el cátodo para proveer energía y, cuando se recargan, los iones viajan de regreso al ánodo. Una de las grandes ventajas que proveen estas baterías es que no tienen efecto de memoria y se descargan muy lentamente cuando no se usan.

III LAS BATERÍAS NICD

Existen dos tipos de baterías de níquel-cadmio: las selladas y las ventiladas. Resulta más sencillo obtener las versiones selladas, incluso en los tamaños comerciales, como AA y AAA. Debemos tener en cuenta que el recargador de baterías funcione con este tipo de versiones en particular.

Características de las baterías

Es fundamental que podamos seleccionar, de manera correcta, el tipo de baterías que necesitará el robot. No sólo es necesario conocer los tipos y sus propiedades más importantes, también necesitamos saber acerca del voltaje y de la corriente que la batería puede proveer. Cuando vemos las especificaciones de las baterías, encontramos indicado un voltaje. Este voltaje, casi siempre, representa el voltaje que la batería puede proveer durante su vida útil. Si la batería es recargable, puede que encontremos otro voltaje indicado: éste es el que corresponde para la recarga. En el mercado, se pueden encontrar cargadores especiales que, incluso, poseen protección contra sobrecarga. Es posible construir un cargador propio, pero solamente si tenemos los conocimientos apropiados de electrónica.

A medida que se usan, las baterías van perdiendo su carga, y la cantidad de voltaje que pueden proveer disminuye con el tiempo. Una batería se considera **útil** mientas pueda proveer más del ochenta por ciento del voltaje marcado; en el momento en que la batería provee menos que esta cantidad, se considera que ya no es útil. Por ejemplo, si la batería que tenemos está marcada con un voltaje de 12 V, con el tiempo este voltaje disminuirá. Cuando comienza a proveer menos de 9.6 V, podemos considerar que la batería terminó su vida útil. Nuestros circuitos deberán poder trabajar en este rango de voltajes, pero no hay que forzarlos a hacerlo con voltajes para los que no fueron diseñados. Para saber el voltaje que tienen nuestras baterías, podemos usar un voltímetro o instalar un medidor de voltaje en el mismo robot. Puede que éste nos dé una lectura directa de los valores o que la lógica del robot lo lleve automáticamente a recargarse cuando el voltaje es bajo. Otra característica de las baterías que podemos encontrar en sus especificaciones es la cantidad de corriente que ella puede proveer, y que se encuentra medida en **amperes-hora**. En algunas baterías, la medida puede ser **microamperes-hora**, pero la idea es la misma. Para entender esto, veamos el siguiente ejemplo. Supongamos que tenemos una batería que está marcada con 10 amperes-hora. Esto significa que la batería nos da una corriente de 10 amperes por el período de una hora, pero también puede darnos 1 ampere de corriente por 10 horas o cualquier otra combinación. De esta forma, es posible calcular el tamaño de la batería que necesitamos para nuestra

{} EL VOLTAJE DE CARGA

Las baterías recargables necesitan de un voltaje de carga que es mayor a su voltaje de trabajo. Por lo general, este voltaje de carga es de un 20% a un 30% mayor que el mencionado voltaje de trabajo. Debemos prestar atención, ya que este voltaje varíará dependiendo del tipo y diseño de la batería.

aplicación robótica. Es conveniente tener baterías que puedan proveer entre un 20% y un 40% más de amperes-hora de los que realmente necesita el robot. Esto es necesario, ya que algunos componentes, en especial los motores, consumen de forma repentina mucha corriente cuando son encendidos. Un motor que consume medio ampere cuando se encuentra trabajando, en el momento de ser encendido, puede consumir hasta 2 amperes durante algunos milisegundos Esto puede agotar, de forma inesperada, nuestra batería.

LOS MOTORES ELÉCTRICOS

Los **motores eléctricos** sirven para producir movimiento a través de una corriente eléctrica. En las aplicaciones con robots, se utilizan mucho, por lo que es necesario conocerlos. Existen diferentes tipos de motores, y todos tienen algunas características que necesitamos aprender.

***Figura 5.** Éste es un motor eléctrico clásico que se utiliza en proyectos de hobby.*

Cuando una corriente eléctrica pasa por un **conductor**, se produce también un **campo magnético**. Este campo magnético puede interactuar con otros campos magnéticos, como los de los imanes permanentes. Podemos enrollar el hilo conductor alrededor de

EVITEMOS DAÑAR NUESTRAS BATERÍAS

Si la batería se ve forzada a descargarse a un ritmo más grande que el que se encuentra marcado en sus especificaciones, esto puede dañarla. Cierto tipo de baterías recargables, como las NiCd, son particularmente sensibles a este problema. El hecho de descargarlas de manera adecuada aumenta su vida útil, y nos ayuda a ahorrar en gastos.

un núcleo de hierro y darle muchas vueltas, lo cual hará que se concentre el campo electromagnético. Si aproximamos un imán a nuestro conductor enrollado, cuando pasemos una corriente por él, se producirá un campo que interactuará con el del imán. Si los campos son de polaridades iguales, se repelerán y, si tienen polaridades diferentes, se atraerán. En caso de que los campos sean lo suficientemente grandes, incluso, obtendremos movimiento por parte de los componentes debido a las fuerzas electromagnéticas. Éste es el principio en el cual se basa el motor eléctrico.

Típicamente, los motores eléctricos se dividen en dos categorías: los de **corriente alterna** y los de **corriente directa**. Aunque podemos utilizar ambos en proyectos robóticos, es muy común que, en los robots autónomos alimentados con baterías, utilicemos los motores de corriente directa. A los motores de corriente directa, también los llamamos motores **DC** por sus siglas en inglés que significan **direct current**. Los motores DC, a su vez, se pueden clasificar en motores **continuos** o motores **paso a paso**. Los motores continuos son más comunes; cuando les aplicamos una corriente, giran de manera constante. Este tipo de motores los encontramos, por ejemplo, en los ventiladores o en los automóviles eléctricos de juguete. Cuando la corriente eléctrica se interrumpe, el motor se detiene.

***Figura 6.** Este modelo de motor eléctrico tiene varios caballos de fuerza y podría servir para mover un robot grande.*

III EL COMIENZO DE LOS MOTORES ELÉCTRICOS

El primer motor eléctrico de la historia fue construido por **Faraday** en 1821. Este motor no podía realizar ningún trabajo útil, sin embargo, mostraba el concepto de transformar la energía eléctrica en mecánica. Poco a poco, los motores eléctricos fueron evolucionando hasta llegar a los de alta eficiencia de estos días.

Los motores paso a paso son diferentes. En este caso, cuando aplicamos una corriente eléctrica, el motor gira, pero solamente unos cuantos grados, y luego se detiene. Si queremos que el motor tenga un movimiento continuo, debemos proveerle una corriente pulsante para poder lograrlo. Este tipo de motor es utilizado cuando necesitamos tener un control más preciso sobre el movimiento del robot.

Especificaciones de los motores

Existen algunas especificaciones de los motores que debemos conocer para poder seleccionar la clase que más conviene a nuestro proyecto robótico. El primer parámetro es el **voltaje**. Los motores tienen un voltaje de operación y es el voltaje en el cual pueden proveer la mayor potencia sin sobrecalentarse. Algunos motores tienen un rango en lugar de un voltaje en particular. Por ejemplo, podemos tener un motor que trabaje en el rango de 8 V a 12 V.

Si el motor se alimenta con un voltaje menor al que se especifica, es posible que trabaje, pero con su potencia disminuida. Al contrario, si se lo alimenta con un voltaje mayor, entonces el motor puede proveer más potencia, pero es posible que se sobrecaliente. Si esto llegara a suceder, lo más seguro es que resulte dañado después de un tiempo. El motor también tendrá una especificación de **corriente**. Los motores necesitan más corriente cuanto mayor sea la carga que deben mover. Nuestras baterías tienen que proveer la corriente necesaria para llevar a cabo el funcionamiento del robot. Si el motor no tiene una carga, consume poca corriente. Los motores tienen una corriente máxima que pueden tolerar; si excedemos esta corriente, el motor se detendrá.

***Figura 7.** Este modelo de motor también se utiliza en los proyectos robóticos ya que incluye una caja con engranes de reducción.*

El siguiente parámetro con el que necesitamos trabajar es la **velocidad**. Dicha velocidad, en realidad, es velocidad rotacional y se mide en **revoluciones por minuto**,

también conocida como **rpm**, por sus siglas. Los motores, por lo general, tienen velocidades altas. Estas velocidades son poco prácticas para aplicaciones robóticas, por lo que necesitamos crear un sistema de engranes para poder reducir las revoluciones por minuto a un rango útil. También, es posible disminuir las revoluciones por minuto por medios electrónicos, pero no siempre es conveniente.

La **torca** es otro de los parámetros que necesitamos conocer del motor y consiste en la fuerza que el motor imprime sobre su carga. Los motores que tienen una torca grande pueden mover cargas grandes. Conocer la torca del motor nos permite establecer si éste será capaz de mover al robot o no. La torca se mide de una forma que puede parecer extraña. Imaginemos que colocamos una palanca de determinada longitud al eje del motor. En el otro extremo de la palanca, se ubica un peso. La torca del motor se medirá con el peso más grande que el motor puede mover para la longitud de dicha palanca. Si tenemos un motor con una torca de 12 libras-pie, esto significa que puede mover doce libras de peso con una palanca de longitud de un pie. Es lo que podemos ver en la siguiente figura.

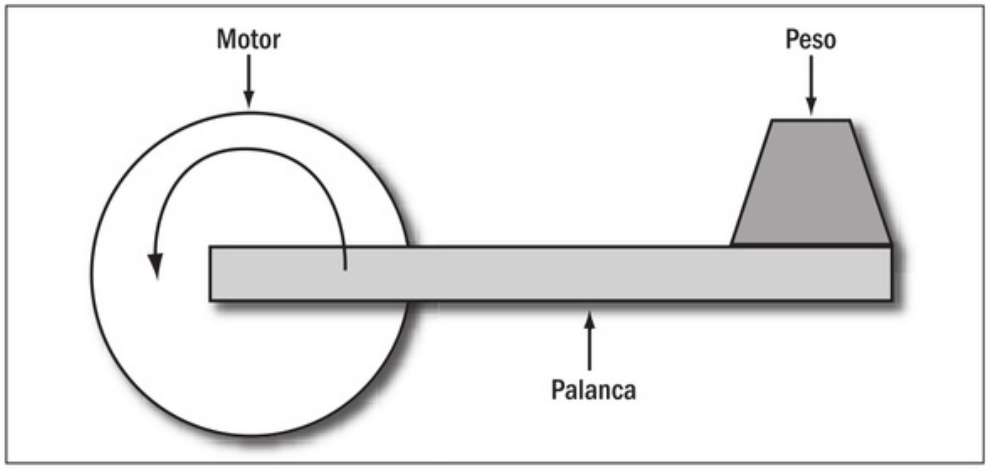


***Figura 8**. Este diagrama nos muestra la forma como se mide la torca de un motor.*

La torca se puede medir en diferentes unidades, por ejemplo: onzas-pulgada, libra-pie, gramo-centímetro, etcétera. Es conveniente tener cerca una tabla que nos permita hacer la conversión entre las diferentes medidas de la torca.

CONVERTIDOR DE TORCA

El sitio web **Bolt Science** (**www.boltscience.com/pages/convert.htm**) tiene una calculadora que se puede utilizar para realizar la conversión entre las unidades de torca más comunes. Es conveniente tenerlo entre nuestros sitios favoritos, pues maneja diferentes unidades, y esta característica nos facilitará la tarea de hacer las transformaciones.

LOS ENGRANES

Los **engranes** son de mucha utilidad, ya que nos permitirán realizar diferentes funciones, que se aprovecharán en distintos lugares y tienen variadas aplicaciones. Lo primero que podemos hacer con los engranes es la transferencia de fuerzas; con ellos, podemos llevar la fuerza producida por el motor hacia algún otro lugar. Para la transferencia de fuerza, también podemos usar algún otro medio de transmisión, como las bandas o las cadenas. Con los engranes, además, es posible modificar la velocidad. Esto es muy útil, en especial, en nuestras aplicaciones robóticas. Los engranes también nos permiten invertir la dirección de giro. Por último, hacer uso de los engranes nos posibilitará modificar la torca, lo cual resulta muy útil si el motor no tiene una torca lo suficientemente grande para nuestro proyecto.

Figura 9. *Esta fotografía nos da un ejemplo de algunos engranes. Los dientes de los engranes pueden tener diferentes diseños.*

Veamos un ejemplo para entender un poco más cómo funcionan los engranes. Imaginemos que tenemos un engrane que posee 20 dientes y está conectado de manera directa al eje de nuestro motor. También tenemos otro engrane, pero en

III LA TORCA EN LOS ENGRANES

En el ejemplo que hemos visto, la torca del engrane mayor es el doble de la del pequeño. Sin embargo, en el mundo real, la torca que obtenemos es ligeramente menor. Esto se debe a que la fricción del sistema nos produce pérdidas. Reducir la fricción ayuda a que la transferencia sea más eficiente, por eso, muchos sistemas de engranes usan grasas.

este caso de 40 dientes, que está en contacto con el primero. Esto lo podemos apreciar en el diagrama que aparece a continuación.

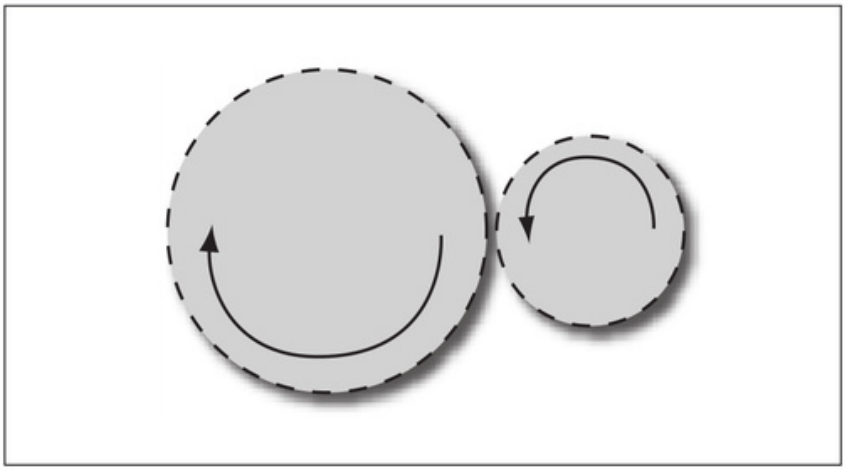


Figura 10. *Aquí vemos unos engranes. El pequeño está conectado al motor y, al girar, transfiere el movimiento al engrane grande.*

Ahora, imaginemos qué es lo que está sucediendo. El segundo engrane tiene el doble del tamaño del primero. Si encendemos el motor y el primer engrane gira una vuelta completa, el segundo engrane gira solamente media vuelta. Es decir, llevamos a cabo una reducción en las revoluciones por minuto. Esta reducción se conoce como de 2:1. Podemos ver entonces que, cambiando los engranes, obtenemos la reducción de revoluciones por minuto que deseamos. Si el motor que usamos gira a 2.500 revoluciones por minuto, el engrane más grande girará a 1.250 revoluciones por minuto.

Los engranes funcionan como pequeñas palancas circulares, por lo que modificarán también la torca. En nuestro ejemplo, el segundo engrane tendrá el doble de la torca del primero. Es importante notar que el aumento de la torca va acompañado de un decremento en las revoluciones por minuto. Si ahora conectamos el engrane más grande al eje del motor, veremos que, en una vuelta del engrane grande, el pequeño efectuó dos. Es decir que ahora estamos incrementando las revoluciones por minuto.

III DESECHAR LAS BATERÍAS

Las baterías contienen químicos que pueden dañar el medio ambiente, por lo que es necesario enviarlas a un centro especializado de reciclaje para este producto. La mayoría de los países ya cuentan con este tipo de instalaciones, y se puede encontrar información sobre ellas en Internet.

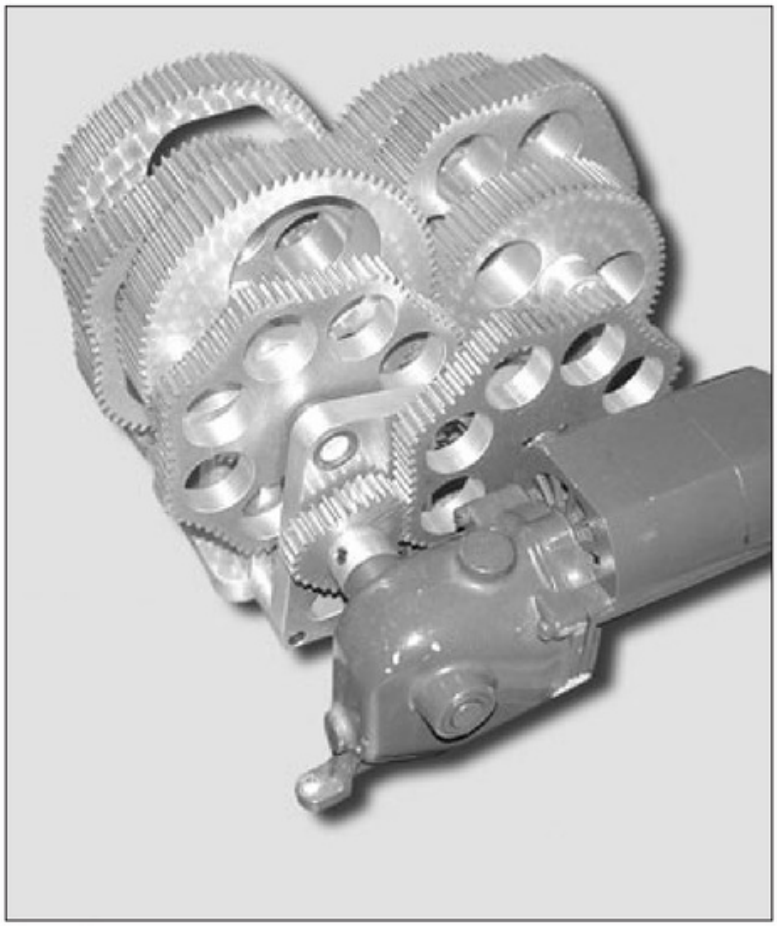


Figura 11. *Éste es un gran ejemplo de una caja de engranes donde tenemos diferentes modelos de engranes.*

A partir de esto, podemos deducir dos reglas para los engranes. La primera es que la velocidad disminuye y la torca aumenta cuando la transmisión va de un engrane menor a uno mayor. La segunda regla es su complemento: la velocidad aumenta y la torca disminuye cuando la transmisión va de un engrane mayor a uno menor. A veces, nos podemos encontrar con el problema de tener que reducir drásticamente la velocidad del motor. Por ejemplo, tenemos un motor de 10.000 revoluciones por minuto y necesitamos una velocidad de 100 revoluciones por minuto. En este caso, sería poco práctico si poseemos un engrane cien veces mayor que otro. Para poder resolverlo, hacemos uso de un sistema compuesto de varios engranes. De esta

SEGURIDAD CON LAS BATERÍAS DE PLOMO-ÁCIDO

Este tipo de baterías producen hidrógeno cuando son recargadas, por lo que es necesario hacerlo en sitios bien ventilados y evitar chispas eléctricas. También hay que tener cuidado de no voltearlas o derramar el líquido que contienen en su interior, ya que es ácido.

manera, reducimos en forma drástica el tamaño de los engranes necesarios. Un ejemplo de esto lo podemos observar en la siguiente figura.

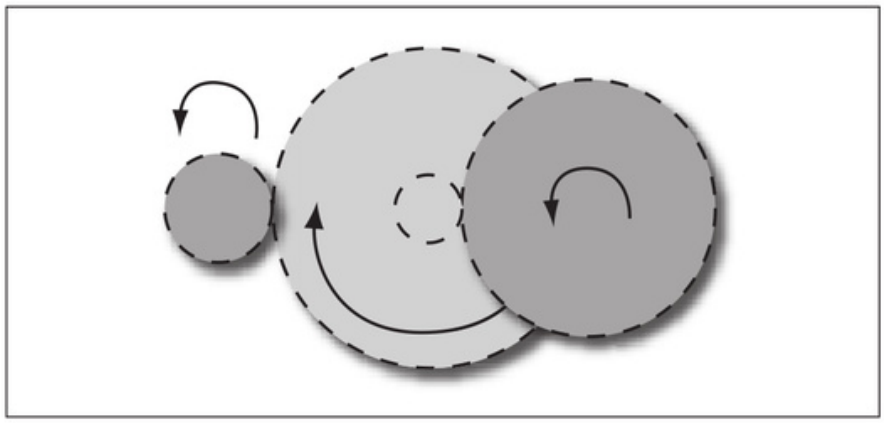


Figura 12. *Esta figura nos muestra un sistema que usa varios engranes para reducir la velocidad de un motor muy rápido.*

LOS SERVOS

Un **servo** es un dispositivo al cual le podemos controlar su posición. La forma como se controla esta posición es por medio de un sistema de **retroalimentación**. Los servos son muy útiles en las aplicaciones robóticas y, en este libro, serán nuestra forma principal para proveer movimiento a los proyectos. Los servos que utilizaremos son eléctricos, pero existen otros que usan diferentes sistemas, como los hidráulicos, neumáticos o magnéticos. El tipo de servo que utilizamos tiene su origen en los modelos a escala controlados remotamente. Esto se debe a que son baratos, proveen una buena torca, y los podemos conectar sin problemas a los Phidgets.

El servo tiene en su interior un motor eléctrico, el cual se encarga de proveer el movimiento. El motor está conectado a un sistema de engranes para reducir su velocidad y aumentar la torca. A este sistema de engranes, también está conectado un potenciómetro. Podemos observar el interior del servo en la siguiente figura.

III NO SE PUEDE RECARGAR ETERNAMENTE

Las baterías secundarias no pueden recargarse infinitas veces, ya que en su interior puede ocurrir una corrosión o pérdida de electrólito que impide recargarla de forma óptima. Cuando notemos que la batería nos muestra un 80% de la carga total que tenía cuando era nueva, es necesario pensar en reemplazarla de manera inmediata.

Figura 13. *Esta figura nos muestra el motor, el sistema de engranes con el potenciómetro debajo y la caja del servo.*

El valor del potenciómetro variará de acuerdo con la cantidad de vueltas que efectúe el motor. Esto nos permite calcular la rotación según el valor del potenciómetro. Para indicar una posición en la que queremos ubicar al servo, se manda una señal. Al comparar esta señal y la que se genera con el potenciómetro como referencia, sabemos en qué dirección debe girar el motor hasta que llegue a la posición deseada.

En el servo, también es posible establecer una velocidad de giro determinada, por lo que podemos tener un control más preciso en nuestras aplicaciones. Los servos tienen un **arco de acción**, es decir, sólo se pueden mover una determinada cantidad de grados. Es importante conocer el rango del movimiento del servo para evitar cualquier intento de ir más allá de sus límites ya que, si tratamos constantemente de exceder estos límites, el mecanismo se puede dañar.

{} MODIFICACIÓN DE LOS SERVOS

Aunque no es recomendable, los servos se pueden modificar para convertirlos a rotación continua. De esta forma, obtenemos un motor al cual le podemos controlar, de forma precisa, su rotación y velocidad. Es necesario experimentar con ellos para poder determinar cómo funcionaría exactamente con los Phidgets.

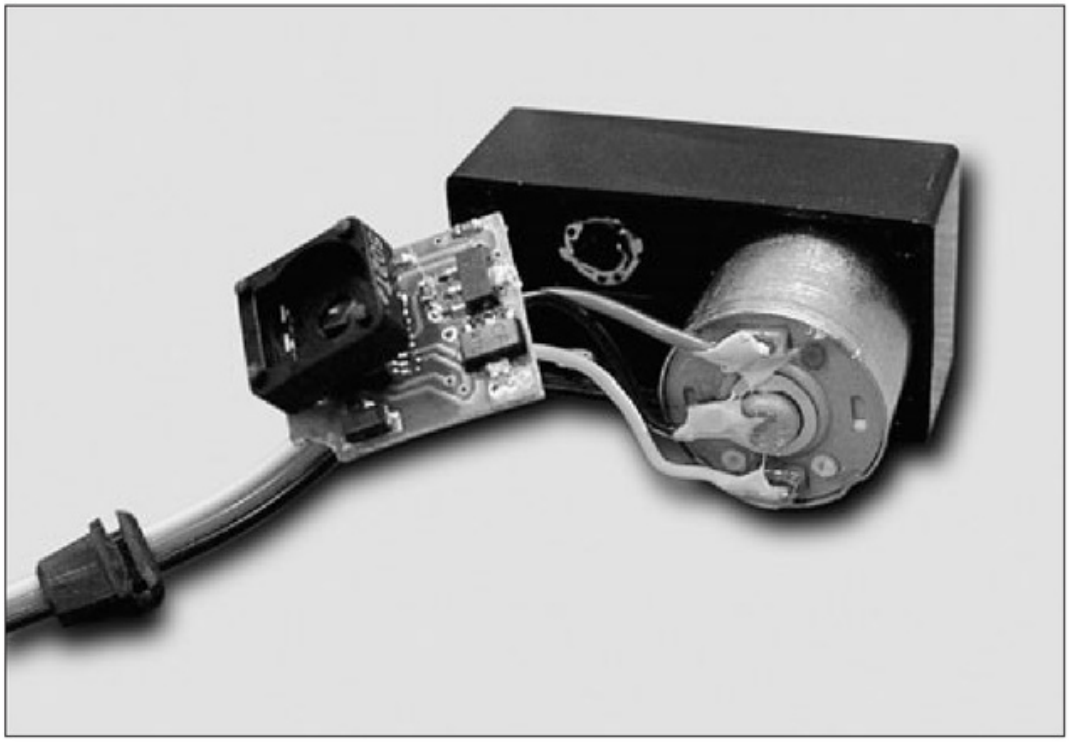


Figura 14. *En el interior de los servos se encuentra un motor eléctrico que lleva a cabo el movimiento.*

CREACIÓN DEL PROGRAMA

Para crear este programa, nos basaremos en el programa del capítulo anterior, pero llevaremos a cabo unos cambios que nos permitirán agregar el control al servo. Habrá algunas variantes, pero nos concentraremos más en los agregados.

Lo primero que hacemos es modificar la interfaz gráfica de usuario para colocar los controles que son necesarios en el manejo del Phidget encargado de manipular al servo. La interfaz deberá lucir tal y como aparece en la siguiente figura.

III PROTEGER LA APLICACIÓN BASE

Si ya tenemos la aplicación base totalmente funcionando, es mejor trabajar sobre una copia del proyecto. Esto último con el fin de evitar dañarla mientras adicionamos el código nuevo. Cada vez que necesitemos crear un proyecto, usamos una copia nueva de esta aplicación.

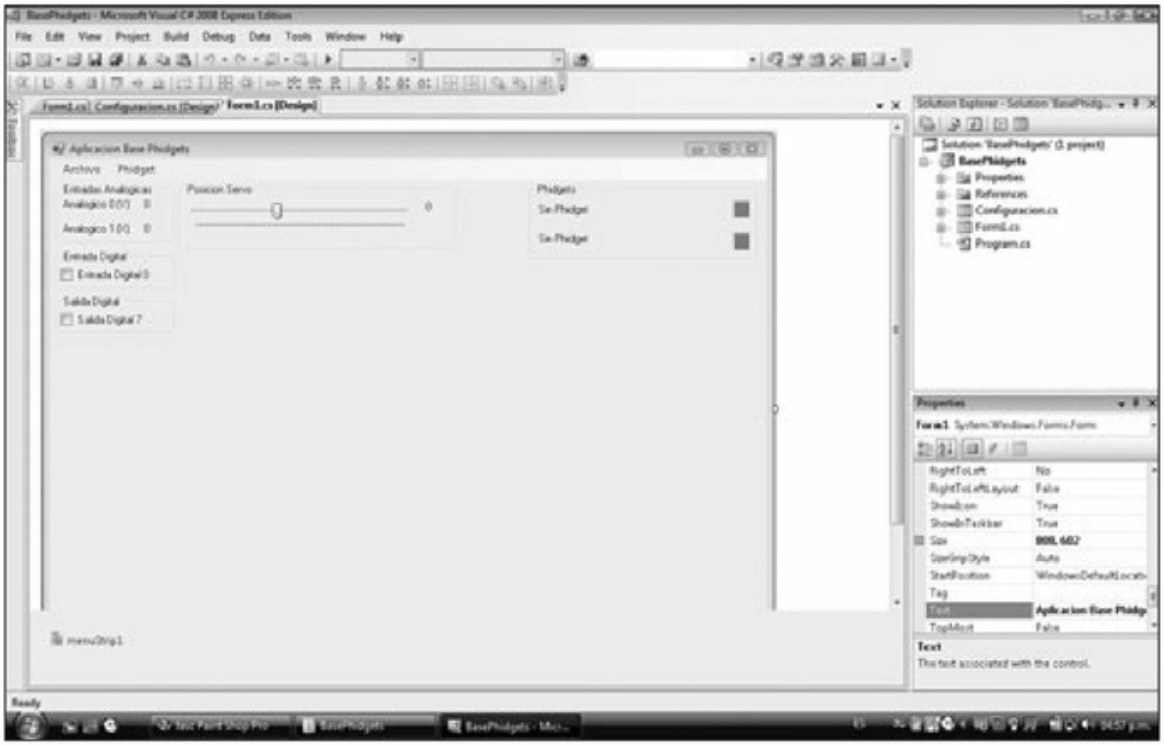


Figura 15. *Aquí podemos observar cómo debe lucir la interfaz de usuario con los controles para el servo adicionados.*

Dentro del **GroupBox** de Phidgets, necesitamos adicionar una nueva etiqueta. Esta etiqueta debe tener por nombre **lblPhidget1**, y, en su propiedad **Text**, debemos ingresar el valor **"Sin Phidget"**. Con esta etiqueta, conoceremos el estado de conexión del segundo Phidget, en nuestro caso, el controlador de los servos. Continuando dentro del mismo **GroupBox**, insertamos un **TextBox** que recibirá el nombre **txtPhidget1**. Al igual que en el otro **TextBox**, debemos insertar su color de fondo en rojo y dejar la propiedad **Text** vacía; también será necesario que conservemos el mismo tamaño que teníamos hasta ahora.

Para controlar al servo, usaremos un **TrackBar**. Primero, creamos un **GroupBox** para contenerlo. Este **GroupBox** tiene en su propiedad **Text** el valor **"Posición Servo"**. En su interior, colocamos el **TrackBar**, que llevará el nombre **trkPosicion**. En la propiedad **Maximum**, colocamos el valor **210** y, en la propiedad **Minimum**, el valor **30**. Éste es el rango de valores que puede tener nuestro servo. Hay que tener en cuenta que estos valores pueden variar dependiendo del modelo de servo que utilicemos. El tamaño del control debe ser de **256** por **45**; esto con el fin de obtener un tamaño que sea fácil de manipular. El valor inicial que escribiremos en el **TrackBar** es de **100**, por lo que insertamos este valor en su propiedad **Value**. Para poder saber cuál es el valor actual del **Trackbar**, insertaremos una etiqueta en su lado derecho. Esta etiqueta lleva el nombre **lblPosicion**,y, en su propiedad **Text**, colocamos el valor **"0"**. Cuando movamos el **TrackBar**, el contenido mostrado en la etiqueta se modificará para enseñarnos el valor de la posición actual.

Al menú de la forma, adicionamos un nuevo menú para configurar al segundo Phidget. Este menú se llamará **ConfigurarPhidget1** y tiene, en su propiedad **Text**, el valor **"Configurar Phidget1"**. Lo insertaremos dentro del menú principal Phidget. Ahora, podemos empezar a ver el código del programa. Gran parte de este código es similar al del programa anterior, pero tiene ciertos cambios y adiciones. Las partes que ya conocemos no serán comentadas, pero sí es necesario que estemos atentos a cualquier cambio. Lo primero que debemos hacer es usar los namespace que son básicos para la aplicación.

```
// Necesario para el registro
using Microsoft.Win32;

// Necesario para los Phidgets
using Phidgets;
using Phidgets.Events;

using System.Threading;
```

Ahora, dentro de la clase de la forma, colocamos los siguientes datos con acceso privado.

```
        // Datos necesarios para la aplicacion base

        private int numeroSerie0;        // Numero de serie del phidget 0
        private int numeroSerie1;        // Numero de serie del phidget 1
        private bool conectado0;         // Para saber si hay conexion con
el phidget 0
        private bool conectado1;         // Para saber si hay conexion con
el phidget 1
        private InterfaceKit iKit;       // Kit de interfaz
        private AdvancedServo mServo;   // Controlador de servos
```

Podemos observar que, prácticamente, hemos duplicado las variables necesarias para, de esta forma, controlar a dos Phidgets. Al final, creamos una variable llamada **mServo** que es de tipo **AdvancedServo**. Esta clase tiene todo lo necesario para trabajar con el Phidget que controla a los servos.

Dentro del constructor de la forma, llevaremos a cabo la inicialización. Esta inicialización es similar a la del capítulo anterior, pero estamos tomando en cuenta que se guardará en el registro la información del número de serie para dos Phidgets.

Algunas de las llaves del registro cambiaron su nombre, con el fin de poder diferenciarlas a las de la aplicación base original.

```
        public Form1()
        {
            InitializeComponent();

            // Inicializamos la aplicacion

            // Empezamos como si no estuvieramos conectados
            conectado0 = false;
            conectado1 = false;

            // Colocamos un valor de default para el numero de serie
            numeroSerie0 = 0;
            numeroSerie1 = 0;

            // Verificamos si existe la llave en el registro
            RegistryKey sfwKey = Registry.LocalMachine.OpenSubKey
("Software");
            RegistryKey redKey = sfwKey.OpenSubKey("RedUsers2");

            // Verificamos si existe
            if (redKey == null) // la llave no existe
            {

                // Creamos la llave
                sfwKey = Registry.LocalMachine.OpenSubKey("Software",
true);
                redKey = sfwKey.CreateSubKey("RedUsers2");

                RegistryKey robotKey0 = redKey.CreateSubKey("Robots2");

                // Colocamos un valor de default
                robotKey0.SetValue("NumeroSerie0", (object)numeroSerie0);
                robotKey0.SetValue("NumeroSerie1", (object)numeroSerie1);

            }
            else // la llave existe
            {
```

```
                RegistryKey robotKey0 = redKey.OpenSubKey("Robots2");

                // Obtenemos el numero de serie guardado
                numeroSerie0 = (int)robotKey0.GetValue("NumeroSerie0");
                numeroSerie1 = (int)robotKey0.GetValue("NumeroSerie1");
            }

        }
```

Observamos el handler para el evento **Load** modificado. En la primera parte del código del handler, tenemos la inicialización para el objeto **InterfaceKit**, la cual no lleva ningún cambio. La segunda parte se refiere a la inicialización del objeto **AdvancedServo**, que comentaremos a fondo más adelante.

```
        private void Form1_Load(object sender, EventArgs e)
        {
            // Inicializamos el objeto InterfaceKit
            try
            {
                // Codigo para el kit de interfaz

                // Instanciamos el objeto
                iKit = new InterfaceKit();

                // Colocamos los handlers relacionados con la conexion
                iKit.Attach += new AttachEventHandler(iKit_Conectar
Handler);
                iKit.Detach += new DetachEventHandler(iKit_Desconectar
Handler);

                // Colocamos el handler relacionado con el error
                iKit.Error += new ErrorEventHandler(iKit_ErrorHandler);

                // Colocamos el handler para las entradas digitales
                iKit.InputChange += new InputChangeEventHandler
(iKit_DigitalHandle);

                // Colocamos el handler para las entradas analogicas
                iKit.SensorChange +=
```

```
new SensorChangeEventHandler(iKit_AnalogoHandle);

                // Abrimos para conexiones
                iKit.open(numeroSerie0);

            } // Fin de try
            catch (PhidgetException ex)
            {
                // Enviamos mensaje con el error
                lblPhidget0.Text = ex.Description;

                // Indicamos con amarillo que hay problemas
                txtPhidget0.BackColor = System.Drawing.Color.Yellow;
            }


            try
            {
                // Codigo para el controlador de servos

                // Instanciamos el objeto
                mServo = new AdvancedServo();

                // Colocamos los handlers relacionados con la conexion
                mServo.Attach += new AttachEventHandler(mServo_Conectar
Handler);
                mServo.Detach += new DetachEventHandler
(mServo_DesconectarHandler);

                // Colocamos el handler relacionado con el error
                mServo.Error += new ErrorEventHandler(mServo_Error
Handler);

                // Abrimos para conexiones
                mServo.open(numeroSerie1);

                if (mServo.Attached == true)
                {
                    // Inicializamos el track
                    trkPosicion.Minimum = 30;
```

```
                trkPosicion.Maximum = 210;
                trkPosicion.Value = 100;

                // Colocamos los limites de posicion del servo
                mServo.servos[0].PositionMin = 30;
                mServo.servos[0].PositionMax = 210;
            }
        }
        catch (PhidgetException ex)
        {
            // Enviamos mensaje con el error
            lblPhidget1.Text = ex.Description;

            // Indicamos con amarillo que hay problemas
            txtPhidget1.BackColor = System.Drawing.Color.Yellow;
        }

    }
```

Como el código para iniciar el objeto **AdvancedServo** puede producir problemas, vamos a usar el try/catch. Lo primero que se lleva a cabo es la instanciación del objeto. Luego, se colocan los handlers para cuando se conecta y desconecta el Phidget. Éstos no son los mismos que en el Phidget anterior, por lo que tenemos que codificar sus propios handlers. Con posterioridad, se coloca el handler para el evento de error, también es propio para este Phidget en particular. A continuación, se abre el Phidget mediante el método **open()**, y pasamos como parámetro su número de serie. Hay que tener cuidado de no mezclar los números de serie de los diferentes Phidgets.

Para poder inicializar de manera correcta el controlador de servos, tenemos que asegurarnos de que se haya creado la conexión. Ahora, ingresamos los valores de inicialización para el **TrackBar** y el servo. Haciendo uso de **trkPosicion**, escribimos

III LOS LÍMITES DEL SERVO

Al hacer uso de las propiedades **PositionMin** y **PositionMax**, estamos protegiendo al servo contra excesos en su movimiento, que pueden dañar el mecanismo. Es posible que los valores se obtengan de forma experimental, para ser ingresados posteriormente en nuestro código.

en la propiedad **Minimum** el valor **30**; en la propiedad **Maximum**, el valor **210**; y en la propiedad **Value**, el valor **100**. Con estos valores, estamos indicando el rango en el cual el **TrackBar** puede moverse y su posición inicial.

El controlador de servos puede manejar hasta un máximo de 8 servos en total, por lo que necesitamos de un índice para indicar cuál es el servo que nos interesa. En esta aplicación, controlaremos al servo con el índice **0**. El objeto **mServo** tiene en su interior un arreglo llamado **servos**. Por medio de este arreglo, accedemos a los servos que deseamos controlar. El índice del arreglo nos sirve para indicar a cuál servo le estamos otorgando alguna propiedad en particular. Como hacemos uso del servo 0, entonces, trabajamos en el arreglo de la forma **servos[0]**. La propiedad de **PositionMin** indica la posición más pequeña que puede tener el servo, que en nuestro caso es **30**.En la propiedad **PositionMax**, ingresamos el límite máximo de movimiento del servo, que, en este caso es **210**.

Si hubiera algún problema en la inicialización, entonces se ejecuta el código que se encuentra dentro del **catch()**. Se recibe una excepción de tipo **PhidgetException** y, por medio de ésta, recibimos la información del error, el cual es desplegado en la etiqueta y, luego, aparece el color del fondo del **TextBox** como amarillo.

En el evento **FormClosing**, debemos quitar los handlers y, por seguridad, vamos a colocar el servo en una posición centrada.

```
        private void Form1_FormClosing(object sender,
FormClosingEventArgs e)
        {
            // Quitamos los handlers
            iKit.Attach -= new AttachEventHandler(iKit_ConectarHandler);
            iKit.Detach -= new DetachEventHandler(iKit_Desconectar
Handler);
            iKit.Error -= new ErrorEventHandler(iKit_ErrorHandler);

            iKit.InputChange -= new InputChangeEventHandler(iKit_Digital
Handle);
            iKit.SensorChange -= new SensorChangeEventHandler
(iKit_AnalogoHandle);

            // Regresamos el servo al centro
            if (mServo.Attached == true)
            {
```

```
                mServo.servos[0].VelocityLimit = 1000;
                mServo.servos[0].Engaged = true;
                mServo.servos[0].Position = 120;
                Thread.Sleep(500);
                mServo.servos[0].Engaged = false;

                mServo.Attach -= new AttachEventHandler(mServo_Conectar
Handler);
                mServo.Detach -= new DetachEventHandler
(mServo_DesconectarHandler);
                mServo.Error -= new ErrorEventHandler(mServo_Error
Handler);
            }


            Application.DoEvents();

            // Cerramos el phidget
            iKit.close();
            mServo.close();

        }
```

En la sección que corresponde al servo, indicamos que la velocidad con la que deseamos que regrese a su posición centrada es de **1000**. Para esto, hacemos uso de la propiedad denominada **VelocityLimit**. Debemos tener en cuenta que el servo sólo se moverá cuando la propiedad **Engaged** se encuentre configurada como **true**. Luego, mediante la propiedad **Position**, indicamos la posición donde deseamos que se estacione el servo, en este caso, **120**.

Para evitar que se presenten problemas, esperamos un poco para darle tiempo al servo de llegar a su posición. Una vez cumplido este lapso, ingresamos **Engaged** como **false** y procedemos a quitar los handlers del controlador de servos. Al final del handler, simplemente, cerramos los Phidgets.

Nuestro objetivo principal es que el movimiento del servo se realice de acuerdo a los cambios que el usuario efectúe sobre el **Trackbar**; por lo que necesitamos un handler específico para el **TrackBar**, leer su posición y mandar esta posición al servo correspondiente. Por esta razón, es importante que los valores que ingresamos al **TrackBar** deben ser similares a los del servo.

```
        private void trkPosicion_Scroll(object sender, EventArgs e)
        {
            // Verificamos si esta conectado
            if (conectado1)
            {
                // Obtenemos el valor y lo mostramos
                lblPosicion.Text = trkPosicion.Value.ToString();

                // Colocamos la posicion en el servo
                mServo.servos[0].Position = trkPosicion.Value;
            }

        }
```

Para hacer el movimiento suave cada vez que cambia la posición del **Trackbar**, se ejecuta el código del siguiente handler.

```
        private void trkPosicion_LocationChanged(object sender,
EventArgs e)
        {
            // Verificamos si esta conectado
            if (conectado1)
            {
                mServo.servos[0].Engaged = true; ;
                Thread.Sleep(500);
                mServo.servos[0].Engaged = false;
            }
        }
```

De esta forma, habilitamos el movimiento del handler, el cual pasará a la posición guardada en la propiedad **Position**. Luego, el programa esperará un poco para que el servo se mueva y se procederá a deshabilitar el movimiento.

El controlador de servos también presenta un número de serie, que utilizamos para conectarnos con el Phidget. Esto último hace necesario que podamos ingresar y guardar el número de serie desde nuestra aplicación. Como podemos ver, el código es muy parecido al del primer Phidget, con la única diferencia de las llaves en el registro a las que se refiere. El código debe aparecer en el handler del elemento del menú correspondiente.

```
        private void configurarPhidget1ToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            // Este codigo es para el phidget 1

            Configuracion dlgConfig = new Configuracion();
            dlgConfig.NumeroSerie = numeroSerie1;

            if (dlgConfig.ShowDialog() == DialogResult.OK)
            {
                // Escribimos el nuevo valor
                numeroSerie1 = dlgConfig.NumeroSerie;

                // Abrimos la llave
                RegistryKey sfwKey =
Registry.LocalMachine.OpenSubKey("Software", true);
                RegistryKey redKey = sfwKey.CreateSubKey("RedUsers2");

                RegistryKey robotKey = redKey.CreateSubKey("Robots2");

                // Colocamos el nuevo valor
                robotKey.SetValue("NumeroSerie1", (object)numeroSerie1);

            }
        }
```

Ahora, procedemos a ingresar el código de los handlers propios del controlador de servos. Empezaremos con el handler de la conexión.

```
        public void mServo_ConectarHandler(object sender,
AttachEventArgs e)
        {
            // Indicamos que hay conexion
            conectado1 = true;

            // Enviamos mensaje de la conexion
            lblPhidget1.Text = e.Device.Name;

            // Indicamos con color verde que hay conexion
```

```
        txtPhidget1.BackColor = System.Drawing.Color.Lime;

        // Configuramos sus caracteristicas
        if (mServo.Attached)
        {
            // Configuramos el servo 0

            // Colocamos la aceleracion
            mServo.servos[0].Acceleration = 1000.0;

            // Colocamos la posicion inicial
            mServo.servos[0].Engaged = true;
            mServo.servos[0].Position = 120.0;
            mServo.servos[0].VelocityLimit = 10.0;

        }

    }
```

Este último handler se invoca cuando el Phidget se ha conectado, por lo que en primer lugar ingresamos el valor **true** en la variable **conectado1**. Obtenemos el nombre del Phidget y lo mostramos en la etiqueta; al igual que con el otro Phidget, ponemos en un color verde el **TextBox**. Con esto último, estamos indicando de forma visual que se ha establecido la conexión. Como próximo paso procedemos a ingresar las características del servo, pero antes es necesario verificar que ya tengamos conexión. Lo primero que ingresamos es la aceleración y, para esto, usamos la propiedad **Acceleration** del servo correspondiente. En nuestro caso, usamos una aceleración de **1000**, pero podríamos usar otro valor para incrementar o decrementar la aceleración del servo. Como deseamos que cuando el Phidget se conecte al servo tome la posición central, entonces, ingresamos el valor **true** en la propiedad **Engaged** y el valor **120** en la propiedad **Position**. Debemos tener en cuenta que cada modelo de servo tiene sus propias características, por lo que no siempre es el valor **120** el que representa la posición central. Para hacer el movimiento del servo suave, ingresamos una velocidad de **10** en la propiedad de **VelocityLimit**. Este valor también puede variar de acuerdo con la velocidad que deseemos imprimir al servo.

El siguiente handler es el de desconexión. Cuando esto sucede, simplemente, indicamos de manera visual que ya no se encuentra conectado el Phidget y colocamos la variable **conectado1** como **false**.

```
        public void mServo_DesconectarHandler(object sender,
DetachEventArgs e)
        {
            // Indicamos que hay desconexion
            conectado1 = false;

            // Quitamos el mensaje
            lblPhidget1.Text = "Sin Phidget";

            // Indicamos con color rojo que no hay conexion
            txtPhidget1.BackColor = System.Drawing.Color.Red;

        }
```

En este último caso, indicamos en el handler mediante el color rojo que se produjo la desconexión. Como podemos apreciar a continuación, el handler para el error tiene un código similar. En este caso, se muestra el error correspondiente y se establece el **TextBox** en color amarillo.

```
        public void mServo_ErrorHandler(object sender, ErrorEventArgs e)
        {
            // Indicamos que hay desconexion
            conectado1 = false;

            // Mandamos mensaje de error
            lblPhidget1.Text = e.Description;

            // Indicamos con color amarillo que hay problemas
            txtPhidget1.BackColor = System.Drawing.Color.Yellow;
        }
```

III CICLOS DE RECARGA

Dentro de las especificaciones de las baterías secundarias, se puede conocer la cantidad de ciclos de recarga. Esto es, cuántas veces se puede recargar la batería y todavía seguir siendo útil. Algunas baterías tienen una duración un poco mayor a los ciclos indicados, y en otras menor, siempre dependiendo del uso que les demos.

Con esto, ya tenemos el programa que controla a los dos Phidgets, y es posible controlar al servo desde nuestra aplicación.

Método para encontrar el rango del servo

Puede suceder que tengamos un servo, pero que no podamos encontrar las especificaciones sobre su rango de movimiento. En este caso, es necesario encontrarlo de forma experimental. Para lograrlo, lo primero que tenemos que hacer es colocar al servo sobre un papel y poner sobre él una pequeña palanca. Esta palanca nos servirá de guía para observar el rango de movimiento. La siguiente figura nos muestra cómo podemos hacer esto.

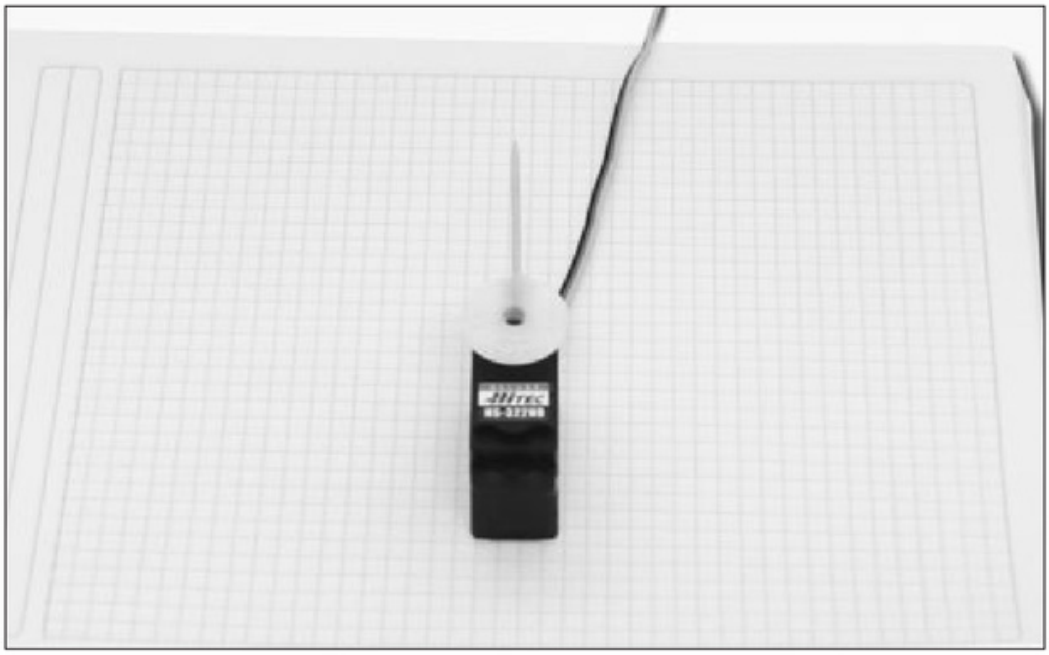


***Figura 16.** Es posible observar el servo colocado sobre el papel. Así, se encuentra listo para que distingamos su rango de movimiento.*

También necesitamos de un programa que pueda decrementar e incrementar el movimiento del servo poco a poco, ya sea cuando se oprime un botón o cuando se usa el teclado. Es importante poder desplegar la posición actual del servo para

III EL POTENCIÓMETRO

El **potenciómetro** es una resistencia que tiene tres terminales y se puede utilizar como una resistencia variable. Lo podemos encontrar con un eje, el cual al girar modifica el valor de la resistencia. Generalmente, la terminal central es la utilizada para obtener el valor cambiante.

conocer su valor. Entonces, así empezamos a mover el servo con lentitud hasta que notamos que ya no puede desplazarse más. En ese momento, podemos ver cuál es el valor de la posición mínima del servo. En el papel, marcamos esta posición, tal y como se muestra en la **figura 17**.

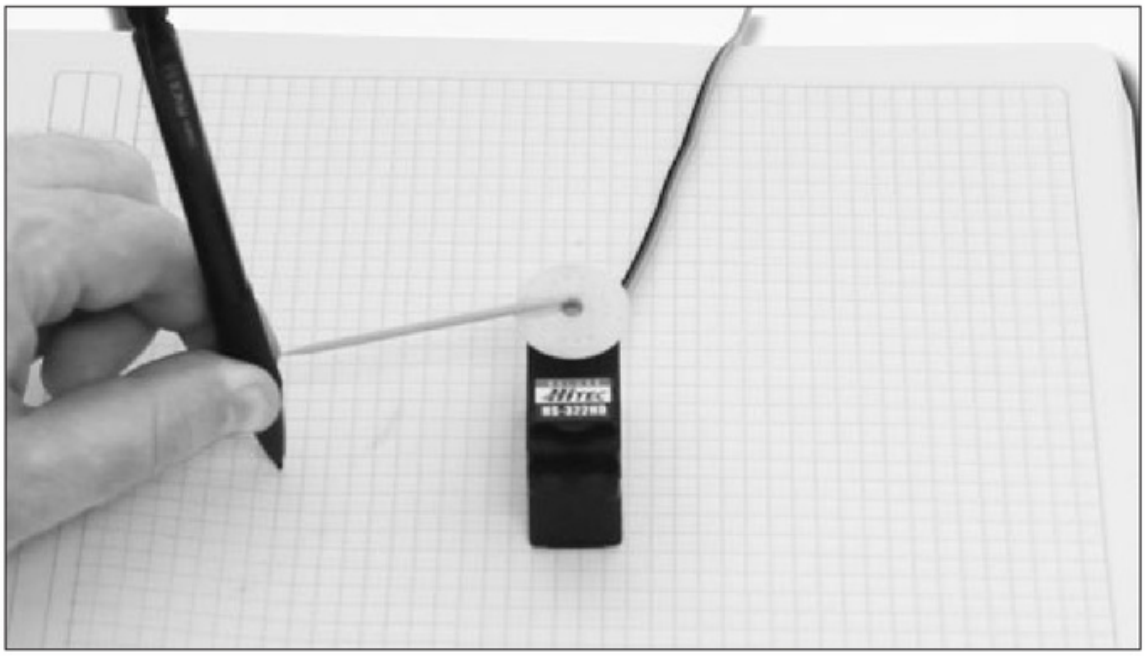


Figura 17. *Una vez que hemos encontrado la posición mínima, la marcamos en el papel.*

Ahora, se empieza a mover el servo en la otra dirección hasta que ya no sea posible desplazarlo más. En ese momento, vemos el valor de posición indicado en la pantalla: hemos encontrado el máximo. También marcamos en el papel esta posición.

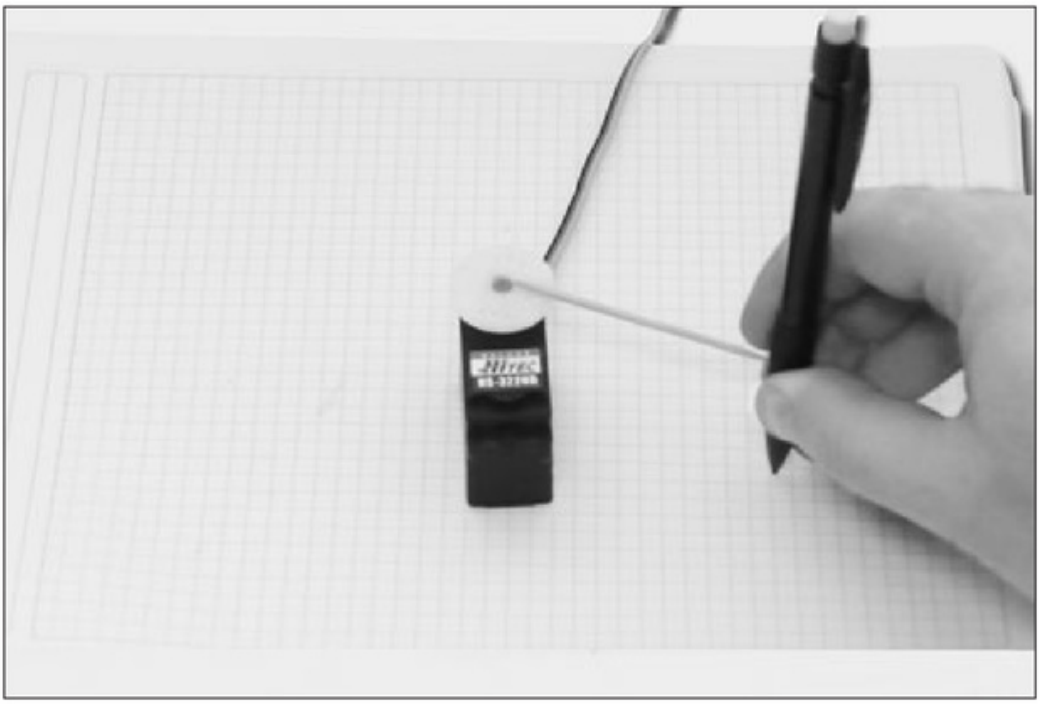


Figura 18. *La posición máxima ha sido encontrada y, también, la marcamos en el papel.*

Ya tenemos los valores que representan la posición mínima y máxima, pero, si lo deseamos, también podemos conocer en grados el recorrido del servo. Esto es muy sencillo de hacer. Simplemente, trazamos una línea que va desde donde estaba el eje del servo a la primera marca, y otra línea que va desde esta misma posición hasta la segunda marca. Luego, sólo debemos medir el ángulo que existe entre ambos radios.

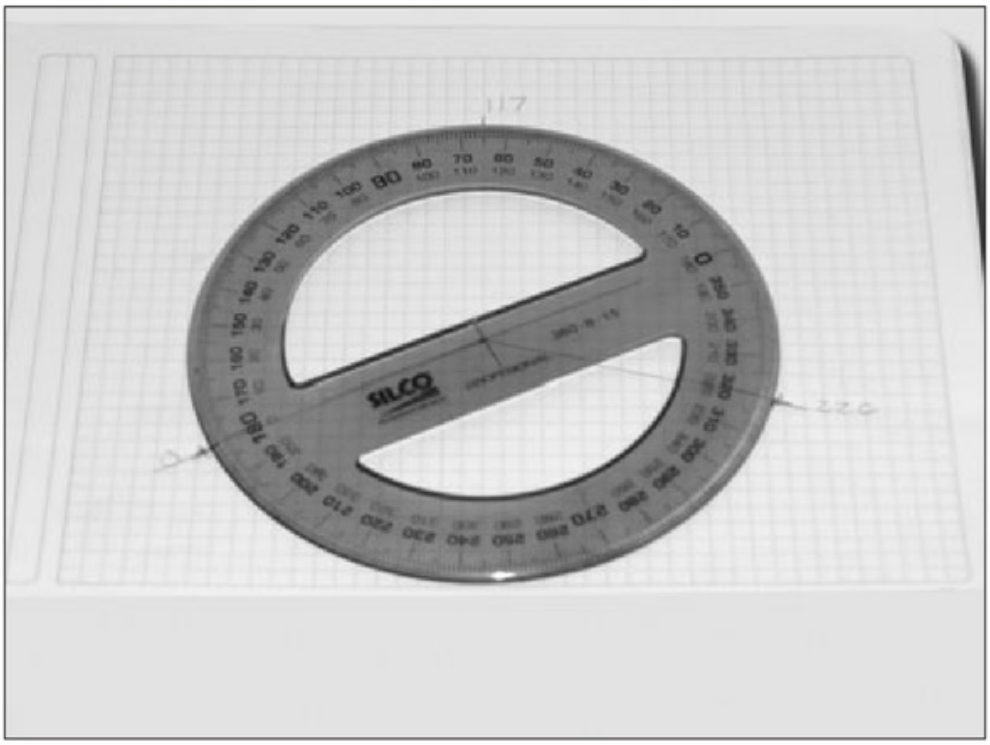


Figura 19. *Ahora, resulta fácil medir el ángulo que recorre el servo. Ya hemos obtenido toda la información necesaria.*

De esta forma, sabemos cómo poder trabajar con los servos y, en los próximos capítulos, crearemos aplicaciones robóticas con ellos.

... RESUMEN

En este capítulo aprendimos que, cuando construimos robots, trabajamos con diferentes componentes. Las baterías proveen la energía necesaria gracias a una reacción química que se lleva a cabo en su interior. Además, algunas baterías se pueden recargar y otras, no. Necesitamos conocer el voltaje y la corriente que puede proveer la batería. Después, vimos que los motores eléctricos suministran el movimiento para el motor. La torca nos permite saber la cantidad de carga que puede mover el motor. Con los engranes es posible modificar la velocidad y la torca del motor. El servo nos permite crear movimiento con control de la posición. Por último, establecimos que, por medio de los Phidgets, podemos controlar a los servos.

ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1. ¿Cómo obtienen su energía las baterías?

2. ¿Qué característica tienen las baterías primarias?

3. ¿Qué característica tienen las baterías secundarias?

4. ¿Qué tan alto es el voltaje de carga?

5. ¿Cómo se mide la cantidad de corriente eléctrica que puede proveer la batería?

6. ¿Cómo funciona un motor eléctrico?

7. ¿Qué es la torca?

8. ¿De qué forma reducimos la velocidad de un motor?

9. ¿Cómo podemos aumentar la torca del motor?

10. ¿Cómo funciona el servo?

ACTIVIDADES

1. Modifique el programa de este capítulo para que pueda variar la velocidad del servo con un Trackbar.

2. Modifique el programa de este capítulo para que pueda controlar dos servos.

Robot con visión por computadora

En los capítulos anteriores, hemos aprendido sobre las bases de los robots y cómo controlar los Phidgets. Ahora, llegó el momento de poner este conocimiento a trabajar y realizar nuestro primer proyecto. Este proyecto muestra cómo es posible construir un sistema robótico que tenga visión por computadora.

DESCRIPCIÓN DEL PROYECTO

El siguiente proyecto consistirá de dos partes fundamentales: el hardware y el software. El hardware es un pequeño robot con dos ejes de libertad. En el cabezal de este robot, colocaremos una **webcam**. La webcam enviará la imagen a nuestra aplicación para que luego sea procesada.

Al tener dos grados de libertad, el robot, podrá rotar en el **eje X** y, también, en el **eje Y**. Para lograr la rotación, usaremos dos servos. El control de los servos se hará por medio del **Phidget Advanced Servo**. El sistema necesita de una base donde poder anclar el primer servo, al cual se le colocará un brazo. El segundo servo se montará sobre el brazo, y el cabezal se colocará en éste. La base puede tener un pedestal pesado, con el fin de evitar que el robot se vuelque al realizar un movimiento brusco. El software nos permitirá conectarnos a la webcam y procesar la información; también, deberá permitirnos vincularnos con el Phidget para llevar a cabo el control de los servos. Con la aplicación, podremos controlar de manera manual el movimiento del robot, si así lo deseamos.

En su modo automático, la aplicación será capaz de reconocer un objeto, basándose en su color, y de controlar al robot para que la webcam siga el movimiento del objeto. Es necesario, entonces, seleccionar el color que deberá seguir la webcam. El programa nos mostrará dos imágenes: la primera de ellas es la señal directa de la webcam, y la segunda es la imagen procesada por la aplicación. Para construir el cuerpo del robot, usaremos cartón rígido de distintos espesores: 1, 3, 4 y 5 milímetros. Si no encontramos piezas de cartón mayores a 1 milímetro, podemos sustituirlo por piezas pegadas entre sí. Por ejemplo, un cartón de 3 milímetros de espesor, se puede sustituir por tres piezas del mismo tamaño, de 1 milímetro de espesor, pero pegadas y apiladas entre sí. Por otra parte, el cartón puede ser remplazado tanto por plástico como por madera, en caso de ser necesario. Para adherir las piezas de cartón, se utilizará un pegamento de silicón líquido, pero cualquier otro pegamento fuerte para cartón también puede ser empleado.

III GRADOS DE LIBERTAD

Los **grados de libertad** son la cantidad de movimientos independientes, ya sean traslaciones o rotaciones, que puede llevar a cabo un sistema mecánico; en este caso, el robot que comenzamos a construir en este proyecto. Cuantos más grados de libertad sumemos, más complejo resultará el movimiento del robot.

Componentes necesarios

Para poder armar este robot, es necesario tener una serie de componentes:

NOMBRE	CANTIDAD	DESCRIPCIÓN
Phidget	1	PhidgetAdvancedServo
Servo	2	Servos
WebCam	1	Webcam USB
A	1	Cartón 12 x 7 cm, 1 mm de espesor
B	2	Cartón 4 x 2.5 cm, 3 mm de espesor
C	2	Cartón 1.5 x 2.5 cm, 3 mm de espesor
D	1	Cartón 11 x 3 cm, 4 mm de espesor
E	1	Cartón 2 x 2.5 cm, 5 mm de espesor
F	2	Cartón 5.5 x 2.5 cm, 3 mm de espesor
G	1	Cartón 2 x 2.5 cm, 3 mm de espesor
H	1	Cartón 2.5 x 2.5 cm, 3 mm de espesor
I	1	Cartón 6 x 4 cm, 1 mm de espesor
J	2	Cartón 2 x 1 cm, 3 mm de espesor
K (Tornillos)	4	Tornillos incluidos en los Servos
L (Bandas)	2	Bandas elásticas
M (Base)	1	Madera de 12 x 7 x 5 cm

***Tabla 1.** Estos serán los componentes necesarios para la construcción del robot con visión por computadora.*

Todos los componentes tienen un nombre para facilitar su reconocimiento. Además, necesitaremos un destornillador, navaja, reglas y pegamento de Silicon líquido.

CONSTRUCCIÓN DEL ROBOT

El primer elemento que comenzaremos a construir es la plataforma del robot. Esta plataforma provee una base firme para que el robot se pueda mover. En ella, colo-

III HERRAMIENTAS Y CORTES

Los proyectos de este libro necesitan del uso de herramientas para llevar a cabo cortes. Siempre que se usan herramientas, se debe tener mucho cuidado para evitar accidentes. Si no sabemos utilizarlas con precisión o somos menores de edad, debemos solicitar la ayuda de una persona adulta que sepa manejarlas correctamente.

caremos el servo que moverá al robot en el eje Y. Para construir la plataforma, necesitaremos los componentes marcados como **A**, **B** y **C**.

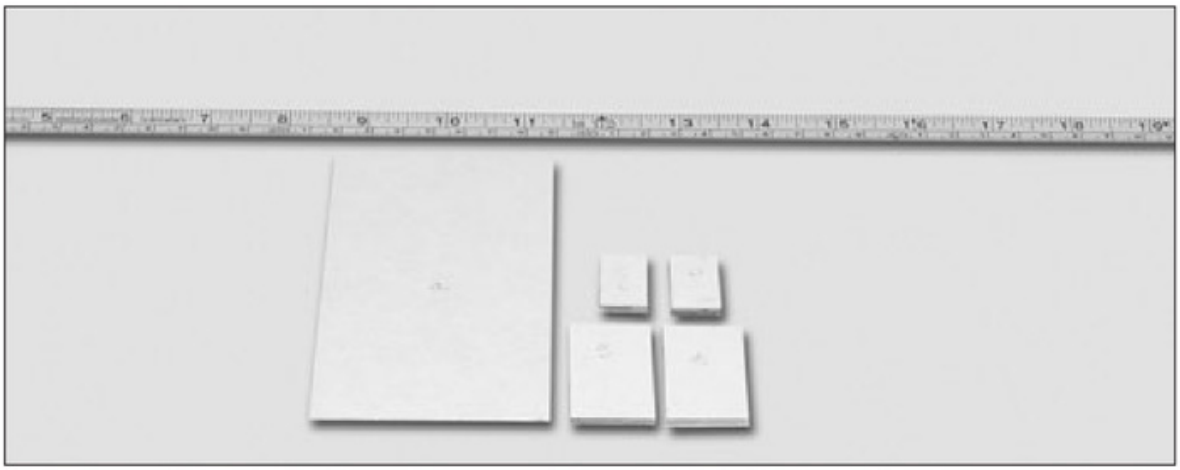


Figura 1. *Éstos son los componentes necesarios para empezar a crear la plataforma.*

En el proceso, también, necesitaremos de un servo. El servo que usamos es el que viene en el paquete del Phidget y se muestra en la siguiente figura.

Figura 2. *Éste es el servo que colocaremos en la plataforma. Es importante notar la zona donde se encuentran los cables de conexión.*

III LICENCIA DEL SOFTWARE Y DE LOS DISEÑOS

El software y los diseños que se proveen en este libro se pueden usar libremente con fines personales y académicos con el único requisito de listar en la bibliografía del proyecto a este libro. Las bibliotecas NCBitmap, NCVideo y otras provistas por el autor son experimentales, no se da ningún tipo de garantía sobre ellas y no se pueden usar en proyectos comerciales.

En primer lugar, para soportar la plataforma, necesitamos también el componente **M**, el cual puede ser una base de madera o de metal con suficiente peso para evitar el volcado del robot cuando se mueva.

Figura 3. *Aquí vemos un ejemplo de base que podemos utilizar; también puede emplearse madera.*

Ahora que tenemos todo lo necesario, procedemos a construir la plataforma. Tomamos el componente **A** y trazamos con un lápiz dos diagonales. Usaremos estas diagonales como referencia para el resto del montaje.

Figura 4. *Las dos diagonales nos servirán para colocar al servo en posición.*

Luego, debemos colocar el servo centrado sobre el componente **A**. Para esto, usamos las diagonales como guía. La colocación del servo en el centro también ayudará a balancear de manera correcta el robot sobre la plataforma.

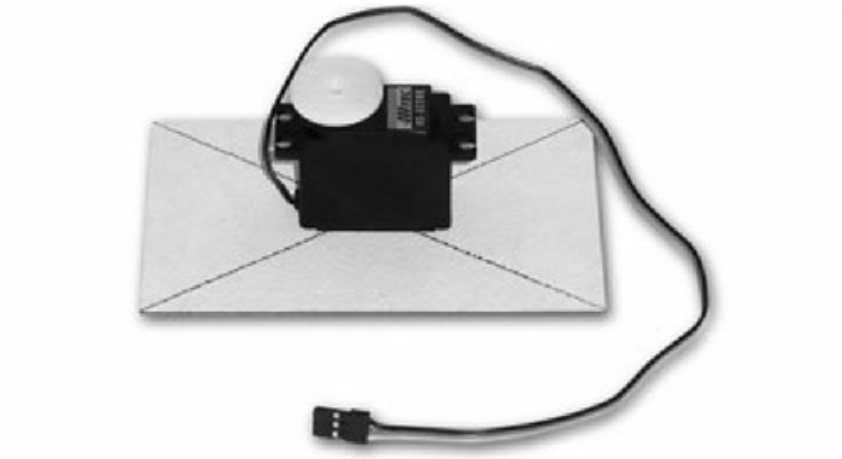


Figura 5. *El servo debe quedar bien centrado sobre el componente A, usando las diagonales como guía.*

A continuación, deberemos utilizar el pegamento y colocar un componente **B** y un componente **C**, aprovechando el servo como soporte. Es importante que el componente **C** no obstruya la salida de los cables del servo. Este montaje lo podemos observar mejor en la siguiente figura.

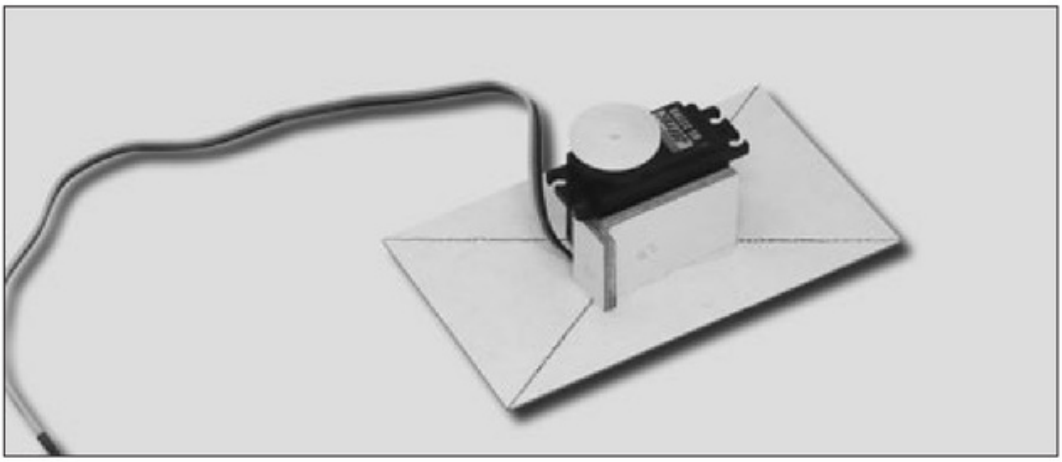


Figura 6. *Los componentes son pegados a la base de la plataforma usando el servo como soporte y guía. El servo no debe ser pegado para así asegurarnos otros usos futuros.*

De forma similar, pegamos los dos componentes **B** y **C** en el otro lado del servo. Aquí también, debemos tener mucho cuidado de no pegar el servo.

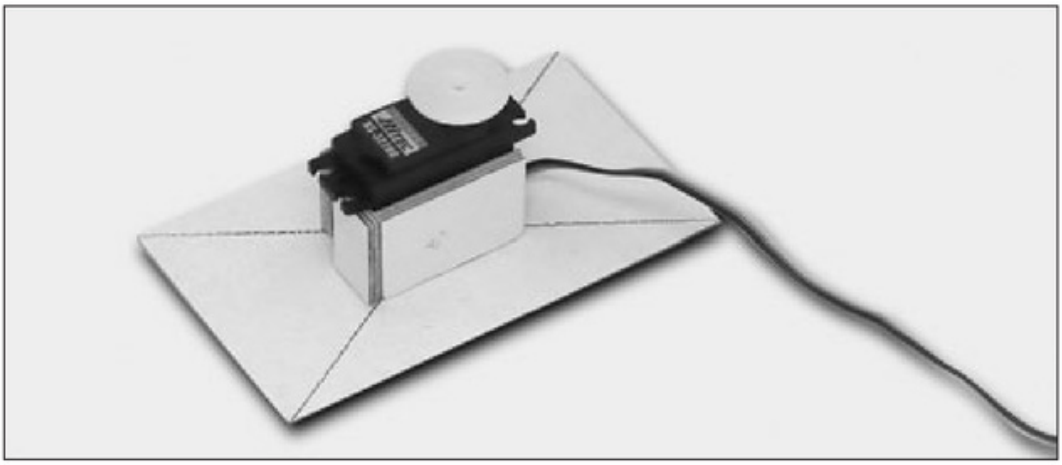


Figura 7. *En el otro lado del servo, también colocamos los componentes en la base.*

III NO PEGAR EL SERVO

Debemos tener en cuenta que, cuando hacemos el montaje de los componentes, no hay que pegar el servo; los únicos que deben llevar el pegamento son los componentes. De esta forma, podemos utilizar el servo en diferentes proyectos. En caso de ser necesario, es mejor utilizar bandas elásticas para mantenerlo en posición.

Con esto, hemos creado una especie de caja ajustada alrededor del servo. Esta caja nos ayudará a tenerlo correcta y firmemente posicionado. Una vez que el pegamento haya secado, podemos retirar el servo con cuidado y observar la caja, tal y como la muestra la siguiente figura.

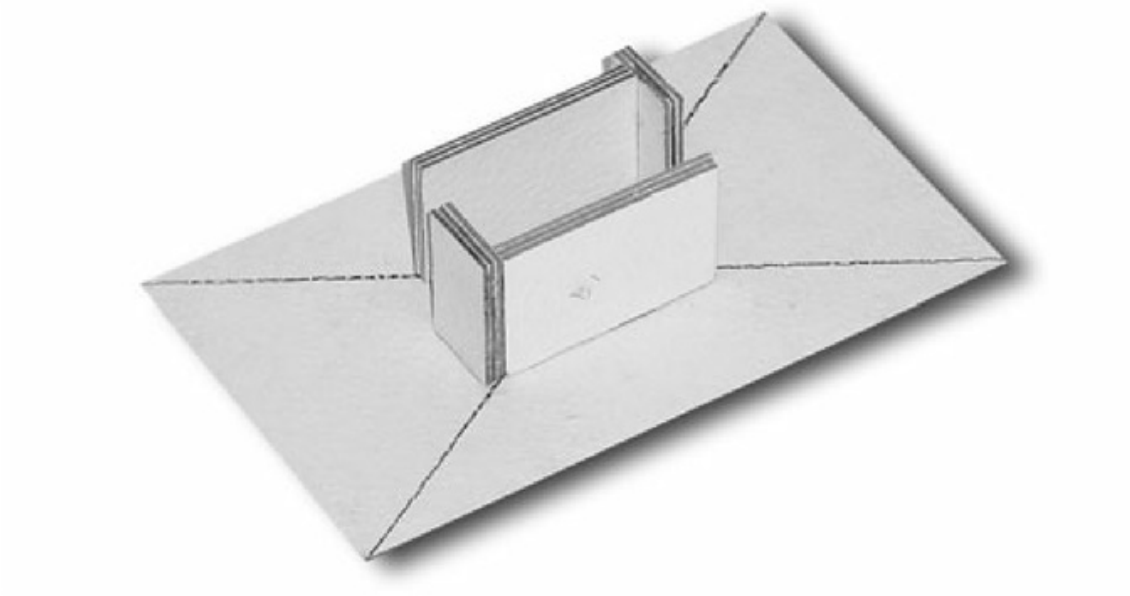


Figura 8. *Una caja firme se ha creado para soportar el servo.*

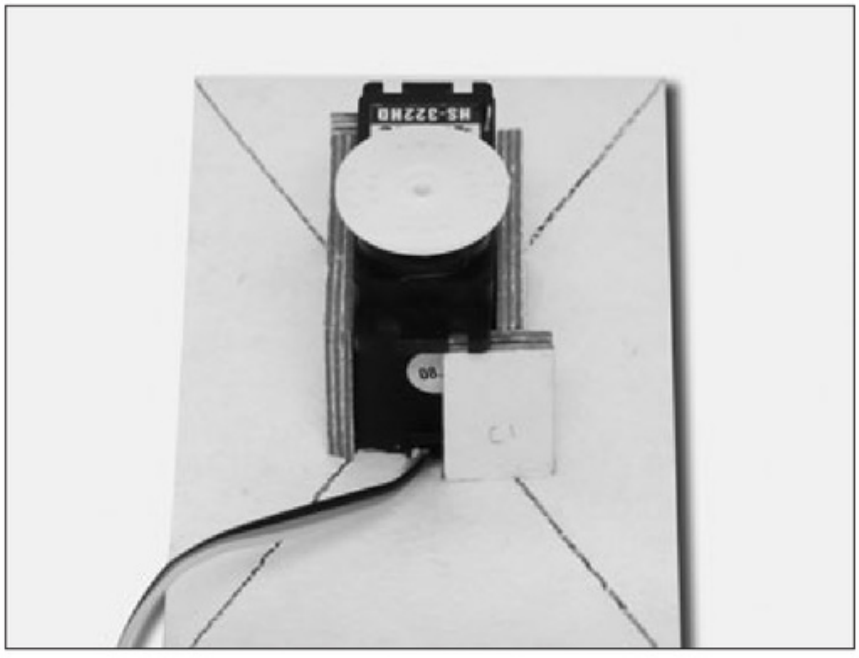


Figura 9. *La caja debe permitir la salida de los cables sin forzarlos.*

Si lo deseamos, podemos pintar la plataforma; para esto, usaremos cualquier color que nos agrade. La plataforma que tenemos estará ubicada sobre el componente **M**. Si deseamos montarlo y desmontarlo, podemos usar las bandas elásticas, pero, si preferimos montarlo de manera definitiva, es mejor usar pegamento. Nuestra plataforma terminada se muestra en la siguiente figura.

Figura 10. *Aquí podemos observar nuestra plataforma finalizada, sobre su base.*

La creación del brazo

La siguiente parte del robot que debemos crear es el brazo, que será movido por el servo y que va unido a la base. Esto hace que el brazo gire en el eje Y. El brazo también tendrá una caja en su extremo que soportará al otro servo. La distancia del brazo está calculada para que el lente de la cámara quede lo más cerca posible del eje de rotación Y.

El ensamble del brazo es sencillo y seguiremos una metodología similar al ensamble de la base. Después de ser acoplado, deberemos montarlo sobre el primer servo.

