Universidad Austral de Chile
Escuela de Mecánica

Desarrollo de experiencias de laboratorio
Usando el prototipo Hunter ll

Trabajo para optar al Título de:
Ingeniero Mecánico

Profesor Patrocinante:
Héctor Noriega Fernández
Ing. (E) Mecánico
Dr. Sc. Ing. Producción

Luis Eduardo Alvarez Sandoval
Cristián Moisés Gallardo Gallardo

Valdivia - Chile

2006

Profesor patrocinante

Sr.
Héctor Noriega Fernández
Ingeniero (E) Mecánico
Dr. Sc. Ing. Producción _____

Profesores Informantes

Sr.
Luis Loncomilla Igor
Ingeniero Mecánico _____

Sr.
Misael Fuentes Paredes
Ingeniero Mecánico _____

Director de Escuela

Sr.
Rogelio Moreno
Ingeniero Civil Mecánico _____

ÍNDICE

CAPITULO 4

DESARROLLO DE EXPERIENCIA DE LABORATORIO 2

CAPITULO 5

DESARROLLO DE EXPERIENCIA DE LABORATORIO 3

CAPITULO 6

DESARROLLO DE ESPERIENCIA DE LABORATORIO 4

AGRADECIMIENTOS CRISTIAN GALLARDO

Sin duda el paso que e dado a través del presente proyecto es el sello a un largo camino de estudio, esfuerzo y superación, el creer que a través de los estudios podía lograr tener éxito en la vida, dar una satisfacción a mis padres y hermanos, pilar fundamental para lograr la tan esperada meta, ser un Ingeniero Mecánico.

En estas líneas quiero agradecer en primer lugar a Dios y mis padres, Elena y Moisés, a mis hermanos Claudio y Marco, ellos que siempre han estado cuando los he necesitado, también recordar a mi herma Maritza de la que estoy seguro también me ayudo desde el cielo, tal vez con esos milagros que pasan en los momentos más difíciles a los que de pronto te ves enfrentado.

Agradezco a todo el cuerpo docente que contribuyo en mi formación, en especial al profesor Luis Loncomilla, por creer en nosotros y en nuestro proyecto, a nuestro Profesor Patrocinante Crispín por su paciencia y apoyo pese a todos nuestros atrasos.

A don Álvaro Uribe gran amigo, gracias de corazón por todos los buenos consejos brindados durante el tiempo que trabajamos juntos, a ti te debo en gran parte lo que he logrado.

A Pancho mi profesor particular sin remuneración, gracias amigo por todas esas clases que no solo me ayudaron en la universidad, sino que además a conocer una gran persona y profesional.

A todos mis compañeros y amigos en especial Ronald y Eduardo con quienes compartí muchos momentos divertidos en la universidad.

Dedicado a mis amados hijos “Josefa y Arturo”.

AGRADECIMIENTOS LUIS ALVAREZ

Cuando un hombre camina en dirección a su destino, se ve forzado muchas veces a cambiar su rumbo. Otras veces las circunstancias externas son más fuertes, y se ve obligado a acobardarse y ceder. Todo eso forma parte del aprendizaje. Pero nadie puede perder de vista lo que quiere. Aunque en algunos momentos piense que el mundo y los demás son más fuertes. El secreto es éste: no desistir...

Con todo mi corazón agradezco la fortaleza necesaria que Dios me ha brindado como también de la sabiduría que recibí de quienes con dedicación entregaron lo necesario para mi formación como ingeniero así de esta misma forma agradezco a quienes han sido una voz de aliento constante y que en los peores momentos fueron la única motivación para seguir adelante.

Mis agradecimientos principalmente están dirigidos a mis padres Maria y Luis, por sus sacrificios y sus desvelos, por su gran apoyo, su cariño y su preocupación durante todo este proceso y agradecerles simplemente por haberme dado la vida.

También quisiera reconocer y agradecer conjuntamente a todos mis profesores que participaron en mi formación profesional y muy especialmente a mi profesor patrocinante Señor Hector Noriega y al Señor Luis Loncomilla que en su gestión como director de escuela creyó en nosotros y posibilito el financiamiento de nuestro proyecto.

Dedicado con cariño a Carolina y a mi hijo Javier.

RESUMEN

El presente proyecto se basa en probar una hipótesis, la cual dice que estudiantes de ingeniería mecánica son capaces de realizar experiencias de laboratorio en robótica básica, usando como medio educativo un prototipo llamado Hunter II.

Hunter II es un vehículo explorador autómata capaz de interactuar con el medio ambiente a través de sensores, los cuales se retroalimentan de este y envían señales a la Basic Stamp que en el fondo es el cerebro del vehículo.

La primera parte de este proyecto se compone de información bibliográfica que representa el marco teórico para la programación y desarrollo de experiencias de laboratorio con el prototipo Hunter II, seguido a esto se comienza con una serie de experiencias basadas en rutinas de movimiento y navegación, la idea es que el estudiante aprenda a programar y armonizar tanto actuadotes como sensores.

De forma concluyente en el proyecto se realiza una experiencia en la cual el Hunter II no será usado como vehículo tal, sino que sus componentes casi en su totalidad serán usados para simular la robotización de un proceso productivo representado en una Maqueta, la cual simula una línea de transporte de madera empaquetada en un aserradero.

SUMMARY

The present project is based in to testing a hypothesis, which says that students of mechanical engineering are capable of carrying out experiences of laboratory in robotic basic, using as educational resource a called prototype Hunter II.

Hunter II is an automaton explorer vehicle, capable to interact with the environment through critics, which itself feedback of this and send signs to the Basic Stamp that in the fund is the brain of the vehicle.

The first part of this project is composed of bibliographical information that represents the theoretical framework for the programming and laboratory experiences development with the prototype Hunter II, followed to this begins with a series of experiences based on routines of movement and navigation, the idea is that student learn to plan and to harmonize as much actuator and sensors.

Of conclusive form in the project an experience is carried out in which the Hunter II will not be used as vehicle such, but its components almost in its totality will be used to simulate the robotics systems of a productive process represented in a Model, which simulates a packaged wood transportation line, which is the solution to a problem in the sawmills.

INTRODUCCION.

A través del tiempo, el hombre ha desarrollado una nueva forma de interpretar inteligencia artificial, la primera palabra que se nos viene a la mente cuando hablamos de inteligencia artificial es robot, o bien palabras asociadas a esta como lo es robótica. El término "robot" se debe a Karel Capek, quien lo utilizó en 1917 por primera vez, para denominar a unas máquinas construidas por el hombre y dotadas de inteligencia, esta palabra deriva del termino "robotnik", el cual define al esclavo del trabajo. Los robots llevan más de 50 años en operación, su origen se remonta a los años 1950, cuando la invención de los transistores y de los circuitos integrados posibilito su desarrollo tecnológico, hoy por hoy existe una gran variedad de robots que cuentan con una variada gama de programas y funciones que van desde robots de uso doméstico, ayuda médica, de uso industrial, trabajos de alto peligro, o misiones que son imposibles de realizar por un ser humano, así como lo es por ejemplo caminar sobre la superficie del planeta Marte como lo hizo un robot explorador altamente sofisticado con programas similares a los del Hunter II.

La ciencia encargada de estudiar y crear estos equipos y programas para el hombre, es la robótica que se define como ciencia o rama que se ocupa del estudio, desarrollo y aplicaciones de los robots. Los robots son dispositivos compuestos de sensores que reciben datos del ambiente para interactuar con este de acuerdo a la retroalimentación que reciba de la computadora, puede ser que los propios robots dispongan de microprocesadores que reciben una señal de entrada o input de los sensores, y que estos microprocesadores ordenen al robot la ejecución de las acciones para las cuales fueron concebidos. En este caso, el propio robot es a su vez una computadora como ocurre por ejemplo en el caso de vehículos exploradores robotizados, como lo es el Hunter II, el cual es un prototipo que navega por el medio terrestre retroalimentándose de información que capta del ambiente a través de sensores conectados a su microprocesador. En general, un sensor mide una característica del ambiente o espacio en el cual se esta posicionado transformando esta señal en otra eléctrica que envía al procesador, en cierta forma los sensores son para el robot lo que para nosotros son los sentidos.

1.1 Problema

- La formación de futuros ingenieros mecánicos requiere estudios de robótica, debido a su creciente aplicación en la industria moderna.
- En la formación profesional se deben adquirir conocimientos básicos sobre robótica, ya sean en conceptos de programación como de experimentación práctica con prototipos.

1.2 Problemática

- Falta de conocimientos en lo concerniente a investigaciones prácticas en temas de tecnología actual como lo es la robótica.
- Falta de conocimientos en el campo de la robótica impide abordar tecnologías que hoy en día son indispensables en el desempeño laboral de los ingenieros mecánicos.

1.3 Hipótesis

- Con el prototipo Hunter II se podrán realizar experiencias de laboratorio para los alumnos de Ingeniería Mecánica, que permitan adquirir los conocimientos para realizar una simulación de un proceso productivo.

1.4 Objetivos generales

- Desarrollar experiencias de laboratorio, para los alumnos de la carrera de Ingeniería Mecánica, las cuales les permitirán programar un vehículo robotizado que interactúe con el ambiente o simular un proceso productivo operado por sensores.

1.5 Objetivos específicos

- Registrar e ilustrar conceptos tecnológicos de robótica.
- Armar y programar el Hunter II, un vehículo robotizado.
- Realizar experiencias prácticas con el prototipo.
- Acercar al estudiante de Ingeniería Mecánica al mundo moderno de la robótica, conocer nuevos conceptos y aplicaciones a través del desarrollo de guías de laboratorio, donde el alumno de Ingeniería Mecánica pueda encontrar un referente de conocimiento, para realizar futuras experiencias.
- Robotización a escala de parte de un proceso productivo maderero.

1.6 Metodología de trabajo

- Para registrar e ilustrar conceptos tecnológicos de robótica es necesario recopilar literatura tanto de Internet como en revistas de tecnología, además es necesario complementar todos estos conceptos adquiridos con investigaciones prácticas y laboratorios desarrollados con el Hunter II, como también entrevistarse con profesionales que se desenvuelvan en temas relacionados con robótica y procesos productivos automatizados.
- Para el armado y programación del Hunter II es necesario armar la estructura del vehículo y montar sobre la protoboard. Luego de esta primera experiencia, se deben calibrar los servomecanismos de manera mecánica, y luego por software de forma iterativa hasta lograr la puesta a punto de estos.
- Una vez puesto en funcionamiento el Hunter II, se procede a conectar los diversos programas de navegación a la Basic Stamp para poder así realizar las diversas experiencias de laboratorio con sus respectivas guías de trabajo.
- Finalmente se procede a desarmar el hunter II, con el fin de demostrar que con su misma tecnología y componentes es posible simular un proceso productivo robotizado, acercándolo más aun a la ingeniería Mecánica.

Capítulo 2 – REVISION BIBLIOGRAFICA.

2.1 Introducción

En el presente capítulo se dan a conocer conceptos e información bibliográfica que representan el marco teórico para la programación y desarrollo de experiencias de laboratorio con el prototipo Hunter II. El Hunter II es un robot de tipo educacional que fue financiado por la escuela de mecánica y que estará disponible para que estudiantes puedan repetir las experiencias realizadas en este trabajo.

2.2 Mecátronica

Acuñada en 1969 por el ingeniero japonés Yakasawa, la palabra mecatrónica ha sido definida de varias maneras. Un consenso común es describir a la mecatrónica como una disciplina integradora de las áreas de mecánica, electrónica e informática cuyo objetivo es proporcionar mejores productos, procesos y sistemas. La mecatrónica no es, por tanto, una nueva rama de la ingeniería, sino un concepto recientemente desarrollado que enfatiza la necesidad de integración y de una interacción intensiva entre diferentes áreas de la ingeniería.

Con base en lo anterior, se puede hacer referencia a la definición de mecatrónica propuesta por J.A. Rietdijk: "Mecatrónica es la combinación sinérgica de la ingeniería mecánica de precisión, de la electrónica, del control automático y de los sistemas para el diseño de productos y procesos". Existen, claro está, otras versiones de esta definición, pero ésta claramente enfatiza que la mecatrónica está dirigida a las aplicaciones y al diseño de equipos robotizados o autómatas.

Un sistema mecatrónico típico recoge señales, las procesa y, como salida, genera fuerzas y movimientos. Los sistemas mecánicos son entonces extendidos e integrados con sensores, microprocesadores y controladores. Los robots, las máquinas controladas digitalmente, los vehículos guiados automáticamente, las cámaras electrónicas, las máquinas de telefax y las fotocopiadoras pueden considerarse como productos mecatrónicos. Al aplicar una filosofía de integración en el diseño de productos y sistemas se obtienen ventajas importantes como son mayor flexibilidad, versatilidad, nivel de inteligencia de los productos, seguridad y confiabilidad así como un bajo consumo de energía.

2.3 Tipos de robots

Según **Angulo J.M y Avilés R. (2001),** existe una amplia gama de robots cuya utilidad depende de las aplicaciones para las cuales fueron diseñados, es así como a lo largo de la historia se ha dado lugar a muchos tipos de clasificaciones posibles, las cuales la mayor parte de las veces no son rigurosas, y se van modificando conforme avanza la tecnología.

Clasificación general, de acuerdo a la utilidad social de los robots

- Robots Industriales.
- Robots de control remoto (Terrestres, Marinos, Aéreos, espaciales).
- Prótesis para uso humano.
- Robots didácticos (Estáticos y Móviles).
- Robots de Juguete.

Los **robots industriales** son los que más aplicación útil han tenido para la sociedad, visto desde el punto de vista práctico, ya que los productos que ellos fabrican por lo general salen para consumo masivo de la humanidad. Estos también son llamados manipuladores, realizan tareas repetitivas y se emplean en gran escala en la industria automotriz, en la electrónica y en otras, donde se utilizan para armar o ensamblar automáticamente los respectivos productos, taladran, ponen componentes, los ajustan, sueldan, pintan, transportan piezas, etc. Generalmente tienen la forma de un brazo mecánico donde se conecta en su extremo la herramienta que sea necesaria.

Los **vehículos de control remoto** pueden ser clasificados dentro de la categoría de robots y se utilizan para movilizar herramientas o instrumentos en los sitios donde el hombre no puede acceder debido a las condiciones físicas o climáticas del lugar siendo los espaciales los más sofisticados. Podemos citar como ejemplos los robots que se emplean para construir túneles, apagar incendios, los militares, los misiles teledirigidos, los vehículos espaciales teledirigidos o autónomos que permiten recorrer la superficie de un planeta o satélites, los que tienden cables submarinos, los que exploran el fondo del mar dirigidos desde un barco, etc.

Las **prótesis para uso humano** también pueden considerarse como robots, ya que reemplazan funciones en los miembros inferiores y superiores de los seres humanos. Se han desarrollado verdaderas obras de arte en aparatos electromecánicos y electrónicos que realizan en forma parecida el trabajo de las manos con sus dedos y las piernas.

Los **robots didácticos o experimentales** están dedicados a la enseñanza y aprendizaje de la robótica, y no cumplen una tarea específica como tal. Generalmente tienen la forma de un brazo mecánico que imita la forma humana o de los robots industriales. Básicamente podemos decir que hay dos tipos de robots didácticos: los estáticos, que van sobre una base fija, y los móviles, que van montados sobre una plataforma que se puede desplazar sobre una superficie lisa.

Como su nombre lo indica, los **robots de juguete** son dispositivos generalmente fabricados en serie, y que imitan o inclusive cumplen algunas funciones similares a las de los robots didácticos o experimentales, y algunas veces se confunden con ellos. Algunos tienen un control remoto, otros funcionan de forma autónoma y otros tienen una interfase a una computadora.

Los **robots de uso casero** son uno de los grandes sueños de la humanidad, ya que con ellos se espera lograr el ayudante perfecto para las tareas domésticas que tanto nos aburren a diario. Este tipo de robot debe tener libre movimiento, es decir no debe estar conectado a un control externo, y por lo tanto tiene su propio sistema de control, podría pensarse en ellos para que limpien, nos preparen y sirvan alimentos, transporten objetos (la basura), etc.

Hay otros tipos de robots, evidentemente, que no se pueden clasificar en las categorías mencionadas, y que tienen diferentes aplicaciones, como las manos teledirigidas que sirven para trabajar con productos radioactivos o peligrosos, o las plataformas automatizadas para el manejo de mercancías en bodegas o libros en bibliotecas, etc., **Angulo J.M y Avilés R. (2001)**.

La siguiente clasificación es vista desde el punto de vista del robot industrial como eje de la misma:

- Manipuladores (Manual, De Secuencia fija, De Secuencia variable).
- Robots de repetición o Aprendizaje.

- Robots controlados por computadora.
- Robots inteligentes.
- Micro-robots.

Los **Manipuladores** son sistemas mecánicos multifuncionales, con un sistema de control simple y se emplean en tareas sencillas y repetitivas, si el movimiento del robot es controlado directamente por el operador humano se dice que es un Manipulador Manual, si, en cambio, el proceso, preparado previamente, se repite de forma invariable se dice que es un Manipulador de Secuencia fija y si se pueden alterar algunas de las características del ciclo de trabajo se dice que es de Manipulador de Secuencia variable.

Un **robot de repetición o aprendizaje** es un manipulador que repite una secuencia de movimientos que fueron previamente ejecutados por un operador humano, haciendo uso de un dispositivo controlador manual. En la actualidad es el que más se utiliza en la industria y recibe el nombre de gestual.

Robot controlado por computadora. En este tipo de robot se precisa de un lenguaje de programación específico con el cual se desarrolla el programa al que responderá la máquina. Este lenguaje se compone de instrucciones a las que responde el robot, y el programa corre en la computadora. A este tipo de programación del robot se la llama textual.

Los **robots inteligentes** son similares a los controlados por computadora, pero a diferencia de éstos, tienen la capacidad de relacionarse con el mundo real (el mundo que les rodea) a través de sensores apropiados y tomar decisiones adecuadas a las circunstancias en tiempo real. Se dice que son autoprogramables.

Los **robots educativos o microrobots**, de entrenamiento o investigación poseen una estructura general y funcionamiento son similares a los de un robot industrial, algunos son autónomos y otros funcionan si están conectados a una computadora hogareña con su correspondiente software. Su costo es accesible al usuario medio, y por ello se transforma en una herramienta valiosa para los que se quieran iniciar en la robótica.

2.4 Partes de un robot

- La estructura o chasis (Endoesqueleto y Exoesqueleto).
- Las fuentes de movimiento.
- Los medios de transmisión de movimiento.
- Los medios de locomoción.
- Los medios de agarre.
- La fuente de alimentación.
- Los sensores.
- Los circuitos de control.

La **estructura o chasis** es la encargada de darle forma al robot y sostener sus componentes esta puede estar constituida por numerosos materiales, como plásticos, metales, etc. y tener muchas formas diferentes.

Así como en la naturaleza, los robots pueden ser del tipo "endoesqueleto", donde la estructura es interna y los demás componentes externos, o "exoesqueleto", donde la estructura está por fuera y cubre los demás elementos, las formas de las estructuras son de lo más variadas, tanto hasta donde la imaginación y la aplicación que se le va a dar al robot lo permitan.

Las **fuentes de movimiento** son las que le otorgan movimiento al robot. Una de las más utilizadas es el motor eléctrico de CC (corriente continua), servomotores y motores paso a paso. Una fuente de movimiento nueva que apareció recientemente en el mercado son los músculos eléctricos, basados en un metal especial llamado Nitinol (aleación de níquel y titanio).

Medio de transmisión de movimiento. Cuando las fuentes de movimiento no manejan directamente los medios de locomoción del robot, se precisa una interfase o medio de transmisión de movimiento entre estos dos sistemas, que se utiliza para aumentar la fuerza o para cambiar la naturaleza del movimiento, por ejemplo para convertir un movimiento circular en lineal, o para reducir la velocidad de giro. Se suelen emplear

conjuntos de engranajes para tal fin, aunque también se usan ruedas de fricción o poleas y correas.

Los **medios de locomoción** son sistemas que permiten al robot desplazarse de un sitio a otro si éste debe hacerlo, el más utilizado y simple es el de las ruedas y le siguen en importancia las piernas y las orugas.

Medios de agarre, algunos robots deben sostener o manipular algunos objetos y para ello emplean dispositivos denominados de manera general medios de agarre. El más común es la mano mecánica, llamada en inglés "gripper" y derivada de la mano humana.

La **fuente de alimentación** de los robots depende de la aplicación que se les dé a los mismos, así si el robot se tiene que desplazar autónomamente, se alimentará seguramente con baterías eléctricas recargables, mientras que si no requiere desplazarse o sólo lo debe hacer mínimamente, se puede alimentar mediante corriente alterna a través de un convertidor. En los robots didácticos se pueden emplear baterías comunes o pilas, y en los de muy bajo consumo celdas solares.

Los sensores le permiten al robot manejarse con cierta inteligencia al interactuar con el medio, los sensores son componentes que detectan o perciben ciertos fenómenos o situaciones. Estos sensores pretenden en cierta forma imitar los sentidos que tienen los seres vivos. Entre los diferentes sensores que podemos encontrar están las fotoceldas, los fotodiodos, los micrófonos, los sensores de tacto, de presión, de temperatura, de ultrasonidos e incluso cámaras de video como parte importante de una "visión artificial" del robot.

Los **circuitos de control,** son el "cerebro" del robot y en la actualidad están formados por componentes electrónicos más o menos complejos dependiendo de las funciones del robot y de lo que tenga que manejar.

Actualmente los modernos microprocesadores y microcontroladores, así como otros circuitos específicos para el manejo de motores y relés, los conversores A/D y D/A, reguladores de voltaje, simuladores de voz, etc. permiten diseñar y construir tarjetas de control para robots muy eficientes y de costo no muy elevado. El bajo costo actual de una computadora personal permite utilizarla para controlar robots de cualquier tipo utilizando las grandes ventajas que supone dicho dispositivo.

Los robots de última generación pueden interactuar con el medio de una manera más inteligente, es decir relacionarse eficazmente con el entorno y tomar decisiones en tiempo real, adaptando el plan de acción a las circunstancias de cada momento, todo esto gracias a los diferentes sensores, que les brindan información de posición, velocidad, aceleración, fuerza, dimensiones de objetos, temperatura, etc. De esta manera, con la información actualizada permanentemente, hace de estos robots autoprogramables, lo que supone disponer de un cierto grado de Inteligencia artificial. A continuación se pueden apreciar la mayor parte de las características que llevan a decidir entre un robot industrial y otro:

Tabla Nº 1.- Características básicas que definen un robot

Término	Definición
Grados de libertad	Es el número de movimientos básicos e independientes que posicionan a los elementos de un robot
Precisión repetitiva (Repetitividad)	Es la capacidad de volver a situarse en un punto determinado un número indefinido de veces
Capacidad de carga	Es el peso máximo que el robot puede manipular
Región espacial de trabajo	Es el volumen en el cual el robot puede manipular objetos. Se define según las coordenadas de programación
Área de trabajo lineal	Es la superficie plana sobre la cual el robot puede manipular objetos
Velocidad	Es la rapidez con que trabaja el robot, una medida de su rendimiento
Coordenadas de los movimientos	Es el tipo de sistema de posicionamiento y orientación del elemento terminal del robot
Tipo de actuador	Es el tipo del elemento motriz que genera los movimientos de las articulaciones del robot
Programabilidad	Es la manera como se programa el robot para sus tareas

Capítulo 3 – DESARROLLO DE EXPERIENCIA DE LABORATORIO 1

Experiencia nº 1

Titulo	:	Construcción y prueba del Hunter II
Sala	:	……..
Profesor	:	……..