En la construcción del brazo, necesitamos los componentes **D**, **E**, **F** y **G**, que podemos apreciar en la siguiente figura.

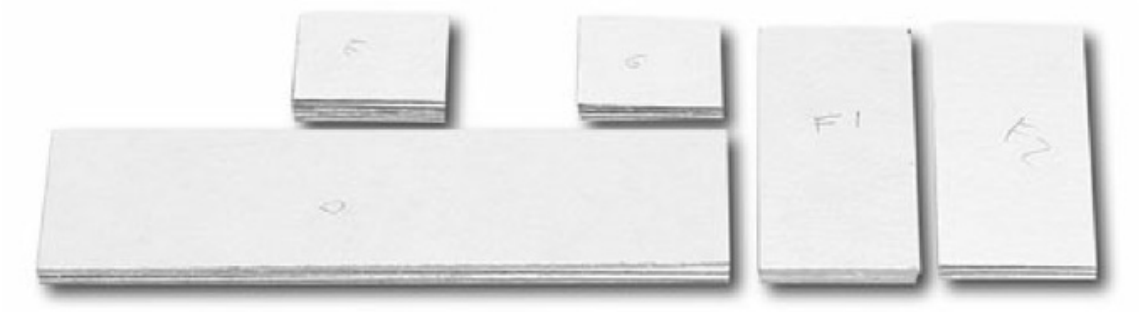


Figura 11. *Éstos son los componentes que necesitamos para poder construir el brazo del robot.*

Tenemos que colocar un asiento para el servo, ya que éste necesita estar en posición vertical: el componente **E** cumplirá esta función. Debemos ubicar el componente **E** cerca de uno de los extremos del componente **D**. Debemos procurar ubicarlo en una posición centrada y aproximadamente a dos milímetros del extremo de **D**. El componente debe estar muy bien pegado.

Figura 12. *El asiento para el servo queda colocado cerca del extremo del componente D.*

Procedemos a colocar el servo sobre su asiento. Utilizaremos el servo para poder guiar el ensamble de los siguientes componentes y debemos asegurarnos de no pegarlo al asiento ni a ningún otro elemento.

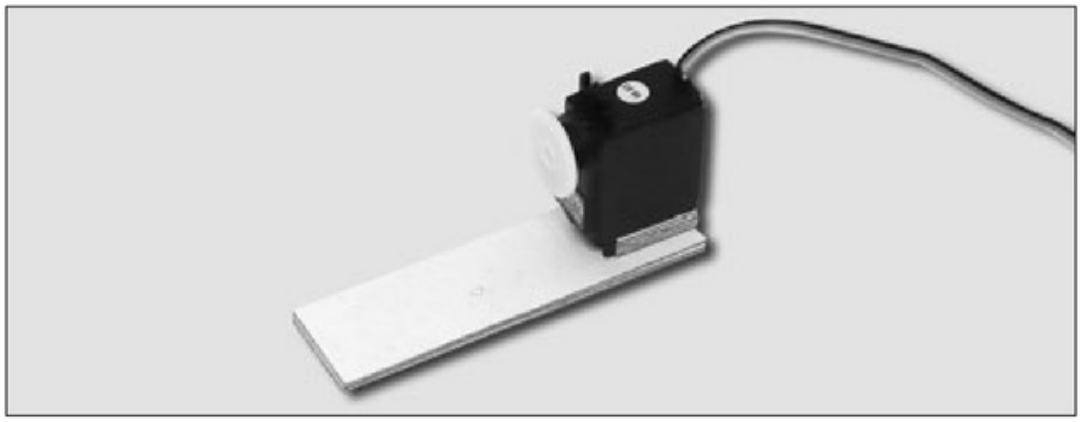


Figura 13. *El servo que usaremos para la rotación en X se coloca sobre su asiento.*

Con el asiento colocado, procedemos a crear la caja que sostendrá al servo. Tomamos uno de los componentes **F** y, usando al servo como guía, lo ubicamos a su lado. Es necesario colocar el pegamento en **F**, de tal forma que quede adherido al componente **D** y al componente **F**. El servo nos guía para que se mantenga erecto. El componente **F** sí debe llegar hasta el extremo del brazo, a diferencia del asiento.

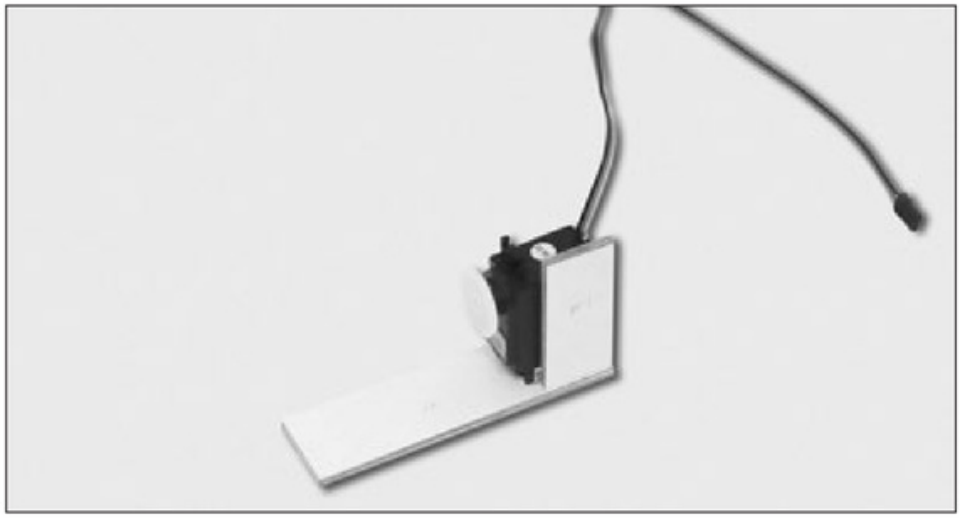


Figura 14. *Colocamos el primer lado de la caja; el servo solamente nos sirve como guía.*

Cuando ese lado de la caja se encuentre en su lugar y el pegamento haya secado, entonces, procedemos a hacer lo mismo con el otro componente **F**, pero en el otro lado del brazo. La siguiente figura muestra esta posición.

Figura 15. *El otro lado de la caja ha sido colocado.*

La caja necesita un poco de soporte estructural, por lo que usaremos el componente **G** para hacer, precisamente, este trabajo. Con el componente **G**, uniremos los componentes **F**. Esto lo hacemos en la parte trasera del servo. Colocamos pegamento en dos lados opuestos de **G** para insertarlo y pegarlo entre los componentes **F**, a una altura adecuada. Es posible que sea necesario hacer un poco de presión con los dedos sobre los componentes **F** hasta que el pegamento termine de secar.

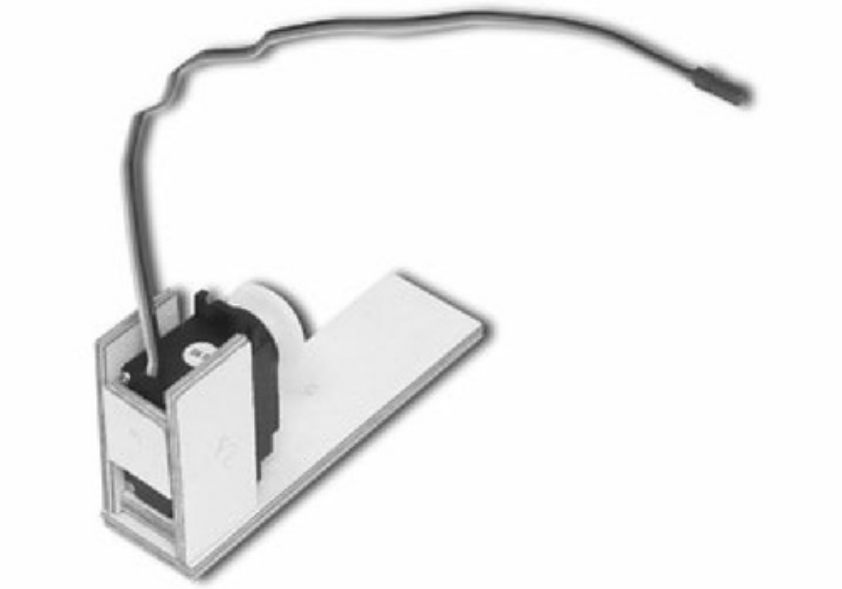


Figura 16. *Ahora, la caja ya es estructuralmente más firme y podrá soportar mejor el movimiento del robot.*

Una vez que el pegamento se encuentre seco, podemos retirar el servo. Esta acción debe poder realizarse con facilidad. Si el pegamento es el correcto, la caja se sentirá firme y podrá mantener al servo en posición con casi nada de movimiento. El brazo debe lucir como en la figura siguiente.

Figura 17. *El brazo está casi terminado; todos los elementos estructurales han sido colocados.*

El brazo que hemos creado deberá ir montado sobre el primer servo. Para facilitar el montaje, podemos utilizar el elemento de actuación del servo como guía y marcar los puntos donde, más adelante, colocaremos los tornillos que unirán el servo con el brazo. El centro del elemento de actuación debe estar a 2,5 centímetros del extremo del brazo. Aquí, lo más conveniente es marcar mediante un lápiz. El elemento de actuación tiene perforaciones que usamos para hacer las marcas.

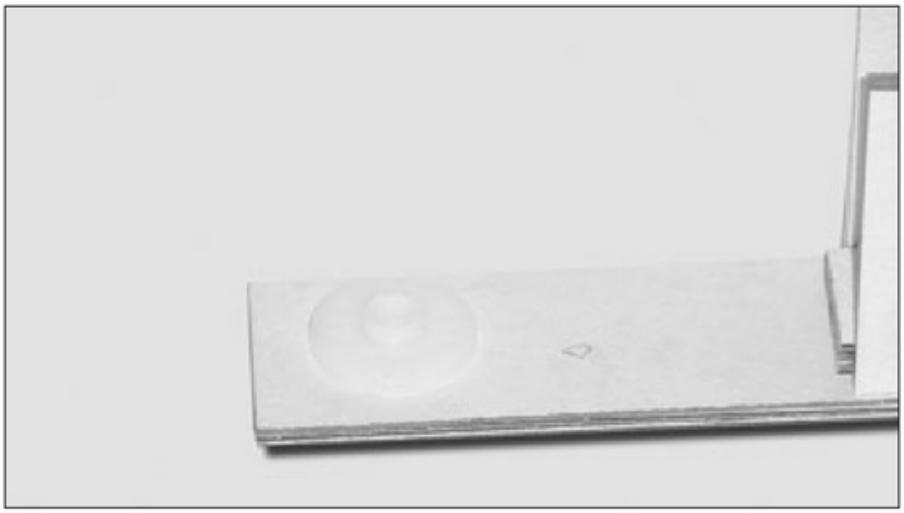


Figura 18. *El elemento de actuación nos permite hacer las marcas donde, después, se colocarán los tornillos para el montaje al servo.*

Es momento de pintar el brazo. Podemos usar el mismo color que para la base y, cuando la pintura se encuentre seca, procedemos a montar el brazo sobre el servo.

Ubicamos, otra vez, el elemento de actuación sobre el servo. Con cuidado, colocamos los tornillos en el brazo, sobre las marcas previamente realizadas y atornillamos hasta que el extremo del tornillo haya salido del otro lado. Estas puntas nos ayudan a colocar el brazo sobre el elemento de actuación por lo que deben encajar sobre los agujeros del elemento de actuación. Cuando se encuentren colocadas de manera correcta, atornillamos aún más y, con esto, queda fijado el brazo al servo.

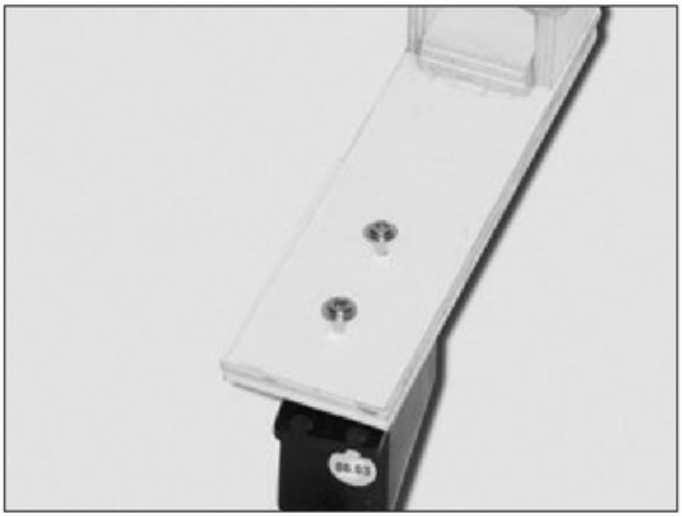


Figura 19. *Con los tornillos, sujetamos el brazo del robot al servo.*

Los tornillos se deben ajustar con lentitud. Hay que tener cuidado de no colocarlos muy profundos dado que pueden impedir el movimiento libre del servo. Sólo se colocarán a la profundidad necesaria para permitir que el brazo esté sujeto de manera firme.

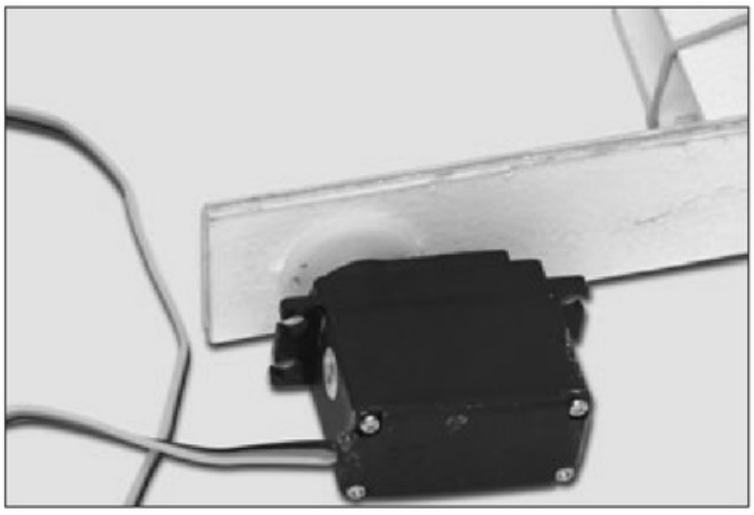


Figura 20. *La función de los tornillos es sujetar el brazo, y no deben impedir el movimiento del servo.*

Ahora que se encuentra montado el brazo sobre el servo, simplemente colocamos el primer servo en su caja, que se encuentra en la plataforma. Podemos observar que el robot ya empieza a tomar forma.

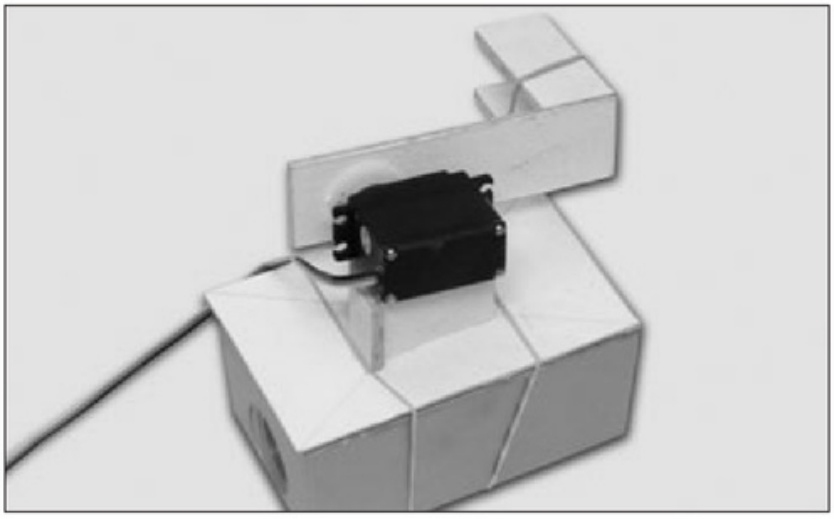


Figura 21. *El servo que tiene el brazo montado se coloca en la caja de la plataforma.*

Figura 22. *El robot ya empieza a tomar forma, pero aún falta montar la cámara.*

Creación del soporte de la cámara

El brazo robótico está prácticamente completo, sólo nos falta crear el soporte de la cámara. Este sopote se montará sobre el segundo servo para que la cámara pueda rotar,

III EL SERVO DEBE ESTAR CENTRADO

Para evitar problemas posteriores, el servo debe estar en su posición central antes de colocar el brazo. Podemos usar el programa del capítulo anterior para hacerlo. Otra forma, es utilizando el panel de control de Phidgets. Al estar centrado, proveemos un mejor rango de movimiento.

también, en el eje X. Para armar el soporte, usaremos los componentes **H**, **I** y **J**. Después, usaremos el elemento de actuación y un par de tornillos para montarlo sobre el servo.

Figura 23. *Para el soporte de la cámara, necesitamos los componentes H, I y J.*

Para empezar, trazamos una línea que cruce el componente **I** por la mitad. Esta línea nos servirá de guía para pegar los otros componentes y ubicar la cámara de manera adecuada. Esta línea debe trazarse paralela al lado mayor del componente. Tomamos uno de los componentes **J** y, también, trazamos una línea por la mitad, pero, en este caso, debe ser paralela al lado menor del componente.

Figura 24. *Trazamos líneas centradas en los componentes I y J que nos servirán de guía durante el montaje.*

Ahora, pegamos el componente **J** sobre el componente **I**. El componente debe pegarse hasta el extremo de **I**. Usamos las líneas que trazamos para ayudarnos a colocar **J** perfectamente centrado. Las siguientes dos figuras nos muestran esto.

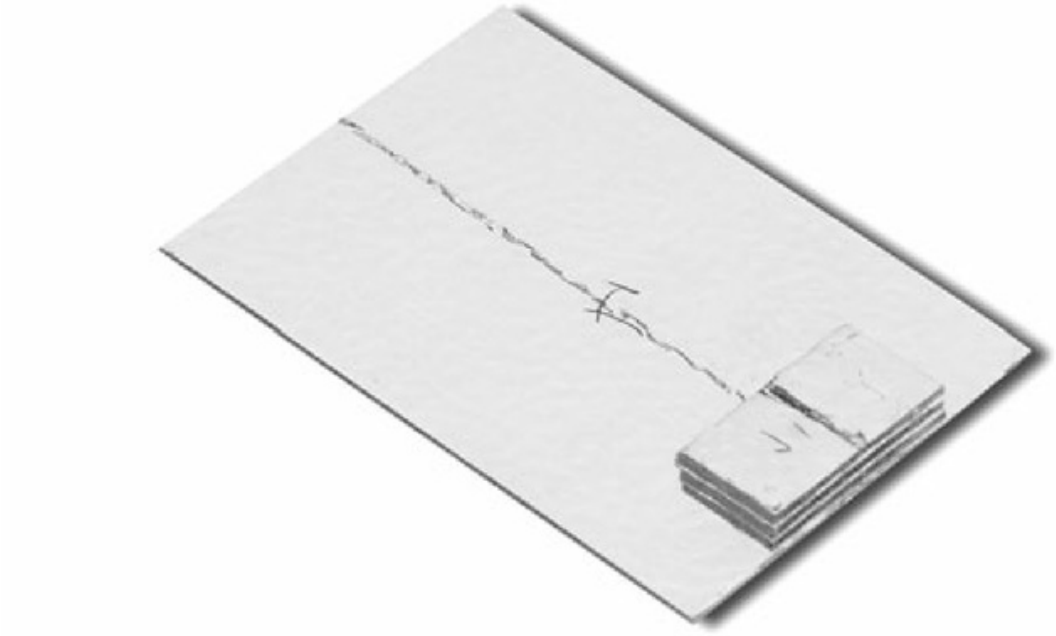


Figura 25. *Debemos pegar bien firme el componente J sobre I.*

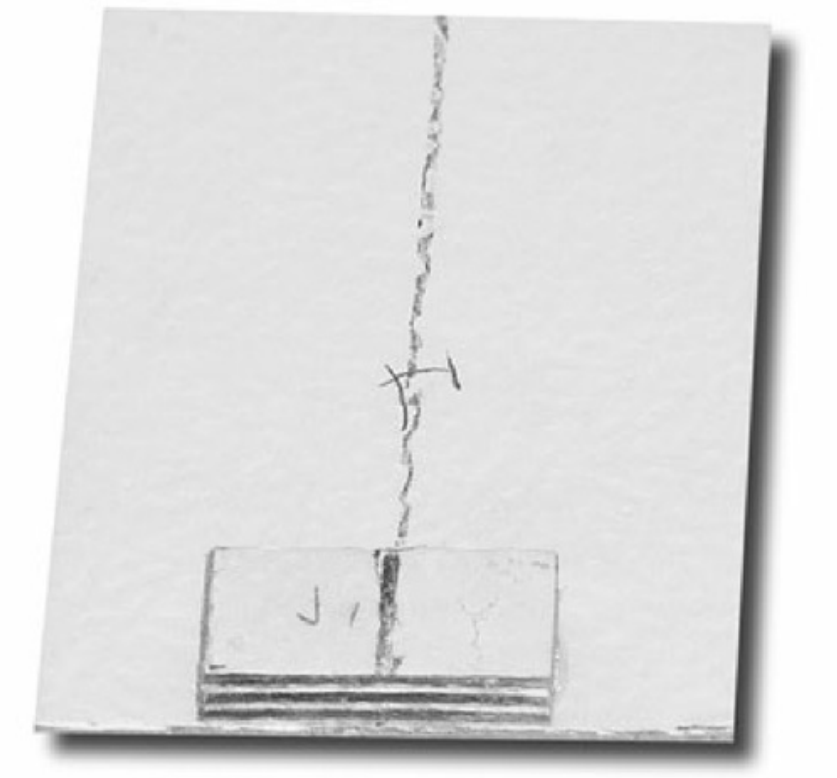


Figura 26. *El componente J debe estar centrado y colocado al extremo de I.*

Una vez que el pegamento haya secado, colocamos el otro componente **J** sobre el componente **I**, pero del lado contrario. De esta forma, los componentes **J** ayudarán a formar un soporte para el componente **I**.

Figura 27. *Otro de los componentes J se pega en el reverso del componente I.*

Debemos procurar que ambos componentes **J** queden pegados en el mismo extremo de **I**, alineados entre sí y hasta el borde.

Figura 28. *Aquí podemos observar cómo deben quedar pegados los componentes J al componente I.*

Si los componentes han quedado pegados de forma correcta y están bien alineados, entonces, van a funcionar como un pie para el componente **I**, incluso se podrá mantener erguido sin problemas.

Figura 29. *Los componentes J forman un pie para el componente I.*

El siguiente componente que utilizamos es el **H**. Este componente será usado para acoplar el soporte de la cámara con el servo. Debemos trazar una línea diagonal que usaremos como guía en los próximos pasos.

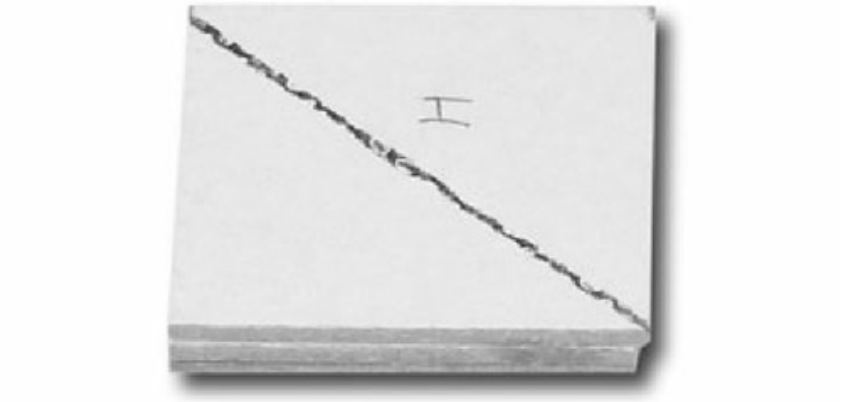


Figura 30. *Trazamos una línea diagonal en el componente H.*

Como en el caso del brazo, usaremos el elemento de actuación del servo para marcar los lugares en donde deben ir los tornillos. Los podemos marcar con la ayuda de un lápiz o de un plumón fino.

Figura 31. *Nos ayudamos con el elemento de actuación del servo para poder marcar los lugares donde se colocarán los tornillos.*

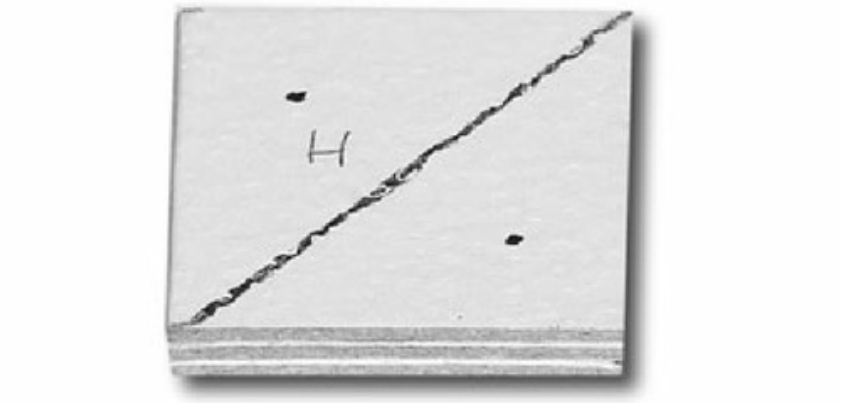


Figura 32. *En esta figura, observamos las marcas que nos indican dónde irán los tornillos.*

A continuación, debemos montar el elemento **I** de forma vertical sobre el elemento **H**. La línea que trazamos nos sirve de guía. Los elementos **J** nos permiten que **I** se mantenga de forma horizontal. Hay que colocar el pegamento a lo largo de la línea y en los componentes **J**.

Figura 33. *El componente I es pegado sobre el componente H, tal y como se muestra en la fotografía.*

Figura 34. *Aquí tenemos otra vista de la forma en que los componentes deben ser colocados.*

A continuación, podemos proceder a pintar el soporte de la cámara una vez que el pegamento haya secado en forma completa. Luego, colocamos el soporte en el segundo servo mediante unos tornillos.

Montaje de la cámara

Existen diferentes tipos de cámaras web en el mercado, por lo que no es posible describir el proceso para desarmar a cada una de ellas. Para poder montar la cámara, es necesario desarmarla y quedarnos únicamente con el circuito electrónico. Para esto, hay que ser cuidadoso y no dañar las lentes de la cámara. En la siguiente figura, podemos ver un ejemplo del circuito de la cámara.

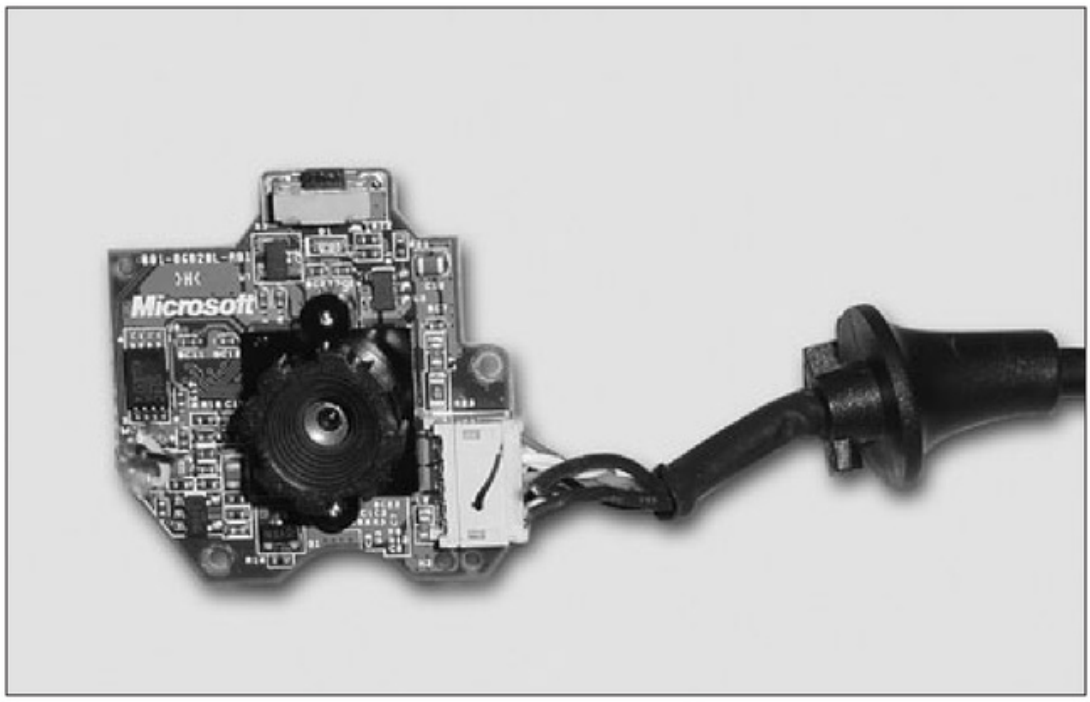


Figura 35. *Éste es el circuito de una cámara web. Lo montaremos sobre el componente I.*

Si lo deseamos, podemos colocar unos recortes de hule para evitar que la cámara quede pegada directamente al soporte. Ubicamos la cámara de tal forma que la lente quede sobre el eje de rotación X.

III EL USO DE ENGAGED

La propiedad de **Engaged** permite el movimiento del servo hacia la posición que tiene guardado el servo cuando posee el valor de **true**. Para evitar comportamientos erráticos del servo, es conveniente establecer en **true** el valor, solamente, cuando sabemos que es necesario llevar a cabo el movimiento y, en **false**, para los demás casos.

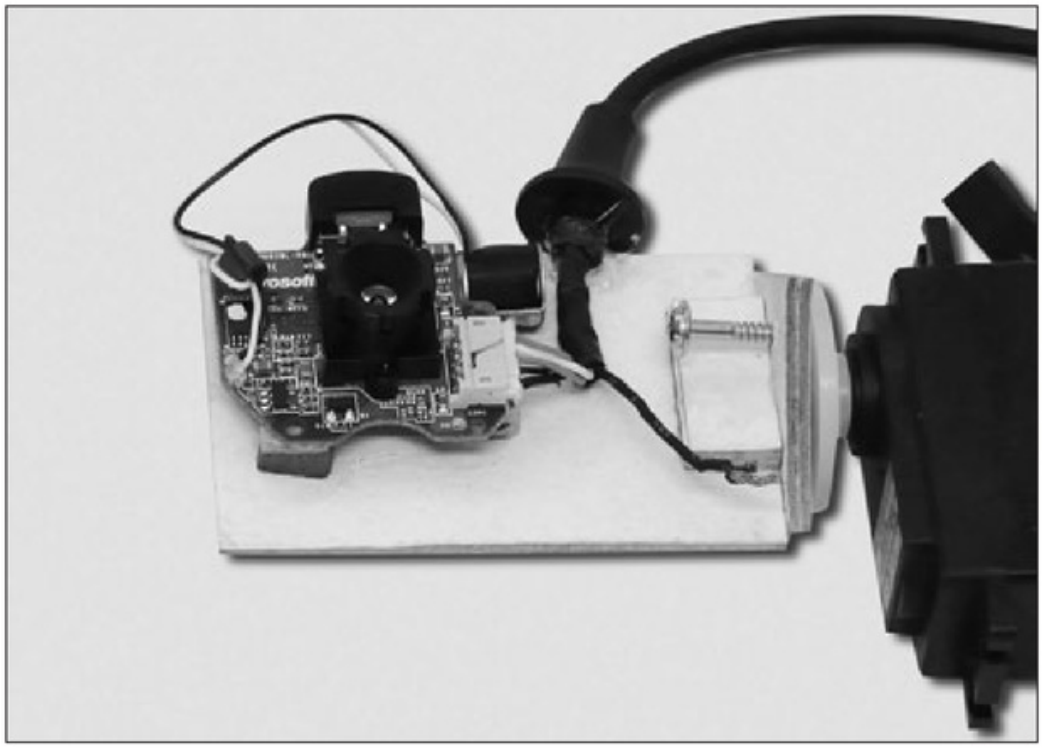


Figura 36. *Esta figura nos muestra cómo queda la cámara montada sobre su soporte, y éste, atornillado al servo.*

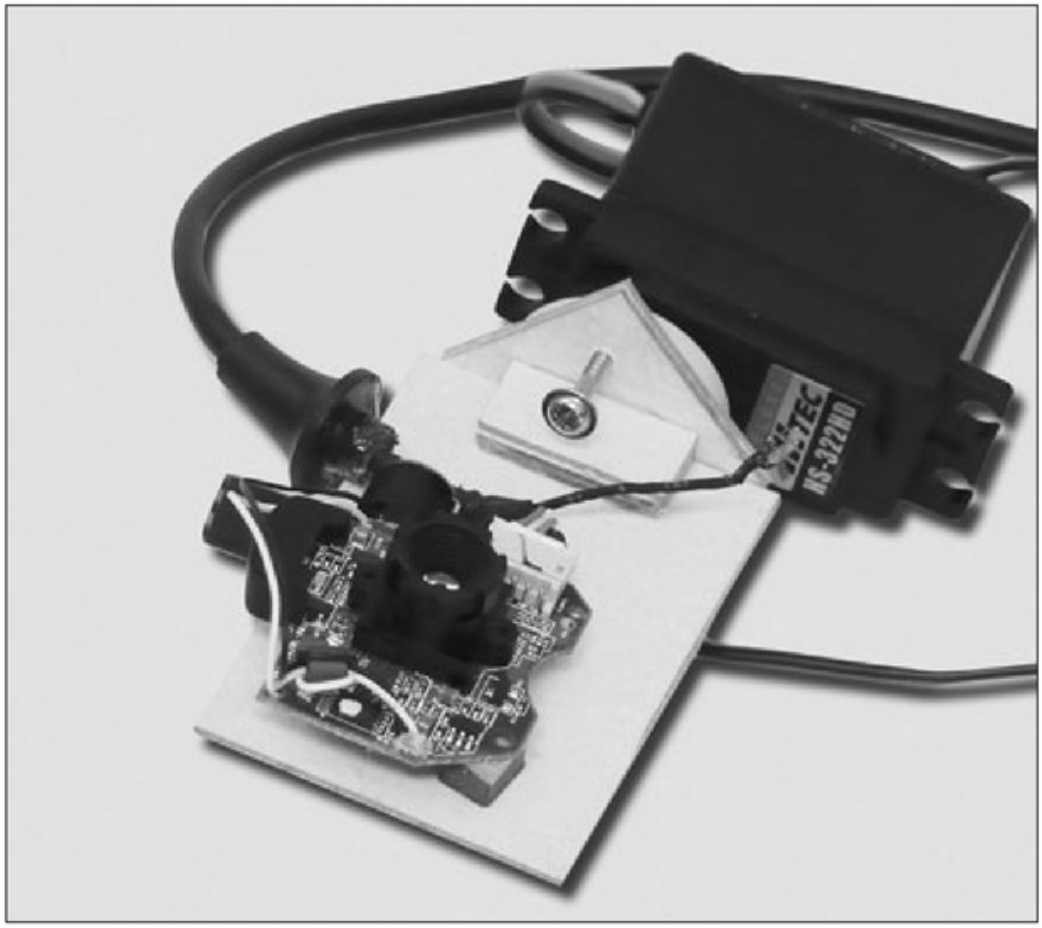


Figura 37. *Otra vista del montaje de la cámara sobre su soporte y el atornillado al servo.*

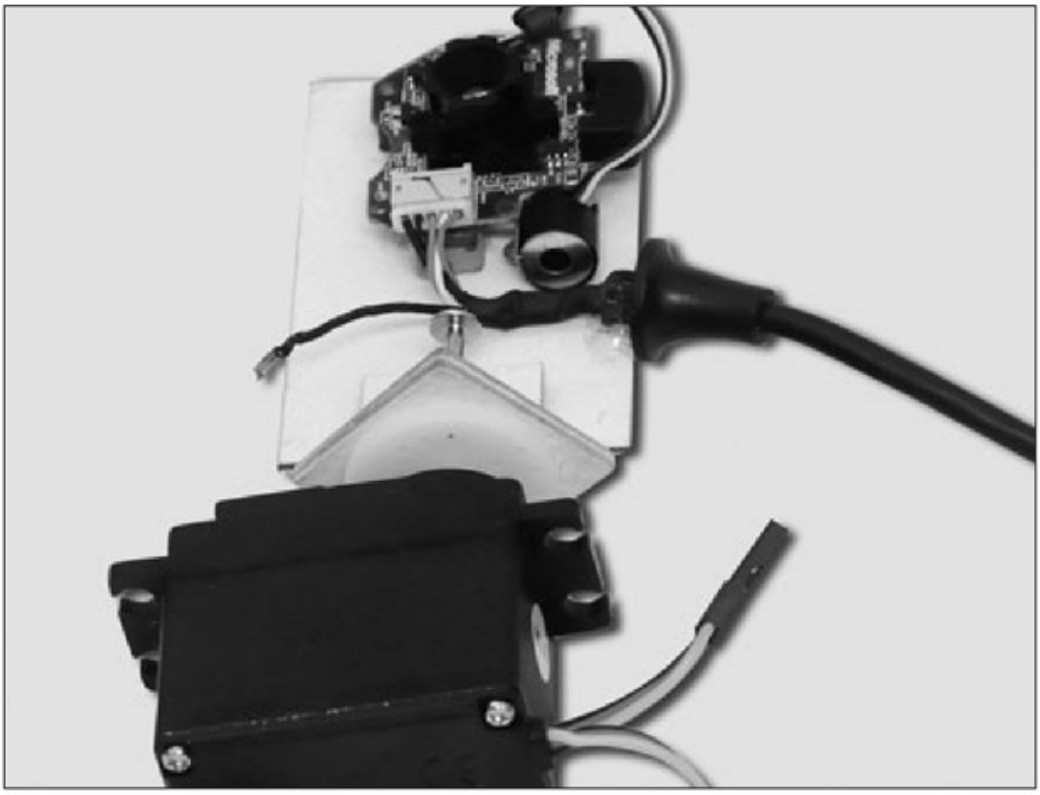


Figura 38. *Los tornillos no deben ajustarse de manera profunda, para permitir el libre movimiento del servo.*

Una vez que tenemos el soporte con la cámara montado sobre el servo, es posible colocar el servo sobre el brazo. En caso de ser necesario, podemos colocar una banda elástica para ayudar al servo a mantenerse en su lugar.

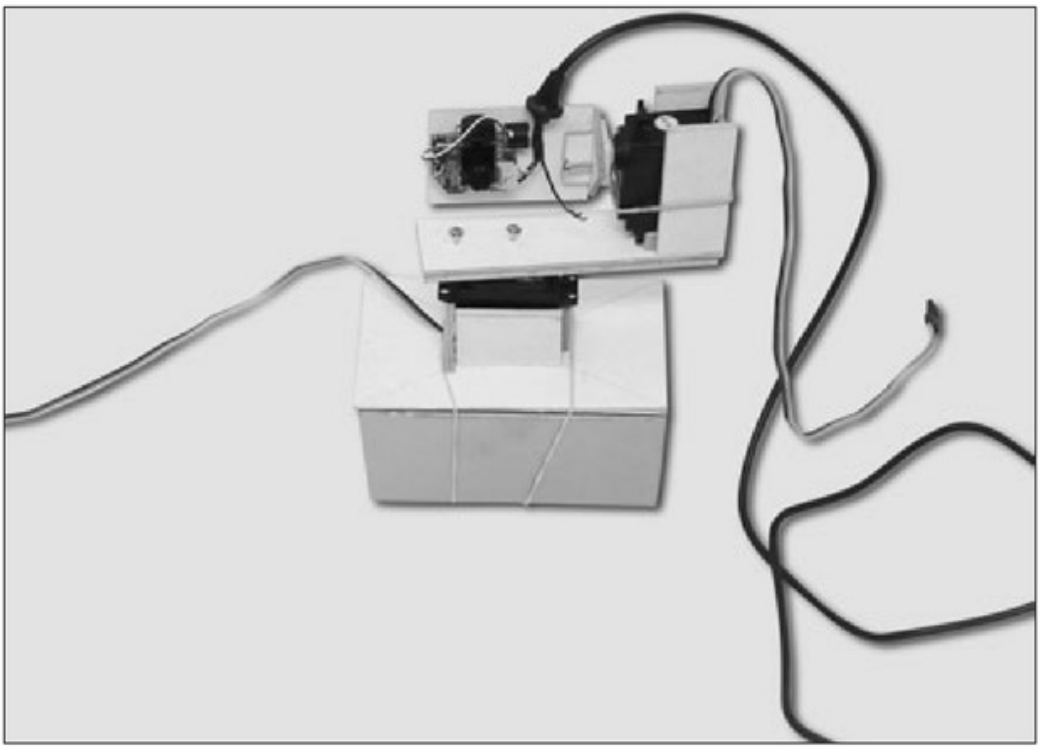


Figura 39. *Aquí podemos apreciar cómo el servo ha sido colocado sobre el brazo.*

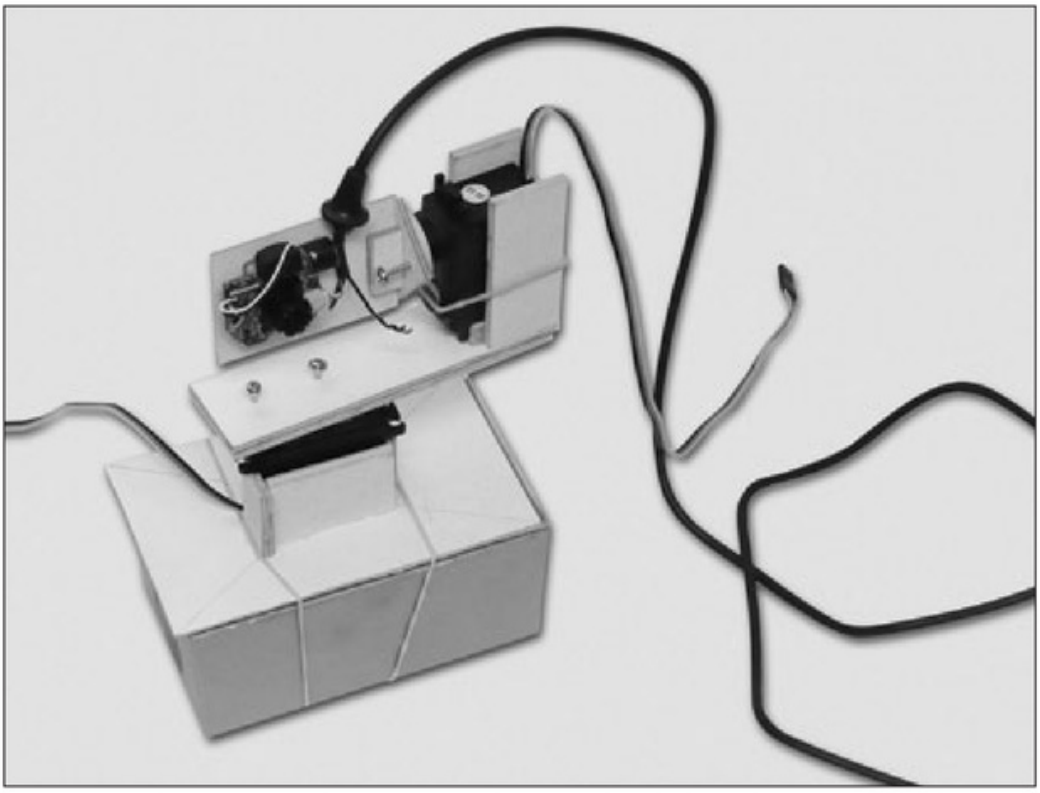


Figura 40. *Como observamos en la imagen, en caso de ser necesario, podemos recurrir a bandas elásticas.*

Para terminar el ensamble del robot, debemos conectar los servos al Phidget. Los servos deberán conectarse en las posiciones **0** y **1** para servos en el Phidget. El servo que rota en el eje Y se conecta en **0**, y el servo que rota en el eje X se conecta en **1**.

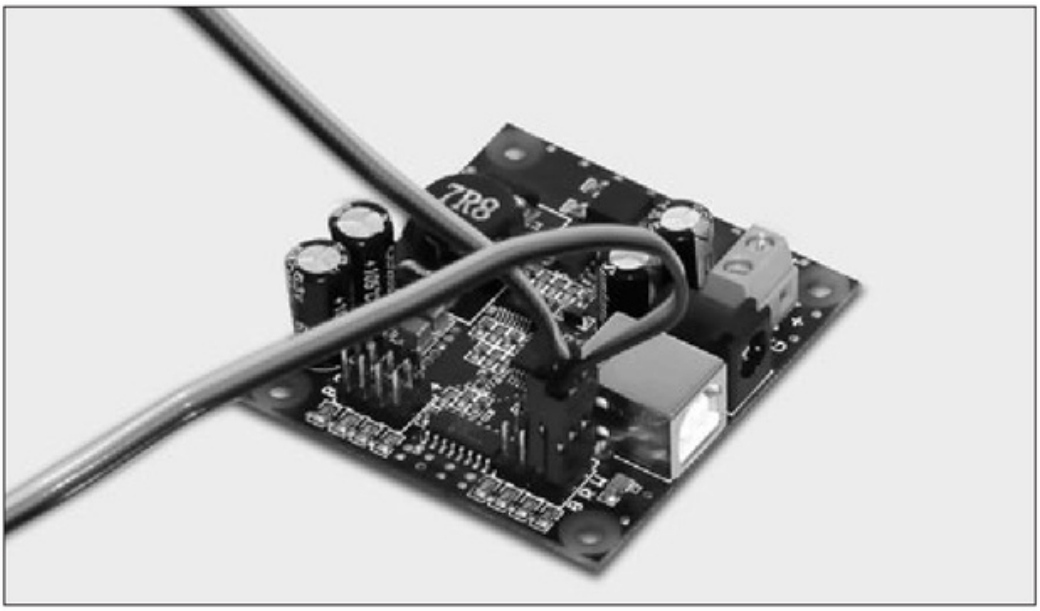


Figura 41. *Los servos se conectan al Phidget en los lugares indicados en la fotografía.*

Con el robot terminado, sólo falta desarrollar el software que lo controlará.

CREACIÓN DE LA APLICACIÓN

Ahora que ya tenemos el hardware terminado, es momento de empezar a realizar el software. El programa estará asentado en la aplicación base que desarrollamos en los capítulos anteriores. Lo mejor es tomar una copia de la aplicación y trabajar sobre ella; esto es necesario pues, en capítulos posteriores, la usaremos para desarrollar otros proyectos. Esta aplicación necesita código para cumplir con tres responsabilidades: control del Phidget, lectura de la webcam y procesamiento de la imagen. El control del Phidget ya es conocido por nosotros, y la aplicación calculará los valores de posición para ambos servos según sea necesario. Para la parte de lectura de la webcam y del procesamiento de la imagen, usaremos unas librerías del autor. No veremos a fondo estas librerías, sino sólo su uso especifico para este proyecto.

La primera librería contiene rutinas básicas para el manejo de **bitmaps**, de tal forma que podamos hacer el procesamiento digital de imágenes y llevar a cabo la visión por computadora. La otra librería encapsula a **DirectShow**, y la usamos para adquirir las imágenes de la webcam.

La interfaz de usuario

Empecemos por crear la interfaz de usuario. Ésta usará controles tradicionales, pero, también, un control propio que muestra las imágenes adquiridas por la webcam. A la aplicación base que contiene el código para controlar los servos, le hacemos unos cambios. En primer lugar, eliminamos todo lo relacionado con el Phidget 8/8/8 y, sólo, dejamos el código referente al control de los servos.

Como vamos a utilizar un control propio para la webcam, debemos adicionar las referencias respectivas a las librerías que usaremos: **Librerias.dll**, **NCDirectShow.dll**, **NCVideo.dll**. Como podemos ver a continuación, en el código de la forma debemos indicar el uso de los namespace correspondientes.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Imaging;
using System.Text;
using System.Windows.Forms;
using System.Threading;
```

```
// Colocamos nuestras librerias para video e imagenes
using NCVideo;
using NCDirectShow;
using NCBitmap;

// Necesario para el registro
using Microsoft.Win32;

// Necesario para los Phidgets
using Phidgets;
using Phidgets.Events;
```

Ahora, debemos crear la interfaz de usuario para la aplicación, la cual deberá lucir como en la figura que aparece a continuación.

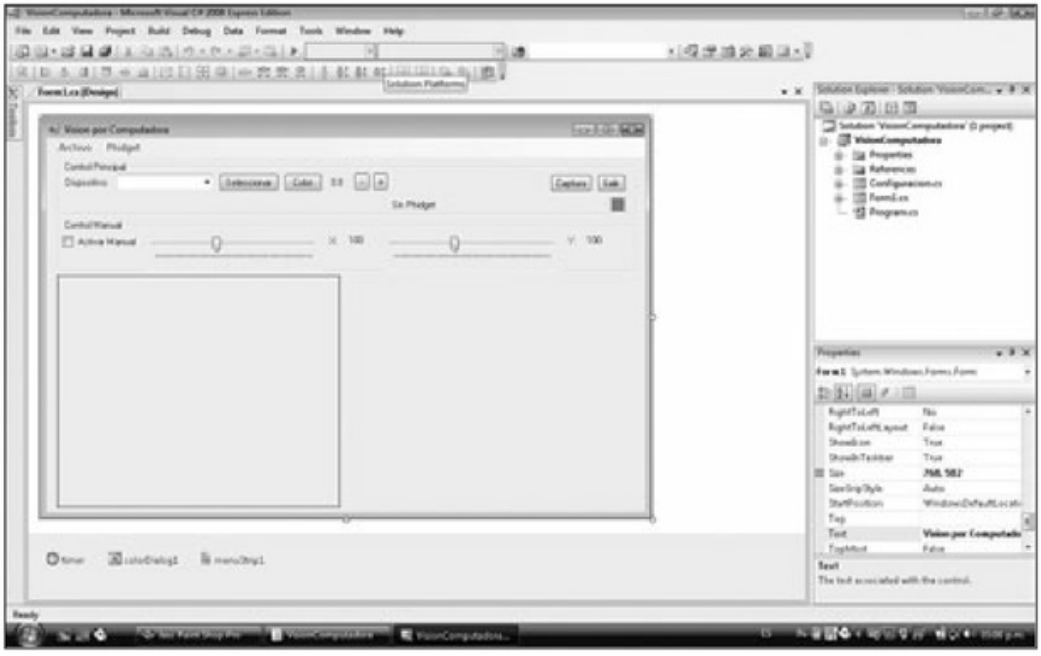


***Figura 42.** Aquí podemos observar la interfaz de usuario para la aplicación.*

A continuación, nos concentramos en modificar la forma. Lo primero que hacemos es cambiar su tamaño; para eso, en la propiedad **Size**, ingresamos el valor de **752, 500**. Esto nos da suficiente área para colocar las dos imágenes y todos los controles. En la propiedad **Text**, ingresamos el valor de **"Visión por Computadora"**, que es el título de nuestra ventana. Extendemos el **GroupBox** del Phidget y modificamos su valor en la propiedad **Text** a **"Control Principal"**; en esta zona, agregamos los controles más importantes de la aplicación. El tamaño nuevo es de **719, 67**. Movemos la etiqueta **lblPhidget1** a la locación **413, 48**, y el **TextBox**, a la locación **687, 45**. Con esto, ya tenemos suficiente espacio para agregar algunos controles nuevos. Vamos a ubicar los

controles de izquierda a derecha, más adelante, agregaremos el código que corresponde a cada uno. Empezamos insertando una etiqueta que contendrá el valor de **"Dispositivo:"**, en su propiedad **Text**. Junto a ella, disponemos un **ComboBox**, al cual le adjudicamos como nombre **comboDispositivos**. En este **ComboBox**, se listarán las webcams conectadas a la computadora, y, con un botón, seleccionaremos una de ellas como fuente de imágenes para nuestro programa.

A continuación, agregamos un botón. Este botón tiene por nombre **btnSeleccionar** y, en su propiedad **Text**, lleva el valor de **"Seleccionar"**. Usaremos este botón para seleccionar la cámara mostrada en el **ComboBox** como nuestra fuente de información. Junto a este botón, insertamos otro más pequeño. El nuevo botón se llama **btnColor** y, en la propiedad **Text**, ingresamos el valor **"Color"**. Con este botón, seleccionaremos el color que el robot reconocerá y seguirá. Usaremos un valor de tolerancia. Como podemos tener diferentes tonos de un mismo color, debemos buscar un rango de tonos que consideramos aceptables para el robot y que éste los pueda reconocer. La tolerancia la damos como un valor porcentual que abarcará de **0.0** a **1.0**. Para mostrar la tolerancia, usaremos una etiqueta y, para modificarla, un par de botones. Los botones controlarán el valor dentro de un rango apropiado para la aplicación.

La etiqueta se llama **lblTolerancia**, y le agregamos el valor de **"0.8"** en su propiedad **Text**. Este valor se encuentra relacionado con el valor inicial que asignaremos a la tolerancia. A su lado, insertamos un botón llamado **btnDecrementa**, que tiene **"-"** en su propiedad **Text**. Este botón será usado para disminuir el valor de la tolerancia. A la derecha, ubicamos un botón similar llamado **btnIncrementa**. La función de este botón será la de incrementar el valor de la tolerancia. En su propiedad **Text**, establecemos el valor **"+"**.

Solamente, nos faltan dos botones para esta sección de la interfaz. Colocamos del lado derecho un botón con el nombre **btnCaptura**, que lleva en su propiedad **Text** el valor de **"Captura"**. Con este botón, se iniciará la captura de las imágenes de la webcam para ser procesadas y para que el robot responda a ellas. En este botón en particular, debemos colocar el valor **False**, en la propiedad **Enabled**. Esto se realiza con el fin de evitar que se pulse antes de tener una webcam adquirida. El siguiente botón es muy sencillo. Lo ubicamos en el extremo derecho, como nombre le asignamos **btnSalir** y, en su propiedad **Text**, indicamos **"Salir"**. Con este botón, saldremos de nuestra aplicación.

Ahora vamos a agregar otro **GroupBox**, el cual contendrá los controles para que podamos mover al robot de forma manual. También, tendremos un **CheckBox** para indicar si el movimiento del robot será realizado de forma manual o automática.

A su lado, pondremos dos **TrackBars**, uno para cada eje. Cada servo mueve un eje del robot. El **CheckBox** lleva por nombre **chkManual** y, en su propiedad **Text**, establecemos el valor **"Activar Manual"**. Dejamos el **CheckBox** en su estado de default, es decir, sin ser verificado. Agregamos a su lado un **TrackBar** que tendrá por nombre **trackX**. Con este **TrackBar**, controlaremos la rotación en el eje de las X. En sus propiedades, debemos establecer el valor máximo como **210** y el mínimo de **30**. Éste es el rango para el modelo de servo que usamos en este proyecto, pero, si tenemos un modelo diferente, podemos establecer los valores que le correspondan. El tamaño del **TrackBar** será de **220** por **45**. La propiedad **Value** tendrá un valor de **100**.

Para poder ver el valor actual de la posición del servo, agregaremos un par de etiquetas. Una de ellas, simplemente, mostrará **"X:"** en su propiedad **Text**. En la otra etiqueta pondremos por nombre **lblX** y el valor de **"100"**, en su propiedad **Text**. Tenemos que llevar a cabo lo mismo para el eje Y. El **TrackBar** tendrá iguales valores, pero con el nombre de **trackY**, y la etiqueta con el valor numérico será **lblY**. El siguiente paso, consiste en insertar el control de tipo **ControlCamara**. Hay que recordar que este control es propietario, es decir, no es nativo de .Net, sino creado por el programador, y su clase existe dentro de la librería de **NCVideo**. Este control será creado, en forma manual, adicionando el código necesario a la aplicación.

Debemos dirigirnos al explorador de soluciones; en él encontraremos una sección que se llama **Form1.cs**. Al abrir esta sección, encontraremos una sección del código de la forma llamada **Form1.Designer.cs**. Esto lo podemos ver en la siguiente figura.

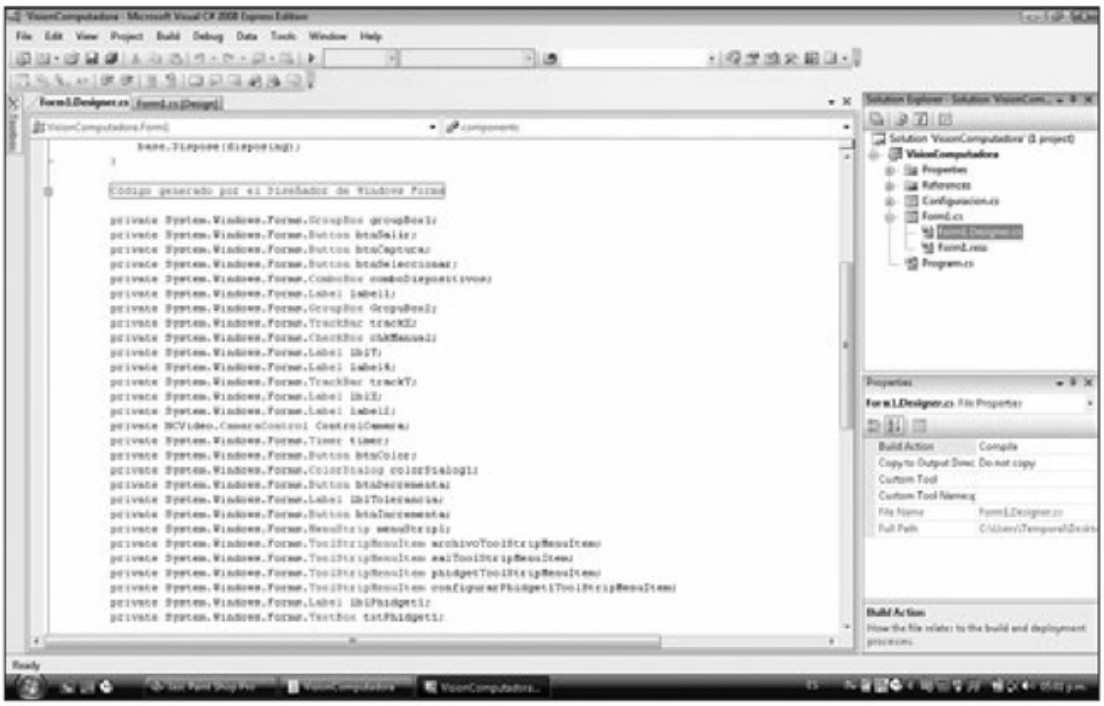


Figura 43. *En esta figura, se observa dónde encontramos la sección de código de la forma con la que trabajaremos.*

Efectuamos un doble clic con el mouse en **Form1.Designer.cs**, y aparece, en el editor, el código correspondiente. Éste es el código que define los controles que tendrá la forma y sus características. Este código es generado de manera automática por VisualStudio cuando agregamos controles, pero, en este caso, nosotros estableceremos el código del control manualmente. En el código, localizamos una región en donde se declaran las variables de los controles. Es fácil de encontrar, ya que todas las variables de controles aparecen listadas y son de acceso **private**. La figura anterior lo muestra. Al final de esta lista de variables, adicionaremos la que corresponde a nuestro control para la cámara.

```
        private NCVideo.CamaraControl ControlCamara;
```

Nuestra variable se llamará **ControlCamara** y es de tipo **CamaraControl**. Ya tenemos la variable, ahora es necesario instanciar el objeto. Localizamos la sección donde se lleva a cabo la instanciación de todos los controles y adicionamos el siguiente código.

```
            this.ControlCamara = new NCVideo.CamaraControl();
```

Ahora, al mismo nivel que los demás controles, agregamos el código que indica las propiedades de **CamaraControl**. Esta sección, también, se localiza por los comentarios para cada control, y vemos cómo se agregan valores a sus propiedades. El código que adicionamos es el siguiente.

```
            //
            // ControlCamara
            //
            this.ControlCamara.BackColor =
System.Drawing.SystemColors.Control;
            this.ControlCamara.Camara = null;
```

III EXPLORADOR DE SOLUCIONES

El explorador de soluciones nos permite ver el proyecto organizado por los documentos que lo componen. Lo encontramos dentro del menú **Ver**, en Visual Studio. Con éste, podemos seleccionar, con facilidad, la forma o código por editar. También, es posible ver las propiedades del proyecto.

```
this.ControlCamara.Location = new System.Drawing.Point(13, 172);
this.ControlCamara.Name = "ControlCamara";
this.ControlCamara.Size = new System.Drawing.Size(352, 288);
this.ControlCamara.TabIndex = 0;
```

Las propiedades más importantes que estamos estableciendo en la cámara son la locación de **13, 172**, y el tamaño de **352** por **288**. Con esto, ya tenemos los valores necesarios para el control. Esto último no es suficiente para usar el control, además, debemos registrarlo ante la forma para que ésta lo muestre y trabaje con él. Para ello, buscamos una sección que se presenta con los siguientes comentarios.

```
//
// Form1
//
```

Aquí veremos una parte donde se adicionan los controles a la forma, como podemos apreciar en el siguiente código.

```
this.Controls.Add(this.GropuBox2);
this.Controls.Add(this.groupBox1);
```

En esta parte, precisamente, adicionaremos el control de la cámara a la forma agregando la siguiente línea de código.

```
this.Controls.Add(this.ControlCamara);
```

Si regresamos a la vista con el diseño de la forma, veremos que aparece el control de cámara, representado por un rectángulo. Aún tenemos que adicionar otros controles

III CONTROLES QUE NO SE VEN EN LA FORMA

Algunos controles que se incluyen en la forma no se muestran directamente en ella, como por ejemplo, el **Timer**. Pero, cuando estamos editando la forma, estos aparecen listados en la parte inferior del editor. Para editarlos, primero efectuamos un clic derecho en esa sección y, a continuación, seleccionamos sus propiedades.

a la forma. Necesitamos de un **Timer** que nos servirá para llevar a cabo el procesamiento de la imagen cada determinado tiempo. Para agregar el **Timer**, es suficiente con arrastrar el control de **Timer** que se encuentra en la caja de herramientas, sobre la forma. Al **Timer**, no le cambiaremos el nombre, pero más adelante haremos uso de su evento.

En este momento, necesitamos adicionar otro control conocido como **ColorDialog**. Este control es un cuadro de diálogo que nos permite seleccionar un color. Lo usaremos para que el usuario pueda elegir el color que desea que el robot siga con su visión. Al igual que con el **Timer**, basta con arrastrarlo sobre la forma.

Con esto, hemos finalizado de construir la interfaz de usuario; ahora, deberemos adicionar código a nuestra aplicación. El código que ya conocemos de los capítulos anteriores no será comentado, únicamente, nos enfocaremos en el código nuevo. La forma necesitará de datos actuales. Éstos se encuentran relacionados en especial con la parte del video y la visión por computadora. En la forma, adicionamos entonces los siguientes datos.

```
        // Variables necesarias video
        private string dispositivo;          // Dispositivo seleccionado
        private FilterCollection filtros;    // Lista de dispositivos
        private NCBitMap miBitmap;           // Bitmap de trabajo
        private byte objetoR, objetoG, objetoB;   // Color a buscar
        private double tolerancia;           // Tolerancia del reconocimiento
        private int xfoco, yfoco;
        private int xcentral, ycentral;


        // Variables necesarias Phidget
        private int numeroSerie1;        // Numero de serie del phidget 1
        private bool conectado1;         // Para saber si hay conexion con
el phidget 1
        private AdvancedServo mServo;  // Controlador de servos
```

Tenemos en primer lugar una cadena, la cual guardará el nombre del dispositivo, en este caso, la webcam seleccionada. Contamos, también, con una variable llamada **filtros**, en la cual disponemos de una colección de los dispositivos conectados a la computadora. La clase **NCBitMap** contiene las funcionalidades necesarias para que podamos trabajar con el **BitMap**. En principio, necesitaremos poder leer y escribir un pixel. Tendremos un **Bitmap** de trabajo al cual llamamos **miBitmap**. Sobre él, llevaremos a cabo el trabajo. Como el robot seguirá un objeto de color,

necesitamos definir las variables que nos servirán para guardar dicho color. Las variables son **objetoR**, **objetoG**, **objetoB**.

En otros apartados, ya hemos hablado de la tolerancia y, ahora, definiremos la variable que usaremos para guardar su valor. Las variables **xfoco** e **yfoco** serán utilizadas para indicar las coordenadas del centro geométrico del área en pantalla, que contiene el color que seguimos. Las otras variables, **xcentral** e **ycentral**, guardarán las coordenadas del pixel central del **BitMap**. En el constructor de la forma, ingresamos el código de inicialización.

```
public Form1()
{
    InitializeComponent();

    #region Video
    // Ponemos un color de default
    objetoR = 255;
    objetoG = 28;
    objetoB = 25;

    // Ponemos el valor inicial de sensibilidad
    tolerancia = 0.80;

    xfoco = 0;

    yfoco = 0;
    xcentral = 0;
    ycentral = 0;

    // Buscamos los dispositivos
    try
```

III EL COLOR RGB

En la computadora, podemos definir cualquier color por medio de una combinación de rojo, verde y azul. Al variar la cantidad que contribuye cada uno de estos colores básicos, obtenemos cualquier color que puede ver el ojo humano. Si los tres componentes de color poseen el mismo valor, obtenemos un tono de gris.

```
            {
                // Nuestros filtros son los dispositivos de entrada de video
                filtros = new FilterCollection(FilterCategory.
VideoInputDevice);

                // Si no hay dispositivos mandamos una excepcion
                if (filtros.Count == 0)
                    throw new ApplicationException();


                // Adicionamos los dispositivos encontrados al combo
                foreach (Filter filter in filtros)
                {
                    comboDispositivos.Items.Add(filter.Name);
                }
            }
            catch (ApplicationException)
            {
                // Indicamos que no hay dispositivos
                comboDispositivos.Items.Add("No hay dispositivos de
captura de video");

            }

            // Seleccionamos el primer dispositivo en la lista como el
default
            comboDispositivos.SelectedIndex = 0;
            #endregion

            #region Phidget

            // Empezamos como si no estuvieramos conectados
            conectado1 = false;

            // Colocamos un valor de default para el numero de serie

            numeroSerie1 = 0;

            // Verificamos si existe la llave en el registro
            RegistryKey sfwKey = Registry.LocalMachine.OpenSubKey
```

```
("Software");
            RegistryKey redKey = sfwKey.OpenSubKey("RedUsers3");

            // Verificamos si existe
            if (redKey == null) // la llave no existe
            {

                // Creamos la llave
                sfwKey = Registry.LocalMachine.OpenSubKey("Software", true);
                redKey = sfwKey.CreateSubKey("RedUsers3");

                RegistryKey robotKey0 = redKey.CreateSubKey("Robots3");

                // Colocamos un valor de default
                robotKey0.SetValue("NumeroSerie1", (object)numeroSerie1);

            }
            else // la llave existe
            {
                RegistryKey robotKey0 = redKey.OpenSubKey("Robots3");

                // Obtenemos el numero de serie guardado
                numeroSerie1 = (int)robotKey0.GetValue("NumeroSerie1");
            }

            #endregion

        }
```

Estamos usando el constructor para llevar a cabo las inicializaciones necesarias. En primer lugar, ingresaremos un color de default para el seguimiento que realizará el robot. En este caso, asignamos un tono de rojo. Necesitamos tres variables, una para cada componente de color. Luego, establecemos el valor inicial de la tolerancia, el cual será de **0.80**. Este valor es adecuado para la mayoría de los casos, pero siempre será posible cambiarlo durante la ejecución del programa. Las variables relacionadas con el foco y el centro de la pantalla se inicializan en cero y serán calculadas con posterioridad.

El siguiente código se encarga de buscar las webcams disponibles y, como es posible tener problemas, lo colocamos en un **try**. Dentro de este **try**, empezamos

por buscar todos los dispositivos que sean entradas de video; éstos quedarán guardados dentro de filtros. Si no tuviéramos ningún dispositivo, es decir, si la cantidad de filtros fuera **0**, lo mejor sería levantar una excepción y no continuar con el programa. Si intentamos hacer operaciones de video sin ningún dispositivo, el programa fallará. En la interfaz de usuario, hemos insertado un **ComboBox**, el cual mostrará la lista de dispositivos. Para rellenar el **ComboBox**, usamos el ciclo **foreach** y, por cada dispositivo encontrado en filtros, se adiciona un elemento en el **ComboBox**. Si el **try** falla en el **catch**, simplemente, mandamos un mensaje al usuario por medio del **ComboBox**. De esta forma, el usuario no podrá seleccionar un dispositivo de video. Por default, estableceremos como seleccionado el primer elemento del **ComboBox**; de esta forma, si sólo hay una cámara, quedará seleccionada para el usuario. Ahora, creamos el handler para el botón **btnSalir**.

```
        private void btnSalir_Click(object sender, EventArgs e)
        {
            // Apagamos el timer
            timer.Stop();

            // Cerramos los dispositivos
            this.CerrarDispositivo();

            // Cerramos la aplicacion
            this.Close();
        }
```

El código resulta muy sencillo. En primer lugar, detenemos el **Timer**. Luego, se invoca al método **CerrarDispositvo()**, el cual se encarga de parar y desconectar la webcam. Por último, procedemos a cerrar la aplicación. A continuación, seguimos con el handler del botón **btnSeleccionar**.

```
        private void btnSeleccionar_Click(object sender, EventArgs e)
        {
            // Obtenemos el nombre del dispositivo seleccionado en el combo
            dispositivo = filtros[comboDispositivos.SelectedIndex].
MonikerString;

            // Creamos la fuente de video
            DispositivoVideo fuenteVideo = new DispositivoVideo();
```

```
            fuenteVideo.FuenteVideo = dispositivo;

            // Lo abrimos
            AbrirVideo(fuenteVideo);

            // Habilitamos el boton de captura
            btnCaptura.Enabled = true;
        }
```

Con este handler, en primer lugar, obtenemos el índice de la webcam que se encuentra seleccionada en el **ComboBox**, y lo marcamos en la colección de filtros. Creamos, entonces, una fuente de video y asignamos este dispositivo a esa fuente de video. Procedemos a invocar el método que estará a cargo de abrir el video, pasando como parámetro la fuente de video actual que ha sido seleccionada. Como ya tenemos una fuente de video, entonces, es posible habilitar el botón de **btnCaptura** para que el usuario, si así lo desea, pueda iniciar el proceso de reconocimiento visual del robot. El código del método **AbrirVideo()** es el siguiente.

```
        // Abrir la fuente de video
        private void AbrirVideo(IFuenteVideo fuente)
        {
            // Colocamos el cursor de espera
            this.Cursor = Cursors.WaitCursor;

            // Cerramos el anterior
            CerrarDispositivo();

            // Creamos camara
            Camara camera = new Camara(fuente);

            // Iniciamos camara
            camera.IniciaCaptura();

            // Pegamos la camara al control
            ControlCamara.Camara = camera;

            // Regresamos al cursor de default
            this.Cursor = Cursors.Default;
        }
```

En este método, comenzamos por mostrar el cursor del reloj de arena, para indicarle al usuario que debe esperar hasta que la fuente de video esté abierta.

Por seguridad, cerramos cualquier dispositivo que pudiera estar en uso. Luego, creamos un objeto de tipo **Cámara**, usando la fuente como el dispositivo que proveerá de información a dicha cámara. Le indicamos a la cámara que puede empezar su captura y la asignamos al control para que ésta pueda ser mostrada en la forma. Para finalizar, con la figura de la flecha, le indicamos al cursor que regrese. El siguiente método que agregamos es el de **CerrarDispositivo()**. Con este método, detenemos la captura de la cámara.

```
private void CerrarDispositivo()
{
    // Obtenemos la camara del control
    Camara camera = ControlCamara.Camara;

    // Si existe la camara procedemos a cerrarla
    if (camera != null)
    {
        // Quitamos la camara del control
        ControlCamara.Camara = null;

        // Enviamos senal para parar la camara
        camera.Parar();

        // Esperamos a la camara
        camera.Parando();

        camera = null;
    }
}
```

III ESPERAR EL VIDEO

Una vez que se ha presionado el botón **Seleccionar**, debemos esperar hasta que se muestre la imagen de video y el cursor regrese a la figura de la flecha, antes de pulsar el botón **Captura**. En caso contrario, la aplicación puede fallar, pues intentará llevar a cabo el reconocimiento en un dispositivo que, quizás, aún no se encuentre listo.

El primer paso, en este método, consiste en obtener la cámara que se encuentra conectada al control, es decir, con la que estamos conectados. Si el valor de esta cámara es **null**, significa que no estamos conectados y no llevamos a cabo las funciones de paro. Si el valor no es **null**, entonces sí, podemos proceder a parar la cámara. Para detener la cámara, debemos quitarla del control. Por eso, asignamos el valor **null** al control de la cámara. Luego, invocamos el método **Parar()** de la cámara y, con el método **Parando()**, esperamos hasta que ésta se detenga en su captura. Por seguridad, asignamos **null** a nuestra variable de trabajo.

Ahora, creamos el handler del **Timer** y agregamos el siguiente código en él.

```
private void timer_Tick(object sender, EventArgs e)
{
    // Obtenemos el BMP desde el cuadro de video
    ObtenBMP();

    // Eliminamos lo que no sea de ese color
    ReductorColor(objetoR, objetoG, objetoB, tolerancia);

    // Actuamos si hay bitmap valido
    if (miBitmap != null)
        if (miBitmap.BitMap != null)
        {
            // Obtenemos el contexto de dispositivo
            Graphics clienteDC = this.CreateGraphics();

            // Sincronizamos acceso a los bitmaps
            Monitor.Enter(miBitmap);

            // Copiamos el bitmap a la forma
            clienteDC.DrawImage(miBitmap.BitMap, new
```

III CREACIÓN DEL HANDLER DEL TIMER

Para crear el handler del **Timer**, disponemos de dos opciones. La primera es efectuar un doble clic sobre el icono del **Timer** en la forma. La segunda, es por medio de la ventana de propiedades, seleccionando el listado de eventos para luego marcar ahí el evento. Esto nos sitúa directamente en el código del handler.

```
Rectangle(370,
                                                                    172,

(int)(miBitmap.Width * miBitmap.Zoom),

(int)(miBitmap.Height * miBitmap.Zoom)));

                    Monitor.Exit(miBitmap);
                }

            Seguimiento();

        }
```

Este handler se llevará a cabo cada vez que el **Timer** dé un tic que, en nuestro caso, es cada 33 milisegundos. Lo primero que hace es invocar al método **ObtenBMP()**, el cual veremos más adelante; este método copia la imagen actual de la cámara a **miBitmap** para poder ser procesada. Luego, se llama al método **ReductorColor()**. Este método es el que lleva a cabo el reconocimiento basado en el color y encuentra el foco. Como ya tenemos el **BitMap** capturado, copiado y procesado, entonces, podemos proceder a mostrarlo en la forma. Primero, es necesario verificar que **miBitmap** se encuentre instanciado y que el **BitMap** que tiene en su interior también lo esté. Si todo se encuentra en forma correcta, entonces, obtenemos el **Device Context** de la forma. Para evitar problemas, sincronizamos el **BitMap**, para que nadie más intente acceder a él mientras lo colocamos en la forma. Por medio de la función **DrawImage()**, procedemos a copiar **miBitmap** a la forma. El primer parámetro es el bitmap que queremos copiar; el segundo parámetro es un rectángulo que indica las coordenadas donde se copiará y las dimensiones correspondientes. Una vez terminada la copia del bitmap a la forma, podemos liberar al bitmap. Para finalizar, el handler invoca al método **Seguimiento()** que, sobre la base de los valores calculados en **ReductorColor()**, mueve los servos para posicionar la cámara.

El siguiente método es avanzado y se encarga de copiar la imagen contenida en la cámara a **miBitmap**. Para esto, usaremos apuntadores. Como utilizamos un lenguaje administrado, esto se considera código no seguro, lo cual no significa que sea peligroso para la computadora, sino que seremos nosotros quienes controlaremos la responsabilidad del acceso a memoria y no, el runtime de C#.

```
        private void ObtenBMP()
         {
```

```
            // Copiamos del cuadro de la camara a nuestro Bitmap

            // Creamos la instancia si no existe el Bitmap todavia
            if (miBitmap == null && ControlCamara.Camara.Capturando)
            {
                // Creamos las instancias con las dimensiones del cuadro
de la camara
                miBitmap = new NCBitMap(ControlCamara.Camara.Cuadro.Width,
ControlCamara.Camara.Cuadro.Height);
            }

            // Si ya existe el Bitmap y estamos capturando llevamos a cabo
la copia
            if (ControlCamara.Camara.Capturando && miBitmap != null)
            {
                // Obtenemos la camara del objeto ControlCamara
                Camara camera = ControlCamara.Camara;

                // Hacemos Lock para utilizar a la camara
                camera.Lock();

                // Obtenemos las dimensiones del cuadro de la camara
                int w = camera.Cuadro.Width;
                int h = camera.Cuadro.Height;

                // Sincronizamos el acceso al Bitmap
                Monitor.Enter(miBitmap);

                #region Copia del cuadro

                // Creamos un bitmap temporal de trabajo
                Bitmap temp;

                // Clonamos el cuadro de la camara al temporal
                temp = (Bitmap)camera.Cuadro.Clone();

                // Creamos variables para la informacion fuente y destino
                BitmapData infoFuente, infoDestino;

                // Bloqueamos los bits
```

```
                infoFuente = temp.LockBits(new Rectangle(0, 0, w, h),
ImageLockMode.ReadOnly, PixelFormat.Format24bppRgb);
                infoDestino = miBitmap.BitMap.LockBits(new Rectangle(0, 0,
w, h), ImageLockMode.ReadWrite, PixelFormat.Format24bppRgb);

                // Obtenemos dimensiones
                int width = infoFuente.Width;
                int height = infoFuente.Height;

                // Obtenemos el offset para cada imagen en bytes
                int srcOffset = infoFuente.Stride - width * 3;
                int dstOffset = infoDestino.Stride - width * 3;

                unsafe
                {
                    // Nos colocamos al inicio de cada Bitmap
                    byte* src = (byte*)infoFuente.Scan0.ToPointer();
                    byte* dst = (byte*)infoDestino.Scan0.ToPointer();

                    // Recorremos la imagen
                    for (int y = 0; y < height; y++)
                    {
                        // Avanzamos por pixel, leyendo de 3 en 3 bytes
                       for (int x = 0; x < width; x++, src += 3, dst += 3)
                        {
                            dst[0] = (byte)(src[0]);
                            dst[1] = (byte)(src[1]);
                            dst[2] = (byte)(src[2]);
                        }
                        src += srcOffset;
                        dst += dstOffset;
                    }

                    // Liberamos los bitmaps
                    temp.UnlockBits(infoFuente);
                    miBitmap.BitMap.UnlockBits(infoDestino);
                }

                #endregion
```

```
                // Desechamos el Bitmap temporal
                temp.Dispose();

                // Desbloqueamos la camara
                camera.Unlock();

                // Salimos de la sincronizacion
                Monitor.Exit(miBitmap);
            }

        }
```

Dentro del método, lo primero que hacemos es revisar si no existe una instancia de **miBitmap**, y la cámara ya está capturando. Si no existiera la instancia, la creamos otorgándole al bitmap el mismo tamaño que la imagen que captura la cámara. Si el bitmap ya se encuentra instanciado, procedemos a la copia. Para copiar el bitmap, empezamos por asignar la cámara a una variable temporal de trabajo que nos ayudará en el proceso. Esta variable se llamará **cámara**. Procedemos a bloquearla para que nadie más tenga acceso y pueda llevarnos a un problema de corrupción de información. Luego, obtenemos las dimensiones del cuadro de la cámara y procedemos a sincronizar a **miBitmap** para también tener acceso a él de forma segura. Como estamos usando apuntadores, es necesario realizar este paso.

Creamos ahora un bitmap temporal de trabajo llamado **temp**. En **Temp.**, guardaremos un clonado del cuadro actual que tenga la webcam. Luego, creamos dos variables de tipo **BitmapData**: una para la información fuente, es decir, la imagen clonada de la cámara, y otra para el destino, que es nuestro bitmap. Obtenemos las dimensiones de la fuente y, luego, calculamos el Offset de cada imagen. Tenemos que multiplicar por 3, debido a que usamos un byte para el rojo, otro para el verde y otro para el azul. Entonces, empezamos con el código no seguro. Lo primero que hacemos es obtener un apuntador al inicio de cada uno de los

III OPTIMIZACIÓN DEL CÓDIGO

El código que presentamos en la presente obra, quizás, no sea el más optimizado, pero lo ingresamos de esta manera para hacerlo lo más fácil posible de comprender. Después de esto, podemos empezar a optimizar el código o a adicionarle otras características al programa.

bitmaps. Luego, por medio de dos ciclos anidados, recorremos todos los pixeles del bitmap. Debemos tener en cuenta que las coordenadas avanzan de uno en uno, pero las direcciones de memoria deben avanzar de tres en tres, debido a los tres bytes necesarios para guardar el color. En la parte interior del ciclo, copiamos el pixel actual de la fuente al destino en la misma locación. El código por afuera del ciclo x nos permite avanzar al siguiente renglón de la imagen. Cuando terminamos la copia, simplemente liberamos a los bitmaps. Ahora, ya tenemos un bitmap que contiene la información de la cámara y podemos procesarlo por medio de las funciones de la clase **NCBitmap**.

En este momento, podemos hacer el handler para el botón **btnCaptura**.

```
        private void btnCaptura_Click(object sender, EventArgs e)
        {
            // Iniciamos la captura desde la camara

            // Iniciamos el timer
            timer.Start();

            // Forzamos el redibujo de la forma
            this.Invalidate();
        }
```

El código es muy sencillo. En primer lugar, iniciamos el timer para que el proceso de captura y seguimiento comience, y, luego, enviamos a redibujar la forma.

Ahora, estableceremos el método llamado **ReductorColor()**. Este método es importante pues localiza dónde está el foco de la imagen y, también, hace modificaciones al bitmap para presentarlo en pantalla.

```
        public void ReductorColor(byte rs, byte gs, byte bs, double
porcentaje)
        {
            // Sincronizamos los Bitmaps
            Monitor.Enter(miBitmap);

            // Bloqueamos los bitmaps
            miBitmap.LockBits();
```

```
            int x = 0, y = 0;

            byte r = 0, g = 0, b = 0;
            double parecido = 0.0;

            // Variables para la zona con el color
            int xmin = miBitmap.Width;
            int xmax = 0;

            int ymin = miBitmap.Height;
            int ymax = 0;

            // Variables para el centro de la imagen
            xcentral = miBitmap.Width / 2;
            ycentral = miBitmap.Height / 2;

            // Recorremos el Bitmap
            for (x = 0; x < miBitmap.Width; x++)
                for (y = 0; y < miBitmap.Height; y++)
                {
                    // Obtenemos el color del pixel
                    miBitmap.GetPixelFast(x, y, ref r, ref g, ref b);

                   // Comparamos con el color ofertado
                   parecido = NCBitMap.Colorabilidad(rs, gs, bs, r, g, b);

                    // Verificamos si pasa el gatillo
                    if (parecido >= porcentaje)
                    {
                        miBitmap.SetPixelFast(x, y, 255 - r, 255 - g,
255 - b);
                        // Actualizamos valores
                        if (x < xmin)
                            xmin = x;
                        if (x > xmax)
                            xmax = x;
                        if (y < ymin)
                            ymin = y;
                        if (y > ymax)
                            ymax = y;
```

```
            }


        }

    // Dibujamos las lineas
    for (x = 0; x < miBitmap.Width; x++)
    {
        miBitmap.SetPixelFast(x, ymin, 255, 0, 0);
        miBitmap.SetPixelFast(x, ymax, 255, 0, 0);

    }
    for (y = 0; y < miBitmap.Height; y++)
    {
        miBitmap.SetPixelFast(xmin, y, 255, 0, 0);
        miBitmap.SetPixelFast(xmax, y, 255, 0, 0);

    }

    // Obtenemos el foco
    xfoco = ((xmax - xmin) / 2) + xmin;
    yfoco = ((ymax - ymin) / 2) + ymin;

    // Dibujamos el foco
    for (x = xfoco - 5; x < xfoco + 5; x++)
    {
        miBitmap.SetPixelFast(x, yfoco - 5, 0, 255, 0);
        miBitmap.SetPixelFast(x, yfoco + 5, 0, 255, 0);

    }

    for (y = yfoco - 5; y < yfoco + 5; y++)
    {
        miBitmap.SetPixelFast(xfoco - 5, y, 0, 255, 0);
        miBitmap.SetPixelFast(xfoco + 5, y, 0, 255, 0);
    }

    // Dibujamos el centro de la imagen
    for (x = xcentral - 5; x < xcentral + 5; x++)
    {
        miBitmap.SetPixelFast(x, ycentral - 5, 255, 255, 0);
```

```
                miBitmap.SetPixelFast(x, ycentral + 5, 255, 255, 0);


            }

            for (y = ycentral - 5; y < ycentral + 5; y++)
            {
                miBitmap.SetPixelFast(xcentral - 5, y, 255, 255, 0);
                miBitmap.SetPixelFast(xcentral + 5, y, 255, 255, 0);
            }


            // Desbloqueamos los bitmaps
            miBitmap.UnlockBits();

            // Salimos de la sincronizacion
            Monitor.Exit(miBitmap);

        }
```

El método recibe cuatro parámetros. Los tres primeros son los componentes rojo, verde y azul del color que estamos buscando en pantalla. El cuarto parámetro es el valor de tolerancia que, en este caso, llamamos porcentaje. Dentro del método, empezamos por sincronizar al bitmap para poder trabajar con él. Luego, simplemente, lo bloqueamos. Esto nos permitirá acceder a la información de sus pixeles de forma segura. Creamos algunas variables de trabajo que nos permitirán realizar el trabajo de manera más fácil. Las primeras están relacionadas con las coordenadas, y las siguientes, con el color. El método creará un rectángulo que englobe a todos los pixeles que tengan un color similar al que buscamos, dentro del rango de la tolerancia. Las coordenadas de los lados se calcularán según se recorra la imagen y se encuentren los pixeles. Las variables **xmin**, **xmax**, **ymin** e **ymax** serán utilizadas para esto. Posteriormente, calculamos el punto central de la imagen, en un cálculo que resultará muy sencillo de hacer.

Tenemos dos ciclos que se encuentran anidados y nos permiten recorrer todos los pixeles de la imagen. Para cada pixel, obtenemos su información de color por medio de la función **GetPixelFast()**. Los primeros dos parámetros de la función son las coordenadas del pixel al cual le queremos leer el color. Los siguientes tres parámetros son las variables donde guardaremos el valor de rojo, verde y azul del pixel. Al tener el color del pixel y el color que deseamos seguir, podemos compa-

rarlos. La función **Colorabilidad()** nos regresa un porcentaje del parecido de los colores. Si la similitud de los colores es mayor o igual a la tolerancia que deseamos, entonces, procedemos a actualizar las variables que definen el área donde se encuentran pixeles del color que buscamos. Como parte de la interfaz, asignamos al pixel con un color que es su complemento; esto con el fin de poder ver, de manera gráfica, los pixeles encontrados. Para escribir un pixel en el bitmap, usamos la función **SetPixelFast()**; los dos primeros parámetros de la función son las coordenadas del pixel que deseamos escribir. Los tres siguientes parámetros son los valores de rojo, verde y azul para ese pixel.

A continuación, tenemos el código que dibujará dos líneas horizontales y dos líneas verticales de color rojo. Con ellas, generamos un rectángulo que encierra el área donde están los pixeles con el color que buscamos. El foco será el centro del área donde se encuentran los pixeles con el color. Tenemos dos fórmulas que lo calculan de manera sencilla. Luego, se dibuja un pequeño cuadro de color rojo alrededor del foco. Esto con el fin de mostrar, en la forma, la ubicación en donde la computadora lo ha encontrado. De manera similar, se procede a dibujar un recuadro de color amarillo alrededor del centro de la imagen. Cuando terminamos, desbloqueamos el bitmap y salimos de la sincronización. Procedemos a crear el handler para el botón de **btnColor**. El código que debemos ingresar en su interior es el siguiente.

```
private void btnColor_Click(object sender, EventArgs e)
{

    // Mostramos el cuadro de dialogo
    if (colorDialog1.ShowDialog() == DialogResult.OK)
    {
        // Colocamos el color seleccionado
        objetoR = colorDialog1.Color.R;
        objetoG = colorDialog1.Color.G;
        objetoB = colorDialog1.Color.B;
    }
}
```

En el handler, invocamos el método **ShowDialog()** del control **colorDialog1**. Este método hace que se muestre la ventana del control, en la cual podemos seleccionar el nuevo color para el seguimiento del robot. Si se efectúa un clic en el botón **OK** dentro del cuadro de diálogo, entonces procedemos a colocar el color seleccionado en nuestro color de seguimiento. Para modificar la tolerancia, haremos uso de dos botones, por lo que creamos el handler para cada uno de ellos.

```
        private void btnDecrementa_Click(object sender, EventArgs e)
        {

            // Decrementamos la tolerancia
            tolerancia -= 0.01;
            if (tolerancia < 0.50)
                tolerancia = 0.50;

            // Actualizamos la etiqueta
            lblTolerancia.Text = tolerancia.ToString();
        }

        private void btnIncrementa_Click(object sender, EventArgs e)
        {
            // Incrementamos la tolerancia
            tolerancia += 0.01;
            if (tolerancia > 1.0)
                tolerancia = 1.0;

            // Actualizamos la etiqueta
            lblTolerancia.Text = tolerancia.ToString();
        }
```

Como podemos ver, el código resulta muy sencillo, simplemente se incrementa o decrementa el valor de la tolerancia. Contamos con un poco de código para evitar que el valor se salga de un rango apropiado de valores y, al final, se actualiza la etiqueta con el valor de la tolerancia. Luego, debemos hacer un pequeño cambio en el handler del menú para salir, de tal forma que incorpore la información necesaria para detener la webcam.

```
        private void salirToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            // Apagamos el timer
            timer.Stop();

            // Cerramos los dispositivos
            this.CerrarDispositivo();
```

```
            // Cerramos la aplicacion
            this.Close();
        }
```

El próximo paso será modificar el handler para el evento **Load** de la forma. Además, debemos adicionar todos los elementos necesarios para inicializar los dos servos que utilizaremos en el proyecto.

```
        private void Form1_Load(object sender, EventArgs e)
        {
            try
            {
                // Codigo para el controlador de servos

                // Instanciamos el objeto
                mServo = new AdvancedServo();

                // Colocamos los handlers relacionados con la conexion
                mServo.Attach += new AttachEventHandler(mServo_Conectar
Handler);
                mServo.Detach += new DetachEventHandler
(mServo_DesconectarHandler);

                // Colocamos el handler relacionado con el error
                mServo.Error += new ErrorEventHandler(mServo_Error
Handler);

                // Abrimos para conexiones
                mServo.open(numeroSerie1);

                if (mServo.Attached == true)
                {
                    // Colocamos los limites de posicion del servo Y
                    mServo.servos[0].PositionMin = 30;
                    mServo.servos[0].PositionMax = 210;

                    // Colocamos los limites de posicion del servo X
                    mServo.servos[1].PositionMin = 30;
                    mServo.servos[1].PositionMax = 210;
```

```
                }
            }
            catch (PhidgetException ex)
            {
                // Enviamos mensaje con el error

                lblPhidget1.Text = ex.Description;

                // Indicamos con amarillo que hay problemas
                txtPhidget1.BackColor = System.Drawing.Color.Yellow;
            }
        }
```

De manera similar, modificamos el handler del evento **FormClosing** de la forma para colocar los servos en su posición final.

```
        private void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {
            // Regresamos el servo al centro
            if (mServo.Attached == true)
            {
                // Colocamos en posicion el servo Y
                mServo.servos[0].VelocityLimit = 1000;
                mServo.servos[0].Engaged = true;
                mServo.servos[0].Position = 120;

                // Colocamos en posicion el servo X
                mServo.servos[1].VelocityLimit = 1000;
                mServo.servos[1].Engaged = true;
                mServo.servos[1].Position = 120;

                Thread.Sleep(500);

                mServo.servos[0].Engaged = false;
                mServo.servos[1].Engaged = false;

                mServo.Attach -= new AttachEventHandler(mServo_Conectar
Handler);
                mServo.Detach -= new DetachEventHandler
```

```
(mServo_DesconectarHandler);
                mServo.Error -= new ErrorEventHandler(mServo_Error
Handler);
            }



            Application.DoEvents();

            // Cerramos el phidget
            mServo.close();

        }
```

El código para el menú **Configurar Phidget 1**, necesita de pequeños cambios. Entonces, usaremos **RedUsers3** y **Robots3**.

```
        private void configurarPhidget1ToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            // Este codigo es para el phidget 1

            Configuracion dlgConfig = new Configuracion();
            dlgConfig.NumeroSerie = numeroSerie1;

            if (dlgConfig.ShowDialog() == DialogResult.OK)
            {
                // Escribimos el nuevo valor
                numeroSerie1 = dlgConfig.NumeroSerie;

                // Abrimos la llave
                RegistryKey sfwKey =
Registry.LocalMachine.OpenSubKey("Software", true);
                RegistryKey redKey = sfwKey.CreateSubKey("RedUsers3");

                RegistryKey robotKey = redKey.CreateSubKey("Robots3");


                // Colocamos el nuevo valor
```

```
            robotKey.SetValue("NumeroSerie1", (object)numeroSerie1);

        }
    }
```

El handler para el evento de conexión del Phidget necesita la adición del segundo servo. Éste ubicará ambos servos en su posición inicial.

```
    public void mServo_ConectarHandler(object sender,
AttachEventArgs e)
    {
        // Indicamos que hay conexion
        conectado1 = true;

        // Enviamos mensaje de la conexion
        lblPhidget1.Text = e.Device.Name;

        // Indicamos con color verde que hay conexion
        txtPhidget1.BackColor = System.Drawing.Color.Lime;

        // Configuramos sus caracteristicas
        if (mServo.Attached)
        {
            // Configuramos el servo Y

            // Colocamos la aceleracion
            mServo.servos[0].Acceleration = 100.0;

            // Colocamos la posicion inicial
            mServo.servos[0].Engaged = true;
            mServo.servos[0].Position = 100.0;
            mServo.servos[0].VelocityLimit = 1000.0;

            // Configuramos el servo X

            // Colocamos la aceleracion
            mServo.servos[1].Acceleration = 100.0;

            // Colocamos la posicion inicial
```

```
            mServo.servos[1].Engaged = true;
             mServo.servos[1].Position = 100.0;
             mServo.servos[1].VelocityLimit = 1000.0;


        }

    }
```

Tenemos dos controles de tipo **TrackBar**, uno para cada eje del robot. Debemos generar los handlers para el evento **Scroll** y el evento **LocationChanged** de ambos.

```
    private void trackX_Scroll(object sender, EventArgs e)
    {
        // Obtenemos el valor y lo mostramos
        lblX.Text = trackX.Value.ToString();

        // Verificamos si esta conectado y en manual
        if (conectado1 && chkManual.Checked)
        {
            // Colocamos la posicion en el servo
            mServo.servos[1].Position = trackX.Value;
        }

    }

    private void trackX_LocationChanged(object sender, EventArgs e)
    {
        // Verificamos si esta conectado
        if (conectado1 && chkManual.Checked)
        {
            mServo.servos[1].Engaged = true;
            Thread.Sleep(500);
            mServo.servos[1].Engaged = false;
        }
    }

    private void trackY_Scroll(object sender, EventArgs e)
    {
```

```
        // Obtenemos el valor y lo mostramos
        lblY.Text = trackY.Value.ToString();

        // Verificamos si esta conectado y en manual
        if (conectado1 && chkManual.Checked)
        {
            // Colocamos la posicion en el servo
            mServo.servos[0].Position = trackY.Value;
        }
    }

    private void trackY_LocationChanged(object sender, EventArgs e)
    {
        // Verificamos si esta conectado
        if (conectado1 && chkManual.Checked)
        {
            mServo.servos[0].Engaged = true;
            Thread.Sleep(500);
            mServo.servos[0].Engaged = false;
        }
    }
```

En el handler del evento **Scroll**, obtenemos el valor del **Trackbar**, que se agrega en la etiqueta correspondiente. Si el **CheckBox chkManual** se encuentra seleccionado y estamos conectados al Phidget, entonces establecemos el valor del **TrackBar** como la posición del servo.

Para el handler de **LocationChanged**, también, verificamos que el **CheckBox chkManual** se encuentre seleccionado y tenga la conexión al Phidget. Entonces, sólo basta con insertar la propiedad de **Engaged** en **true** para que inicie el movimiento; hacemos una pequeña espera para que el servo llegue a su nueva posición y regresamos la propiedad de **Engaged** al valor **false**. Por último, necesitamos establecer el método **Seguimiento()**. Este método modificará la posición de los servos usando la información calculada por el método **ReductorColor()**.

```
    private void Seguimiento()
    {
        // Codigo de seguimiento
```

```
            // Variables necesarias
            double deltaX;
            double deltaY;
            double posX;
            double posY;

            // Verificamos que es posible llevar a cabo el seguimiento
            if (chkManual.Checked == false && conectado1 == true)
            {
                // Eje X
                // Obtenemos la posicion actual
                posX = mServo.servos[0].Position;

                // Obtenemos que tanto esta desfasado el foco con
relacion al centro
                deltaX = ((((double)xfoco - (double)xcentral) /
(double)miBitmap.Width) * 20.0);

                // Calculamos la nueva posicion
                posX += deltaX;

                // Verificamos la nueva posicion dentro de los limites
del servo
                if (posX < 30.0)
                    posX = 30.0;
                else if (posX > 210.0)
                    posX = 210.0;

                // Colocamos la posicion en el servo
                mServo.servos[0].Position = posX;

                // Colocamos la posicion en el trackbar y la etiqueta
                trackY.Value = (int)posX;
                trackY.Invalidate();
                lblY.Text = trackY.Value.ToString();

                // Eje Y
                // Obtenemos la posicion actual
                posY = mServo.servos[1].Position;
```

```
                // Obtenemos que tanto esta desfasado el foco con
relacion al centro
                deltaY = ((((double)yfoco - (double)ycentral) /
(double)miBitmap.Height) * 15.0);

                // Calculamos la nueva posicion
                posY += deltaY;

                // Verificamos la nueva posicion dentro de los limites
del servo
                if (posY < 30.0)
                    posY = 30.0;
                else if (posY > 210.0)
                    posY = 210.0;

                // Colocamos la posicion en el servo
                mServo.servos[1].Position = posY;

                // Colocamos la posicion en el trackbar y la etiqueta
                trackX.Value = (int)posY;
                trackX.Invalidate();
                lblX.Text = trackX.Value.ToString();

                // LLevamos a cabo el movimiento de los servos
                mServo.servos[0].Engaged = true;
                mServo.servos[1].Engaged = true;

                // Si hay problemas con la aplicacion aumentar el tiempo
de espera
                Thread.Sleep(80);

                mServo.servos[0].Engaged = false;
                mServo.servos[1].Engaged = false;

            }
        }
```

En primer lugar, creamos una serie de variables de trabajo que nos apoyarán en el cálculo de los valores necesarios. Para poder llevar a cabo el seguimiento, es preciso que el Phidget se encuentre conectado y que no estemos usando el movimiento

manual. Luego, continuamos calculando los valores para el eje X. Para empezar, obtenemos la posición actual del servo y , luego, calculamos el valor de **deltaX**. Este valor es la desviación que existe entre el valor x del foco y la coordenada x del centro de la imagen; debemos encargarnos de normalizar esta distancia con relación al ancho de la imagen. Por último, la multiplicamos por un factor, en este caso 20. Podemos modificar el valor para hacer el movimiento más brusco o más suave.

Una vez obtenido el valor **deltaX**, lo sumamos a **posX** para encontrar la nueva posición del servo. Hay que notar que **deltaX** puede tener valores tanto positivos como negativos, por lo que el servo se puede mover en dos direcciones, no necesariamente siempre incrementar. Para evitar que el servo intente moverse más allá de su rango, agregamos una pequeña cantidad de código que restringe el valor. Una vez verificado el valor, procedemos a ingresarlo en el servo. Lo mismo sucederá con el eje Y, pero usando sus variables respectivas. Con los valores de los servos colocados, establecemos la propiedad de **Engaged** en **true** para que se efectúe el movimiento. Hacemos una espera y, luego, la ubicamos otra vez en **false**.

Ahora, podemos tomar un objeto de color rojo, compilar y ejecutar la aplicación, y observar cómo el robot lleva a cabo el seguimiento del objeto.

... RESUMEN

El robot de este capitulo está constituido de diferentes subsistemas. Tenemos un sistema mecánico que se encarga de mover la cámara, un sistema de captura de video y uno de procesamiento de la imagen, que le indica al sistema mecánico cuánto debe moverse. Se utilizan dos servos: uno para girar la cámara en el eje Y; el otro nos permite girar la cámara en el eje X. El procesamiento de imagen busca un área dentro de la imagen donde se pueda encontrar el color buscado. El centro de esta área es tomado como el lugar que debe observar el robot. Para calcular el movimiento del robot, se verifica la distancia de este punto al centro de la imagen. El control de los servos se lleva a cabo mediante un Phidget.

TEST DE AUTOEVALUACIÓN

1. Describa el algoritmo para el procesamiento de la imagen.

2. ¿De que forma se modifica la velocidad de los servos?

3. ¿De que manera se puede hacer que cambie la velocidad de los servos dependiendo de la distancia por recorrer?

ACTIVIDADES

1. Diseñe un brazo robótico para la cámara que sea más compacto que el brazo que vimos en el presente capítulo.

2. Modifique el programa para que el movimiento de la cámara sea más suave.

3. El algoritmo tiene muchos lugares que brindan la oportunidad de optimizar el código, busque estos lugares para optimizarlo.

4. Transforme el programa para que el robot siga el movimiento en lugar del color.

5. Modifique el programa para que el robot inicie en una posición predefinida por medio de la interfaz de usuario.

6. Cambie el brazo y el programa para que el robot tenga tres grados de libertad.

7. Modifique el programa para que la cámara pueda llevar a cabo recorridos preprogramados.

8. Modifique el programa para poder grabar el movimiento del robot y, luego, reproducirlo.

9. Modifique la interfaz para que dé más información sobre el movimiento.

Dedo robótico y guante VR

En este capítulo, haremos un nuevo proyecto que consta de dos componentes. Por una parte, un dedo robótico que podemos controlar con el software y, por otra, un guante inspirado en los guantes de realidad virtual, con el cual controlaremos al dedo. Éste va a ser el primer proyecto en el que utilizaremos dos Phidgets de forma simultánea.

DESCRIPCIÓN DEL PROYECTO

El dedo robótico será sencillo y, únicamente, podrá llevar a cabo flexiones para cerrarse, aun así, puede ser la base para diseños todavía más complejos. El dedo funcionará de manera similar a un dedo humano, en el sentido que un tendón, al jalarse, provee la contracción. Nosotros usaremos un tensor a manera de tendón, y un servo proveerá el movimiento dado por el músculo. El sistema puede modificarse, con facilidad, para adicionar más dedos y servos. El dedo necesita una base firme donde el servo estará colocado y una parte flexible que será la que se mueva. El dedo puede ser controlado en forma directa por el software, pero, para hacer el proyecto más interesante, construiremos un guante similar a los guantes de realidad virtual, aunque, en nuestro caso, con un solo sensor para medir la flexión del dedo índice del usuario.

Como sensor, usaremos un tipo especial de resistencia. Esta resistencia no sólo es flexible, sino también elástica. Este tipo de material varía su resistencia cuando se estira; de esta forma, podremos medir qué tan contraído está el dedo del usuario. De igual manera, podemos adicionar más sensores al guante y tener más control sobre una futura mano robótica u otros proyectos.

El software resultará muy sencillo. Podremos mover el servo de manera manual y por medio del guante. El software nos indicará la cantidad de flexión del dedo del usuario y flexionará el dedo robótico en un porcentaje correspondiente. Debido a las variaciones entre las resistencias y los modelos de servo, es necesario establecer algunos mecanismos de configuración y de ajuste. Antes de poder usar el sistema de manera completa, será necesario configurar el rango de valores de la resistencia y el máximo de movimiento del servo.

Componentes necesarios

En la siguiente tabla podemos observar todos los componentes que necesitaremos para la realización del actual proyecto.

III MEJORAR LOS CORTES

Cuando llevamos a cabo el corte de los componentes, es preferible que estén cuadrados en forma adecuada. Para esto, podemos ayudarnos con escuadras y reglas. De esta manera, será más sencillo ensamblar cada sección del proyecto. Algunos materiales se cortan mejor con un estilete en varias pasadas de corte consecutivas.

NOMBRE	CANTIDAD	DESCRIPCIÓN
Controlador servos	1	PhidgetAdvancedServo
Phidget 8/8/8	1	Phidget 8/8/8
Servo	1	Servo
Tubo	1	Tubo plástico flexible 13.5 cm x 1.5 cm de diámetro
Cuerda	1	45 cm de cuerda
A	1	Cartón 5 x 11 cm, 1 mm de espesor
B	2	Cartón 4 x 4 cm, 3 mm de espesor
C	1	Cartón 2.5 x 3 cm, 3 mm de espesor
D	2	Cartón1.5 x 6 cm, 4 mm de espesor
E	1	Cartón 2.6 x 6 cm, 2 mm de espesor
F	1	Cartón 1.5 x 2 cm, 4 mm de espesor

Tabla 1. *Éstos son los componentes que usaremos en la construcción del dedo robótico y del guante VR.*

CONSTRUCCIÓN DEL ROBOT

El primer elemento que necesitamos es una base que sostendrá al servo y al dedo. Esta base permitirá tener al servo colocado de manera firme para que, cuando el tensor sea jalado, el movimiento del dedo se lleve a cabo. Lo primero que haremos para la base será construir la sección que se encarga de sostener al dedo robótico. Para esto, necesitaremos los componentes **A**, **D** y **F**.

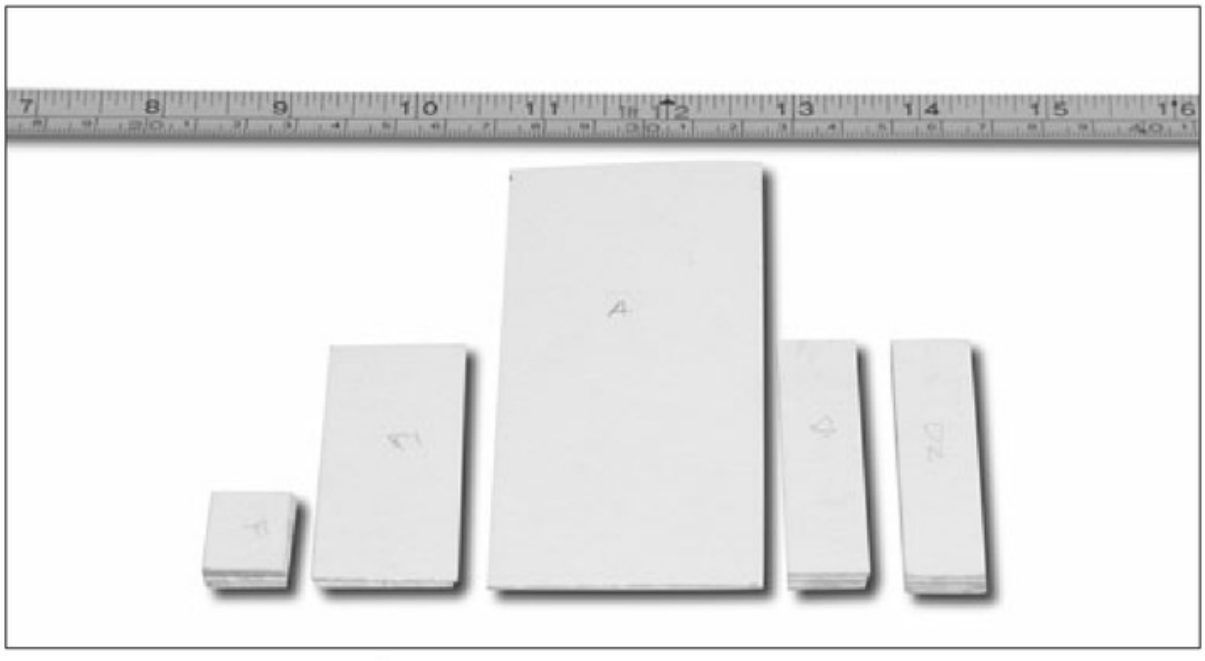


Figura 1. *Éstos son los componentes que necesitamos para construir el soporte para el dedo robótico.*

Ahora, necesitamos llevar a cabo unos trazos sobre el componente **A**. Lo primero que hacemos es marcar a 14 mm del borde superior y a 24 mm del borde derecho. La siguiente figura nos muestra las marcas.

Figura 2. *Debemos colocar marcas a las distancias indicadas sobre el componente A.*

Usando las marcas que hemos establecido como guía, procedemos a trazar un par de líneas. El punto donde se produce la intersección es importante. La línea horizontal nos servirá, también, como guía para centrar de manera adecuada el dedo robótico.

Figura 3. *Las líneas trazadas nos servirán de guías para el dedo y para el servo que agregaremos más adelante.*

Por el punto donde se cruzan las líneas, pasará el eje del servo. Para lograr esto, es necesario realizar un agujero del tamaño adecuado. Para hacerlo más sencillo y seguro, lo llevaremos a cabo por etapas. Empezamos por introducir un clavo en el punto de intersección. El clavo debe perforar el componente **A** y salir del otro lado.

Figura 4. *Con un clavo, hacemos una perforación inicial en el punto de intersección de las líneas.*

Para hacer el agujero más grande, podemos apoyarnos con un bolígrafo o un portaminas. Aprovechamos el agujero del clavo para guiarnos.

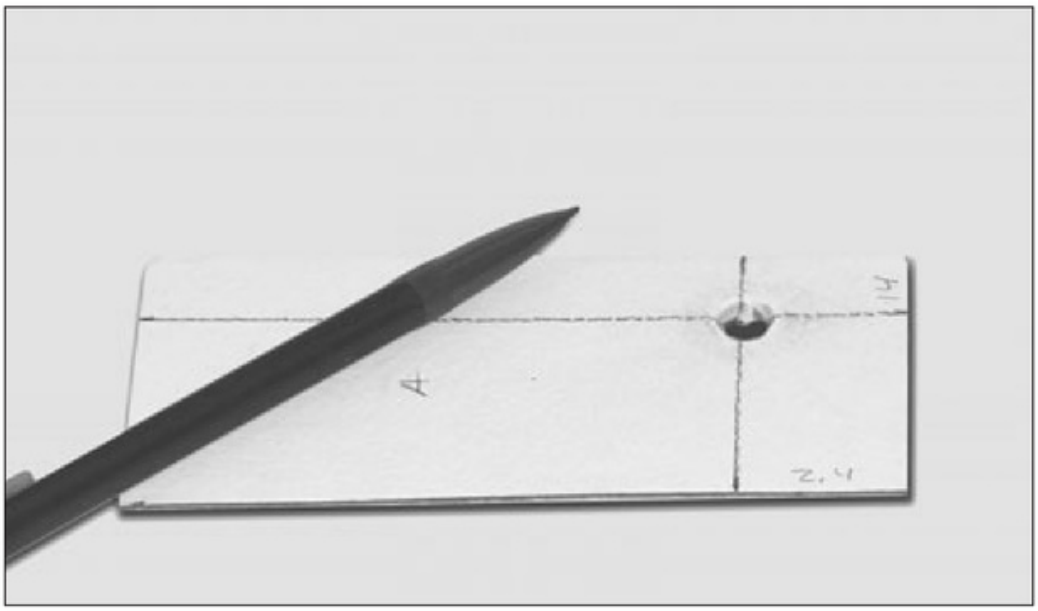


Figura 5. *Un portaminas nos ha ayudado a agrandar el diámetro del agujero.*

El agujero debe tener un diámetro de aproximadamente 1 cm. En caso de que el portaminas no tenga el diámetro correcto, podemos apoyarnos en un plumón o marcador grueso para finalizar el agujero.

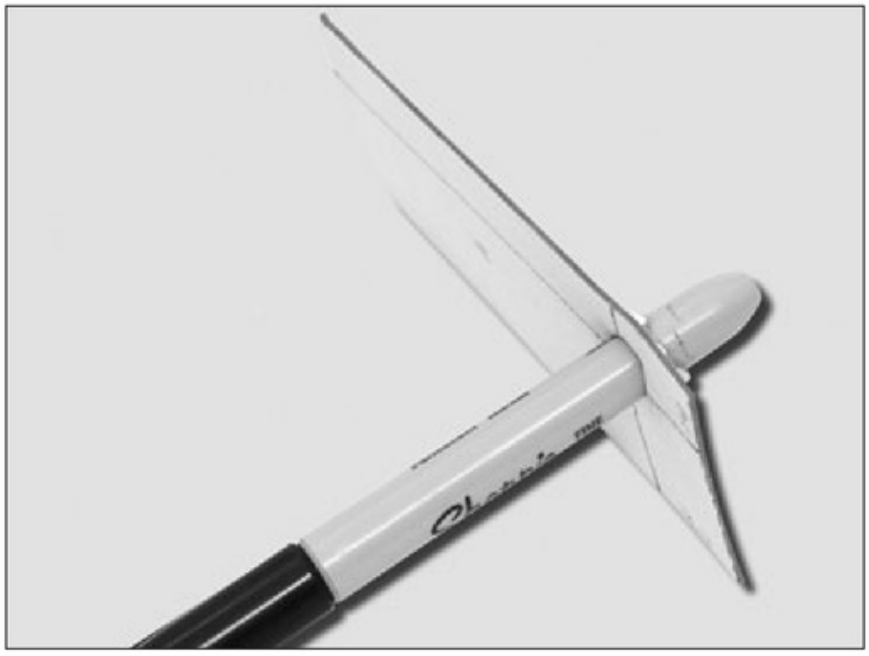


Figura 6. *Un marcador que tenga el grosor necesario puede permitirnos hacer el agujero con el componente* ***A***

Es muy probable que, al hacer las perforaciones, el cartón presente unos pedazos que sobresalgan en el lado opuesto. Esto es natural y lo solucionaremos muy pronto.

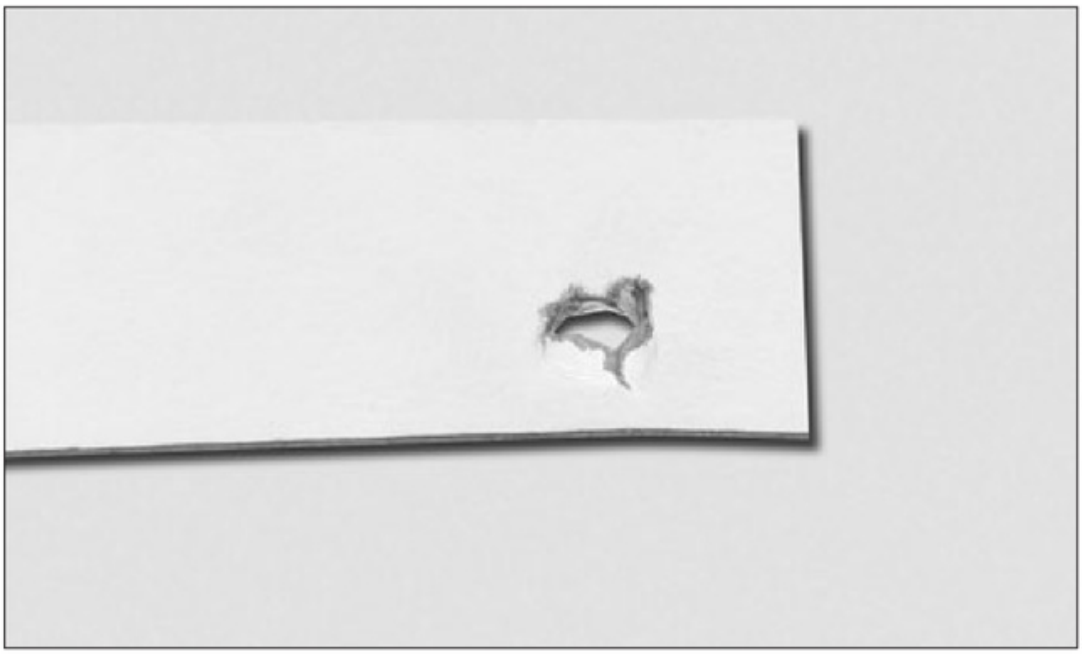


Figura 7. *Nuestro cartón puede presentar trozos irregulares del lado opuesto, los cuales necesitan ser limpiados.*

Con mucho cuidado y usando una navaja, será posible eliminar estos restos indeseables y dejar el componente **A** con una superficie lisa en ambos lados.

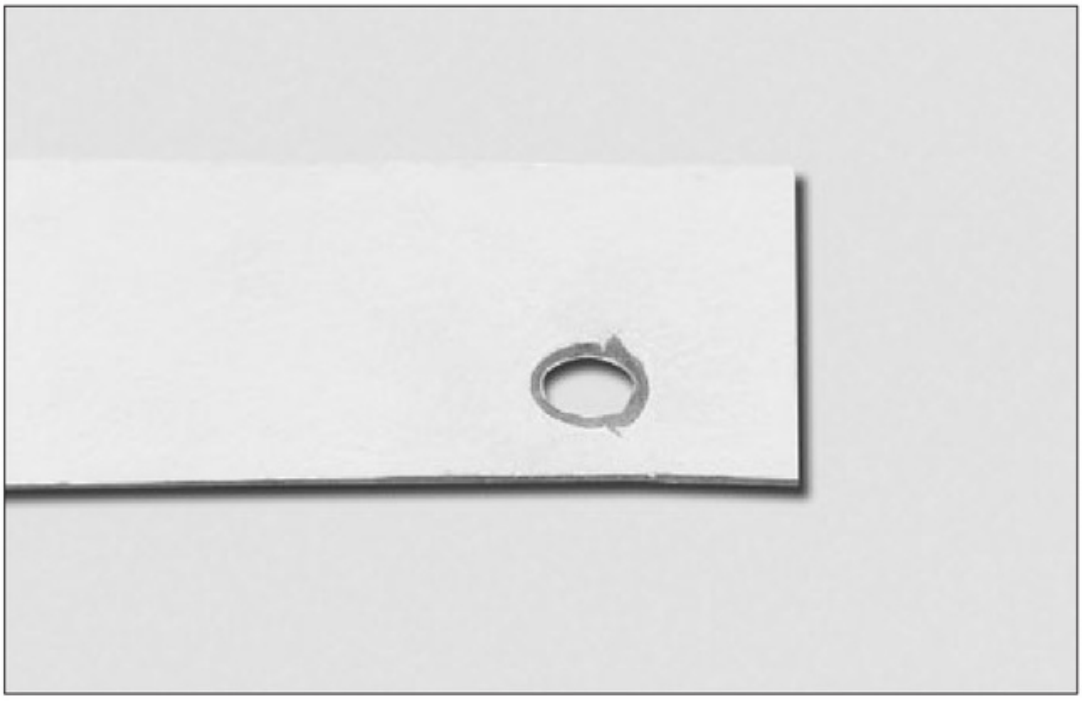


Figura 8. *Podemos ver la diferencia en la superficie una vez que han sido eliminados los sobrantes. Ahora, el componente* ***A*** *se encuentra limpio.*

Para empezar a colocar el soporte del dedo robótico, tomamos uno de los componentes **D** y lo colocamos sobre la parte superior de **A**. Debe estar pegado sobre el borde y llegar hasta el extremo izquierdo.

***Figura 9.** En esta figura, podemos observar cómo se debe montar el primer componente para el sostén del dedo robótico.*

Tomamos el otro componente **D** y a 1,5 cm de distancia del anterior, lo pegamos sobre el componente **A**. Ambos componentes **D** deben quedar paralelos. La línea que trazamos antes nos puede ayudar en este paso.

***Figura 10.** Ambos componentes **D** han sido colocados sobre **A**. Es importante que se encuentren de forma paralela y que lleguen hasta el extremo izquierdo.*

III EL USO DEL PEGAMENTO

Es importante leer las instrucciones del pegamento y asegurarnos de que sirve para el material que estamos usando para construir el robot. Algunos adhesivos requieren de una capa previa antes de llevar a cabo el contacto final. También, es útil no mover las piezas hasta que el pegamento se encuentre totalmente seco.

El componente **F** tiene una función especial: será usado como tope para que el dedo robótico no resbale hacia el servo. Colocamos el componente **F** entre los componentes **D**. Debe estar pegado de manera firme sobre el componente **A** y colocado hacia el extremo cercano del lugar donde estará el servo. Esto lo podemos observar en la siguiente figura.

Figura 11. *En esta figura, puede verse el lugar donde se insertará el componente F.*

Para poder finalizar el lugar donde se montará el dedo robótico, debemos colocar el componente **E** como si fuera un techo sobre los componentes **D**. Este componente forma parte de la estructura que resistirá el movimiento del dedo, por lo que debe quedar pegado de manera firme.

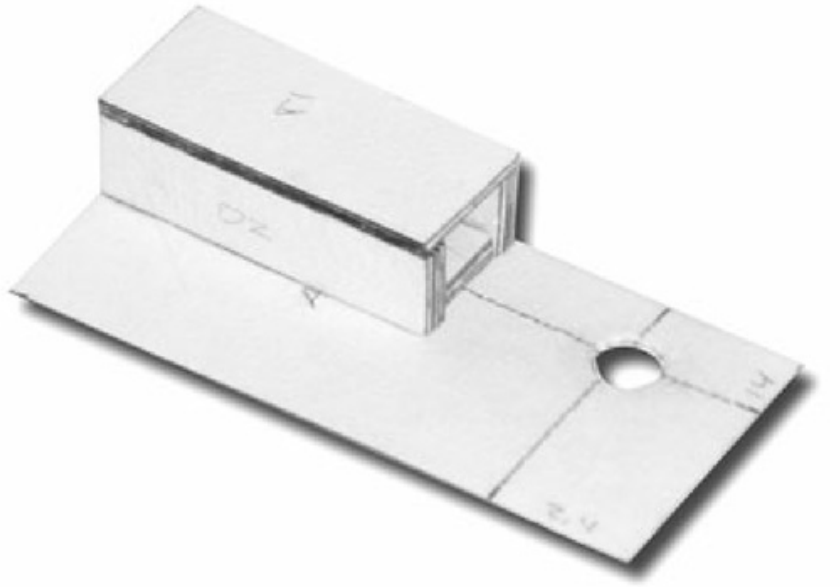


Figura 12. *El componente **E** finaliza el lugar de montaje para el dedo robótico.*

Para poder proseguir, debemos hacer un montaje temporal del servo. Primero, es necesario separar el elemento de actuación del servo. Luego, hacemos pasar el eje del servo por el agujero y colocamos, otra vez, el elemento de actuación. No debemos usar el tornillo del elemento de actuación, ya que este montaje es temporal.

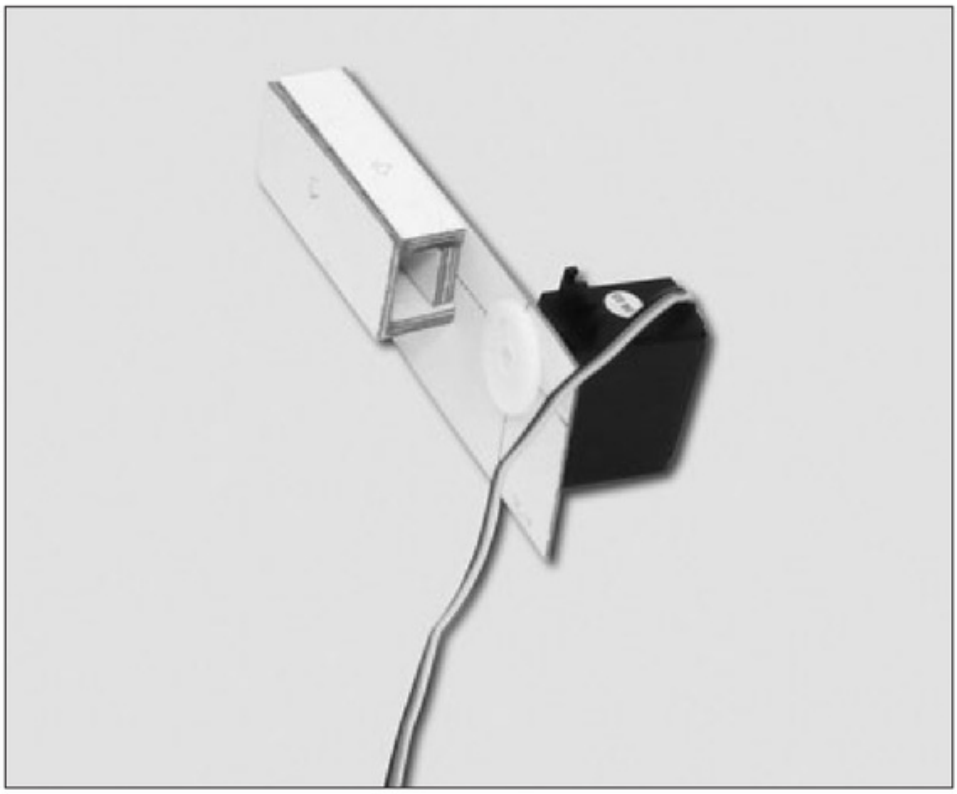


Figura 13. *Esta figura nos muestra la forma en que debemos montar el servo pasando su eje por el agujero.*

El servo también necesita tener una estructura de soporte. Esto es preciso con el fin de evitar cualquier movimiento innecesario. Como ya tenemos el servo montado, su soporte será construido alrededor de él. Los componentes que usaremos para construir este soporte son los **B** y **C**.

Figura 14. *Estos componentes serán usados para construir una caja que soporte al servo.*

Debido a que la base del proyecto no es simétrica, resulta un poco difícil poder colocar los componentes en su lugar. Para ayudarnos, es posible usar o hacer un objeto que nos brinde soporte y que mantenga derecha la base, mientras el soporte es construido.

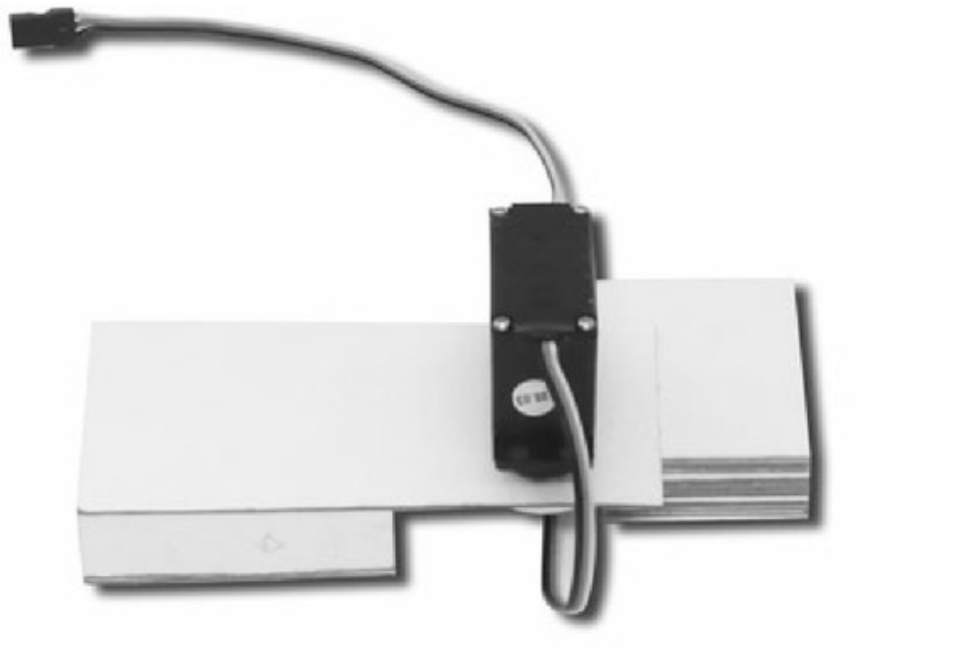


Figura 15. *Estamos usando un soporte para ayudarnos en la construcción de la base. Puede ser cualquier objeto que permita estabilizarla.*

El servo debe estar alineado. Su lado más largo debe ser paralelo al lado menor del componente **A**. Una vez que se encuentre de forma correcta en su lugar, entonces, procedemos a pegar uno de los componentes **B**. Es importante que el servo no quede pegado también. La siguiente figura nos muestra cómo colocarlo.

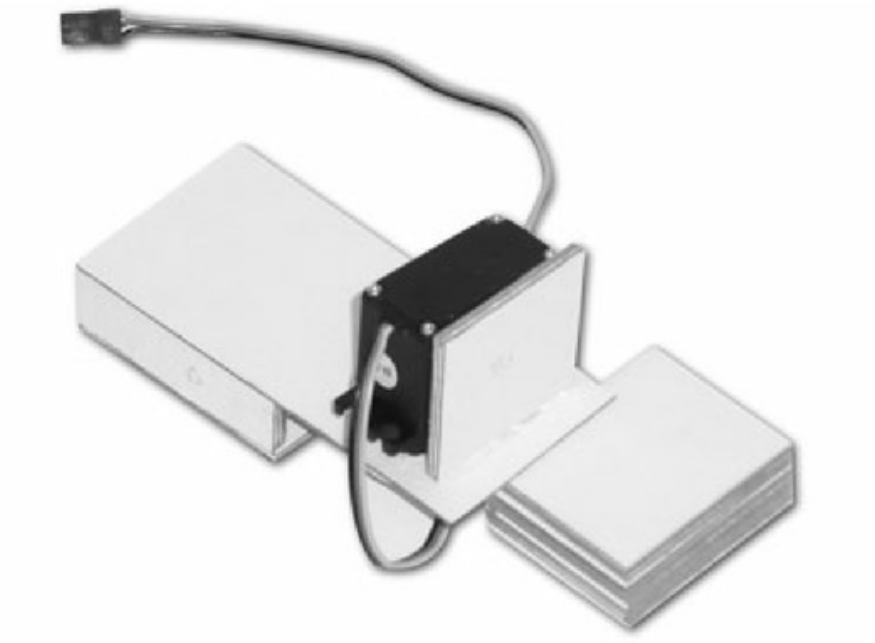


Figura 16. *El componente* ***B*** *será uno de los lados que conforman la caja que sostendrá al servo.*

El servo necesita estar sujeto del otro lado también, por lo que colocamos el otro componente **B**. Este componente, además, debe estar pegado al componente **A**.

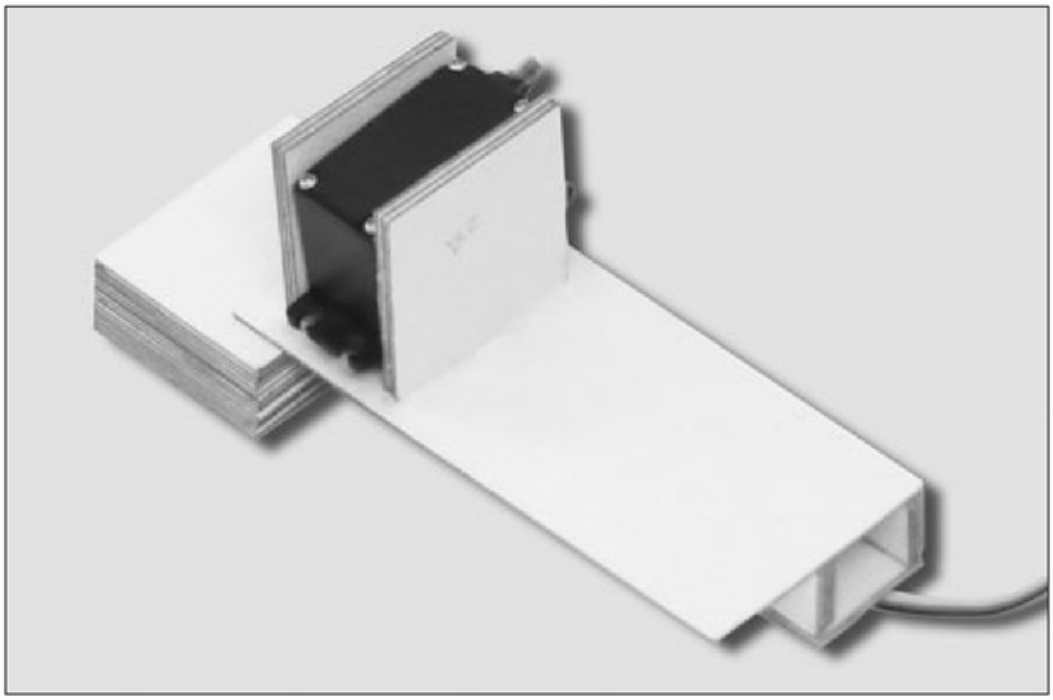


Figura 17. *El servo ahora tiene paredes en ambos lados que ayudan a sostenerlo y evitar que gire de forma incorrecta.*

Las dos paredes son apropiadas, pero hace falta que demos mayor soporte estructural al sostén del servo. Para eso, utilizaremos el componente **C**. Este componente debe ir pegado en la parte contraria a donde sale el cable del servo. Se pegará a los dos componentes **B** que ya se encuentran colocados.

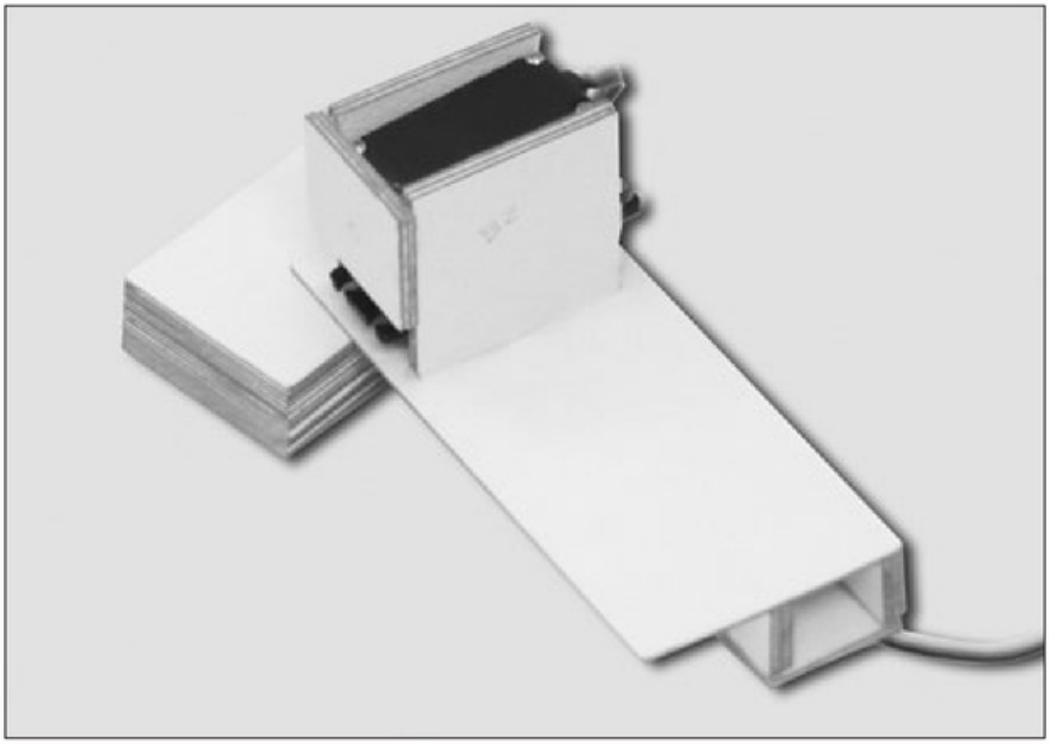


Figura 18. *El componente* ***C*** *brinda firmeza estructural al contenedor del servo. No debe estorbar la salida de los cables.*

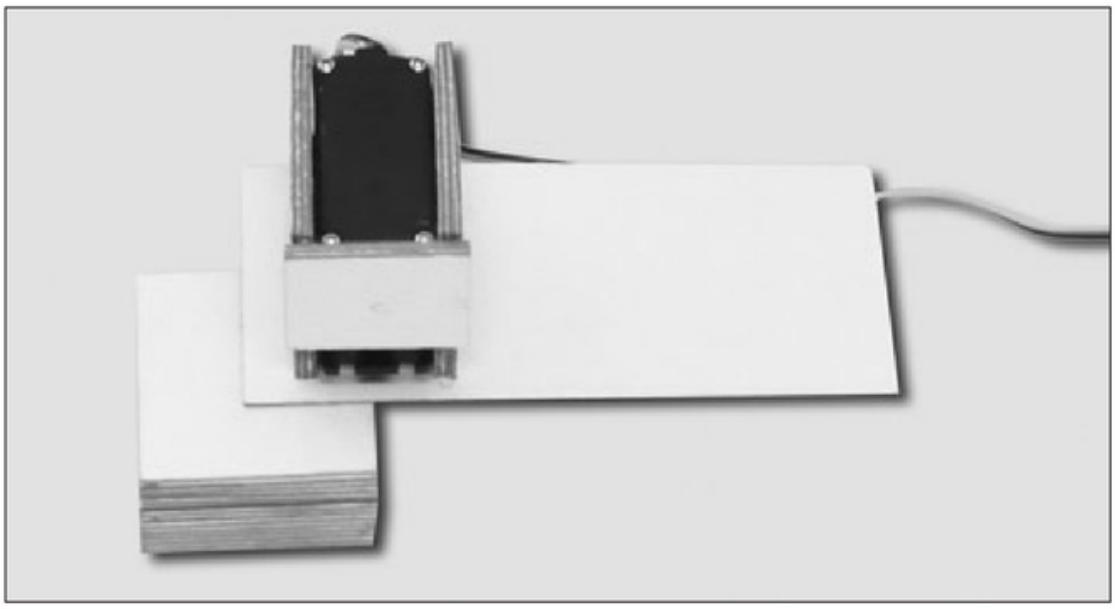


Figura 19. *Cuando diseñamos un soporte para servo, tenemos que cuidar que el servo sea fácil de colocar y de quitar.*

Con esto, tenemos la base para el dedo robótico lista. Podemos quitar el servo de manera temporal y proceder a pintar la base para que tenga un mejor acabado.

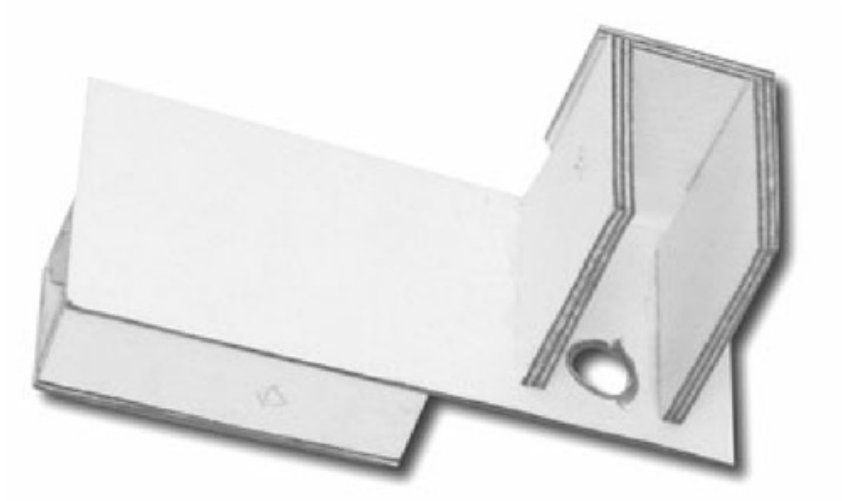


Figura 20. *Ésta es nuestra base finalizada. En ella, instalaremos el dedo robótico y el servo.*

III INSTALACIÓN DEL SERVO

El servo puede ser instalado por medio de contenedores similares a los que usamos en nuestros proyectos o podemos modificar los contenedores para que el servo se atornille a ellos. Esto lo hacemos dependiendo de nuestras necesidades de diseño. Si el proyecto deberá funcionar por mucho tiempo, lo mejor es atornillar.

Construcción del dedo robótico

La siguiente etapa en nuestro proyecto es la creación del dedo robótico. Para éste, necesitaremos el tubo flexible, un resorte y la cuerda. Estos materiales se pueden conseguir con facilidad. En caso de ser necesario, el resorte puede ser fabricado por nosotros mismos usando alambre metálico. En última instancia, el resorte no es absolutamente necesario, si no lo conseguimos, aun así puede llevarse a cabo el proyecto. La función del resorte es la de ayudar a regresar el tubo flexible a su posición original de forma más rápida. Existen diferentes tipos de resortes que podemos utilizar; se requiere un poco de experimentación para encontrar el resorte que sea el adecuado a nuestro proyecto.

El tubo que usamos puede ser un tipo de tubo para proyectos flexibles. Este tubo se encuentra en varios colores, y, por supuesto, es posible escoger el color que más nos agrade. La siguiente figura muestra un ejemplo del tubo por utilizar.

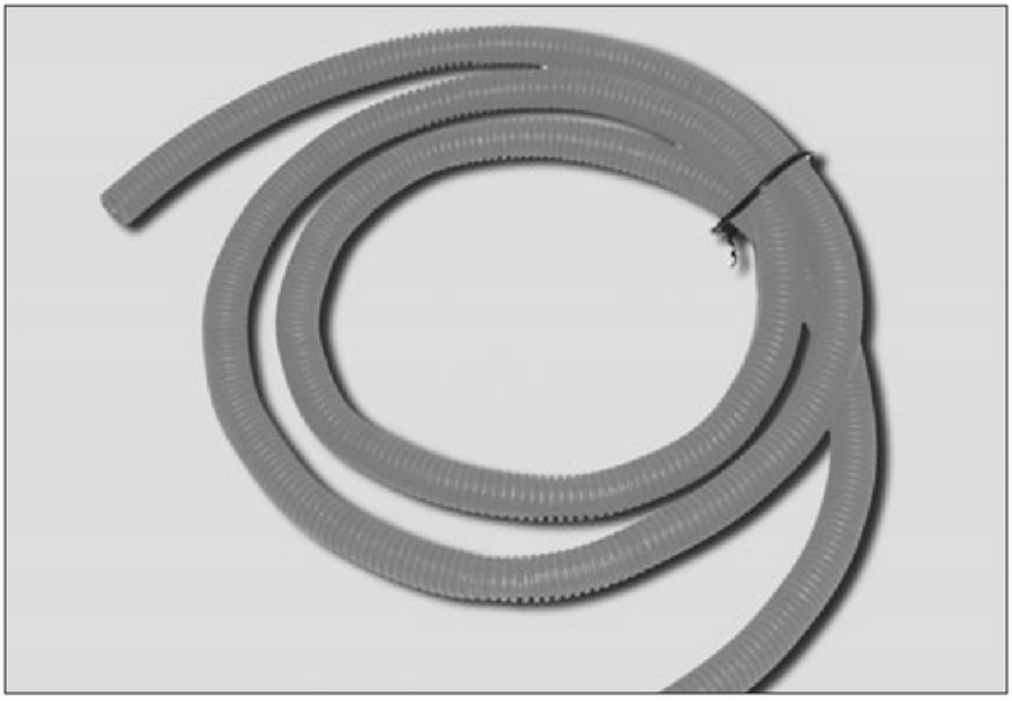


Figura 21. *Este tipo de tubo se consigue en las ferreterías y tiendas de artículos eléctricos. Hay que verificar que sea flexible, como se muestra en la figura.*

III SOLVENTES EN LA PINTURA

Algunas pinturas tienen solventes que pueden lastimar el plástico o afectar el pegamento. Antes de usar la pintura, debemos hacer una pequeña prueba para verificar que no lastime el material con el que construimos nuestro robot. Esto es necesario realizarlo cada vez que compremos pintura, pues el fabricante puede modificar los solventes que usa en la fabricación.

Una vez que conseguimos el tubo, tenemos que cortarlo a una longitud de 13,5 cm. De ser posible, debemos tomar el tramo más recto.

Figura 22. *El tubo ha sido cortado al largo deseado, siempre procurando que sea lo más recto posible.*

Para que funcione como un dedo, necesitamos brindarle articulaciones. Al dedo, le colocaremos tres articulaciones. Con un marcador o plumón, establecemos tres marcas, cada una de ellas a 2,5 cm a partir del borde izquierdo del tubo.

Figura 23. *Los puntos en donde se crearán las articulaciones han sido marcados. Hay una separación de 2,5 cm entre ellos.*

A continuación, tenemos que crear la primera articulación. Todas las articulaciones se hacen de la misma forma, por lo que veremos en detalle cómo hacer la primera de ellas. Empezamos por realizar un corte al nivel de la primera marca. El corte debe ser en un ángulo de 45 grados.

III RESORTES MUY RÍGIDOS

Si el resorte es muy rígido, puede impedir el movimiento del servo o forzarlo demasiado. El resorte debe ser lo suficientemente suave para ser movido por el servo, pero ayudar a extender el dedo otra vez. Esto requiere de prueba y error, pero no debe ocasionarnos problemas.

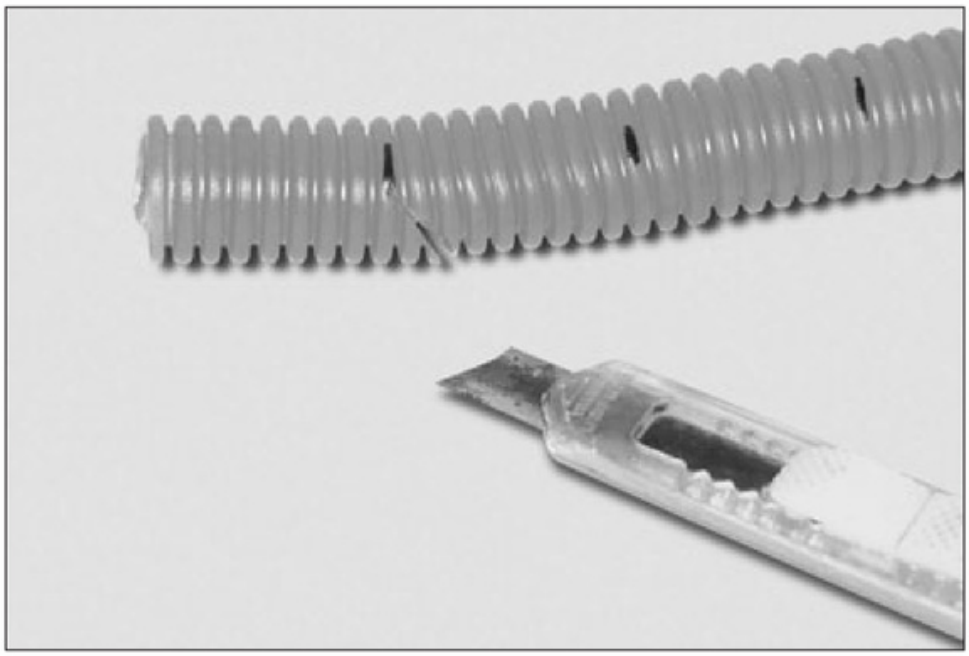


Figura 24. *El corte debe realizarse a 45 grados y atravesar el tubo. Esta figura nos muestra la forma en que debe quedar.*

Para terminar la articulación, debemos hacer otro corte. Este nuevo corte, también, es a 45 grados. Ambos cortes deben formar una especie de triángulo. Este espacio vacío que queda es lo que permitirá que se lleve a cabo la articulación.

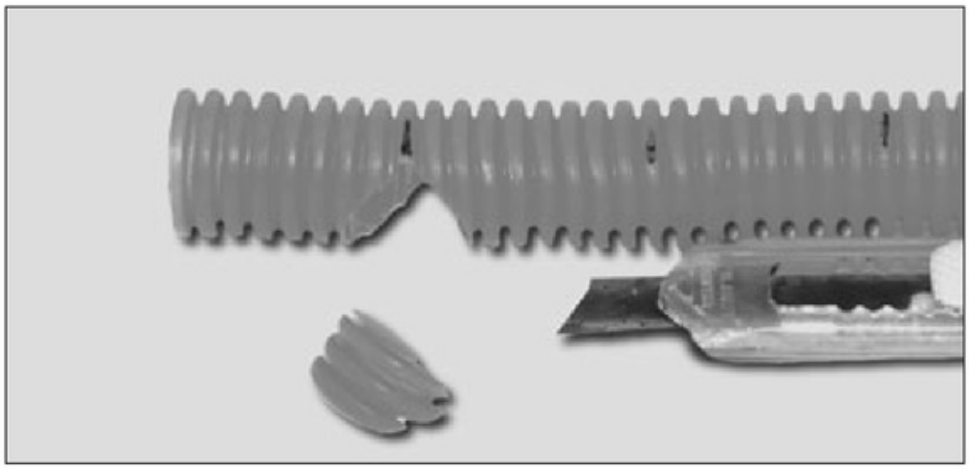


Figura 25. *Los dos cortes forman un triángulo que le permitirá al tubo doblarse en ese punto.*

III CORTAR EL TUBO ELÉCTRICO

El tubo eléctrico que vamos a utilizar es de material plástico; esto facilita su corte. Podemos cortar el material con una navaja, pero, siempre que se use este elemento debemos ser cuidadosos para no provocar un accidente al realizar el corte. Si el tubo es corrugado, lo mejor es realizar el corte en la parte baja del corrugado.

Haciendo uso del mismo procedimiento, en la segunda marca que hicimos, llevamos a cabo también dos cortes. Con esto, generamos la segunda articulación.

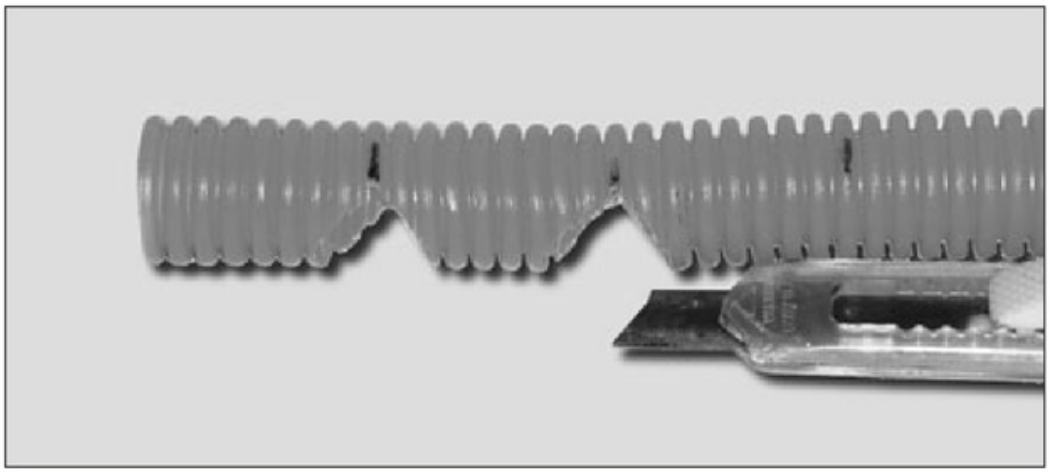


Figura 26. *La segunda articulación se crea de forma similar a la primera.*

Ahora, sólo nos falta una articulación más. Esta articulación la colocamos en el lugar en donde se encuentra la tercera marca sobre el tubo.

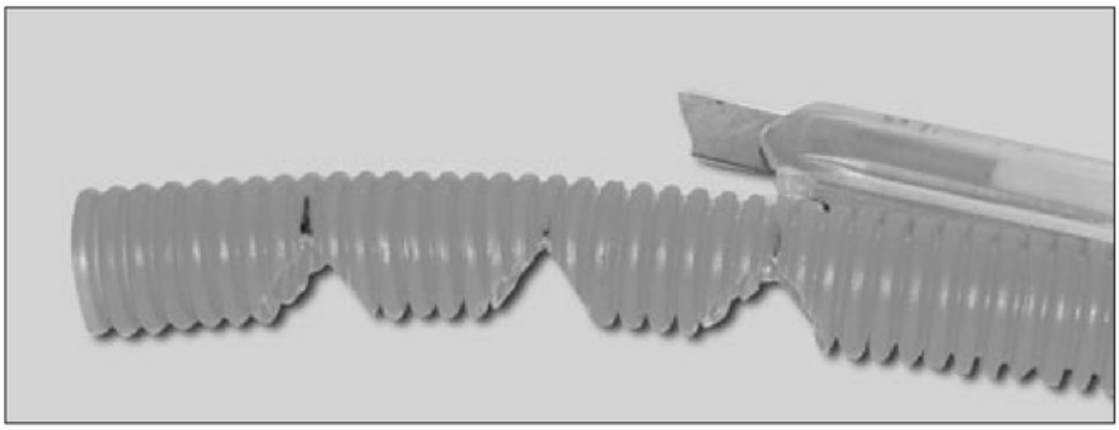


Figura 27. *Con los últimos cortes, finalizamos las articulaciones necesarias para el dedo robótico.*

Aún necesitamos hacer un corte extra en el tubo. En el extremo, debemos hacer una pequeña perforación. Esta perforación nos servirá para atar el hilo que trabajará

III LAS ARTICULACIONES DEL DEDO

Con el fin de que la articulación del dedo funcione de manera correcta y el tubo no se parta por la flexión, el corte triangular preferentemente no debe pasar de la mitad del tubo. Lo más sencillo es marcar primero los lugares para hacer los cortes y, así, evitar problemas.

como tensor en el dedo. El corte debe estar en el mismo lado que los cortes de la articulación. La siguiente fotografía lo muestra.

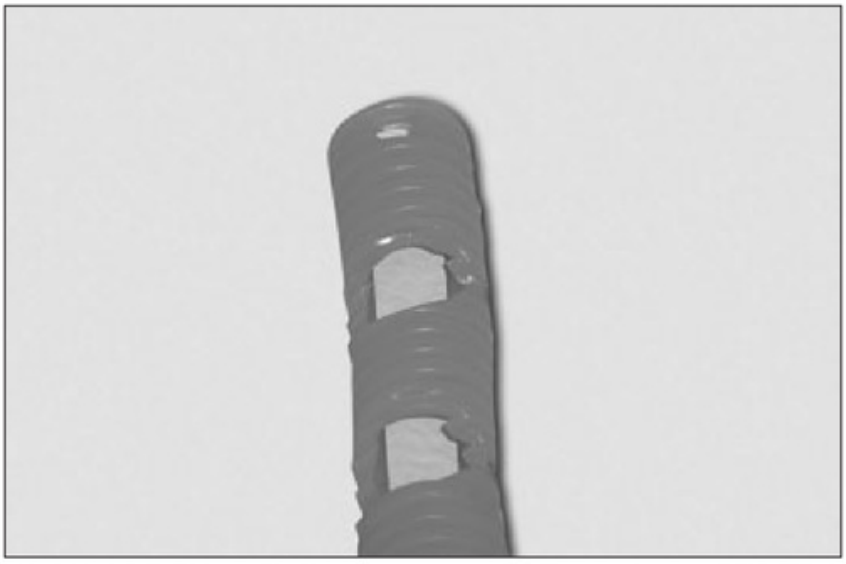


Figura 28. *El orificio que hacemos nos permitirá atar el hilo. En esta figura, se ha insertado un papel blanco al tubo para resaltar el agujero.*

Ahora, vamos a concentrarnos un poco en el resorte. En caso de que no podamos obtener un resorte, es posible construirlo nosotros mismos. Para realizarlo, necesitamos un alambre de metal y un cilindro sobre el cual enrollarlo para darle la forma.

Como cilindro, podemos utilizar el cuerpo de un destornillador. El mango del destornillador nos será útil para poder enrollar el cable de manera adecuada. Un ejemplo de esto lo podemos observar en la siguiente figura.

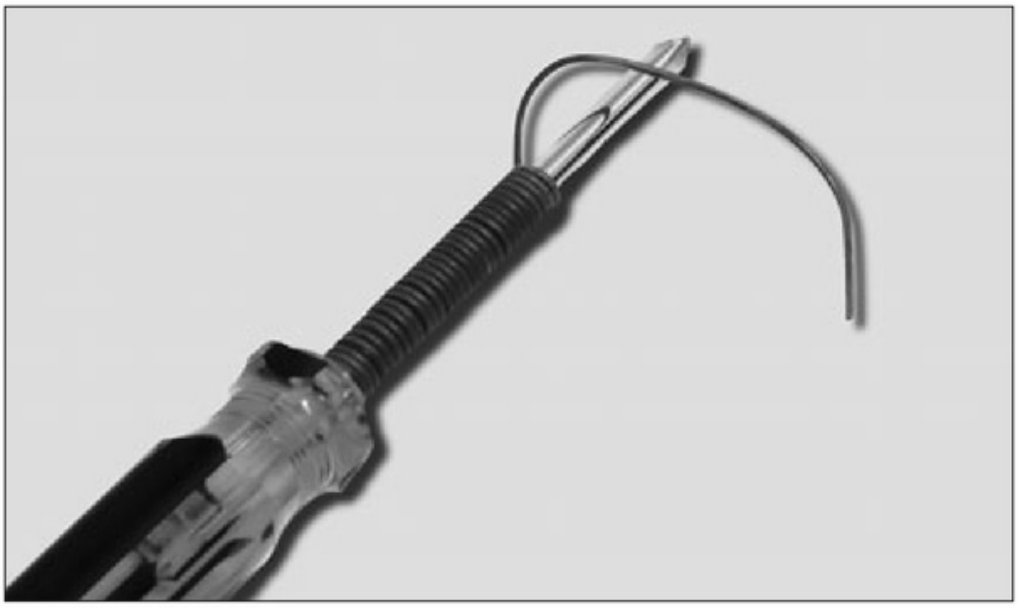


Figura 29. *Aquí vemos cómo el cuerpo del destornillador puede ser usado para formar el resorte.*

El resorte se colocará en el interior del dedo robótico. Esto significa que su diámetro debe ser lo suficientemente pequeño para poder introducirse en el tubo. Además, debe entrar con facilidad y no a presión.

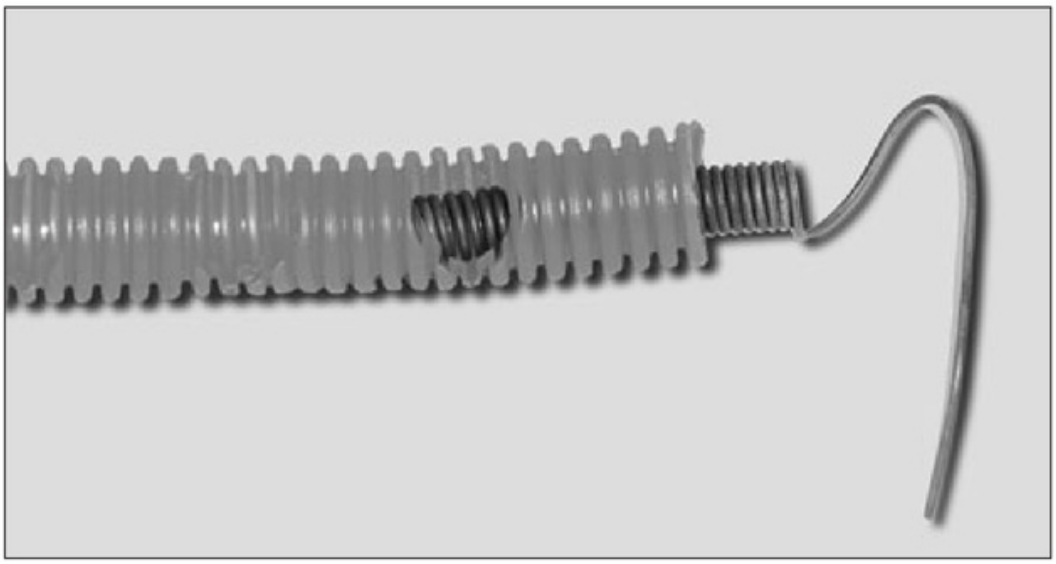


Figura 30. *Antes de construir un resorte completo, debemos comprobar, con un resorte pequeño, que el diámetro que hemos elegido sea el adecuado.*

El resorte que debemos construir o conseguir debe tener el mismo largo que el dedo robótico. En sus extremos, dejaremos varios centímetros de cable, que nos servirán para sujetarlo de manera firme al tubo.

Figura 31. *El tubo para el dedo robótico y el resorte deben tener el mismo largo.*

III EL USO DE GUANTES

Cuando construimos el resorte, es importante usar guantes para proteger la mano y los dedos, dado que esto evitará que nos lastimemos. Hay que ser cuidadosos siempre que trabajemos con herramientas. Si no deseamos construir el resorte, quizá sea posible mandarlo a hacer con un herrero.

Debemos usar el hilo en este momento. Pasamos el hilo por el interior del tubo y atamos su extremo en el orificio que perforamos antes. El hilo debe estar atado con firmeza mediante un nudo, ya que funcionará como tensor en el interior del tubo y, si se resbala, el dedo no podrá doblarse.

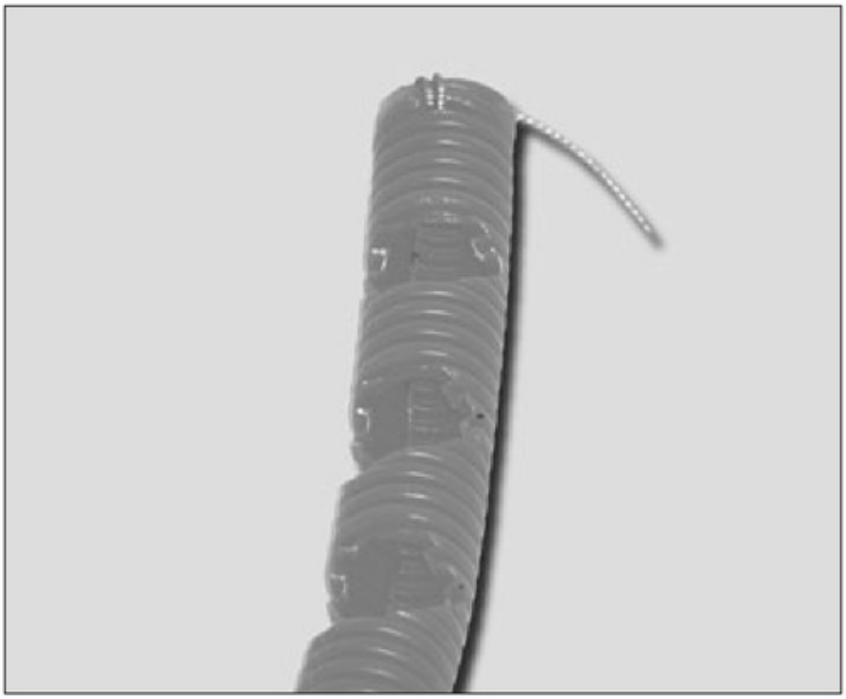


Figura 32. *Aquí podemos observar la forma en que se ha colocado el nudo en el orificio del tubo.*

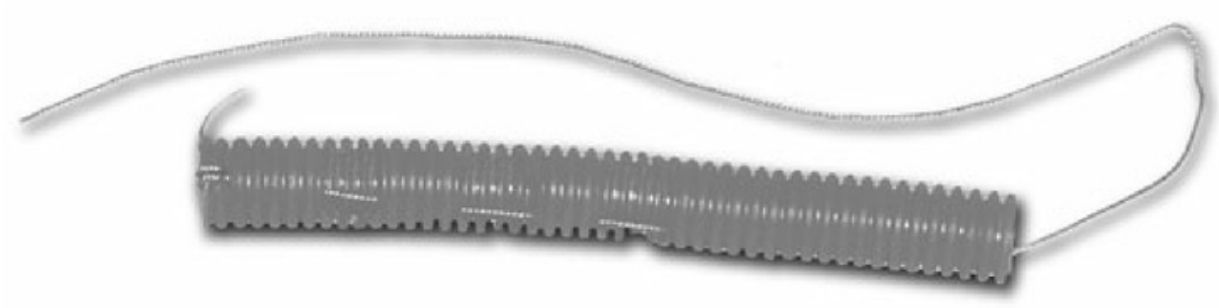


Figura 33. *El hilo atraviesa el interior del tubo hasta el orificio. Tenemos suficiente hilo que sale por el otro extremo.*

III PROBAR CON EL DEDO

Una vez atado el hilo, podremos probar la flexión del dedo. Para esto, simplemente sostenemos el dedo con una mano por el extremo contrario al orificio y, con la otra mano, jalamos del hilo. Veremos que el dedo se empieza a contraer alrededor de las articulaciones.

En este momento, introducimos el resorte en el tubo. Para facilitar esto, debemos mantener el hilo ligeramente tensionado. Hay que cuidar que el hilo no se enrosque con el resorte y que quede en la parte donde se encuentran los cortes que hicimos para las articulaciones.

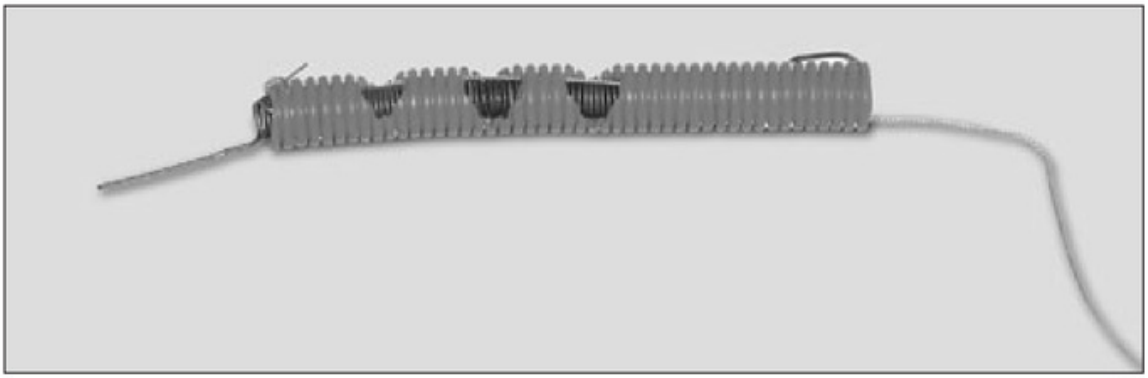


Figura 34. *El resorte ha sido introducido en el tubo, y el hilo se encuentra ubicado de manera correcta.*

Ahora, debemos doblar los extremos del resorte de tal manera que sostengan al tubo. No deben quedar demasiado ajustados, ya que es necesario cierto movimiento para que todo funcione bien.

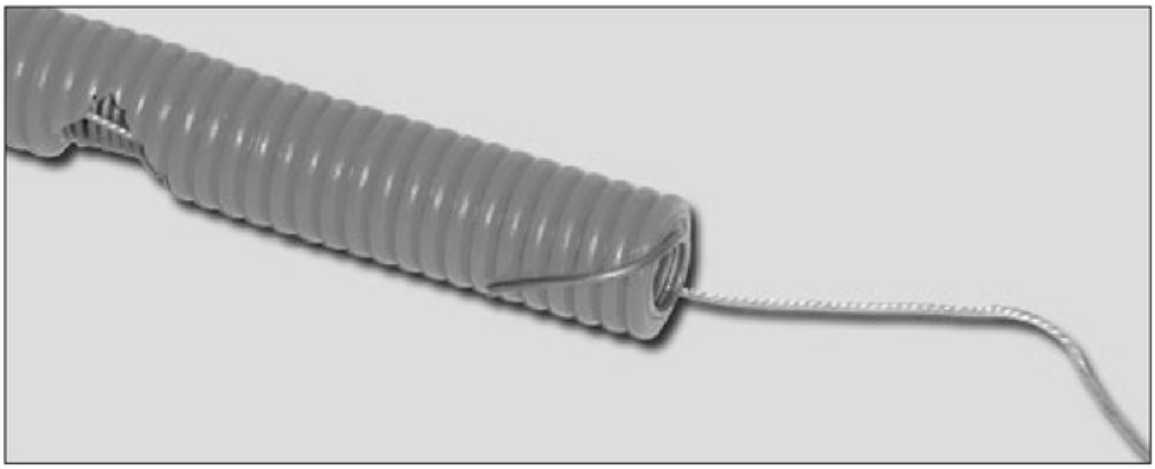


Figura 35. *Éste es uno de los extremos del tubo con el resorte ya ajustado.*

III EL NUDO EN EL DEDO

Debemos tener en cuenta que el nudo en el orificio debe ser firme, pero si el hilo se resbala de manera constante, podemos colocar una gota de pegamento en el nudo para hacerlo fijo. Lo mejor es crear un nudo que no sea corredizo y atarlo con firmeza. Cambiar el hilo por uno de otro material, también nos ayudará a evitar que se deslice.

Figura 36. *En el otro extremo del dedo robótico, también, doblamos el resorte para ajustarlo al tubo.*

Con el dedo robótico ya completo, podemos proceder a montarlo sobre la base. El montaje es muy sencillo, y empezamos por pasar el hilo a través de la caja que sostendrá el dedo en nuestra base. El hilo debe dirigirse desde el exterior hacia el lugar en donde se encontrará el eje del servo.

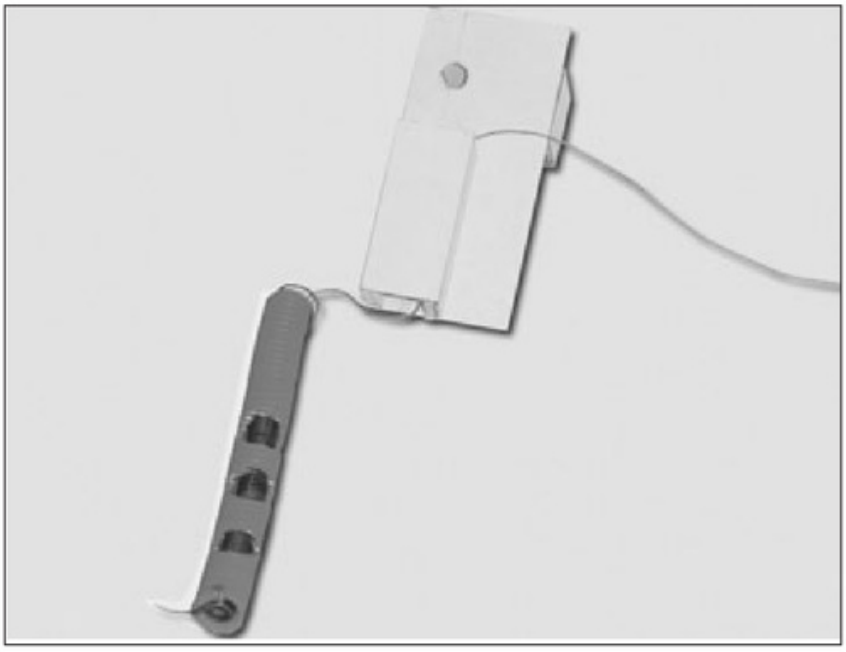


Figura 37. *Esta figura nos muestra cómo debemos pasar el hilo. El hilo debe salir por el extremo cercano al eje del servo.*

Ahora, introducimos el dedo robótico en el interior de la caja. Los espacios de las articulaciones deben quedar hacia arriba. El dedo debe meterse hasta que llegue al tope que hemos colocado en el interior de la caja. Debemos jalar el hilo para que éste no se enrede en el interior de la caja.

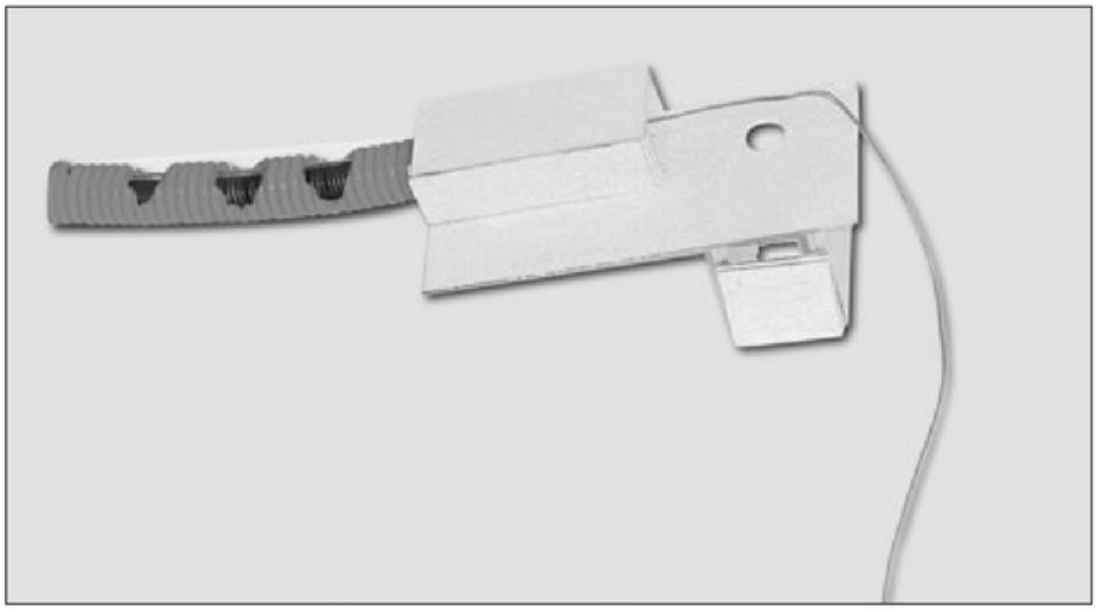


Figura 38. *Esta figura nos muestra la forma como debemos montar el dedo dentro de la caja. Debemos tener en cuenta que el hilo no tiene que enredarse en el interior.*

Los servos, por lo general, traen diferentes elementos de actuación. Es necesario utilizar un elemento de actuación que tiene forma recta. Este elemento presenta varias perforaciones por las cuales deberemos pasar el hilo. La siguiente figura nos muestra cómo tenemos que pasar el hilo por las perforaciones. Esto nos servirá para fijar el hilo al servo de tal forma que éste pueda tensarlo.

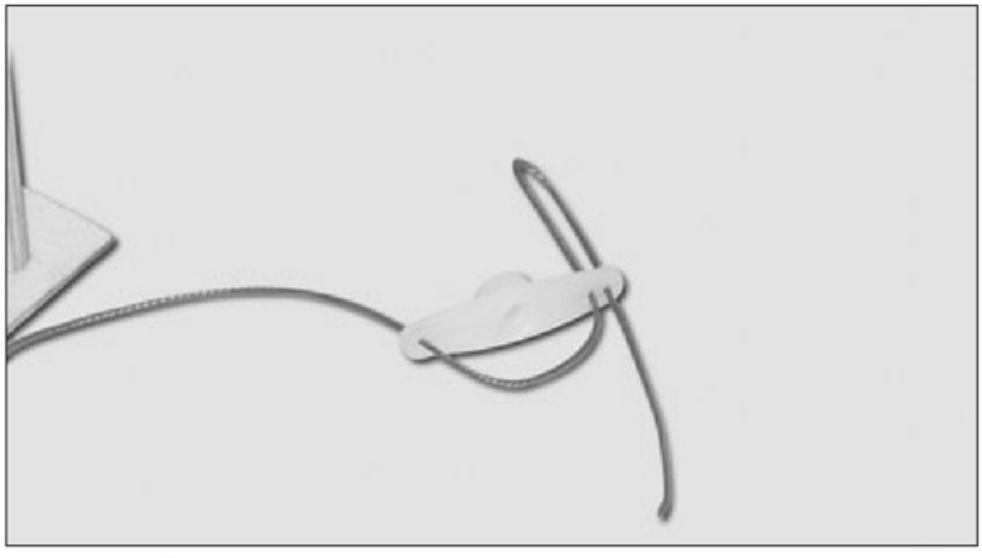


Figura 39. *Ésta es la forma como debemos pasar el hilo por el elemento de actuación para empezar a fijarlo. En este momento, aún no tensamos el hilo.*

Con el hilo ya pasado, colocamos otra vez el servo en la base. La rotación del servo debe estar dispuesta en su posición más pequeña. Sobre el eje del servo, colocamos el elemento de actuación. Aún no lo fijamos con el tornillo, pues no hemos tensado el hilo a su posición definitiva.

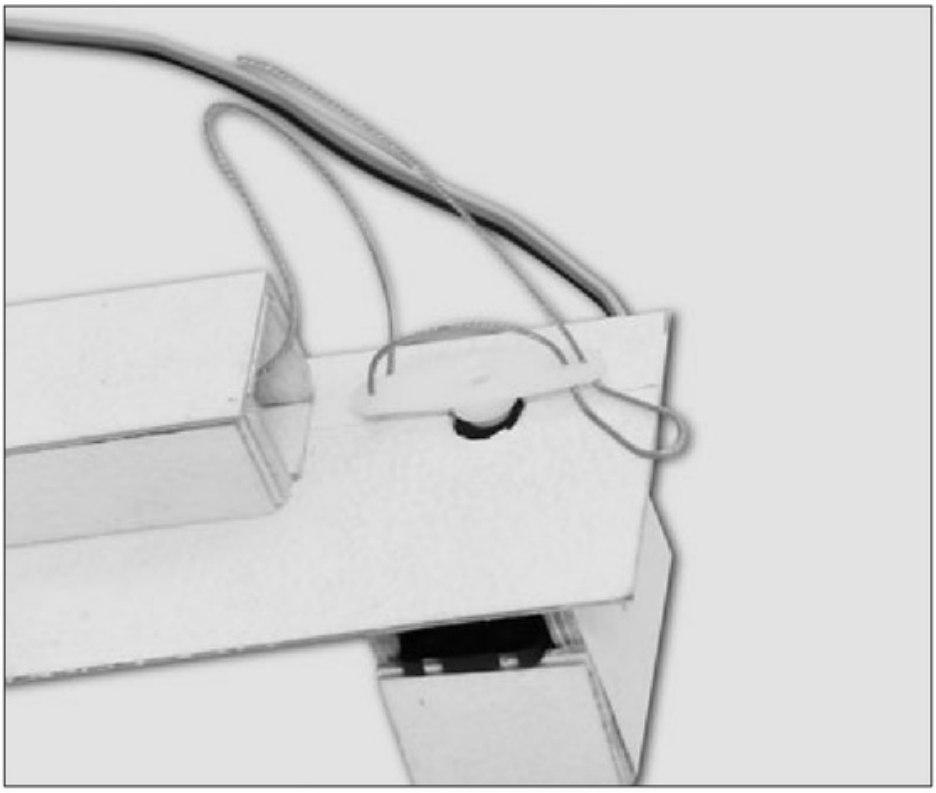


Figura 40. *El elemento de actuación se coloca sobre el servo. El hilo todavía no se debe tensar.*

Ahora empezamos a tensar con lentitud el hilo, pasándolo por los orificios del elemento de actuación. El hilo sólo debe tener una ligera tensión, que no debe ser tan grande que flexione el dedo robótico, pero, tampoco, tan pequeña que el hilo se curve.

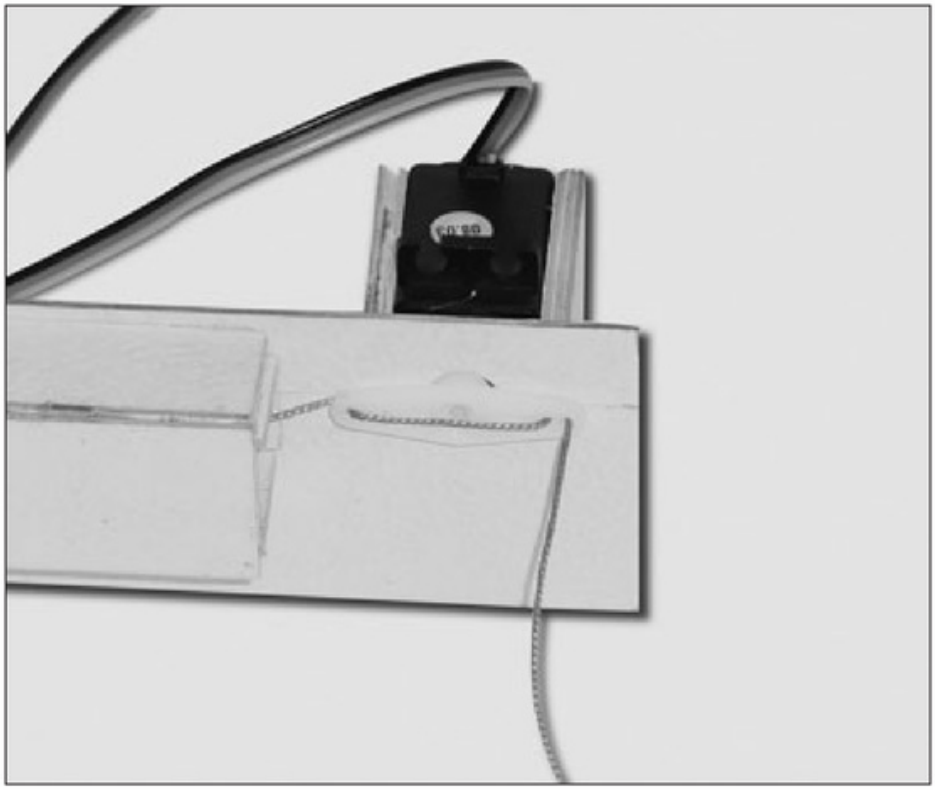


Figura 41. *Esta fotografía nos muestra cómo debe lucir el hilo, y su nivel de tensión.*

Con el hilo en su lugar, lo que nos falta hacer es un nudo en el extremo de tal forma que el hilo no pierda tensión cuando el servo gire. Con esto, finalizamos la creación del dedo robótico y procedemos al armado del guante.

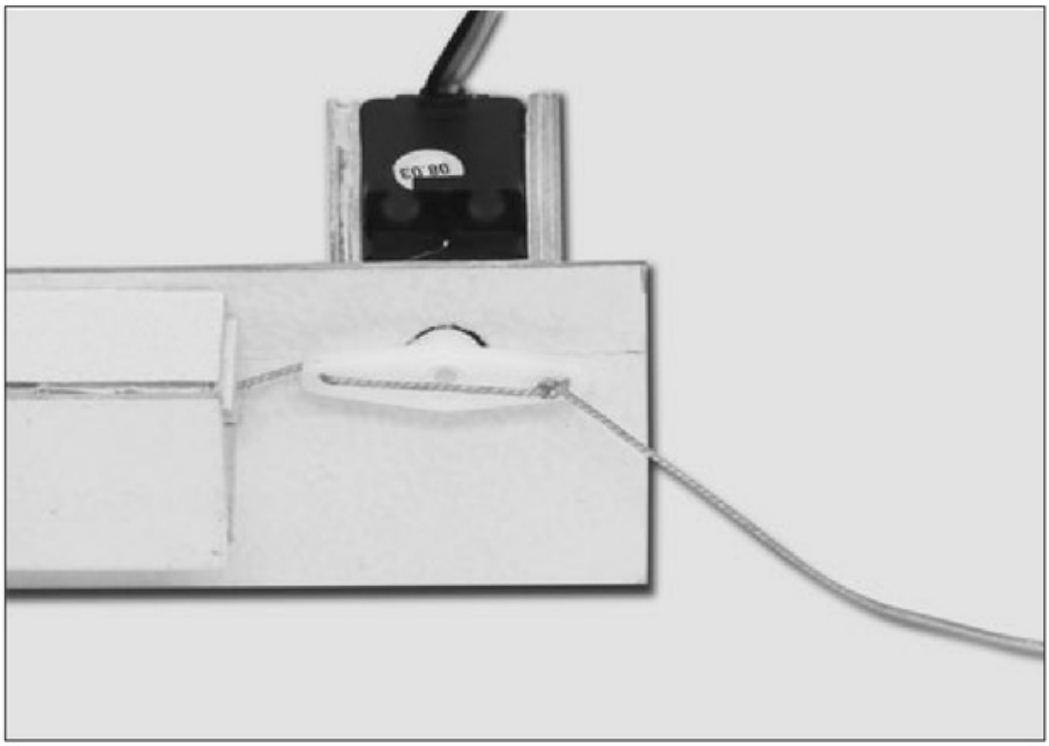


Figura 42. *Aseguramos el hilo por medio de un nudo en el extremo, para evitar su deslizamiento.*

Construcción del guante

Es el momento de empezar a construir el guante que controlará el dedo robótico. Para esto, necesitamos un guante. Como vamos a coserlo, es conveniente que sea de un material por el cual podamos pasar, con facilidad, la aguja. Para coser, vamos a utilizar hilo común, como el usado para coser los botones a las camisas. Necesitaremos, además, un cautín o punta para soldar, y una soldadura. Debemos tener a mano, también un cable para conectar el sensor al Phidget. El circuito que usaremos con el sensor es sencillo y precisaremos una resistencia de 1.2 kilo ohms.

COMPRA DEL SENSOR

El sensor se puede comprar por Internet en el siguiente sitio web: **www.robotshop.ca/**. Existen sensores de diferentes tamaños y distintas propiedades eléctricas. Si lo deseamos, podemos modificar el proyecto para colocar un sensor de flexión en su lugar, siempre teniendo cuidado de que doble en la dirección correcta.

Figura 43. *Éste es un ejemplo de guante que podemos utilizar, pero cualquier otro tipo que sea flexible puede funcionar bien.*

El sensor que utilizaremos es un sensor flexible de estiramiento. Estos sensores se caracterizan porque modifican su resistencia eléctrica dependiendo de cuánto se ha estirado el material. El sensor que necesitamos es de 2 pulgadas de longitud. Este sensor se ubicará en la parte superior del guante, más precisamente sobre el dedo. Cuando el dedo humano se flexione, el sensor se verá estirado, y su cambio en la resistencia eléctrica nos permitirá saber qué tanto se ha flexionado el dedo.

Figura 44. *Éste es el sensor que nosotros utilizaremos. Tiene como ventajas el ser barato y fácil de colocar en nuestro guante.*

Para construir nuestro guante, debemos ubicarle el sensor sobre su parte superior. Para fijarlo usaremos hilo. El dedo sobre el que estableceremos el sensor es el

índice. El primer paso que debemos realizar es coser una de las terminales del sensor al extremo cercano a la punta del dedo en el guante. La siguiente figura nos muestra esto.

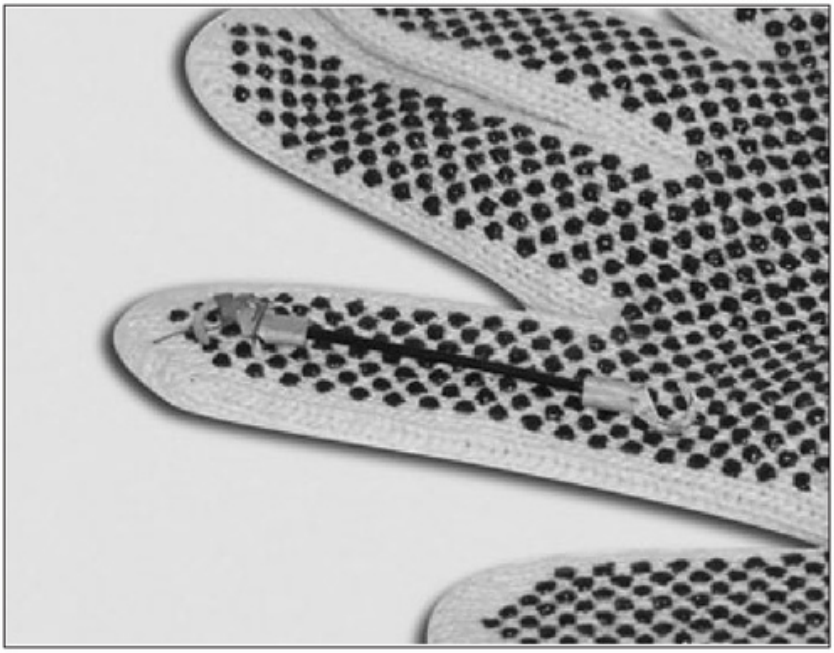


Figura 45. *El sensor ha sido cosido al guante cerca de la punta del dedo. La costura debe ser firme para evitar que el sensor se salga durante el movimiento del dedo.*

Dejamos que el sensor se relaje sobre el guante. Nos fijamos en el extremo que no ha sido cosido y, a una distancia aproximada de dos centímetros de dicho extremo, hacemos una marca. Ahora, cosemos el extremo libre del sensor en esta marca. El dedo del guante se doblará ligeramente para que podamos hacerlo.

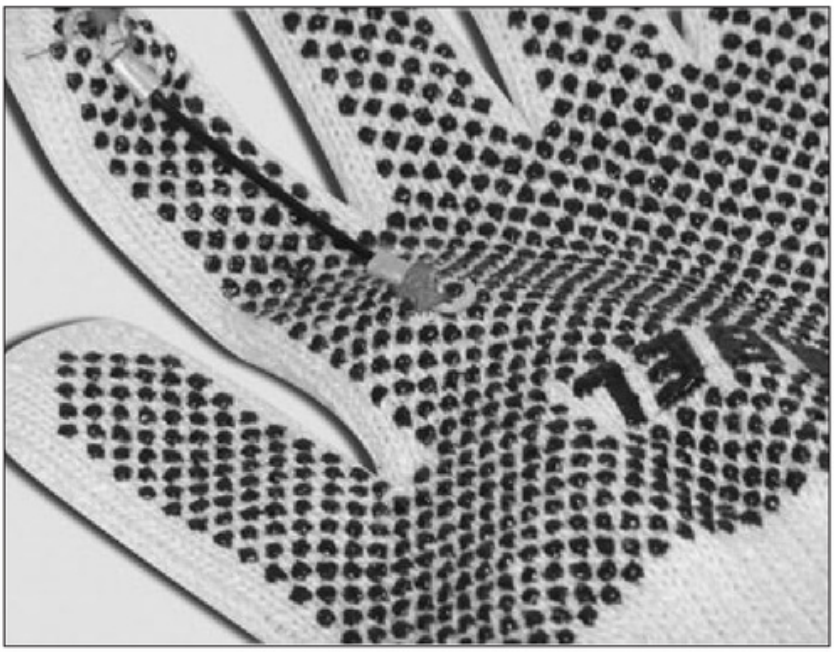


Figura 46. *El otro extremo del sensor también es cosido al guante, pero debemos hacerlo un poco más lejos de su posición relajada.*

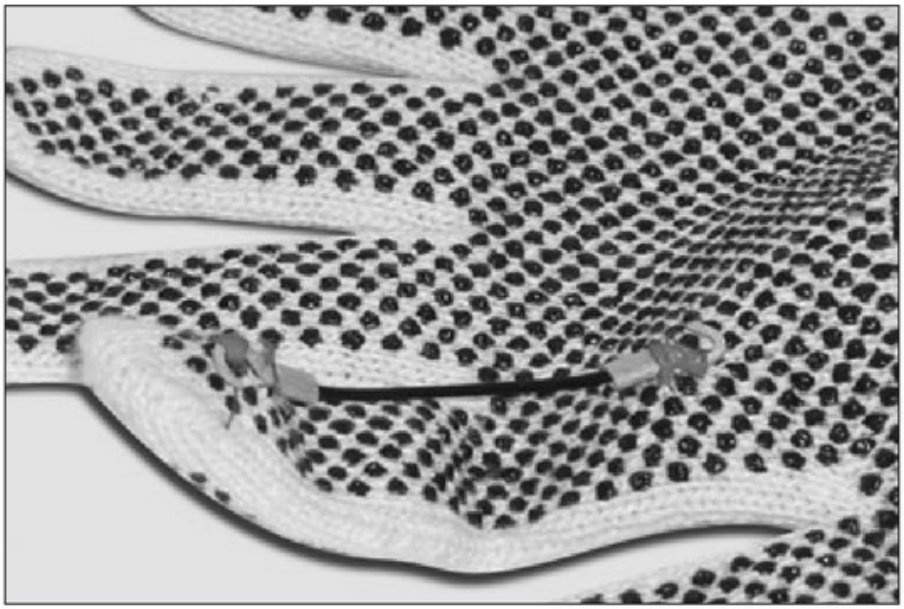


Figura 47. *El dedo del guante se ha doblado un poco para permitir coser el sensor en ambos extremos.*

El circuito que tenemos que construir es muy sencillo. El cable rojo estará soldado a una de las terminales del sensor; el blanco, a otra de las terminales y a la resistencia; y el negro, a la resistencia. En la siguiente figura, podemos observar el circuito.

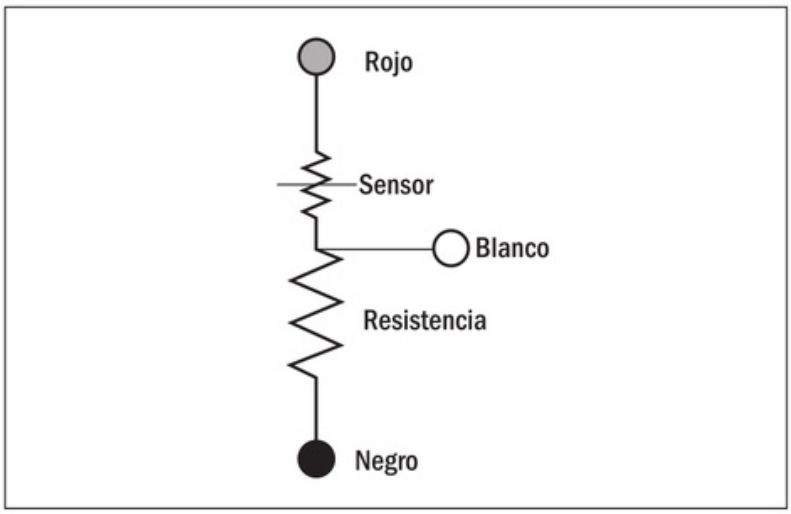


Figura 48. *Éste es el circuito que debemos crear para el guante.*

III MEJORAR LA SENSIBILIDAD DEL GUANTE

El guante que hacemos es muy sencillo, por lo que el sensor se puede mover, aunque no siempre se estira de una manera óptima. Un guante mejor diseñado, con un canal para el sensor, puede mejorar en forma considerable la medición de la flexión del dedo. Otro factor importante es la ubicación del sensor. Cuando éste se encuentra bien ubicado, se obtiene una mejor lectura.

Para realizar el circuito, usaremos el cable que tiene el conector para el 8/8/8, y lo cortamos a la mitad. Con cuidado, separamos cada uno de los hilos del cable. Tendremos de esta forma un hilo blanco, otro rojo y otro negro. Luego, pelamos las puntas de cada hilo para llegar al conductor eléctrico en su interior.

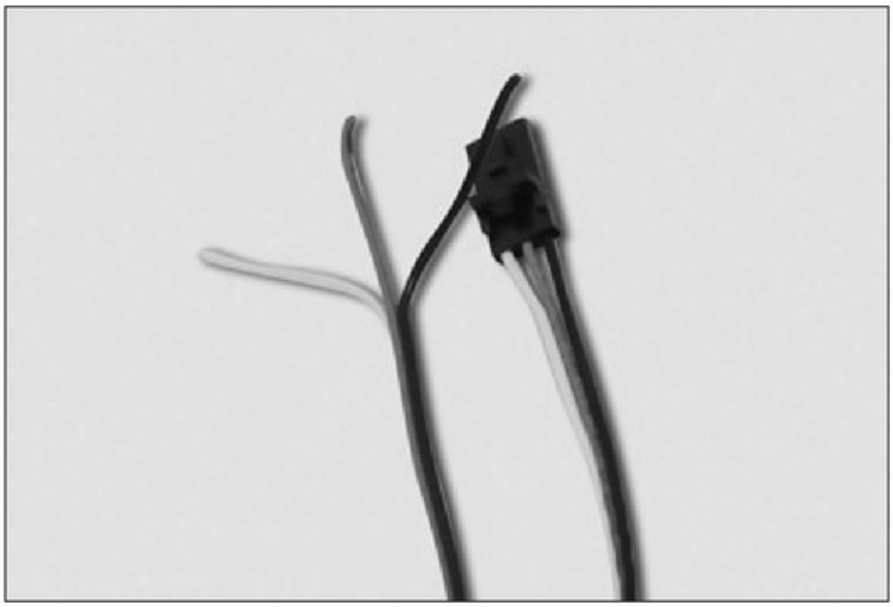


Figura 49. *Con cuidado hemos separado los hilos del cable. Cada uno de ellos irá soldado como se indica en el diagrama del circuito.*

Empezamos por soldar el conductor del hilo rojo al sensor. Lo soldaremos en la terminal que se encuentra cerca de la punta del dedo del guante. Si es necesario, para facilitarnos la soldadura, podemos separar más los hilos del cable.

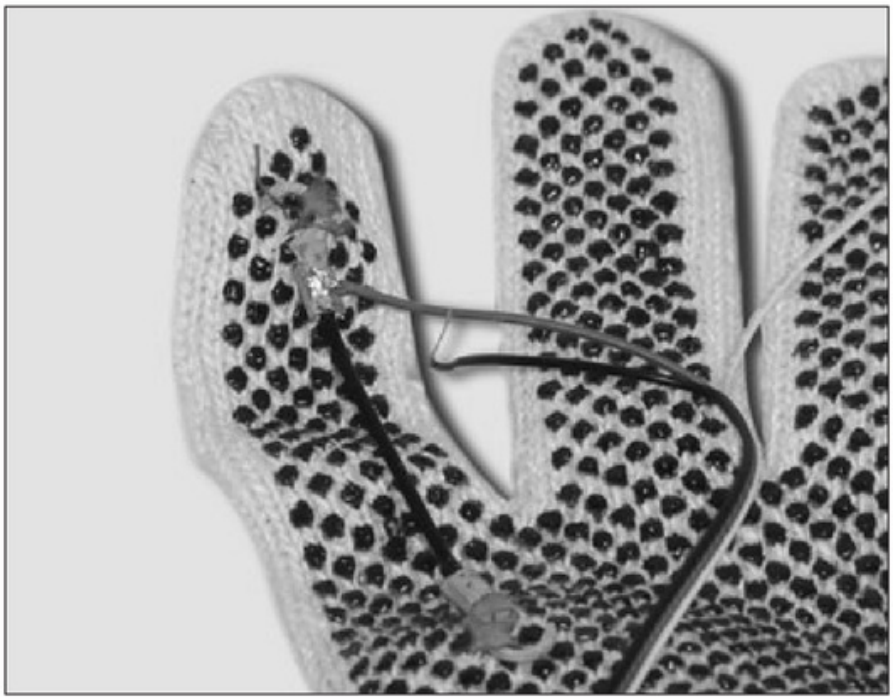


Figura 50. *El cable rojo es soldado a la terminal del sensor.*

En la otra terminal del sensor, habrá dos elementos soldados. Tenemos que soldar el conductor del hilo blanco y, también, una de las terminales de la resistencia. Ambos deben quedar soldados en ese punto.

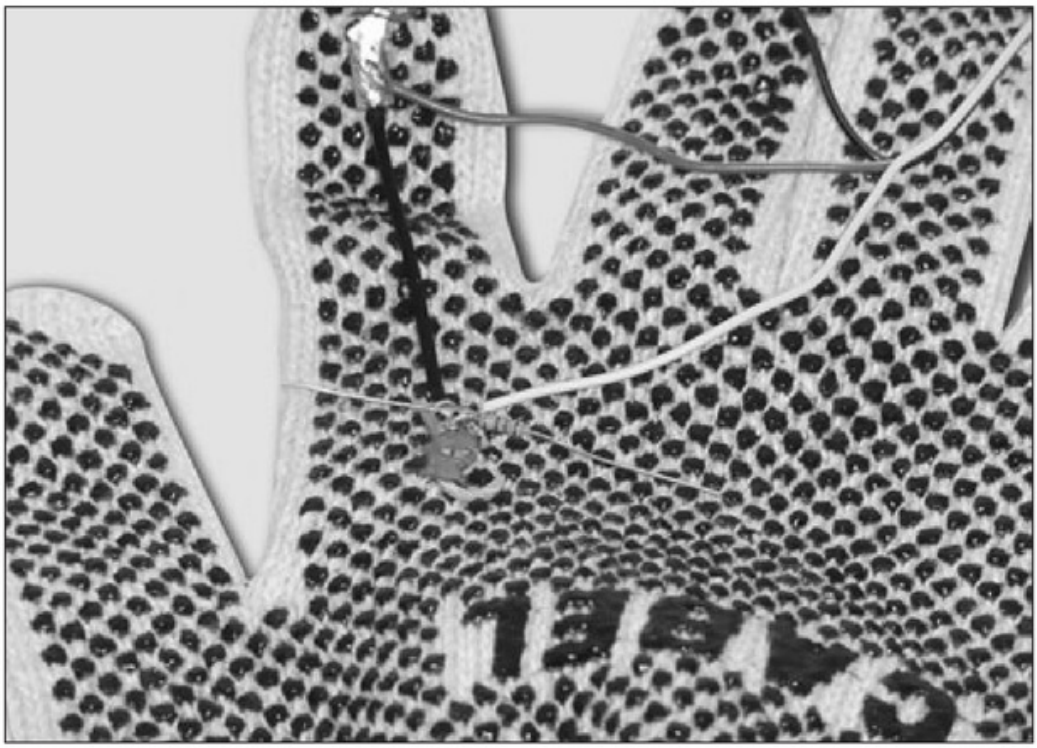


Figura 51. *La resistencia y el hilo blanco quedan soldados a la otra terminal del sensor.*

Como podemos ver en la siguiente figura, una de las terminales de la resistencia quedó soldada al sensor, pero la otra terminal se encuentra libre. El conductor del hilo negro debe quedar soldado en esta terminal de la resistencia.

Figura 52. *La otra terminal ha sido soldada al hilo negro.*

Con esto, hemos finalizado el hardware del proyecto, por lo que podemos continuar adelante con el desarrollo del software.

CREACIÓN DEL SOFTWARE

El código de nuestra aplicación está basado en el programa base que hace uso de los dos Phidgets; en él, llevaremos a cabo adiciones y modificaciones en algunos elementos. Los conceptos que ya son conocidos de los capítulos anteriores no serán comentados. Sólo explicaremos aquellos conceptos que sean nuevos o importantes.

Nuestra aplicación podrá mover el servo de forma manual o automática. Para mover el servo usaremos un **TrackBar**. Un **CheckBox** nos permitirá señalar cuando queremos hacer uso del control manual o ir en automático. Tendremos también un **ProgressBar** que mostrará, de manera gráfica, el valor de la resistencia del sensor, o el grado de flexión del dedo. Estos valores entran de forma analógica, y usaremos el Phidget para obtener el valor adecuado. Hay una serie de valores que necesitamos conocer para que la aplicación funcione lo mejor posible y para evitar que el servo se exceda en su movimiento. Debemos recordar que el servo se encuentra en su posición mínima cuando el dedo robótico está extendido.

Tendremos un concepto al cual llamaremos **tope**. El tope es el valor que tendrá el servo cuando el dedo robótico esté totalmente flexionado. Esto es necesario, ya que no queremos forzar el servo a moverse más allá de lo imprescindible, siendo que el dedo se encuentra flexionado. Para establecer el tope, moveremos el dedo robótico de forma manual y, al ver que ha logrado su máxima flexión, presionamos un botón. Como el sensor puede tener diferentes valores de resistencia y podemos usar distintos sensores, lo mejor resulta indicar cuál es el valor de la resistencia cuando el dedo del guante se encuentra relajado y cuando está flexionado. El concepto de **menor** será el valor de la resistencia del sensor cuando el dedo se encuentra relajado, y el concepto de **mayor**, el del valor de la resistencia del sensor cuando el dedo está

III LAS TERMINALES DE LA RESISTENCIA

Las resistencias no tienen polaridad, por lo que podemos soldar cualquiera de sus terminales al sensor. Los anillos de colores que presentan nos permiten saber el valor de la resistencia y cada color de dichos anillos representa un valor numérico. Para soldar correctamente, primero debemos limpiar las terminales de la resistencia.

flexionado. Por medio de una fórmula muy sencilla, se interpolará el valor de la resistencia a un valor de giro del servo.

La interfaz de usuario

Ahora, tenemos que concentrarnos en la creación de la interfaz de usuario. Por fortuna, ésta resulta más sencilla que la del capítulo anterior. La siguiente figura muestra cómo debe lucir la interfaz que crearemos.

Figura 53. *Ésta es la interfaz de usuario que tendrá nuestra aplicación del dedo robótico.*

Empecemos creando la parte que se encarga del movimiento del dedo robótico. Necesitamos crear un **GroupBox** en la esquina superior izquierda. Ingresamos, en su propiedad **Text**, el valor de **"Posicion Servo"**. En su interior, ubicamos un **TrackBar** con el nombre de **trkPosicion**. La propiedad de **mínimo** con el valor de **30** y la de **máximo**, con **210**. Del lado derecho del **TrackBar**, agregamos un **Label** llamado **lblPosicion** con un valor de **30** en su propiedad **Text**. En la parte inferior del **TrackBar**, insertamos un **CheckBox** con el valor de **"Control manual"** en su propiedad **Text** y, de **True**, en la propiedad **Checked**. En la parte inferior y del lado derecho del **TrackBar**, ubicamos un botón llamado **btnTope**. Este botón nos permi-

III EL PROGRESSBAR

El **ProgressBar** es un control que nos permite apreciar de forma visual el porcentaje completado de un proceso, pero, en este caso, lo usamos para poder observar el porcentaje de flexión del dedo. Hacia el final del capítulo, aprenderemos a modificar sus valores de **Maximum** y **Minimum** para que nuestro rango de movimiento sea representado por toda la longitud del control.

tirá colocar el valor del tope dependiendo del valor actual del **TrackBar**. En la propiedad **Text**, agregamos el valor de **"Poner tope"**.

Tenemos que agregar un **GroupBox** para la información de las entradas analógicas. Este **GroupBox** lleva, en la propiedad **Text**, el valor de **"Entradas Analógicas"**. Insertamos una columna de etiquetas con los valores: **"Analogico 0:"**, **"Valor menor:"**, **"Valor mayor:"**. Junto a esta columna, agregaremos otras etiquetas. Todas tienen el valor de **0** en su propiedad **Text**, y para sus nombres ingresaremos los siguientes títulos: **lblAnalogico0**, **lblValorMenor**, **lblValorMayor**. También colocaremos dos botones que nos permitirán determinar el valor menor y el mayor de la resistencia del sensor. El primer botón se llama **btnPonerMenor** con el valor de **"Poner menor"** en su propiedad **Text**. El segundo botón lleva por nombre **btnPonerMayor**, y, en su propiedad **Text**, establecemos **"Poner mayor"**.

Para finalizar la interfaz de usuario, agregamos un nuevo **GroupBox** al cual le escribiremos en su propiedad **Text** el valor de **"Movimiento Dedo"**. Debemos ubicar **GroupBox** en la esquina inferior derecha de la forma. En la parte superior, y centrada en el **GroupBox**, ponemos una etiqueta llamada **lblPorcentaje**, con el valor de **"%"** en la propiedad **Text**. En ese mismo **GroupBox**, agregamos un **ProgessBar**, al cual llamaremos **pbarDedo**. Su valor **Maximum** debe ser establecido en **1000**. El tamaño es de **234** por **68** en la propiedad **Step**, insertamos el valor **1**. Inicialmente, establecemos el valor **120** en la propiedad **Value**. Empecemos a ver el código de la forma y el inicio de la clase correspondiente:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

// Necesario para el registro
using Microsoft.Win32;

// Necesario para los Phidgets
using Phidgets;
using Phidgets.Events;
```

```
using System.Threading;

namespace BasePhidgets
{
    public partial class Form1 : Form
    {
        // Datos necesarios para la aplicacion base

        private int numeroSerie0;        // Numero de serie del phidget 0
        private int numeroSerie1;        // Numero de serie del phidget 1
        private bool conectado0;         // Para saber si hay conexion con
el phidget 0
        private bool conectado1;         // Para saber si hay conexion con
el phidget 1
        private InterfaceKit iKit;       // Kit de interfaz
        private AdvancedServo mServo;  // Controlador de servos

        // Datos necesarios para el control del dedo robotico
        private double resistencia;      // valor actual de la resistencia
        private double menor;             // valor menor de la resistencia
        private double mayor;             // valor mayor de la resistencia
        private int tope;                // Valor maximo que el servo gira
para mover el dedo
        private double posCalculada;     // Posicion en la que debe estar
el servo en relacion a la resistencia
```