3.1 Objetivos

El objetivo de esa experiencia es lograr la comunicación entre la BASIC Stamp y el computador para luego armar y programar el Hunter II de forma que se mueva hacia adelante, atrás y gire en el lugar. También, se determinar ajustes de software para que el Hunter II se mueva en línea recta y para que quede estático.

Nota: Para todas las Actividades de esta experiencia, necesitará una computadora personal (PC) con el sistema operativo Windows 95, 98, 2000, Me o XP.

3.2 Herramientas y Componentes

Las siguientes herramientas se recomiendan para las Actividades de la Experiencia Nº 1.

- Destornillador de Cruz.
- Destornillador de Paleta.
- Llave Punta-Corona de ¼”.
- Alicate de corte.
- Alicate de punta.
- Alicate Corta/pela cable.

Componentes requeridos.

La **Figura 3.1** muestra los componentes necesarios para la experiencia Nº 1.

- **A** 1-Chasis del Hunter II.
- **B** 1-Transformador 9 Volt.
- **C** 2-Servos.
- **D** 2-Ruedas plásticas.
- **E** 1-Rueda giratoria.
- **F** 1-Plaqueta de Educación.
- **G** 6-Tuercas 1/8”.
- **H** 7-Tornillos cabeza plana 1/8”.
- **I** 7-Separadores 1/8".
- **J** 1-Cable Serial.

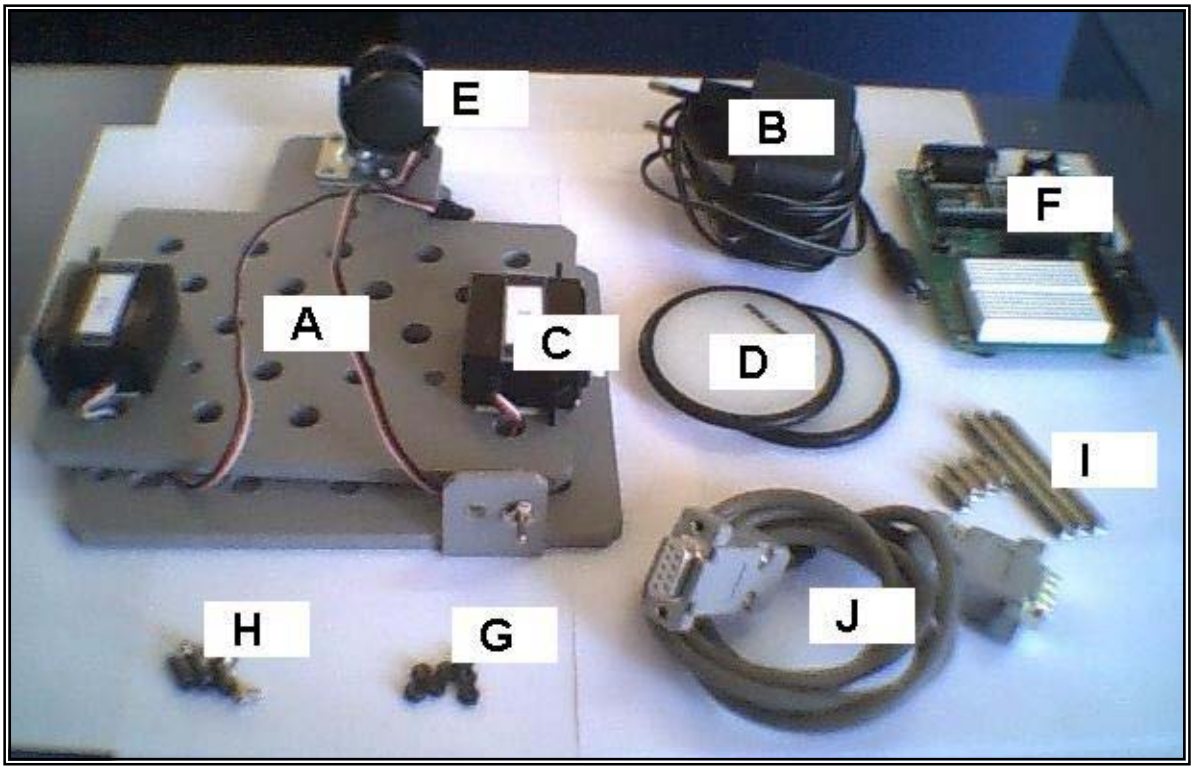


Figura 3.1 Componentes requeridos.

3.3 Procedimiento general.

La experiencia nº 1 no trata solamente sobre el armado del Hunter II. También sirve para asegurarse de que el Robot funcione correctamente al ir probando los subsistemas claves en todo el proceso de montaje. Siguiendo estas instrucciones, obtendrá experiencias prácticas sobre sistemas y desarrollo de subsistemas, control y solución de problemas.

Sistemas, subsistemas y competencias de robótica.

Cuando se diseña y construye un robot, es mejor imaginarlo como un conjunto de sistemas, subsistemas y elementos básicos. Un buen ejemplo de un sistema que puede ser descompuesto en subsistemas y elementos básicos podrían ser los servos del Hunter II. Como sistema, un par de servos modificados que funcionan como motores, trabajando juntos, hacen que el Hunter II se desplace. Cada servo puede ser visto como un subsistema. Cada servo tiene una pequeña plaqueta de circuito impreso en su interior, con componentes electrónicos. Este es un ejemplo de un subsistema dentro de otro subsistema.

Cada componente electrónico, si no puede seguir siendo descompuesto en componentes más pequeños, podría considerarse un elemento básico. Cada servo también tiene un subsistema de engranajes. Un engranaje en particular no puede ser separado en más partes, así que será un elemento básico. Cada servo recibe señales eléctricas, que le dicen lo que debe hacer, desde el BASIC Stamp, el cerebro del Hunter II. El Basic Stamp es otro sistema.

Una de las actividades más importantes, cuando se hace un robot, es el desarrollo y prueba de cada subsistema individual, de un sistema dado. Luego, también deben realizarse pruebas a nivel del sistema, para asegurarse que todos los subsistemas trabajan conjuntamente en la forma que se esperaba. Por último, pero no menos importante, se realiza la integración de sistemas, asegurándose que los mismos funcionen coordinadamente. La prueba y la solución de problemas en cada fase del desarrollo, en niveles de sistema y subsistema es, hasta cierto punto, una habilidad que se perfecciona con la práctica.

Armar, probar, modificar, probar, armar, probar…

El desarrollo de robots es un proceso iterativo en varios sentidos. La clave del desarrollo iterativo está en que los resultados de las pruebas son usados para realizar ajustes finos. Luego se realizan más pruebas y ajustes en la siguiente "iteración" de pruebas. En esta experiencia, el proceso iterativo será desarrollado, probado, ajustado si es necesario, luego desarrollado un poco más, probado nuevamente, etc. El objetivo principal es armar el Hunter II y hacerlo funcionar, sin tener que desarmarlo para realizar más pruebas y reparaciones.

Para el final de la experiencia, habrá aprendido a programar su Hunter II para que se desplace hacia atrás y adelante y gire en su lugar. En el camino, probará y calibrará los servos para ajustar el movimiento hacia delante y atrás. Aunque la mayor parte de la calibración de los servos se realiza desarmándolos y siguiendo las instrucciones de la actividad, también hay un ajuste fino que se realiza simplemente cambiando algunos de los números de los programas de ejemplo. Esta técnica es llamada "calibración por software".

3.4 Control de comunicación PC - BASIC Stamp.

En esta experiencia se señalan instrucciones para que siga al conectar el BASIC Stamp, el PC y la batería a la Plaqueta de Educación. También tiene instrucciones resumidas sobre la instalación del "BASIC Stamp Editor v2.1 Beta 1" y la ejecución de un programa PBASIC simple. Esa actividad también provee un ejemplo simple de comprobación a nivel de sistema e integración. Su tarea es seguir las instrucciones para conectar los sistemas y lograr que se comuniquen.

Los comandos en el lenguaje de programación PBASIC son ingresados en el Stamp Editor. Cuando su programa PBASIC esté listo, también usará el Stamp Editor para tokenizar el programa (traducirlo a lenguaje simbólico) y descargarlo en el BASIC Stamp. Dependiendo del programa, el Hunter II puede ser instruido para realizar muchas tareas.

El Stamp Editor también tiene una característica llamada Debug Terminal (Terminal de Depuración). Puede usar la Debug Terminal para mostrar mensajes recibidos desde el BASIC Stamp y también mandar mensajes hacia el BASIC Stamp. La Debug Terminal será una de sus asistentes más útiles para la prueba y solución de problemas. Lograr que

el BASIC Stamp se comunique con la Debug Terminal es muy fácil de hacer usando el lenguaje de programación PBASIC. Para comenzar, todo lo que se necesita es una línea de código PBASIC. La **Figura 3.2** muestra la presentación de la Plaqueta de Educación.

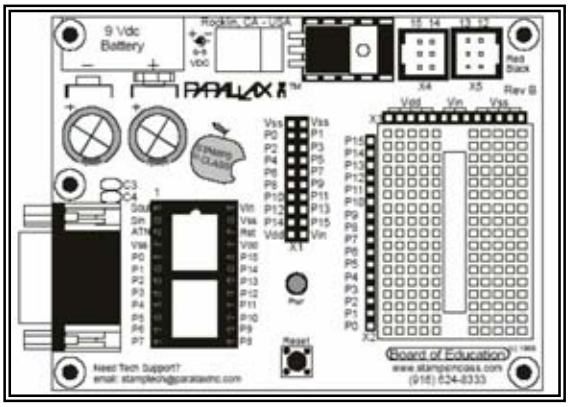


Figura 3.2 Plaqueta de Educación.

Montar la Basic Stamp.

El BASIC Stamp tiene medio círculo impreso en el centro de su extremo, cuando coloque el BASIC Stamp en su zócalo, asegúrese que el semicírculo esté cercano a los rótulos Sout y Vin. A modo de verificación, asegúrese que el chip negro más grande con el rótulo PIC16C57 quede en la parte inferior, entre los rótulos P7 y P8. Asegúrese de alinear los pines (patitas) del BASIC Stamp con los agujeros del zócalo, luego presione el BASIC Stamp firmemente con su pulgar. Los pines del BASIC Stamp deberían introducirse unos 5 milímetros en el zócalo. La **Figura 3.3** muestra al BASIC Stamp montado en su zócalo en Plaqueta de Educación.

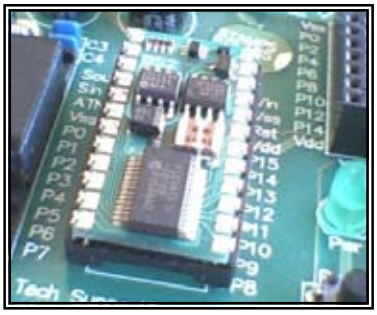


Figura 3.3 Montaje Basic Stamp.

Conexión del cable serial y batería.

Conectar el cable serial en el puerto com. en la parte trasera de una computadora, luego conecte el cable serial y la batería de 9 volt. a la plaqueta de educación. La **Figura 3.4** muestra el cable serial conectado a un puerto com en la parte trasera del computador y el cable serial y el Transformador conectados a la Plaqueta de educación, que está apoyada en una mesa sobre sus separadores. Estos separadores evitan que las soldaduras de la cara inferior de la Plaqueta de educación entren en contacto con la superficie de trabajo, que en el caso de ser conductiva, podría originar cortocircuitos.

Figura 3.4 Conexión de comunicación.

Instalación del "BASIC Stamp Editor v2.1 Beta 1".

Si el computador en el que esta trabajando no tiene instalado el "BASIC Stamp Editor v2.1 Beta 1" siga las instrucciones de esta sección, de lo contrario si su ordenador ya lo tiene pase a la siguiente sección de la experiencia.

Instalar el Basic Stamp Editor.

La **Figura 3.5** muestra el icono de instalación del Stamp Editor, ejecútelo y siga las instrucciones de instalación típica.

Figura 3.5 Icono de Instalación.

La **Figura 3.6** muestra el icono de acceso directo creado en el escritorio del PC, este será usado en el futuro para el arranque del Stamp Editor.

Figura 3.6 Icono de ejecución.

Establecer comunicación PC – BASIC Stamp.

Ejecutar el Stamp Editor, luego aparecerá la ventana que muestra la **Figura 3.7**

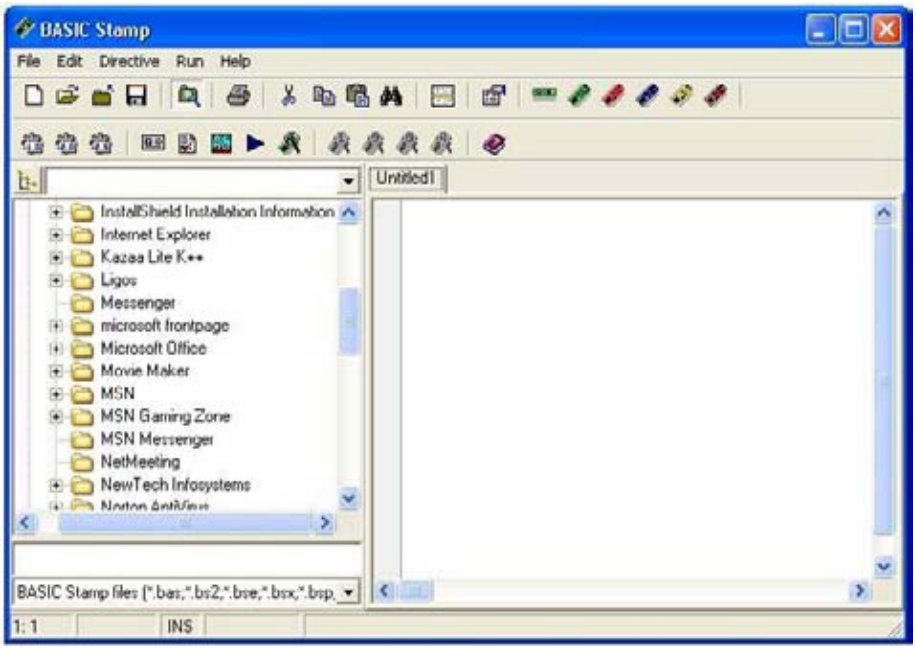


Figura 3.7 Ventana principal.

Hacer clic en Run y seleccionar Identify (identificar) como se muestra en la **Figura 3.8**, cuando todo esta conectado y trabaja apropiadamente, aparecerá una ventana como

muestra la **Figura 3.9** que indica que la Basic Stamp esta conectada correctamente y significa que esta y el PC se están comunicando.

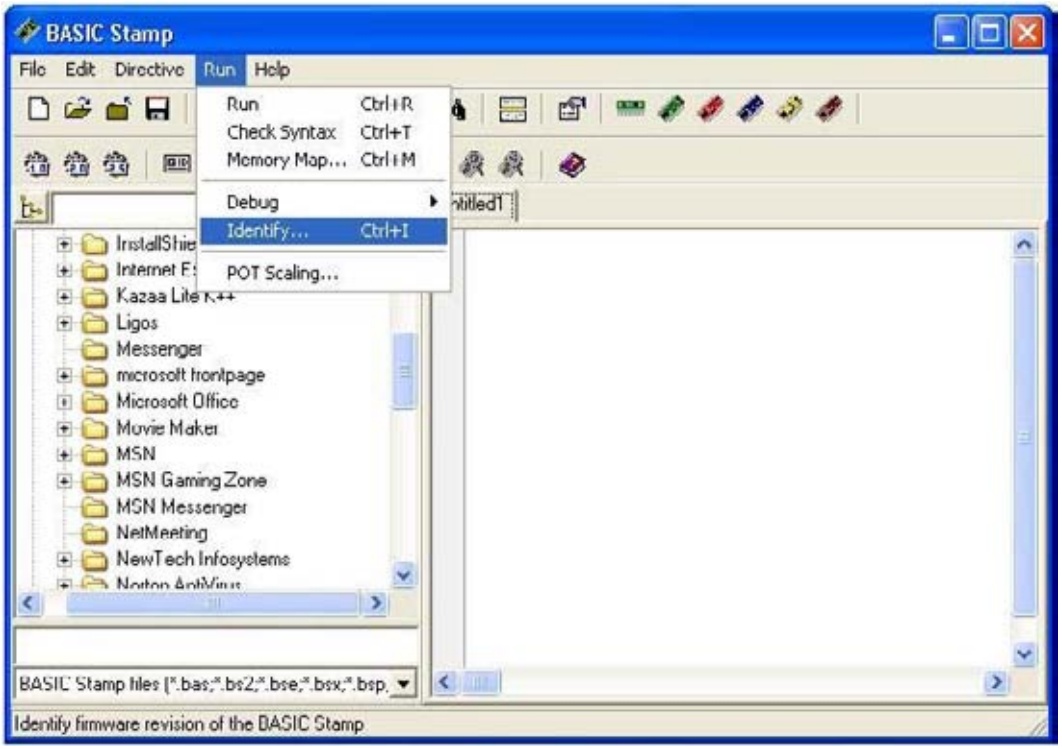


Figura 3.8 Ejecutar Identify.

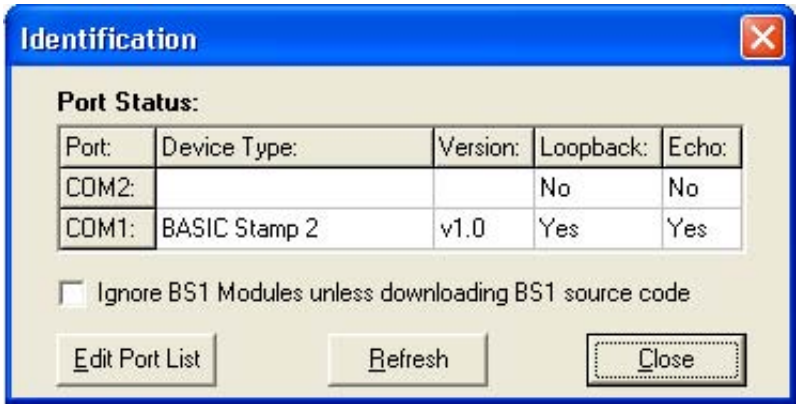


Figura 3.9 Indicación de conexión.

Ejecutar primer programa de prueba.

El primer programa demostrará la habilidad del BASIC Stamp para comunicarse con el mundo exterior, usando Debug Terminal (Terminal de Depuración). Esta herramienta puede ser usada para lograr la comunicación bidireccional entre su PC y el BASIC Stamp.

Escriba el Programa 1.1 en el Stamp Editor como se muestra en la **Figura 3.10**.

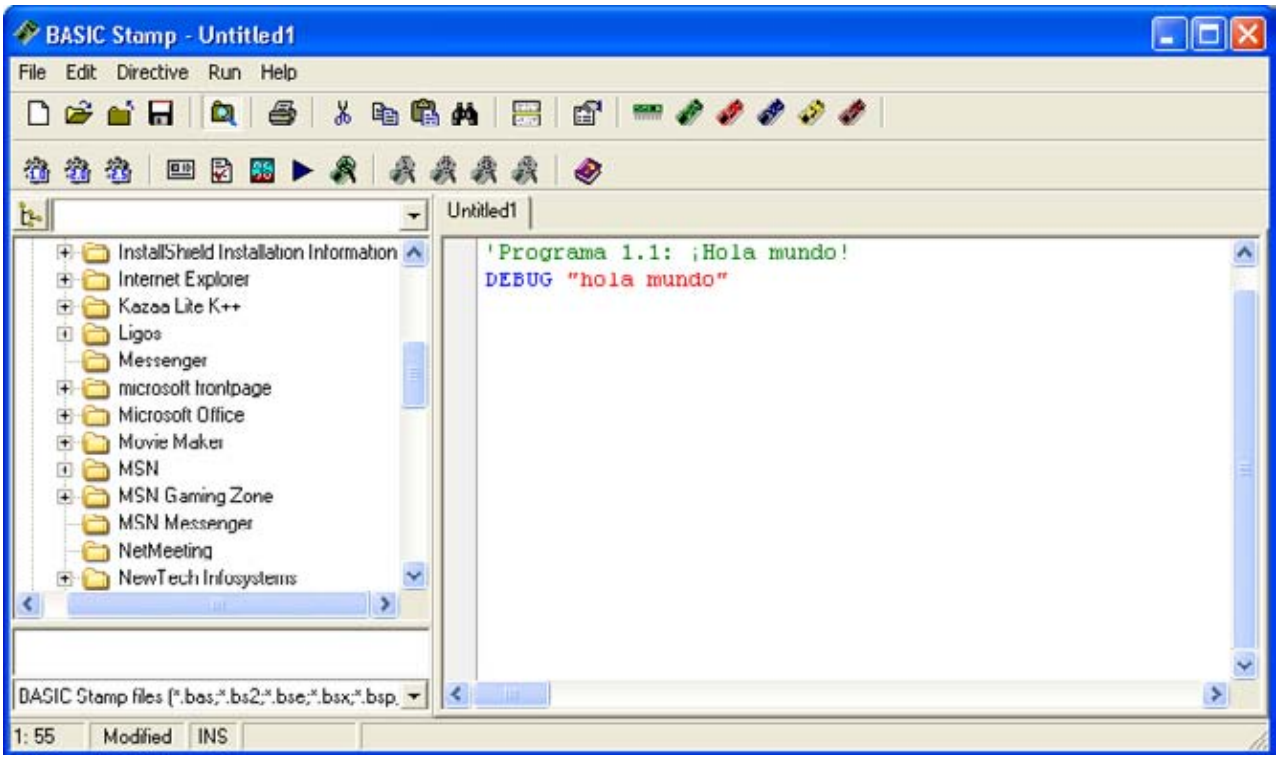


Figura 3.10 Programa 1.1.

Hacer clic en Run. La Debug Terminal debería aparecer en una segunda ventana, como se muestra en la **Figura 3.11**.

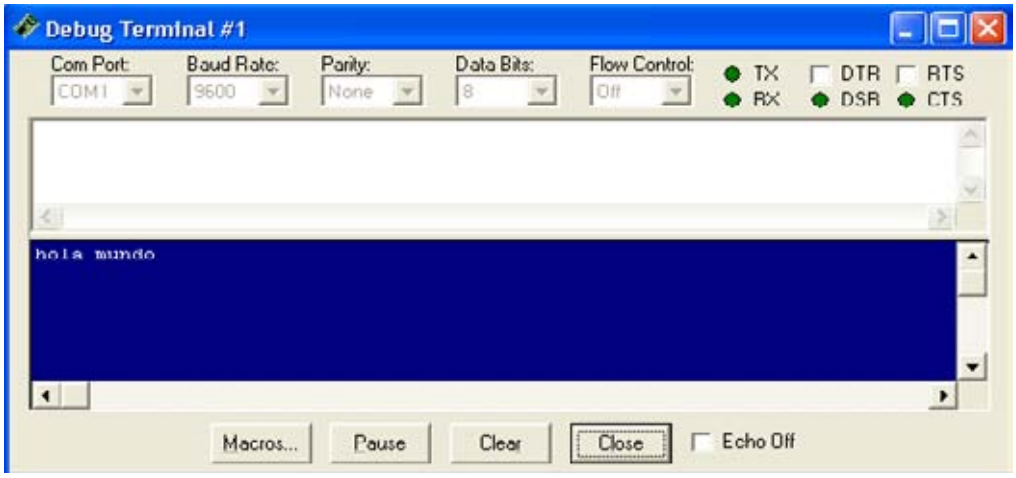


Figura 3.11 Ventana Hola mundo.