Los datos nuevos que tenemos en este código están relacionados con la aplicación. Podemos ver una variable que guardará el valor de la resistencia del sensor, según lo lea el Phidget. También, tenemos las variables de menor y mayor tal y como las comentamos con anterioridad. La variable de tope también es declarada. Tenemos una variable extra que se llama **posCalculada**. Esta variable será usada para guardar el valor de la posición que debe tener el servo con relación al valor de la resistencia. Veamos, ahora, el constructor de la forma.

```
        public Form1()
        {
            InitializeComponent();

            // Inicializamos la aplicacion
```

```
            // Empezamos como si no estuvieramos conectados
            conectado0 = false;
            conectado1 = false;

            // Colocamos un valor de default para el numero de serie
            numeroSerie0 = 0;
            numeroSerie1 = 0;

            // Verificamos si existe la llave en el registro
            RegistryKey sfwKey = Registry.LocalMachine.OpenSubKey
("Software");
            RegistryKey redKey = sfwKey.OpenSubKey("RedUsers2");

            // Verificamos si existe
            if (redKey == null) // la llave no existe
            {

                // Creamos la llave
                sfwKey = Registry.LocalMachine.OpenSubKey("Software",
true);
                redKey = sfwKey.CreateSubKey("RedUsers2");

                RegistryKey robotKey0 = redKey.CreateSubKey("Robots2");


                // Colocamos un valor de default
                robotKey0.SetValue("NumeroSerie0", (object)numeroSerie0);
                robotKey0.SetValue("NumeroSerie1", (object)numeroSerie1);

            }
            else // la llave existe
            {
                RegistryKey robotKey0 = redKey.OpenSubKey("Robots2");

                // Obtenemos el numero de serie guardado
                numeroSerie0 = (int)robotKey0.GetValue("NumeroSerie0");
                numeroSerie1 = (int)robotKey0.GetValue("NumeroSerie1");
            }

            // Inicializamos los valores de la aplicacion
```

```
            menor = 0.0;
            mayor = 1000.0;
            tope = 210;

        }
```

Lo único nuevo que encontramos aquí es la inicialización de los valores de las variables que usaremos. El valor de tope queda en **210**, dado que este valor es el máximo de la posición del servo que usamos. Si se utiliza otro modelo de servo, entonces, tenemos que ajustar este valor. El handler para el evento **FormClosing** necesita de unos ajustes que vemos a continuación.

```
        private void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {
            // Quitamos los handlers
            iKit.Attach -= new AttachEventHandler(iKit_ConectarHandler);
            iKit.Detach -= new DetachEventHandler(iKit_Desconectar
Handler);
            iKit.Error -= new ErrorEventHandler(iKit_ErrorHandler);

            iKit.SensorChange -= new SensorChangeEventHandler
(iKit_AnalogoHandle);

            // Regresamos el servo al centro
            if (mServo.Attached == true)
            {
                mServo.servos[0].VelocityLimit = 1000;
                mServo.servos[0].Engaged = true;
                mServo.servos[0].Position = 30;
                Thread.Sleep(500);
                mServo.servos[0].Engaged = false;

                mServo.Attach -= new AttachEventHandler
(mServo_ConectarHandler);
                mServo.Detach -= new DetachEventHandler
(mServo_DesconectarHandler);
                mServo.Error -= new ErrorEventHandler(mServo_Error
Handler);
            }
```

```
            Application.DoEvents();

            // Cerramos el phidget
            iKit.close();
            mServo.close();

        }
```

El cambio que se presenta es ubicar al servo en la posición más baja. Esto lo hacemos para que, cuando finalice la aplicación, el dedo robótico se relaje. De esta manera, evitamos tensión en el servo cuando no se encuentra en uso. El handler para el valor analógico del Phidget será usado para llevar a cabo los cálculos que son necesarios en la aplicación.

```
        public void iKit_AnalogoHandle(object sender, SensorChangeEventArgs e)
         {
            // Verificamos que estemos conectados
            if (conectado0)
            {
                // Verificamos que sea la entrada 0
                if (e.Index == 0)
                {
                    // Colocamos el valor
                    lblAnalogico0.Text = e.Value.ToString();
                    resistencia = e.Value;

                    if(pbarDedo.Minimum<resistencia &&
pbarDedo.Maximum>resistencia)
                        pbarDedo.Value = (int)resistencia;
                }


                // Calculamos rango de movimiento del servo
                double rangoServo = tope - mServo.servos[0].PositionMin;

                // Calculamos el rango de la resistencia
                double rangoResistencia = mayor - menor;
```

```
                // Calculamos el porcentual de la posicion de la
resistencia
                double posResistencia = (resistencia - menor) /
rangoResistencia;

                // Verificamos que el rango este correcto
                if (posResistencia < 0.0)
                    posResistencia = 0.0;
                else if (posResistencia > 1.0)
                    posResistencia = 1.0;

                lblPorcentaje.Text = (posResistencia * 100.0).ToString()
+ " %";

                // Calculamos el valor para el servo
                posCalculada = (rangoServo * posResistencia) +
mServo.servos[0].PositionMin;

                // Si no estamos en manual, movemos el servo
                if (chkControlManual.Checked == false)
                {

                    mServo.servos[0].Position = posCalculada;
                    mServo.servos[0].Engaged = true; ;
                    Thread.Sleep(500);
                    mServo.servos[0].Engaged = false;
                    trkPosicion.Value = (int)posCalculada;
                }

            }
        }
```

III FORMA DE CONFIGURAR EL MÁXIMO

En la configuración del máximo, debemos ponernos el guante y doblar el dedo. A continuación, vemos que se modifica el valor de la resistencia. Cuando la resistencia llega a su mayor valor, entonces, presionamos el botón para establecer el máximo. También aquí, un valor promediado puede ayudarnos a lograr aún mejores lecturas.

Como acabamos de ver, lo primero que hacemos en el handler es verificar si el Phidget se halla conectado, porque sólo llevaremos a cabo los cálculos si el Phidget se encuentra en dicha condición, de otra forma, no podremos leer el valor del sensor. El sensor se localiza en el índice **0**. Si ésta es la entrada analógica que generó el evento, entonces, procedemos a actualizar la etiqueta y el valor de la resistencia. Por seguridad, verificamos que el valor de la resistencia no se salga del rango acordado. Una vez verificado esto, ingresamos el valor actual de la resistencia en el **ProgressBar**. Tenemos una variable llamada **rangoServo**. Esta variable guarda el valor del rango de movimiento del servo desde su posición menor hasta el tope. De forma similar, calculamos el rango de la resistencia; para esto, usamos el valor mayor y menor de ella.

Luego, calculamos la posición de la resistencia. Esta posición está dada como un porcentaje que depende del rango de la resistencia y de su valor actual dentro de este rango. Por seguridad, verificamos que la posición se encuentre dentro del rango correcto de valores. Como es un valor porcentual, nuestro rango debe ser de **0.0** a **1.0**. Con este valor, actualizamos la etiqueta que nos indica el porcentaje de flexión del dedo. La razón de multiplicar el valor por 100 es para dejarlo en términos porcentuales. Después, procedemos a calcular la posición del servo. El valor calculado quedará en la variable **posCalculada**.

Para mover el servo, verificamos que no nos encontremos en el modo manual. Ingresamos la posición calculada en la posición del servo, luego, simplemente movemos el servo a la nueva posición y, para finalizar, actualizamos la posición del **TrackBar**. En este momento, empezaremos a trabajar con los handlers relacionados con el **TrackBar**. Primero, ingresaremos el código para el evento **Scroll**. Este código es muy similar al del programa anterior, pero lleva un cambio relacionado con la variable **tope**.

```
private void trkPosicion_Scroll(object sender, EventArgs e)
{
    // Verificamos si esta conectado
    if (conectado1)
    {
        // Verificamos que estamos en control manual
        if (chkControlManual.Checked == true)
        {
            // Verificamos que no exceda el tope
            if (trkPosicion.Value < tope)
            {
                // Obtenemos el valor y lo mostramos
```

```
                    lblPosicion.Text = trkPosicion.Value.ToString();

                    // Colocamos la posicion en el servo
                    mServo.servos[0].Position = trkPosicion.Value;
                }
                else
                    trkPosicion.Value = tope;
            }

        }

    }
```

Vemos que hemos ingresado un poco de código de seguridad para que el valor de la posición del **TrackBar** no exceda el valor **tope**. Esto es necesario para evitar forzar el servo más allá de la flexión máxima del dedo robótico. El otro handler está relacionado con el evento **LocationChanged**, y no lo mostramos en este momento porque no necesita de ningún cambio. Hay un handler que sí debemos modificar y es el de la conexión del Phidget controlador de servos.

```
    public void mServo_ConectarHandler(object sender, AttachEventArgs e)
    {
        // Indicamos que hay conexion
        conectado1 = true;

        // Enviamos mensaje de la conexion
        lblPhidget1.Text = e.Device.Name;

        // Indicamos con color verde que hay conexion
        txtPhidget1.BackColor = System.Drawing.Color.Lime;



        // Configuramos sus caracteristicas
        if (mServo.Attached)
        {
            // Configuramos el servo 0

            // Colocamos la aceleracion
```

```
            mServo.servos[0].Acceleration = 1000.0;

            // Colocamos la posicion inicial
            mServo.servos[0].Engaged = true;
            mServo.servos[0].Position = 30.0;
            mServo.servos[0].VelocityLimit = 1000.0;

            // Colocamos los limites
            mServo.servos[0].PositionMin = 30.0;
            mServo.servos[0].PositionMax = 210.0;


        }

    }
```

Como podemos ver, el cambio es muy pequeño. Apenas estamos ingresando el valor **30** en la posición, para que el servo se mueva a su estado menor cuando sea conectado. Esto hace que el dedo inicie en una posición relajada. Ahora, debemos crear el handler para el botón **btnPonMenor**.

```
    private void btnPonMenor_Click(object sender, EventArgs e)
    {
        // Colocamos el valor menor de la resistencia
        menor = resistencia;
        pbarDedo.Minimum = (int)menor;
        lblValorMenor.Text = menor.ToString();
    }
```

Este handler es muy sencillo. Simplemente, toma el valor actual de la resistencia y lo ubica en la variable **menor**. Luego, le ingresamos, al **ProgressBar**, este valor en su propiedad **Minimum** y mostramos el valor en la etiqueta correspondiente. Hacemos algo similar para el botón **btnPonMayor**.

```
    private void btnPonMayor_Click(object sender, EventArgs e)
    {
        // Colocamos el valor mayor de la resistencia
        mayor = resistencia;
```

```
            pbarDedo.Maximum = (int)mayor;
            lblValorMayor.Text = mayor.ToString();
        }
```

En este caso, ubicamos el valor de la resistencia en la variable **mayor** y lo asignamos a la propiedad **Maximum** del **ProgressBar**. También, actualizamos la etiqueta relacionada con mayor. El handler para el botón de **btnTope** es muy sencillo.

```
        private void btnTope_Click(object sender, EventArgs e)
        {
            // Colocamos un nuevo valor para el tope
            tope = trkPosicion.Value;
        }
```

Como podemos observar, sólo se trata de una asignación. Tomamos el valor de la posición actual del **TrackBar** y se lo asignamos a la variable **tope**.

Con esto, nuestro programa ya está terminado. Podemos compilarlo y probarlo con el hardware que, también, hemos diseñado. En el siguiente capítulo, crearemos un proyecto interesante que hace uso del sonar para detectar objetos.

... RESUMEN

El proyecto desarrollado en este capítulo consiste de tres partes: un dedo robótico, un guante con sensor y el software de control. El dedo robótico es construido mediante un tubo flexible y un resorte, y el hilo funciona como tensor para que pueda flexionarse. Un servo jala del hilo y, al controlar el servo, cambiamos la flexión del dedo. El guante tiene un sensor maleable que varía su resistencia. Podemos mapear esta resistencia a una flexión del dedo. Aprendimos que se usan dos Phidgets, uno para controlar el servo y el otro para leer el sensor. El software tiene controles que nos permiten ajustar el dedo y el guante. También se encarga de realizar los cálculos y de comunicarse con los Phidgets.

TEST DE AUTOEVALUACIÓN

1. Cambie los valores de espera para el servo para que el dedo responda más rápido.

2. Modifique el software para que sea más sensible al cambio de resistencia del sensor.

3. Ajuste el resorte para facilitar la flexión del dedo.

4. Adicione más dedos al proyecto.

5. Modifique el dedo y aumente la cantidad de servos por dedo para tener una flexión más controlada.

6. Modifique el software para que pueda grabar el movimiento del dedo en el guante y, luego, que el dedo robótico lo repita.

7. Varíe el programa para que se autoajuste para el mínimo y el máximo de la resistencia del sensor.

8. Modifique el programa para que guarde, en el registro, el valor del tope y no tener que indicarlo cada vez que se inicia el programa.

9. Busque, en la documentación de Phidgets, los métodos para leer valores de las entradas analógicas.

10. Cree un guante con más sensores.

11. Use el guante para controlar otros dispositivos que no sean los dedos de un robot.

Sonar robótico

Hemos realizado distintos proyectos interesantes en los capítulos anteriores y, en este capítulo, construiremos un sonar robótico. El sonar nos permite medir las distancias por medio del eco producido por los objetos. A lo largo de todo el capítulo, veremos cómo el robot automatizará el proceso y escaneará 180 grados alrededor de él desde dos posiciones distintas.

DESCRIPCIÓN DEL PROYECTO

El robot estará constituido por una plataforma sobre la cual colocaremos un brazo robótico muy sencillo. En el extremo del brazo robótico, ubicaremos un sonar. El brazo tendrá dos posiciones, de tal forma que el sonar pueda hacer su escaneo desde dos lugares diferentes. El escaneo del sonar es de 180 grados en cada una de las posiciones. El sonar se moverá de grado en grado y hará una medición de la distancia en cada uno de ellos. Esta información se guarda por un tiempo y, después, se usa para mostrar de manera gráfica, las distancias encontradas. El proceso está totalmente automatizado, por lo que sólo deberemos oprimir un botón y observar los resultados.

Componentes necesarios

Este proyecto necesita de muchos componentes, que vemos a continuación:

NOMBRE	CANTIDAD	DESCRIPCIÓN
Controlador servos	1	PhidgetAdvancedServo
Phidget 8/8/8	1	Phidget 8/8/8
Servo	2	Servos para mover el sonar
Base	1	Bloque de madera como para base del proyecto
Sonar	1	LV-MaxSonar-EZ1 o equivalente
A	2	Cartón 8 x 29 cm, 1 mm de espesor
B	14	5 x 4 cm, 1 mm de espesor
C	1	2 x 2 cm, 5 mm de espesor
D	1	2 x 2 cm, 2 mm de espesor
E	2	4 x 4.5 cm, 2 mm de espesor
F	1	4.5 x 2 cm, 2 mm de espesor
G	3	9.5 x 2.5 cm, 3 mm de espesor
H	1	2.5 x 2 cm, 3 mm de espesor
I	2	2.5 x 1 cm, 2 mm de espesor
J	1	12 x 7 cm, 1 mm de espesor
K	1	2.5 x 2.5 cm, 3 mm de espesor
L	1	4 x 4 cm, 1 mm de espesor
M	2	2 x 1 cm, 3 mm de espesor

Tabla 1. *En esta tabla observamos todos los componentes necesarios para el proyecto.*

También, podemos conseguir varias bandas elásticas que nos servirán para ayudar a sujetar los servos y la base. En el capítulo anterior, usamos la mitad del cable analógico para el sensor; en éste, usaremos la otra mitad para instalar el sonar. También, será necesario tener soldadura y un cautín o soldador, dado que es preciso soldar el circuito del sonar al cable.

CONSTRUCCIÓN DEL SONAR

Comenzamos con este proyecto por medio de la construcción del brazo. Para el brazo, usaremos los componentes **A**. Estos componentes no serán usados en su forma rectangular: tendremos que llevar a cabo algunos cortes para darles la forma que deseamos. Para facilitar el corte, debemos hacer unos trazos sobre el cartón. Empezamos trazando una serie de marcas. En el lado izquierdo del cartón, establecemos una marca a 4 cm del borde superior. Luego, en el lado superior, hacemos otra marca a 4 cm de distancia del borde izquierdo. El borde inferior también debe ser marcado. En este borde, ubicamos una marca a 8 cm de distancia del borde izquierdo. Por último, una marca a 7 cm de distancia del borde derecho. La forma como queda marcado el componente **A** se muestra en la siguiente figura.

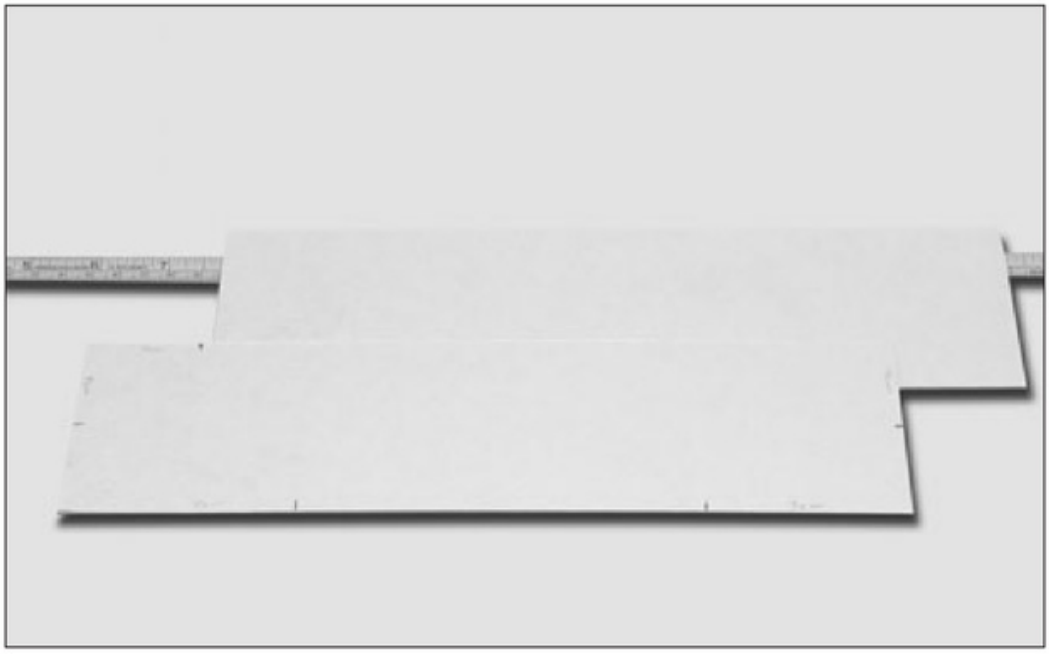


Figura 1. *Aquí podemos observar cómo han quedado las marcas en las distancias correspondientes.*

A partir de estas marcas, trazamos una serie de líneas verticales y una línea horizontal. En cada marca, empezamos la línea y cruzamos todo el componente **A** con ella.

Figura 2. *Tenemos que trazar líneas que atraviesen el componente de forma vertical y horizontal.*

Ahora, usando estas líneas como guías y prestando especial atención a la **figura 3**, llevamos a cabo el trazado de la forma del cuerpo del brazo para el sonar robótico. Lo mejor en este caso es delinear con un lápiz de color más oscuro para poder distinguir bien la forma del brazo.

Figura 3. *Este trazado muestra la forma del brazo y será la guía para el recorte del componente* ***A***

El primer recorte es el de la parte superior del brazo. Empieza en la diagonal superior y, luego, sigue de forma horizontal hasta finalizar el brazo. Para facilitar el corte, se puede hacer en dos etapas: primero, la diagonal y, luego, la horizontal.

Figura 4. *Observamos el componente* ***A*** *con la parte superior removida. Es importante notar que se corta la sección de la diagonal, y la parte horizontal superior queda más larga.*

De forma similar, llevamos a cabo el recorte del extremo inferior izquierdo. Este corte es aún más pequeño y fácil de realizar. Podemos hacer tanto los cortes horizontal como diagonal por separado. Con esto, obtenemos la forma que tendrá el brazo para el sistema de sonar robótico.

III EL BRAZO TERMINADO

Si tenemos dudas de la forma del trazo que debemos hacer, podemos observar las figuras del proyecto terminado. Esto nos dará una idea clara de cómo hacer el delineado del brazo, usando las guías que actualmente tenemos. La forma no es complicada, y resulta fácil identificar los lugares en donde realizar el dibujo.

***Figura 5**. El componente **A** ahora presenta la forma del brazo robótico.*

Para que el brazo sea resistente, usaremos dos componentes **A** unidos. Para esto, es necesario repetir el proceso de trazado y de corte en el otro componente **A**. Los dos deben quedar iguales como vemos en la siguiente figura.

***Figura 6.** Tenemos dos componentes A recortados de igual manera. Ambos serán necesarios para dar una estructura firme al brazo.*

Debemos colocar pegamento en uno de los componentes **A**. El adhesivo se debe esparcir de manera uniforme por toda la parte superior de este componente. Luego, y antes de que el pegamento se termine de secar, colocamos el otro componente **A** sobre él. Por último, debemos procurar que estos elementos se empalmen de manera correcta y presionamos para que, con el adhesivo, forme una unión firme.

***Figura 7.** Los dos componentes **A** forman una estructura más firme. No nos olvidemos de pegarlos en forma correcta.*

El brazo estará unido a un servo, por lo que necesitamos, desde este momento, indicar el lugar en donde se insertarán los tornillos. Lo primero consiste en localizar el lugar donde se encontrará el eje del servo. Para esto, necesitamos llevar a

cabo un trazado más. En la parte más larga del brazo, marcamos una línea horizontal justo en la mitad del componente.

Figura 8. *El trazado de esta línea horizontal nos ayudará a encontrar el lugar por donde pasa el eje del servo.*

Cerca del extremo derecho, tenemos un punto en donde se cruzan la línea horizontal que acabamos de trazar y una línea vertical trazada anteriormente. En este punto se encontrará el eje del servo. Ubicamos el dispositivo de actuación centrado precisamente sobre este punto y, como hicimos antes, marcamos los lugares donde se encontrarán los tornillos.

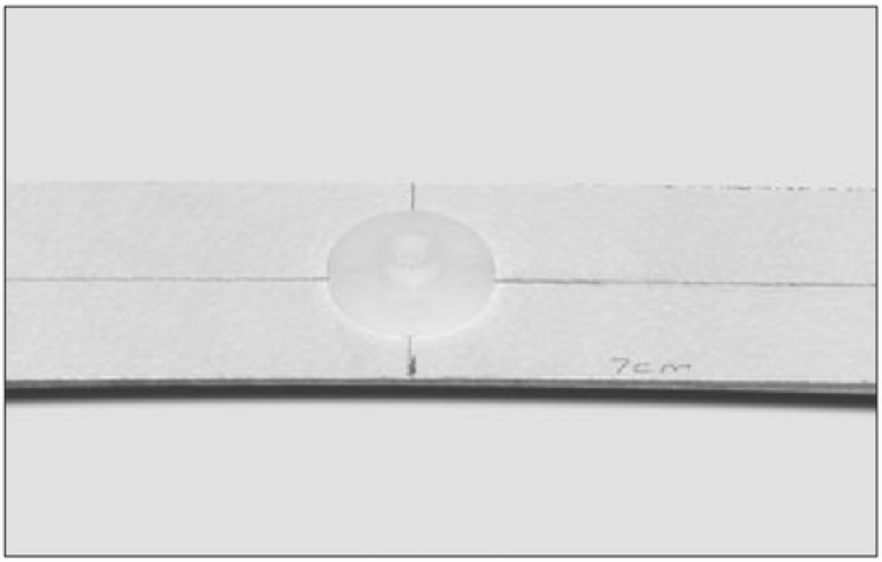


Figura 9. *Debemos ubicar el dispositivo de actuación en el punto exacto por donde pasa el eje del servo.*

{} EL SONAR EN EL MUNDO ANIMAL

El sonar también es utilizado en la naturaleza por los animales para poder reconocer objetos sin necesidad de verlos. Este proceso se conoce como **ecolocalización**. Los cetáceos, como las ballenas y los delfines, lo utilizan, pero también está presente los mamíferos voladores, como los murciélagos y los vampiros.

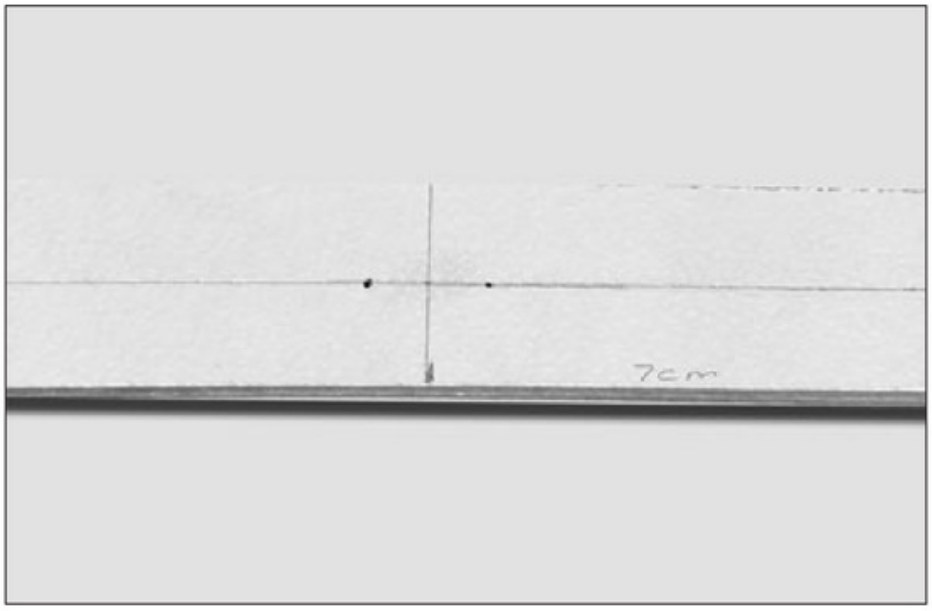


Figura 10. *Podemos observar los puntos por donde pasarán los tornillos para unir el brazo al servo.*

Para ayudar al movimiento del brazo, es necesario construir un contrapeso. El contrapeso ayuda al servo, facilitándole el movimiento, al reducir la cantidad de torca que necesita para mover el brazo. El contrapeso que proponemos se construye con cartón, pero podemos usar algún otro material. Empezamos con 14 componentes de tipo **B**. El tamaño ha sido seleccionado para que no estorbe el movimiento del brazo. La siguiente figura los muestra.

Figura 11. *Necesitamos 14 componentes **B** para construir el contrapeso que usaremos en el brazo robótico.*

III EL CONTRAPESO

Si elegimos hacer un contrapeso de otro material, el peso deberá ser similar al de un servo. Es importante tener cuidado con el tamaño para que no estorbe el movimiento del brazo. Se recomienda que el tamaño sea igual o menor al del diseño original para así evitarnos problemas.

Ahora que tenemos los componentes **B**, empezamos a pegarlos entre sí, procurando mantenerlos apilados. Deben quedar bien pegados y firmes, de manera que obtengamos una especie de ladrillo pequeño.

Figura 12. *Éste es el contrapeso una vez construido y que, después, adicionaremos al brazo robótico.*

El contrapeso es colocado en el extremo derecho del brazo. Para sujetarlo, simplemente, lo pegamos sobre el brazo del robot.

Figura 13. *El contrapeso es colocado sobre el brazo. Podemos observar que no tapa el lugar por donde pasa el eje del servo.*

Construcción del sostén para el servo del sonar

El hardware del sonar estará montado en un servo: esto nos permitirá rotarlo y, así, lograr el escaneo de las distancias alrededor del sonar en diferentes posiciones. El servo debe ser montado en el brazo del robot. Para poder montar el servo, crearemos un soporte. Este soporte debe permitir al servo permanecer firme

mientras se escanea y, también, mantener al servo en su lugar cuando el brazo rota entre sus diferentes posiciones. El servo se encontrará ubicado en el extremo izquierdo del brazo. Necesitamos los componentes **C**, **D**, **E** y **F**, presentes en la **figura 14**, para construir el sostén del servo.

Figura 14. *Éstos son los componentes necesarios para armar el sostén.*

El servo debe ser colocado en la posición correcta para facilitarnos, otra vez, el armado. En primer lugar, realizaremos una serie de trazos. Tomando el extremo izquierdo del brazo robótico, trazamos una línea horizontal a trece milímetros de distancia del borde inferior. Con esta línea trazada, entonces, dibujamos una línea vertical a diez milímetros de distancia desde el borde izquierdo.

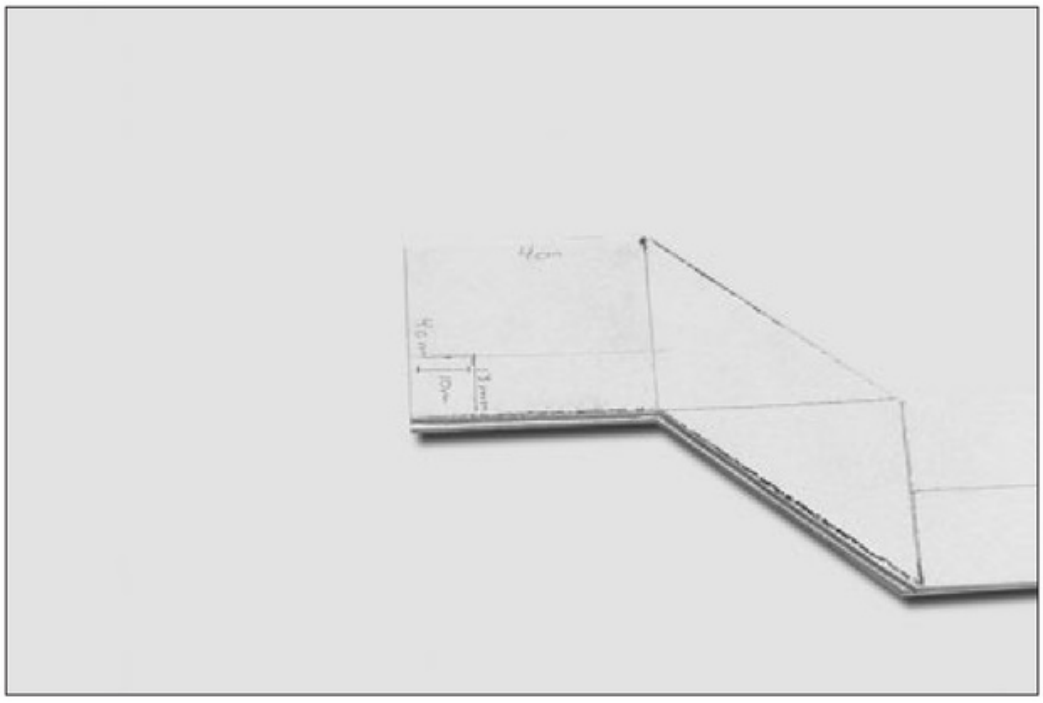


Figura 15. *Los trazos se hacen en el extremo izquierdo del brazo y nos servirán de guía para el montaje de los componentes.*

Ahora, tomamos el componente **C** y lo pegamos sobre el brazo. El extremo inferior izquierdo del componente **C** debe quedar sobre el punto donde se cruzarán las líneas que acabamos de trazar. Este componente servirá como asiento para el servo.

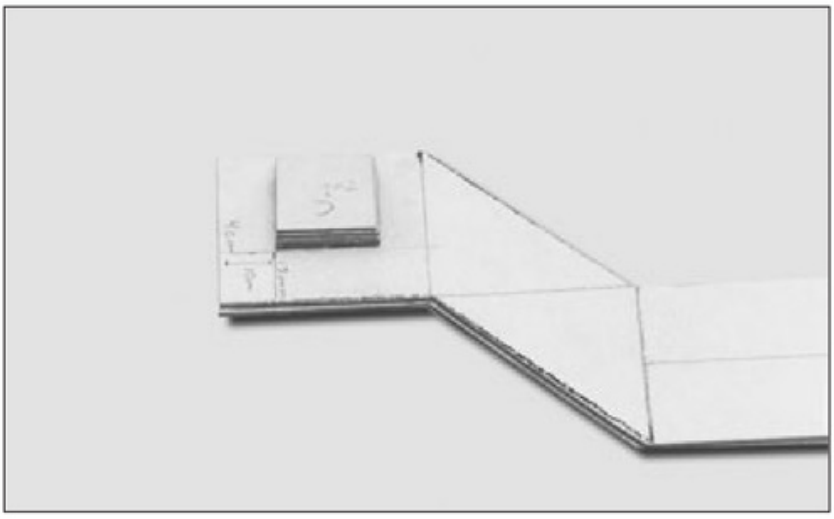


Figura 16. *Es importante notar la posición correcta del componente* ***C*** *sobre el brazo robótico.*

Cuando el pegamento del componente **C** se encuentre completamente seco y no se mueva de su lugar, entonces sentamos el servo sobre él. Como en los proyectos anteriores, usaremos el servo mismo para ayudarnos a crear una caja que lo contenga.

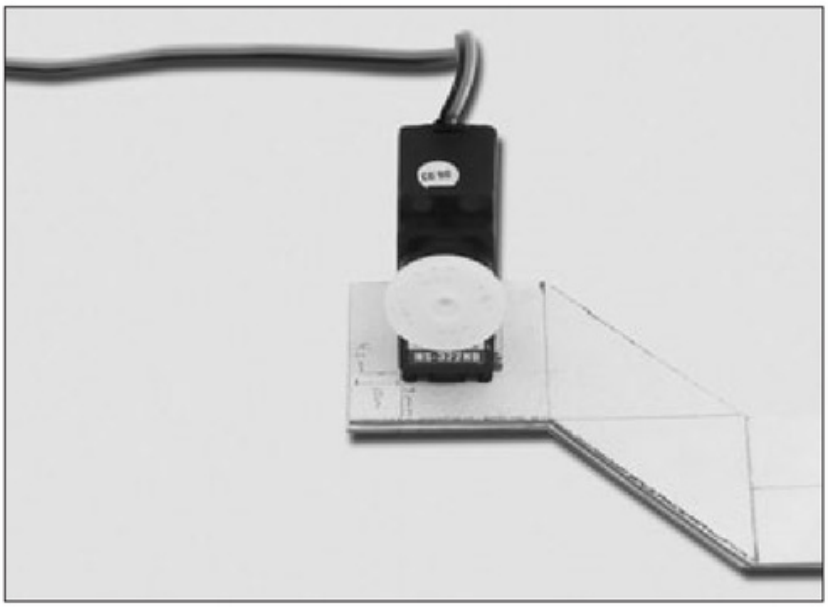


Figura 17. *El servo se sienta sobre el componente* ***C****. Debemos notar la forma correcta de colocarlo, con los cables hacia arriba.*

III NO PEGAR LOS SERVOS

Los servos son usados sólo para ayudarnos en el fijado de los componentes, pero en ningún momento, debemos pegarlos. Los servos deben poder quitarse y volver a colocarse de manera libre. Si necesitamos instalar un servo de forma permanente, es mejor recurrir a los tornillos en lugar del adhesivo que lo fijará de manera definitiva.

Con el asiento ubicado en su lugar es momento de que construyamos las paredes del sostén para el servo. Empezaremos utilizando los componentes **E**. Primero, pegamos un primer componente al brazo del robot. Este primer componente lo instalaremos del lado derecho del servo.

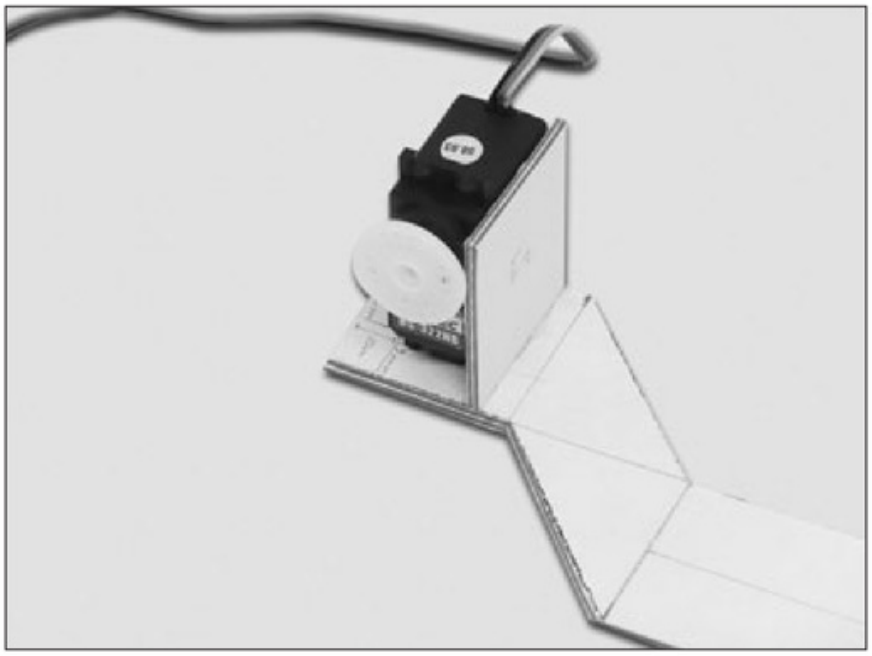


Figura 18. *Empezamos a construir las paredes del sostén colocando el primer componente* ***E***

De forma similar, instalamos en el lado izquierdo del servo el otro componente **E**, al que pegamos en el brazo robótico. Debemos fijar los componentes lo más cerca posible al servo, de tal manera que, cuando el sostén se encuentre completo, el servo quede ajustado y se mantenga bien firme.

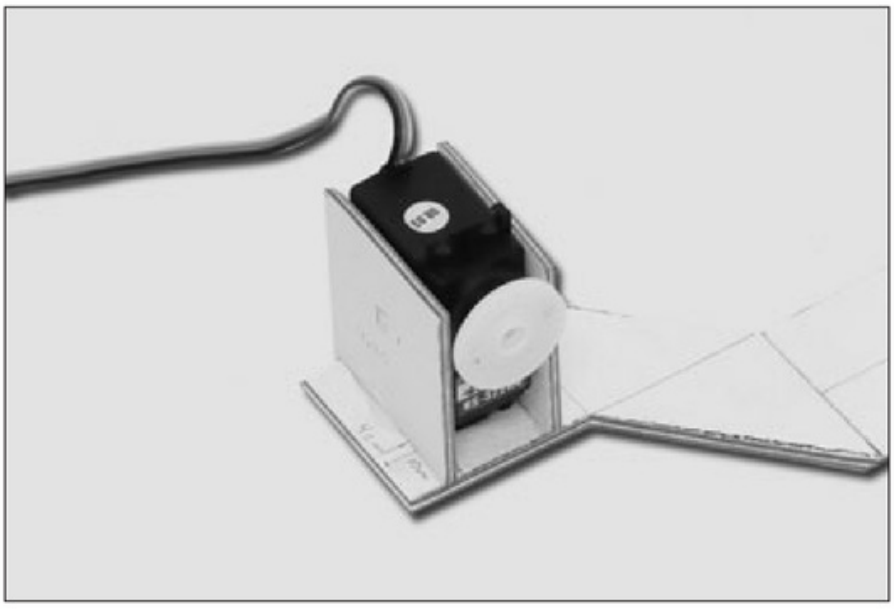


Figura 19. *El otro lado del sostén es construido por el otro componente* ***E****, el cual se pega de manera firme al brazo.*

Para brindar un soporte extra al servo y que el sostén tenga una estructura más firme, instalaremos el componente **D**. Este componente se ubica al frente del sostén y se pega tanto al brazo como a los dos componentes **E**. Esto también ayuda a que los componentes **E** permanezcan ajustados.

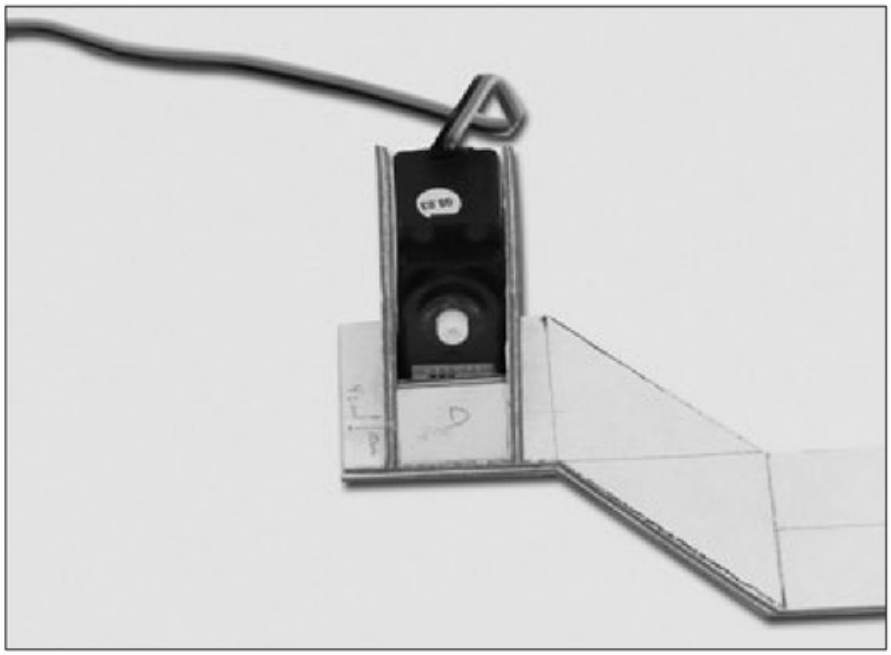


Figura 20. *El componente **D** se ubica al frente y en la parte inferior del sostén para el servo.*

Un proceso similar emplearemos en la parte trasera del sostén. En este caso, recurriremos al componente **F**. El componente tiene una altura mayor y ocupa toda el área trasera, por no haber movimiento en esta sección. De esta manera, tiene como ventaja que nos ayuda a mantener el sostén ajustado.

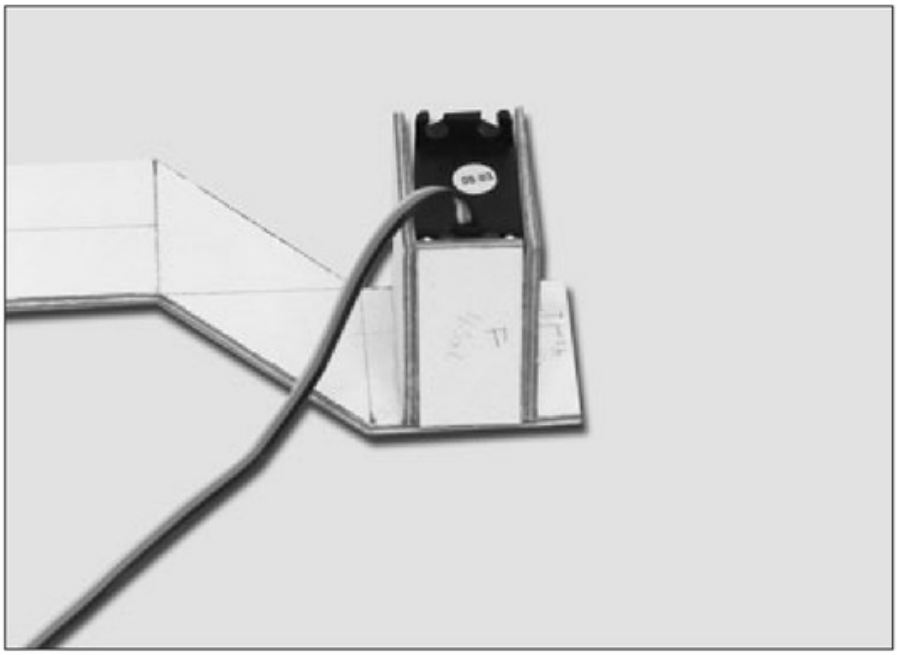


Figura 21. *El componente **F** se usa para la parte posterior del sostén.*

El sostén está finalizado. Es posible quitar y poner el servo, que debe entrar de forma ajustada en el sostén. Si llegara a ser necesario, podemos usar una banda elástica para ayudar a mantener el servo en su lugar. En en la siguiente figura, podemos observar el sostén sin el servo.

Figura 22. *El sostén para el servo está terminado.*

Construcción de la base

El brazo donde se encuentra el sonar será movido por un servo. Este movimiento nos permitirá ubicar al sonar en dos posiciones distintas. Para instalar este servo, necesitaremos una base, que deberá mantener al servo bien firme. Al mismo tiempo, tendrá que ser lo suficientemente estable como para poder soportar el vaivén del brazo sin provocar que éste se caiga.

Para comenzar, necesitamos varios componentes. En este caso, usaremos los componentes **J**, **G**, **H** e **I**, que podemos observar en la siguiente figura.

III UNA BASE ESTABLE

La base se encuentra montada sobre un trozo de madera que deberá ser sólida y pesada para poder soportar el movimiento del brazo. En caso de no conseguir este tipo de madera, podemos reemplazarla por una caja metálica rellena de arena. Otra opción puede ser atornillar o sujetar la base a una mesa de trabajo.

Figura 23. *Estos componentes son necesarios para el ensamble de la base que sostendrá al brazo del robot.*

Otra vez, realizaremos un delineado para facilitarnos el proceso de ensamble. Tomamos dos de los componentes **G** y trazamos una línea horizontal en cada uno de ellos. Esta línea debe estar a 4 cm del borde superior del componente.

Figura 24. *Llevamos a cabo el trazado de dos líneas horizontales para ayudarnos con el ensamble de la base.*

III EL FUNCIONAMIENTO DEL SONAR

Entender cómo funciona el sonar resulta muy sencillo. Se lanza una onda acústica, por lo general de ultrasonido. Esta onda, al chocar contra un objeto, rebota. Si conocemos la velocidad de esa onda y el tiempo que le insumió desde que salió, rebotó y regresó de vuelta a su origen, podemos calcular la distancia al objeto.

Ahora, debemos ubicar el servo de forma lateral sobre uno de los componentes **G**. Pegamos el componente **H** sobre el componente **G**. La línea que trazamos nos ayuda a que **H** quede derecho sobre **G** y ubicado de manera correcta.

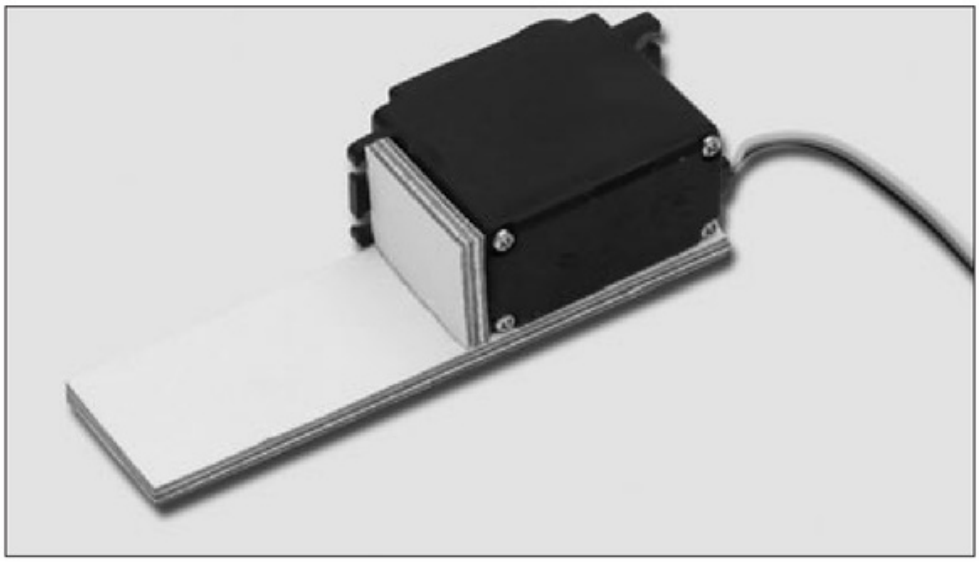


Figura 25. *El componente **H** se pega sobre el componente **G**. Más adelante, nos servirá como asiento para el servo.*

Es necesario que ubiquemos uno de los componentes **G** en el otro lado del servo. El componente se pegará exclusivamente al componente **H**. Para ello, podemos usar el trazado anterior para guiarnos. Si fuera necesario, utilizaremos un objeto pequeño para ayudar a mantener en posición el componente mientras pega.

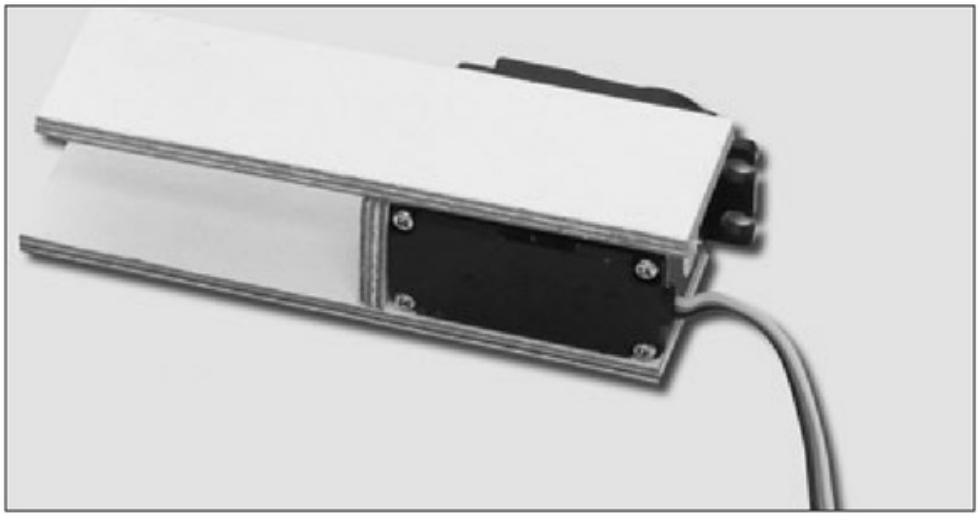


Figura 26. *El servo tendrá un componente **G** en cada lado. Esto ayudará a mantenerlo en posición.*

Aún tenemos un componente **G** que no hemos utilizado. Con este elemento, construiremos la parte trasera del soporte para el servo. Su instalación es muy sencilla, sólo hay que pegarlo a los otros dos componentes **G** y al componente **H**.

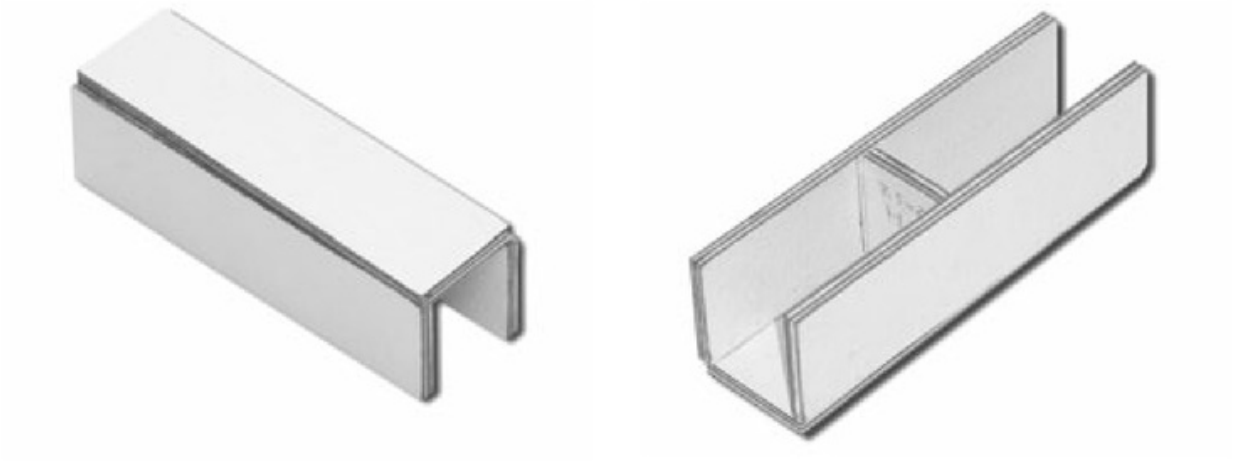


Figura 27. *El otro componente* ***G*** *es usado para crear la parte posterior del soporte para el servo. Aquí podemos ver la forma que adquiere en dos tomas distintas.*

El servo puede ser instalado en el soporte. Es preferible que lo dejemos un poco ajustado. En caso de que el servo se mueva, podemos recurrir a unas bandas elásticas para mantenerlo fijo en su lugar.

Figura 28. *El servo queda ajustado sobre el soporte. Los cables no aparecen obstruidos.*

El soporte se encuentra instalado en forma correcta, pero debemos otorgarle una mayor estabilidad. Otro problema que presenta es el área de contacto que tendrá con el componente **J**. Para solucionar estos inconvenientes, usaremos los compo-

nentes **I**. Estos elementos se ubicarán a los lados del soporte en su parte inferior. Debemos pegar uno de los componentes **I** sobre un componente lateral **G** en su parte inferior. Es necesario que quede bien alineado, dado que deseamos incrementar el área de contacto de la base del soporte.

Figura 29. *El componente **I** nos ayuda a incrementar el área de contacto del soporte.*

Una vez adherido el componente, hacemos lo mismo en el otro lado del soporte. Pegamos el otro componente **I**, también alineado a la base del soporte.

Figura 30. *Ambos lados del soporte tienen un componente **I** que ayuda a mantener la estabilidad del soporte.*

III RANGO DEL SONAR

El modelo de sonar que nosotros utilizaremos tiene un límite de distancia hasta donde puede realizar la detección. La distancia máxima a la que puede detectar un objeto es de 254 pulgadas, es decir, aproximadamente 6 metros con 45 centímetros. La distancia más pequeña que puede medir es de 6 pulgadas ó 15,24 centímetros.

Una vez que el soporte esté listo, debemos pegarlo al componente **J** para finalizar la base. El pegado debe ser centrado de forma horizontal, pero en el extremo inferior del componente **J**, con el fin de que el brazo no golpee la base al girar.

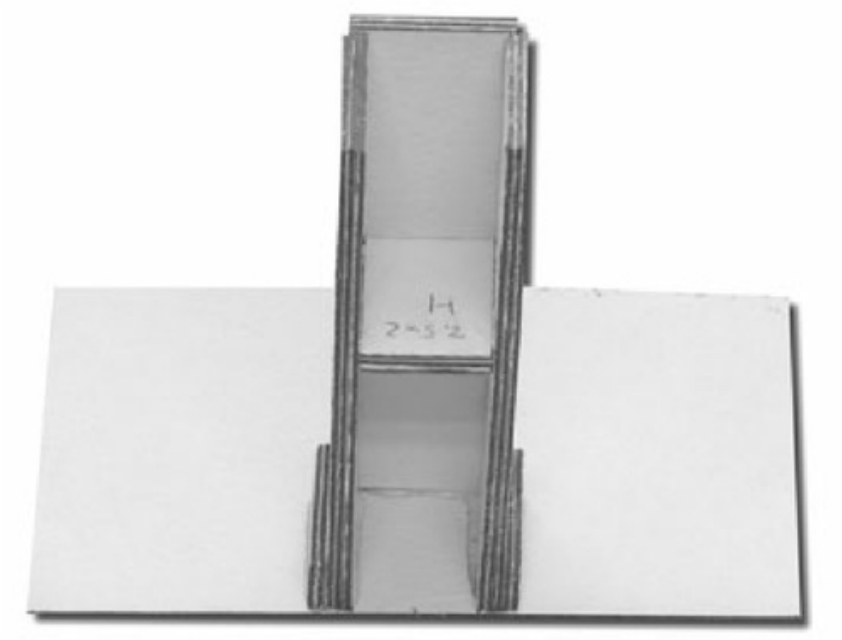


Figura 31. *El soporte es instalado sobre el componente J para crear la base. Es importante tomar nota de su posición.*

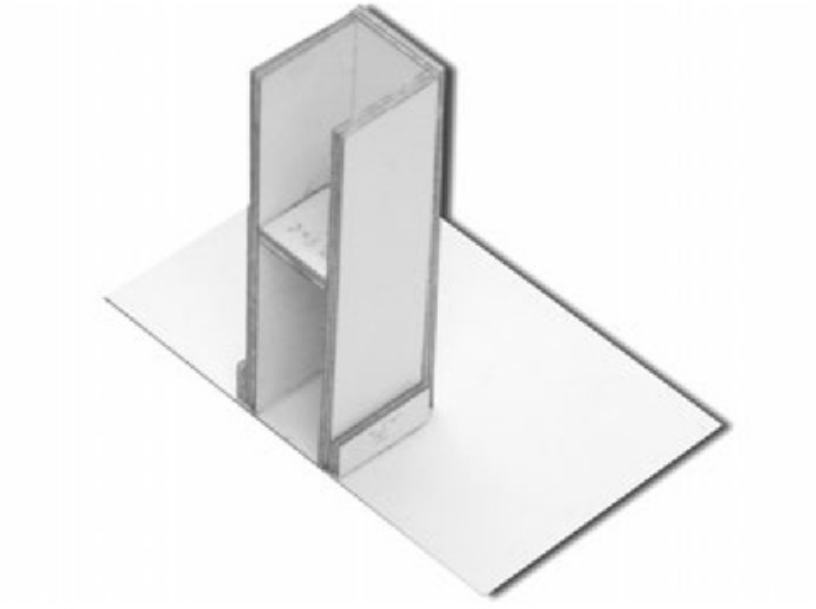


Figura 32. *En esta vista de la base, podemos apreciar el soporte y la forma en que los componentes I ayudan a la estabilidad.*

Creación del soporte del sonar

Deberemos instalar el sonar en el servo que se encuentra sobre el brazo. Esto nos permitirá rotar el servo en diferentes direcciones y hacer un recorrido del espacio

que se halla alrededor. Este soporte es muy similar al que realizamos en el proyecto del **capítulo 4** para sostener la webcam.

Necesitamos pocos componentes para crear este sostén. Usaremos los componentes **L**, **K** y **M**, que podemos observar en la siguiente figura.

Figura 33. *Éstos son los componentes necesarios para crear el soporte del sonar.*

El componente **L** sostendrá al sonar, y el componente **K** se atornillará al servo. Para poder unirlos, primero, tenemos que pegar el componente **M** sobre el componente **L**. El componente quedará pegado de forma centrada y alineada con relación a uno de los extremos del componente **L**.

Figura 34. *El componente* ***M*** *ha quedado pegado sobre el componente* ***L****.*

El otro componente **M** se pega en la misma posición, pero del otro lado del componente **L**. Esto nos permitirá tener una buena superficie de contacto. Es importante que los tres elementos se encuentren alineados de manera correcta.

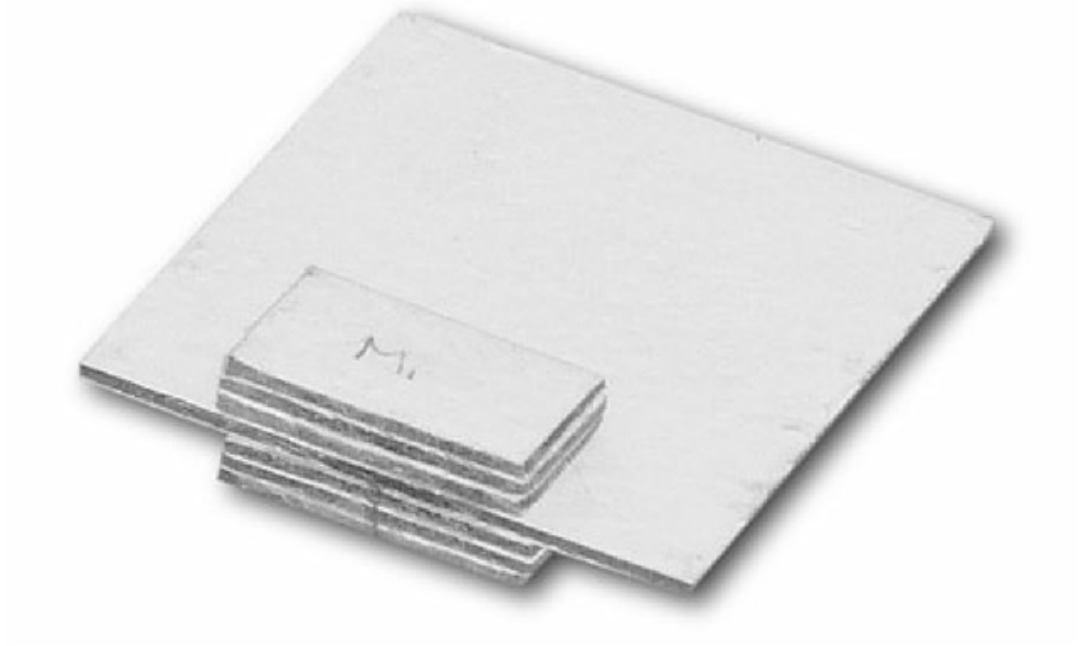


Figura 35. *Con los dos componentes **M**, hemos creado una buena base para el componente **L**.*

La estructura que hemos creado debe ser pegada sobre el componente **K**. La pegamos de forma diagonal sobre **K** y debe quedar completamente perpendicular sobre él como vemos en la figura 36.

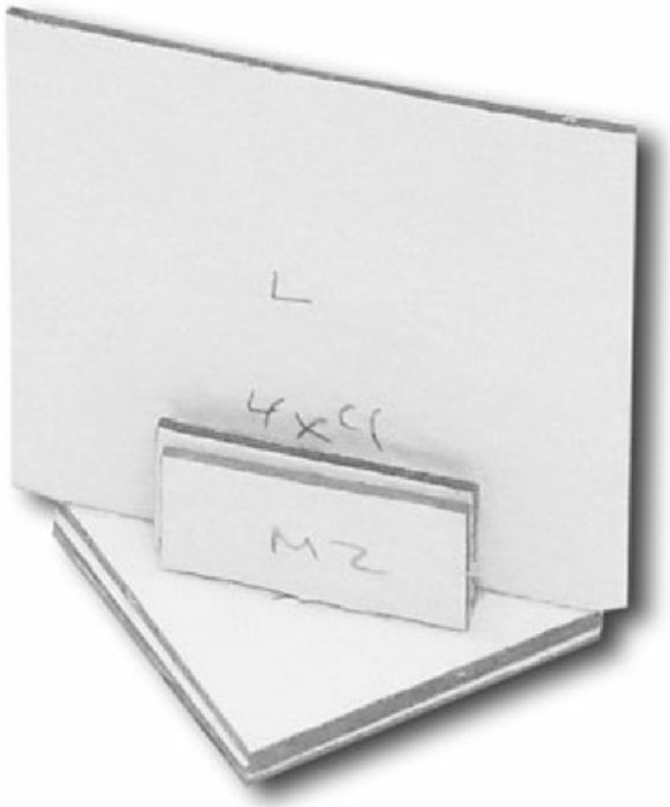


Figura 36. *El soporte para el sonar se encuentra finalizado; todos los componentes han sido pegados con firmeza.*

Ensamble del sonar

El sonar que utilizaremos es el modelo **LV-MaxSonar-EZ1**, que resulta muy sencillo de usar. El sonar brindará una salida con un valor analógico a la distancia que está midiendo. Usaremos este valor como entrada para el Phidget 8/8/8 y, de esta manera, nuestro software podrá leer el valor de la distancia. El hardware es muy pequeño y viene preensamblado; nosotros sólo tendremos que hacer algunas soldaduras para poder conectarlo al Phidget.

***Figura 37.** Éste es el sonar que utilizaremos en nuestro proyecto. Podemos ver que su tamaño es bastante pequeño.*

Si observamos la parte posterior del sonar, encontraremos varias perforaciones donde podemos llevar a cabo la soldadura. Estos agujeros tienen unas letras en su lado izquierdo, que nos indican cuál es la señal que les corresponde. A nosotros, nos interesan aquellas marcadas como **GND**, **+5** y **AN**. En la perforación GND, soldaremos el cable que nos conecta a tierra. De esta forma, tanto el Phidget como el sonar tendrán la misma referencia para sus valores de voltaje. En el marcado como +5, se coloca la alimentación de **+5V**, que provee el Phidget por medio de su conexión analógica, por lo que no debemos preocuparnos mucho por ella. Todavía tenemos

CUIDADO CON LA SOLDADURA

El cautín utilizado para soldar y la misma soldadura pueden alcanzar temperaturas altas que generan riesgos de quemaduras o de incendio. Por esto es recomendable que la soldadura la lleve a cabo un adulto que sepa cómo realizarla. Se sugiere, también, usar un soporte especial para cautín para apoyarlo mientras no se encuentra en uso.

una conexión más, la que se encuentra marcada como AN. Esta conexión es importante, pues es la señal analógica que nos indica la distancia a la que se encuentra el sensor. Podemos observar estas perforaciones en la siguiente figura.

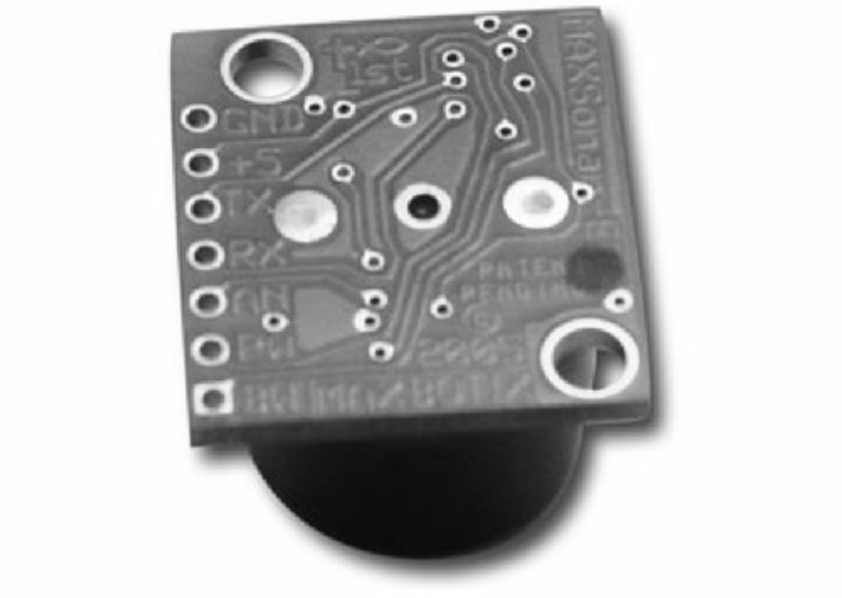


Figura 38. *Ésta es la parte inferior del circuito del sensor donde soldaremos el cable que se dirige hacia el Phidget.*

Con un conector para los puertos análogos del Phidget, soldamos los cables donde corresponde en el sonar. Podemos usar el cable con un conector sobrante del proyecto anterior. Empezamos por colocar la tierra, que es el cable de color negro. Este cable debe soldarse en el orificio marcado como GND en la placa de circuito del sonar.

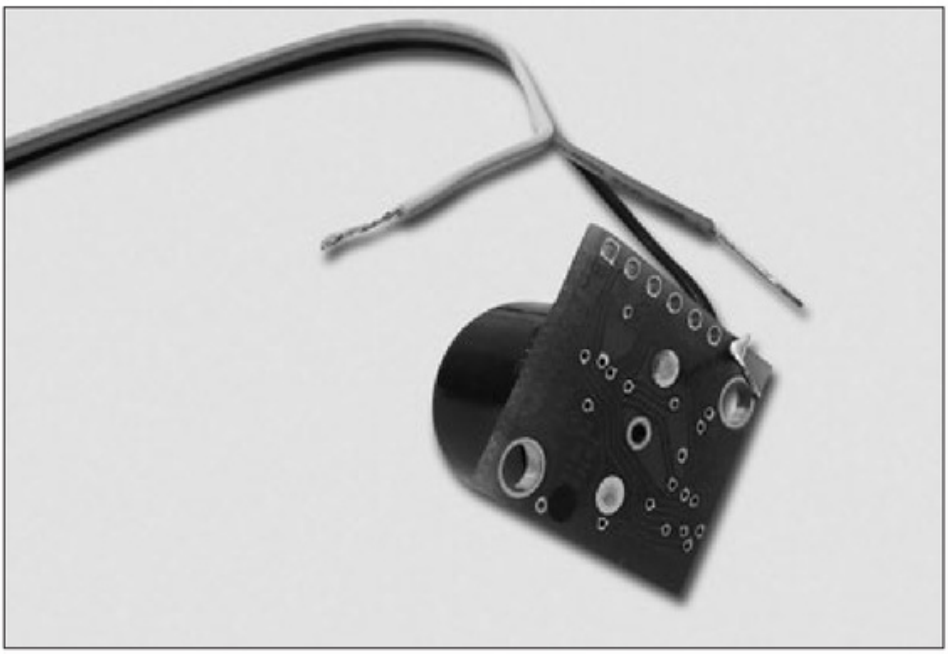


Figura 39. *El primer cable que soldamos es el de tierra, que tiene color negro y va en el orificio GND.*

El cable de color rojo es el que lleva el voltaje, y debemos soldarlo a la placa del circuito del sonar. Este cable lo uniremos en el orificio que está marcado como +5.

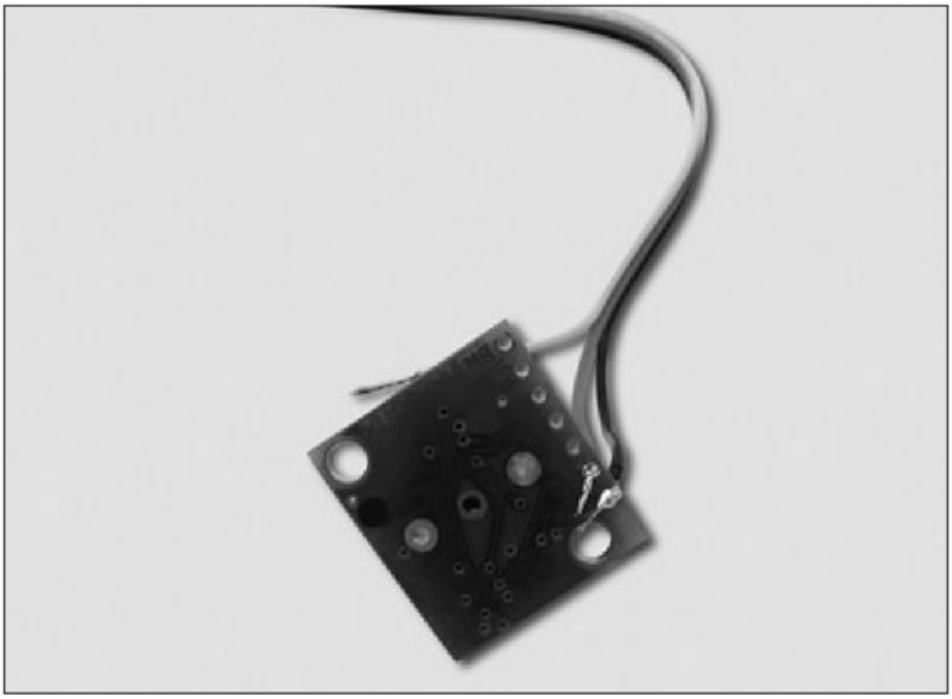


Figura 40. *El cable rojo se debe soldar en el orificio marcado como +5, que se encuentra al lado de la soldadura anterior.*

La señal analógica viajará por medio del cable blanco, que deberá ser soldado al orificio que se encuentra marcado como AN. Una vez que los tres cables estén soldados, podemos recortar cualquier exceso de ellos en el lado de la soldadura. Esto nos brindará un terminado más profesional y reducirá posibles errores eléctricos al evitar que se toquen los cables.

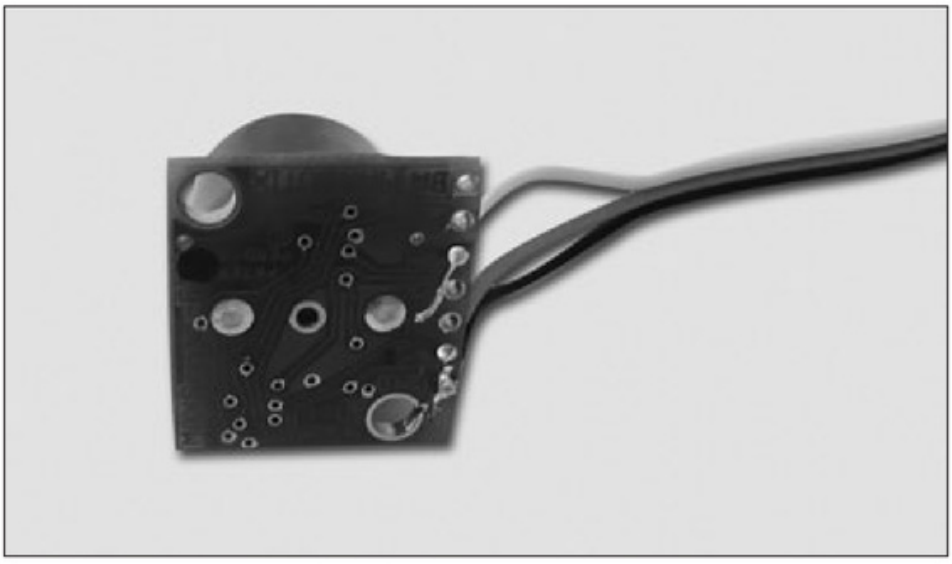


Figura 41. *Los tres cables fueron soldados al sonar. Vemos que sobresale un poco por el lado de la soldadura. Podemos cortar el exceso para evitar problemas.*

En el próximo paso, para ayudarnos en el montaje del sensor, utilizamos dos recortes pequeños de espuma plástica o de algún otro material similar. Debemos pegar ambos trozos en la parte trasera del sonar.

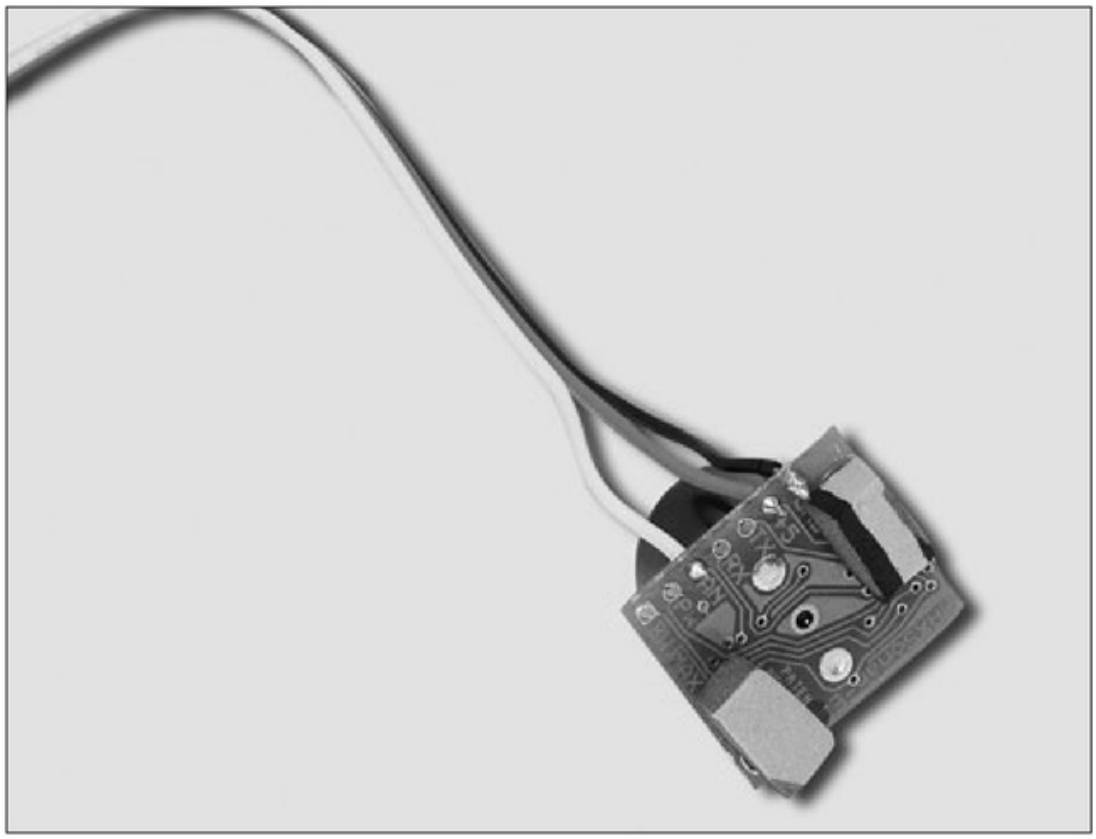


***Figura 42.** Estos dos pequeños pedazos de espuma plástica pegados en la parte posterior del sonar nos ayudarán en su montaje.*

Montaje final del brazo y el sonar

Finalmente, llegamos al momento de ensamblar las diferentes piezas que creamos durante el capítulo. Si lo deseamos, podemos pintarlas para que nuestro robot luzca mejor. La ejecución del ensamble final no es complicada, pero requiere de un poco de trabajo. El brazo quedará unido a un servo. Antes de unirlo, es necesario que el servo se encuentre en su posición más pequeña. Esto es preciso para que el brazo

III LA RESOLUCIÓN DEL SONAR

Conocer cuál es la resolución del sonar es importante: este valor nos permite saber la distancia mínima que el sonar puede detectar y considerar como dos valores diferentes. En el modelo de sonar de este proyecto, la resolución es de una pulgada. Esto significa que sólo puede hacer mediciones de pulgada en pulgada.

pueda girar 180 grados y, así, sea capaz de realizar las lecturas del sonar desde dos lugares diferentes. Si el servo se encontrara en su posición intermedia, no podría girar lo suficiente. Tomamos el brazo y ubicamos el elemento de actuación en la parte donde estará el eje del servo. Podemos encontrar fácilmente este sector, pues nuestros trazados anteriores han formado una intersección ahí.

Figura 43. *Ubicamos el elemento de actuación para luego poder marcar el lugar de los tornillos.*

Como lo realizamos en proyectos anteriores, marcamos los lugares donde se ubicarán los tornillos. Es preferible procurar que los puntos que marcamos se ubiquen sobre el trazado horizontal del brazo.

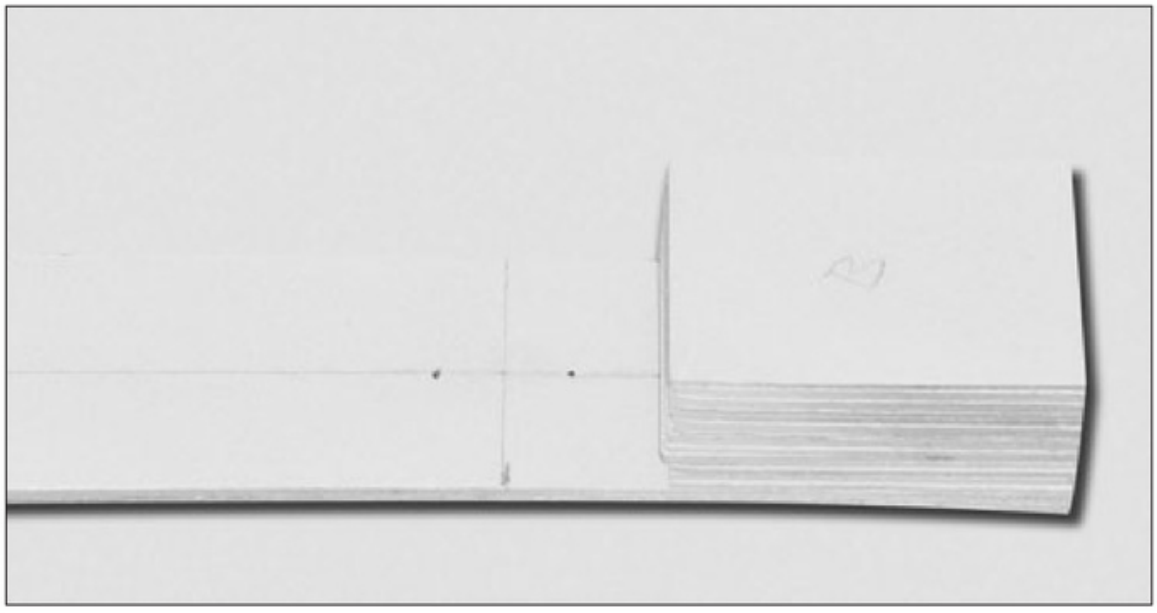


Figura 44. *Estos dos puntos nos indican dónde se insertarán los tornillos que nos ayudarán a fijar el brazo al servo.*

Aprovechando los puntos marcados, efectuamos dos pequeñas perforaciones. El diámetro de estas perforaciones debe ser menor que el diámetro de los tornillos. Estos orificios facilitan la inserción de los tornillos.

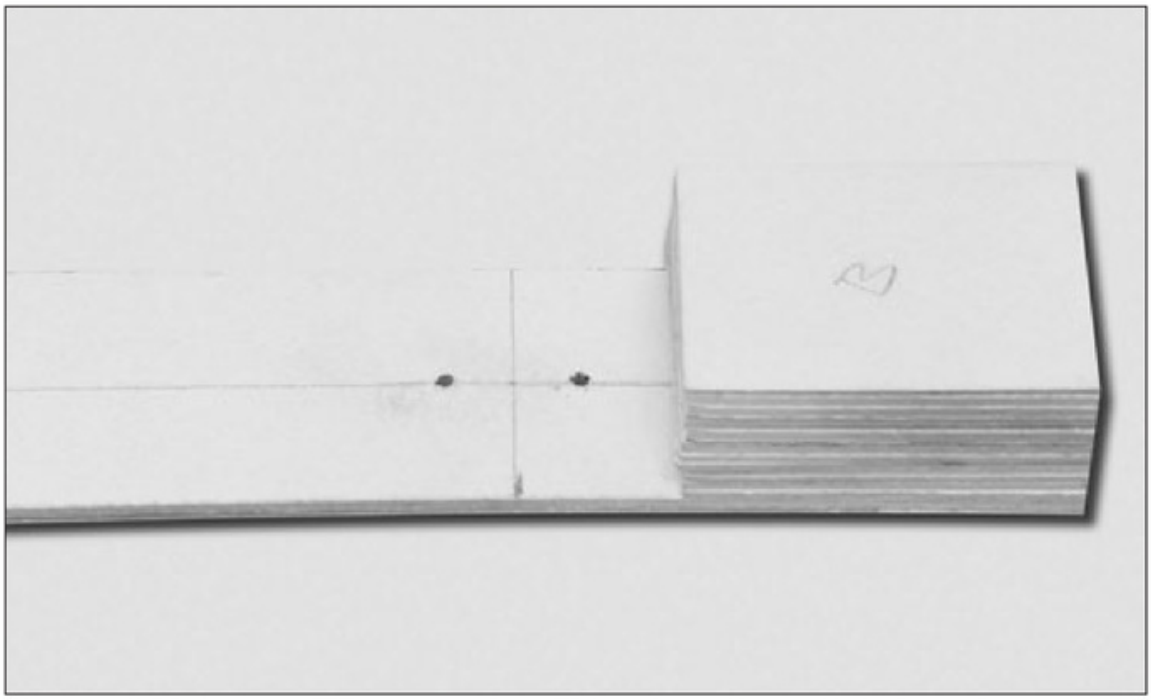


Figura 45. *Hacemos dos pequeños agujeros para facilitar la inserción de los tornillos.*

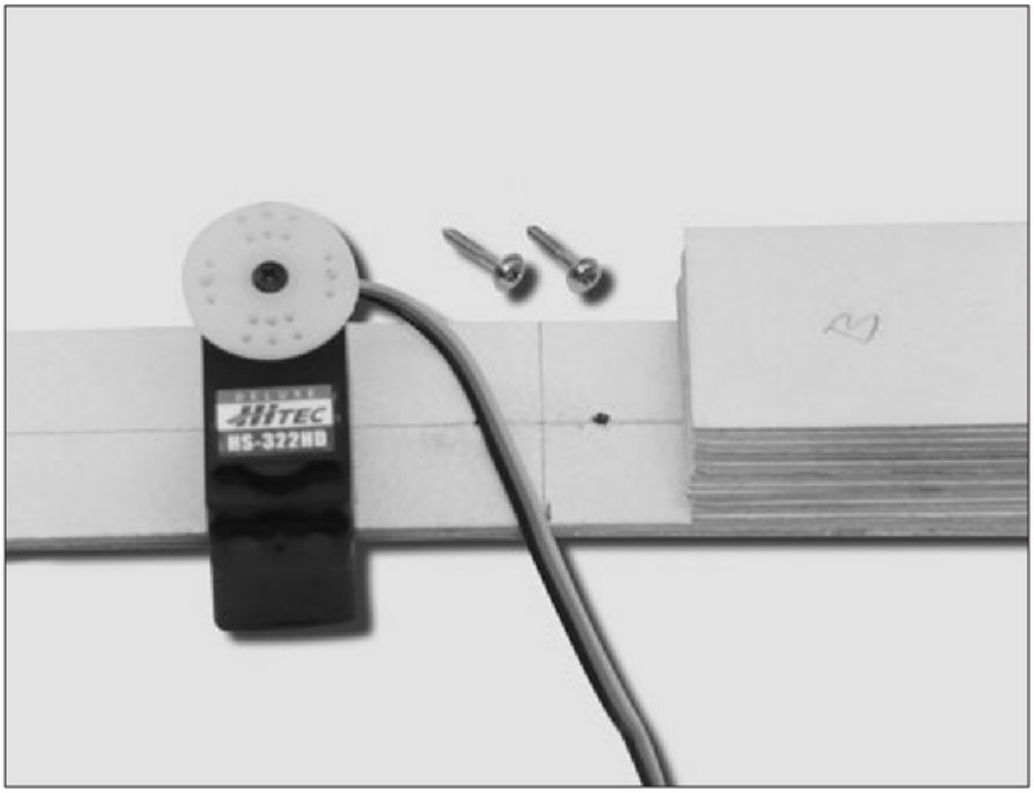


Figura 46. *Éstos son los elementos que necesitamos para fijar el brazo robótico al servo.*

Ahora, con cuidado, empezamos a atornillar el brazo al servo. Los tornillos deben entrar sólo a una profundidad en la que sostengan el brazo, pero que no interfieran con el libre movimiento del servo.

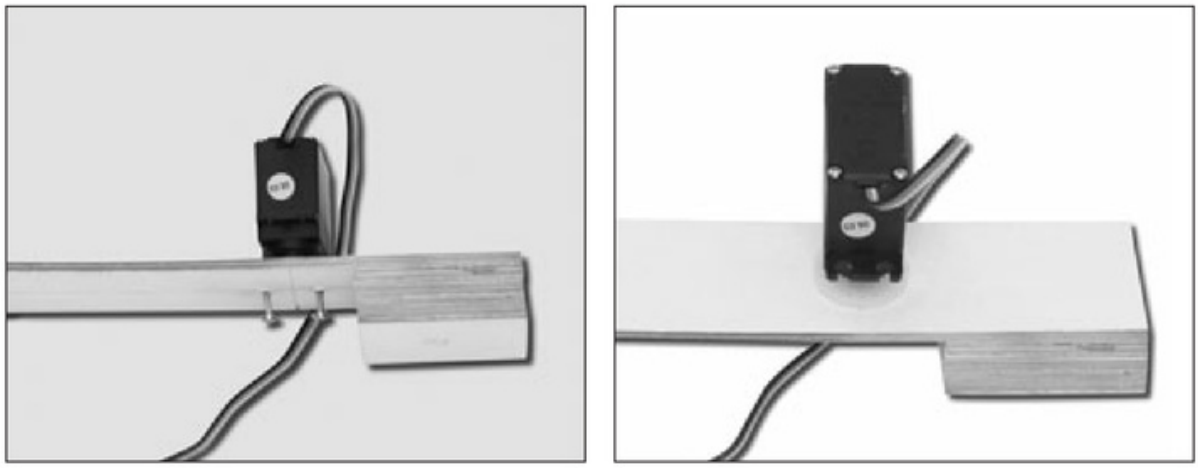


Figura 47. *En la primera imagen, podemos observar cómo el brazo ha quedado fijo al servo. Mientras que, en la segunda, vemos mejor el ensamble con el servo en la parte posterior del brazo.*

A continuación, armamos la base con el bloque de madera y los componentes que construimos antes. Lo importante es que la base tenga el peso suficiente para evitar que el brazo robótico se vuelque cuando se mueva. El bloque de madera puede pegarse a la base o puede ser sujeto por medio de bandas elásticas: debemos procurar evitar el movimiento.

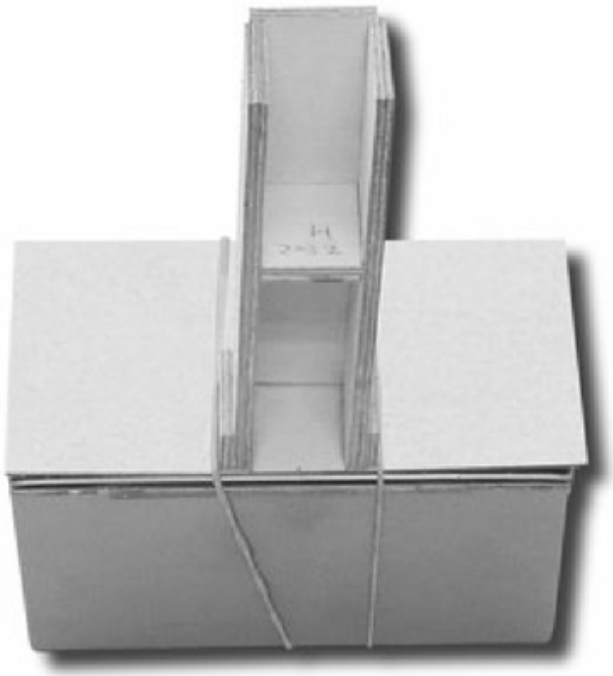


Figura 48. *Ésta es la base lista para colocar sobre ella el brazo robótico.*

Una vez que la base se encuentra firme, colocamos el brazo sobre ella. El servo que sostiene al brazo debe asentarse sobre la base, sin problemas, y entrar de forma ajustada. Si notamos que existe movimiento de más o que el servo no está lo bastante firme, entonces, podemos recurrir a bandas elásticas para fijarlo mejor.

Figura 49. *El brazo se encuentra montado sobre la base. Debemos procurar que se mantenga firme y seguro.*

Todavía nos falta trabajar un poco con el sostén del sonar. Lo primero que debemos hacer es marcar, en el sostén, los lugares en donde se ubicarán los tornillos que nos servirán para sujetarlo al servo.

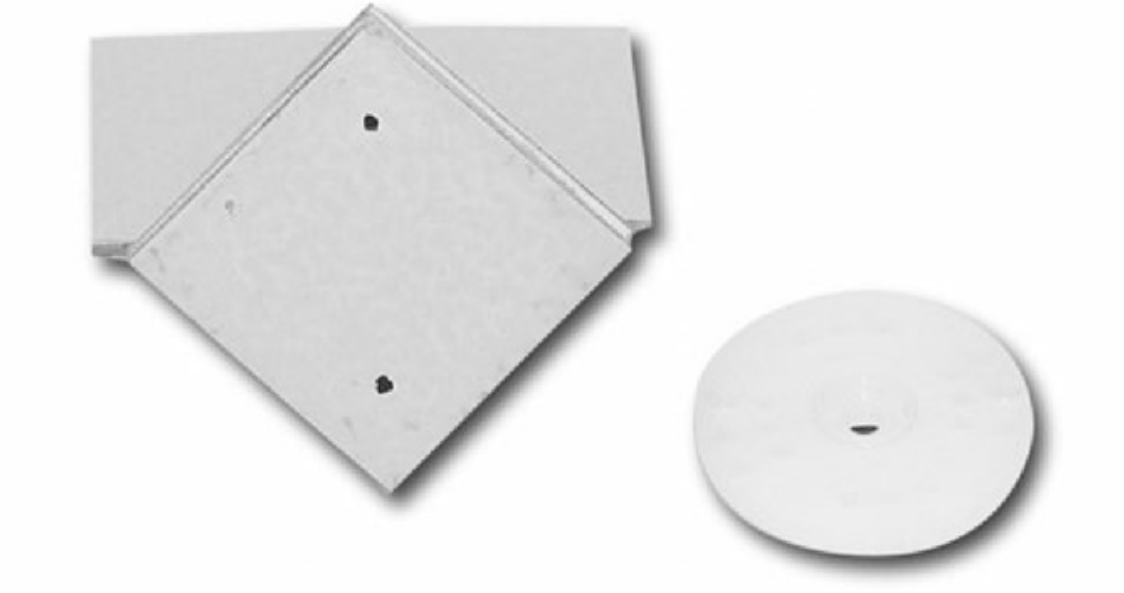


Figura 50. *Es importante notar la posición donde se han establecido las marcas de los tornillos.*

III MODIFICACIÓN DE LA SENSIBILIDAD

Es posible llevar a cabo un ajuste de sensibilidad en las entradas analógicas del 8/8/8. Esto nos puede ayudar a ajustar las lecturas del sonar para mejorar, de esta forma, su desempeño y precisión. En la documentación de Phidgets, podemos encontrar cómo hacerlo, al igual que en el panel de control dentro de la aplicación.

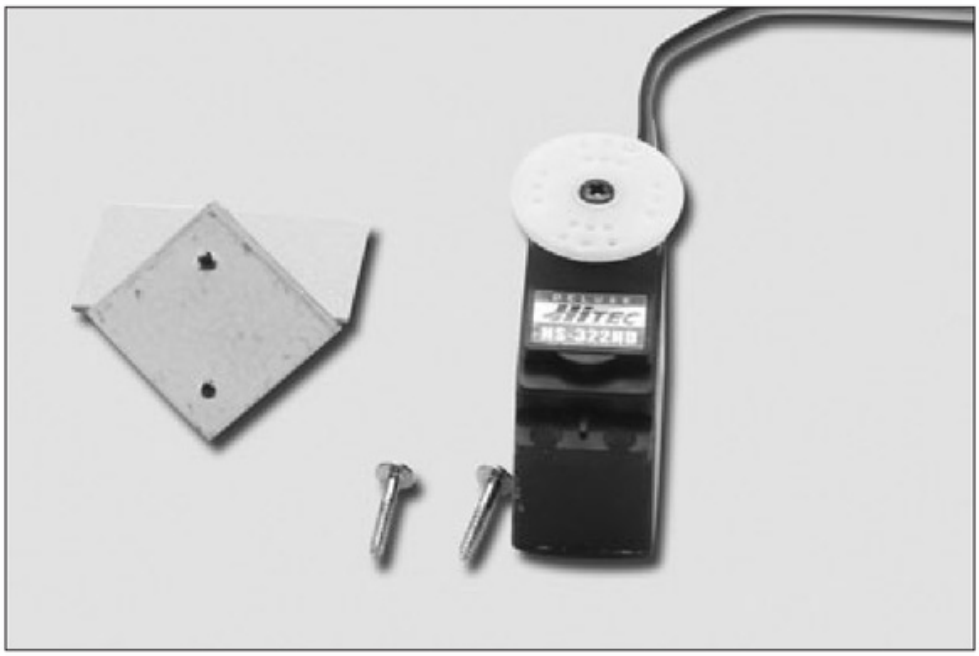


Figura 51. *Necesitamos del sostén, del servo y los tornillos para finalizar el montaje del sonar.*

El servo debe estar ubicado en su posición centrada. En este montaje, el sonar quedará viendo hacia el frente. Una vez que el servo se encuentre en su posición, procedemos a montar el sostén. Debemos recordar que los tornillos deben ser colocados sin una excesiva profundidad para no estorbar el movimiento del servo.

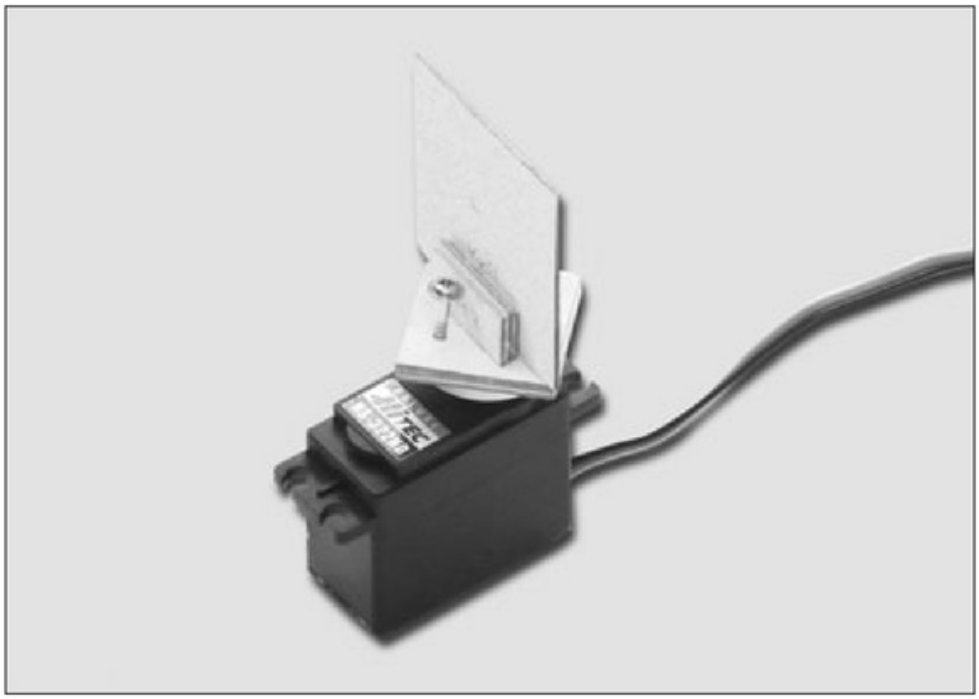


Figura 52. *El sostén queda montado sobre el servo que se encuentra en su posición central.*

Ahora, debemos colocar este servo en el brazo. El sostén queda viendo hacia el frente, mientras que el cable del servo sale por la parte frontal. Por último, lo único que nos falta es instalar el sonar.

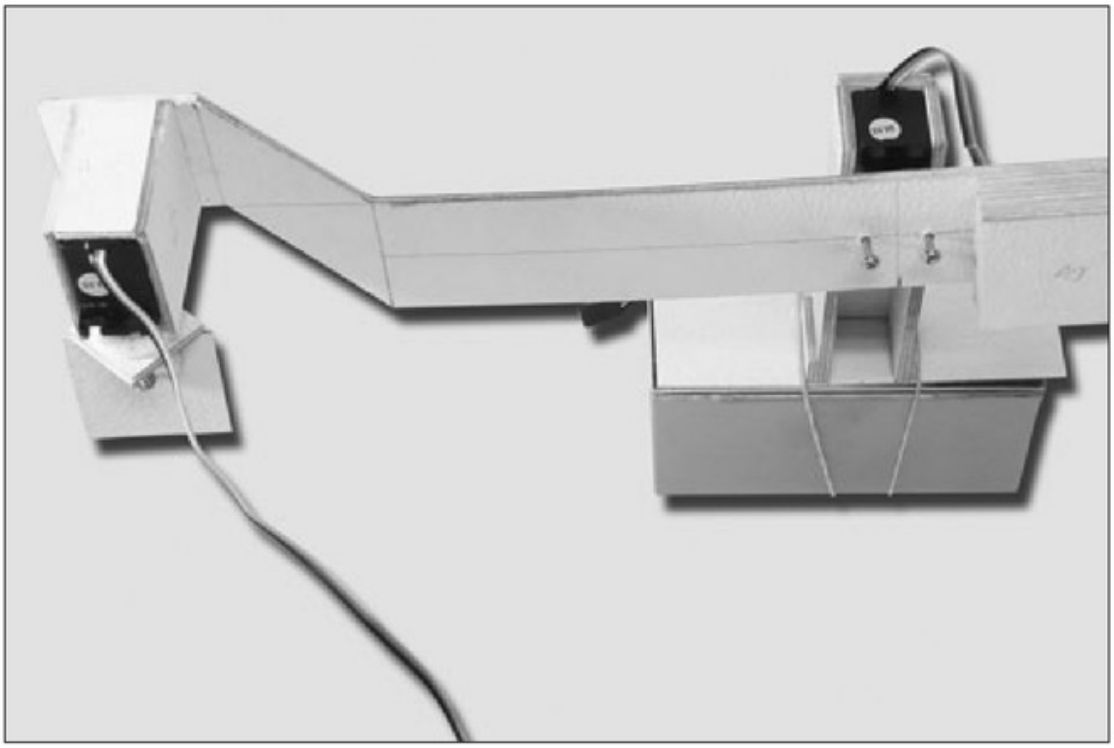


Figura 53. *El servo que controla al sonar ha sido colocado en el brazo robótico.*

Tomamos el circuito del sonar y colocamos un poco de pegamento en la parte posterior de la espuma plástica. Luego, ubicamos el sonar sobre su sostén. Esperamos a que el adhesivo se seque y, entonces, estaremos listos para utilizar el robot. El servo que mueve la base se coloca en el conector **0** del Phidget; el servo que se encarga de mover al sonar se conecta en el **1**. El cable que lleva la señal analógica del sonar se acopla en el puerto **0** del Phidget 0/0/0.

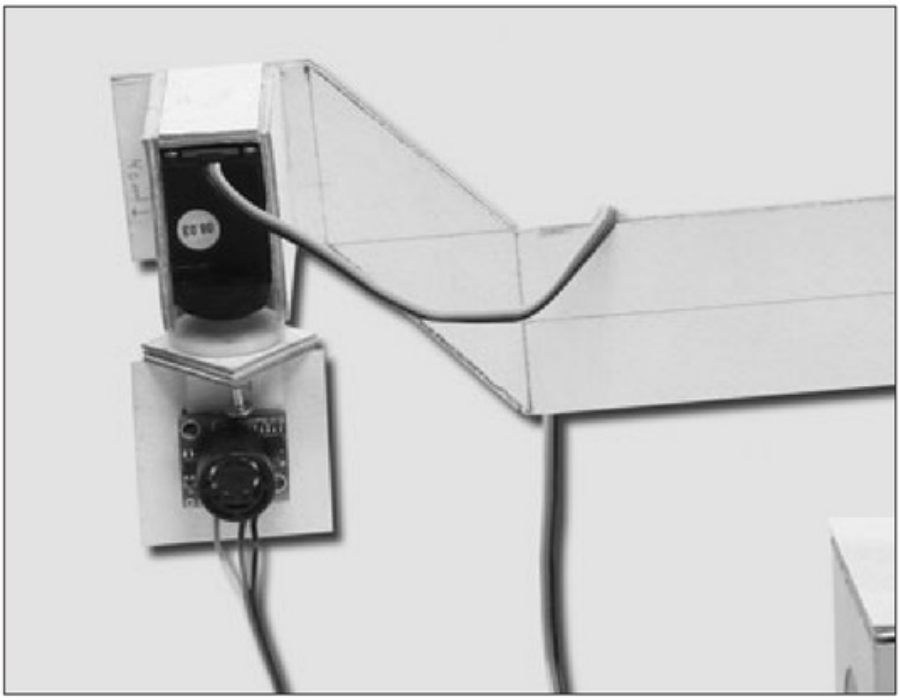


Figura 54. *El servo ha sido montado en su lugar, y el hardware del robot está finalizado.*

PROGRAMACIÓN DEL SOFTWARE PARA EL ROBOT

El programa para controlar el sonar robótico tendrá varias responsabilidades. En primer lugar, necesitará controlar el movimiento del brazo para ubicar el sonar en posición y, luego, deberá girarlo para escanear el espacio alrededor de él. También, será necesario que obtenga los valores que detecta el sonar y, al final, deberá dibujarlos. Como en los capítulos anteriores, usaremos el programa base que hemos creado y lo modificaremos para el proyecto del sonar.

Creación de la interfaz de usuario

La interfaz de usuario es muy sencilla: simplemente mostrará el valor leído por el sonar en ese momento. También, presentará un botón para iniciar el proceso de escaneo por parte del sonar y una sección de la forma donde se dibujará la información recopilada por él. El sonar lee información tanto del lado derecho como del lado izquierdo, por lo que se dibujará la información de ambos lados. Para hacer las gráficas, nos apoyaremos en el **GDI**. Las gráficas que hacemos son muy sencillas, pero pueden modificarse para mostrar mejores resultados. Empecemos a crear la interfaz de usuario; para esto, nos basamos en la siguiente figura.

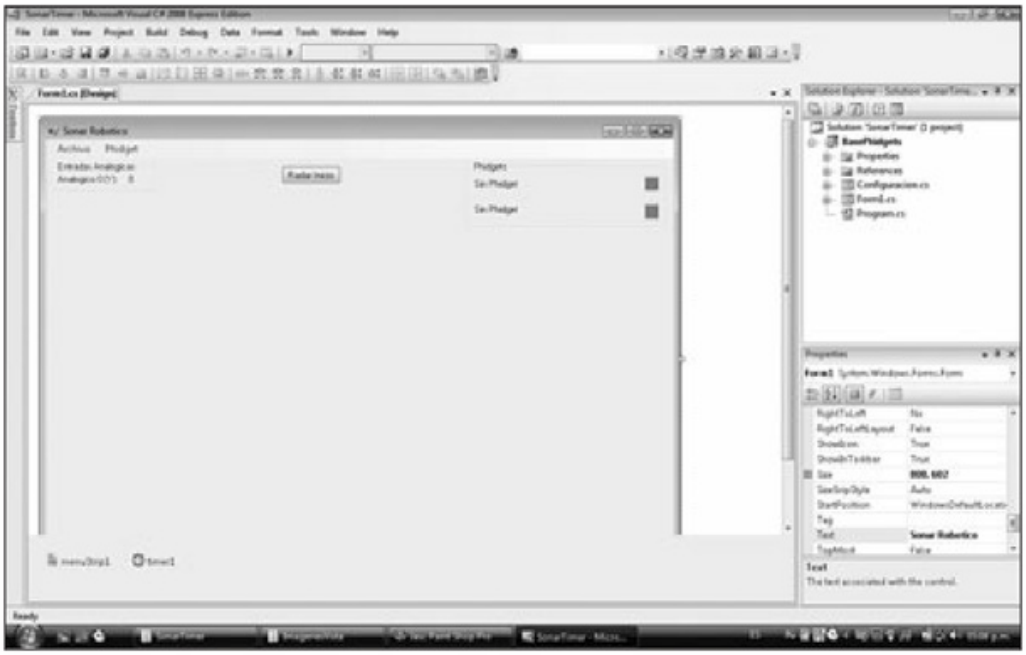


Figura 55. *Ésta es la interfaz de usuario para la aplicación del sonar robótico.*

Para empezar a crear la interfaz de usuario, le damos a la forma un tamaño de **800x600**. Este tamaño es necesario para que se dibuje la información del sonar. De la interfaz de la aplicación base, eliminamos todo menos la información de la entrada analógica **0** y lo relacionado a los Phidgets. Insertamos un botón, al cual llamaremos **btnRadarInicio** y, en su propiedad **Text**, ingresamos el texto **"Radar Inicio"**. Este botón será utilizado para que el robot empiece a escanear con el radar. También es

necesario que adicionemos un **Timer**. El **Timer** debe tener en su propiedad **Enabled** el valor **False**. Esto es importante, pues no queremos que el programa inicie con el **Timer** activado. En su propiedad de **Interval**, establecemos el valor **200**. El **Timer** nos apoyará para el proceso de escaneo.

El código del programa no es complicado. En este caso, también, sólo comentaremos los aspectos nuevos o con cambios importantes. Gran parte del código que veremos a continuación es de nuestro conocimiento luego de haberlo visto en detalle en los capítulos anteriores. En primer lugar, comenzamos por los namespace que estamos utilizando. No debemos olvidar que Visual Studio introduce por nosotros algunos namespace de forma automática. Es importante tener en cuenta que, para este proyecto, estamos agregando el namespace **System.Drawing.Drawing2D**.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

// Necesario para el registro
using Microsoft.Win32;

// Necesario para los Phidgets
using Phidgets;
using Phidgets.Events;

using System.Threading;
using System.Drawing.Drawing2D;
```

III EL SONAR SE AUTOCALIBRA

El modelo de sonar que usamos realiza una calibración de forma automática cuando se enciende. Para que se calibre de manera correcta, es importante que no tenga objetos cercanos enfrente de él. Como mínimo, debe tener 40 centímetros de espacio libre para hacerlo correctamente.

Utilizaremos varios datos para esta aplicación base, los cuales son declarados como datos de la clase de la forma.

```
        // Datos necesarios para la aplicacion base

        private int numeroSerie0;        // Numero de serie del phidget 0
        private int numeroSerie1;        // Numero de serie del phidget 1
        private bool conectado0;         // Para saber si hay conexion con
el phidget 0
        private bool conectado1;         // Para saber si hay conexion con
el phidget 1
        private InterfaceKit iKit;       // Kit de interfaz
        private AdvancedServo mServo;    // Controlador de servos

        private int mAngulo;              // Angulo del sensor
        private double mAvanceServo;     // Valor de avance del servo
        private double mAnguloServo;     // Angulo del servo
        private int[] mDatosDerecho = new int[180]; // Informacion del
sonar
        private int[] mDatosIzquierdo = new int[180];   // Informacion del
sonar
        private int estado;               // Estado actual de operación
```

La variable **mAngulo** es usada para saber cuál es el ángulo que tiene el sensor en ese momento. El ángulo aparece en grados y, también, nos servirá como índice para guardar la información de distancia para ese ángulo en particular. El servo debe avanzar una determinada cantidad por cada grado que movemos el sensor. Ese valor es calculado y se guarda en la variable **mAvanceServo**. La variable **mAnguloServo** actúa como un acumulador y nos indica la posición actual del servo en las unidades propias de posición del servo.

III MÁQUINAS DE ESTADOS FINITOS

Las máquinas de estados finitos son explicadas en el libro *Inteligencia Artificial*, de esta misma editorial. Son máquinas que solamente pueden estar en un número finito de diferentes estados. Los estados son preestablecidos en el momento en que se diseñan.

Tenemos dos arreglos: uno para la información de distancia del lado derecho y el otro para la información de distancia del lazo izquierdo. Cada uno de los arreglos puede guardar 180 valores, es decir, un valor para cada grado. Para facilitar la programación del robot, usaremos una máquina de estados finitos. El estado actual en el que se encuentra el robot es guardado en la variable **estado**.

El constructor de la clase no es diferente de los constructores que hemos visto en los proyectos anteriores. Simplemente se lleva a cabo el proceso de verificar los valores contenidos en las llaves del registro.