Anteriormente se había hablado del tokenizado, en efecto cuando un programa PBASIC es tokenizado, sus instrucciones son convertidas a instrucciones numéricas héxadecimales (tokens) que el chip intérprete del BASIC Stamp ejecuta. Luego, una ventana muestra el Progreso de la Descarga. Durante la descarga, el Stamp Editor envía el código héxadecimal al BASIC Stamp en forma de señales binarias, a través del cable serial. Luego aparece la Debug Terminal (Pantalla de Depuración) y se muestra el mensaje “hola mundo”, que el BASIC Stamp estaba programado para mostrar.

Como trabaja “Hola mundo”.

Antes de leer esta sección de la experiencia, vaya al **Anexo 1**: Referencia Rápida de PBASIC, y lea sobre el comando **debug** introducido en este programa.

La primera línea del programa comienza con un apóstrofe. Esto significa que no es un comando, sino simplemente un comentario. El programa funciona exactamente igual si no se introducen los comentarios en los programas PBASIC del Stamp Editor.

La segunda línea comienza con un comando llamado **debug**. Cuando se ejecuta un programa que contiene un comando **debug**, el Stamp Editor abre una ventana de Debug Terminal. Cuando el BASIC Stamp ejecuta el comando **debug**, envía el mensaje “hola mundo” a la computadora a través del cable serial. El mensaje “hola mundo” es una cadena de texto, que es uno de los varios tipos de datos de salida que el BASIC Stamp puede ser programado para enviar, mediante el comando **debug**. Los datos de salida pueden tomar muchas formas distintas. Algunos ejemplos incluyen variables, constantes, expresiones, modificadores de formato y caracteres de control.

El mismo comando **debug** puede ser usado para enviar más de un mensaje. Cada mensaje debe ser separado del anterior por una coma. Los caracteres de control, tales como **cr** pueden ser enviados para indicar un salto de línea hasta el margen izquierdo.

Una o más constantes pueden ser mostradas con su propio formato. Un ejemplo de una constante podría ser el número 16. Indicadores de formato tales como

dec, pueden ser usados para mostrar valores decimales. Los indicadores de formato **dec1, dec2** y hasta **dec5** pueden ser usados para mostrar valores decimales con un número fijo de dígitos. Otros ejemplos de indicadores de formato son **bin** y **hex,** útiles para mostrar datos en formato binario o héxadecimal.

Además de constantes, se pueden usar algunas expresiones matemáticas para resolver ciertos problemas, antes que el comando **debug** muestre los valores. Por ejemplo, la expresión **dec 7+9** dará el mismo resultado en la Debug Terminal que **dec 16**.

Para comprender mejor las variadas formas de usar **debug**, intente realizar las modificaciones al Programa 1.2 de la **Figura 3.12** que se listan a continuación. Luego de realizar cada modificación en su programa, recuerde volver a ejecutar su programa haciendo clic en Run.

```
'{$STAMP BS2}
    'Programa 1.2: Más comando debug.
DEBUG CR, CR, "hola mundo", CR, CR ' 2 líneas, hola mundo y 2 líneas más
DEBUG DEC 16 ' muestra el valor decimal 16.
```

Figura 3.12 Programa 1.2.

1.- Sustituir el indicador de formato **dec** por **dec3**. También pruebe con **dec2**, **bin** y **hex**. No olvidar ejecutar el programa después de cada modificación para ver los resultados.

2.- Reemplazar el número **16** con la expresión: **11 + 5**

3.- Probar con la expresión: **2 x 8**

4.- Al final del programa, agregar una tercera línea de código: **debug, "hola mundo, otra vez"**.

3.5 Prueba de servos.

Como se mencionó anteriormente, los servos modificados trabajarán juntos para conformar el sistema de motorización del Hunter II. En esta actividad, cada servo será aislado y probado como un subsistema.

Las pruebas simples que realizará en esta actividad, responderán algunas inquietudes y además indicarán que hacer a continuación. Por ejemplo, si los servos son nuevos, ellos funcionarán como servos, en lugar de funcionar como motores.

Como funcionan los servos.

Los servos para hobby son motores especiales con realimentación de posición interna. Su rango de giro es típicamente de 90º ó 180º y son especiales para aplicaciones donde se requiera un movimiento de mucha fuerza con precisión y a bajo costo. Son muy populares en los sistemas de control de autos, botes y aviones radio controlados. Los servos están diseñados para controlar la posición de un alerón en un avión o el timón en un bote radio controlado. En esta experiencia modificaremos los servos del Hunter II para que controlen la velocidad y la dirección de las ruedas del robot.

La **Figura 3.13** muestra el circuito que se establece cuando un servo es conectado en el puerto para servos rotulado con un 12, en la esquina superior derecha de la Plaqueta de educación. Los cables rojo y negro se conectan a la fuente de alimentación y el blanco es conectado a la fuente de señal. Cuando un servo es conectado en el puerto para servos 12, la fuente de señal para el servo es el pin P12 del BASIC Stamp.

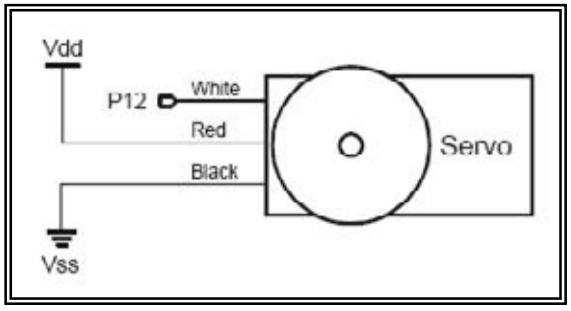


Figura 3.13 Circuito de un servo.

El BASIC Stamp puede ser programado para enviar señales a través de P12 que hagan que el servo se mueva.

Con servos sin modificar, una señal de control dada hace que el engranaje de salida se mueva a un lugar en particular, dentro de su rango de movimiento de 180º. El engranaje de salida de un servo modificado, por otro lado, girará continuamente con la misma señal.

Un servo sin modificar tiene una posición central, que se encuentra en la posición media entre ambos extremos del recorrido. La misma señal que hace que un servo sin modificar se mueva hacia su posición central es la que hace que un servo modificado no produzca ningún movimiento. Es posible tener control del sentido de giro de los servos modificados

porque una señal que mueve un servo sin modificar a una posición en sentido antihorario, es la misma que hará girar continuamente a un servo modificado en el mismo sentido.
Lo mismo se aplica para el sentido horario. También se puede controlar la velocidad de un servo modificado.
Enviando una señal próxima a la posición central se moverá lentamente, cuando la señal se aleje de esta posición, aumenta la velocidad.

Modificar servos.

Para esta parte de la experiencia, la Plaqueta de Educación deberá estar desconectada de la batería, pero conectada con el computador.
La **Figura 3.14** muestra los puertos de los servos de la Plaqueta de Educación. Los números en la parte superior indican el número de puerto. Si conecta un servo en el Puerto 12, significa que la línea de control del servo está conectada al pin de E/S P12.
Los rótulos del costado derecho de los puertos para servos están para asegurarse que los conecte en el sentido correcto. **La Figura 3.15** muestra un servo conectado al puerto 12 de forma que el cable negro (black) coincida con el rótulo black y que el cable rojo (red) coincida con el rótulo red. Aunque el tercer cable aparece rotulado como “white” (blanco), este también puede ser amarillo (yellow).

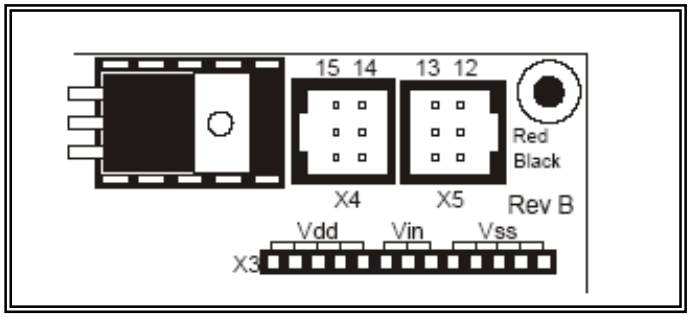


Figura 3.14 Puertos en la Plaqueta.

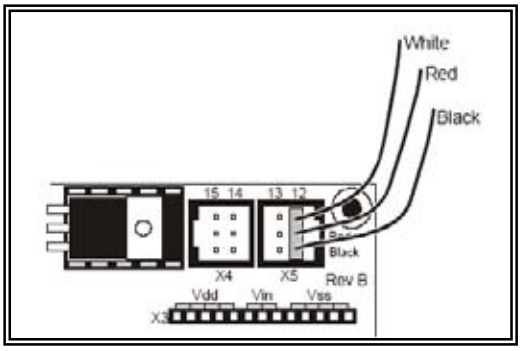


Figura 3.15 Conexión de los servos.

En esta Experiencia, nos referiremos al tiempo en unidades de segundos (s), milisegundos (ms) y microsegundos (µs). Los segundos son representados por una letra s minúscula. Así, un segundo se escribirá como: 1 s. Los milisegundos se abrevian como ms y representan una milésima de segundo. Un microsegundo es una millonésima de segundo. Su conexionado debería verse como el de la **Figura 3.16**.

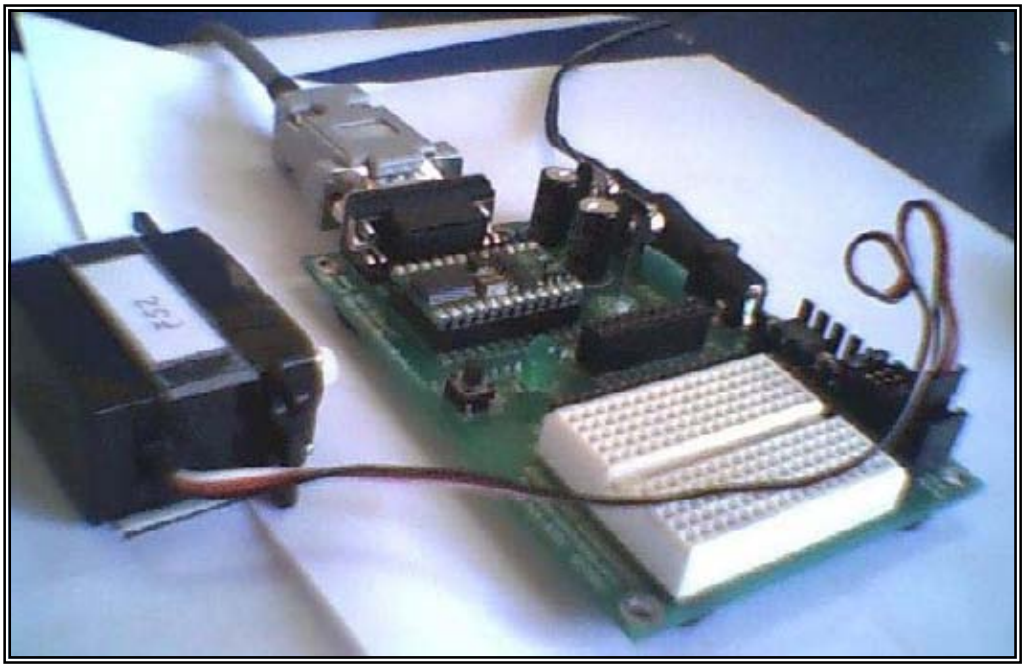


Figura 3.16 Conexión de comunicación.

Un nivel de tensión se mide en volts (voltios). La Plaqueta de Educación tiene conectores rotulados **Vss**, **Vdd** y **Vin**. **Vss** es llamada masa del sistema o tensión de referencia.

Cuando se enchufa la batería, **Vss** es conectado al terminal negativo. En lo que respecta a la Plaqueta de Educación, BASIC Stamp y conexión serial a la computadora, **Vss** es siempre 0 Volt. **Vin** son los 6 volt sin regular y están conectados al terminal positivo del porta pilas. **Vdd** representa a los 5 Volt regulados en la Plaqueta de Educación por el regulador de tensión, que serán usados junto con **Vss** para alimentar los circuitos que se construyan en la protoboard.

Programar los servos para que permanezcan estáticos y "centrados".

La señal de control que el BASIC Stamp envía a la línea de control del servo es llamada "tren de pulsos," y se muestra en la **Figura 3.17**. El BASIC Stamp puede ser programado para producir esta forma de onda en cualquiera de sus pines de E/S. En esta actividad, usaremos el pin de E/S P12, que ya se encuentra conectado al puerto de servos. Primero, el BASIC Stamp fija la tensión de P12 a 0 Volt (estado bajo) por 20 ms. Luego, fija la tensión de P12 a 5 Volt (estado alto) durante 1.5 ms. luego, repite el ciclo con un estado bajo por 20 ms y una salida en estado alto por 1.5 ms y así sucesivamente.

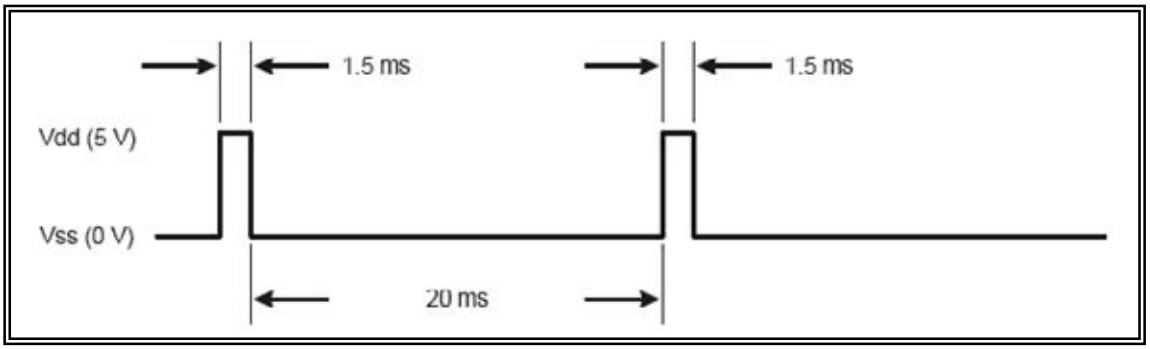


Figura 3.17 Tren de pulsos.

Este tren de pulsos tiene un tiempo de encendido de 1.5 ms y un tiempo de apagado de 20 ms. El tiempo de encendido o de estado alto se suele llamar ancho del pulso. Cuando hablamos de pulsos se sobreentiende que son pulsos positivos. Los pulsos negativos se toman como tiempo de descanso o separación entre pulsos positivos. Los trenes de pulsos tienen otros parámetros técnicos tales como ciclo de trabajo.

Un servo tiene una salida analógica, lo que significa que puede girar dentro de un rango de valores continuo.

Así trabajan los servos del Hunter II. Si un servo sin modificar recibe pulsos de 1 ms, su engranaje de salida rotará en sentido horario lo más lejos que pueda, hasta llegar a los 180º de su rango de movimiento. Si el servo recibe pulsos de 2 ms, rotará en sentido anti-horario hasta su límite de movimiento. Los pulsos de 1.5 ms harán girar a un servo sin modificar hasta quedar estable en el centro de su rango de movimiento de 180º.

Esta se llama la posición central del servo. Pulsos de 1.3 ms harán que el engranaje de salida de un servo sin modificar rote ligeramente en sentido horario, partiendo desde el centro y pulsos de 1.7 ms harán lo mismo pero en sentido anti-horario.

El ancho del pulso es lo que controla el movimiento del servo. La separación entre pulsos puede variar entre 10 y 40 ms sin afectar el rendimiento del servo.

Luego que un servo es modificado, se le pueden enviar pulsos para hacerlo girar constantemente. Los anchos de pulsos para los servos modificados típicamente oscilan entre 1.3 y 1.7 ms para velocidad máxima en sentido horario y anti-horario respectivamente. El ancho del pulso central sigue siendo 1.5 ms y un servo modificado y calibrado correctamente debería permanecer estacionario cuando recibe pulsos de 1.5 ms. Si gira muy lentamente en respuesta a estos pulsos puede realizarse un ajuste por software. Si gira rápidamente con los mismos pulsos, el servo deberá ser desarmado y recalibrado.

La **Figura 3.18** muestra el programa para enviar el tren de pulsos “centrales”que muestra la **Figura 3.19**. Esto debería hacer que un servo girase hasta su posición central y permaneciese allí. Un servo modificado quedaría estático, o rotaría muy lentamente.

- Tome notas del comportamiento del servo, para realizar un diagnóstico.
- Ingrese el Programa 1.3 de la **Figura 3.18** en el Stamp Editor.

```
'{$STAMP BS2}
'Programa 1.3: Programa para centrar los servos.
LOW12 ' Configura a P12 como salida en estado bajo
BUCLE: ' Rótulo que indica el inicio de un bucle
PULSOUT 12, 750 ' envía pulsos de 1.5 ms por P12
PAUSE 20 ' cada 20 ms.
GOTO BUCLE ' envía el programa hacia el rótulo "bucle:".
```

Figura 3.18 Programa 1.3.

- Guardar el programa usando un nombre descriptivo, preferiblemente incluyendo “1.3”, para indicar que es el Programa 1.3.
- Ejecutar el programa.
- Observar y registrar el comportamiento del servo.

Cómo trabaja el programa.

- Buscar cada una de las siguientes funciones en el **Anexo 1**, antes de continuar: **low, pulsout, pause, goto.**

Como antes, la primera línea del programa comienza con un apóstrofe, convirtiéndola en un comentario en lugar de un comando PBASIC. También hay un comentario al costado derecho de cada comando PBASIC.

Estos comentarios también comienzan con apóstrofes y dan una breve explicación de lo que hace el comando.

Cuando ingrese los comandos del programa en el Stamp Editor, no necesita incluir comentarios. Los comentarios y otras formas de documentar sus programas se vuelven importantes si escribe códigos más complejos, o si alguien más debe trabajar con el mismo programa. Si quiere ahorrar tiempo, ingrese solamente los comandos en el Stamp Editor.

El comando **low 12** hace dos cosas. Configura al pin de E/S P12 del BASIC Stamp como salida, luego fija su valor de salida en bajo (0 Volt). Como salida, P12 puede enviar señales de tensión, lo opuesto a cuando se usa **input**, que significa que P12 podrá únicamente leer señales. Fijar el valor de la salida en bajo significa que la tensión que envía P12 es igual a **Vss**, 0 volts. Si fuera usado el comando **high 12**, P12 enviaría un estado alto, que sería igual a **Vdd**, 5 volts.

Cuando se escribe una palabra que no es ningún comando de PBASIC seguida por dos puntos, se denomina etiqueta. A medida que se familiarice con el PBASIC, empezará a reconocer automáticamente que palabras son comandos y cuáles son etiquetas. La etiqueta **bucle:** trabaja junto con el comando **goto bucle** del final del programa. La etiqueta **bucle:** está marcando un lugar del programa y cada vez que el programa ejecuta el comando **goto bucle**, automáticamente comienza a ejecutar los comandos posteriores a la etiqueta **bucle:** El resultado es que los comandos **pulsout** y **pause** se ejecutan una y otra vez, enviando la señal que centra al servo.

La estructura del programa **bucle:...goto bucle** es llamada bucle infinito. Es infinito porque el código se repite una y otra vez sin posibilidad de que alguna instrucción detenga la ejecución del mismo conjunto de instrucciones. Con programas de computadoras normales, esto es un problema. Sin embargo, los bucles son muy usados en la programación de microcontroladores.

El comando **pulsout 12, 750** envía un pulso de 1.5 ms. El comando tiene dos argumentos, el número de pin de E/S y la duración. El uso del número de pin de E/S es obvio; el número 12 se refiere al pin de E/S P12.

¿Qué pasa con el argumento que indica una duración de **750** y como es que corresponde a un pulso de 1.5 ms?, la respuesta es que el argumento de duración del comando **pulsout** está especificado en incrementos de 2 µs. Así, si quiere que el pulso dure 1.5 ms, debe usar un número que le dé 1.5 ms cuando sea multiplicado por los incrementos de 2 µs. El siguiente ejemplo es la demostración de que un **pulsout** de **750** es el correcto.

Ejemplo:

$750 \times 2\ \mu s = 750 \times (2 \times 10^{-6})\ s$

$= 1500 \times 10^{-6}\ s$

$= 1.5 \times 10^{-3}\ s$

$= 1.5\ ms$

El comando **pause 20** es mucho más obvio. Esto es debido a que el argumento de duración del comando **pause** está especificado en incrementos de ms. Así, si quiere una pausa de 20 ms, **pause 20** cumplirá esa tarea.

Hacen falta dos pruebas más para decidir qué hacer con los servos. Registre sus observaciones sobre cada prueba.

- Ejecute el Programa 1.4 que se muestra **Figura 3.19**. Tome nota del comportamiento del servo.
- Reemplace la duración del comando **pulsout** con el número 850 y reejecute el programa. Tome notas del comportamiento.
- Después de usar los Programas 1.3 y 1.4 para probar un servo, repetir las mismas pruebas para el otro.

```
' Programa 1.4: Programa para centrar el servo.
LOW 12 ' Configura P12 como salida baja.
bucle: ' Etiqueta hacia donde saltar
PULSOUT 12, 650 ' Envía pulsos de 1,3 ms por P12
PAUSE 20 ' cada 20 ms.
GOTO bucle ' Salta hacia "bucle: ".
```

Figura 3.19 Programa 1.4.

Diagnóstico del servo.

Dependiendo de cada servo y de su historia, puede haber exhibido uno de varios comportamientos distintos.

A continuación se dará una lista de los comportamientos. Cada comportamiento es seguido por una explicación de las acciones que se deberán realizar. Si ambos servos no se comportan igual, asegurarse de encontrar el comportamiento que se ajuste para cada uno en particular.

- **Comportamiento del servo:**

En respuesta a pulsos de 1.5 ms (**pulsout 12, 750**), el engranaje de salida del servo gira a cierta posición (posición central) y permanece quieto. Hace fuerza si se quiere alejar de esa posición. En respuesta a pulsos de 1.7 ms (**pulsout 12, 850**), el servo rota ligeramente en sentido anti-horario desde la posición central y se detiene. En respuesta a pulsos de 1.3 ms (**pulsout 12, 650**), el servo gira ligeramente en sentido horario y se detiene. Este es el funcionamiento normal de un servo sin modificar. Modificar y calibrar este servo.

- **Comportamiento del servo:**

En respuesta a pulsos de 1.5 ms, el servo permanece estacionario o gira lenta y constantemente. Muy lentamente sería si da menos de dos vueltas completas por minuto. En respuesta a pulsos de 1.7 ms, el servo gira en sentido anti-horario y se mantiene girando rápidamente, aproximadamente a 50 revoluciones por minuto (RPM). En respuesta a pulsos de 1.3 ms, el servo se mantiene girando a aproximadamente la misma velocidad, pero en sentido horario. Este comportamiento es de un servo que ya fue modificado y calibrado.

- **Comportamiento del servo:**

El servo gira constantemente a más de 2 RPM cuando es usado el comando pulsout 12, 750. Este comportamiento se refiere a un servo modificado pero no calibrado apropiadamente, o puede haberse descalibrado. Un servo puede descalibrarse si es expuesto a fuertes vibraciones, golpes fuertes y algunas otras condiciones.