```
        public Form1()
         {
              InitializeComponent();

              // Inicializamos la aplicacion

              // Empezamos como si no estuvieramos conectados
              conectado0 = false;
              conectado1 = false;

              // Colocamos un valor de default para el numero de serie
              numeroSerie0 = 0;
              numeroSerie1 = 0;

              // Verificamos si existe la llave en el registro
              RegistryKey sfwKey =
Registry.LocalMachine.OpenSubKey("Software");
              RegistryKey redKey = sfwKey.OpenSubKey("RedUsers2");

              // Verificamos si existe
              if (redKey == null) // la llave no existe
              {

                  // Creamos la llave
                  sfwKey = Registry.LocalMachine.OpenSubKey("Software", true);
                  redKey = sfwKey.CreateSubKey("RedUsers2");

                  RegistryKey robotKey0 = redKey.CreateSubKey("Robots2");

                  // Colocamos un valor de default
```

```
                robotKey0.SetValue("NumeroSerie0", (object)numeroSerie0);
                robotKey0.SetValue("NumeroSerie1", (object)numeroSerie1);

            }
            else // la llave existe
            {
                RegistryKey robotKey0 = redKey.OpenSubKey("Robots2");

                // Obtenemos el numero de serie guardado
                numeroSerie0 = (int)robotKey0.GetValue("NumeroSerie0");
                numeroSerie1 = (int)robotKey0.GetValue("NumeroSerie1");
            }

        }
```

Otro código que permanece similar a los proyectos anteriores es el relacionado con el evento del menú **Salir** y el del menú **Configuración**.

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

        private void configurarToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            // Este codigo es para el phidget 0

            Configuracion dlgConfig = new Configuracion();
```

PARA CONOCER EL SONAR

Si necesitamos mayor información técnica sobre el sonar que estamos utilizando, la podemos buscar en el sitio web de su fabricante. En **www.maxbotix.com/**, encontraremos las especificaciones técnicas necesarias y otros recursos que nos pueden ser útiles.

```
            dlgConfig.NumeroSerie = numeroSerie0;

            if (dlgConfig.ShowDialog() == DialogResult.OK)
            {
                // Escribimos el nuevo valor
                numeroSerie0 = dlgConfig.NumeroSerie;

                // Abrimos la llave
                RegistryKey sfwKey =
Registry.LocalMachine.OpenSubKey("Software", true);
                RegistryKey redKey = sfwKey.CreateSubKey("RedUsers2");

                RegistryKey robotKey = redKey.CreateSubKey("Robots2");

                // Colocamos el nuevo valor
                robotKey.SetValue("NumeroSerie0", (object)numeroSerie0);

            }
        }


        private void configurarPhidget1ToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            // Este codigo es para el phidget 1

            Configuracion dlgConfig = new Configuracion();
            dlgConfig.NumeroSerie = numeroSerie1;

            if (dlgConfig.ShowDialog() == DialogResult.OK)
            {
                // Escribimos el nuevo valor
                numeroSerie1 = dlgConfig.NumeroSerie;

                // Abrimos la llave
                RegistryKey sfwKey =
Registry.LocalMachine.OpenSubKey("Software", true);
                RegistryKey redKey = sfwKey.CreateSubKey("RedUsers2");

                RegistryKey robotKey = redKey.CreateSubKey("Robots2");
```

```
                // Colocamos el nuevo valor
                robotKey.SetValue("NumeroSerie1", (object)numeroSerie1);

            }
        }
```

A continuación vemos el handler del evento **Load**, que sí presenta ciertas modificaciones que deberemos tomar en cuenta.

```
        private void Form1_Load(object sender, EventArgs e)
        {
            // Inicializamos el objeto InterfaceKit
            try
            {
                // Codigo para el kit de interfaz

                // Instanciamos el objeto
                iKit = new InterfaceKit();

                // Colocamos los handlers relacionados con la conexion
                iKit.Attach += new AttachEventHandler(iKit_ConectarHandler);
                iKit.Detach += new
DetachEventHandler(iKit_DesconectarHandler);

                // Colocamos el handler relacionado con el error
                iKit.Error += new ErrorEventHandler(iKit_ErrorHandler);

                // Colocamos el handler para las entradas digitales
                //iKit.InputChange += new
InputChangeEventHandler(iKit_DigitalHandle);
```

{} CONVERTIR LA DISTANCIA A CENTÍMETROS

El sonar nos brinda la distancia por medio de un valor analógico, pero es posible que necesitemos convertirla a centímetros. Lo importante del software es que trabaje todo el tiempo con las mismas unidades. Para obtener la distancia en centímetros, usamos la siguiente formula: **DistCm= ValorSensor * 1.296**.

```
                // Colocamos el handler para las entradas analogicas
                iKit.SensorChange += new
SensorChangeEventHandler(iKit_AnalogoHandle);

                // Abrimos para conexiones
                iKit.open(numeroSerie0);

                // Usamos entrada ratiometric
                iKit.ratiometric = true;



            } // Fin de try
            catch (PhidgetException ex)
            {
                // Enviamos mensaje con el error
                lblPhidget0.Text = ex.Description;

                // Indicamos con amarillo que hay problemas
                txtPhidget0.BackColor = System.Drawing.Color.Yellow;
            }


            try
            {
                // Codigo para el controlador de servos

                // Instanciamos el objeto
                mServo = new AdvancedServo();

                // Colocamos los handlers relacionados con la conexion
                mServo.Attach += new
AttachEventHandler(mServo_ConectarHandler);
                mServo.Detach += new
DetachEventHandler(mServo_DesconectarHandler);

                // Colocamos el handler relacionado con el error
                mServo.Error += new ErrorEventHandler(mServo_ErrorHandler);

                // Abrimos para conexiones
```

```
            mServo.open(numeroSerie1);

            if (mServo.Attached == true)
            {
                // Inicializamos el track

                // Colocamos los limites de posicion de los servos
                mServo.servos[0].PositionMin = 35;
                mServo.servos[0].PositionMax = 210;

                mServo.servos[1].PositionMin = 30;
                mServo.servos[1].PositionMax = 210;
            }
        }
        catch (PhidgetException ex)
        {
            // Enviamos mensaje con el error
            lblPhidget1.Text = ex.Description;

            // Indicamos con amarillo que hay problemas
            txtPhidget1.BackColor = System.Drawing.Color.Yellow;
        }

    }
```

Debemos notar que, en el handler, establecemos los límites de posición de dos servos. Uno de ellos se encuentra en el índice **0** y es el que se encarga de rotar el brazo. El otro servo se encuentra en el índice **1** y es el que mueve al sonar.

Por otro lado, los handlers para los eventos de conexión, desconexión y errores del Phidget 8/8/8 no presentan cambios.

III VALORES PARA EL RANGO DE LOS SERVOS

Los valores para el rango de los servos que establecimos en el programa se obtuvieron de manera experimental para el modelo de servos que usamos en este proyecto en especial. Si se usan otros servos o se modifica el diseño es muy probable que tengamos que cambiar estos valores.

```
// Handler de la conexion
public void iKit_ConectarHandler(object sender, AttachEventArgs e)
{
    // Indicamos que hay conexion
    conectado0 = true;

    // Enviamos mensaje de la conexion
    lblPhidget0.Text = e.Device.Name;

    // Indicamos con color verde que hay conexion
    txtPhidget0.BackColor = System.Drawing.Color.Lime;

}

// Handler de desconexion
public void iKit_DesconectarHandler(object sender, DetachEventArgs e)
{
    // Indicamos que hay desconexion
    conectado0 = false;

    // Quitamos mensaje
    lblPhidget0.Text = "Sin phidget";

    // Indicamos con color rojo que no hay conexion
    txtPhidget0.BackColor = System.Drawing.Color.Red;

}

// Handler del error
public void iKit_ErrorHandler(object sender, ErrorEventArgs e)
{
```

III FRECUENCIA DE SENSADO

El sonar tiene una frecuencia de sensado de 20 Hz. Esto nos indica que puede hacer hasta 20 mediciones en un segundo. Dicho valor es importante, pues debemos brindar al sonar suficiente tiempo para realizar su medición y evitar así lecturas erróneas. La lógica del programa debe limitarse a trabajar con esta frecuencia.

```
            // Indicamos que hay desconexion
            conectadoO = false;

            // Mandamos mensaje de error
            lblPhidgetO.Text = e.Description;

            // Indicamos con color amarillo que hay problemas
            txtPhidgetO.BackColor = System.Drawing.Color.Yellow;

        }
```

En el handler del evento **FormClosing**, sí tenemos que modificar el código.

```
        private void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {
            // Quitamos los handlers
            iKit.Attach -= new AttachEventHandler(iKit_ConectarHandler);
            iKit.Detach -= new DetachEventHandler(iKit_DesconectarHandler);
            iKit.Error -= new ErrorEventHandler(iKit_ErrorHandler);

            //iKit.InputChange -= new
InputChangeEventHandler(iKit_DigitalHandle);
            iKit.SensorChange -= new
SensorChangeEventHandler(iKit_AnalogoHandle);

            // Regresamos el servo al centro
            if (mServo.Attached == true)
            {
                mServo.servos[0].VelocityLimit = 25;
                mServo.servos[0].Engaged = true;
                mServo.servos[0].Position = 35;
                Thread.Sleep(500);
                mServo.servos[0].Engaged = false;

                mServo.servos[1].VelocityLimit = 1000;
                mServo.servos[1].Engaged = true;
                mServo.servos[1].Position = 120;
                Thread.Sleep(500);
                mServo.servos[1].Engaged = false;
```

```
                mServo.Attach -= new
AttachEventHandler(mServo_ConectarHandler);
                mServo.Detach -= new
DetachEventHandler(mServo_DesconectarHandler);
                mServo.Error -= new
ErrorEventHandler(mServo_ErrorHandler);
            }


            Application.DoEvents();

            // Cerramos el phidget
            iKit.close();
            mServo.close();

        }
```

El punto más importante de este cambio es la ubicación de los servos. Debemos ubicar ambos servos en una posición que sea útil para el robot. Al servo con el índice **0**, encargado de mover el brazo, debemos ubicarlo en su posición más baja. De esta forma, el robot queda estable. Para evitar el volcado del robot, establecemos una velocidad baja para este movimiento. Al otro servo, lo ubicamos en una posición centrada y, al ser el sonar muy pequeño, podemos moverlo rápido y sin problemas.

```
        public void iKit_AnalogoHandle(object sender, SensorChangeEventArgs e)
        {
            // Verificamos que estemos conectados
            if (conectado0)
            {
                // Verificamos que sea la entrada 0
                if (e.Index == 0)
                {
                    // Colocamos el valor
                    lblAnalogico0.Text = e.Value.ToString();

                }

            }
        }
```

La función del handler para el cambio del valor analógico es muy sencilla. Sólo obtenemos el valor del sensor y lo mostramos en la etiqueta correspondiente. La información del sonar debe entrar por el puerto analógico **0** para que la etiqueta sea actualizada. El handler para el evento de conexión del Phidget controlador de los servos lleva unas adiciones, pues en este caso tenemos que lidiar con dos servos.

```
public void mServo_ConectarHandler(object sender, AttachEventArgs e)
 {
     // Indicamos que hay conexion
     conectado1 = true;

     // Enviamos mensaje de la conexion
     lblPhidget1.Text = e.Device.Name;

     // Indicamos con color verde que hay conexion
     txtPhidget1.BackColor = System.Drawing.Color.Lime;



     // Configuramos sus caracteristicas
     if (mServo.Attached)
     {
         // Configuramos el servo 0

         // Colocamos la aceleracion
         mServo.servos[0].Acceleration = 1000.0;

         // Colocamos la posicion inicial
         mServo.servos[0].Engaged = true;
         mServo.servos[0].Position = 30.0;
         mServo.servos[0].VelocityLimit = 25.0;

         // Configuramos el servo 1

         // Colocamos la aceleracion
         mServo.servos[1].Acceleration = 1000.0;

         // Colocamos la posicion inicial
         mServo.servos[1].Engaged = true;
         mServo.servos[1].Position = 120.0;
```

```
                mServo.servos[1].VelocityLimit = 1000.0;


            }

        }
```

En este handler, configuramos ambos servos. Indicamos su posición inicial, la aceleración y la velocidad de ambos. Cada uno tiene diferente posición inicial. La diferencia en las velocidades de los servos es para evitar el volteo del robot cuando el brazo se mueva a su posición inicial.

Los handlers para el controlador de servos que hacen referencia al evento de desconexión y error no sufren cambios.

```
        public void mServo_DesconectarHandler(object sender,
DetachEventArgs e)
        {
            // Indicamos que hay desconexion
            conectado1 = false;

            // Quitamos el mensaje
            lblPhidget1.Text = "Sin Phidget";

            // Indicamos con color rojo que no hay conexion
            txtPhidget1.BackColor = System.Drawing.Color.Red;

        }
```

III CUIDADO CON LA PRIMERA CONEXIÓN

En algunas ocasiones puede ocurrir que, al conectar por primera vez el controlador de servos de Phidgets, éste realiza un movimiento de los servos. Debemos estar atentos para evitar que el robot se vuelque en caso de que dicho movimiento se efectúe de una forma excesivamente rápida.

```
public void mServo_ErrorHandler(object sender, ErrorEventArgs e)
{
    // Indicamos que hay desconexion
    conectado1 = false;

    // Mandamos mensaje de error
    lblPhidget1.Text = e.Description;

    // Indicamos con color amarillo que hay problemas
    txtPhidget1.BackColor = System.Drawing.Color.Yellow;
}
```

Para llevar a cabo algunos cálculos necesarios para mostrar la información del sonar, haremos uso de una función de apoyo. Esta función se llama **AnguloRadian** y nos sirve para convertir de grados a radianes. Esto es necesario, pues las funciones trigonométricas que utilizaremos deben tener los ángulos en radianes. La función necesita de un parámetro de tipo **double**, que es el valor del ángulo en grados por transformar. Como resultado, se devolverá un valor de tipo **double** que es el ángulo convertido a radianes.

```
private double AnguloRadian(double angulo)
{
    double radianes;
    radianes = angulo * Math.PI / 180;

    return radianes;
}
```

Ahora, podemos crear el handler para el botón. Este handler es el responsable de hacer las inicializaciones necesarias para que el robot realice su trabajo.

```
private void btnRadarInicio_Click(object sender, EventArgs e)
{
    // Inicializamos valores
    mAvanceServo = (205.0 - 45.0) / 180.0;
    mAngulo = 0;
    mAnguloServo = 45.0;
```

```
            // Colocamos el estado actual
            estado = 1;

            // Inicializamos los servos

            // Colocamos el brazo en el lado derecho
            mServo.servos[0].Engaged = true;
            mServo.servos[0].Position = 35;
            Thread.Sleep(500);
            // mServo.servos[0].Engaged = false;

            // Colocamos el radar en su posicion de inicio
            mServo.servos[1].Engaged = true;
            mServo.servos[1].Position = 45;
            Thread.Sleep(500);
            mServo.servos[1].Engaged = false;

            // Prendemos el timer
            timer1.Enabled = true;
        }
```

El handler empieza calculando el valor de avance del servo. Si tenemos servos de modelos diferentes, éste es el lugar donde debemos hacer los cambios necesarios. La fórmula para calcular el avance resulta sencilla de realizar. Sabemos que el servo del sonar debe barrer **180** grados avanzando de grado en grado. El valor **205** es el valor de la posición del servo cuando el sonar se encuentra viendo hacia la derecha, es decir, en su posición de **180** grados. El valor **45**, es el correspondiente a la posición del servo cuando el sonar se encuentra viendo totalmente hacia la izquierda, es decir, en su posición de **0** grados. En todos estos valores, nos referimos al servo del sonar, no al del movimiento del brazo.

Inicializamos **mAngulo** en el valor **0**, pues el sonar se iniciará en el ángulo **0** grados. La variable **mAnguloServo** es inicializada en **45.0**, dado que éste es el valor de su posición cuando el sonar está viendo a hacia la posición **0** grados. Como vamos a iniciar el primer recorrido del sonar, establecemos el estado con el valor **1**. Este estado corresponde al recorrido del sonar en el lado derecho. Luego, simplemente, ubicamos los dos servos en su posición inicial. La mayor parte de la lógica está contenida en el handler del **Timer**, para ese propósito usamos la máquina de estados finitos. Por este motivo, es necesario que iniciemos el **Timer**. Esto lo hacemos al establecer su propiedad de **Enabled** con el valor **true**. Es el momento de crear el

handler para el **Timer**. Dentro de él, estableceremos la lógica principal de la aplicación. Para entender de manera correcta el código del handler, es necesario recordar que usamos una máquina de estados finitos.

```
        private void timer1_Tick(object sender, EventArgs e)
         {

             if (estado == 1) // Recorrido del lado derecho
             {
                 // Verificamos que estemos en el rango
                 if (mAngulo >= 180)
                 {

                     // Cambiamos el estado
                     estado = 2;
                 }
                 else
                 {

                     // Colocamos el servo en posicion
                     mServo.servos[1].Engaged = true;
                     mServo.servos[1].Position = mAnguloServo;
                     Thread.Sleep(150);
                     mServo.servos[1].Engaged = false;

                     // Asignamos el valor
                     System.Diagnostics.Debug.WriteLine("Value = " +
mAngulo.ToString());
                     mDatosDerecho[mAngulo] = iKit.sensors[0].Value;

                     // Mostramos el valor
                    lblAnalogico0.Text = mDatosDerecho[mAngulo].ToString();

                     // Avanzamos el angulo
                     mAngulo++;
                     // Incrementamos el servo
                     mAnguloServo += mAvanceServo;
                 }
             }
```

```
if (estado == 2)
{
    // Mandamos el sonar al lado izquierdo

    // Colocamos el brazo en el lado derecho
    mServo.servos[0].Engaged = true;
    mServo.servos[0].Position = 195;
    Thread.Sleep(500);

    // Colocamos el servo en posicion
    mServo.servos[1].Engaged = true;
    mServo.servos[1].Position = 160;
    Thread.Sleep(150);
    mServo.servos[1].Engaged = false;

    // Reiniciamos los valores
    mAngulo = 0;

    // Cambiamos el estado
    estado = 3;

}
if (estado == 3)
{
    // Leemos la informacion del lado izquierdo

    // Verificamos que estemos en el rango
    if (mAngulo >= 180)
    {
        timer1.Enabled = false; // Paramos el timer

        // Cambiamos el estado
        estado = 4;
    }
    else
    {

        // Colocamos el servo en posicion
        mServo.servos[1].Engaged = true;
        mServo.servos[1].Position = mAnguloServo;
```

```
                    Thread.Sleep(150);
                    mServo.servos[1].Engaged = false;

                    // Asignamos el valor
                    System.Diagnostics.Debug.WriteLine("Value = " +
mAngulo.ToString());
                    mDatosIzquierdo[mAngulo] = iKit.sensors[0].Value;

                    // Mostramos el valor
                    lblAnalogico0.Text =
mDatosIzquierdo[mAngulo].ToString();

                    // Avanzamos el angulo
                    mAngulo++;
                    // Incrementamos el servo
                    mAnguloServo -= mAvanceServo;
                }

            }

            if (estado == 4)
            {
                // Mandamos a dibujar
                DibujaSonar();
            }


        }
```

En este punto, tenemos varios estados. El estado **0** es cuando el robot está en reposo sin hacer nada. El estado **1** se encarga del recorrido del lado derecho. En el estado **2**, mandamos al brazo robótico y al sonar hacia el lado izquierdo. Durante el estado **3**, se lee la información del lado izquierdo y, por último, el estado **4** es el encargado de invocar el dibujo de la información del sonar. Durante el estado **1**, verificamos que no hayamos excedido el valor máximo de grados, ya que, si lo hacemos, cambiamos de estado. Si estamos dentro del rango, entonces, ubicamos al servo del sonar en la posición que le corresponde, esperamos unos milisegundos y luego continuamos.

Después, obtenemos el valor del sonar; esto lo hacemos al leer el valor del sensor analógico que se encuentra en el índice **0**. Ese valor es ingresado en el arreglo de

mDatosDerechos en el índice que le corresponde de acuerdo con **mAngulo**. La etiqueta, también, se actualiza de acuerdo con el nuevo valor. Luego, avanzamos el ángulo y el valor de la posición del servo. En el estado **2**, tan sólo se realiza el movimiento del brazo. El brazo gira para ubicar al sonar en el lado izquierdo, y el servo que mueve al sonar es posicionado para que el sonar pueda llevar a cabo su escaneo. El valor **mAngulo** se resetea a **0**, pues vamos a escanear de nuevo desde el ángulo **0**. El paso siguiente será hacer el cambio al caso **3**.

El estado **3** es similar al estado **1**, pero los eventos suceden en el lado izquierdo. Los datos se guardan en el arreglo **mDatosIzquierdo**. Debemos notar que la posición del servo decrece en este estado. El estado **4** se encarga de mandar a dibujar la información recopilada del sonar. Para finalizar, cambiamos al estado **0**: así, hemos completado el ciclo del robot. La última función que tiene el programa es la encargada de realizar el dibujo, y se llama **DibujaSonar()**.

```
private void DibujaSonar()
{
    // Variables necesarias

    // Centro del primer radar
    int c1x = 255;
    int c1y = 510;

    // Centro del segundo radar
    int c2x = 271;
    int c2y = 510;

    // Variables para el sonar
    int distancia = 100;
    double radianActual = 0.0;
    float seno = 0.0f;
    float coseno = 0.0f;
    float x1 = 0.0f;
    float y1 = 0.0f;
    float x2 = 0.0f;
    float y2 = 0.0f;

    // Creamos las plumas
    Pen plumaIzquierda = new Pen(Color.Blue, 2);
    plumaIzquierda.DashStyle = DashStyle.Dot;
```

```
            Pen plumaDerecha = new Pen(Color.Red, 2);
            plumaDerecha.DashStyle = DashStyle.Dot;

            // Obtenemos un objeto grafico de la ventana
            Graphics g = Graphics.FromHwnd(this.Handle);

            // Dibujamos el area de sensado izquierdo
            g.FillPie(Brushes.Black, new Rectangle(0, 255, 512, 512), 180,
180);

            // Dibujamos el area del sensado derecho
            g.FillPie(Brushes.Black, new Rectangle(16, 255, 512, 512),
180, 180);

            // Dibujamos los origenes del radar
            g.FillEllipse(Brushes.Blue, c1x, c1y, 5, 5);
            g.FillEllipse(Brushes.Red, c2x, c2y, 5, 5);

            //Dibujamos lado derecho
            // Recorremos 180 grados
            for (int angulo = 0; angulo < 180; angulo++)
            {

                // Leemos la distancia
                distancia = mDatosDerecho[angulo];

                // Obtenemos el angulo en radianes
                radianActual = AnguloRadian((double)angulo);

                // Obtenemos el seno y coseno del angulo
                seno = (float)Math.Sin(radianActual);
                coseno = (float)Math.Cos(radianActual);

                // Obtenemos las coordenadas del inicio de la linea
                x1 = c2x + (distancia * coseno);
                y1 = c2y - (distancia * seno);

                // Obtenemos las coordenadas del fin de la linea
                x2 = c2x + (255 * coseno);
                y2 = c2y - (255 * seno);
```

```
                g.DrawLine(plumaDerecha, x1, y1, x2, y2);

            }

            //Dibujamos lado izquierdo
            // Recorremos 180 grados
            for (int angulo = 0; angulo < 180; angulo++)
            {

                // Leemos la distancia
                distancia = mDatosIzquierdo[angulo];

                // Obtenemos el angulo en radianes
                radianActual = AnguloRadian((double)angulo);

                // Obtenemos el seno y coseno del angulo
                seno = (float)Math.Sin(radianActual);
                coseno = (float)Math.Cos(radianActual);

                // Obtenemos las coordenadas del inicio de la linea
                x1 = c1x + (distancia * coseno);
                y1 = c1y - (distancia * seno);

                // Obtenemos las coordenadas del fin de la linea
                x2 = c1x + (255 * coseno);
                y2 = c1y - (255 * seno);

                g.DrawLine(plumaIzquierda, x1, y1, x2, y2);

            }

            g.Dispose();

        }

    }
}
```

Necesitaremos diferentes variables. Se dibujará la información recopilada del lado derecho y del lado izquierdo. Al ser el dibujo de ambos lados muy similar, expli-

caremos sólo una. Primero, declaramos las variables que indican las coordenadas de la forma en que se encontrará el centro del dibujo para la información del lado derecho y del izquierdo. Estas posiciones simbolizan el lugar donde se encuentra el sonar cuando está escaneando.

Necesitamos una serie de variables de apoyo para realizar los cálculos indispensables para el dibujo. Cada valor del sonar será representado por una línea que empieza a dibujarse a una distancia igual a la leída por el sonar y se traza en el ángulo correspondiente a esa lectura. Creamos dos plumas para dibujar las líneas. La pluma que dibuja el lado izquierdo es azul, y la que dibuja el lado derecho es roja. Las plumas marcarán líneas punteadas. A continuación, obtenemos el objeto gráfico de la ventana para poder dibujar sobre ella. Dibujamos el área de sensado de cada lado. Éstas se dibujan como medios círculos de color negro. Sobre ellas, se perfilarán las líneas con la información. Las posiciones del sonar son dibujadas como dos pequeños círculos, cada uno de ellos del color que corresponde a su lado.

Para dibujar las líneas, recorremos el arreglo que contiene la información. Leemos el valor del arreglo y lo guardamos dentro de la variable distancia. También, cambiamos el ángulo que se encuentra en grados a radianes y, por medio de operaciones trigonométricas, calculamos la posición inicial y la final de la línea que dibujaremos en ese momento. Luego, se invoca a la función **DrawLine()** para dibujar la línea correspondiente en el color que le es asignado, dependiendo del lado de lectura del sonar. Por último, liberamos el objeto gráfico por medio de la función **Dispose()**.

Ahora, ya podemos compilar y probar el programa. El robot será capaz de escanear la distancia a los objetos que se encuentran a su alrededor, tanto del lado derecho, como del izquierdo. Más adelante, podremos ver una representación gráfica de las distancias a los objetos.

... RESUMEN

En este capítulo, aprendimos que el sonar es un dispositivo que nos permite medir distancias por medio del eco producido por los objetos. Nuestro robot tiene un pequeño sonar que da salidas analógicas que podemos usar en nuestra aplicación. Además, vimos que el robot tiene dos grados de libertad y puede hacer mediciones de sonar desde dos posiciones diferentes. El sonar está montado sobre un servo, lo cual le permite hacer lecturas en un arco de 180 grados. Para facilitar el proceso de automatización, hacemos uso de una máquina de estados finitos. La información leída durante el escaneo del sonar es dibujada para poder observarla de manera gráfica.

TEST DE AUTOEVALUACIÓN

1. Modifique el programa para que las distancias sean dadas en centímetros en lugar de unidades del sonar.

2. Modifique el programa para poder indicar el arco por escanear, en lugar de los 180 grados predeterminados

3. Modifique el programa para que se puedan salvar los valores medidos durante el escaneo del sonar

4. Modifique el programa para que suene una alarma si un objeto se acerca a menos de determinada distancia.

5. Modifique el programa para poder mover manualmente el brazo robótico.

6. Modifique el programa y el robot para que use dos sonares al mismo tiempo.

7. Modifique el programa para que detecte movimiento de objetos.

8. Modifique el programa para que indique en qué ángulo hay espacio libre para poder moverse.

9. Modifique el programa para que muestre la distancia más corta y la más larga.

10. Modifique el robot para adicionar una webcam y que se muestre en la imagen la distancia de lo que se está midiendo.

11. Modifique el programa para que realice varias lecturas y, de esta forma, disminuir cualquier posible error de lectura.

12. Modifique el programa para que, con la información de las dos posiciones, muestre la información del espacio en 3D.

Robot animatrónico

En este capítulo, realizaremos un proyecto sumamente divertido: la creación de un robot animatrónico. Este tipo de robots tienen un uso muy extendido en aplicaciones usadas para entretenimiento. El robot, por lo general, tiene la forma de un personaje histórico o imaginario y realiza una animación programada con anterioridad. Nosotros crearemos un pequeño personaje ficticio para mostrar en nuestro proyecto.

DESCRIPCIÓN DEL PROYECTO

Crearemos un pequeño **robot animatrónico** que representa un cíclope. Tendrá diferentes elementos móviles como la mandíbula, el ojo y la ceja. El software controlará la posición de cada uno de esos elementos durante el paso del tiempo. Sólo crearemos un minuto de animación, pero esto será suficiente para entender el concepto. Este proyecto necesita bastante trabajo para poder crearlo, y usaremos tres servos, sin embargo, el código será más sencillo que el de los proyectos anteriores.

Componentes necesarios

La cantidad de componentes es numerosa, y la siguiente tabla los lista:

NOMBRE	CANTIDAD	DESCRIPCIÓN
A	1	Cartón 18 x 22 cm, 1 mm de espesor
B	1	Cartón 2.5 x 2.5 cm, 2 mm de espesor
C	1	Cartón 2.5 x 2.5 cm, 1 mm de espesor
D	2	Cartón 2 x 1 cm, 4 mm de espesor
E	2	Cartón 5 x 2 cm, 3 mm de espesor
F	2	Cartón 2 x 2 cm, 3 mm de espesor
G	1	Cartón 5 x 12 cm, 1 mm de espesor
H	2	Cartón 4 x 3.5 cm, 3 mm de espesor
I	1	Cartón 2.5 x 2.8 cm, 3 mm de espesor
J	1	Cartón 2.5 x 2 cm, 5 mm de espesor
K	1	Cartón 2.5 x 2 cm, 3 mm de espesor
L	2	Cartón 3 x 5.5 cm, 3 mm de espesor
M	2	Cartón 4.5 x 5 cm, 1 mm de espesor
N	2	Cartón 2 x 1 cm, 3 mm de espesor
O	1	Cartón 2.7 x 7 cm, 1 mm de espesor
P	1	Cartón 2.7 x 1 cm, 3 mm de espesor
Q	1	Cartón 2.7 x 1 cm, 1 mm de espesor
R	1	Cartón 2.7 x 4.5 cm, 1 mm de espesor
S	2	Cartón 7 x 2 cm, 3 mm de espesor
T	2	Cartón 7 x 2 cm, 2 mm de espesor
U	4	Cartón 2 x 1 cm, 3 mm de espesor
V	1	Cartón 12 x 7 cm, 1 mm de espesor
Controlador servos	1	PhidgetAdvancedServo
Servo	3	Servos para mover el animatrónico
Esfera	1	Esfera de unicel (telgopor)

Tabla 1. *En la tabla anterior, podemos observar la lista de los componentes que nos permitirán construir el proyecto actual.*

CONSTRUCCIÓN DEL ROBOT

El proyecto muestra el rostro del personaje, que es un cíclope. La parte más importante es la cara, no sólo porque es lo que mostramos al usuario, sino porque servirá de soporte para todo el sistema mecánico y para los servos. Empezamos la cara haciendo uso del componente **A**, el cual es un rectángulo de 18 por 22 centímetros. A este componente, será necesario que le realicemos algunos trazos para hacer los cortes que indicarán el lugar donde colocaremos otros elementos.

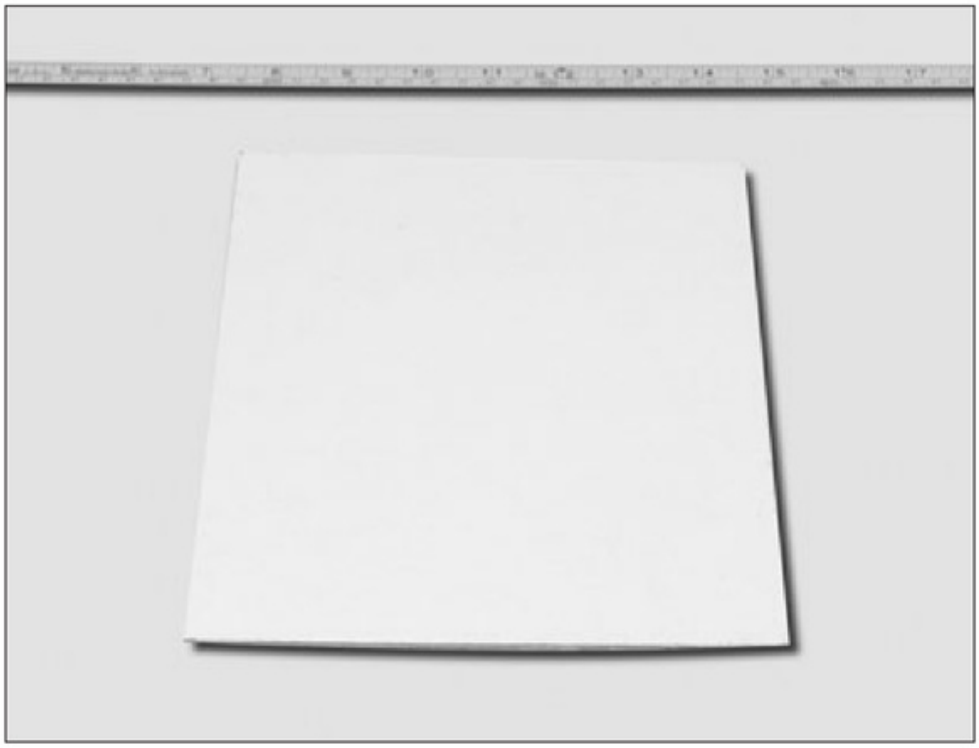


Figura 1. *Con este cartón, construiremos la cara y el soporte para los elementos mecánicos.*

Empecemos a hacer los trazos necesarios para la cara. Lo primero es establecer una línea horizontal exactamente en el centro del cartón, esto es, a 9 cm de su extremo derecho. Sobre esta línea, trazamos una marca a 1.5 cm del borde superior y dibujamos otra a 10 cm del mismo borde. Un poco más abajo, a 16 cm del borde superior, establecemos una tercera marca. Usando esta última marca como guía, trazamos una línea horizontal. Usamos un compás para dibujar una circunferencia de 6.5 cm de diámetro. El centro de la circunferencia debe estar sobre la línea vertical que dibujamos inicialmente. La parte más baja de la circunferencia debe tocar la marca de los 10 cm. Ahora, nos ubicamos en la esquina superior izquierda. Usando esta esquina como referencia, trazamos una marca a 2 cm del borde derecho del cartón y otra, a 2 cm del borde superior. Hacemos lo mismo con la esquina superior derecha, pero, en este caso, una marca se encuentra a 2 cm del borde superior y otra, a 2 cm del borde derecho. Usando como referencia la línea horizontal que dibujamos, nos dirigimos hasta su borde derecho y realizamos una

marca a 2 cm de la línea horizontal sobre el borde derecho. Procedemos a realizar lo mismo, pero, en esta ocasión, del lado izquierdo.

Sólo faltan dos marcas. Usamos el borde inferior y buscamos el lugar por donde pasa la línea vertical. Hacemos uso de esta línea como referencia y trazamos una marca a 2 cm de la línea hacia la derecha y otra, hacia la izquierda a la misma distancia. Las marcas deben quedar sobre el borde inferior del cartón. La siguiente figura nos muestra cómo ha quedado marcado el cartón.

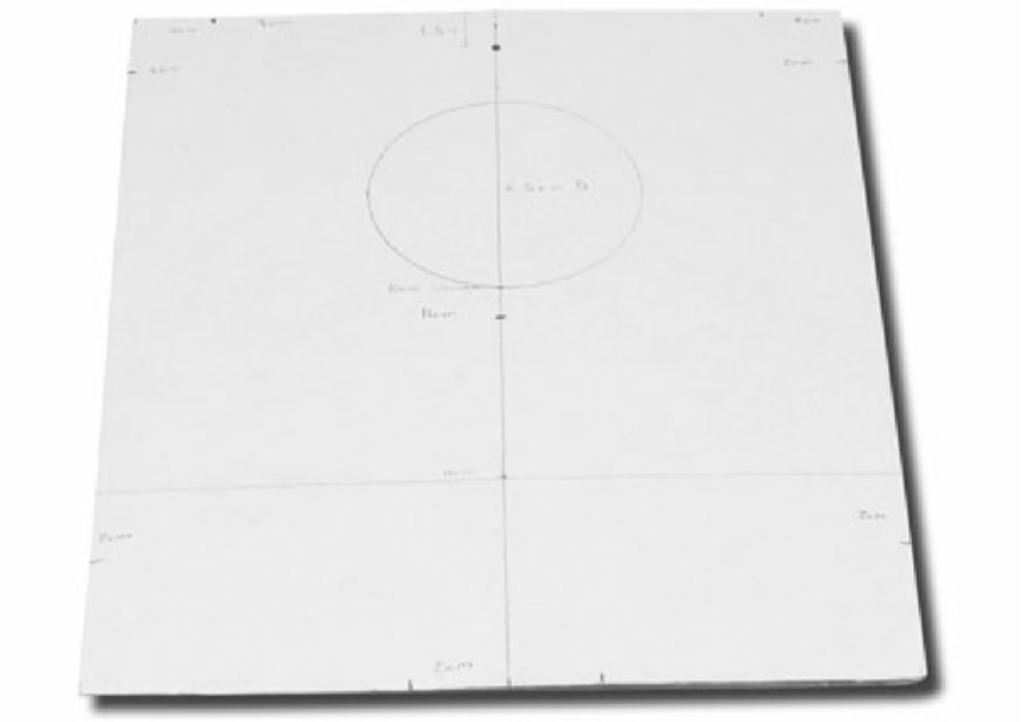


***Figura 2.** Éstas son las señales que debemos marcar sobre el cartón y que, más adelante, nos servirán como guías.*

A continuación, debemos recortar la circunferencia. Esto lo debemos realizar con mucho cuidado para así evitar dañar al cartón o tener un accidente. Este espacio servirá para que el ojo se asome y el usuario pueda observar cómo se mueve.

III LOS ROBOTS ANIMATRÓNICOS

Los robots animatrónicos son robots que trabajan con movimientos guardados en lugar de reaccionar a la lógica de una computadora o a los sensores del mundo exterior. Dichos movimientos son las animaciones a las que nos referimos al comenzar el capítulo. En la actualidad, existen varias empresas que fabrican robots animatrónicos.

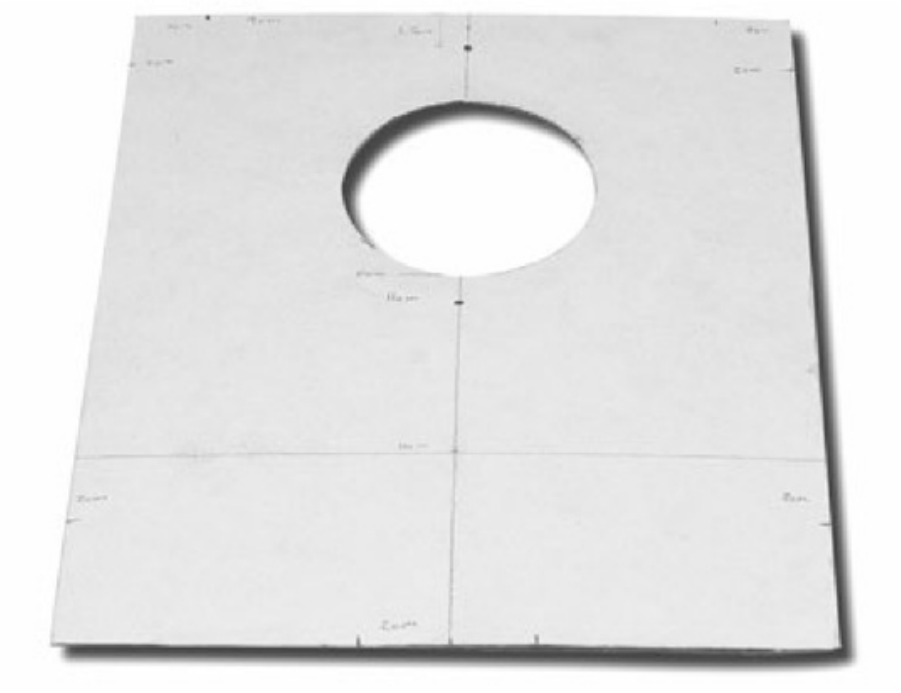


Figura 3. *El espacio para el ojo es el primer corte que realizamos. Se puede hacer con una navaja para cartón.*

Tomando como punto de partida las marcas que efectuamos cerca de los bordes, trazaremos cuatro líneas. Estas líneas nos indicarán los lugares de corte que realizaremos con posterioridad.

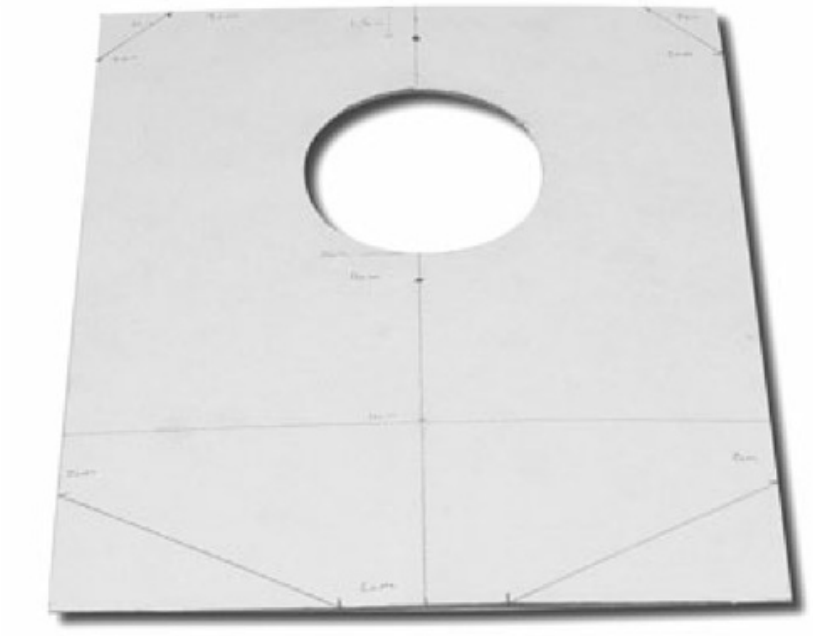


Figura 4. *Trazamos líneas que unen las marcas realizadas en los bordes.*

Con mucho cuidado, usamos las líneas como guías para proceder a cortar las esquinas del cartón. Como podemos observar en la próxima figura, los cortes en la parte inferior son más largos que los superiores.

Figura 5. *Cortamos las esquinas del cartón aprovechando las líneas como guías.*

El último corte que haremos por el momento es el de la línea horizontal. Este corte dividirá al cartón en dos secciones distintas. La parte inferior será utilizada para crear la mandíbula móvil del robot.

Figura 6. *Usando la línea horizontal como guía cortamos el cartón y obtenemos la mandíbula del robot.*

Creación del ojo y de su sistema mecánico

Como ahora tenemos la base lista, podemos continuar creando los demás sistemas. Uno de los sistemas mecánicos del robot es el ojo. El ojo será sencillo y tendrá un movimiento único en un eje. Para crear el ojo usaremos una **esfera de unicel** (telgopor) de entre 6 y 7 centímetros de diámetro. También necesitamos pintura vinílica y un marcador color negro. Tomamos la esfera de unicel y, con un marcador, trazamos una circunferencia de unos 5 centímetros de diámetro. Esta circunferencia delimitará la zona del iris del ojo.

Figura 7. *La circunferencia nos permite indicar el área donde quedará dibujado el iris del ojo. El trazado no tiene por qué ser perfecto.*

Podemos escoger cualquier color de pintura vinílica para el ojo. Con la pintura, rellenamos el interior del iris. Si lo deseamos, podemos incluso utilizar más de un color.

Figura 8. *El iris es pintado del color que consideremos apropiado para la personalidad del personaje; el azul es una buena opción en este ejemplo.*

Con un poco de pintura negra, delineamos la pupila del ojo justo en el centro del iris. La pupila debe ser redonda, pero no es un problema si tiene pequeñas imperfecciones ya que esto ayuda a que se vea más natural.

Si dibujamos unos pequeños círculos de color blanco, podemos dar el aspecto de brillo en el ojo. El círculo superior debe ser el más grande. Dos de los círculos, se colocan cerca del límite del iris, y el otro, cerca del límite de la pupila.

Figura 9. *Aquí podemos observar el ojo finalizado con toda la pintura necesaria.*

Tenemos que montar el ojo sobre un servo y, para esto, es necesario crear una montura que se pueda instalar en el servo y que fije al ojo. Esta montura se construye usando los componentes **B**, **C** y un clavo de alrededor de 3 centímetros de longitud.

Figura 10. *Éstos son los elementos que necesitamos para crear la montura del ojo, que se colocará en el servo.*

III LA CIRCUNFERENCIA DEL OJO

Para poder realizar con facilidad la circunferencia del ojo, podemos usar un vaso o algún otro recipiente similar. Lo colocamos sin presionar sobre el unicel y, con el marcador, procedemos a dibujar la circunferencia. Otro objeto que podemos usar es una rollo de cinta adhesiva.

Al elemento **B**, le dibujamos las marcas en donde irán los tornillos del servo. En el elemento **C**, trazamos dos diagonales para encontrar el punto central.

Figura 11. *Dibujamos los puntos para indicar la posición de los tornillos y el centro del componente* ***C****.*

Con cuidado, hacemos las perforaciones que nos servirán de guía para los tornillos sobre el elemento **B**. Al elemento **C**, le hacemos una perforación en el centro. Esta perforación puede ser realizada con el clavo que utilizaremos.

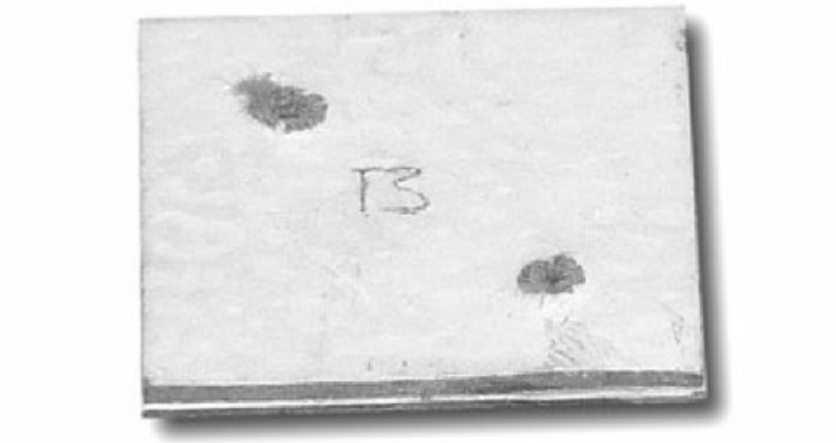


Figura 12. *Llevamos a cabo perforaciones en ambos elementos.*

{} PARA MEJORAR EL ASPECTO DEL OJO

Si queremos que el ojo se vea más natural, podemos pintar con un tono más oscuro del color, la periferia del iris y, con un tono más claro, la parte cercana a la pupila. Los trazos del pincel de forma radial en lugar de circular ayudan mucho a mejorar el aspecto de nuestro ojo ficticio.

Con cuidado, marcamos sobre una de las diagonales del componente **C** las posiciones por donde pasarán los tornillos hacia el servo. Tomamos el clavo y lo pasamos por el orificio que se encuentra en el centro del componente **C**. El clavo debe quedar lo más derecho posible respecto del componente.

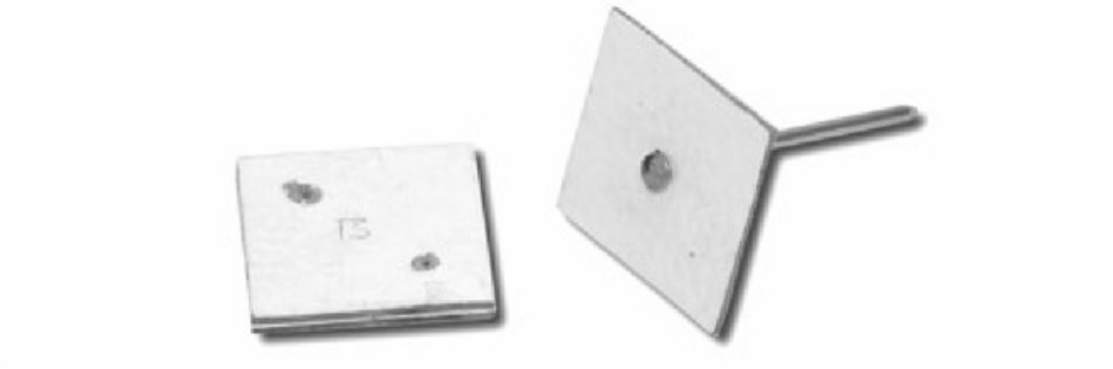


Figura 13. *El clavo debe ser insertado en el orificio del componente **C**.*

A continuación, aplicamos pegamento en la superficie del componente **B** y colocamos el componente **C** con el clavo sobre éste. Debemos tener cuidado de hacer coincidir las marcas de los tornillos con las perforaciones de los tornillos del componente **B**. Además, el montaje debe ser firme en su estructura, y el clavo debe estar lo más perpendicular posible.

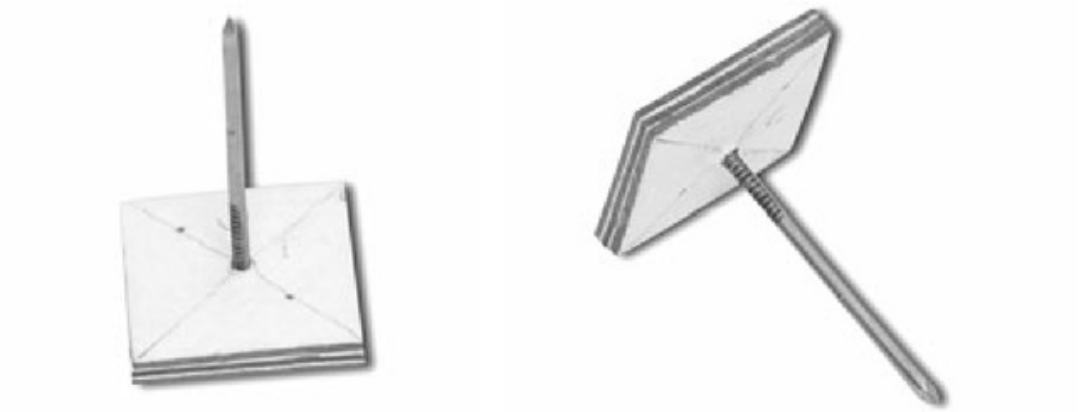


Figura 14. *Como podemos ver desde distintos ángulos en ambas imágenes, el componente **C** es montado sobre el componente **B**. La unión debe ser firme.*

{} OTROS TIPOS DE OJOS

Podemos experimentar dibujando otros tipos de ojos. Por ejemplo, ojos con pupilas verticales o con dibujos de venas en su interior. Esto nos ayuda a darle una personalidad diferente al robot animatrónico. Otra idea que podemos llevar a cabo es ubicar un **LED** en el ojo para iluminarlo.

Seleccionamos uno de los servos para utilizarlo en el movimiento del ojo. Colocamos el servo en posición central. Esto lo hacemos con el fin de que el ojo pueda moverse con libertad y con la mayor amplitud posible en ambas direcciones.

Con el uso de los tornillos, instalamos el sistema de montaje para el ojo sobre el servo. Debemos tener cuidado de que los tornillos no se ubiquen a mucha profundidad, pues podrían estorbar el movimiento.

Figura 15. *El sistema que sostendrá al ojo ha sido montado sobre el servo.*

Ahora que se encuentra montado, recortamos la esquina del mecanismo que se ubica del lado de donde salen los cables del servo. El corte debe realizarse justo arriba del lugar en el que empieza el mecanismo de actuación del servo.

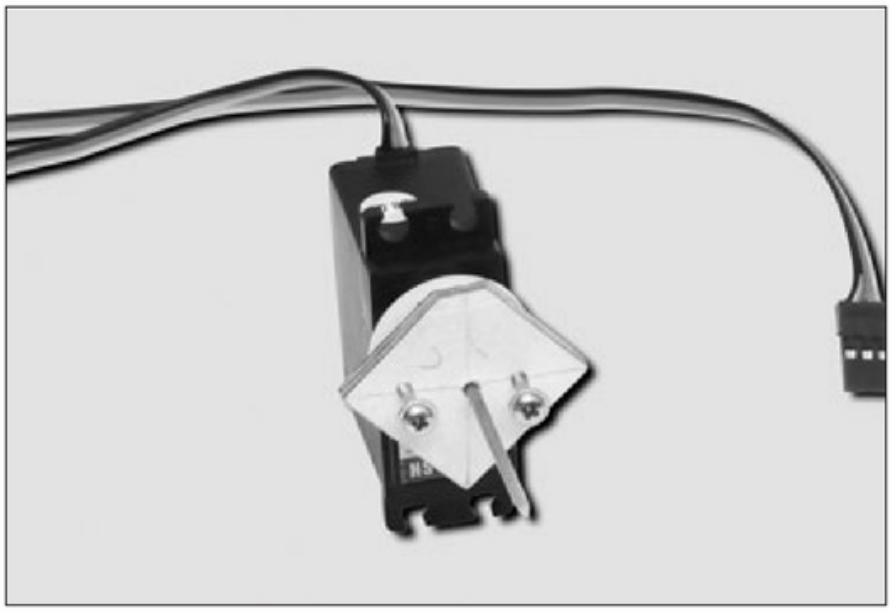


Figura 16. *Este corte facilitará el movimiento del ojo a partir de que el mecanismo no rozará con el soporte de la cara.*

Ahora podemos comenzar a hacer el montaje sobre la cara. Para eso, precisaremos varios componentes. En este caso, serán necesarios los componentes **F**, **E** y **D**, que vienen en pares. Para ayudarnos con el montaje, utilizaremos, también, un servo. Puede ser el servo que acabamos de utilizar o algún otro, porque sólo será necesario como soporte para el armado.

Figura 17. *Estos componentes serán usados para armar el soporte del servo en la cara.*

Tomamos uno de los componentes **E** y, sobre él, ubicamos el servo. Pegamos un componente **D** en el extremo, tal como aparece en la **figura 18**, y usamos el servo como apoyo mientras se termina de pegar.

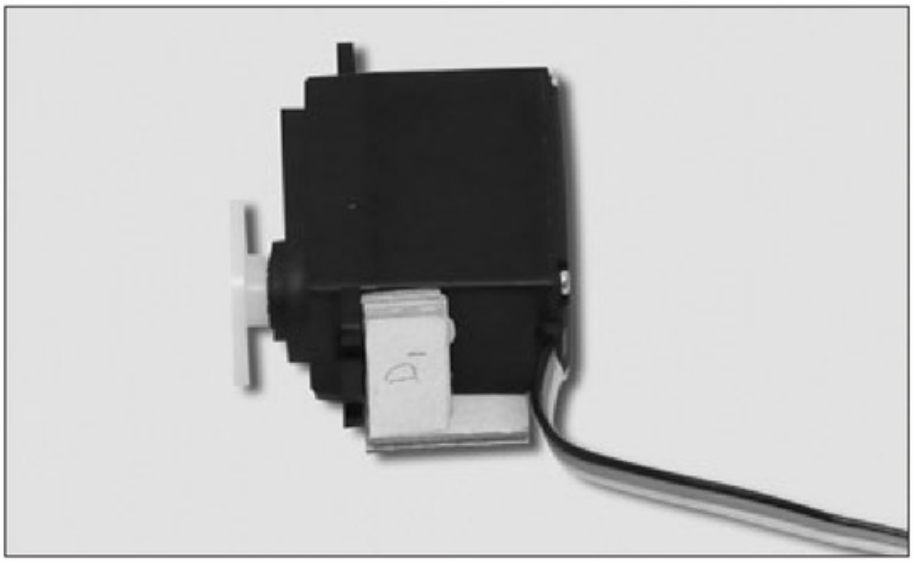


Figura 18. *El componente* ***D*** *sostendrá al servo por medio de uno de sus estribos. Aquí lo vemos ubicado en posición.*

III UNA MEJOR POSICIÓN PARA EL SERVO

Cuando instalamos el servo en posición central, también, podemos cambiar la ubicación del elemento de actuación en cualquier momento. Sólo basta con desatornillarlo y situarlo de tal forma que sus perforaciones queden en una posición que nos sea más útil. Después, volvemos a atornillarlo para que vuelva a quedar firme.

Hacemos el mismo procedimiento del otro lado del servo. Pegamos otro de los componentes **D** y usamos el servo como soporte. Recordemos que el servo no debe ser pegado, sólo los elementos.

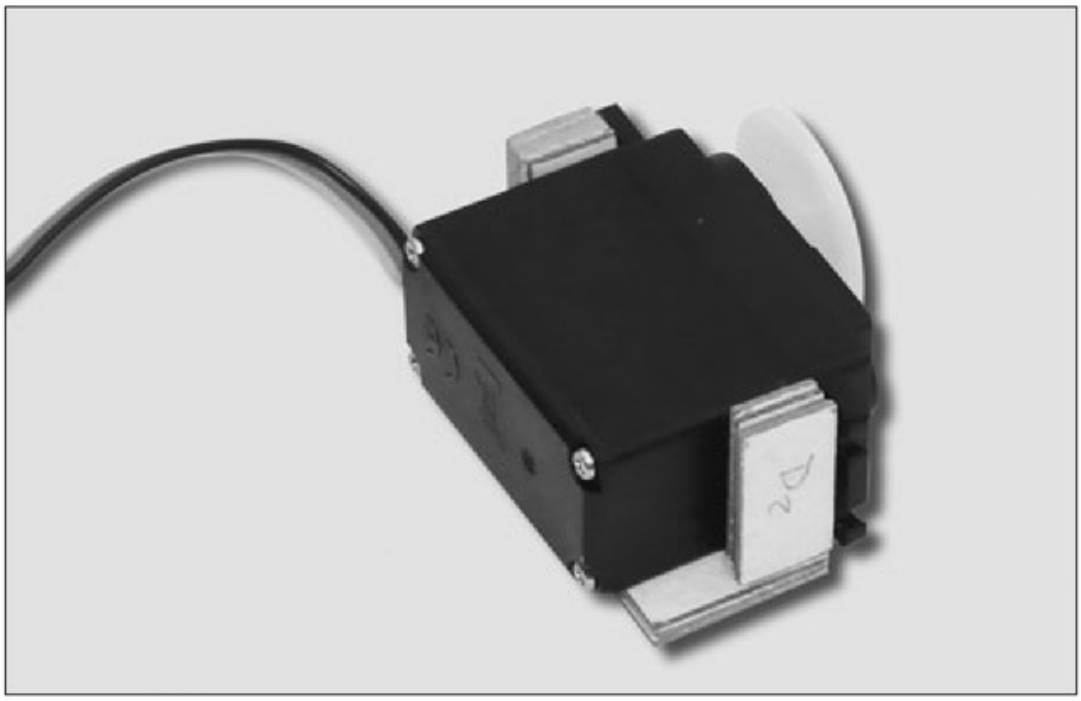


Figura 19. *El otro componente está en posición. Es importante notar la forma en que los estribos del servo son soportados por estos componentes.*

Para terminar de cerrar la caja que actuará como soporte para el servo y brindarle una mayor estabilidad estructural, necesitamos colocar el otro componente **E** y pegarlo a los extremos de los componentes **D**.

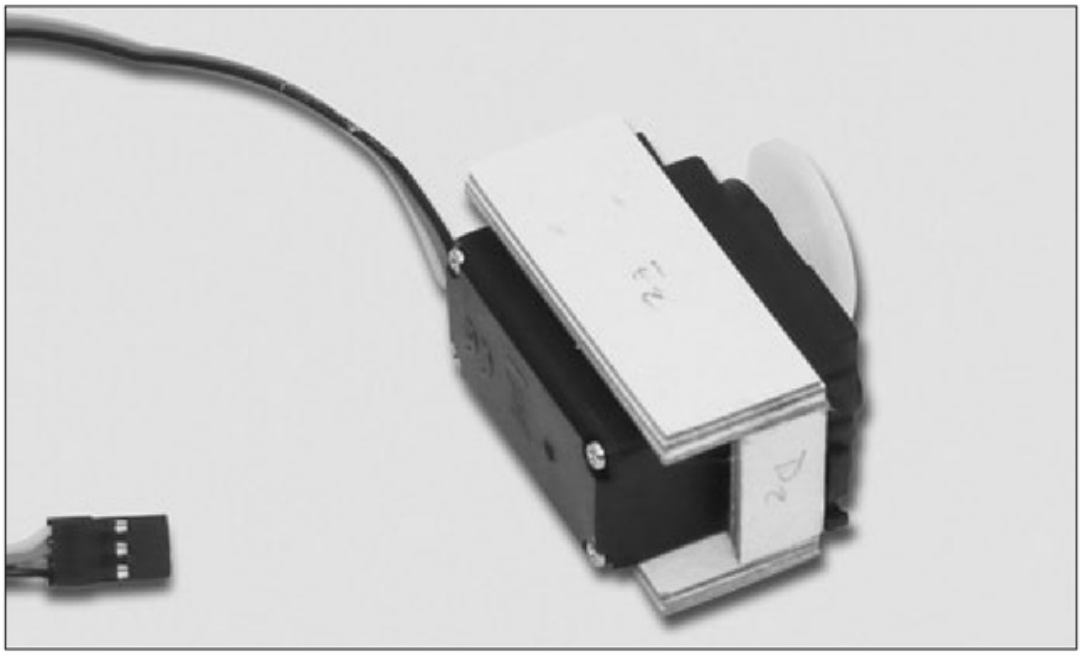


Figura 20. *Con los cuatro componentes, la caja del servo posee una estructura muy fuerte.*

Esta caja, que es un soporte para el servo, tiene que instalarse en la cara del robot animatrónico. Necesitamos poseer una buena área de pegado para evitar problemas, en especial cuando el servo se encuentre en movimiento. Para esto, usaremos los componentes **F**. Éstos no brindan únicamente una mayor área de pegado, sino que también permiten sostener la caja de forma lateral y no sólo frontal. Paramos la caja y, en uno de sus lados hasta la parte inferior, pegamos uno de los componentes **F**. Es importante aplicar suficiente pegamento para cubrir toda la cara de **F**.

Figura 21. *El componente* **F** *es pegado en el costado de la caja. Debe ser pegado de manera firme.*

Ahora, debemos ubicar el otro componente **F** en el extremo restante de la caja y, también, tendremos que pegarlo bien firme. Con esto, la caja está completa y podemos proceder a colocarla en la cara del robot.

Figura 22. *La caja ahora tiene dos soportes adicionales para colocarla en la cara.*

Tomamos el cartón que representa la cara y, sobre él, pegamos el soporte del servo que acabamos de crear. El soporte debe estar centrado y a aproximadamente 5 milímetros del borde inferior. Aplicamos pegamento en los componentes **F** y el componente **D** para proveer una buena área de contacto con la cara.

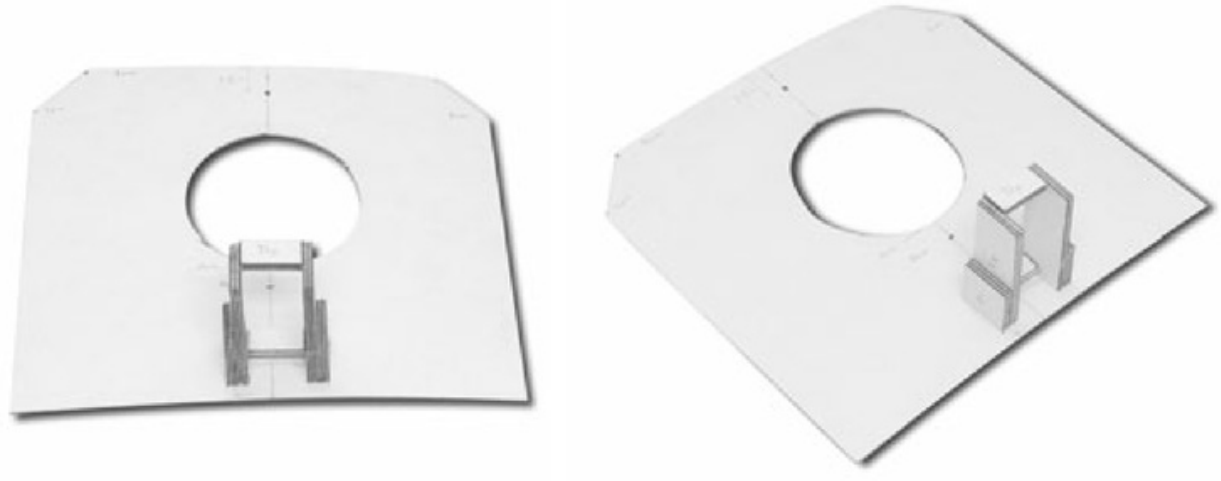


Figura 23. *Estas fotografías muestran la posición y la forma en que el soporte del servo debe pegarse a la cara. Ambas vistas nos permiten ver la forma como debemos pegar el soporte del servo.*

Creación de la ceja

Con el ojo listo, procedemos a construir la ceja. La ceja es controlada por un servo y realizará un movimiento de vaivén. Al cambiar la posición del servo, el ángulo de la ceja se modificará y, de esta forma, podrá mostrarnos diferentes expresiones. El servo necesita también de una caja que lo sostenga, y ésta se colocará en la cara. Aprovechando que estamos trabajando con el cartón de la cara, procedemos a hacer un orificio en él. Este orificio debe estar centrado en la marca que se encuentra en el centro, cerca del borde superior. Por esta abertura sale el eje del servo, el cual otorga el movimiento a la ceja. El orificio debe medir, aproximadamente, 12 milímetros de diámetro. En caso de que fuera necesario, podemos expandirlo, con el fin de facilitar el movimiento del servo.

ROBOTS DIVERTIDOS

En el sitio web *Android World* (**www.androidworld.com**), encontramos información sobre diversos tipos de robots, incluidos los animatrónicos. En él, se ofrecen secciones en las cuales se muestran distintos componentes para crear este tipo de robots. También podremos acceder a una sección con datos sobre conferencias.

Figura 24. *El orificio debe quedar centrado en la marca que hicimos antes para permitir que pase el eje del servo.*

Tomamos el componente **G** y procedemos a realizar algunos trazos en él. Justo en el centro, trazamos una línea vertical. A partir del borde superior, hacemos una marca a 2 centímetros de distancia y pasamos por ella una línea horizontal. También, desde el borde inferior, hacemos una marca a 2 centímetros de distancia y, otra vez, trazamos una línea horizontal.

Figura 25. *Estas marcas sobre el componente nos ayudarán a poder trazar la forma de la ceja.*

{ } CEJAS MÁS INTERESANTES

Podemos diseñar cejas de diferentes tipos e, incluso, usar distintos materiales para hacerlas lucir más reales. En todos los casos, no debemos olvidarnos de dejar un área con suficiente espacio para los tornillos. Incluso, podemos tener toda una colección de cejas intercambiables para usar con nuestro robot animatrónico.

Tomamos un lápiz y delineamos con él la forma de una ceja. Las líneas que hemos trazado nos pueden ayudar a guiarnos. Es importante que la ceja no sea demasiado delgada, ya que necesita suficiente espacio para los tornillos que unen a la ceja con el mecanismo de actuación del servo.

Figura 26. *La ceja queda dibujada en el componente. Es necesario contar con un área que tenga 2 centímetros de ancho, como mínimo, para los tornillos.*

Ahora que tenemos el trazo de la ceja, procedemos a recortarla. Hay que tener cuidado al hacerlo para no doblar el cartón.

Figura 27. *Recortamos el componente* ***G*** *siguiendo el dibujo de la ceja.*

Debemos unir la ceja con el servo por medio del mecanismo de actuación. De forma similar a lo que hemos realizado con otros proyectos, ubicamos el mecanismo de actuación sobre la ceja y marcamos los lugares por donde deben ingresar los tornillos.

Figura 28. *Señalamos el lugar donde se ubicarán los tornillos para la ceja.*

Después de marcar los puntos, realizamos dos perforaciones para facilitar la instalación de los tornillos. Debemos cuidar que estas perforaciones sean de un diámetro menor o igual al de los tornillos.

Figura 29. *Aquí podemos observar las perforaciones que nos servirán de guía para los tornillos.*

Usando pintura de color negro, pintamos toda la superficie de la ceja. Para dar un mejor acabado, es conveniente pintar ambos lados de la ceja y, también, todo su contorno. Podemos utilizar otro color para la ceja si lo deseamos.

Figura 30. *La ceja ha sido pintada y está lista para ser montada.*

Con la ceja preparada, entonces, podemos empezar a trabajar con el soporte para el servo de la ceja. Para crear este soporte, usaremos los componentes **H** e **I**. La creación de esta caja es más sencilla que la de la anterior.

Figura 31. *Éstos son los componentes necesarios para hacer el soporte del servo de la ceja.*

Ubicamos el servo sobre el cartón de la cara. El eje del servo debe pasar por el orificio que creamos al inicio de esta sección. El servo está alineado con el borde superior de la cara. Entonces, pegamos el componente **H** al cartón de la cara. Podemos aprovechar el servo para sostener el componente mientras el pegamento se seca.

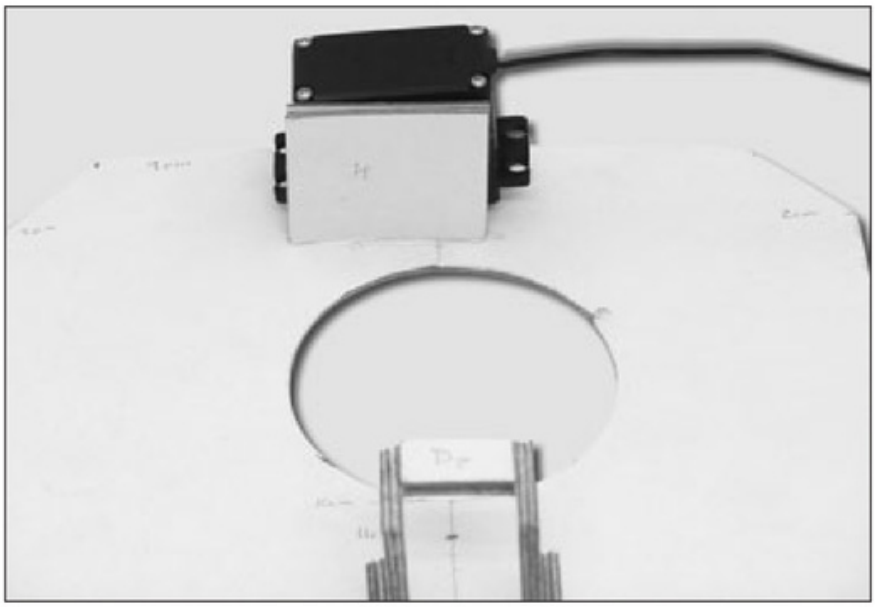


Figura 32. *El servo es ubicado en posición, y el componente* ***H*** *se coloca a su lado.*

En el otro lado del servo, ubicamos el otro componente **H**. Los componentes **H** deben pegarse únicamente al cartón de la cara y no al servo.

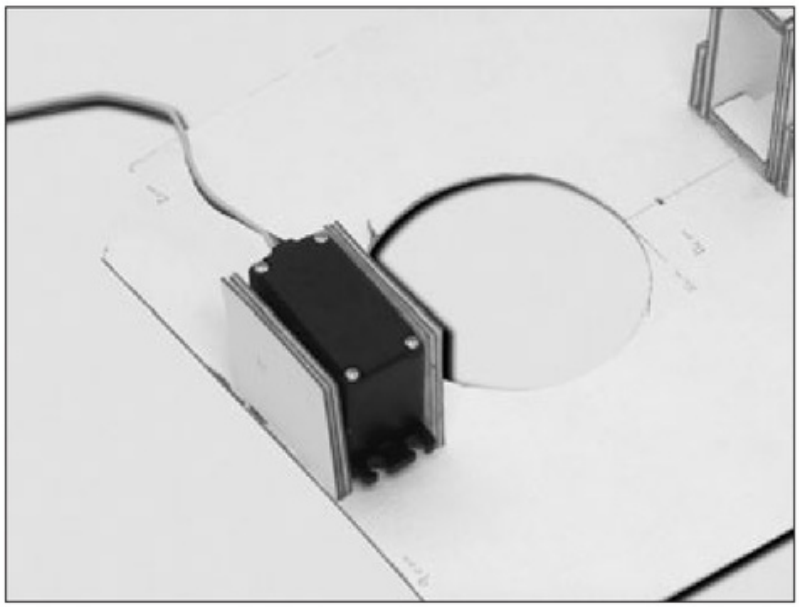


Figura 33. *El servo queda resguardado por los dos componentes* ***H*** *que luego lo sostendrán.*

Para finalizar el soporte de este servo y darle mayor firmeza estructural, pegamos el componente **I** en el lado derecho del soporte. Es importante que se pegue en la parte superior, ya que el estribo del servo pasará debajo de él.

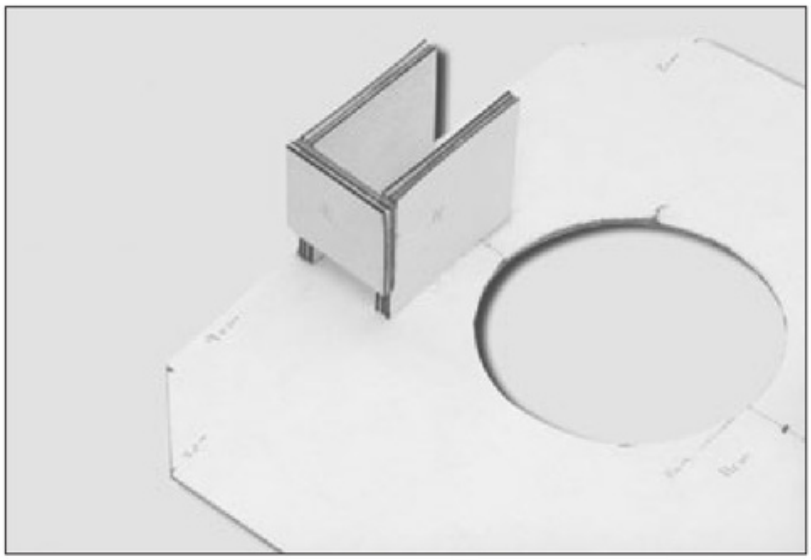


Figura 34. *El componente* ***I*** *completa el soporte del servo. Es importante observar la posición en la que debe pegarse.*

Creación de la mandíbula

Otro elemento que tendrá movimiento en el robot animatrónico es la mandíbula. Un servo proveerá el sencillo movimiento circular que hará que la mandíbula se abra o se cierre. En este caso, es mejor empezar instalando el soporte para el servo y, luego, colocar la mandíbula. De esta forma, podremos centrarla con mayor facilidad. Empezamos usando el componente **J**. Este componente sirve como asiento para el servo que utilizaremos. Para ubicarlo en posición, lo pegamos sobre el cartón de la cara. Hacemos una marca a 18 milímetros del borde izquierdo del cartón. Ubicamos el componente **J** de tal forma que su extremo izquierdo se encuentre a la altura de esa marca. El componente debe quedar a tres milímetros del borde inferior del cartón de la cara y tendrá que estar pegado firmemente.

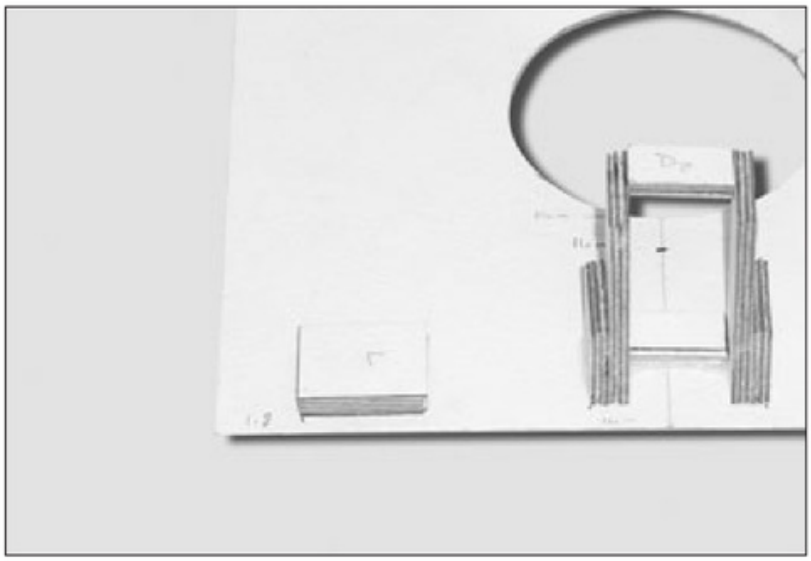


Figura 35. *Esta fotografía nos muestra la posición exacta que debe tener el componente J.*

A continuación, sentamos el servo sobre el componente **J**. Tomamos uno de los componentes **L** y lo pegamos al cartón de la cara. Usamos el servo para ayudar a sostener el componente **J** mientras se seca el pegamento.

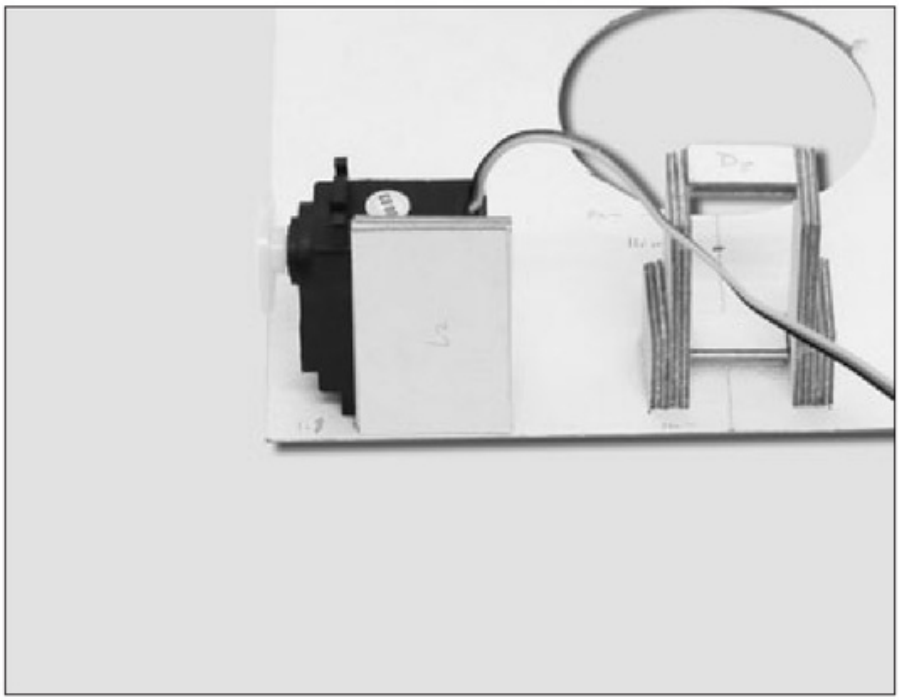


Figura 36. *El componente **L** es una de las caras para la caja que sostiene al servo.*

En el otro lado del servo, ubicaremos el siguiente componente **L**. Éste también debe pegarse al cartón de la cara. El servo, nuevamente, nos ayuda a sostenerlo mientras se pega. Es conveniente que los componentes **L** queden lo más ajustados posible al servo para que así puedan sostenerlo de manera firme.

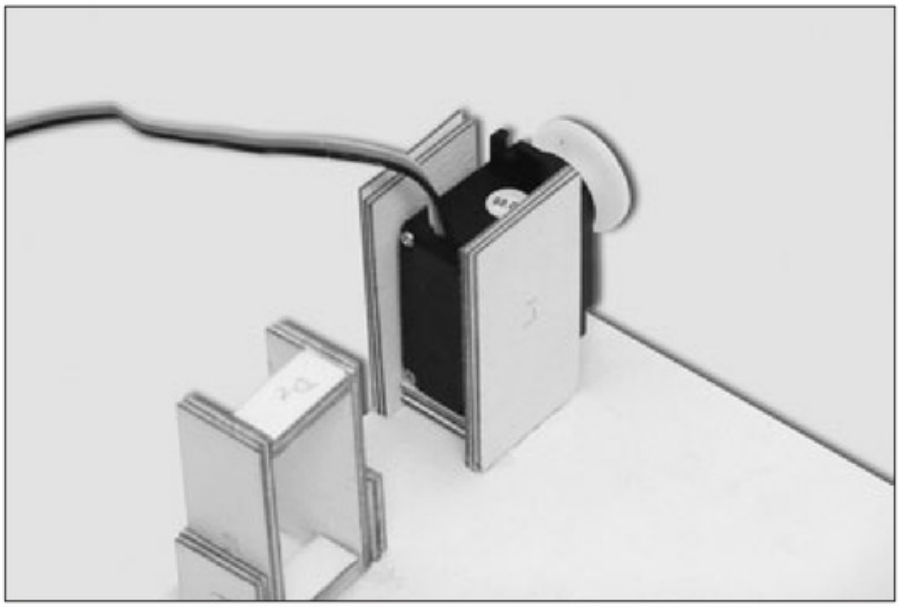


Figura 37. *Aquí podemos observar la posición que deben tener los componentes **L** alrededor del servo.*

Terminamos la caja al instalar el componente **K**. Este componente se pega entre los dos componentes **K**. Con esto, damos un soporte extra al servo y, también, hacemos que la caja sea más resistente y estable. La posición del componente **K** no debe obstruir la salida del cable del servo.

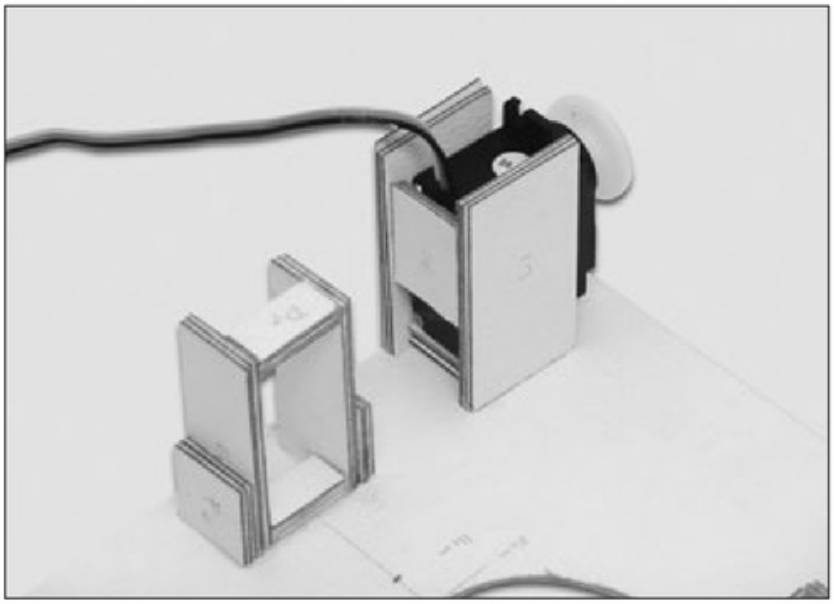


***Figura 38.** Hemos completado la caja que sostiene al servo al pegar el componente **K** en la parte posterior.*

La mandíbula necesita dos soportes laterales; uno de ellos se usará para unir la mandíbula con el servo, y el otro sólo ayudará al movimiento de la articulación. Estas piezas las creamos con los componentes **M**. El lado del componente **M** que mide 4.5 centímetros es el que utilizaremos como base. En este componente tendremos que hacer trazados que nos ayudarán a darle la forma correcta al soporte lateral. Desde el borde derecho, colocamos una marca a **2** centímetros de distancia y, en la posición de la marca, trazamos una línea vertical. De la misma forma, dibujamos una marca a **2** centímetros de distancia del borde derecho y, también, trazamos una línea vertical.

Además necesitamos trazar algunas líneas horizontales. Desde el borde superior, medimos dos centímetros y dibujamos una marca, luego, sobre esa marca, trazamos

INFORMACIÓN SOBRE ANIMATRÓNICOS

El sitio web **Animatronics** presenta variada información sobre recursos y empresas de robots animatrónicos. Muchas de estas empresas crean robots profesionales para parques de diversiones temáticos. El sitio también tiene una colección de videos. Para ingresar en el sitio web nos dirigimos a **www.animatronics.org**.

una línea horizontal. Tomando como referencia el borde inferior, del lado derecho del componente, marcamos otra vez, pero en esta ocasión a un centímetro de distancia, y trazamos una línea horizontal hasta la primera vertical que encontremos. Todo esto lo podemos ver en detalle en la siguiente figura.

Figura 39. *Esta figura nos muestra la forma como han quedado los trazos que nos servirán de guía.*

Con estas líneas guías resulta fácil dibujar la forma que necesitamos. Con un lápiz grueso o un marcador, trazamos unas líneas como se muestra en la siguiente figura. Con esto, tenemos la forma que necesitamos para el soporte lateral de la mandíbula.

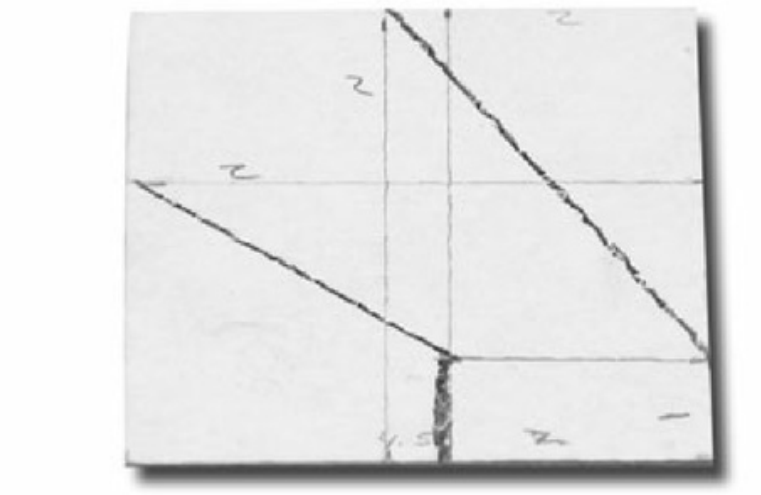


Figura 40*. Con el marcador, podemos resaltar la forma del soporte lateral. Usamos las guías anteriores para hacerlo mejor.*

Recortamos el cartón siguiendo las líneas que se trazaron con el marcador y debemos tener cuidado de no doblar en forma accidental el cartón. De esta manera, tenemos el primer sostén para la mandíbula.

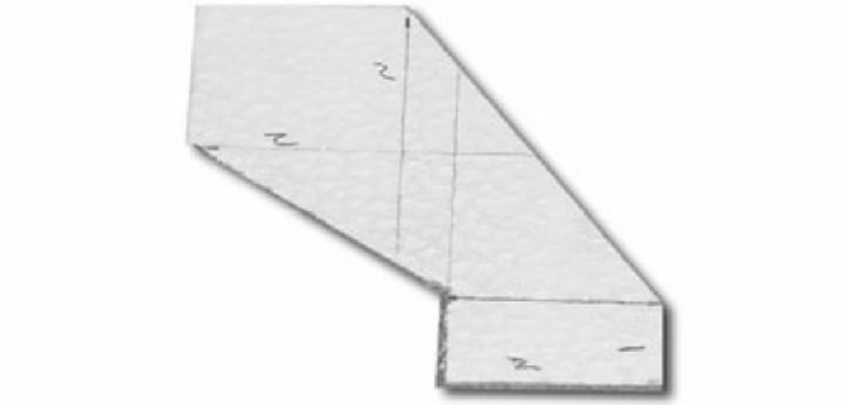


Figura 41. *El soporte lateral ha sido recortado y está casi listo para utilizarse.*

El primer sostén será usado para unir la mandíbula con el servo que la moverá. Por esto, es necesario hacer las marcas por donde pasarán los tornillos que colocaremos. Para hacerlo, nuevamente usaremos el elemento de actuación como guía y, con un lápiz delgado, establecemos las marcas para los tornillos.

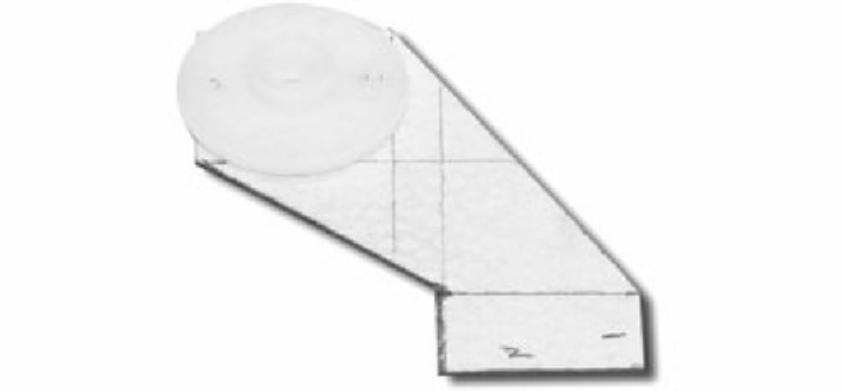


Figura 42. *Con el elemento de actuación del servo como guía, llevamos a cabo las marcas para los tornillos.*

Con cuidado, hacemos las perforaciones por las que pasarán los tornillos. Estas perforaciones sólo funcionan como una guía y no deben ser muy anchas.

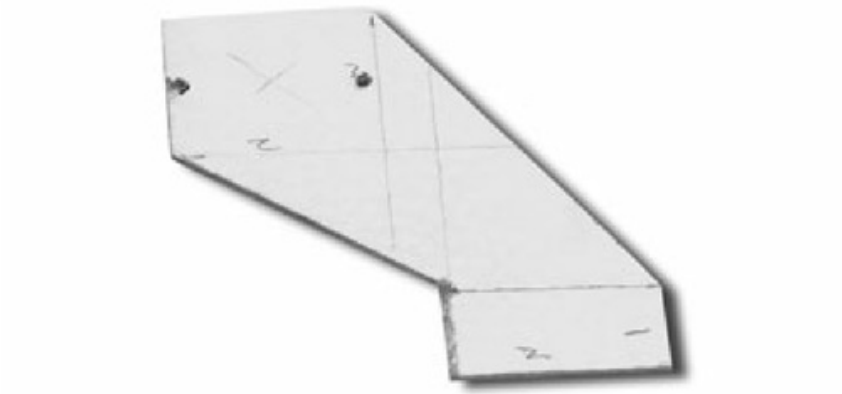


Figura 43. *Hacemos unas pequeñas perforaciones que sirvan de guía para los tornillos.*

Tomando como modelo el soporte lateral que hemos creado antes, recortamos uno más. El nuevo soporte será usado para el otro lado de la mandíbula. Debemos tener en cuenta que este soporte no lleva las perforaciones para los tornillos del servo.

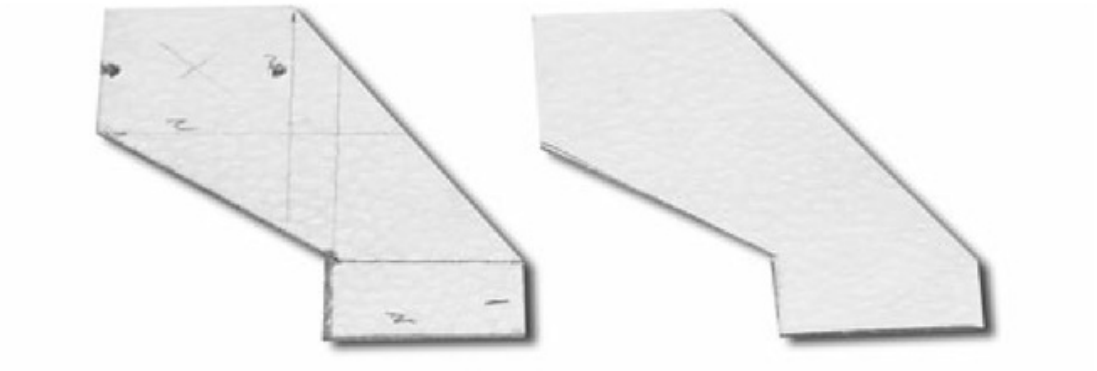


Figura 44. *Hacemos otro soporte con las mismas dimensiones, pero sin las perforaciones.*

Ahora, empecemos a formar la mandíbula. Necesitamos el componente **N**, el soporte con las perforaciones de los tornillos y la mandíbula. Cuando armamos la mandíbula, hay que tener especial cuidado con la orientación de los componentes, de otra forma, ésta no podría girar de manera correcta.

Figura 45. *Éstos son los componentes necesarios para iniciar el ensamble de la mandíbula.*

En la parte posterior del soporte que tiene las perforaciones, debemos pegar el componente **N**. Observemos con atención la siguiente figura para hacerlo en forma correcta.

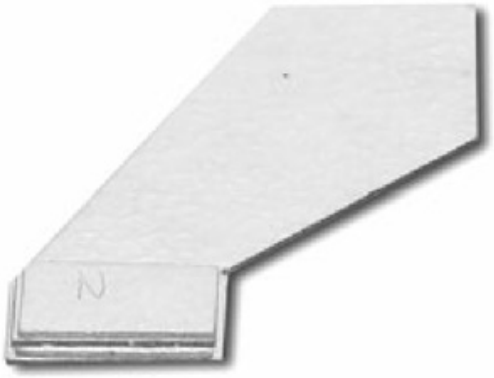


Figura 46. *Es importante notar cómo debemos pegar el componente* ***N*** *sobre el soporte con las perforaciones de los tornillos.*

Tomamos el soporte con el componente **N** pegado y, a su vez, lo pegamos sobre la mandíbula en el extremo izquierdo. El soporte debe quedar vertical. Podemos utilizar algún objeto para ayudarnos a estabilizarlo mientras se termina de pegar. El componente **N** estará en el interior de la mandíbula. Debemos utilizar suficiente pegamento para hacer una unión fuerte y firme.

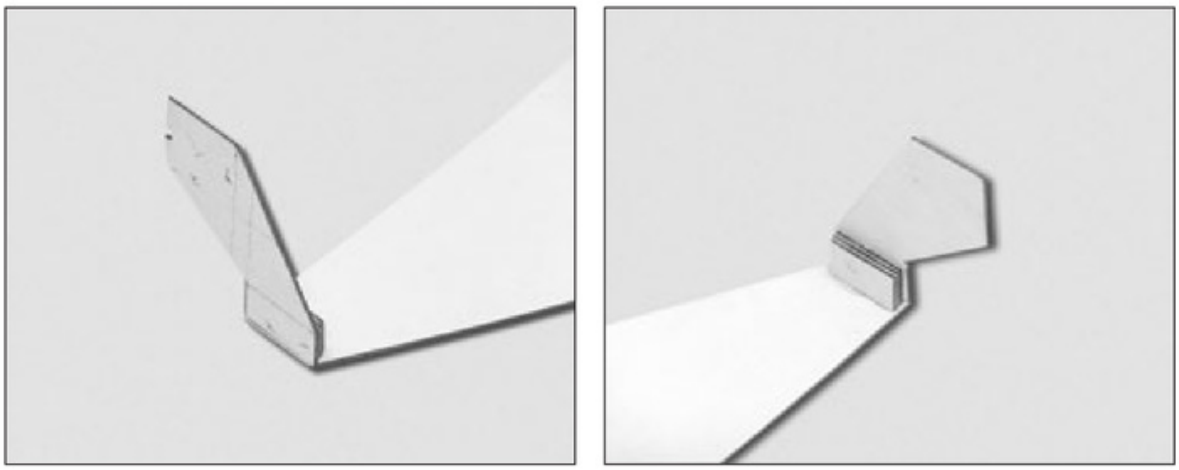


Figura 47. *En ambas fotografías, podemos observar la forma en la que debemos pegar el soporte con las perforaciones de los tornillos sobre la mandíbula.*

Para ubicar el soporte del otro lado de la mandíbula, necesitamos otro componente **N** y el segundo soporte que construimos antes. Este soporte no tiene perforaciones.

Figura 48. *Estos componentes son necesarios para instalar el segundo soporte de la mandíbula.*

El siguiente soporte también necesita de un componente **N** para poder ser instalado correctamente en la mandíbula. Cuando colocamos este elemento, debe hacerse en la posición correcta para que todo termine montado de una forma óptima.

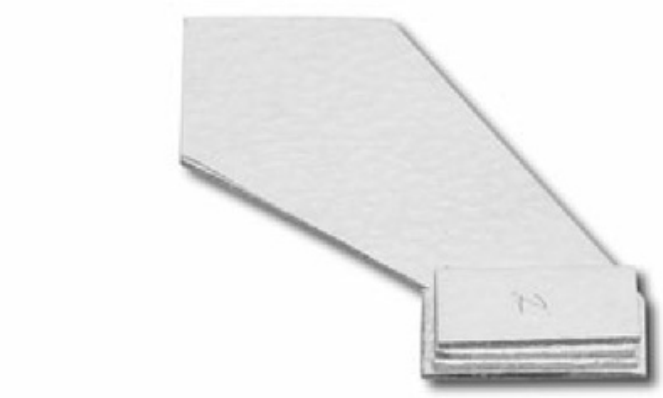


Figura 49. *Ésta es la posición en la que debe montarse el componente* ***N*** *sobre el otro soporte para la mandíbula.*

Cuando el componente **N** se encuentre pegado, podemos proceder a ubicar este soporte en el otro lado de la mandíbula. Debe pegarse hasta el extremo derecho y quedar firme; es conveniente que usemos una buena cantidad de pegamento para tal fin.

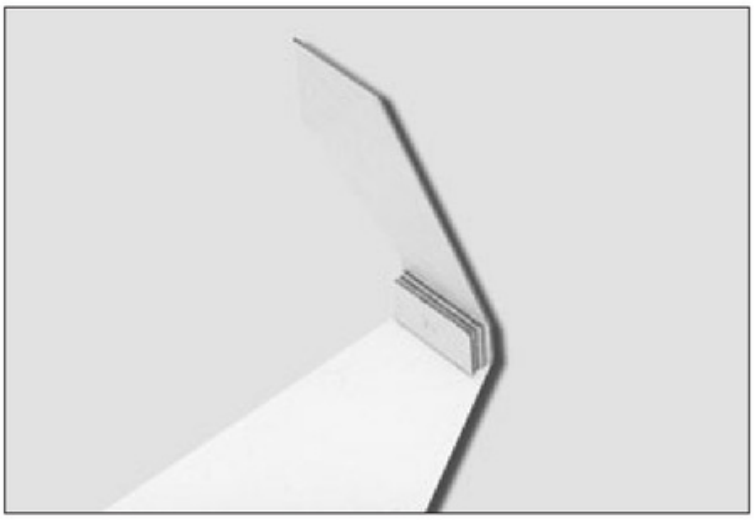


Figura 50*. Esta fotografía nos muestra la forma como el segundo soporte queda colocado en la mandíbula.*

Ahora, la mandíbula está completa, pero necesitamos construir un poco más, pues esta parte debe poder sostenerse y rotar en ambos lados.

III LOS AUDIO-ANIMATRÓNICOS

Los **Audio-animatrónicos** son una marca registrada de *Walt Disney Imagineering* y son utilizados, en especial, en atracciones turísticas y parques temáticos. Estos robots han representado desde presidentes de los Estados Unidos hasta dinosaurios de todos los tamaños imaginables.

Figura 51. *Nuestra mandíbula queda de esta forma con sus dos soportes ubicados en posición.*

El soporte del lado izquierdo estará sostenido por el servo, y el mismo eje del servo servirá como su eje de movimiento. Sin embargo, el soporte del lado derecho aún no tiene algo que lo sostenga y quedará colgando. Para corregir esto, creamos un pequeño sostén que también permita tener un eje. De esta forma, la mandíbula queda sostenida y puede moverse sin problemas. El soporte necesita de algunos componentes. Los componentes que usaremos serán el **O**, **P**, **Q** y **R**. También necesitaremos un clavo pequeño para las funciones de eje.

Figura 52. *Éstos son los componentes que necesitamos utilizar para sostener el lado derecho de la mandíbula.*

En primer lugar, en la parte superior del componente **O**, pegamos el componente **Q**, que debe quedar bien alineado, como vemos en la figura 53.

Figura 53. *Hemos pegado el componente* ***Q*** *sobre el componente* ***O****, que debe quedar en el extremo superior.*

A continuación, aplicamos pegamento sólo en el componente **Q** y pegamos el componente **R** sobre él. El componente **R** no debe quedar adherido al componente **O**.

Figura 54. *El componente* ***R*** *es pegado únicamente sobre el componente* ***Q****.*

El componente **P** también se pega sobre el componente **O**, pero, en este caso, lo hacemos en el extremo inferior y del otro lado del elemento. Luego, este componente nos ayudará a pegar este sostén en el cartón de la cara.

Figura 55. *Aquí vemos el sostén finalizado. El componente* ***P*** *en el extremo inferior.*

Llegamos al momento de colocar este sostén en el cartón de la cara del robot animatrónico. Ubicamos el sostén en el lado derecho del cartón, en su parte inferior. Usamos el componente **P** para pegar y colocar en posición el sostén.

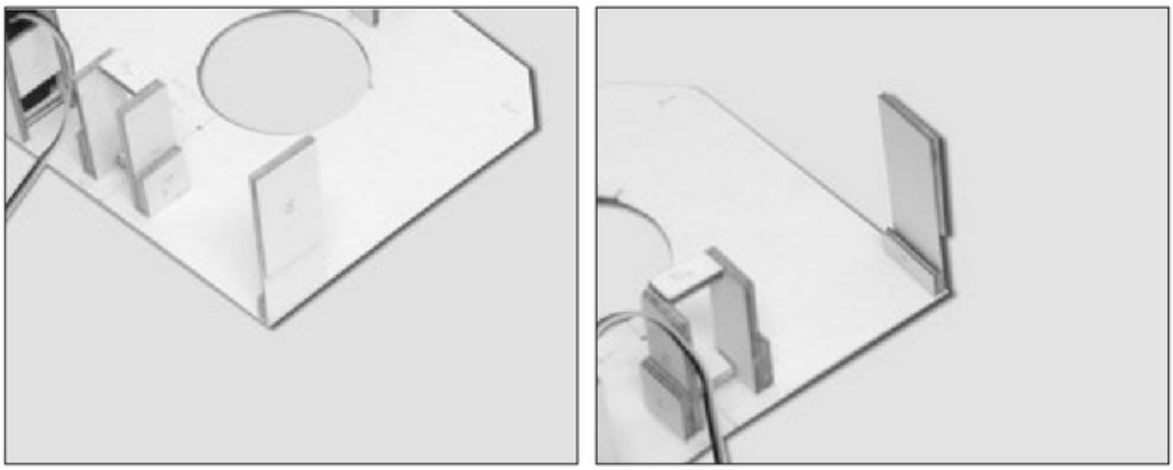


Figura 56. *Logramos ubicar el sostén en su posición. Es importante que se mantenga de pie y firme, como podemos observar en ambas imágenes.*

Este sostén debe tener una perforación por donde pasaremos el eje que ayuda al movimiento de la mandíbula. La mejor forma de hacer esto es realizar una perforación en el soporte de la mandíbula en la parte central del cuadro superior y, luego, a la misma altura, hacer una perforación en el sostén. De esta forma, garantizamos que el eje pasará por ambos orificios.

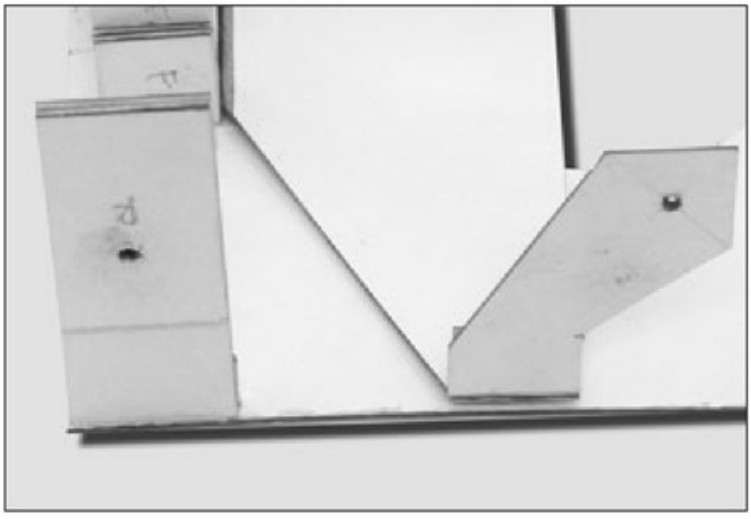


Figura 57. *Tanto el soporte de la mandíbula como el sostén han sido perforados. Ambas perforaciones se encuentran a la misma altura.*

El siguiente paso consiste en insertar el soporte dentro del sostén. Es decir, el soporte quedará entre los componentes **O** y **R** del sostén. Es necesario alinearlos para que, por la perforación, pueda pasar un clavo y funcionar como eje.

Figura 58. *El soporte queda dentro del sostén, y el clavo actúa como eje para su movimiento.*

Unión del robot con la base

El robot debe tener una base para que podamos observar el rostro y los movimientos que realizará. Podemos usar una de las bases de los proyectos anteriores o un nuevo bloque de madera que funcione como ella. La unión a la base es una especie de cuello. Debido al peso que debe cargar, es necesario que las piezas queden pegadas con firmeza, de esta manera evitaremos problemas con el robot. Para la base, necesitamos varios componentes. Los componentes son: **S**, **T**, **U** y **V**.

Figura 59. *Éstos son los componentes que necesitamos para la creación de la base del robot.*

VIDEOS DE ROBOTS ANIMATRÓNICOS

En la Web, podemos encontrar muchos ejemplos de robots animatrónicos profesionales. En **YouTube**, se ofrecen varias listas con ejemplos de estos robots producidos por diferentes empresas. Para encontrar los videos, ingresamos en **www.youtube.com** y realizamos una búsqueda con la palabra clave *animatronics*.

Tomamos los elementos **T** y, en su parte inferior, pegamos un elemento **U** a cada uno de ellos. Los elementos **T** formarán parte del cuello y ayudan a sostener la cara del robot.

Figura 60. *Cada elemento **T** tiene un elemento **U** pegado en su base*

Una vez que los elementos **U** están pegados, entonces, tomamos los demás elementos **U** y los pegamos en el mismo extremo del componente **T**, pero del otro lado.

Figura 61. *En el otro lado del componente **T**, pegamos otro componente **U**.*

Por un momento, dejamos a un lado los componentes **T** y tomamos uno de los componentes **S**. Pegamos el componente **S** en el lado del soporte para el servo del ojo. Quedará ubicado en la misma orientación que el componente **F**. Es importante que toda el área que tenga contacto con el soporte del servo lleve pegamento para que la unión sea fuerte y resistente.

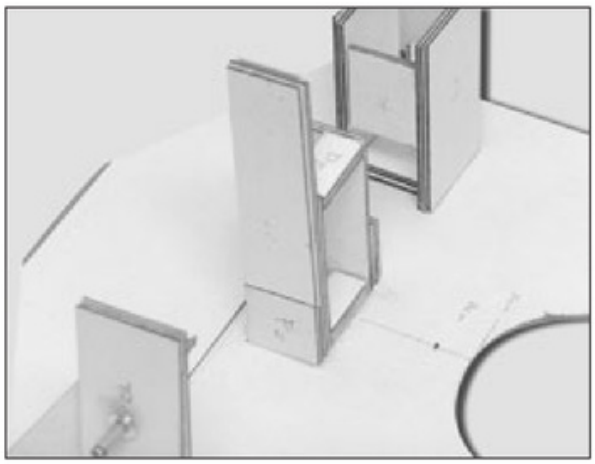


Figura 62. *Ésta es la posición del componente **S** pegado sobre el soporte del servo del ojo.*

Tomamos ahora el otro componente **S** y, de forma similar a como lo hicimos antes, lo pegamos en el otro lado del soporte para el servo del ojo. También, tenemos que colocar pegamento en toda el área de contacto para hacer la unión fuerte.

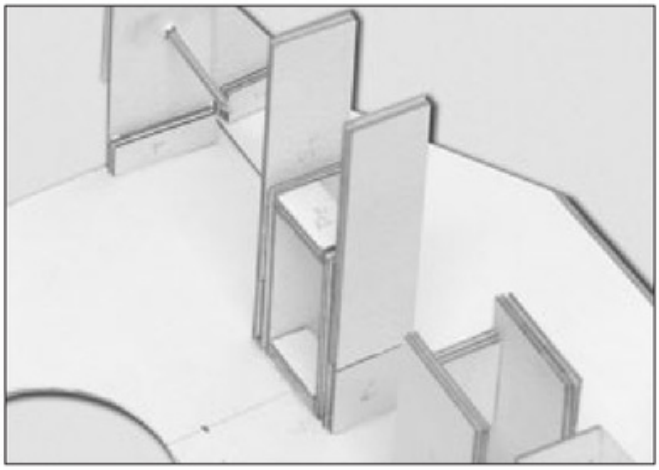


Figura 63. *Contamos con los elementos **S** instalados a cada lado del soporte del servo para el ojo.*

Cuando el pegamento de los componentes **S** se encuentre completamente seco, procedemos a pegar sobre ellos los elementos **T**. Los componentes **T** se pegan de forma ortogonal a los componentes **S**, para formar un ángulo de **90** grados. El extremo de los componentes **T** que tienen al componente **U** queda flotando con libertad.

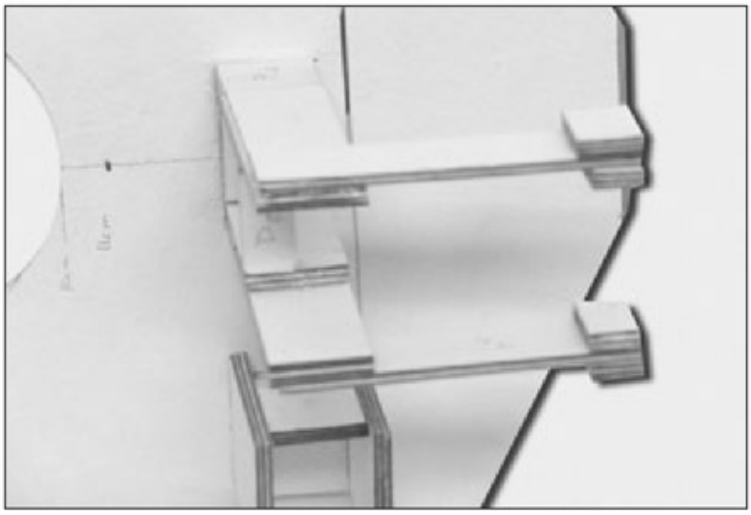


Figura 64. *En esta fotografía, podemos observar la forma como quedan montados los componentes **T**, en ángulo recto en relación con los componentes **S**.*

Cuando los elementos **T** se encuentren firmes, entonces, procedemos a pegarlos al componente **V**, el cual podemos ubicar sobre la base. Esta etapa de pegado es crítica, y necesitaremos sostener el rostro del robot mientras el pegamento seca. Los componentes **U** son usados como apoyo en el proceso de pegado y no debemos olvidarnos de utilizar una buena cantidad de adhesivo.

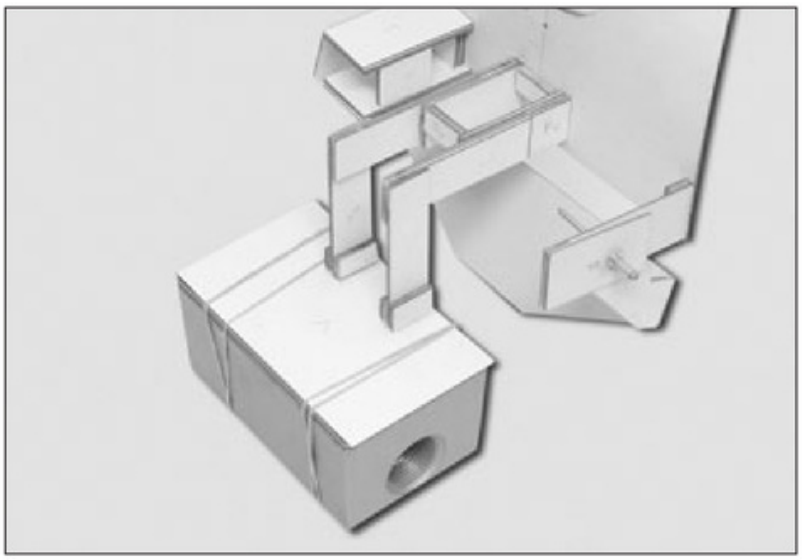


Figura 65. *Podemos observar el robot montado sobre la base, que debe sostenerse sin ningún tipo de problema.*

Ensamble final

En esta etapa, es una buena idea pintar la cara del robot animatrónico. La forma de hacerlo depende de nuestro gusto personal y de nuestra creatividad. Después de pintar el rostro del robot, podemos proceder a ensamblarlo. La mandíbula se encuentra colocada de un lado, pero aún falta sujetarla al servo que la moverá. El servo debe estar colocado en su posición intermedia. Esto nos permite darle un rango de movimiento adecuado. Aun así, la mandíbula no rotará mucho. Una vez que el servo se encuentre en su posición intermedia debemos instalarlo en el soporte del servo de la mandíbula. Con cuidado, ponemos la mandíbula en posición y colocamos los tornillos para sujetarla con firmeza al elemento de actuación del servo.

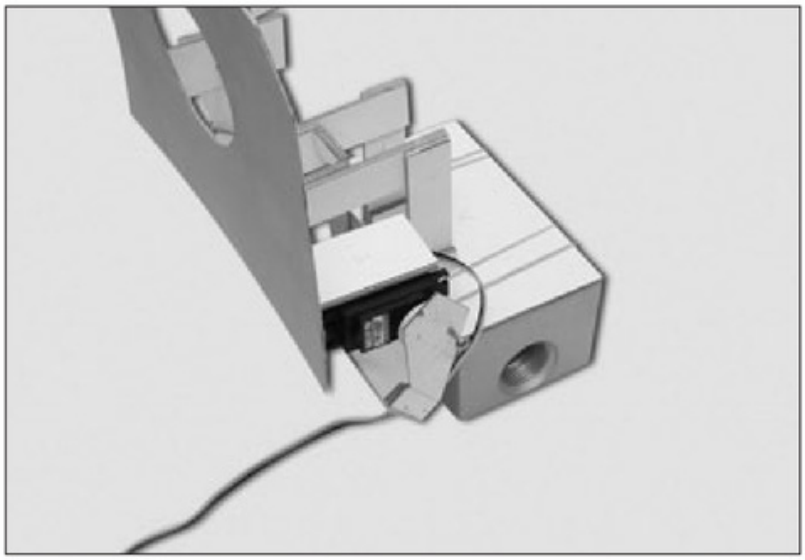


Figura 66. *Podemos ver la forma de ensamblar la mandíbula al servo. El primer tornillo está en su lugar.*

A continuación, instalamos el servo que mueve al ojo. Este servo ya tiene ensamblado encima el soporte con el clavo. El servo, también, debe estar en posición centrada. Con cuidado, incrustamos el ojo en el clavo. El ojo debe quedar bien centrado y lo más recto posible. Luego, ponemos el servo en su posición dentro de la cara.

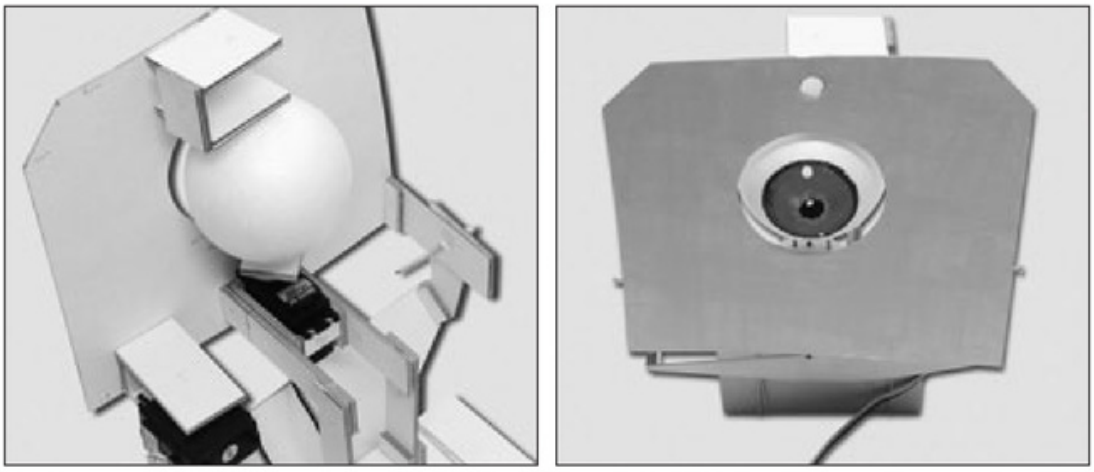


Figura 67. *Como podemos ver, el servo con el ojo ha quedado en posición. La esfera del ojo debe estar clavada al sostén en el servo.*

La última parte que nos queda por ensamblar es la ceja. En este caso, también, usamos un servo que debemos ubicar en posición centrada y al que, primero, instalaremos en su soporte correspondiente.

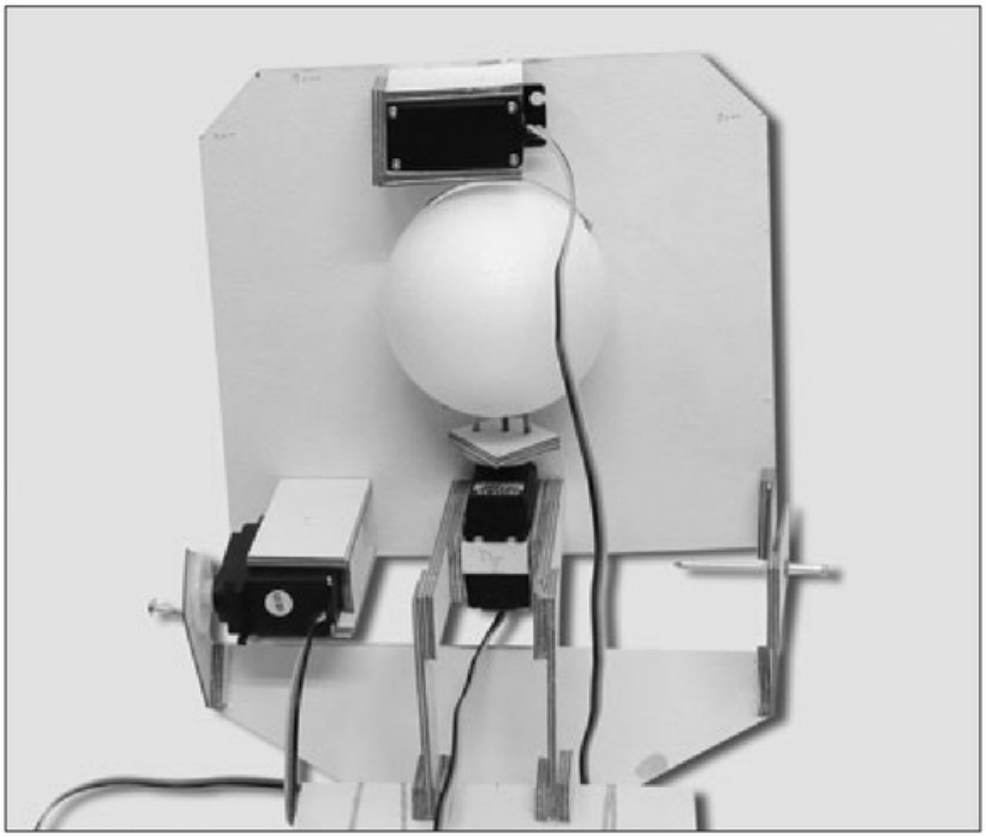


Figura 68. *El servo que mueve a la ceja es ubicado en posición.*

En el lado de la cara, ponemos el elemento de actuación en el servo y lo atornillamos. El elemento de actuación debe poder girar con libertad. En caso de que el orificio no sea lo suficientemente ancho, éste es el momento indicado para corregirlo.

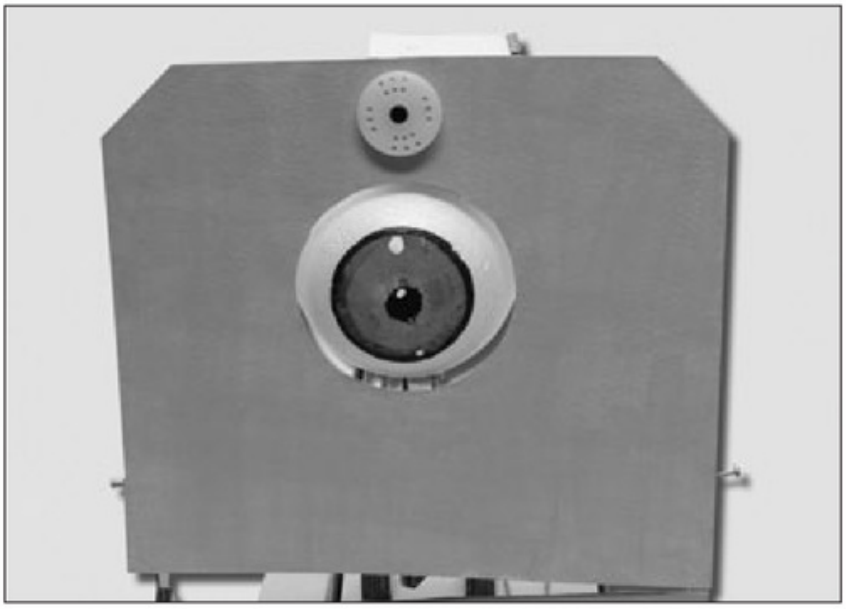


Figura 69. *El elemento de actuación es instalado en el servo que mueve a la ceja.*

Por último, montamos la ceja atornillándola al elemento de actuación. Debemos tener cuidado de no lastimar la pintura en el momento de atornillar. Con esto, hemos finalizado el montaje del robot; ahora, sólo falta conectar los servos al Phidget. El servo del ojo se conecta en la posición **0**; el servo de la ceja, en la posición **1**; y el de la mandíbula, en la posición **2**.

Figura 70. *La ceja se ha instalado y hemos finalizado el montaje.*

CREACIÓN DEL SOFTWARE

El siguiente paso es la creación del software que controlará al robot animatrónico. El software es muy sencillo y usará muchos de los elementos que ya conocemos. La única diferencia es que, en este proyecto, utilizaremos tres servos al mismo tiempo. Esto no presenta dificultad, ya que todas las técnicas de programación que conocemos para los servos las usaremos en el tercero. La interfaz del software va a consistir en tres secciones. Una sección es para la ceja; otra, para el ojo; y la última, para la boca. En cada una de estas secciones, tenemos una serie de **TrackBars**; cada uno de ellos simboliza la posición del servo. Al hacer clic en el botón correspondiente, el robot realizará la animación siguiendo las posiciones marcadas en los **TrackBars**.

Creación de la interfaz de usuario

Empecemos por crear la interfaz de usuario. Para esto, comenzamos con una copia limpia de nuestra aplicación base y adicionamos los controles necesarios tal y como lo vemos en la siguiente figura.

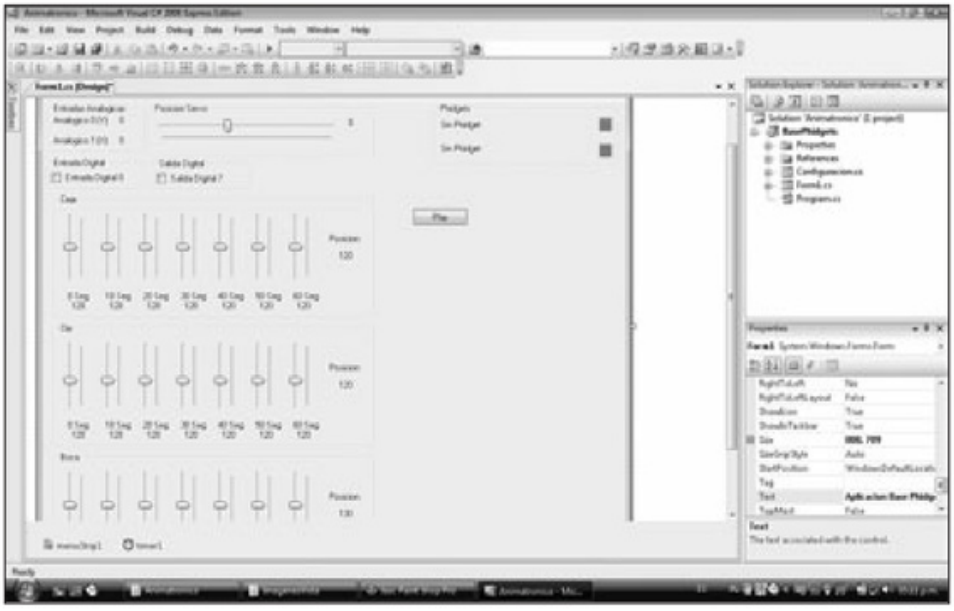


Figura 71. *Esta figura nos muestra la forma como debemos crear nuestra interfaz de usuario para el robot animatrónico.*

{} COLOCAR LA ORIENTACIÓN DE LOS TRACKBARS

El **TrackBar** se puede encontrar orientado de forma horizontal o vertical. El default de este control es la orientación horizontal, pero, por medio de la propiedad **Orientation**, podemos modificarlo. El comportamiento es exactamente el mismo sin importar la orientación que tenga.

La creación de las secciones de la ceja, del ojo y de la boca es muy similar; apenas cambian los nombres correspondientes. Empezamos por crear un **GroupBox** al cual escribimos **Ceja** en su propiedad **Text**. Dentro del **GroupBox**, insertamos siete **TrackBars**. Debemos ingresar en cada uno de ellos el valor **Vertical** en la propiedad **Orientation**.

Los **TrackBars** para la ceja deben tener el valor **95** en su propiedad **Minimum** y **145** en la propiedad **Maximum**. Esto nos permite fijar el límite y evitar que la ceja se exceda posteriormente en su movimiento. Inicializamos la propiedad **Value** de cada uno de ellos a **120** para dejarlo en un valor similar al del centro del servo. Los nombres que asignamos a los **TrackBars** son: **tbCeja0**, **tbCeja10**, **tbCeja20**, **tbCeja30**, **tbCeja40**, **tbCeja50**, **tbCeja60**. Esto lo hacemos en el orden correspondiente de izquierda a derecha, según aparecen en la forma. En la parte inferior, insertamos una serie de etiquetas que nos servirán para saber el **TrackBar** que corresponde a determinado tiempo. Los valores de las etiquetas son: **0 Seg**, **10 Seg**, **20 Seg**, **30 Seg**, **40 Seg**, **50 Seg**, **60 Seg**.

Debajo de estas etiquetas, agregaremos otras que nos indicarán el valor del **TrackBar**. Esto permitirá al usuario poder ver con precisión el valor que está ingresando. En principio, ponemos en su propiedad de **Text** el valor **120**. Los nombres de estas etiquetas son: **lblCeja0**, **lblCeja10**, **lblCeja20**, **lblCeja30**, **lblCeja40**, **lblCeja50**, **lblCeja60**. En el lado derecho, ubicamos una etiqueta que dice **Posición** y, debajo, insertamos otra etiqueta con el valor **120** en su propiedad **Text**. El nombre de la segunda etiqueta es **lblPosicionCeja**. Esta etiqueta nos muestra la posición actual de la ceja durante la animación. De manera similar, realizamos el proceso correspondiente al ojo. En el caso del ojo, los **TrackBars** llevan los nombres: **tb0jo0**, **tb0jo10**, **tb0jo20**, **tb0jo30**, **tb0jo40**, **tb0jo50**, **tb0jo60**. Los nombres que escribimos en las etiquetas para establecer el valor de cada uno de los **TrackBars** son: **lbl0jo0**, **lbl0jo10**, **lbl0jo20**, **lbl0jo30**, **lbl0jo40**, **lbl0jo50**, **lbl0jo60**. Y, por último, para la etiqueta que nos muestra la posición del ojo durante la animación, ingresamos el nombre **lblPosición0jo**.

La última sección es para la boca y se crea de la misma forma que las dos anteriores, aunque variando los nombres de sus controles. Los **TrackBars** para la boca llevan los siguientes nombres: **tbBoca0**, **tbBoca10**, **tbBoca20**, **tbBoca30**, **tbBoca40**, **tbBoca50**, **tbBoca60**. En las etiquetas para el valor de cada **TrackBar**, establecemos los siguientes nombres: **lblBoca0**, **lblBoca10**, **lblBoca20**, **lblBoca30**, **lblBoca40**, **lblBoca50**, **lblBoca60**. La etiqueta para la posición del servo durante la animación lleva el nombre de **lblPosicionBoca**. La sección de mayor complejidad de la interfaz está lista y sólo debemos adicionar un botón que nos permita iniciar la animación. El botón lleva el nombre de **btnPlay** y, en su propiedad **Text**, establecemos el valor **Animación**. Para finalizar la forma, agregamos un **Timer**. El **Timer** deberá tener un valor de **10000** en su propiedad **Interval**. Esto último nos brinda diez segundos entre cada tic del **Timer**.

Creación del código de la aplicación

El código tiene elementos similares a los proyectos anteriores, por lo que sólo nos concentraremos en los cambios y en las adiciones. Empezamos por ingresar las variables necesarias en la clase de la forma.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

// Necesario para el registro
using Microsoft.Win32;

// Necesario para los Phidgets
using Phidgets;
using Phidgets.Events;

using System.Threading;

namespace BasePhidgets
{
    public partial class Form1 : Form
    {
        // Datos necesarios para la aplicacion base

        private int numeroSerie0;       // Numero de serie del phidget 0
        private int numeroSerie1;       // Numero de serie del phidget 1
        private bool conectado0;        // Para saber si hay conexion con
el phidget 0
        private bool conectado1;        // Para saber si hay conexion con
el phidget 1
        private InterfaceKit iKit;      // Kit de interfaz
        private AdvancedServo mServo;  // Controlador de servos
        private int[] ceja = new int[7];    // Valores de posicion para la
ceja
        private int[] ojo = new int[7];   // Valores de posicion para el
```

```
ojo
        private int[] boca = new int[7];     // Valores de posicion para la
boca
        private int indice = 0;           // Indice de la posicion
actual
```

En el código, podemos observar que tenemos tres arreglos distintos de tipo **int**, que se llaman **ceja**, **ojo** y **boca**. En estos arreglos guardaremos las posiciones de los **TrackBars** con el fin de facilitar la programación de la animación. Los arreglos tienen una longitud de **7**. Este número surge a partir de que son siete los **TrackBars** para cada servo. Una variable de tipo **int** llamada **índice** nos ayudará a saber cuál es el índice actual durante la animación.

El constructor de la forma, el handler para salir de la aplicación y el handler para la configuración no presentan ningún cambio importante.

```
        public Form1()
         {
             InitializeComponent();

             // Inicializamos la aplicacion

             // Empezamos como si no estuvieramos conectados
             conectado0 = false;
             conectado1 = false;

             // Colocamos un valor de default para el numero de serie
             numeroSerie0 = 0;
             numeroSerie1 = 0;

             // Verificamos si existe la llave en el registro
             RegistryKey sfwKey = Registry.LocalMachine.OpenSubKey
("Software");
             RegistryKey redKey = sfwKey.OpenSubKey("RedUsers2");

             // Verificamos si existe
             if (redKey == null) // la llave no existe
             {
```

```
                // Creamos la llave
                sfwKey = Registry.LocalMachine.OpenSubKey("Software", true);
                redKey = sfwKey.CreateSubKey("RedUsers2");

                RegistryKey robotKey0 = redKey.CreateSubKey("Robots2");


                // Colocamos un valor de default

                robotKey0.SetValue("NumeroSerie0", (object)numeroSerie0);
                robotKey0.SetValue("NumeroSerie1", (object)numeroSerie1);

            }
            else // La llave existe
            {
                RegistryKey robotKey0 = redKey.OpenSubKey("Robots2");

                // Obtenemos el numero de serie guardado
                numeroSerie0 = (int)robotKey0.GetValue("NumeroSerie0");
                numeroSerie1 = (int)robotKey0.GetValue("NumeroSerie1");
            }

        }

    private void salirToolStripMenuItem_Click(object sender, EventArgs e)
        {
            // Cerramos la aplicacion
            this.Close();
        }

        private void configurarToolStripMenuItem_Click(object sender,

EventArgs e)
        {
            // Este codigo es para el phidget 0

            Configuracion dlgConfig = new Configuracion();
            dlgConfig.NumeroSerie = numeroSerie0;

            if (dlgConfig.ShowDialog() == DialogResult.OK)
```

```
            {
                  // Escribimos el nuevo valor
                  numeroSerie0 = dlgConfig.NumeroSerie;

                  // Abrimos la llave
                  RegistryKey sfwKey = Registry.LocalMachine.OpenSubKey
("Software", true);
                  RegistryKey redKey = sfwKey.CreateSubKey("RedUsers2");

                  RegistryKey robotKey = redKey.CreateSubKey("Robots2");

                  // Colocamos el nuevo valor
                  robotKey.SetValue("NumeroSerie0", (object)numeroSerie0);

            }
         }
```

En el handler del evento **Load** de la forma, sí encontramos algunos cambios.

```
        private void Form1_Load(object sender, EventArgs e)
         {
              // Inicializamos el objeto InterfaceKit
              try
              {
                  // Codigo para el kit de interfaz

                  // Instanciamos el objeto
                  iKit = new InterfaceKit();

                  // Colocamos los handlers relacionados con la conexion
                  iKit.Attach += new AttachEventHandler(iKit_Conectar

Handler);

                  iKit.Detach += new DetachEventHandler(iKit_Desconectar
Handler);

                  // Colocamos el handler relacionado con el error
                  iKit.Error += new ErrorEventHandler(iKit_ErrorHandler);
```

```
                // Colocamos el handler para las entradas digitales
                iKit.InputChange += new InputChangeEventHandler
(iKit_DigitalHandle);

                // Colocamos el handler para las entradas analogicas
                iKit.SensorChange += new SensorChangeEventHandler
(iKit_AnalogoHandle);

                // Abrimos para conexiones
                iKit.open(numeroSerie0);

            } // Fin de try

            catch (PhidgetException ex)
            {
                // Enviamos mensaje con el error
                lblPhidget0.Text = ex.Description;

                // Indicamos con amarillo que hay problemas
                txtPhidget0.BackColor = System.Drawing.Color.Yellow;
            }


            try
            {
                // Codigo para el controlador de servos

                // Instanciamos el objeto
                mServo = new AdvancedServo();

                // Colocamos los handlers relacionados con la conexion
                mServo.Attach += new AttachEventHandler(mServo_Conectar

Handler);
                mServo.Detach += new DetachEventHandler

(mServo_DesconectarHandler);

                // Colocamos el handler relacionado con el error
                mServo.Error += new ErrorEventHandler(mServo_Error
```

```
Handler);

                // Abrimos para conexiones
                mServo.open(numeroSerie1);

                if (mServo.Attached == true)
                {

                    // Colocamos los limites de posicion del servo

                    // Ojo
                    mServo.servos[0].PositionMin = 95;
                    mServo.servos[0].PositionMax = 155;

                    // Ceja

                    mServo.servos[1].PositionMin = 95;
                    mServo.servos[1].PositionMax = 145;

                    // Boca
                    mServo.servos[2].PositionMin = 105;
                    mServo.servos[2].PositionMax = 155;
                }
            }
            catch (PhidgetException ex)
            {
                // Enviamos mensaje con el error
                lblPhidget1.Text = ex.Description;

                // Indicamos con amarillo que hay problemas
                txtPhidget1.BackColor = System.Drawing.Color.Yellow;
            }
        }
```

Observamos que el cambio principal aparece al ubicar los valores de posición máxima y mínima para cada uno de los servos. El rango de movimiento del ojo es de **95** a **155**, el de la ceja se encuentra entre **95** y **145** y, por último, el de la boca es de **105** a **155**.

Otros handlers que no sufren cambios son los que usamos para los eventos de la conexión, desconexión y error del Phidget 8/8/8.

```
// Handler de la conexion
public void iKit_ConectarHandler(object sender, AttachEventArgs e)
{
    // Indicamos que hay conexion
    conectado0 = true;

    // Enviamos mensaje de la conexion
    lblPhidget0.Text = e.Device.Name;

    // Indicamos con color verde que hay conexion
    txtPhidget0.BackColor = System.Drawing.Color.Lime;

}

// Handler de desconexion
public void iKit_DesconectarHandler(object sender, DetachEventArgs e)
{
    // Indicamos que hay desconexion
    conectado0 = false;

    // Quitamos mensaje
    lblPhidget0.Text = "Sin phidget";

    // Indicamos con color rojo que no hay conexion
    txtPhidget0.BackColor = System.Drawing.Color.Red;

}

// Handler del error
public void iKit_ErrorHandler(object sender, ErrorEventArgs e)
{
    // Indicamos que hay desconexion
    conectado0 = false;

    // Mandamos mensaje de error
    lblPhidget0.Text = e.Description;

    // Indicamos con color amarillo que hay problemas
    txtPhidget0.BackColor = System.Drawing.Color.Yellow;
}
```

El handler para el evento **ForClosing** sí presenta cambios, debido a que tenemos que incluir el código necesario para los tres servos.

```
      private void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {
            // Quitamos los handlers
            iKit.Attach -= new AttachEventHandler(iKit_ConectarHandler);
            iKit.Detach -= new DetachEventHandler(iKit_Desconectar
Handler);
            iKit.Error -= new ErrorEventHandler(iKit_ErrorHandler);

            iKit.InputChange -= new InputChangeEventHandler(iKit_Digital
Handle);
            iKit.SensorChange -= new SensorChangeEventHandler
(iKit_AnalogoHandle);

            // Regresamos los servos al centro
            if (mServo.Attached == true)
            {
                mServo.servos[0].VelocityLimit = 500;
                mServo.servos[0].Engaged = true;
                mServo.servos[0].Position = 120;

                mServo.servos[1].VelocityLimit = 500;
                mServo.servos[1].Engaged = true;
                mServo.servos[1].Position = 120;

                mServo.servos[2].VelocityLimit = 500;
                mServo.servos[2].Engaged = true;
                mServo.servos[2].Position = 120;

                Thread.Sleep(500);

                mServo.servos[0].Engaged = false;
                mServo.servos[1].Engaged = false;
                mServo.servos[2].Engaged = false;

                mServo.Attach -= new AttachEventHandler(mServo_Conectar
Handler);
                mServo.Detach -= new DetachEventHandler
```

```
(mServo_DesconectarHandler);
                mServo.Error -= new ErrorEventHandler(mServo_Error
Handler);
            }


            Application.DoEvents();

            // Cerramos el phidget
            iKit.close();
            mServo.close();

        }
```

El cambio es muy sencillo y lo podemos identificar con facilidad. Estamos ubicando los tres servos en su posición central antes de finalizar la aplicación. Debemos notar cómo el código es usado en cada uno de ellos.

Aunque en este momento no los estamos utilizando, tenemos los handlers relacionados con las entradas analógicas del Phidget 8/8/8, y el control de uno de los servos, por medio del **TrackBar** de la aplicación base. Es bueno tenerlos de esta forma, ya que nos permiten hacer extensiones del programa a futuro. Los handlers equivalentes del controlador de servos no presentan ningún cambio importante.

```
        public void iKit_DigitalHandle(object sender, InputChangeEventArgs e)
        {
            // Verificamos que estemos conectados
            if (conectado0)
            {
                // Verificamos que sea la entrada digital 0

                if (e.Index == 0)
                {
                    // Verificamos el valor de la entrada
                    if (e.Value == true)
                        chbDigital0.Checked = true;
                    else
                        chbDigital0.Checked = false;
                }
```

```
        }
    }

    public void iKit_AnalogoHandle(object sender, SensorChangeEventArgs e)
    {
        // Verificamos que estemos conectados
        if (conectado0)
        {
            // Verificamos que sea la entrada 0
            if (e.Index == 0)
            {
                // Colocamos el valor
                lblAnalogico0.Text = e.Value.ToString();
            }

            // Verificamos que sea la entrada 1
            if (e.Index == 1)
            {
                // Colocamos el valor
                lblAnalogico1.Text = e.Value.ToString();
            }

        }
    }

    private void chbSalida7_CheckedChanged(object sender, EventArgs e)
    {
        // Verificamos que estemos conectados
        if (conectado0)
        {
            // Verificamos el valor del checkbox
            if (chbSalida7.Checked == true)

                iKit.outputs[7] = true; // Prendemos el LED
            else
                iKit.outputs[7] = false; // Apagamos el LED
        }
    }

    private void trkPosicion_Scroll(object sender, EventArgs e)
```

```
        {
            // Verificamos si esta conectado
            if (conectado1)
            {
                // Obtenemos el valor y lo mostramos
                lblPosicion.Text = trkPosicion.Value.ToString();

                // Colocamos la posicion en el servo
                mServo.servos[0].Position = trkPosicion.Value;
            }

        }

        private void configurarPhidget1ToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            // Este codigo es para el phidget 1

            Configuracion dlgConfig = new Configuracion();
            dlgConfig.NumeroSerie = numeroSerie1;

            if (dlgConfig.ShowDialog() == DialogResult.OK)
            {
                // Escribimos el nuevo valor
                numeroSerie1 = dlgConfig.NumeroSerie;

                // Abrimos la llave
                RegistryKey sfwKey =
Registry.LocalMachine.OpenSubKey("Software", true);
                RegistryKey redKey = sfwKey.CreateSubKey("RedUsers2");

                RegistryKey robotKey = redKey.CreateSubKey("Robots2");


                // Colocamos el nuevo valor
                robotKey.SetValue("NumeroSerie1", (object)numeroSerie1);

            }
        }
```

```
        public void mServo_DesconectarHandler(object sender,
DetachEventArgs e)
        {
            // Indicamos que hay desconexion
            conectado1 = false;

            // Quitamos el mensaje
            lblPhidget1.Text = "Sin Phidget";

            // Indicamos con color rojo que no hay conexion
            txtPhidget1.BackColor = System.Drawing.Color.Red;
        }

        public void mServo_ErrorHandler(object sender, ErrorEventArgs e)
        {
            // Indicamos que hay desconexion
            conectado1 = false;

            // Mandamos mensaje de error
            lblPhidget1.Text = e.Description;

            // Indicamos con color amarillo que hay problemas
            txtPhidget1.BackColor = System.Drawing.Color.Yellow;
        }

        private void trkPosicion_LocationChanged(object sender, EventArgs e)
        {
            // Verificamos si esta conectado
            if (conectado1)
            {
                mServo.servos[0].Engaged = true; ;
                Thread.Sleep(500);

                mServo.servos[0].Engaged = false;
            }
        }
```

El handler para el evento de la conexión del controlador de servos es importante, a partir de que nos permite poder ubicar los servos en su posición central y darle un inicio neutro al robot animatrónico.

```
public void mServo_ConectarHandler(object sender, AttachEventArgs e)
 {
     // Indicamos que hay conexion
     conectado1 = true;

     // Enviamos mensaje de la conexion
     lblPhidget1.Text = e.Device.Name;

     // Indicamos con color verde que hay conexion
     txtPhidget1.BackColor = System.Drawing.Color.Lime;


     // Configuramos sus caracteristicas
     if (mServo.Attached)
     {
         // Configuramos los servos

         // Colocamos la aceleracion
         mServo.servos[0].Acceleration = 1000.0;

         // Colocamos la posicion inicial
         mServo.servos[0].Engaged = true;
         mServo.servos[0].Position = 120.0;
         mServo.servos[0].VelocityLimit = 10.0;

         // Colocamos la aceleracion
         mServo.servos[1].Acceleration = 1000.0;

         // Colocamos la posicion inicial
         mServo.servos[1].Engaged = true;

         mServo.servos[1].Position = 120.0;
         mServo.servos[1].VelocityLimit = 10.0;

         // Colocamos la aceleracion
         mServo.servos[2].Acceleration = 1000.0;

         // Colocamos la posicion inicial
         mServo.servos[2].Engaged = true;
         mServo.servos[2].Position = 120.0;
```

```
            mServo.servos[2].VelocityLimit = 10.0;


        }

    }
```

La posición central de cada servo se encuentra en el valor **120**. Debemos tener en cuenta que, también, podemos establecer tanto la aceleración como la velocidad de cada servo de forma independiente.

El siguiente punto es importante, porque tenemos que hacer los handlers para cada uno de los **TrackBars** que creamos antes. Todos hacen prácticamente lo mismo, pero cada uno se refiere a las variables y controles particulares a ese **TrackBar**. Lo primero que hacemos en los handlers es leer el valor del **Trackbar** y ubicarlo en la posición que corresponde en el arreglo. Luego, basta con actualizar el valor de la etiqueta para que muestre este nuevo valor.

```
    private void tbCeja0_Scroll(object sender, EventArgs e)
    {
        // Asignamos el valor en la posicion del arreglo
        ceja[0] = tbCeja0.Value;

        // Actualizamos el valor de la etiqueta
        lblCeja0.Text = ceja[0].ToString();
    }

    private void tbCeja10_Scroll(object sender, EventArgs e)
    {

        // Asignamos el valor en la posicion del arreglo
        ceja[1] = tbCeja10.Value;

        // Actualizamos el valor de la etiqueta
        lblCeja10.Text = ceja[1].ToString();
    }

    private void tbCeja20_Scroll(object sender, EventArgs e)
```

```
{
        // Asignamos el valor en la posicion del arreglo
        ceja[2] = tbCeja2O.Value;

        // Actualizamos el valor de la etiqueta
        lblCeja2O.Text = ceja[2].ToString();
    }

    private void tbCeja3O_Scroll(object sender, EventArgs e)
    {
        // Asignamos el valor en la posicion del arreglo
        ceja[3] = tbCeja3O.Value;

        // Actualizamos el valor de la etiqueta
        lblCeja3O.Text = ceja[3].ToString();
    }

    private void tbCeja4O_Scroll(object sender, EventArgs e)
    {
        // Asignamos el valor en la posicion del arreglo
        ceja[4] = tbCeja4O.Value;

        // Actualizamos el valor de la etiqueta
        lblCeja4O.Text = ceja[4].ToString();
    }

    private void tbCeja5O_Scroll(object sender, EventArgs e)
    {
        // Asignamos el valor en la posicion del arreglo
        ceja[5] = tbCeja5O.Value;

        // Actualizamos el valor de la etiqueta

        lblCeja5O.Text = ceja[5].ToString();
    }

    private void tbCeja6O_Scroll(object sender, EventArgs e)
    {
        // Asignamos el valor en la posicion del arreglo
        ceja[6] = tbCeja6O.Value;
```

```
        // Actualizamos el valor de la etiqueta
        lblCeja6O.Text = ceja[6].ToString();
    }

    private void tbOjoO_Scroll(object sender, EventArgs e)
    {
        // Asignamos el valor en la posicion del arreglo
        ojo[0] = tbOjoO.Value;

        // Actualizamos el valor de la etiqueta
        lblOjoO.Text = ojo[0].ToString();
    }

    private void tbOjo1O_Scroll(object sender, EventArgs e)
    {
        // Asignamos el valor en la posicion del arreglo
        ojo[1] = tbOjo1O.Value;

        // Actualizamos el valor de la etiqueta
        lblOjo1O.Text = ojo[1].ToString();
    }

    private void tbOjo2O_Scroll(object sender, EventArgs e)
    {
        // Asignamos el valor en la posicion del arreglo
        ojo[2] = tbOjo2O.Value;

        // Actualizamos el valor de la etiqueta
        lblOjo2O.Text = ojo[2].ToString();
    }

    private void tbOjo3O_Scroll(object sender, EventArgs e)

    {
        // Asignamos el valor en la posicion del arreglo
        ojo[3] = tbOjo3O.Value;

        // Actualizamos el valor de la etiqueta
        lblOjo3O.Text = ojo[3].ToString();
    }
```

```
private void tbOjo40_Scroll(object sender, EventArgs e)
{
    // Asignamos el valor en la posicion del arreglo
    ojo[4] = tbOjo40.Value;

    // Actualizamos el valor de la etiqueta
    lblOjo40.Text = ojo[4].ToString();
}

private void tbOjo50_Scroll(object sender, EventArgs e)
{
    // Asignamos el valor en la posicion del arreglo
    ojo[5] = tbOjo50.Value;

    // Actualizamos el valor de la etiqueta
    lblOjo50.Text = ojo[5].ToString();
}

private void tbOjo60_Scroll(object sender, EventArgs e)
{
    // Asignamos el valor en la posicion del arreglo
    ojo[6] = tbOjo60.Value;

    // Actualizamos el valor de la etiqueta
    lblOjo60.Text = ojo[6].ToString();
}

private void tbBoca0_Scroll(object sender, EventArgs e)
{
    // Asignamos el valor en la posicion del arreglo
    boca[0] = tbBoca0.Value;

    // Actualizamos el valor de la etiqueta
    lblBoca0.Text = boca[0].ToString();
}

private void tbBoca10_Scroll(object sender, EventArgs e)
{
    // Asignamos el valor en la posicion del arreglo
    boca[1] = tbBoca10.Value;
```

```
        // Actualizamos el valor de la etiqueta
        lblBoca10.Text = boca[1].ToString();
    }

    private void tbBoca20_Scroll(object sender, EventArgs e)
    {
        // Asignamos el valor en la posicion del arreglo
        boca[2] = tbBoca20.Value;

        // Actualizamos el valor de la etiqueta
        lblBoca20.Text = boca[2].ToString();
    }

    private void tbBoca30_Scroll(object sender, EventArgs e)
    {
        // Asignamos el valor en la posicion del arreglo
        boca[3] = tbBoca30.Value;

        // Actualizamos el valor de la etiqueta
        lblBoca30.Text = boca[3].ToString();
    }

    private void tbBoca40_Scroll(object sender, EventArgs e)
    {
        // Asignamos el valor en la posicion del arreglo
        boca[4] = tbBoca40.Value;

        // Actualizamos el valor de la etiqueta
        lblBoca40.Text = boca[4].ToString();
    }


    private void tbBoca50_Scroll(object sender, EventArgs e)
    {
        // Asignamos el valor en la posicion del arreglo
        boca[5] = tbBoca50.Value;

        // Actualizamos el valor de la etiqueta
        lblBoca50.Text = boca[5].ToString();
    }
```

```
private void tbBoca60_Scroll(object sender, EventArgs e)
{
    // Asignamos el valor en la posicion del arreglo
    boca[6] = tbBoca60.Value;

    // Actualizamos el valor de la etiqueta
    lblBoca60.Text = boca[6].ToString();
}
```

Cuando se presiona el botón para la animación, establecemos la variable índice en el valor **0**. Esto con el fin de empezar a recorrer los arreglos desde su primer elemento. Luego, llevamos a cabo una actualización de los valores de los arreglos y, con todo listo, prendemos el **Timer**.

```
private void btnPlay_Click(object sender, EventArgs e)
{
    // Inicializamos el indice
    indice = 0;

    // Actualizamos los valores de los arreglos
    ceja[0] = tbCeja0.Value;
    ceja[1] = tbCeja10.Value;
    ceja[2] = tbCeja20.Value;
    ceja[3] = tbCeja30.Value;
    ceja[4] = tbCeja40.Value;
    ceja[5] = tbCeja50.Value;
    ceja[6] = tbCeja60.Value;

    ojo[0] = tbOjo0.Value;
    ojo[1] = tbOjo10.Value;

    ojo[2] = tbOjo20.Value;
    ojo[3] = tbOjo30.Value;
    ojo[4] = tbOjo40.Value;
    ojo[5] = tbOjo50.Value;
    ojo[6] = tbOjo60.Value;

    boca[0] = tbBoca0.Value;
    boca[1] = tbBoca10.Value;
```

```
        boca[2] = tbBoca20.Value;
        boca[3] = tbBoca30.Value;
        boca[4] = tbBoca40.Value;
        boca[5] = tbBoca50.Value;
        boca[6] = tbBoca60.Value;

        // Prendemos el timer
        timer1.Enabled = true;

    }
```

El handler del **Timer** se encarga de actualizar la posición de los servos cada vez que se realiza un tic. Dependiendo del servo, usamos el arreglo que le corresponde para encontrar su posición. Una vez que se han establecido las posiciones, incrementamos la variable índice en **1**. Al final, llevamos a cabo una verificación para no exceder el tamaño de los arreglos y para detener el **Timer** en donde termina la información de la animación.