- **Comportamiento del servo:**

Cuando el servo gira, hace un ruido intermitente y si el ruido se repite una o dos veces por giro completo, indica que el servo fue modificado pero una parte del tope del engranaje no fue quitado correctamente. Para arreglarlo, repase la Actividad 3 y asegúrese de remover completamente el tope del engranaje. Si el servo gira continuamente pero hace un zumbido, puede tener algún problema mecánico. Cuando se realice la modificación de un servo, asegurarse que los engranajes del este estén limpios y que nada obstruye su movimiento, antes de rearmarlos.

- **Comportamiento del servo:**

Si el servo nunca se movió podría haber muchas explicaciones posibles para esto. Si un servo no hace nada, intente con el otro. Si uno funciona pero el otro no, el problema se aísla hacia un servo. Revise las conexiones eléctricas desde la ficha de conexión hasta el servo.

Modificación de servos.

Si esta comenzando con un servo sin modificar, esta Experiencia le enseñará a desarmar, modificar, calibrar y probar cada servo. Si su objetivo es recalibrar un servo, seguirá los mismos pasos excepto la parte de la modificación. Es muy importante hacer las pruebas de verificación después que los servos han sido calibrados. Este es un ejemplo de desarrollo y prueba iterativo.

El circuito de un servo sin modificar compara el valor de su potenciómetro con el ancho de los pulsos que recibe por la línea de control. Luego activa su motor para corregir cualquier diferencia entre ambos valores. Hace esto con cada pulso, así que si intenta desplazar de su posición el eje de control de un servo, su circuito detectará una diferencia entre el valor del potenciómetro y los pulsos, causando el encendido del motor, para anular esta diferencia. Esto sucede tan rápidamente que usted solamente siente que el eje del servo resiste la fuerza que intenta desplazarla de su posición.

La información que el servo recibe del potenciómetro se llama realimentación. La comparación entre el valor del potenciómetro y el ancho del pulso y las correcciones que origina, son los componentes del proceso llamado control de lazo cerrado. Para hacer que los servos giren como motores, hay que eliminar la realimentación. Cuando la unión entre el engranaje de salida y el potenciómetro es eliminada, el funcionamiento se realizará a lazo abierto en lugar de a lazo cerrado.

Una vez que los servos funcionan a lazo abierto, aún no serán capaces de realizar giros completos hasta que se elimine el tope mecánico de uno de sus engranajes. Los engranajes de plástico son bastante blandos y el tope puede ser eliminado con facilidad.

Antes de rearmar el servo, el potenciómetro debe ser ajustado para que el engranaje de salida quede inmóvil cuando el servo reciba la señal para ir a la posición central. Esta vez, cuando el tren de pulsos de 1.5 ms sea enviado, el motor del servo comenzará a girar. Luego se puede ajustar manualmente el potenciómetro para lograr que el servo se detenga. Este proceso es llamado centrado.

Luego de centrar el servo y quitarle la realimentación y el tope de engranaje, se procede a armarlo y probarlo.

Procedimiento para la modificación.

- Desconectar la batería de la Plaqueta de Educación.
- Desconectar el servo de Plaqueta de Educación.
- Usando un destornillador, quitar el tornillo que sujeta la rueda del servo.
- Quitar los cuatro tornillos de la parte trasera del servo usando un destornillador.
- Poner el servo de forma que el engranaje de salida quede hacia arriba.
- Quitar la cubierta de engranajes del servo de forma que el sistema de engranajes quede a la vista.

Los engranajes montados se muestran en la **Figura 3.19**. Los nombres de cada uno de los componentes se muestran en **La Figura 3.20**.

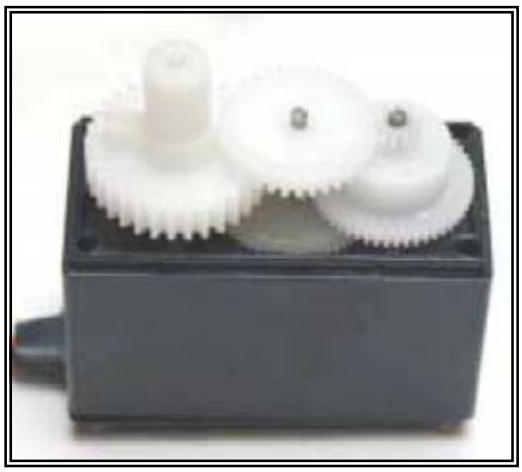


Figura 3.19 Engranajes del servo.

Figura 3.20 Partes Internas.

- Quitar el último y el tercer engranaje como se muestra en la figura anterior.
- Encontrar y quitar el mando del potenciómetro. Se encontrará en el eje del potenciómetro o en la base del último engranaje. Esta pieza es el vínculo de la realimentación que debe ser removido para que el servo funcione a lazo abierto.

La **Figura 3.21** muestra el último engranaje e indica el tope que debe ser eliminado.

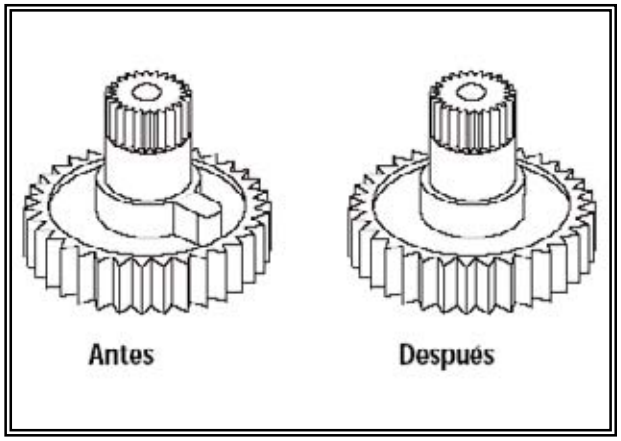


Figura 3.21 Engranaje a modificar.

Calibración del servo.

Antes de rearmar el servo, se lo debe centrar. En otras palabras, el potenciómetro debe ser ajustado para que el servo permanezca estático cuando recibe un ancho de pulso de 1,5 ms.

- Conectar el servo desarmado a la Plaqueta de Educación.
- Conectar la batería a la Plaqueta de Educación.
- Abrir el Programa 1.3 haciendo clic en el menú File del Stamp Editor y seleccionando open, o rescriba el programa en el Stamp Editor.
- Después de asegurarse de haber ingresado correctamente el Programa 1.3, y ejecutar.
- El motor del servo comenzará a funcionar y el primer y segundo engranaje girarán.
- Girar el eje del potenciómetro.

Si se gira el eje en una dirección, la velocidad del servo aumentará y en la dirección opuesta, disminuirá.

- Girar cuidadosamente el eje del potenciómetro hasta que el servo se detenga.
- Desconectar el servo de la Plaqueta.
- Armar, sin colocar el mando del potenciómetro.

Ambos servos deben ser modificados y calibrados. Repetir todos los pasos para cada servo.

Prueba del servo

- Conectar el servo rearmado en la Plaqueta de Educación.
- Ahora que los servos están rearmados, asegúrese de que el BASIC Stamp esté ejecutando el Programa 1.3 y conecte cada servo en el puerto 12.
- Cuando cada servo es conectado en el puerto 12, su engranaje de salida debería permanecer quieto o rotar lentamente (menos de 2 RPM).
- Probar cada servo con el Programa 1.4. Asegurarse de usar un argumento de duración de **pulsout** igual a **650**. El engranaje de salida debería rotar bastante rápido, a aproximadamente 50 RPM.
- Probar cada servo con el Programa 1.4, pero con un argumento de duración de **pulsout** de **850**. Esta vez, el engranaje de salida debería rotar en sentido antihorario a aproximadamente 50 RPM.

Si el servo pasó estas tres pruebas, está calibrado. Si falló en alguna, repetir la sección de Diagnóstico de Servos.

Centrado de servos – calibración por software.

La calibración por software involucra probar un servo con duraciones de pulsos cercanas al valor de **pulsout** central ideal de 750 (1.5 ms). Lo que se busca es el valor de duración más exacto que logre detener totalmente el engranaje de salida del servo. Esto es especialmente importante si alguno de los servos giró lentamente cuando se ejecutó el Programa 1.3 después de ser rearmado. Este es un ejemplo de una segunda iteración en el proceso de calibración.

- Modificar el Programa 1.4 para que la duración del comando **pulsout** sea 735 y reejecute el programa. El servo debería girar lentamente en sentido horario.
- Modificar el argumento de duración a 736 y reejecutar el programa. Repetir para 737, 738 y así hasta 765.
- Tomar nota de los argumentos de duración que hacen detener al servo y de aquellos que lo ponen nuevamente en movimiento.
- Tomar el promedio de los anchos de pulso que detienen al servo y llamar a ese valor ancho de pulso central.

El verdadero valor del argumento de duración para centrar el servo está en algún punto entre el valor que detiene al servo y el valor que lo hace girar nuevamente. Por ejemplo, si

el servo deja de girar con 749 y comienza a girar nuevamente con 751, es fácil. Use 750 como ancho central para ese servo.

El argumento de duración del comando **pulsout** solamente puede tomar valores enteros entre 0 y 65535. Algunos ejemplos de argumentos de duración válidos son: 0, 1, 2,...,754, 755,…. así que, 754,5 no puede ser usado como argumento válido. La duración promedio de **pulsout** debe ser redondeada, ¿pero hacia dónde? Aunque es una práctica común redondear hacia arriba si el valor decimal es 0,5 o superior, el servo puede no responder a esta práctica. Podría determinar experimentalmente hacia qué sentido redondear probando duraciones de **758** y **751** en su programa.

- Escribir el argumento de duración del pulso central para cada servo sobre una etiqueta autoadhesiva y pegar sobre el servo. De esta forma, al escribir futuros programas, se sabrá la duración central de cada servo.

3.6 Armado del Hunter II

- Usar tornillos y tuercas de 1/8 pulgadas para colocar los tres separadores largos y ensamblar el chasis.
- Colocar los servos y la rueda giratoria en el chasis.
- Montar las ruedas en los Servos.

La **Figura 3.22** muestra el montaje del chasis, los servos, las ruedas y la rueda giratoria.

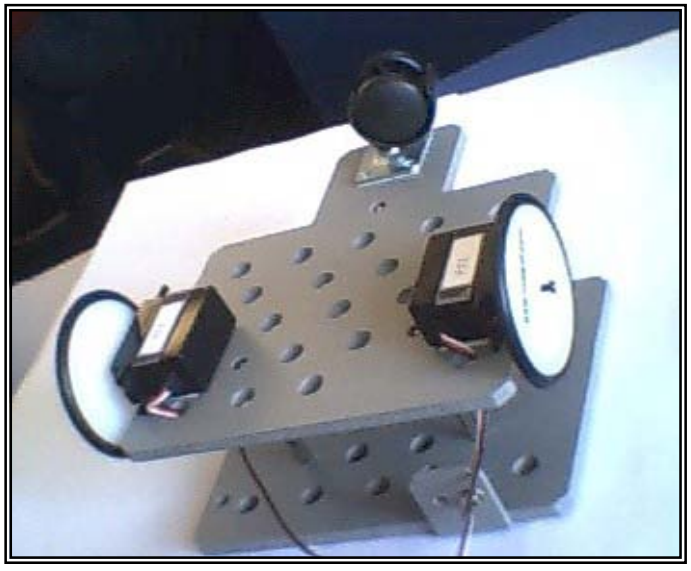


Figura 3.22 Armado de Chasis.

Montaje de la plaqueta de educación al chasis del Hunter II.

- Usar tuercas para colocar los cuatro separadores cortos sobre el chasis.
- Instalar la Plaqueta de Educación sobre el Chasis ensamblado usando los tornillos de 1/8 pulgadas.

La **Figura 3.23** muestra la Plaqueta de Educación, BASIC Stamp y chasis completo.

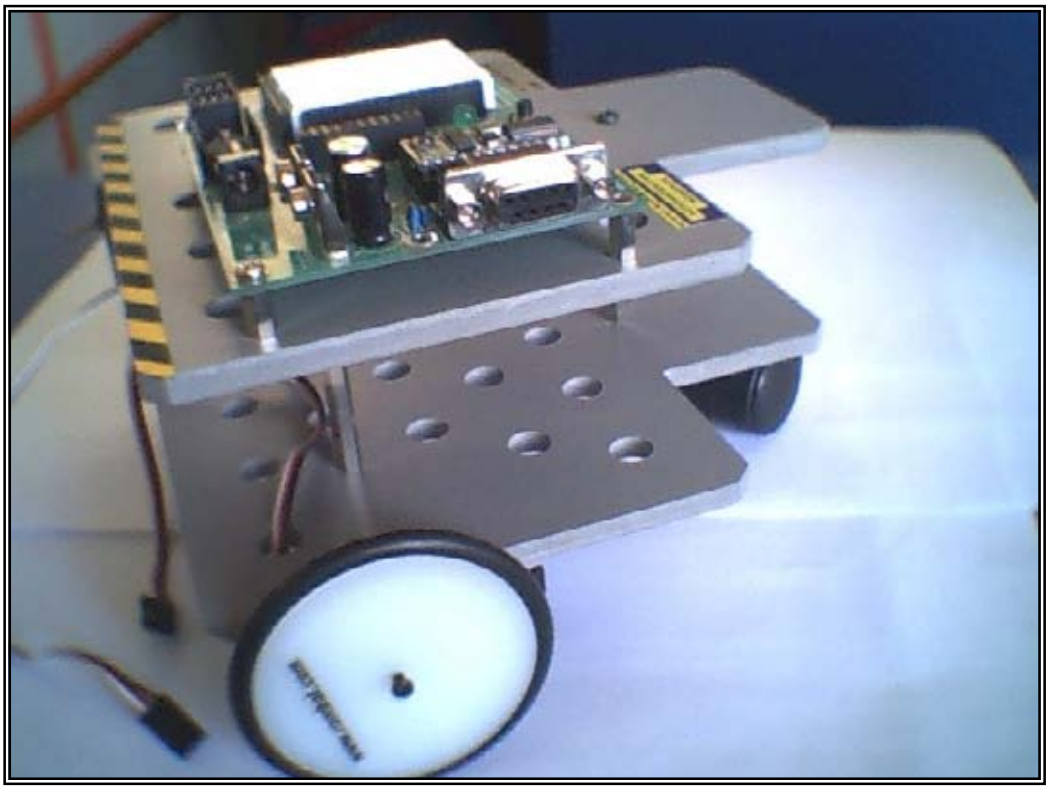


Figura 3.23 Robot ensamblado.

Programación de movimientos básicos.

Ahora el Hunter II esta listo para programarle movimientos. Esta experiencia permitirá aprender a programar el robot para que se mueva hacia adelante, atrás y gire. Se realizará un ajuste fino del programa para asegurarse que el robot pueda moverse en línea recta.

Los servos no necesariamente funcionarán parejos. Como resultado, aún cuando el Hunter II es programado para avanzar en línea recta, podría girar lentamente hacia la

izquierda o la derecha. La calibración por Software es usada para solucionar este problema.

La **Figura 3.24** muestra las referencias para izquierda, derecha, adelante y atrás del Robot.

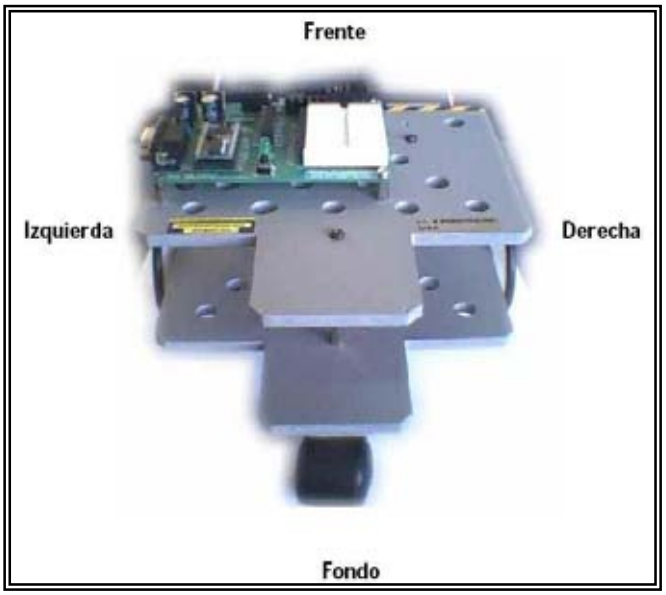


Figura 3.24 Referencias.

Para hacer funcionar al robot es tan fácil como agregar un segundo comando **pulsout**. La parte difícil es determinar los parámetros de duración de cada pulso.

Mirando el costado derecho del Hunter II, para hacer que esta rueda gire hacia adelante, el servo debe girar en sentido horario. Esto implica un argumento de duración menor que el central de tal forma que una duración de 650 servirá para lograr la velocidad máxima hacia adelante. Ahora mirando el costado izquierdo del Hunter II, para hacer girar esta rueda hacia adelante, el servo debe girar en sentido anti-horario. En lugar de 650, es necesaria una duración de 850.

Ingresar y ejecutar el **Programa 1.5** que muestra la **Figura 3.25**.

```
'{$STAMP BS2}
'Programa 1.5: Movimiento hacia adelante.
LOW 12 ' Ajusta a P12 como salida baja.
LOW 13 ' Ajusta a P13 como salida baja.
bucle: ' Etiqueta hacia donde saltar
PULSOUT 12, 650 ' Envía pulsos de 1.3 ms por P12
PULSOUT 13, 850 ' y de 1.7 ms por P13
PAUSE 20 ' cada 20 ms.
GOTO bucle ' Salta hacia la etiqueta "bucle:".
```

Figura 3.25 Programa 1.5.

Si el Hunter II se movió hacia atrás en lugar de hacia adelante, las líneas de los servos están invertidas.

Significa que el servo que debería estar conectado al puerto para servos 12, fue conectado al puerto 13 y viceversa.

Cómo funciona el programa 1.5.

Se han agregado dos líneas de programa, una para configurar a P13 como salida baja y una segunda para enviar pulsos por P13. Esto es todo lo que se necesita para controlar un segundo servo con un BASIC Stamp.

El servo derecho, que está conectado a P12, recibe un pulso con un argumento de duración de 650 (1.3 ms) para hacerlo girar en sentido horario. Mientras tanto el servo izquierdo, que está conectado a P13, recibe un pulso con un argumento de duración de 850 (1.7 ms) para girar en sentido anti-horario.

Guardar el **Programa 1.5** con otro nombre modificando los argumentos de duración de los comandos **pulsout**. Intercambiando los argumentos de duración el robot se moverá hacia atrás. Colocando los argumentos de duración para el valor central que se determinó, el robot permanecerá estático. Colocando ambos argumentos en 650 el robot rotará en su lugar en sentido anti-horario, mientras que con ambos en 850, lo hará en sentido horario.

Por ejemplo, si el robot se desvía hacia la derecha cuando se programa para ir hacia adelante en línea recta, hay que disminuir la velocidad de la rueda izquierda, o aumentar la de la derecha. Dado que los servos giran casi a su velocidad máxima, la mejor solución es disminuir la del izquierdo. Reduzca la duración del pulso del servo izquierdo (P13). En lugar de usar el comando **pulsout 13, 850**, podría intentar con **pulsout 13, 840**, esto

reducirá la velocidad de la rueda izquierda. Probando con diferentes valores (iterativamente), se encontrará finalmente los valores que hagan girar las ruedas del robot a la misma velocidad. Después de algunos intentos, el Hunter II se moverá en línea recta.

- Tomar notas de las duraciones de **pulsout** que obtuvo para el movimiento en línea recta.
- Agregar los argumentos de duración de pulso para los movimientos en línea recta en la etiqueta de cada servo, que ya tiene el valor de duración central.

3.7 Conclusión.

Finalizada esta experiencia se logro armar y probar el hunter II, realizar experiencias prácticas sobre sistemas y desarrollo de subsistemas, control y solución de problemas que llevaron a calibrar exitosamente los servos y lograr la comunicación de estos con el Basic Stamp. Con esto se ha logrado desarrollar al Hunter II como un sistema integrado que le permitirá seguir perfeccionándolo camino a un nivel básico de inteligencia, o simplemente ser programado para un fin especifico que el usuario determine.

Capítulo 4 – DESARROLLO DE EXPERIENCIA DE LABORATORIO 2

Experiencia nº 2

Titulo	:	Programación de Movimientos del Hunter II
Sala	:	……..
Profesor	:	……..

4.1 Objetivos.

El objetivo de esta experiencia es programar el Hunter II para que se mueva en distintas direcciones usando solo un programa y que a su vez ajuste la precisión de las maniobras. Además se escribirán programas que puedan almacenar listas lagas de instrucciones de movimiento.

Nota: Para todas las Actividades de esta experiencia, necesitará una computadora personal (PC) con el sistema operativo Windows 95, 98, 2000, Me o XP.

4.2 Herramientas y componentes.

(1) Hunter II armado y probado

4.3 Procedimiento general.

En la Experiencia anterior si el Hunter II se programaba para ir hacia adelante, debía ser reprogramado para que fuera hacia atrás y reprogramado nuevamente para que gire en el lugar. En esta Experiencia se incorporarán todas las direcciones en un único programa. Midiendo cuantos pulsos son necesarios para que el Hunter II rote un cierto ángulo durante un giro, se puede realizar programas para efectuar maniobras más precisas

Las maniobras preprogramadas son útiles, pero cuando se trata de largas listas de maniobras, estos programas se volverán complicados. El PBASIC tiene un método simple y eficiente para grabar y leer listas de direcciones en la memoria del programa. Observará que mientras el Hunter II realiza sus maniobras, ejecuta cambios bruscos de dirección.

Para solucionar este problema se pueden agregar comandos al programa para que el Hunter II acelere y desacelere en los cambios de dirección. Esto prolongará la vida útil de los servos.

4. 4 Controlando la distancia.

Hasta ahora, los programas del Hunter II terminaban en bucles infinitos. Por ejemplo, el Programa 1.5 hizo que el Robot se moviera hacia adelante, pero eso es todo.
Esta experiencia introduce una técnica para controlar la distancia que recorre el Hunter II.

Programación con control de distancia.

Un bucle infinito puede cumplir una tarea, pero no sabe cuándo detenerse. La mejor forma de resolver el problema es reemplazar el bucle infinito por otro llamado bucle **for…next**. Puede usar un bucle **for...next** para especificar la cantidad de veces que se ejecutarán los comandos que están en su interior.
Un bucle **for…next** usa una variable para contabilizar el número de repeticiones. Un bucle **for...next** usa la variable para almacenar un número que se incrementa cada vez que se repite el bucle. En PBASIC, una variable debe ser declarada antes de usarla. El **Programa 2.1** de la **figura 4.1** muestra ejemplos de la declaración de la variable y un bucle **for…next**. El bucle **for…next** es usado para reemplazar el bucle infinito que se usó en el **Programa 1.5**. El bucle **for...next** hace que el Hunter II se mueva hacia delante y luego se detenga, controlando la cantidad de pulsos que se envían a los servos.

- Conecte la Bateria al Hunter II.
- Conecte el cable serial al computador.
- Ingrese el Programa 2.1 en el Stamp Editor.
- Ejecute el programa haciendo clic en Run.
- Desconecte el cable serial.
- Coloque el Hunter II en la superficie sobre la cual desea que se mueva.
- Observe el comportamiento del Hunter II.

```
'{$STAMP BS2}
'Programa 2.1: Control de distancia


'-----Declaración-------

cuenta_pulsos VAR Word          ' Declara una variable para contar pulsos.

'-----Inicialización-----

LOW 12                          ' Configura a P12 y
LOW 13                          ' P13 como salidas bajas.
'-----Rutina Principal-------
principal:

adelante:                       ' Inicio de rutina.

FOR cuenta_pulsos = 1 TO 100    ' Bucle que envía 100 pulsos para adelante.

PULSOUT 12, 650                 ' Pulso de 1.3 ms en el servo derecho.
PULSOUT 13, 850                 ' Pulso de 1.7 ms en el servo izquierdo.
PAUSE 20                        ' Pausa de 20 ms.

NEXT
STOP                            ' Espera hasta el reset.
```

Figura 4.1 Programa 2.1

Cómo funciona el programa de control de distancia.

Busque los comandos **for…next** y **stop** en el **Anexo 1** antes de continuar**.** Los programas de esta experiencia estarán organizados por secciones. Tres de las cinco secciones comúnmente usadas en un programa son: declaraciones, inicializaciones y rutina principal.

Una variable llamada **cuenta_pulsos** se declara con un tamaño de dos bytes de capacidad de almacenamiento, mediante el comando **cuenta_pulsos var word**. Una variable con tamaño **word** puede almacenar números entre 0 y 65535.

El tamaño del número que una variable puede almacenar depende de cuántos bits contiene. Un simple bit, es una ubicación de memoria que puede almacenar un “1” o un “0.” A mayor cantidad de bits mayor es el número binario que puede almacenar.

Los dos primeros comandos **low** son usados para fijar los valores iniciales de las líneas de E/S usadas para controlar los servos.

La rutina **principal** usa un bucle **for...next** seguido por un comando **stop**. El bucle **for...next** reemplaza el bucle infinito **goto** usado en la experiencia anterior. Los comandos interiores al bucle **for...next,** son los comandos que envían los pulsos a los servos del Hunter II. Los valores del argumento ***duración*** usados en los comandos **pulsout,** son los valores para hacer que el Hunter II se mueva hacia adelante.

La primera vez que se ejecuta el bucle **for next**…, el valor de **cuenta_pulsos** es fijado en "1." Luego se ejecutan los dos comandos **pulsout** y el comando **pause**. Cuando el programa llega a la instrucción **next**, salta al principio de la instrucción **for**. La segunda vez que se ejecuta el bucle, la instrucción **for** le suma uno al valor de la variable **cuenta_pulsos**. Ahora el valor de **cuenta_pulsos** es "2," y la instrucción **for** controla si **cuenta_pulsos** es igual a 100. Dado que **cuenta_pulsos** aún no llega a 100, el programa continúa ejecutando las líneas interiores del bucle hasta la instrucción **nex**t. Los tres comandos, los pulsos para controlar los servos y la pausa de 20 ms, se ejecutan nuevamente y la instrucción **next** vuelve a enviar el control del programa a la instrucción **for**. El valor de **cuenta_pulsos** es incrementado y comparado con el valor final nuevamente y así sucesivamente.

La 100ª vez que el programa llega a la instrucción **next** del bucle **for...next**, envía el programa hasta la instrucción **for** nuevamente. Esta vez, el valor de **cuenta_pulsos** es incrementado a 101. Ahora, cuando **cuenta_pulsos** es comparado con el argumento ***duración***, se encuentra que es mayor que el valor límite 100. Así que en lugar de ejecutar los comandos que envían otro pulso, la instrucción **for** envía el control del programa a la instrucción inmediatamente posterior a **next**.

El comando **stop** es ejecutado inmediatamente después del bucle **for...next** y hace que el programa se detenga. Lo único que puede reactivar un BASIC Stamp después de un comando **stop** es un reset (reinicio) por hardware.

4.5 Maniobras – haciendo giros.

Si el mismo valor es sumado al ancho de pulso central de un servo y restado del ancho de pulso central del otro, el Hunter II se moverá en línea recta, hacia adelante o atrás. Cuando el servo derecho recibe una duración de **pulsout** de 650 (1.3 ms) y el servo izquierdo recibe una duración de **pulsout** de 850 (1.7 ms), el Hunter II se mueve hacia

adelante. Cuando se intercambian las duraciones de pulsos de cada servo, el Hunter II va hacia atrás. Si ambos servos reciben pulsos de 1.3 ms, girarán en la misma dirección, haciendo que el Hunter II gire en sentido anti-horario. Si se aplican muchos pulsos, el Hunter II seguirá rotando. Si se envían unos 30 pulsos, el efecto neto será un giro a la izquierda de 90°. El mismo principio se aplica si ambos servos reciben pulsos de 1.7 ms, girando el Hunter II en sentido horario, en lugar de anti-horario.

Programando giros a la derecha e izquierda.

El Programa 2.2 de la **Figura 4.2** muestra como modificar las rutinas **adelante** y **atrás** del Programa 2.1 para lograr que el Hunter II realice giros.
Ingresar y ejecutar el Programa 2.2.

```
'{$STAMP BS2}
'Programa 2.2: Girando en el lugar.

'-----Declaración-------

cuenta_pulsos VAR Word       ' Declara un contador tipo word.

'-----Inicialización-----

LOW 12              ' Fija a P12 y 13 como salidas en
LOW 13              ' estado bajo.

'-----Rutina Principal-------

principal:              ' Rutina principal
giro_izq:               ' Rutina de giro a la izquierda.
FOR cuenta_pulsos = 1 TO 30     ' Envía 30 pulsos.
PULSOUT 12, 650         ' Pulso de 1.3 ms al servo derecho.
PULSOUT 13, 650         ' Pulso de 1.3 ms al servo izquierdo.

PAUSE 20                ' Pausa de 20 ms.
NEXT
PAUSE 500               ' Pausa de 0.5 s.

giro_der:           ' Rutina de giro a la derecha.
FOR cuenta_pulsos = 1 TO 30     ' Envía 30 pulsos.
PULSOUT 12, 850     ' Pulso de 1.7 ms al servo derecho.
PULSOUT 13, 850     ' Pulso de 1.7 ms al servo izquierdo.

PAUSE 20        ' Pausa de 20 ms.
        NEXT
PAUSE 500       ' Pausa de 0.5 ms.

STOP            ' Se detiene hasta un reset.
```

Figura 4.2 Programa 2.2

Cómo funciona el programa 2.2.

Solamente se hicieron tres modificaciones al Programa 2.1 para hacer el Programa 2.2. Primero, las etiquetas **adelante:** y **atras:** se cambiaron por **giro_izq:** y **giro_der:** respectivamente. Luego, se fijaron los dos argumentos de duración de **pulsout** de la rutina **giro_izq** en 650. Los argumentos de **pulsout** de la rutina **giro_der** se fijaron en 850.

4.6 Recordando listas largas usando la EEPROM.

El BASIC Stamp almacena una versión tokenizada del programa PBASIC en su memoria de sólo lectura, programable y borrable eléctricamente (EEPROM). Físicamente, la EEPROM es el pequeño chip negro del módulo BASIC Stamp II rotulado "24LC16B". La EEPROM del BASIC Stamp puede almacenar 2048 bytes (2 kB) de información. Lo que no se usa para almacenar el programa (que comienza desde la dirección 2047 hacia la dirección 0) puede ser usado para almacenar datos (comenzando desde la dirección 0 hacia la dirección 2047).

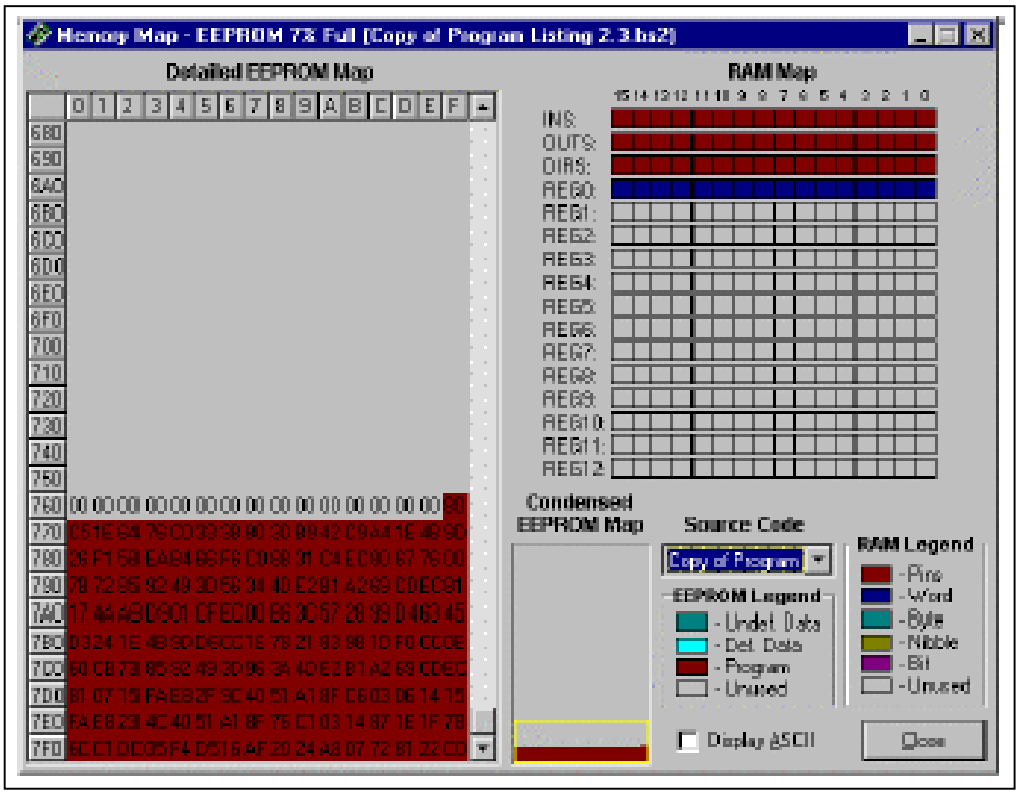
La memoria EEPROM es diferente de la de almacenamiento de variables (RAM), en varios aspectos:

- La EEPROM necesita más tiempo para almacenar datos, algunas veces hasta varios milisegundos.
- La EEPROM solamente tolera un número limitado de ciclos de escritura, de aproximadamente 10 millones de ciclos. La RAM tiene capacidades ilimitadas de lectura/escritura.
- La función primaria de la EEPROM es almacenar programas; los datos se almacenan en el espacio sobrante.

Puede ver el contenido de la memoria EEPROM del BASIC Stamp haciendo clic en Run y seleccionando Memory Map. **La Figura 4.3** muestra el mapa de memoria del Programa 2.2. Observe el Condensed EEPROM Map (Mapa de EEPROM Condensado) en la parte inferior central de la figura. La franja horizontal angosta de color rojo representa la pequeña porción de EEPROM que ocupa el Programa 2.2.

Este programa puede haberle parecido grande mientras lo escribía, pero sólo ocupa 128 de los 2048 bytes de memoria de programa disponibles.

Hay suficiente lugar para una larga lista de instrucciones. Una de las aproximaciones más simples es almacenar caracteres que indiquen en que sentido ir. Dado que los caracteres ocupan un byte de memoria, hay lugar suficiente para 1920 instrucciones de dirección de un caracter.



La Figura 4.3 Contenido de la memoria EEPROM.

Navegación por EEPROM.

El comando de instrucción **data** es colocada normalmente en la sección **declaración** de un programa en PBASIC y es la mejor forma de almacenar listas largas. El programa 2.3 de la **Figura 4.4** muestra un ejemplo de cómo este comando **data** puede ser usada para almacenar instrucciones de navegación para el Hunter II.

```
'Programa 2.3: Navegación por EEPROM

'----- Declaración --------------          ' Rótulo para la rutina de declaración.

cuenta_pulsos var word                     ' Declara la variable cuenta_pulsos.
direc_EE var byte                          ' Almacena e inc. la dirección EEPROM.
instruccion var byte                       ' Almacena el caracter recuperado de la EE.

data "FFFBBLFFRFFQ"                        ' Lista de instrucciones para el Boe-Bot.

'----- Inicialización ------------

       output 2                            ' Fija como salida la línea del parlante.
       freqout 2, 2000, 3000               ' Emite tono indicador de batería baja.
       low 12                              ' Configura P12 como salida baja.
       low 13                              ' Configura P13 como salida baja.

'----- Rutina Principal --------------

principal:                                 ' Etiqueta de la rutina principal.

       read direc_EE, instruccion          ' Lee la direc_EE y almacena la instrucción
       direc_EE = direc_EE + 1             ' Incrementa direc_EE para la siguiente lectura
       if instruccion = "F" then adelante  ' Busca instrucción adelante.
       if instruccion = "B" then atras     ' Busca instrucción atras.
       if instruccion = "R" then giro_der  ' Busca instrucción giro_der.
       if instruccion = "L" then giro_izq  ' Busca instrucción giro_izq.

stop ' Detiene el programa hasta un reset

'----- Rutinas de Navegación -------

adelante:                                  ' Rutina desp. adelante.
       for cuenta_pulsos = 1 to 60         ' Envía 60 pulsos hacia adelante.
              pulsout 12, 650              ' Pulsos de 1.3 ms al servo derecho.
              pulsout 13, 850              ' Pulsos de 1.7 ms al servo izquierdo.
              pause 20                     ' Pausa de 20 ms.
       next
goto principal                             ' Envía el programa a la rutina principal.

atras:
       for cuenta_pulsos = 1 to 60         ' Envía 60 pulsos hacia atrás.
              pulsout 12, 850              ' Pulsos de 1.7 ms al servo derecho.
              pulsout 13, 650              ' Pulsos de 1.3 ms al servo izquierdo.
              pause 20                     ' Pausa de 20 ms.
       next
goto principal                             ' Envía el programa a la rutina principal.
```

```
giro_izq:                               ' Rutina giro izquierda.
        for cuenta_pulsos = 1 to 30     ' Envía 30 pulsos de giro.
                pulsout 12, 650         ' Pulsos de 1.3 ms al servo derecho.
                pulsout 13, 650         ' Pulsos de 1.3 ms al servo izquierdo.
                pause 20                ' Pausa de 20 ms.
        next
goto principal                          ' Envía el programa a la rutina principal.


giro_der:                               ' Rutina giro derecha.
        for cuenta_pulsos = 1 to 30     ' Envía 30 pulsos de giro.
                pulsout 12, 850         ' Pulsos de 1.7 ms al servo derecho.
                pulsout 13, 850         ' Pulsos de 1.7 ms al servo izquierdo.
                pause 20                ' Pausa de 20 ms.
        next
goto principal                          ' Envía el programa a la rutina principal.
```

Figura 4.4 Programa 2.3.

Cómo funciona el programa de navegación por EEPROM.

Buscar los comandos **data**, **read** e **if...then** en el **Anexo 1** antes de continuar**.** El Programa 2.4 introduce dos nuevas variables. En lugar de ser de tipo word, son tipo byte, lo que significa que pueden almacenar números entre cero y 255. La primer variable es **direc_EE**, que es usada para especificar la dirección de la EEPROM a leer con el comando **read**. La segunda variable se llama **instruccion**. Esta variable es usada para almacenar el caracter de instrucción leído de la EEPROM.

```
'----- Declaración ----------
cuenta_pulsos var word
direc_EE var byte
instruccion var byte
```

La siguiente declaración es la lista de datos a ser almacenada en la EEPROM. Estos datos son almacenados como una cadena de caracteres. Cuando estos caracteres son almacenados, son guardados como números que corresponden a las letras que ve en el

comando **data**. En esta experiencia, verá el mapa de memoria y observará los caracteres y su representación numérica en la EEPROM.

```
data "FFFBBLFFRFFQ"      ' Datos de movimiento
```

Aunque la rutina de inicialización es la misma que se usó en el Programa 2.2, no ocurre lo mismo con la rutina principal. La rutina principal primero lee el valor en la dirección 0 de la EEPROM y lo almacena en la variable **instruccion**. Luego, **direc_EE** es incrementado para que en el siguiente ciclo de lectura señale la dirección 1. Una serie de instrucciones **if...then** son usadas para decidir qué hacer, basándose en el caracter leído de la EEPROM, mediante la variable **instruccion**. Las instrucciones **if...then** buscan uno de los cuatro caracteres de instrucción: “F,” “B,” “R,” y “L.” Por ejemplo, si el caracter es una “R,” las primeras dos instrucciones **if...then** son salteadas debido a que ninguna es verdadera. Como la tercera instrucción **if...then** es verdadera, el programa salta a la rutina **giro_der**.

```
principal:
read direc_EE, instruccion
direc_EE = direc_EE + 1
if instruccion = "F" then adelante
if instruccion = "B" then atras
if instruccion = "R" then giro_der
if instruccion = "L" then giro_izq
stop
```

Cuando el programa ingresa en la rutina **giro_der**, ejecuta 30 pulsos de giro a la derecha. Luego el comando **goto principal** envía el programa hacia la rutina **principal**.

```
giro_der:
for cuenta_pulsos = 1 to 30
pulsout 12, 850
pulsout 13, 850
pause 20
next
goto principal
```

Cuando el programa regresa a la rutina **principal**, se lee la siguiente instrucción de la EEPROM y es controlada por las cuatro instrucciones **if...then**. El proceso se repite hasta que el caracter de salida “Q” es leído de la EEPROM. Cuando se carga “Q” en la variable **instruccion**, fallan las cuatro pruebas de **if...then**. En este caso, el programa no salta a ninguna de las rutinas de navegación, sino que ejecuta el comando que se encuentra a continuación de la serie de instrucciones **if...then**, que es el comando **stop**.

4.7 Conclusión.

En esta experiencia se logro que el Hunter II fuera capaz de desarrollar rutinas de múltiples movimiento en base a un solo programa ingresado a la Basic Stamp, dando como resultado un programa que mide los pulsos que son necesarios para que el Hunter II rote un cierto ángulo durante un giro, además con este tipo de programas se pueden lograr maniobras más precisas, por ejemplo que el Hunter II realice movimientos que describen cuadrados, sinusoides o triángulos todo esto dependiendo de las necesidades que el programador requiera satisfacer para el uso del Hunter II.

En el camino de esta experiencia se logro perfeccionar aun más al Hunter II como sistema, de esta manera se completa la preparación para dejarlo a las puertas de darle inteligencia o la capacidad de tomar decisiones que le permitan lograr el objetivo para el cual sea programado.

Capítulo 5– DESARROLLO DE EXPERIENCIA DE LABORATORIO 3

Experiencia nº 3

Titulo	:	Programación de Fotorresistores
Sala	:	……..
Profesor	:	……..

5.1 Objetivos.

El Objetivo de esta experiencia es programar el Hunter II para probar sensores fotosensibles y hacer funcionar el robot como un compás de luz, para luego escribir un programa que le permita seguir una fuente de luminosidad.

Nota: Para todas las actividades de esta experiencia, necesitará una computadora personal (PC) con el sistema operativo windows 95, 98, 2000, Me o XP.

5.2 Herramientas y componentes.

La **Figura 5.1** muestra los componentes introducidos en esta experiencia, junto con sus símbolos esquemáticos. Debajo hay una lista de los componentes que necesitará, todos los componentes son no polarizados, lo que quiere decir que los terminales 1 y 2 pueden ser invertidos sin afectar el circuito.

(1) piezoeléctrico
(2) fotorresistores
(2) capacitores 0.1 uF
(2) capacitores 0.01 uF
(2) resistores 220 Ohm
(Varios) cables de interconexión

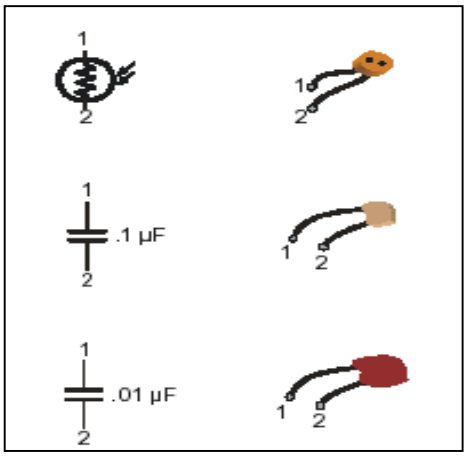


Figura 5.1 Componentes.

5.3 Procedimiento general.

Los fotorresistores pueden ser usados para que el Hunter II detecte variaciones en el nivel de luz. Con un poco de programación, Hunter II puede ser transformado en fotófilo (una criatura que es atraída por la luz), o fotofóbico (una criatura que evita la luz).

Para medir la presencia e intensidad de luz, se construirá un par de circuitos con fotorresistores montados en el Hunter II.

Un fotorresitor es un resistor cuyo valor depende de la intensidad de luz (light-dependent resistor o LDR), que cubre un espectro similar al del ojo humano. Los elementos activos de estos fotorresistores están hechos de sulfuro de cadmio (CdS). La luz entra en la capa semiconductora que está aplicada sobre un sustrato cerámico y produce portadores de carga libres. Se produce una resistencia eléctrica definida, que es inversamente proporcional a la intensidad de iluminación. En otras palabras, la oscuridad produce valores altos de resistencia y la claridad produce valores pequeños.

5.4 Programación y prueba de sensores fotosensibles.

La **Figura 5.2** muestra (a) el circuito resistor/capacitor (RC) para cada fotorresistor y (b) un ejemplo de armado sobre la protoboard. Un fotorresistor es un dispositivo analógico. Su valor cambia en forma continua de acuerdo a la luminancia, que es otro valor analógico. La resistencia del fotorresistor es muy baja cuando se expone directamente a la luz del sol. A medida que desciende el nivel de luz, su resistencia aumenta. En

completa oscuridad, la resistencia del fotorresistor puede alcanzar valores de hasta 1 MΩ. Aunque el fotorresistor es analógico, esto no quiere decir que sea lineal. Esto significa que si la fuente de luz (luminancia) varía a una proporción constante, el valor del fotorresistor no necesariamente variará a una proporción constante.

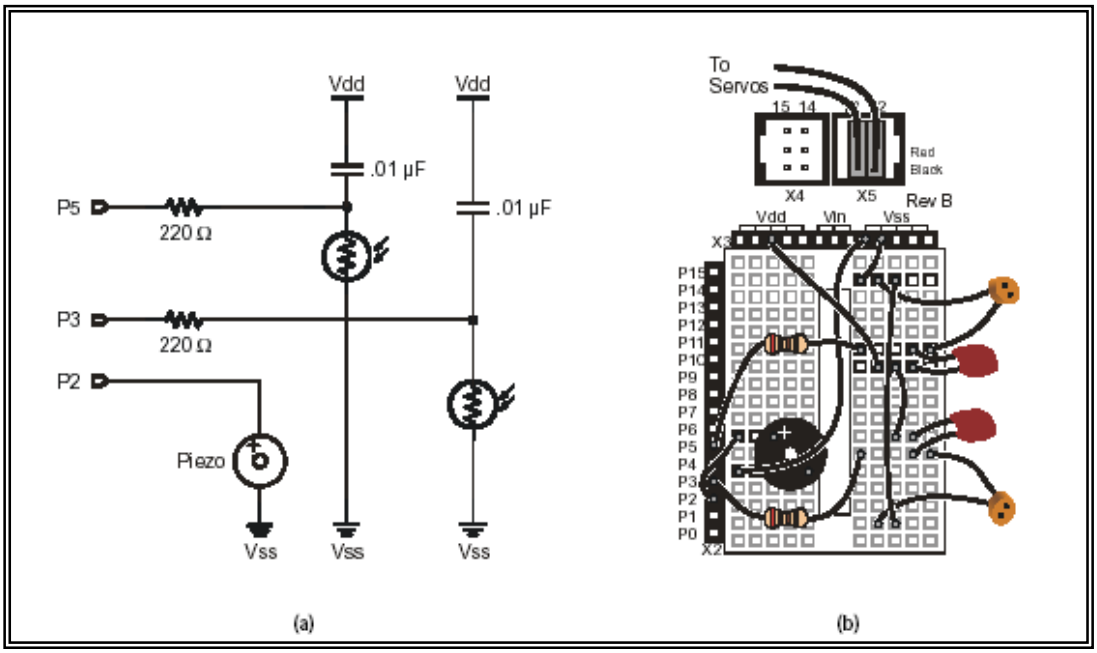


Figura 5.2: (a) Dos circuitos RC para medir resistores que varían con la luz y (b) ejemplo de distribución en la protoboard.