```
    private void timer1_Tick(object sender, EventArgs e)
     {
        // Activamos los servos
        mServo.servos[0].Engaged = true;
        mServo.servos[1].Engaged = true;
        mServo.servos[2].Engaged = true;

        // Actualizamos el ojo
        lblPosicionOjo.Text = ojo[indice].ToString();
        mServo.servos[0].Position = ojo[indice];

        // Actualizamos la ceja
        lblPoscionCeja.Text = ceja[indice].ToString();
        mServo.servos[1].Position=ceja[indice];

        // Actualizamos la boca
        lblPosicionBoca.Text = boca[indice].ToString();
        mServo.servos[2].Position = boca[indice];

        // Esperamos que los servos lleguen a su posicion
        Thread.Sleep(500);
```

```
            // Desactivamos los servos
            mServo.servos[0].Engaged = false;
            mServo.servos[1].Engaged = false;
            mServo.servos[2].Engaged = false;

            // Actualizamos el indice
            indice++;

            // Verificamos si hemos finalizado
            if (indice == 7)
            {
                timer1.Enabled = false;
                indice = 0;

            }

        }

    }
}
```

Ahora, podemos compilar la aplicación y probar el funcionamiento de nuestro robot animatrónico. Cuando se ejecuta la aplicación, primero colocamos las posiciones en los **TrackBars** y, después, presionamos el botón de animación.

... RESUMEN

En el presente capítulo, aprendimos que el robot animatrónico tiene tres partes móviles: podemos hacer que mueva la ceja, el ojo o la boca. Para cada parte que se moverá, usamos un servo; todo queda montado de manera que los movimientos no se estorben entre sí. El software permite establecer una animación por medio de diferentes TrackBars. Cada Trackbar simboliza la posición del servo en un momento en particular. Por medio de un Timer recorremos los valores ingresados en los TrackBars y movemos los servos de acuerdo acon ellos.

ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1. Construya un robot animatrónico que presente dos ojos con movimiento independiente.

2. Modifique el programa para que, por medio de un TextBox, podamos ingresar el valor del intervalo del Timer.

3. Modifique el programa para que, por medio de una serie de TextBox, podamos establecer la velocidad de cada uno de los servos de manera independiente.

4. Modifique el programa para que podamos mover el ojo y la ceja con la ayuda del joystick que podemos conectar al Phidget 8/8/8.

5. Modifique el programa para que pueda salvar y leer la animación ingresada, en un archivo de disco.

6. Modifique el programa para que el movimiento del servo sea controlado por una interpolación lineal entre los valores de los TrackBars.

7. Modifique el cartón de la cara para crear un personaje diferente.

8. Modifique el robot para que tenga orejas que se puedan mover.

9. Modifique el software para poder controlar las orejas.

10. Construya un robot animatrónico con diseño propio.

Plataforma robótica

El último proyecto de este libro es la creación de una plataforma robótica. Una plataforma nos permite tener un robot que puede moverse por diferentes lugares y realizar más tareas que un robot que siempre se mantiene fijo. La plataforma resulta sencilla de construir y podemos utilizarla con nuestros propios componentes para implementar cualquier idea que tengamos en mente.

DESCRIPCIÓN DEL PROYECTO

A lo largo de este capítulo crearemos una **plataforma robótica** para computadoras tipo laptop. La plataforma cuenta con dos pisos. En el primer piso, hay espacio suficiente para poder ubicar cualquier tipo de componente como baterías, motores, sensores y circuitos electrónicos. Mientras que el piso superior es el lugar que destinaremos a la computadora laptop.

La plataforma tendrá tres ruedas colocadas en un esquema de triciclo invertido. Es decir que dos ruedas se montarán al frente de la plataforma y, en la parte posterior, tendremos una rueda que servirá para brindar una mayor estabilidad. Tanto la tracción como la dirección se realizan por medio de las ruedas delanteras. En este proyecto, sólo mostramos su posición, pero, después, podemos instalar otro tipo de motor que corresponda a nuestras necesidades.

Componentes necesarios

Para la construcción de la plataforma, usaremos **tubos de PVC** de media pulgada de diámetro. Este tipo de tubos ofrece una serie importante de ventajas: es barato, fácil de cortar, ligero y otorga buen soporte para pesos pequeños. Debemos cortar el tubo en diferentes medidas. También usaremos varios tipos de conectores.

Los tubos necesarios son los siguientes:

NOMBRE	CANTIDAD	DESCRIPCIÓN
A	3	44 cm
B	4	32 cm
C	4	24 cm
D	2	17 cm
E	4	4 cm

***Tabla 1.** En la tabla, podemos ver la cantidad de tubos que necesitaremos para la construcción de la plataforma.*

III PROYECTOS DE LOS LECTORES

Se agradece a todas las personas que realicen sus propios proyectos envien imágenes o videos de los mismos a **wnicosio@yahoo.com**. Recordamos que todo el contenido de este libro se puede utilizar con fines personales y académicos, siempre que se mencione al libro, y no pueden usarse para proyectos comerciales.

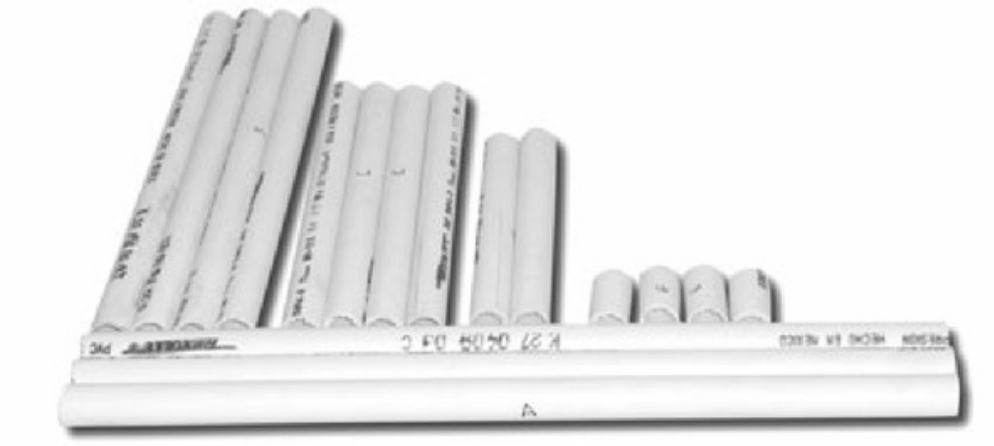


Figura 1. *En esta fotografía, observamos los tubos de PVC necesarios para la plataforma.*

También, usaremos los siguientes conectores de PVC:

CANTIDAD	DESCRIPCIÓN
2	Codo de 90 grados
2	Conexión tipo T
8	Conexión tipo Y a 90 grados

Tabla 2. *Estos serán los conectores de PVC que usaremos para unir los tubos.*

Además, necesitamos dos ruedas de 7 pulgadas de diámetros y una rueda de patín de 3.5 pulgadas de diámetro. Para unir las piezas, podemos usar pegamento para plástico PVC, pero, en la mayoría de los casos, las piezas entran a presión. Para sostener la computadora y los componentes, necesitaremos dos piezas de madera delgada o de cartón rígido. El piso de la computadora debe tener como tamaño 52 cm x 42 cm, y la base para los componentes en el piso inferior debe ser de 52 cm x 40 cm. Con esto, ya tenemos todo lo necesario para empezar a ensamblar la plataforma.

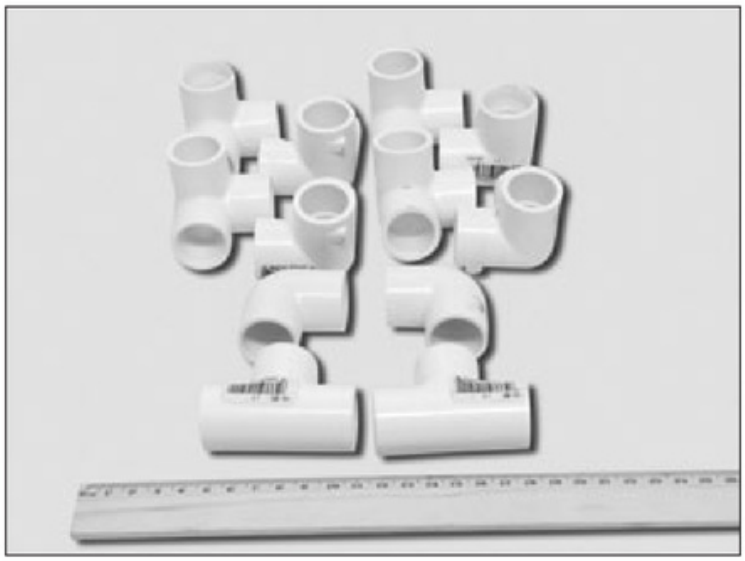


Figura 2. *Éstos son los conectores que necesitaremos para construir la plataforma robótica.*

CONSTRUCCIÓN DE LA PLATAFORMA

Para empezar a construir la plataforma, realizaremos el soporte de la rueda de patín. Este soporte sostiene la rueda que estará en la parte trasera de la plataforma. Para construirla, necesitamos los cuatro tubos **E**, los conectores a **90 grados** y los conectores **T**. Al unirlos, tendremos una especie de marco que sostendrá a la rueda.

Figura 3. *Éstos son los componentes necesarios. Aquí los vemos en sus futuras posiciones.*

El primer paso por realizar es la unión de los conectores a **90 grados**. Entre ellos, ubicamos uno de los tubos **E**. El tubo debe entrar a presión con los conectores. Si llegara a moverse demasiado, podemos aplicar un poco de pegamento en las uniones.

Figura 4. *La primera unión que llevamos a cabo es la de los conectores a 90 grados.*

Luego, tomamos los conectores tipo **T**, que también estarán unidos mediante un tubo **E**. Es importante notar cuál de los tres extremos usamos para el armado, de tal forma que queden dos extremos que, después, podamos unir con los conectores a **90 grados**.

Figura 5. *Los extremos laterales de los conectores tipo T son unidos por medio de un tubo E.*

A continuación, tomamos dos tubos **E** y los colocamos en los extremos libres de los conectores a **90 grados**. Los conectores tipo **T** serán insertados en los otros extremos de estos tubos. Para finalizar esta parte, nos aseguramos de que todos los conectores queden bien sujetados y de manera compacta. Al finalizar, debemos tener un marco formado por todos estos componentes.

Figura 6. *Los tubos* ***E*** *son colocados en los conectores para preparar el ensamble final de esta sección.*

Figura 7. *Ésta es la forma como luce el soporte para la rueda del patín una vez que todas las piezas se encuentran unidas.*

El soporte para la computadora

El soporte para la computadora forma la parte superior de la plataforma móvil. La construcción es sencilla, pues consiste en un marco de PVC. En una etapa posterior, instalaremos el cartón que sostendrá a la computadora. Para este soporte, necesitamos cuatro conectores tipo **Y a 90 grados**, dos tubos tipo **A** y dos tubos tipo **E**. Los tubos de cada tipo serán paralelos entre sí, dándole a este soporte una forma rectangular.

Figura 8. *Éstos son los elementos que usaremos para crear el soporte de la computadora.*

Con todo el material reunido, podemos comenzar a ensamblar. Tomamos los tubos **A** y colocamos, en cada uno de sus extremos, un conector tipo **Y**. Éstos formarán los lados largos de la base de la computadora.

Figura 9. *Empezamos armando los lados largos del soporte; cada uno de ellos necesita un conector Y.*

Luego, simplemente, usamos los tubos **B** y los insertamos en los conectores **Y**, de tal manera que obtengamos un marco rectangular con el PVC. Si lo deseamos, podemos aplicar pegamento para PVC en estas uniones para que el marco quede bien firme.

Figura 10. *El soporte para la computadora está completo y es lo bastante firme como para sostener un laptop.*

El soporte para los componentes

El soporte para los componentes es la parte inferior de la plataforma móvil. En esta sección, podemos instalar motores, baterías, circuitos electrónicos y sensores. Para su construcción, en principio, necesitaremos cuatro conectores tipo **Y a 90 grados**, un tubo **A**, dos tubos **B**, dos tubos **D** y el soporte para la rueda de patín.

Figura 11. *En esta fotografía, podemos observar los elementos que necesitamos inicialmente para el soporte de los componentes.*

Para crear la parte que se encontrará al frente, debemos tomar un tubo **A** e instalar un conector tipo **Y** en cada uno de sus extremos. Los conectores se colocarán de tal forma que tengamos una salida para arriba y otra para la parte trasera del soporte.

Figura 12. *Empezamos con la parte frontal del soporte con la instalación de dos conectores en los extremos del tubo.*

Continuamos con la parte trasera del soporte. Tomamos los tubos **D** y dos conectores tipo **Y**. A cada uno de los conectores, le colocamos un tubo **D**. Es importante tener cuidado con la forma en que lo hacemos, debido a que deben permitir que, después, se ubiquen los tubos **B** y podamos contar con un espacio de conexión libre hacia arriba.

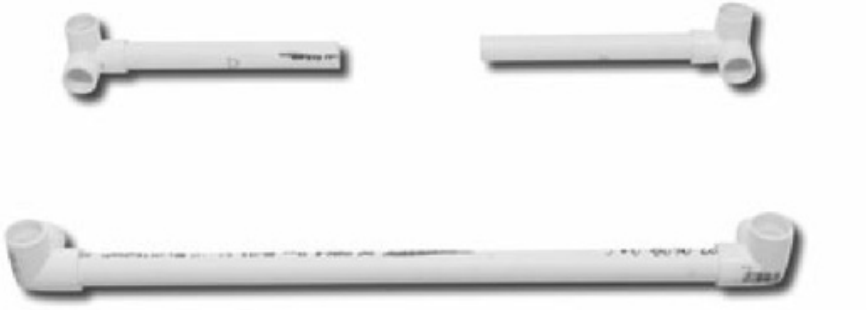


Figura 13. *Es importante notar la orientación de los conectores y los tubos para permitir el resto del ensamble.*

Ahora, tomamos el soporte para la rueda de patín y la conectamos a los extremos libres de los tubos **D**. Esta unión es importante, y todos los elementos deben quedar en una posición horizontal porque, en caso contrario, la rueda podría no tener una nivelación ajustada. Recordemos, entonces, que esta conexión debe ser firme y segura. Si el PVC por sí mismo no tiene suficiente presión para evitar movimientos, entonces, es recomendable aplicar pegamento para hacerla lo más segura posible.

Figura 14. *El soporte para la rueda de patín se coloca en la parte trasera del soporte para los componentes.*

Tomamos, entonces, los tubos **B** y, con ellos, unimos el frente y la sección posterior del soporte. Al igual que en el soporte de la computadora, obtendremos un marco rectangular.

Figura 15. *Aquí podemos observar el soporte de los componentes en etapa de ensamblar los tubos **B** para formar el marco.*

Aún falta para terminar esta sección. Tomamos cuatro tubos **C**, que funcionarán como columnas para sostener el soporte de la computadora. Los tubos deben estar anclados de manera firme y pegados al soporte de los componentes, pero no debemos adherirlos al soporte de la computadora. Esto lo hacemos con el fin de que el soporte de la computadora sea un techo removible que, una vez construida la plataforma, nos permita un acceso más fácil y cómodo a los componentes.

Figura 16. *Agregamos los tubos* ***C*** *que actuarán como columnas para el otro soporte.*

Con cuidado, colocamos uno a uno los tubos **C** en las conexiones libres sin olvidarnos de aplicar pegamento en el extremo inferior del tubo. Los cuatro tubos deben quedar perfectamente perpendiculares al soporte.

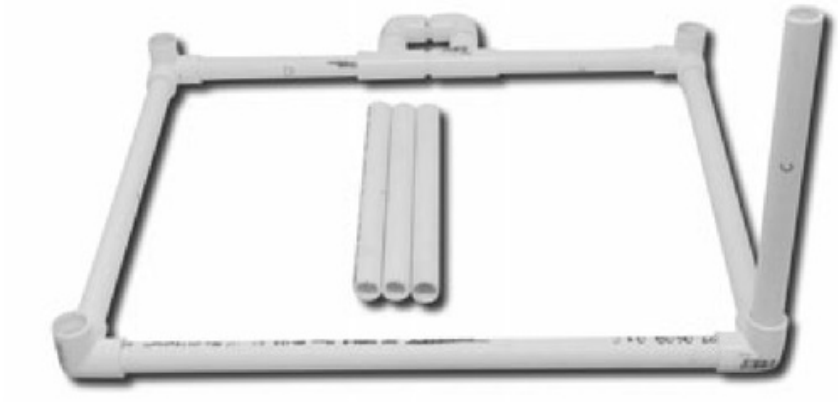


Figura 17. *Colocamos los tubos* ***C*** *en las conexiones disponibles del soporte de componentes.*

Figura 18. *Las cuatro columnas están fijadas y se mantienen perpendiculares al soporte de componentes.*

Instalación de la rueda delantera

Para instalar la rueda de patín, sólo necesitaremos del soporte de los componentes y de varios precintos, también conocidos como cinchos, de plástico. Estos precintos, que vemos en la **figura 19**, son usados para sostener la rueda sin tener que hacer perforaciones al PVC y, si se ajustan de manera correcta, la rueda debe mantenerse sin problemas en distintos tipos de superficies.

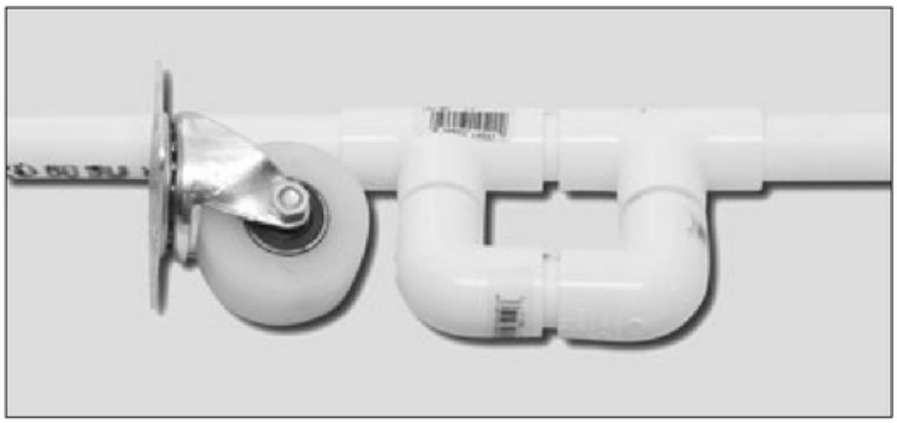


Figura 19. *Éstos son los componentes necesarios para instalar la rueda de patín en nuestra plataforma.*

La rueda de patín tiene una base con cuatro perforaciones. Debemos ubicar la base de forma que se asiente sobre el soporte. Recordemos siempre que la rueda debe orientarse hacia abajo, es decir, en dirección contraria a los tubos que colocamos como columnas. En principio, pasamos los precintos de plástico sin apretarlos demasiado. Esto nos permite mover y centrar la rueda de patín. El precinto debe pasar por el orificio de sujeción de la base y envolver al PVC de la plataforma.

Figura 20. *Los cinchos, o precintos, son colocados en los orificios de la base, en este punto, sin ajustarlos demasiado.*

Cuando la base se encuentre en una posición adecuada, empezamos a estrechar los precintos uno por uno. Conforme los vamos apretando, no debemos descuidar la posición de la base.

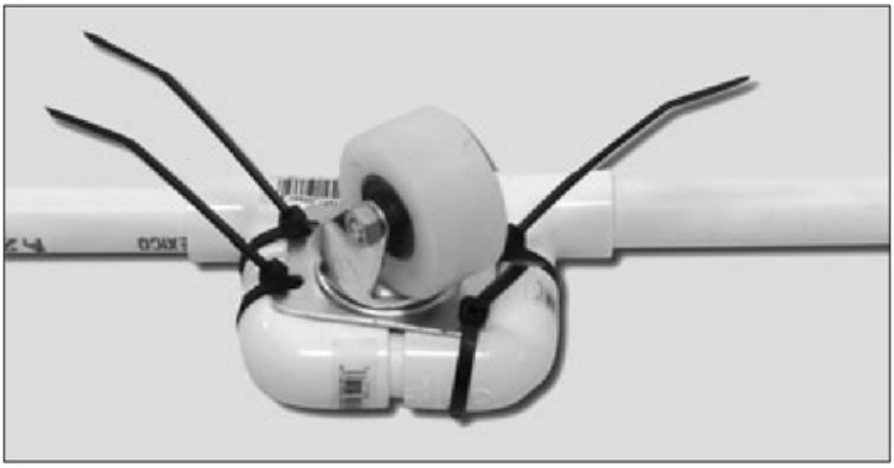


Figura 21. *Los precintos en posición sujetando a la base, listos para su ajuste final.*

Por último, apretamos con fuerza los precintos. Incluso, podemos ayudarnos mediante el uso de unas pinzas. No habría problema si la base quedara un poco fuera de centro. Con unas tijeras o con una navaja, podemos cortar los extremos que sobran de los precintos, para que éstos no estorben el movimiento de la plataforma.

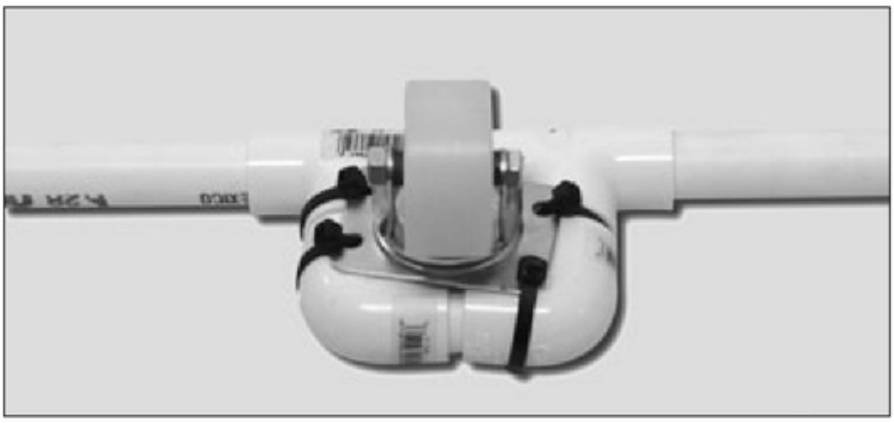


Figura 22. *Recortamos los extremos sobrantes de los precintos, y la rueda ya se encuentra en posición.*

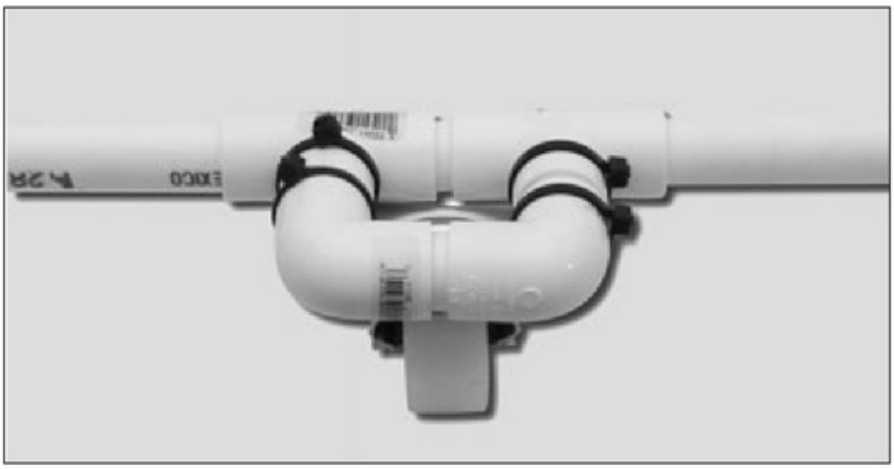


Figura 23. *Podemos observar la rueda en posición y funcionando.*

Finalización de los soportes

Tenemos dos soportes en esta plataforma. Uno de ellos es para la computadora, y el otro, para los componentes. Para facilitar el ensamblado de los soportes usaremos otra vez precintos de plástico debido a que, con ellos, reducimos la necesidad de perforar los tubos de PVC. Tomamos los cartones o maderas y, en su perímetro, hacemos una serie de orificios. Estos orificios deben tener el diámetro necesario para que el sujetacable pueda pasar por él sin problema. Lo mejor es hacer los agujeros a dos centímetros de distancia del borde del cartón. Con cuatro orificios en cada lado del cartón, es suficiente. Estos cartones actúan como pisos de cada uno de los soportes.

Figura 24. *Llevamos a cabo las perforaciones que usaremos en los soportes de la plataforma.*

Ahora, tomamos el piso y el soporte de la computadora y los alineamos entre sí. Debemos tener en cuenta que el piso de la computadora es ligeramente más grande que el piso que usaremos para sostener los componentes. Ubicamos el piso en la parte superior del soporte de la computadora.

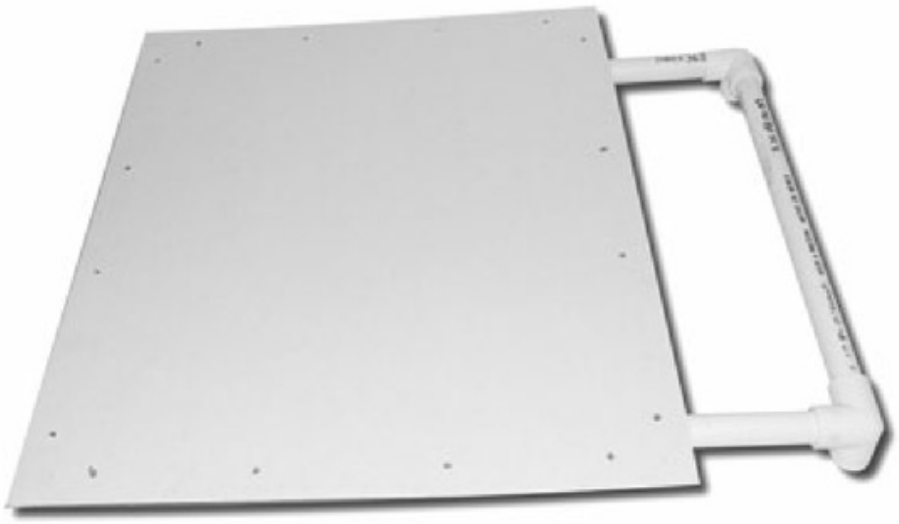


Figura 25. *Podemos observar el piso sobre el soporte de la computadora.*

Empezamos a colocar precintos de plástico alrededor del piso. Los precintos deben pasar por el orificio del piso y envolver el tubo de PVC. Otra vez, debemos recordar no apretar del todo los precintos para permitir cualquier ajuste de la posición del piso.

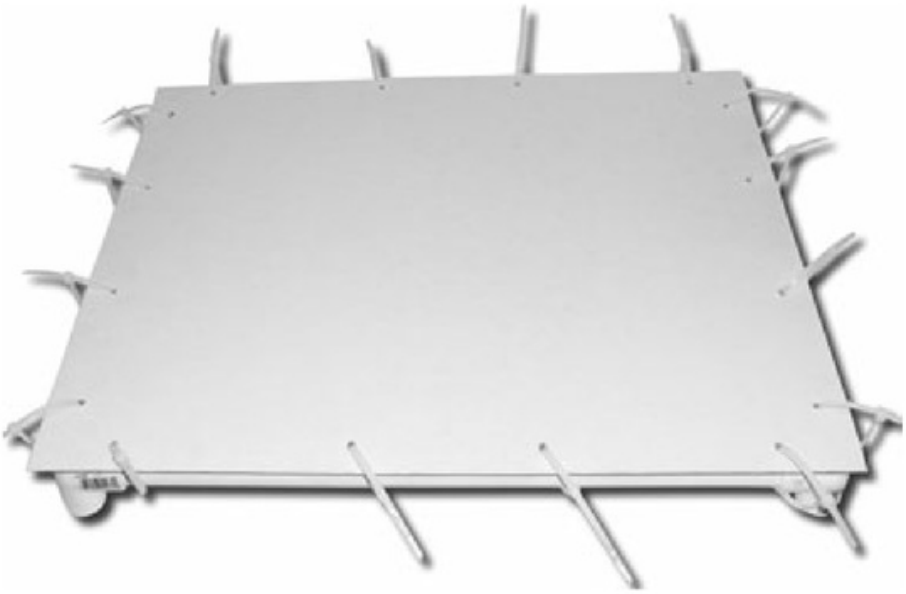


Figura 26. *Ubicamos los precintos alrededor del piso, y los aprovechamos para sujetarlo al soporte de PVC.*

Cuando el piso se encuentre centrado en forma correcta, entonces, ajustamos los precintos para sujetarlos con firmeza. Si estamos usando cartón para el piso, no hay que apretar en exceso para evitar que éste se doble. Después de esto, con mucho cuidado, cortamos todo el exceso de plástico.

Figura 27. *El soporte de la computadora está finalizado y listo para usarse.*

El proceso de ubicar los componentes en el piso es similar al que realizamos con anterioridad. En este caso, debemos colocar el piso en la parte inferior del soporte y poner los precintos a su alrededor para sujetarlo a los tubos de PVC.

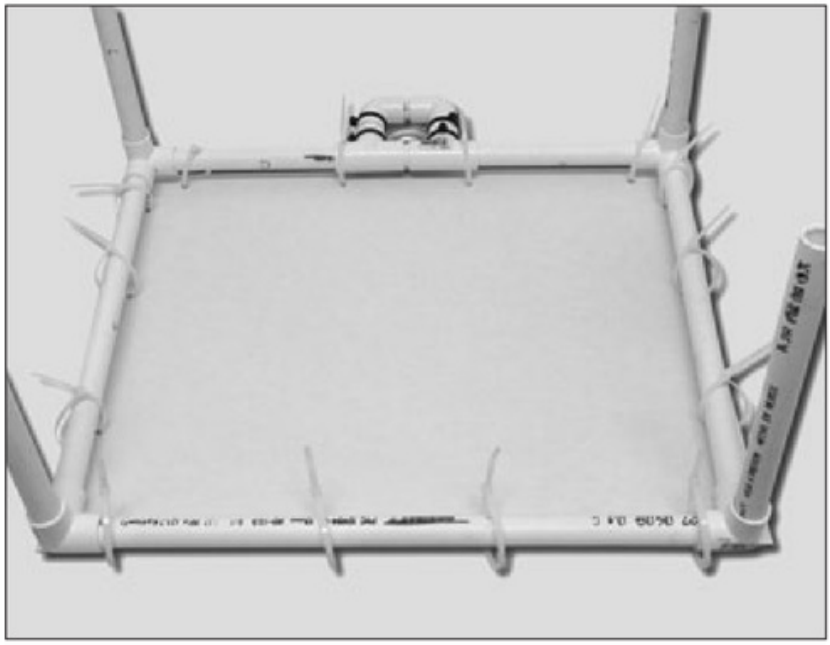


Figura 28. *Hemos colocado los precintos alrededor del soporte del componente, y el piso se encuentra en posición para ser fijado en forma definitiva.*

Cuando esté todo listo, apretamos los precintos para fijar el piso al soporte y, por último, recortamos los excesos de plástico para que no estorben en el proceso de instalar los componentes que necesitemos.

Figura 29. *El piso ha quedado sujeto al soporte de los componentes.*

Instalación de las ruedas traseras

La plataforma que estamos armando en este capítulo es sólo un modelo. Esto se debe a que existe una gran variedad de tipos de ruedas y de motores, por lo que no es posible cubrirlos todos. Por esto, sólo instalaremos las ruedas de manera simbólica y, en la plataforma final, las colocaremos dependiendo de los motores que se hayan conseguido.

Figura 30. *En esta fotografía, podemos observar la posición de la primera rueda.*

Las ruedas con sus motores deben ser instaladas en la parte frontal de la plataforma, cada una de un lado. El motor debe ser colocado sobre el soporte de componentes. Nuestras ruedas deben quedar de tal forma que no tengan problemas al rotar y, siempre, lo más al frente posible.

Figura 31. *La segunda rueda ha sido colocada en posición.*

Detalles finales

Con todos los componentes en su lugar, ahora podemos proceder al ensamble final de la plataforma. Tomamos el soporte de la computadora y lo ponemos sobre el soporte de componentes. Los conectores del soporte de la computadora deben coincidir con los tubos que usamos como columnas. Por último, debemos verificar que queden correctamente conectados y firmes.

Figura 32. *Instalamos el soporte de la computadora como un techo de la plataforma.*

Ahora, podemos ubicar la computadora en la parte superior de la plataforma; cualquier cable que sea necesario puede pasar sin problemas entre los espacios abiertos.

Figura 33. *Hemos finalizado la plataforma y vemos cómo podemos ubicar la computadora sobre ella.*

Con esto, ya tenemos la plataforma correcta y, con lo que hemos aprendido en los capítulos anteriores, podemos programar sin problema un robot autónomo. Los Phidgets pueden ser utilizados para controlar los motores y dar dirección al robot, también, pueden capturar información con sus sensores y reaccionar de acuerdo con ellos.

Figura 34. *Ésta es otra vista de la plataforma que hemos creado.*

Controlar los motores de la plataforma

Es el momento de adentrarnos en cómo podemos controlar las direcciones de los motores por medio de software. Para los motores de las plataformas robóticas, por lo general, usamos motores de corriente directa. Cuando el motor recibe la corriente, veremos que gira en una dirección y, si cambiamos la polaridad, también cambia la dirección de la rotación. Ésta es la base para el circuito que construiremos en este proyecto en donde necesitaremos dos motores eléctricos. El modelo que usamos en este caso es el **Pololu 250:1**.

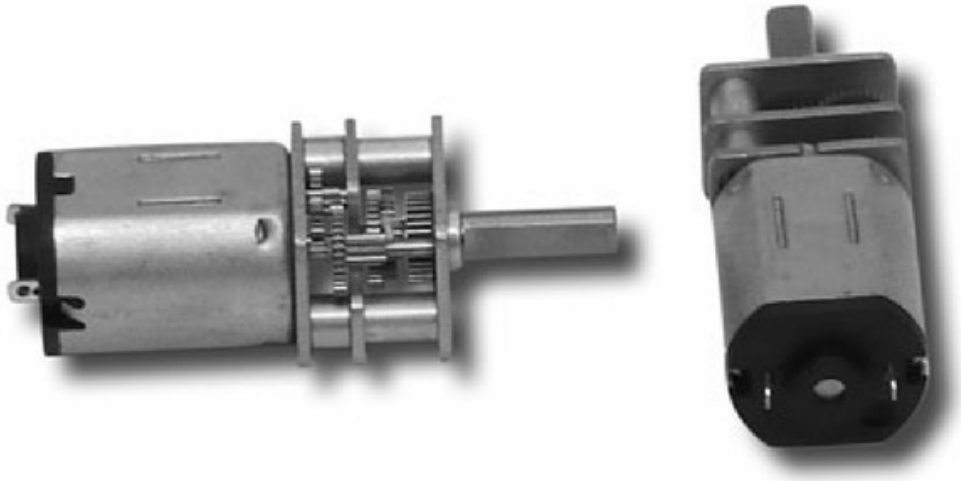


Figura 35. *En esta fotografía, observamos los motores que usaremos; son motores con engranes de reducción.*

Además, necesitamos ruedas para los motores. Las ruedas son del mismo fabricante, y sus dimensiones son de 42 x 19 milímetros. Podemos construir una pequeña base para sostener a los motores; esto ya lo realizar hacer sin problemas.

Figura 36. *Un ejemplo del tipo de rueda que se puede instalar en los motores. Existen de diferentes tamaños y diseños.*

Para facilitarnos la construcción del circuito y el control de los motores, usaremos dos tableros de control de relé fabricados por Phidgets. Este tablero, que mostramos en la **figura** 37, es el modelo **3051** que contiene dos relés.

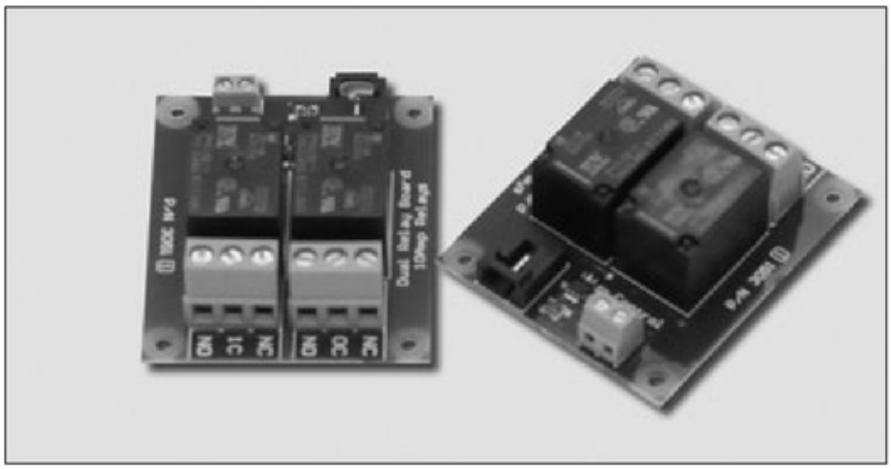


Figura 37. *Necesitamos dos tableros 3051 para este proyecto. Uno para cada motor.*

Los motores no tienen cables conectados, por lo que necesitaremos soldar dos cables de 20 centímetros cada uno en sus terminales. Es recomendable usar cables de diferente color en cada terminal. Como tenemos dos motores, las terminales equivalentes deben tener el mismo color, esto facilita, más adelante, la construcción del circuito. En el ejemplo del libro, usamos un cable de color claro, en este caso **amarillo**, y otro de un color oscuro que será **verde**.

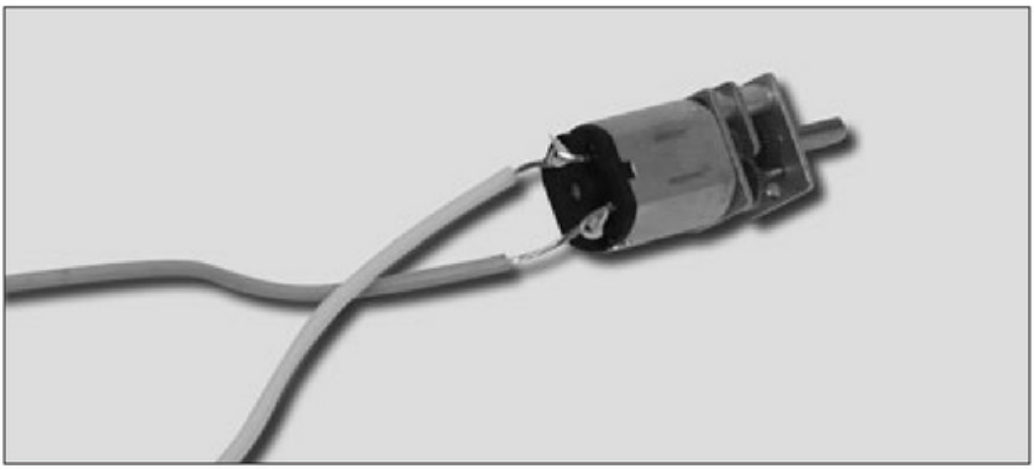


Figura 38. *Los cables han sido soldados a las terminales del motor.*

Los relés actúan como interruptores. Mientras reciban una corriente, estarán conectados a una terminal y, cuando no la reciban, a otra. La terminal que se puede conectar a ambas es la común. A las otras terminales, las llamamos normalmente abierta o **NO** y normalmente cerrada o **NC**. Tomamos los cables de los motores y los insertamos en los comunes de cada uno de los relés de los tableros. Hay que notar que los colores de los cables están conectados en la misma posición en ambos tableros.

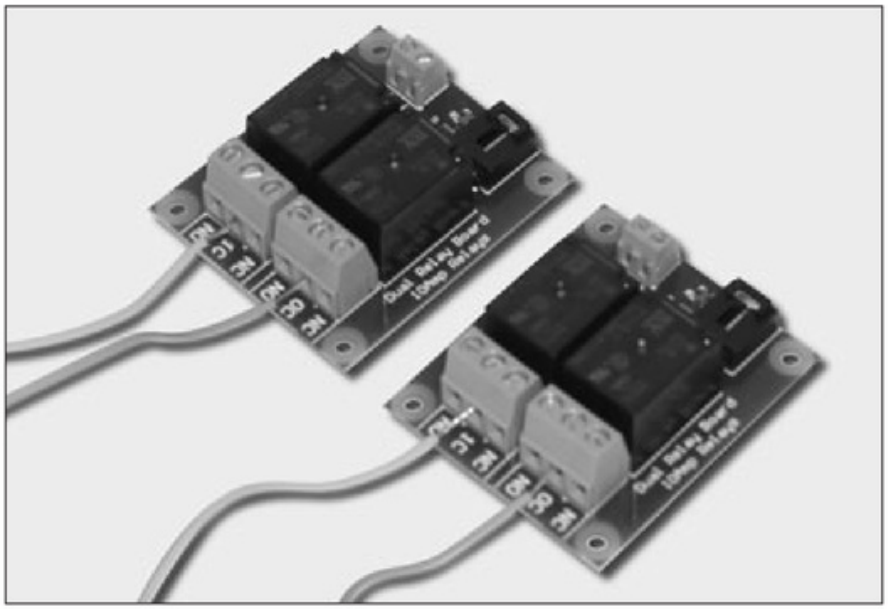


Figura 39. *Los cables de los motores han sido insertados en los tableros.*

Para el siguiente paso, debemos ser cuidadosos. Colocaremos el cable que alimentará con voltaje a los motores. Necesitamos un cable **rojo** de alrededor de 20 centímetros y tres más pequeños de unos 10 centímetros de longitud. El cable largo se conecta al **NC** del primer relé del tablero que tenemos a nuestra derecha. De esa misma conexión, llevamos un cable más pequeño al **NO** del segundo relé del mismo tablero. Luego, conectaremos los tableros entre sí. A partir del **NO** del segundo relé, llevamos un cable al **NC** del primer relé del segundo tablero. De esta última

terminal, colocamos otro cable al **NO** del segundo relé del segundo tablero. Para ejemplificar esto mejor, veamos la siguiente figura.

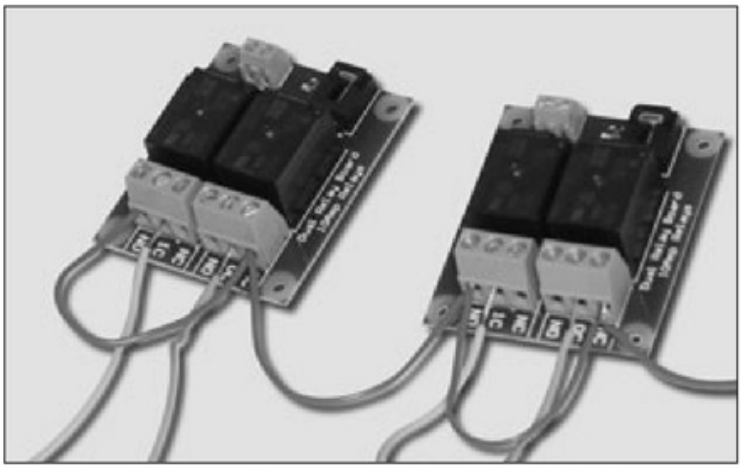


Figura 40. *En esta fotografía, podemos observar cómo conectar los cables rojos que llevan la alimentación eléctrica.*

De una manera similar, colocaremos el cable de tierra. Este cable es de color **negro**. El cable se conecta al **NO** del primer relé del primer tablero y, de ahí, con otro cable, al **NC** del segundo relé del mismo tablero. A partir de la última terminal conectada, colocamos otro cable al **NO** del primer relé del segundo tablero y continuamos con otro cable al **NC** del segundo relé en el segundo tablero. También, podemos guiarnos con la siguiente figura para hacer la conexión de forma correcta.

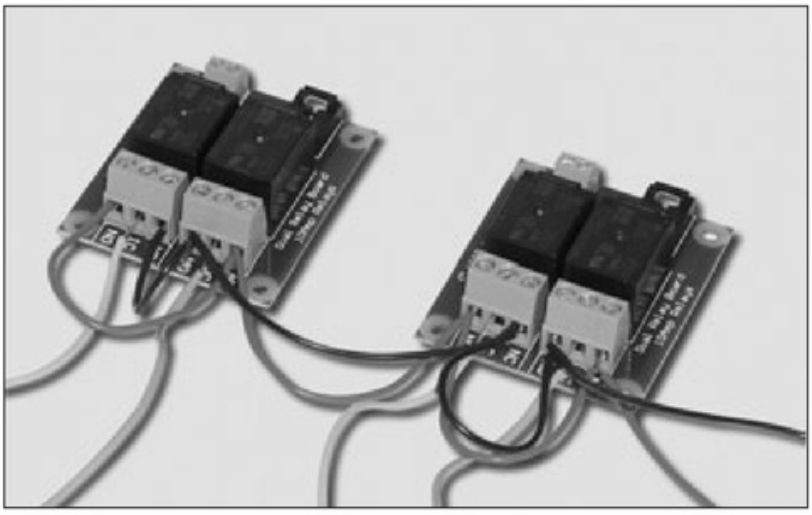


Figura 41. *La forma como debemos conectar los cables negros se muestra en esta figura.*

Los tableros 3051 serán controlados por el Phidget 8/8/8. En la parte posterior, poseen un conector, y cada una de las terminales permite controlar un relé. También, hay una conexión para el puerto analógico del 8/8/8, pero ésta es usada sólo para alimentar eléctricamente al tablero y no para controlarlo. Necesitamos cuatro cables más, también de 20 centímetros. Con estos cables, conectaremos los

tableros a las salidas digitales del 8/8/8 y, de esta forma, los controlaremos.

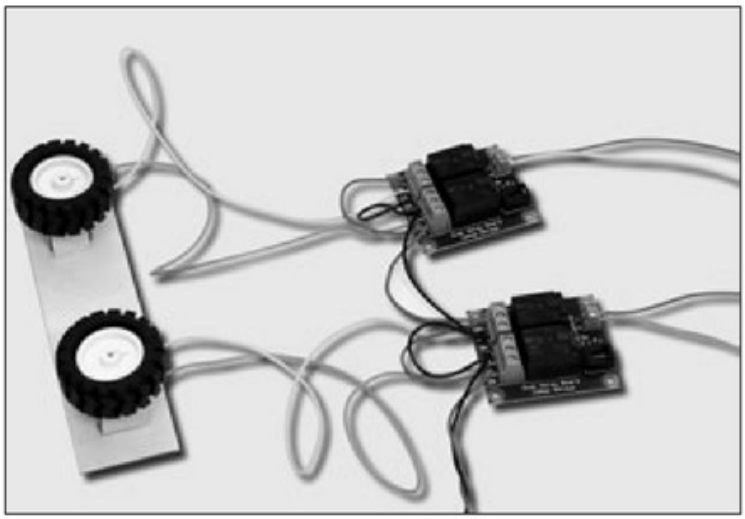


Figura 42. *El circuito con todos los cables está listo, sólo falta conectarlo al 8/8/8.*

Conservando el orden en el que están los cables, los conectaremos a las salidas digitales del 8/8/8. Usaremos las salidas **0** y **1** para el primer tablero, y las salidas **2** y **3**, para el segundo.

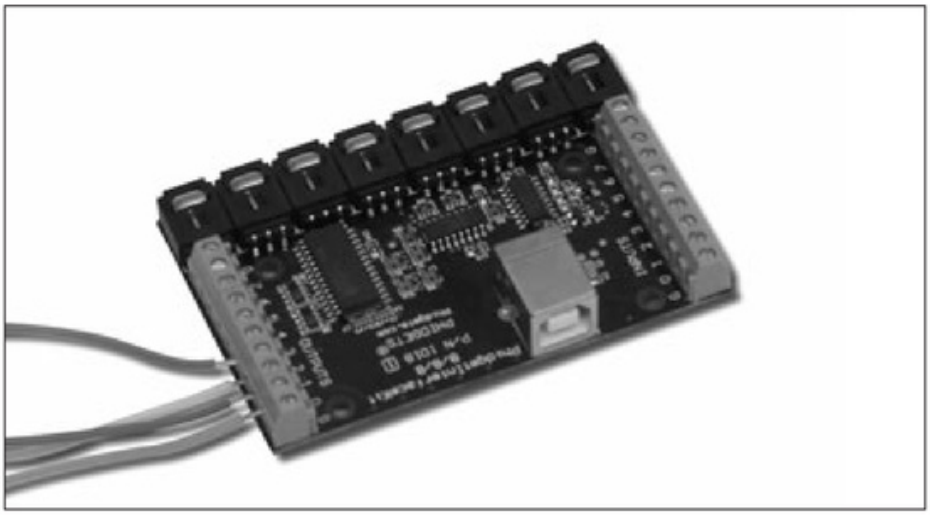


Figura 43. *Ésta es la forma como conectamos los 3051 al 8/8/8*

III POLARIDAD CORRECTA

Si nos equivocamos en la polaridad, ya sea al soldar o al colocar los cables de los motores a las tabletas, obtendremos un motor que girará en la dirección contraria a la que esperamos. Por eso, es conveniente colocar un color diferente para cada cable del motor, ya que esto nos permite identificar su polaridad.

EL SOFTWARE PARA CONTROLAR LOS MOTORES

El software para controlar los motores es el más sencillo del libro. Sólo necesitamos cuatro **RadioButtons** para controlar la dirección de rotación de los motores. La rotación es controlada por medio de las salidas digitales del Phidget 8/8/8. Es preciso, entonces, establecer el valor de **true** o **false** en el índice que pertenece a la salida correspondiente. El software está basado en la aplicación base, y los cambios son mínimos.

Creación de la interfaz de usuario

Para este programa, debemos adicionar un **GroupBox**, el cual tiene el valor **Movimiento Robot** en su propiedad **Text**. Dentro de este **GroupBox**, estableceremos cuatro **RadioButtons**. El primer **RadioButton** tiene por nombre **rbtnAdelante** y, en su propiedad **Text**, ingresamos el valor **Adelante**. El siguiente lleva el nombre de **rbtnDerecha** con el valor de **Derecha** en su propiedad **Text**. De manera similar, los otros dos **RadioButtons** tienen por nombre **rbtnAtras** y **rbtnIzquierda**, y, respectivamente, los valores en la propiedad **Text** serán **Atrás** e **Izquierda**.

Figura 44. *Ésta es la interfaz de usuario que debemos crear. Lo hacemos adicionando los cuatro botones de dirección.*

El código no tiene grandes cambios, y ya conocemos todos los elementos. No es necesario adicionar ninguna variable extra al código.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
```

```
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading;
using System.Windows.Forms;

// Necesario para el registro
using Microsoft.Win32;

// Necesario para los Phidgets
using Phidgets;
using Phidgets.Events;

namespace BasePhidgets
{
    public partial class Form1 : Form
    {
        // Datos necesarios para la aplicacion base

        private int numeroSerie;         // Numero de serie del phidget
        private bool conectado;          // Para saber si hay conexion
        private InterfaceKit iKit;       // Interfaz al kit

        public Form1()
        {
            InitializeComponent();

            // Inicializamos la aplicacion

            // Empezamos como si no estuvieramos conectados
            conectado = false;

            // Colocamos un valor de default para el numero de serie
            numeroSerie = 0;

            // Verificamos si existe la llave en el registro
            RegistryKey sfwKey = Registry.LocalMachine.OpenSubKey
("Software");
            RegistryKey redKey = sfwKey.OpenSubKey("RedUsers");
```

```
            // Verificamos si existe
            if (redKey == null) // la llave no existe
            {

                // Creamos la llave
                sfwKey = Registry.LocalMachine.OpenSubKey("Software",
true);
                redKey = sfwKey.CreateSubKey("RedUsers");

                RegistryKey robotKey = redKey.CreateSubKey("Robots");

                // Colocamos un valor de default
                robotKey.SetValue("NumeroSerie0", (object)numeroSerie);

            }
            else // la llave existe
            {
                RegistryKey robotKey = redKey.OpenSubKey("Robots");

                // Obtenemos el numero de serie guardado
                numeroSerie = (int)robotKey.GetValue("NumeroSerie0");
            }

        }

        private void salirToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            // Cerramos la aplicacion
            this.Close();
        }

        private void configurarToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            Configuracion dlgConfig = new Configuracion();
            dlgConfig.NumeroSerie = numeroSerie;

            if (dlgConfig.ShowDialog() == DialogResult.OK)
            {
```

```
                // Escribimos el nuevo valor
                numeroSerie = dlgConfig.NumeroSerie;

                // Abrimos la llave
                RegistryKey sfwKey =
Registry.LocalMachine.OpenSubKey("Software", true);
                RegistryKey redKey = sfwKey.CreateSubKey("RedUsers");

                RegistryKey robotKey = redKey.CreateSubKey("Robots");

                // Colocamos el nuevo valor
                robotKey.SetValue("NumeroSerieO", (object)numeroSerie);

            }
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            // Inicializamos el objeto InterfaceKit
            try
            {
                // Instanciamos el objeto
                iKit = new InterfaceKit();

                // Colocamos los handlers relacionados con la conexion
                iKit.Attach += new AttachEventHandler(iKit_Conectar
Handler);
                iKit.Detach += new DetachEventHandler(iKit_Desconectar
Handler);

                // Colocamos el handler relacionado con el error
                iKit.Error += new ErrorEventHandler(iKit_ErrorHandler);

                // Colocamos el handler para las entradas digitales
                iKit.InputChange += new InputChangeEventHandler
(iKit_DigitalHandle);

                // Colocamos el handler para las entradas analogicas
                iKit.SensorChange += new
SensorChangeEventHandler(iKit_AnalogoHandle);
```

```
            // Abrimos para conexiones
            iKit.open(numeroSerie);


        } // Fin de try
        catch (PhidgetException ex)
        {
            // Enviamos mensaje con el error
            lblPhidget0.Text = ex.Description;

            // Indicamos con amarillo que hay problemas
            txtPhidget0.BackColor = System.Drawing.Color.Yellow;
        }

    }
```

En el handler de la conexión del Phidget, simplemente, establecemos los motores en la posición que consideremos hacia delante. En este caso, indicamos en el handler que las cuatro salidas digitales tienen el valor **false**.

```
    // Handler de la conexion
    public void iKit_ConectarHandler(object sender, AttachEventArgs e)
    {
        // Indicamos que hay conexion
        conectado = true;

        // Enviamos mensaje de la conexion
        lblPhidget0.Text = e.Device.Name;

        // Indicamos con color verde que hay conexion
        txtPhidget0.BackColor = System.Drawing.Color.Lime;

        // Inicializamos con los dos motores avanzando hacia el frente
        iKit.outputs[0] = false;
        iKit.outputs[1] = false;
```

```
            iKit.outputs[2] = false;
            iKit.outputs[3] = false;


        }
```

La siguiente parte del programa es la misma que en la aplicación base.

```
        // Handler de desconexion
        public void iKit_DesconectarHandler(object sender, DetachEventArgs e)
        {
            // Indicamos que hay desconexion
            conectado = false;

            // Quitamos mensaje
            lblPhidget0.Text = "Sin phidget";

            // Indicamos con color rojo que no hay conexion
            txtPhidget0.BackColor = System.Drawing.Color.Red;

        }

        // Handler del error
        public void iKit_ErrorHandler(object sender, ErrorEventArgs e)
        {
            // Indicamos que hay desconexion
            conectado = false;

            // Mandamos mensaje de error
            lblPhidget0.Text = e.Description;

            // Indicamos con color amarillo que hay problemas
            txtPhidget0.BackColor = System.Drawing.Color.Yellow;

        }

        private void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {
            // Quitamos los handlers
```

```
            iKit.Attach -= new AttachEventHandler(iKit_ConectarHandler);
            iKit.Detach -= new DetachEventHandler(iKit_Desconectar
Handler);
            iKit.Error -= new ErrorEventHandler(iKit_ErrorHandler);

            iKit.InputChange -= new InputChangeEventHandler(iKit_Digital
Handle);
            iKit.SensorChange -= new SensorChangeEventHandler
(iKit_AnalogoHandle);

            Application.DoEvents();

            // Cerramos el phidget
            iKit.close();

        }

        public void iKit_DigitalHandle(object sender, InputChangeEventArgs e)
        {
            // Verificamos que estemos conectados
            if (conectado)
            {
                // Verificamos que sea la entrada digital 0
                if (e.Index == 0)
                {
                    // Verificamos el valor de la entrada
                    if (e.Value == true)
                        chbDigital0.Checked = true;
                    else
                        chbDigital0.Checked = false;
                }
            }
        }

        public void iKit_AnalogoHandle(object sender, SensorChangeEventArgs e)
        {
            // Verificamos que estemos conectados
            if (conectado)
            {
                // Verificamos que sea la entrada 0
```

```
            if (e.Index == 0)

            {
                // Colocamos el valor
                lblAnalogico0.Text = e.Value.ToString();
            }

            // Verificamos que sea la entrada 1
            if (e.Index == 1)
            {
                // Colocamos el valor
                lblAnalogico1.Text = e.Value.ToString();
            }

        }
    }

    private void chbSalida7_CheckedChanged(object sender, EventArgs e)
    {
        // Verificamos que estemos conectados
        if (conectado)
        {
            // Verificamos el valor del checkbox
            if (chbSalida7.Checked == true)
                iKit.outputs[7] = true; // Prendemos el LED
            else
                iKit.outputs[7] = false; // Apagamos el LED
        }
    }

    private void trkPosicion_Scroll(object sender, EventArgs e)
    {
        lblPosicion.Text = trkPosicion.Value.ToString();
    }
```

Una vez que el programa se encuentre en ejecución y presionemos los diferentes **RadioButtons**, la dirección de las ruedas deberá cambiar de acuerdo con nuestras órdenes. La forma más sencilla de hacer esto es creando el handler para el evento **CheckedChanged** de cada uno de los **RadioButtons**. Dentro del handler, ingresamos las salidas digitales según sea necesario para cada cambio de dirección.

```
private void rbtnAdelante_CheckedChanged(object sender, EventArgs e)
{
    if (rbtnAdelante.Checked == true && conectado==true)
    {
        // Colocamos los motores en modo de avance
        iKit.outputs[0] = false;
        iKit.outputs[1] = false;

        iKit.outputs[2] = false;
        iKit.outputs[3] = false;

        Thread.Sleep(250);

    }
}

private void rbtnDerecha_CheckedChanged(object sender, EventArgs e)
{
    if (rbtnDerecha.Checked == true && conectado == true)
    {
        // Colocamos los motores en modo de derecha
        iKit.outputs[0] = false;
        iKit.outputs[1] = false;

        iKit.outputs[2] = true;
        iKit.outputs[3] = true;

        Thread.Sleep(250);
    }
}

private void rbtnAtras_CheckedChanged(object sender, EventArgs e)
{
    if (rbtnAtras.Checked == true && conectado == true)
    {
        // Colocamos los motores en modo de atras
        iKit.outputs[0] = true;
        iKit.outputs[1] = true;

        iKit.outputs[2] = true;
```

```
                iKit.outputs[3] = true;

                Thread.Sleep(250);
            }
        }

        private void rbtnIzquierda_CheckedChanged(object sender, EventArgs e)
        {
            if (rbtnIzquierda.Checked == true && conectado == true)
            {
                // Colocamos los motores en modo de izquierda
                iKit.outputs[0] = true;
                iKit.outputs[1] = true;

                iKit.outputs[2] = false;
                iKit.outputs[3] = false;

                Thread.Sleep(250);
            }
        }


    }
}
```

Con esto, estamos listos para compilar el programa. Es necesario conectar los cables rojo y negro de los relés a una fuente de poder que tenga 5V para poder alimentar de manera correcta a los motores. De esta forma, ya podemos probar el programa y el control de los motores.

... RESUMEN

En este capítulo, vimos que es posible diseñar una plataforma para robots mediante materiales baratos y de fácil utilización. Hemos usado tubos de PVC y cartón rígido para crear una plataforma de dos niveles. En el primer piso, hay espacio suficiente para motores, baterías y componentes electrónicos, como los Phidgets y los sensores. El segundo piso puede tener una computadora laptop que lleve a cabo la lógica y el control de los Phidgets. Para controlar los motores y la dirección del robot, es posible utilizar relés que sean controlados por el software del sistema.

ACTIVIDADES

TEST DE AUTOEVALUACIÓN

1. Adicione otro par de relés para poder apagar o encender los motores.

2. Agregue potenciómetros al circuito para modificar la velocidad a la que giran los motores.

3. Instale los motores reales en la plataforma y pruebe el control por medio de la computadora.

4. Añada luces a la plataforma.

5. Diseñe una plataforma propia con cuatro ruedas y tracción en cada una de ellas.

6. Diseñe una plataforma que funcione para exteriores y todo tipo de clima.

7. Adicione el sistema de sonar para que la plataforma decida hacia dónde dirigirse.

8. Use un joystick para controlar el movimiento de la plataforma.

9. Use el sonar para que la plataforma no colisione con los objetos que tiene enfrente.

10. Modifique la rueda de patín para que un servo la mueva y, de esa forma, controle la dirección de la plataforma.

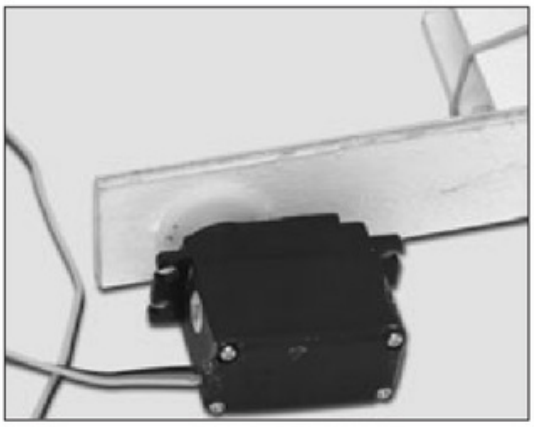
11. Use switches para que la plataforma detecte colisiones laterales.

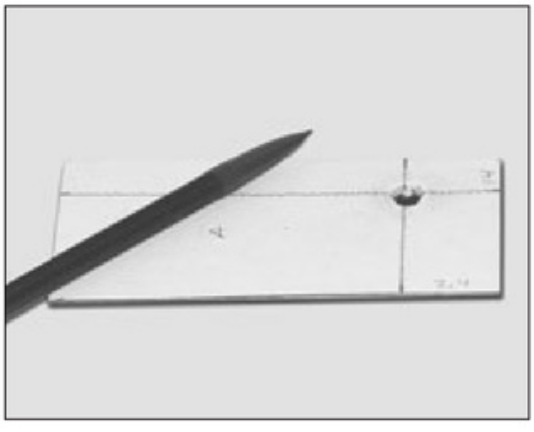
12. Modifique la plataforma para computadoras tipo Palm.

Servicios al lector

En esta última sección confeccionamos un índice que nos permitirá encontrar de forma más sencilla los temas que necesitemos. Además, ponemos a disposición un listado de sitios recomendados que nos ayudarán a mantenernos actualizados y a aumentar nuestros conocimientos.

ÍNDICE TEMÁTICO



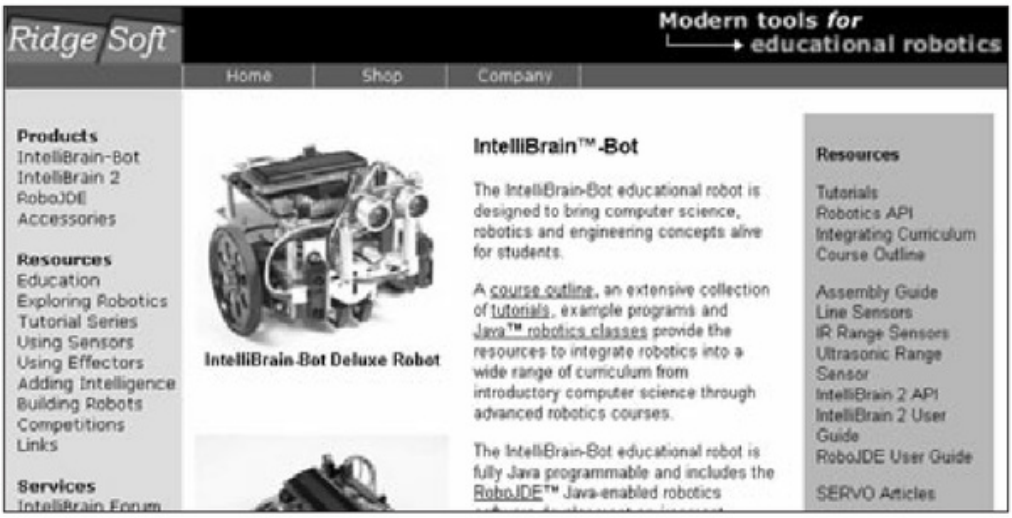


LISTADO DE SITIOS

IntelliBrain-Bot

www.ridgesoft.com/intellibrainbot/intellibrainbot.htm

Si deseamos continuar experimentando con los robots podemos hacer uso de **IntelliBrain-Bot**. Este pequeño robot tiene todos los elementos necesarios para ser una buena herramienta educativa relacionada con los robots y la inteligencia artificial. El sitio web presenta muchos recursos como tutoriales y diseños completos de cursos.



Simulador de robots Actin

www.energid.com/products-actin.htm

Algunas veces es necesario llevar a cabo simulaciones de los robots antes de comenzar a construirlos. En robots profesionales esto puede llevarnos a ahorrar costos y tiempos de desarrollo. Podemos utilizar algún software de simulación robótica como Actin. Este software se basa en software utilizado por la **NASA**.

SmartRobots

www.smartrobots.com/

La empresa **SmartRobots** fabrica plataformas y otros productos robóticos. Estas plataformas son de alta tecnología y sus precios no son bajos pero funcionan muy bien como base de desarrollo para robots más avanzados e investigación en robótica. También tienen un brazo robótico e investigación en redes de robots comunicados por medio de Internet.

Cool robot of the week

http://ranier.hq.nasa.gov/telerobotics_page/coolrobots.html

Un sitio web interesante que nos provee la **NASA** es Cool Robot of the Week, en donde se presenta cada semana un sitio web relacionado con los robots. Como condición, el sitio web debe brindar información innovadora y útil. Este sitio es un muy buen lugar para aprender sobre los robots y obtener inspiración para nuestros propios proyectos.

Robot marino

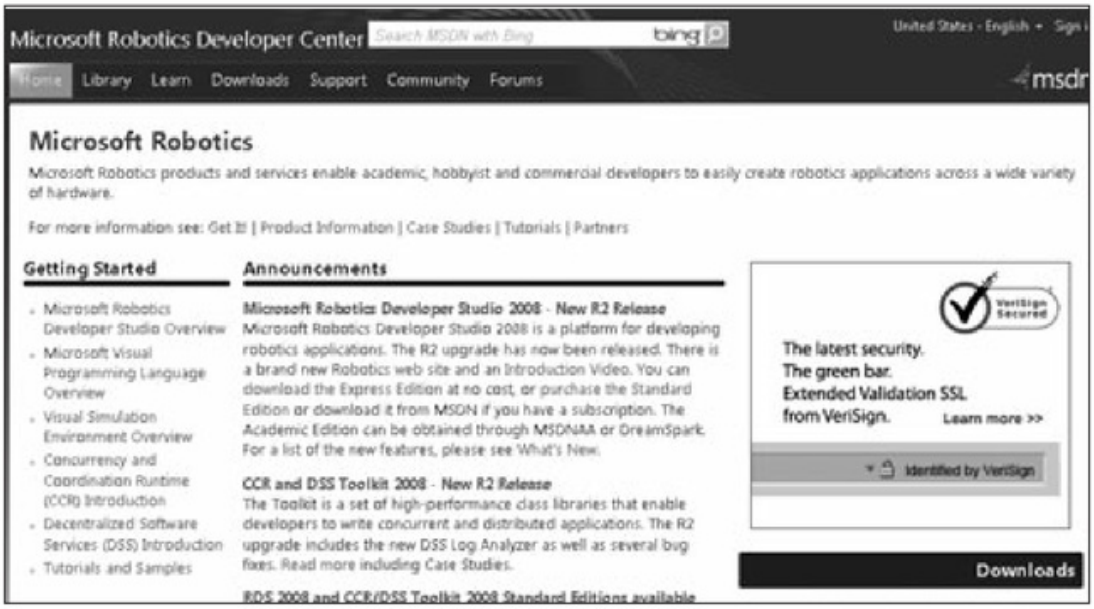
www.apl.washington.edu/projects/seaglider/summary.html

La mayor parte de los robots que hemos visto se mueven en la tierra e incluso algunos pueden volar, pero también existen robots que se pueden mover en el agua. Estos robots se usan principalmente con fines de investigación científica o militar. Uno de estos robots es el **Seaglider**, muy eficiente en términos de movimiento.

Microsoft Robotics Studio

http://msdn.microsoft.com/en-us/robotics/default.aspx

Otro software con el que podemos desarrollar robots y controlar su lógica es **Microsoft Robotics Studio**. Este software no solamente tiene herramientas de desarrollo, también nos da herramientas para llevar a cabo simulaciones. Tiene un lenguaje de programación visual muy sencillo que ayuda a desarrollar las aplicaciones en poco tiempo.



Lego Mindstorms

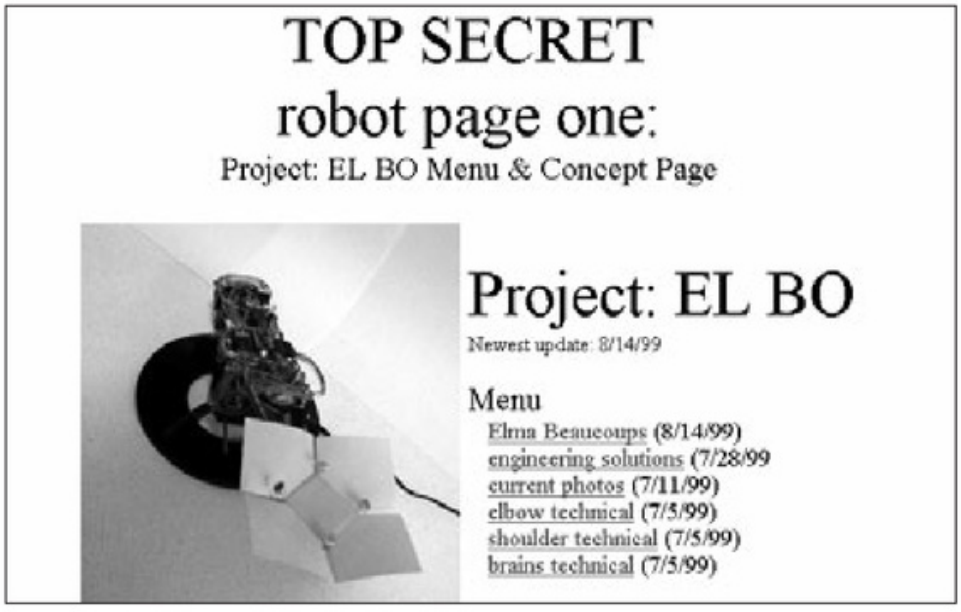
http://mindstorms.lego.com/en-us/Default.aspx

Si deseamos acceder a una forma más sencilla de construir robots podemos recurrir a los **Mindstorms**. Estos kits robóticos fabricados por Lego presentan la facilidad de construcción de los kits Lego con una forma sencilla de programarlos. Estos kits contienen servos y sensores, y además el armado mecánico es muy sencillo.

EL BO

www.bentmachine.com/robot/elma/robot.html

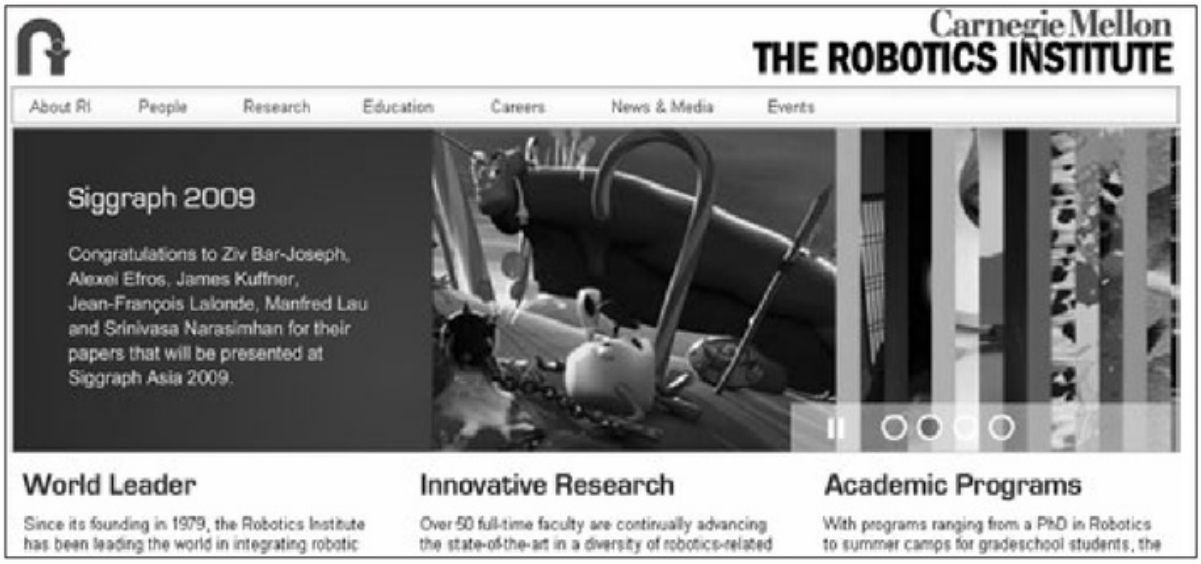
No todos los robots son creados por las grandes empresas o los gobiernos, muchos son creados por aficionados. Estos robots presentan conceptos novedosos. La tecnología en algunos casos no es muy avanzada, pero las ideas tienden a ser muy buenas. Uno de estos robots es **EL BO**, un robot que presenta un efecto fototrópico.



Robots en Carnegie Mellon

www.ri.cmu.edu/

Una de las instituciones más importantes y de mayor prestigio en el mundo de la robótica es el **Instituto de Robots Carnegie Mellon**. Aquí se concentran 50 de los científicos en robótica más importantes del mundo, que desarrollan y dan cursos sobre robots en todas las áreas, incluyendo robots para misiones espaciales.



FIRST

www.usfirst.org/

Una de las organizaciones que más han ayudado a llevar la ciencia y la robótica a personas de todas las edades, y más en particular a los niños, es **FIRST**. Organizan diferentes eventos y competencias relacionados con los robots. Además, la mayor parte de los robots son construidos con los sistemas de LEGO.

Lynxmotion

www.lynxmotion.com/

Un sitio web donde podemos conseguir muchos kits interesantes de robots es Lynxmotion. Sus kits nos pueden servir de base para proyectos más avanzados. También podemos encontrar diferentes productos que nos pueden ayudar a desarrollar más fácilmente nuestros robots tanto a nivel de software como hardware.

Arrick Robotics

www.arrickrobotics.com/robomenu/

Podemos crear muchos proyectos robóticos y mostrarlos al mundo. Lo podemos hacer en diferentes sitios, pero uno de los mejores es el menú de robots de Arrick Robotics. En este sitio muchas personas dan a conocer sus proyectos robóticos. Además, podemos subir nuestros propios proyectos al sitio.

CLAVES PARA COMPRAR
UN LIBRO DE COMPUTACIÓN

1 SOBRE EL AUTOR Y LA EDITORIAL

Revise que haya un cuadro "sobre el autor", en el que se informe sobre su experiencia en el tema. En cuanto a la editorial, es conveniente que sea especializada en computación.

2 PRESTE ATENCIÓN AL DISEÑO

Compruebe que el libro tenga guías visuales, explicaciones paso a paso, recuadros con información adicional y gran cantidad de pantallas. Su lectura será más ágil y atractiva que la de un libro de puro texto.

3 COMPARE PRECIOS

Suele haber grandes diferencias de precio entre libros del mismo tema; si no tiene el valor en tapa, pregunte y compare.

4 ¿TIENE VALORES AGREGADOS?

Desde un sitio exclusivo en la Red hasta un CD-ROM, desde un Servicio de Atención al Lector hasta la posibilidad de leer el sumario en la Web para evaluar con tranquilidad la compra, o la presencia de adecuados índices temáticos, todo suma al valor de un buen libro.

5 VERIFIQUE EL IDIOMA

No sólo el del texto; también revise que las pantallas incluidas en el libro estén en el mismo idioma del programa que usted utiliza.

6 REVISE LA FECHA DE PUBLICACIÓN

Está en letra pequeña en las primeras páginas; si es un libro traducido, la que vale es la fecha de la edición original.

¡Léalo antes Gratis!

En nuestro sitio, obtenga GRATIS un capítulo del libro de su elección antes de comprarlo.

Vista avanzado

Este manual es una pieza imprescindible para convertirnos en administradores expertos de este popular sistema operativo. En sus páginas haremos un recorrido por las herramientas fundamentales para tener máximo control sobre todo lo que sucede en nuestra PC.

→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-663-007-8

101 Secretos de Excel

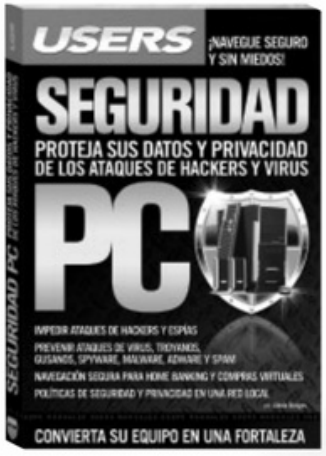
Una obra absolutamente increíble, con los mejores 101 secretos para dominar el programa más importante de Office. En sus páginas encontraremos un material sin desperdicios que nos permitirá realizar las tareas más complejas de manera sencilla.

→ COLECCIÓN: MANUALES USERS
→ 336 páginas / ISBN 978-987-663-005-4

Electrónica & microcontroladores PIC

Una obra ideal para quienes desean aprovechar al máximo las aplicaciones prácticas de los microcontroladores PIC y entender su funcionamiento. Un material con procedimientos paso a paso y guías visuales, para crear proyectos sin límites.

→ COLECCIÓN: MANUALES USERS
→ 368 páginas / ISBN 978-987-663-002-3



Seguridad PC

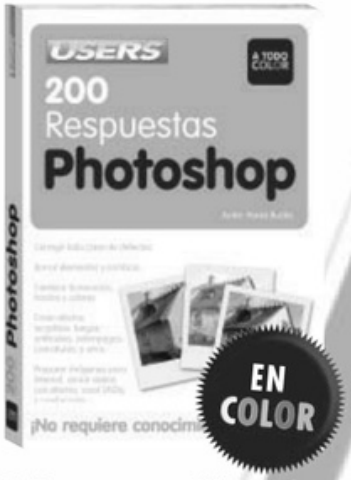
Este libro contiene un material imprescindible para proteger nuestra información y privacidad. Aprenderemos cómo reconocer los síntomas de infección, las medidas de prevención a tomar, y finalmente, la manera de solucionar los problemas.

→ COLECCIÓN: MANUALES USERS
→ 336 páginas / ISBN 978-987-663-004-7

Hardware desde cero

Este libro brinda las herramientas necesarias para entender de manera amena, simple y ordenada cómo funcionan el hardware y el software de la PC. Está destinado a usuarios que quieran independizarse de los especialistas necesarios para armar y actualizar un equipo.

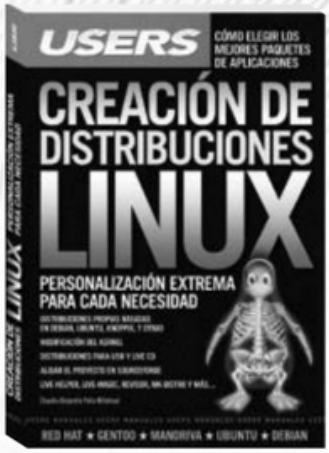
→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-663-001-6



200 Respuestas: Photoshop

Esta obra es una guía que responde, en forma visual y práctica, a todas las preguntas que necesitamos contestar para conocer y dominar Photoshop CS3. Definiciones, consejos, claves y secretos, explicados de manera clara, sencilla y didáctica.

→ COLECCIÓN: 200 RESPUESTAS
→ 320 páginas / ISBN 978-987-1347-98-8



Creación de distribuciones Linux

En este libro recorreremos todas las alternativas para crear distribuciones personalizadas: desde las más sencillas y menos customizables, hasta las más avanzadas, que nos permitirán modificar el corazón mismo del sistema, el kernel.

→ COLECCIÓN: MANUALES USERS
→ 336 páginas / ISBN 978-987-1347-99-5

Métodos ágiles

Este libro presenta una alternativa competitiva a las formas tradicionales de desarrollo y los últimos avances en cuanto a la producción de software. Ideal para quienes sientan que las técnicas actuales les resultan insuficientes para alcanzar metas de tiempo y calidad.

→ COLECCIÓN: DESARROLLADORES
→ 336 páginas / ISBN 978-987-1347-97-1

SuperBlogger

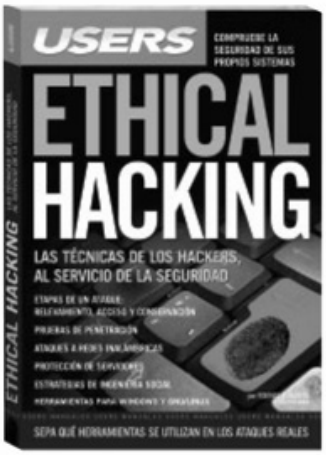
Esta obra es una guía para sumarse a la revolución de los contenidos digitales. En sus páginas, aprenderemos a crear un blog, y profundizaremos en su diseño, administración, promoción y en las diversas maneras de obtener dinero gracias a Internet.

→ COLECCIÓN: MANUALES USERS
→ 352 páginas / ISBN 978-987-1347-96-4

UML

Este libro es la guía adecuada para iniciarse en el mundo del modelado. Conoceremos todos los constructores y elementos necesarios para comprender la construcción de modelos y razonarlos de manera que reflejen los comportamientos de los sistemas.

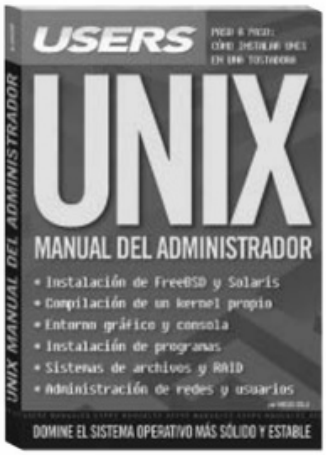
→ COLECCIÓN: DESARROLLADORES
→ 320 páginas / ISBN 978-987-1347-95-7



Ethical Hacking

Esta obra expone una visión global de las técnicas que los hackers maliciosos utilizan en la actualidad para conseguir sus objetivos. Es una guía fundamental para obtener sistemas seguros y dominar las herramientas que permiten lograrlo.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-1347-93-3



Unix

Esta obra contiene un material imperdible, que nos permitirá dominar el sistema operativo más sólido, estable, confiable y seguro de la actualidad. En sus páginas encontraremos las claves para convertirnos en expertos administradores de FreeBSD.

→ COLECCIÓN: MANUALES USERS
→ 320 páginas / ISBN 978-987-1347-94-0

LLEVE SU PASIÓN A LA PRÁCTICA

En el interior de este libro encontraremos un recorrido teórico y práctico por cada uno de los conceptos fundamentales de la robótica, explicando en detalle cuáles son las partes que componen un robot, y los pasos que se deben seguir para armarlo y programarlo a gusto.

» HARDWARE
» 288 PÁGINAS
» ISBN 978-987-1347-56-8

USERS

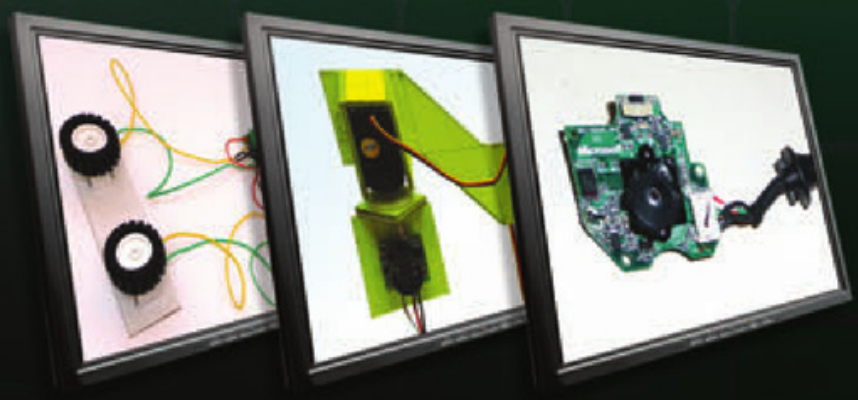
CONTENIDO

NIVEL DE USUARIO

PRINCIPIANTE | INTERMEDIO | AVANZADO | EXPERTO

ROBÓTICA AVANZADA

La robótica está cada vez más presente en nuestras vidas, con un uso cada vez más extendido en todo el mundo, y sus principios no son tan complicados ni están tan alejados de nuestra realidad. Este libro contiene toda la información necesaria para armar robots complejos con elementos caseros. En sus páginas también encontraremos los principios de programación necesarios para redactar en C# las líneas de código requeridas para dar vida a nuestras creaciones, por medio de interfaces gráficas profesionales. Comprenderemos cómo utilizar el puerto USB de la PC como medio de comunicación con el microcontrolador Phidget 8/8/8 e, incluso, de qué manera observar el mundo que ve nuestro robot a través de nuestro propio monitor, o tomar mediciones de distancia repecto de otros objetos por medio de su sonar, para luego guardar los datos en nuestro disco duro. Una obra ideal para todos los fanáticos, hobbystas y técnicos que deseen adentrarse en el fascinante mundo de la automatización y los robots.



redusers.com

En este sitio encontrará una gran variedad de recursos y software relacionado, que le servirán como complemento al contenido del libro. Además, tendrá la posibilidad de estar en contacto con los editores, y de participar del foro de lectores, en donde podrá intercambiar opiniones y experiencias.

Si desea más información sobre el libro puede comunicarse con nuestro Servicio de Atención al Lector: usershop@redusers.com

ADVANCED ROBOTICS

This book contains the necessary skills to build our own robots out of ordinary materials. It will help us develop the full set of code lines that will control each component. Using the Phidget 8/8/8 we will gain access to complex functions, such as distance measurement and camera control.

ISBN 978-987-663-020-7

MANUALES USERS MANUALES USERS MANUAL

¡APLIQUE TODO EL PODER DE C# A SUS CREACIONES!