Programa para medir la resistencia.

El circuito de la **Figura 5.2** (a) fue diseñado para ser usado con el comando **rctime**. Este comando puede ser usado con un circuito RC donde uno de los valores, R o C, varía, mientras el otro permanece constante. El comando **rctime** puede medir valores variables debido a que toma ventaja de las variaciones de tiempo que generan los circuitos RC. El tiempo que demora un circuito RC en variar una tensión depende de R × C, llamada constante de tiempo RC. La constante de tiempo RC a menudo es representada por la letra griega Tau (τ).

Para uno de los circuitos RC mostrados en la **Figura 5.2** (a), el primer paso para tomar una lectura con **rctime** es descargar el capacitor, colocando 5 Volt en la placa inferior. Si

se configura como salida en estado alto al pin de E/S correspondiente, en pocos milisegundos se lograra esto. Luego, el comando **rctime** puede ser usado para medir el tiempo que le lleva al capacitor cargarse, hasta que la tensión en el pin cambie de 5 a 1.4 Volt. ¿Por qué 1.4 Volt?. Debido a que este es el umbral de tensión de los pines del BASIC Stamp. Cuando la tensión de un pin es mayor de 1.4 Volt, el valor del bit de registro de entrada conectado a ese pin de E/S es “1”. Cuando la tensión cae por debajo de 1.4 Volt, el valor del bit de registro de entrada es “0.”

El comando **rctime** del circuito que se muestra en la **Figura 5.2** mide cuánto tiempo demora la tensión de la placa inferior de capacitor en bajar de 5 a 1.4 Volt. Este tiempo varía con la fórmula:

$$\frac{t}{R \times C} = \ln\left(\frac{V_{inicial}}{V_{final}}\right)$$

$$\frac{t}{R \times 0.01 \times 10^{-6}} = \ln\left(\frac{5\ V}{1.4\ V}\right)\ s$$

$$t = \ln(3.57) \times R \times 0.01 \times 10^{-6}\ s$$

$$t = 1.27 \times 10^{-8} \times R\ \ s$$

Figura 5.3 Ecuación 5.1.

La ecuación 5.1 de la **Figura 5.3** indica que el tiempo que tarda en caer la tensión de la placa inferior de uno de los capacitores de los circuitos RC de la **Figura 5.2 (a)**, es directamente proporcional a la resistencia del fotorresistor. Dado que la resistencia varía con la luminancia (exposición a niveles variados de luz), así también lo hace el tiempo.

Midiendo este tiempo, se puede inducir la exposición de luz relativa.

El comando **rctime** cambia la dirección de un pin de E/S, de salida a entrada. Tan pronto como el pin se convierte en entrada, la tensión de la placa inferior del capacitor empieza a caer, de acuerdo a la ecuación temporal anterior. El BASIC Stamp comienza a contar en incrementos de 2 microsegundos hasta que el nivel de tensión de la placa inferior del capacitor cae por debajo de 1.4 Volt.

Ejecutar el Programa 3.1 que muestra la **Figura 5.4**, este programa demuestra cómo usar el comando **rctime** para leer fotorresistores. Este programa usa la Debug Terminal, así

que deberá dejar conectado el cable serial a la hunter II mientras se esté ejecutando el Programa 3.1.

```
'{$STAMP BS2}
' Programa 5.1: Fotorresitores y rctime
'----- Declaración --------------
foto_izq VAR Word ' Almacenan los tiempos RC de los
foto_der VAR Word ' fotorresitores izq. y derecho.
'----- Inicialización ------------
DEBUG CLS              ' Abre y limpia la Debug Terminal.
'----- Rutina Principal --------------
principal:
' Mide el tiempo RC del fotorresitor derecho.
HIGH 3                 ' Fija a P3 como salida alta.
PAUSE 3                ' Pausa de 3 ms.
RCTIME 3,1,foto_der ' Mide tiempo RC en P3.
' Mide el tiempo RC del fotorresitor izquierdo.
HIGH 5                 ' Fija a P5 como salida alta.
PAUSE 3                ' Pausa de 3 ms.
RCTIME 5,1,foto_izq ' Mide tiempo RC en P5.
' Muestra las mediciones de tiempo RC usando la Debug Terminal.
DEBUG HOME, "I ", DEC5 foto_izq, " D ", DEC5 foto_der
GOTO principal
```

Figura 5.4 Programa 3.1.

Cómo funciona fotorresitores y rctime.

Dos variables tipo word, **foto_izq** y **foto_der** se declaran para almacenar los valores de los tiempos RC de los fotorresistores izquierdo y derecho. La rutina **principal** mide y muestra los valores para cada circuito RC. El código para leer el circuito RC derecho se muestra a continuación. Primero, el pin de E/S P3 se fija como salida en estado alto. Luego, una pausa de 3 milisegundos le permite al capacitor descargarse.

Luego de los 3 milisegundos, la placa inferior del capacitor está suficientemente cerca de 5 Volt y está lista para la medición de **rctime**. El comando **rctime** mide el tiempo RC en el pin P3, comenzando en el estado “1” (5 Volt) y almacena el resultado en la variable **foto_der**. Recuerde, el valor almacenado en **foto_der** es un número, este número dice cuántos incrementos de 2 microsegundos pasaron antes que la tensión de la placa inferior del capacitor cayera por debajo del umbral de 1.4 Volt.

high 3
pause 3
rctime 3,1,foto_der

5.5 Hunter II como un compás de luz.

Si se enfoca un haz de luz en el frente del Hunter II, el circuito y las técnicas de programación recién discutidas pueden ser usadas para que el Hunter II gire y se dirija hacia la luz. Asegurar que los fotorresistores estén colocados de forma que puedan realizar una comparación de luz incidente, deberían estar a un ángulo de 45° de la línea central del Hunter II hacia afuera y a 45° por debajo de la horizontal. En otras palabras, apuntar las caras de los fotorresistores hacia la mesa. Luego, use una luz brillante para hacer que el Hunter II siga la dirección de la luz.

Programar el hunter II para que apunte hacia la luz.

Para lograr que el Hunter II siga una fuente de luz, se debe comparar en el programa los valores medidos en cada fotorresistor. Debe recordarse que a medida que la intensidad de luz se debilita, el valor del fotorresistor aumenta. Así, si el valor del fotorresistor de la derecha es mayor que el de la izquierda, significa que hay más claridad a la izquierda. Dada esta situación, el Hunter II debería girar a la izquierda. Por otro lado, si **rctime** del fotorresistor de la izquierda es mayor que el de la derecha, el lado derecho es el más luminoso, así que el Hunter II debería girar a la derecha.

Para evitar que el Hunter II cambie de dirección muy seguido, se introduce un intervalo (**deadband**) en el que no debería haber intentos de corrección. Si los números van por encima o por debajo de este intervalo, el sistema efectuará las correcciones necesarias. La forma más conveniente de medir este intervalo es restar el **rctime** izquierdo del **rctime** derecho, o viceversa y luego tomar el valor absoluto. Si este valor absoluto está dentro de los límites del intervalo, no se generará respuesta; caso contrario, programe el ajuste apropiado. Ingresar y ejecutar el Programa 3.2 mostrado en la **Figura 5.5**.

Al encender una luz brillante en el frente del Hunter II este debería rotar para quedar apuntando a la fuente de luz.

Ahora en lugar de usar una linterna, usar la mano o una hoja para provocar una sombra en uno de los fotorresistores. El Hunter II debería rotar para alejarse de la sombra.

```
'Programa 3.2: Compás de Luz
'----- Declaración ---------------
foto_izq VAR Word                    ' Almacenan los tiempos RC de los
foto_der VAR Word                    ' fotorresitores izq. y derecho.

'----- Inicialización ------------
LOW 12                               ' P12 y 13 salidas bajas.
LOW 13

'----- Rutina Principal ----------------
principal:
' Mide el tiempo RC del fotorresitor derecho.
HIGH 3                               ' Fija a P3 como salida alta.
PAUSE 3                              ' Pausa de 3 ms.
RCTIME 3,1,foto_der                  ' Mide tiempo RC en P3.

' Mide el tiempo RC del fotorresitor izquierdo.
HIGH 5                               ' Fija a P5 como salida alta.
PAUSE 3                              ' Pausa de 3 ms.
RCTIME 5,1,foto_izq                  ' Mide tiempo RC en P5.

' Toma la diferencia entre foto_der y foto_izq, luego decide qué hacer.
IF ABS(foto_izq-foto_der) < 2 THEN principal
IF foto_izq > foto_der THEN pulso_der
IF foto_izq < foto_der THEN pulso_izq

'----- Rutinas de Navegación -------
pulso_izq:          ' Aplica un pulso a la izq., luego
PULSOUT 12, 650
PULSOUT 13, 650
PAUSE 10
GOTO principal      ' regresa a la rutina principal.
pulso_der:          ' Aplica un pulso a la der., luego
PULSOUT 12, 850
PULSOUT 13, 850
PAUSE 10
GOTO principal      ' regresa a la rutina principal.
```

Figura 5.5 Programa 3.2

Cómo trabaja el compás de luz.

El **Programa 3.2** mide los tiempos RC y controla si la diferencia entre los valores obtenidos por los comandos **rctime** está dentro del intervalo de tolerancia (en el que no toma ninguna acción):

if abs(foto_izq - foto_der) < 2 then principal si la diferencia está dentro del intervalo, el programa salta a la etiqueta **principal:**. Si la diferencia medida supera el valor del intervalo, dos instrucciones **if...then** deciden a qué rutina llamar, **pulso_izq** o **pulso_der**.

if foto_izq > foto_der then pulso_der

if foto_izq < foto_der then pulso_izq

La rutina **pulso_izq** se muestra a continuación. Observe que los comandos **pulsout** y el comando **pause** no están anidados dentro del bucle **for...next**. En lugar de eso, se emite un único pulso con una pausa un poco más pequeña de lo normal y luego el control regresa a la rutina **principal**. Esto le permite al programa controlar y actualizar los valores de los fotorresistores entre pulsos de los servos. Observe que la pausa no es de 20 ms. Esto es debido a que cada comando **rctime** lleva cierta cantidad de tiempo, que puede restarse de la pausa. Para las condiciones de luz usadas en este ejemplo, el promedio de las pausas causadas por **rctime** fue de 10 milisegundos. Sus condiciones probablemente sean diferentes.

```
pulso_izq:
pulsout 12, 650
pulsout 13, 650
pause 10
goto principal
```

5.6 Programar hunter II para que siga una fuente de luz.

Simplemente agregando movimiento hacia adelante al Hunter II, puede convertirlo en un vehiculo seguidor de luz, un fotófilo. Un experimento interesante es tratar de programar el Hunter II para que se mueva hacia adelante y busque la luz. Luego, colocarlo en una habitación oscura con la puerta abierta hacia un ambiente iluminado. Asumiendo que no haya obstáculos en su camino, el Hunter II se dirigirá a la puerta y saldrá de la habitación oscura.

Programa para Seguir la Luz.

Programar el Hunter II para que siga la luz, solamente requiere unas pocas modificaciones al **Programa 3.2**. La principal modificación es que una medición que caía dentro de la **deadband**, daba como resultado cero movimientos en el **Programa 3.2**. En el Programa 3.3 que muestra la **Figura 5.6**, cuando las diferencias en rctime caen dentro de la deadband, dan como resultado movimiento hacia adelante. Veamos cómo se hace.

```
'Programa 3.3: Seguidor de luz.
'----- Declaración --------------

foto_izq VAR Word      ' Almacenan los tiempos RC de los
foto_der VAR Word      ' fotorresitores izq. y derecho.

'----- Inicialización ------------
OUTPUT 2               ' Fija a P2 como salida.
FREQOUT 2, 2000, 3000 ' Indicador de reset.
LOW 12                 ' P12 y 13 salidas bajas.
LOW 13

'----- Rutina Principal --------------
principal:
' Mide el tiempo RC del fotorresitor derecho.
HIGH 3                 ' Fija a P3 como salida alta.
PAUSE 3                ' Pausa de 3 ms.
RCTIME 3,1,foto_der    ' Mide tiempo RC en P3.
' Mide el tiempo RC del fotorresitor izquierdo.
HIGH 5                 ' Fija a P5 como salida alta.
PAUSE 3                ' Pausa de 3 ms.
RCTIME 5,1,foto_izq    ' Mide tiempo RC en P5.

IF ABS(foto_izq-foto_der) > 2 THEN control_dir
adelante_pulso:
PULSOUT 12, 650
PULSOUT 13, 850
PAUSE 20
GOTO principal
' Salta hacia giro_der o giro_izq dependiendo del mayor tiempo RC.
control_dir:
IF foto_izq > foto_der THEN pulso_der
IF foto_izq < foto_der THEN pulso_izq
```

```
'----- Rutinas de Navegación -------
pulso_izq: ' Aplica un pulso a la izq., luego
PULSOUT 12, 650
PULSOUT 13, 650
PAUSE 20
GOTO principal ' regresa a la rutina principal.
pulso_der: ' Aplica un pulso a la der., luego
PULSOUT 12, 850
PULSOUT 13, 850
PAUSE 20
GOTO principal ' regresa a la rutina principal.
```

Figura 5.6 Programa 3.3.

Cómo funciona el programa seguidor de luz.

Como en el Programa 3.2, la primera instrucción **if...then** controla si la diferencia en las mediciones del tiempo RC cae dentro de la **deadband**. Esta instrucción ha sido modificada para que saltee la rutina **adelante_pulso** si la diferencia de tiempos RC cae fuera de la **deadband**. Por otro lado, si la diferencia del tiempo RC cae en la **deadband**, se ejecuta un pulso adelante. Luego, el programa vuelve a **principal** y controla nuevamente los tiempos RC.

```
if abs(foto_izq-foto_der) > 2 then control_dir
adelante_pulso:
pulsout 12, 650
pulsout 13, 850
pause 20
goto principal
```

Si la diferencia entre tiempos RC no cae dentro de la deadband, el programa salta a la etiqueta **control_dir**. Las instrucciones **if...then** que están a continuación de la etiqueta **control_dir** se usan para decidir si se aplicará un pulso a la izquierda o un pulso a la derecha, dependiendo de la diferencia entre los valores **foto_der** y **foto_izq**. De esta forma, el programa aplica un pulso hacia adelante o un pulso de giro, cada vez que se controlan los fotorresistores.

```
control_dir:
if foto_izq > foto_der then pulso_der
if foto_izq < foto_der then pulso_izq
```

5.7 Conclusión.

Con el desarrollo de esta experiencia se logro dar un grado de inteligencia artificial al Hunter II que le permite con los fotorresistores detectar variaciones en el nivel de luz y procesar una decisión donde la Basic Stamp enviara una orden a los actuadores de acuerdo al programa o rutina que se desee ejecutar, con esto se logra transformar al prototipo Hunter II en un vehículo explorador fotosensible, dejando abierta posibilidad a la imaginación del programador para usar el Hunter II para algún fin mas especifico.

Capítulo 6 – DESARROLLO DE EXPERIENCIA DE LABORATORIO 4

Experiencia nº 4

Titulo	:	Programación de sensores táctiles
Sala	:	……..
Profesor	:	……..

6.1 Objetivos

El objetivo de esta experiencia es instalar y programar sensores táctiles de navegación que le permitan al Hunter II explorar e interactuar con el medio, y a su vez librar obstáculos y decidir cuando se encuentre atrapado.

Nota: Para todas las actividades de esta experiencia, necesitará una computadora personal (PC) con el sistema operativo Windows 95, 98, 2000, Me o XP.

6.2 Herramientas y componentes.

Para esta experiencia se utilizaran los siguientes componentes.

Materiales.

(2) Resistores de 10 kΩ
(2) Conectores de 3 pines.
(2) Separadores macho/hembra.
(2) Tornillos.
(2) Sensores táctiles.
(2) Arandelas de nylon.

6.3 Procedimiento general.

Los sensores táctiles le dan al Hunter II la capacidad de reconocer el mundo exterior. El Hunter II puede usar estos sensores para navegar solamente por tacto. Aunque las actividades de esta experiencia se enfocan en usar solamente los sensores táctiles, también pueden usarse junto con otros sensores para aumentar la funcionalidad del Hunter II.

6.4 Instalar y probar los sensores táctiles.

La Figura 6.2 muestra el diagrama de conexión de los sensores. Seguir este esquema para realizar las conexiones eléctricas necesarias y asegúrese de orientar los sensores táctiles como en la **Figura 6.1**.

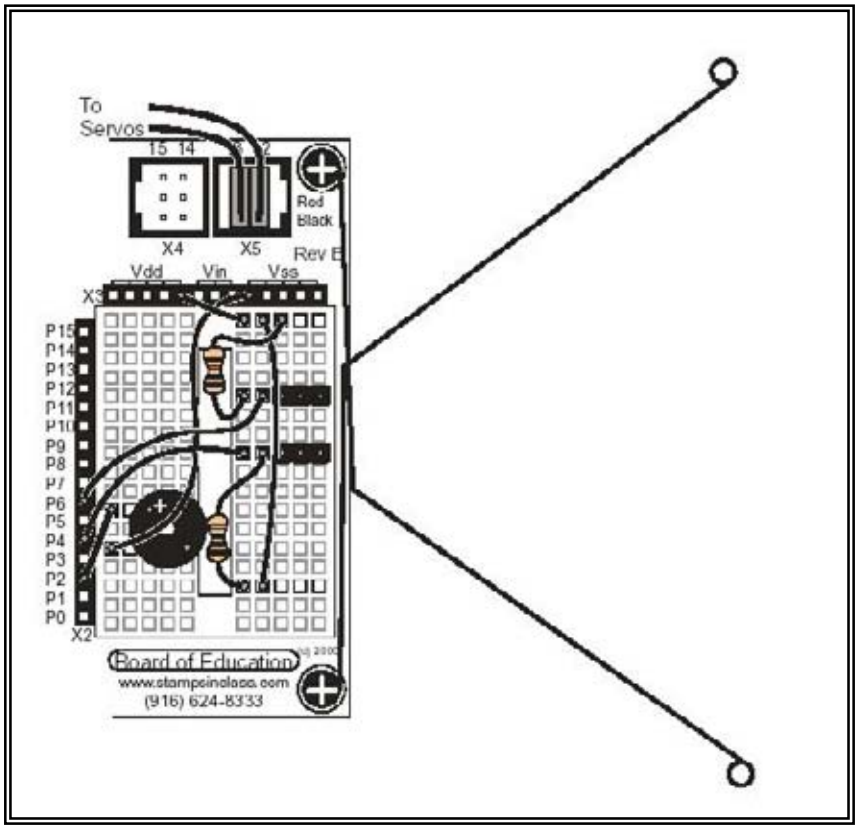


Figura 6.1 Diagrama de conexión de sensores táctiles.

Usar los sensores táctiles como entradas.

La Figura 6.3 es una representación esquemática del circuito. Cada sensor es una extensión mecánica, así como también la conexión eléctrica a masa de un interruptor normalmente abierto. El BASIC Stamp puede ser programado para detectar cuándo un bigote es presionado. Los pines de E/S conectados a cada interruptor, controlan la tensión en el resistor de 10 kΩ. Cuando un sensor no está presionado, la tensión del pin es 5 Volt (1 lógico). Cuando un sensor es presionado, la línea de E/S es puesta a masa, de forma que se detecta 0 Volt (0 lógico).

Probando los sensores táctiles.

Se puede probar cada sensor con la Debug Terminal. Esta vez, en lugar de imprimir un mensaje, la Debug Terminal es usada para mostrar la tensión de entrada que se observa en los pines de E/S conectados a los sensores. Cada pin de E/S del BASIC Stamp está conectado a una posición de memoria RAM. Estas posiciones de memoria pueden verse en el Memory Map (Mapa de Memoria) que se muestra en la **Figura 6.3**.

- Abrir el Mapa de Memoria seleccionando Memory Map en el menú Run del Stamp Editor.

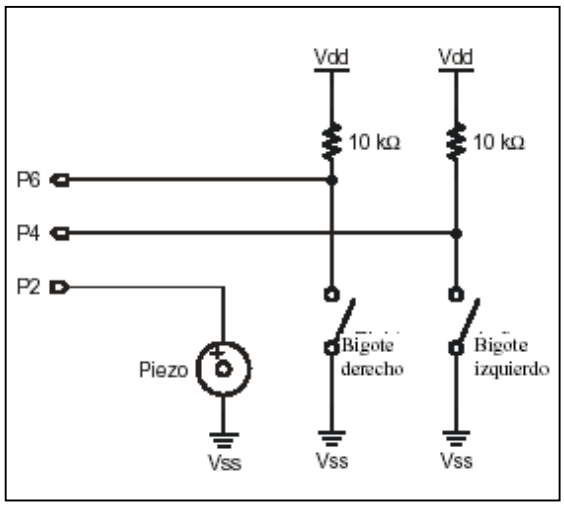


Figura 6.2 Esquema de Sensores Tactiles.

Las primeras tres filas de la RAM Map son ubicaciones de memoria conectadas directamente a los pines de E/S. La fila de números de la parte superior indica la dirección

de bit para cada posición de una fila dada. Las filas **ins**, **outs** y **dirs** son los registros de E/S.

La primera fila corresponde al registro **ins** y representa una zona de memoria que almacena 16 bits, cada uno de los cuales puede ser 1 o 0. Cada bit del registro **ins** controla el estado de un pin de E/S. Bit-0 corresponde al pin de E/S P0, bit-1 a P1 y así hasta bit-15, que corresponde al pin de E/S P15.

Cada bit del registro **outs**, puede ser usado para fijar el valor de salida de un pin de E/S en 5 ó 0 Volt. Cuando el valor de un bit del registro **outs** es 1, la salida del pin correspondiente será 5 Volt. Cuando el valor del bit es 0, la salida será 0 Volt.

Dado que un pin de E/S de un BASIC Stamp no puede ser entrada y salida a la vez, la dirección de cada pin es fijada por el registro **dirs**. Un pin de E/S es fijado como entrada o salida, dependiendo del valor del registro **dirs**.

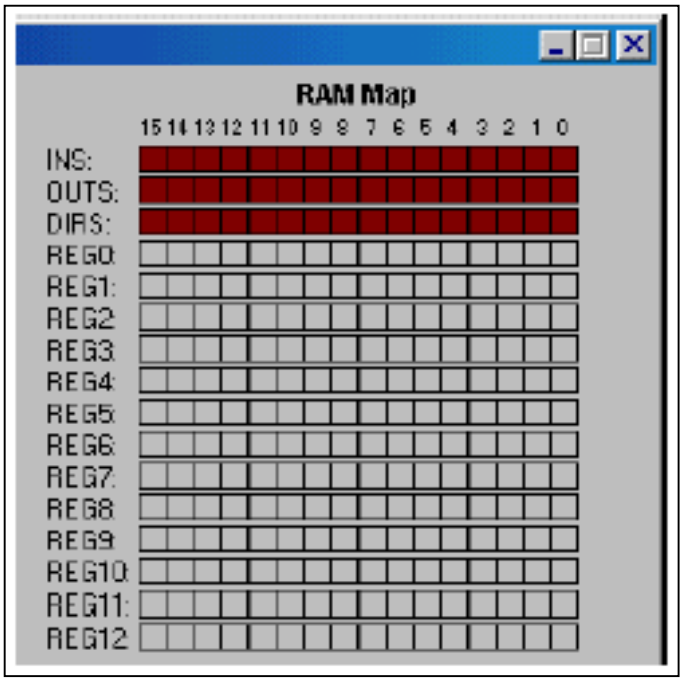


Figura 6.3 Mapa RAM de la Ventana Memory Map.

El Programa 4.1 está diseñado para verificar que los sensores táctiles funcionen apropiadamente. Controla y muestra el estado de los pines de E/S del BASIC Stamp conectados a los sensores táctiles. Todos los pines de E/S se configuran como entradas cada vez que se inicia un programa PBASIC, esto significa que los pines conectados a los

sensores táctiles, funcionarán automáticamente como entradas. Como entrada, un pin de E/S conectado a un sensor táctil, hará que su bit correspondiente del registro **ins** muestre un 1 si la tensión es 5 Volt (sensor no presionado) y 0 si la tensión es 0 Volt (sensor presionado). La Debug Terminal puede ser usada para mostrar estos valores.

- Para los siguientes programas se usara el termino Bigotes para referirse a los sensores táctiles
- Ingresar y ejecutar el Programa 4.1 que muestra la **Figura 6.4**.
- Este programa usa la Debug Terminal, así que se deberá dejar conectado el cable serial a la Plaqueta de educación mientras el Programa 4.1 se esté ejecutando.

```
'{$STAMP BS2}
'Programa 4.1: Prueba de Bigotes.
bucle:
DEBUG HOME, "P6 = ", BIN1 IN6, " P4 = ", BIN1 IN4
PAUSE 50
GOTO bucle
```

Figura 6.4 Programa 4.1

- Observar los valores mostrados en la Debug Terminal; debería verse a P6 y P4 iguales a 1.
- Mirar en la **Figura 6.1** para identificar al “sensor izquierdo” y el “sensor derecho”.
- Presionar el sensor derecho hasta que toque el conector de tres pines de la derecha y observar los valores mostrados en la Debug Terminal nuevamente, se debería leer: P6 = 1 P4 = 0.
- Presionar el sensor izquierdo hasta que toque el conector de tres pines y observar los valores mostrados en la Debug Terminal nuevamente. Esta vez se debería leer: P6 = 0 P4 = 1.
- Presionar ambos sensores contra los conectores de tres pines. Ahora se debería leer P6 = 0 P4 = 0.

Si los sensores táctiles pasaron todas las pruebas, se puede continuar; caso contrario, buscar errores en el programa y en el circuito.

Cómo funciona el programa que prueba los sensores táctiles.

Los únicos elementos nuevos en el Programa 4.1 son **in6** y **in4**. En lugar de variables tipo **word o byte**, estas son variables tipo **bit**. Estas variables tipo bit son especiales debido a que almacenan el valor de entrada de un pin de E/S en particular. La variable **in4** se refiere al bit-4 del registro **ins** mientras que **in6** se refiere al bit-6. Cuando el pin de E/S P4 del BASIC Stamp detecta 0 Volt, el valor de **in4** es 0. Cuando P4 detecta 5 Volt, el valor de **in4** es 1. La variable **in6** trabaja del mismo modo, excepto que controla a P6 en lugar de P4.

6.5 Programar el Hunter II para que explore basándose en los sensores tactiles.

En la Actividad anterior, el BASIC Stamp fue programado para detectar si algún sensor táctil había sido presionado. Ahora el BASIC Stamp será programado para usar esa información para guiar al Hunter II. Cuando el Hunter II está avanzando y un sensor es presionado, significa que se ha topado con algún obstáculo. Un programa de navegación debe recibir esta información, decidir que significa y llamar a una rutina de navegación que pueda evitar el obstáculo.

El Programa 4.2 mostrado en la **Figura 6.5** hace que el Hunter II vaya hacia adelante hasta que encuentra un obstáculo. Es este caso, el Hunter II sabe que encontró un obstáculo, al chocar con él con uno o ambos sensores. Tan pronto como los sensores táctiles detectan el obstáculo, se usan las rutinas y subrutinas de navegación desarrolladas anteriormente para que el Hunter II retroceda y gire. Luego, el Hunter II reasume su movimiento hacia adelante hasta chocar con otro obstáculo.

Cuando se presiona un sensor, debido a un obstáculo, se cierra el interruptor del tipo normalmente abierto.

Los pines P6 y P4 se configuran como entradas y se usan para monitorear el estado de los sensores táctiles. Los dos sensores pueden estar en uno de cuatro estados posibles:

1. Ambos en 1 – ningún objeto detectado.
2. Izquierdo 0, derecho 1 – objeto detectado a la izquierda.
3. Derecho 0, izquierdo 1 – objeto detectado a la derecha.
4. Ambos en 0 – indica colisión frontal contra un objeto grande (una pared por ejemplo).

El Programa 4.2 muestra un ejemplo de cómo pueden usarse los sensores para seleccionar la rutina de navegación del Hunter II apropiada. Por ejemplo, estado 1 significa que el Hunter II puede seguir adelante.
Estado 2 significa que el Hunter II debería retroceder y luego girar a la derecha. Estado 3 significa que el Hunter II debería retroceder y girar a la izquierda y estado 4 sería un buen momento para retroceder y hacer un giro en U.

- Ejecutar el Programa 4.2 y vea cómo se comporta el Hunter II cuando choca contra una pared.

```
' Programa 4.2: Exploración con Bigotes.
'----- Declaración --------------
cuenta_pulsos VAR Byte ' Contador del for...next.
'----- Inicialización ------------
OUTPUT 2 ' Fija a P2 como salida.
FREQOUT 2, 2000, 3000 ' Indicador de reset.
LOW 12 ' P12 y 13 salidas bajas.
LOW 13
'----- Rutina Principal --------------
principal:
control_bigotes: ' Controla cada bigote.
IF IN6 = 0 AND IN4 = 0 THEN giro_u ' Atrás si tocan ambos.
IF IN6 = 0 THEN giro_der ' Derecha si toca el izquierdo.
IF IN4 = 0 THEN giro_izq ' Izquierda si toca el derecho.
adelante: ' Si no detecta, un pulso adelante.
PULSOUT 12,650
PULSOUT 13,850
PAUSE 20
GOTO principal ' Controla nuevamente.
'----- Rutinas de Navegación -------
giro_izq: ' Rutina giro izquierda.
GOSUB atras ' Llama a atras: antes de girar.
FOR cuenta_pulsos = 0 TO 30
PULSOUT 12, 650
PULSOUT 13, 650
PAUSE 20
NEXT
GOTO principal
giro_der: ' Rutina giro derecha.
GOSUB atras ' Llama a atras: antes de girar.
FOR cuenta_pulsos = 0 TO 30
```

```
PULSOUT 12, 850
PULSOUT 13, 850
PAUSE 20
NEXT
GOTO principal
giro_u: ' Rutina de giro en U.
GOSUB atras ' Llama a atras: antes de girar.
FOR cuenta_pulsos = 0 TO 60
PULSOUT 12, 650
PULSOUT 13, 650
PAUSE 20
NEXT
GOTO principal
'----- Subrutina de Navegación -----
atras: ' Usada por cada rutina de
FOR cuenta_pulsos = 0 TO 60 ' navegación.
PULSOUT 12, 850
PULSOUT 13, 650
PAUSE 20
NEXT
RETURN
```

Figura 6.5 Programa 4.2.

Cómo funciona exploración con sensores táctiles.

Las instrucciones **if...then** de la rutina **principal** primero revisan el estado de los sensores. Si ambos están presionados, llama la rutina **giro_u**. Si solamente se encuentra presionado el sensor izquierdo, se activa la entrada P6 y se llama la rutina **giro_der**. Si se presiona el sensor derecho, se activa la entrada P4, llamando a la rutina **giro_izq**.

principal:
control_bigotes:
if in6 = 0 and in4 = 0 then giro_u
if in6 = 0 then giro_der
if in4 = 0 then giro_izq

Si no se presiona ninguno de los sensores, la acción por defecto es ejecutar una versión modificada de la rutina **adelante**. La rutina **adelante** ha sido modificada de forma que solamente entrega un pulso, antes de regresar a verificar el estado de los sensores.

La clave para entender como está estructurada la rutina **principal,** es que controla los sensores entre cada pulso que se envía a los servos, cuando el Hunter II se mueve hacia adelante. Observe que las instrucciones **pulsout** y **pause** en la rutina adelante no están anidadas dentro de un bucle **for...next**. Solamente se envía un único pulso a los servos. Después del pulso, el programa salta a la etiqueta **principal:** y controla los sensores táctiles nuevamente. Este proceso se produce tan rápidamente que no es necesario reducir el valor de **pause** en la rutina adelante.

```
adelante:
pulsout 12,650
pulsout 13,850
pause 20
goto principal
```

Cuando los sensores detectan un objeto, el Hunter II ya está demasiado cerca del mismo. Para evadir el objeto, el Hunter II primero debe retroceder. Esto es fácil de solucionar con una subrutina. Por ejemplo, la rutina **giro_izq** que se muestra a continuación llama a la subrutina **atras** usando el comando **gosub atras**. La ventaja de las subrutinas es que regresan el control del programa a la línea inmediatamente posterior a la llamada de la subrutina. Así, después de ejecutar **atras**, se inicia el bucle **for...next** de la rutina **giro_izq**. Otra ventaja de tener una subrutina **atras** es que regresa el control del programa al lugar correcto. Si **giro_der** llama a la subrutina, en lugar de **giro_izq,** el control del programa es regresado a la rutina **giro_der**.

```
giro_izq:
gosub atras
for cuenta_pulsos = 0 to 30
pulsout 12, 650
pulsout 13, 650
pause 20
next
goto principal
```

La subrutina **atras:** es una versión modificada de la rutina **atras:** desarrollada en el la experiencia anterior de fotorresistores.
En lugar de finalizar con el comando **goto principal**, finaliza con **return**. El comando **return** trabaja correctamente en esta situación, debido a que regresa el control del programa a cualquiera de las tres rutinas que llamaron a la subrutina.

```
atras:
for cuenta_pulsos = 0 to 60
pulsout 12, 850
pulsout 13, 650
pause 20
next
return
```

6.6 Hunter II decide cuándo está atrapado.

Tal vez se haya notado que el Hunter II tiende a quedar atrapado en las esquinas. A medida que el Hunter II se acerca a una esquina, supongamos que su sensor toca la pared de la izquierda, entonces dobla a la derecha, cuando el Hunter II se vuelve a mover hacia adelante, su sensor derecho tocará la otra pared, así que girará a la izquierda.
Luego tocará la pared izquierda y volverá a girar a la derecha, donde se encontrará con la otra pared y así sucesivamente, hasta que alguien lo rescate.

Programa para liberarse de los rincones.

El problema de los rincones puede solucionarse agregando un contador que registre las transiciones entre los estados 1 y 2 en el Programa 4.2. El Programa 4.3 que aparece en la **Figura 6.6** es una versión modificada del Programa 4.2, que detecta y cuenta la cantidad de veces que los sensores detectan un obstáculo después de otro.

```
' Programa 4.3: Escape de Rincones.
'----- Declaración -------
cuenta_pulsos VAR Byte ' Contador del bucle for...next.
estado VAR Nib ' Almacena datos de bigotes en binario.
estado_ant VAR Nib ' Guarda estado del pulso anterior.
contador VAR Nib ' Contador de secuencia de eventos.
'----- Inicialización -----
OUTPUT 2 ' Fija a P2 como salida.
FREQOUT 2, 2000, 3000 ' Indicador de reset.
LOW 12 ' P12 y 13 salidas bajas.
LOW 13
estado_ant = %0001 ' Inicializa estado_ant.
'----- Rutina Principal -------
principal:
control_bigotes:
estado.BIT1 = IN6 ' Almacena bigotes en estado.
estado.BIT0 = IN4
' Más adelante en el programa, un contador se incrementa cada vez que se pre-
' sionan bigotes alternadamente. La instrucción if...then controla que no se
' trate del mismo bigote. Si es así, reinicia la variable contador.
IF estado_ant <> estado THEN sin_reset ' Si repite mismo bigote, reset.
contador = 0 ' Pone a cero el contador.
sin_reset: ' Rótulo para que el if... salte.
' Si estado_ant o-exclusiva estado no es igual al binario-0011, salta al rótu-
' lo continuar:. De otra forma incrementa el contador y decide si es hora de
' dar un giro en U.
IF estado_ant ^ estado <> %0011 THEN continuar
contador = contador +1
estado_ant = estado ' Guarda una copia de estado.
IF contador = 4 THEN giro_u ' Rutina de escape de esquina.
continuar: ' Salto condicional según el valor de estado.
BRANCH estado, [giro_u, giro_der, giro_izq, adelante]
adelante:
PULSOUT 12,650
PULSOUT 13,850
PAUSE 20
GOTO principal
```

```
'----- Rutinas de Navegación -------
giro_izq: ' Rutina de giro a la izquierda.
GOSUB atras ' Llama a atras: antes de girar.
FOR cuenta_pulsos = 0 TO 30
PULSOUT 12, 650
PULSOUT 13, 650
PAUSE 20
NEXT
GOTO principal
giro_der: ' Rutina giro derecha.
GOSUB atras ' Llama a atras: antes de girar.
FOR cuenta_pulsos = 0 TO 30
PULSOUT 12, 850
PULSOUT 13, 850
PAUSE 20
NEXT
GOTO principal
giro_u: ' Rutina de giro en U.
GOSUB atras ' Llama a atras: antes de girar.
FOR cuenta_pulsos = 0 TO 60
PULSOUT 12, 650
PULSOUT 13, 650
PAUSE 20
NEXT
contador = 0 ' Pone a cero el contador.
GOTO principal
'----- Subrutina de Navegación -----
atras: ' Usada por cada rutina de
FOR cuenta_pulsos = 0 TO 60 ' navegación.
PULSOUT 12, 850
PULSOUT 13, 650
PAUSE 20
NEXT
RETURN
```

Figura 6.6 Programa 4.3

Cómo funciona el programa escape de rincones.

Las variables tipo nibble llamadas **estado_ant** y **contador** se agregaron a la sección de **declaraciones** para mantener un registro de los giros. La variable **estado_ant** guarda el estado previo antes de que la variable **estado** sea actualizada con los valores actuales de

los sensores. La variable **contador** registra la cantidad de transiciones de **estado** de uno a dos y viceversa.

```
declaraciones:
cuenta_pulsos var byte
estado var nib
estado_ant var nib
contador var nib
```

La rutina **control_bigotes** está diseñada para controlar los sensores como antes, pero se ha agregado código para detectar la condición “atrapado en una esquina”. Esta condición es indicada por cuatro transiciones derecha-izquierda-derecha (o izquierda-derecha-izquierda). Antes de actualizar la variable **estado** con los valores nuevos de los sensores, la instrucción **estado_ant = estado** guarda una copia del estado previo, como referencia. Luego la variable **estado** es actualizada con los valores actuales, como en el Programa 4.3.

```
control_bigotes:
estado_ant = estado
estado.bit1 = in6
estado.bit0 = in4
```

Se asume que la condición para que esté atrapado en la esquina, es que se presionen en forma alternada los sensores, en varias oportunidades. Sin embargo, si el Hunter II encuentra un obstáculo dos veces seguidas con el mismo sensor, es porque no debe estar atrapado en una esquina. Si esto sucede, el programa debería empezar a contar desde cero nuevamente. El comando **estado_ant <> estado then sin_reset** verifica si el mismo sensor fue presionado dos veces seguidas. Si es así, reinicia el contador, caso contrario, el programa salta a la etiqueta **sin_reset:.**

```
if estado_ant <> estado then sin_reset
contador = 0
sin_reset:
```

Luego, el programa controla si los sensores se han presionado en forma alternada mediante el operador excluyente (**^**). Si el resultado de **estado_ant ^estado** nuevo, no es igual al número binario 0011, significa que el sensor opuesto no fue tocado, en este caso, la instrucción **if...then** envía el programa a la etiqueta **continuar:**.

```
if estado_ant ^ estado <> %0011 then continuar
```

Por otro lado, si la comparación es falsa (si **estado_ant^estado** es igual a 0011), entonces se incrementa la variable **contador**. Otro **if...then** controla si **contador** ha llegado a 4. Si es así, se ejecuta la rutina **giro_u**. Caso contrario el programa salta a la maniobra apropiada basándose en el valor de **estado**.

```
controla_contador:
contador = contador +1
if contador = 4 then giro_u
continuar:
branch estado, [giro_u, giro_der, giro_izq, adelante]
```

Cuando el contador de contactos en sensores opuestos llega a 4, se llama la rutina **giro_u**. Esta rutina ha sido modificada para que reinicie el contador cada vez que es llamada. De esa forma, el Hunter II comienza a contar de 0 cada vez que se escapa de una esquina.

```
giro_u:
gosub atras ' Llama a atras: antes de girar.
for cuenta_pulsos = 0 to 60
pulsout 12, 650
pulsout 13, 650
pause 20
next
contador = 0
goto principal
```

6.7 Conclusión.

Con el termino de esta experiencia se ha completado un proceso de perfeccionamiento del prototipo Hunter II al incorporar la sensibilidad táctil del robot, esto permite al estudiante usar esta herramienta para dejar al Hunter II en condiciones de tomar decisiones en lo que respecta a los estimulo externos que va recogiendo durante su navegación por medio de sus sensores de tacto.

Capítulo 7 – DESARROLLO DE EXPERIENCIA DE LABORATORIO 5

Experiencia nº 5

Titulo	:	Programación de movimientos del transporte
Sala	:	……..
Profesor	:	……..

7.1 Objetivos.

El objetivo de esta experiencia es usar los componentes del Hunter II en una maqueta y combinar algunas de las experiencias realizadas anteriormente como base para poner en funcionamiento la simulación de un transporte de madera robotizado.

Nota: Para todas las Actividades de esta experiencia, necesitará una computadora personal (PC) con el sistema operativo Windows 95, 98, 2000, Me o XP.

7.2 Herramientas y componentes.

Para esta experiencia se diseñó y fabricó una maqueta a la cual se le agregaron los siguientes componentes que en su mayoría se utilizaron en experiencias anteriores en el Hunter II:

- (3) Servomotores
- (2) Fotorresistores
- (4) Resistores de 220 Ω
- (3) LEDs Verdes
- (1) Parlante piezoeléctrico
- (2) capacitores 0.01 uF
- (1) Basic stam
- (varios) cables de interconexión

7.3 Procedimiento general.

En las experiencias anteriores se programó el Hunter II para realizar rutinas de movimiento repetitivos los cuales estaban predeterminados por un programa, luego estas rutinas de movimiento fueron complementadas con la instalación de sensores que le otorgan la inteligencia artificial necesaria para que el vehículo pueda navegar y sortear obstáculos sin problemas, en síntesis a través de estos sensores el Hunter II logra un nexo entre el ambiente exterior y la lógica de sus movimiento. Para esta experiencia el Hunter II no será usado como vehículo tal, sino que sus componentes casi en su totalidad serán usados para simular la robotización de un proceso productivo. Para ello primero se construyo una maqueta, basada en la simulación de una línea de transporte de madera empaquetada. Para lograr que esta maqueta tenga funcionalidad se le instalaran los servomotores del Hunter II, los que funcionaran como actuadotes que le darán el movimiento a las cadenas de transporte de la maqueta y a la mesa de levante y además se instalarán sensores fotorresistores que cumplen la función de automatizar el proceso a través de su retroalimentación con el ambiente.

7.4 Armado del hardware o maqueta.

Para el montaje y armado de la Maqueta se debe desarmar el Hunter II y montar los componentes en la Maqueta, la **Figura 7.1** muestra la Maqueta en una vista general.

Figura 7.1 Maqueta.

La **Figura 7.2** muestra el montaje de un Fotorresistor en la Maqueta.

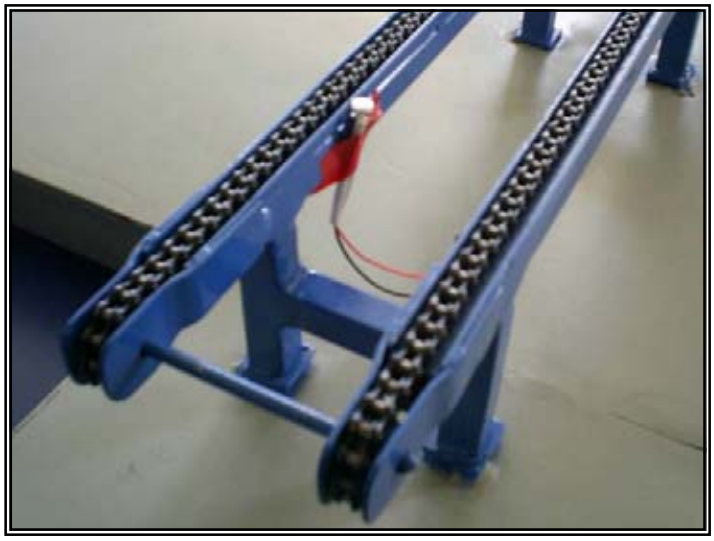


Figura 7.2 Montaje de Fotorresistor

La **Figura 7.3** muestra el montaje de un LED en la Maqueta.

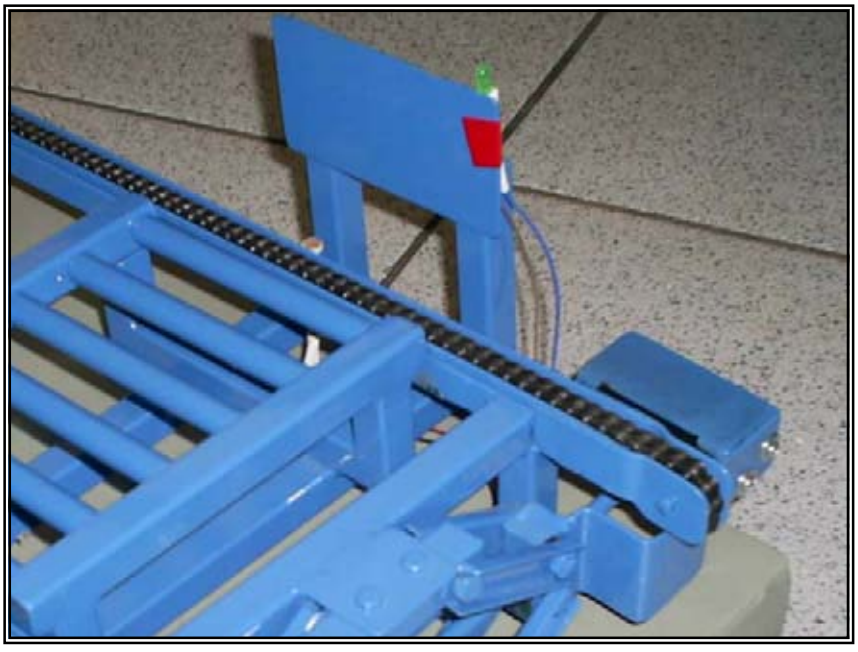


Figura 7.3 Montaje de LED

La **Figura 7.4** Muestra la Plaqueta de Educación y la BASIC Stamp montadas en la Maqueta.

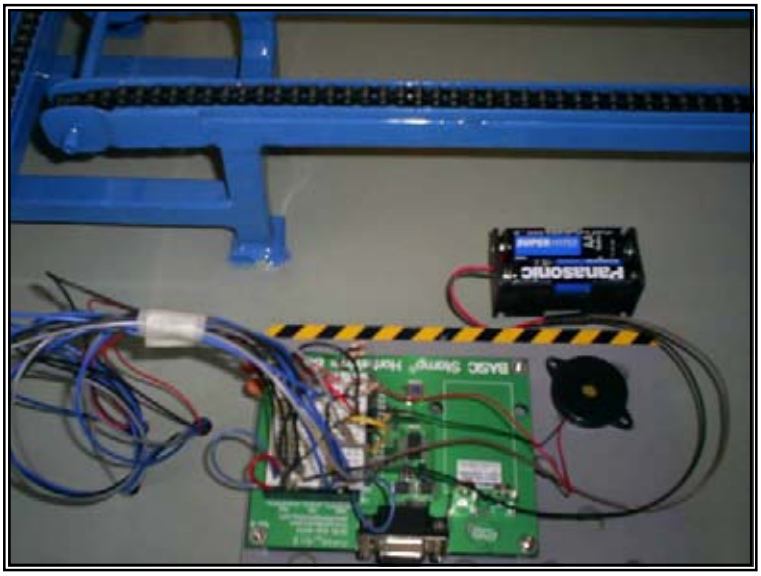


Figura 7.4 Basic Stamp

Puesta en marcha transporte de entrada

Ahora el transporte de entrada que muestra la **Figura 7.5** esta listo para programarle movimientos. En esta actividad se programará el transporte para que se mueva hacia adelante y se detenga después de avanzar la distancia necesaria para que la carga sea posada sobre la cama de rodillos.

Figura 7.5 Transporte de entrada

Hacer funcionar al transporte de entrada es tan fácil como conectar el servo a la salida P13 y enviar mediante el comando **pulsout** pulsos de 1.7 ms que harán que el servomotor conectado al transporte gire en sentido antihorario, a esto también se debe agregar un comando **cuenta_pulsos** que controla la cantidad de giros necesarios que debe ejecutar el servo para lograr un recorrido completo del transporte.

Ingresar y ejecutar el Programa 5.1 que muestra la **Figura 7.6** y controlar que la cantidad de pulsos y la velocidad es suficiente para lograr el recorrido completo del transporte de entrada que termina con la llegada de la carga a la cama de rodillos.

```
'{$STAMP BS2}
' Programa 5.1: Transporte de entrada

'----- Declaración --------------
cuenta_pulsos VAR Word ' Declara una variable para contar pulsos.

'----- Inicialización ------------
LOW 13     ' 13 salida baja.
'----- Rutina Principal --------------

principal:

FOR cuenta_pulsos = 1 TO 2500 ' Envía 2500 pulsosa servo entrada.
PULSOUT 13, 700
PAUSE 10
NEXT

STOP ' Se detiene hasta un reset.
```

Figura 7.6 Programa 5.1

Puesta en marcha de mesa de levante

La puesta en marcha de este mecanismo denominado mesa de levante que se muestra en la **Figura 7.7** en principio es similar al paso anterior, con la diferencia de que este mecanismo debe realizar dos movimientos, es decir el servo debe ser calibrado para que una vez elevada la mesa, retorne a su posición inicial para así quedar preparada para una siguiente secuencia de movimiento.

Figura 7.7 Mesa de levante

Ingresar y ejecutar el Programa 5.2 que muestra la **Figura 7.8** y controlar que la cantidad de pulsos del programa es suficiente para lograr la altura necesaria para que la carga se desprenda de la cama de rodillos para así poder ser tomado por el transporte de salida, además debe controlar el retorno a la posición inicial de la mesa para que pueda ser cargada nuevamente por el transporte de entrada.

```
'{$STAMP BS2}
' Programa 5.2:Mesa de levante

'----- Declaración --------------
cuenta_pulsos VAR Word ' Declara una variable para contar pulsos.

'----- Inicialización ------------
LOW 12     ' P12 como salida baja.
'----- Rutina Principal --------------

principal:

FOR cuenta_pulsos = 1 TO 2900 ' Levanta la mesa .
PULSOUT 12, 850 ' Pulso al servo mesa.
PAUSE 1 ' Pausa de 20 ms.
NEXT
FOR cuenta_pulsos = 1 TO 2900 ' baja la mesa de levante.
PULSOUT 12, 500 ' Pulso  al servo mesa.
PAUSE 1 ' Pausa de 20 ms.
NEXT
STOP ' Se detiene hasta un reset.
```

Figura 7.8 Programa 5.2

En esta oportunidad se usara P12 como salida baja para conectar el servo que acciona la mesa de levante.

Puesta en marcha transporte de salida

El transporte de salida que muestra la **Figura 7.9** se programa de la misma forma que el transporte de entrada, pero esta vez se usara P14 como salida baja para conectar el servo que acciona el transporte de salida.

Figura 7.9 Transporte de salida

Ingresar y ejecutar el Programa 5.3 que muestra la **Figura 7.10** y controlar que la cantidad de pulsos del programa es suficiente para lograr el recorrido completo del transporte de salida.

```
'{$STAMP BS2}
' Programa 5.3: Transporte de salida

'----- Declaración --------------
cuenta_pulsos VAR Word ' Declara una variable para contar pulsos.

'----- Inicialización ------------
LOW 14     ' 14 salida baja.
'----- Rutina Principal --------------

principal:

FOR cuenta_pulsos = 1 TO 2200 ' Envía 2200 pulsos a servo de salida.
PULSOUT 14, 850
PAUSE 10
NEXT

STOP ' Se detiene hasta un reset.
```

Figura 7.10 Programa 5.3

7.5 Puesta en marcha de maqueta

La puesta en marcha de esta maqueta consiste en lograr la combinación y armonía de los movimientos anteriores, de manera que un solo programa pueda ser capaz de desarrollar un ciclo completo, el cual comprende desde la entrada de una carga pasando por la cama de rodillos y la mesa de levante hasta lograr que llegue a la salida de la línea de producción.

La **Figura 7.11** muestra el programa 5.4 donde están combinados los movimientos de las secuencias de cada parte del transporte. Este programa realiza una secuencia completa de movimientos, el cual para ser ejecutado nuevamente debe hacerse con un reset en la BASIC Stamp.

```
'{$STAMP BS2}
' Programa 5.4: Movimiento del transporte

'----- Declaración --------------
cuenta_pulsos VAR Word ' Declara una variable para contar pulsos.

'----- Inicialización ------------
LOW 12     ' 12 salida baja.
LOW 13     ' 13 salida baja.
LOW 14     ' 14 salida baja.

'----- Rutina Principal --------------

principal:
FOR cuenta_pulsos = 1 TO 2500 ' Envía 2500 pulsosa servo entrada.
PULSOUT 13, 700
PAUSE 10
NEXT
GOTO principal ' regresa a la rutina principal.

FOR cuenta_pulsos = 1 TO 2900 ' Levanta la mesa .
PULSOUT 12, 850 ' Pulso al servo mesa.
PAUSE 1 ' Pausa de 20 ms.
NEXT

FOR cuenta_pulsos = 1 TO 2200 ' expulsa la carga del transporte.
PULSOUT 14, 850 ' Pulso al servo salida.
PAUSE 1 ' Pausa de 20 ms.
NEXT

FOR cuenta_pulsos = 1 TO 2900 ' baja la mesa de levante.
PULSOUT 12, 500 ' Pulso  al servo mesa.
PAUSE 1 ' Pausa de 20 ms.
NEXT
STOP ' Se detiene hasta un reset.
```

Figura 7.10 Programa 5.4

7.6 Instalación y programación de fotorresistores

En esta parte de la experiencia se usaran los dos fotorresistores instalados en la Maqueta, uno ubicado en el transporte de entrada y el otro se instalara después de la cama de rodillos y fijar los Pines de E/S P4 y P5 como salidas de estado alto. En la **Figura 7.11** se muestra el Programa 5.5 que es similar al Programa 5.4, con la diferencia que este programa tiene incluido los fotorresistores.

```
'{$STAMP BS2}
' Programa 5.5: Movimiento completo del transporte

'----- Declaración --------------
foto_entrada VAR Word ' Almacenan los tiempos RC de los
foto_salida VAR Word  ' fotorresitores entrada y salida.
cuenta_pulsos VAR Word ' Declara una variable para contar pulsos.

'----- Inicialización ------------
LOW 13     ' 13 salida baja.
LOW 14     ' 14 salida baja.
'----- Rutina Principal --------------
principal:
' Mide el tiempo RC del fotorresitor entrada.
HIGH 4 ' Fija a P5 como salida alta.
PAUSE 3 ' Pausa de 3 ms.
RCTIME 4,1,foto_entrada ' Mide tiempo RC en P4.
' Mide el tiempo RC del fotorresitor salida.
HIGH 5 ' Fija a P4 como salida alta.
PAUSE 3 ' Pausa de 3 ms.
RCTIME 5,1,foto_salida ' Mide tiempo RC en P3.
'Toma la diferencia entre los fotoresistores.
IF ABS(foto_entrada-foto_salida) < 99 THEN principal
IF foto_entrada > 100 THEN pulso_entrada
IF foto_salida > 100 THEN pulso_salida
'----- Rutinas de Navegación -------
pulso_entrada:     ' Aplica un pulso de entrada.
FOR cuenta_pulsos = 1 TO 2500 ' Envía 2500 pulsos.
PULSOUT 13, 700
PAUSE 10
NEXT
```

```
GOTO principal ' regresa a la rutina principal.
pulso_salida: ' Aplica un pulso de salida.
FOR cuenta_pulsos = 1 TO 2900 ' Levanta la mesa de levante.
PULSOUT 12, 850 ' Pulso al servo mesa.
PAUSE 1 ' Pausa de 20 ms.
NEXT
FOR cuenta_pulsos = 1 TO 2200 ' expulsa la carga del transporte.
PULSOUT 14, 850 ' Pulso al servo salida.
PAUSE 1 ' Pausa de 20 ms.
NEXT
FOR cuenta_pulsos = 1 TO 2900 ' baja la mesa de levante.
PULSOUT 12, 500 ' Pulso  al servo mesa.
PAUSE 1 ' Pausa de 20 ms.
NEXT
GOTO principal ' regresa a la rutina principal.
```

Figura 7.11 Programa 5.5

Al Igual que en la experiencia de navegación con fotorresistores el Programa 5.5 mide los tiempos RC y controla si la diferencia entre los valores obtenidos por los comandos **rctime** están dentro del intervalo de tolerancia en este caso no toma ninguna acción.

En caso que la diferencia esté dentro del intervalo, el programa salta a la etiqueta **principal,** de lo contrario si la medida supera el valor determinado en el programa, las instrucciones **if...then** deciden a qué rutina llamar, **pulso_entrada** o **pulso_salida**.

if **foto_entrada** > 100 THEN **pulso_entrada**
if **foto_salida** > 100 THEN **pulso_salida**

Cuando la instrucción **if…then** llama a la rutina **pulso_entrada** se acciona el servo de entrada y realiza el movimiento definido en el Programa 5.1, esto sucede cuando la carga de material puesta en el transporte de entrada interfiere en la fuente de luz del fotorresistor de entrada, de esta forma el valor de la resistencia del fotorresistor aumenta hasta el punto en que el programa llama a comenzar la rutina del transporte de entrada.

Al llegar la carga al final del recorrido que realiza el transporte de entrada, se posa sobre la cama de rodillos donde esta ubicado otro fotorresistor el cual aumenta su resistencia al punto tal que la instrucción **if…then** llama a la rutina **pulso_salida,** que ejecuta una

combinación de los Programas 5.2 y 5.3 que hace subir la mesa de levante , para luego comenzar con la rutina de salida la cual desliza la carga hasta el final del transporte de salida, luego la mesa de levante baja y vuelve a su punto de inicio para esperar un nuevo pulso de salida cuando la carga halla interrumpido nuevamente la fuente de luz del fotorresistor de la cama de rodillos.

7.7 Conclusión.

En esta experiencia se logro integrar algunos conocimientos adquiridos anteriormente con la finalidad de poner en funcionamiento una maqueta, que en este caso simula un sistema de transporte robotizado, con esto el estudiante queda con la capacidad de usar los conocimientos adquiridos para usarlos en un sin fin de aplicaciones donde se requiera automatizar algún sistema ya sea en una simple simulación como también en sistemas muchos mas complejos que se puedan presentar en la realidad.

CONCLUSIONES

Las experiencias realizadas en este proyecto de laboratorio con el prototipo Hunter II demuestra ser una solución a la falta de conocimientos en lo concerniente a investigaciones prácticas en temas de tecnología actual como lo es la robótica, problema que nos impide poder abordar tecnologías que hoy en día son indispensables en el desempeño laboral de los ingenieros en la industria moderna, como así en la interacción con otros profesionales en el campo laboral.

Este proyecto ha cumplido con el objetivo de desarrollar experiencias de laboratorio, para los alumnos de la carrera de Ingeniería Mecánica, las cuales le permitirán a los estudiantes desarrollar sus propias experiencia través del desarrollo de guías, ya sea programando un vehículo explorador robotizado que interactúe con el ambiente, o simulando en maquetas algún proceso productivo.

A través de las diversas experiencias se fue cumpliendo con los objetivos planteados en cada una de estas a fin de validar la hipótesis planteada. Tomando en cuenta la importancia que a adquirido la automatización en la industria, este proyecto se transforma en una herramienta de formación transversal de ingenieros que convergen en el camino hacia el avance de la ciencia y tecnología, abriendo un sendero al estudio y desarrollo de soluciones que requieren el uso de nuevas tecnologías en lo concerniente a la robótica.

REFERENCIAS

Referencias Bibliográficas

- **ASIMOV ISAAC.,** “I Robot” (Yo Robot). Principios éticos para robots y máquinas inteligentes, Primera edición 1950. Editorial Sudamericana.

- **Angulo J.M y Avilés R.,** “Introducción a la Inteligencia Artificial”,2001. Editorial Paraninfo S.A.

- **Mónica María Sánchez.,** “Implementación de Robótica Pedagógica”. Tesis de Maestría en Ingeniería Eléctrica, Universidad de los Andes, 2003.

Referencias Electrónicas:

- www.rambal.com/descarga/doc/BsManSpaninish.pdf "Manual Basic Stamp 2 versión 1.9". (Archivo descargado marzo 2003).
- www.superrobotica.com (Archivo descargado marzo 2003)
- www.roboticajoven.mendoza.edu.arg (Consultado marzo 2003).
- www.enlaces.cl (Consultado marzo 2003).

ANEXOS.

ANEXO 1

BRANCH

BRANCH indicador, [dirección0, dirección1, …direcciónN]

Se dirige a la dirección especificada por el indicador (si está en el rango).

- **indicador** es una variable / constante que especifica a cuál de las direcciones listadas dirigirse (0-N).
- **direcciones** son las etiquetas que especifican a qué lugar dirigirse.

BUTTON

BUTTON pin, presionado, retardo, velocidad, espaciotrabajo, estado, etiqueta

Elimina el rebote, realiza auto-repetir y se dirige a la etiqueta si un botón es activado.

- **pin** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..15.
- **presionado** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..1.
- **retardo** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..255.
- **velocidad** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..255.
- **espaciotrabajo** es una variable tipo byte o word. Se inicia con 0 antes de ser usada.
- **estado** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..1.
- **etiqueta** es la etiqueta a la que debe saltar el programa cuando se presiona el botón.

COUNT

COUNT pin, período, resultado

Cuenta el número de ciclos en un pin, por un período de milisegundos y guarda ese número en resultado.

- **período** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..65535.
- **resultado** es una variable tipo bit, nibble, byte o word.

DATA

{ puntero} DATA {@ ubicación,} { WORD} { datos}{(tamaño)} {, { WORD} { datos}{(tamaño)}...}

Almacena datos en la EEPROM antes de descargar el programa PBASIC

- **puntero** es un nombre opcional de una constante no definida o variable tipo bit, nibble, byte o word al que se le asigna el valor del primer lugar de memoria en el que los datos son almacenados.
- **ubicación** es una constante, expresión o variable tipo bit, nibble, byte o word opcional que designa el primer lugar de memoria en el que los datos son escritos.
- **word** es una llave opcional que hace que los datos sean almacenados en la memoria como dos bytes separados.
- **datos** es una constante o expresión opcional a ser escrita en la memoria.
- **tamaño** es una constante o expresión opcional que designa el número de bytes de datos definidos o no, para escribir o reservar en la memoria. Si datos no es especificado, entonces se reserva una cantidad de espacio indefinido y si datos es especificado, se reserva la cantidad especificada por tamaño.

DEBUG

DEBUG datosalida{, datosalida...}

Envía variables a ser vistas en la PC.

- **datosalida** es una cadena de texto, constante o variable tipo bit, nibble, byte o word. Si no se especifican modificadores DEBUG muestra caracteres en ascii sin espacios ni separación de oraciones a continuación del valor.

DTMFOUT

DTMFOUT pin, { tiempoencendido, tiempoapagado,}[tono{, tono...}]

Genera pulsos telefónicos DTMF.

- **tiempoencendido** y **tiempoapagado** son constantes, expresiones o variables tipo bit, nibble, byte o word en el rango de 0..65535.
- **tono** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..15.

END

END

- Duerme hasta que se interrumpa y reinicie la alimentación o se conecte a la PC. El consumo de energía es reducido a 50 uA.

FOR...NEXT

FOR variable = inicial to final { paso} …NEXT

Crea un bucle repetitivo que ejecuta las líneas de programa entre FOR y NEXT, incrementando o disminuyendo el valor de la variable de acuerdo al paso, hasta que el valor de la variable iguala al valor final.

- **variable** es una variable tipo nib, byte, o word usada como contador.
- **inicial** es una variable o una constante que especifica el valor inicial de la variable.
- **final** es una variable o una constante que especifica el valor final de la variable. Cuando el valor de la variable supera el valor final, end, el bucle FOR . . . NEXT detiene su ejecución y el programa continúa en la instrucción siguiente a NEXT.
- **paso** es una variable o constante opcional que fija el valor en que aumenta o disminuye la variable en cada bucle de FOR / NEXT. Si inicial es mayor que final, PBASIC2 interpreta que paso es negativo, aunque no se use un signo menos.

FREQOUT

FREQOUT pin, milisegundos, freq1 {, freq2}

Genera una o dos ondas sinusoidales de las frecuencias especificadas (0 – 32.767 hz.).

- milisegundos es una constante, expresión o variable tipo bit, nibble, byte o word.
- **freq1** y **freq2** son constantes, expresiones o variables tipo bit, nibble, byte o word en el rango de 0..32767 que representan las frecuencias correspondientes.

GOSUB

GOSUB direcciónEtiqueta

Guarda la dirección de la instrucción siguiente a GOSUB, luego se dirige al punto del programa especificado por direcciónEtiqueta.

- **direcciónEtiqueta** es una etiqueta que especifica a qué lugar dirigirse.

GOTO

GOTO direcciónEtiqueta

Se dirige al punto del programa especificado por direcciónEtiqueta.

- **direcciónEtiqueta** es una etiqueta que especifica a qué lugar dirigirse.

HIGH

HIGH pin

Convierte al pin en salida y la pone en estado alto (escribe un 1 en el bit correspondiente de DIRS y OUTS).

- **pin** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..15.

IF...THEN

IF condición THEN direcciónEtiqueta

Evalúa la condición y, si es verdadera, se dirige al punto del programa marcado por direcciónEtiqueta.

- **condición** es una expresión, tal como “x=7”, que puede ser evaluada como verdadera o falsa.
- **direcciónEtiqueta** es una etiqueta que especifica dónde ir en el caso que la condición sea verdadera.

INPUT

INPUT pin

Convierte al pin especificado en entrada (escribe un 0 en el bit correspondiente de DIRS).

- **pin** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..15.

LOOKDOWN

LOOKDOWN valor, { comparador,} [valor0, valor1,... valorN], variable

Compara un valor con los de la lista en función del comparador y guarda la ubicación (índice), en la variable.

- **valor** es una constante, expresión o variable tipo bit, nibble, byte o word.
- **comparador** es =, <>, >, <, <=, =>. (= es la opción por defecto).
- Valor0, valor1, etc. son constantes, expresiones o variables tipo bit, nibble, byte o word.
- **variable** es una variable tipo bit, nibble, byte o word.

LOOKUP

LOOKUP índice, [valor0, valor1,... valorN], variable

Busca el valor especificado por el índice y lo guarda en la variable. Si el índice excede el máximo valor de índice de la lista, la variable no es afectada. Un máximo de 256 valores puede ser incluido en la lista.

- **Índice** es una constante, expresión o variable tipo bit, nibble, byte o word.
- **Valor0**, **valor1**, etc. son constantes, expresiones o variables tipo bit, nibble, byte o word.
- **Variable** es una variable tipo bit, nibble, byte o word.

LOW

LOW pin

Convierte al pin en salida y la pone en estado bajo (escribe un 1 en el bit correspondiente de DIRS y un 0 en el bit correspondiente de OUTS).

- **pin** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..15.

NAP

NAP período

Descansa por un corto período. El consumo de energía es reducido a 50 uA (sin cargas conectadas).

- **período** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..7 que representa intervalos de 18ms.

OUTPUT

OUTPUT pin

Convierte al pin especificado en salida (escribe un 1 en el bit correspondiente de DIRS).

- **pin** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..15.

PAUSE

PAUSE milisegundos

Hace una pausa en la ejecución por 0–65535 milisegundos.

- **milisegundos** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..65535.

PULSIN

PULSIN pin, estado, variable

Mide un pulso de entrada (resolución de 2 µs).

- **pin** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..15.
- **estado** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..1.
- **variable** es una variable tipo bit, nibble, byte o word.

Las mediciones se toman en intervalos de 2uS y el tiempo máximo es de 0.13107 segundos.

PULSOUT

PULSOUT pin, período

Genera un pulso de salida (resolución de 2 µs).

- **pin** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..15.
- **período** es una constante, expresión o variable tipo bit, nibble, byte o word, en el rango de 0..65535 que especifica la duración del pulso en unidades de 2uS.

PWM

PWM pin, duty, ciclos

Puede ser usado para generar voltajes de salida analógicos (0-5V) usando un capacitor y un resistor. Genera una salida por modulación de ancho de pulso y luego el pin vuelve a ser entrada.

duty es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..255.

ciclos es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..255 que representa la cantidad de ciclos de 1ms a enviar a la salida.

RANDOM

RANDOM variable

Genera un número pseudo-aleatorio.

Variable es una variable tipo byte o word en el rango de 0..65535.

RCTIME

RCTIME pin, estado, variable

Mide un tiempo de carga / descarga RC. Puede ser usado para medir potenciómetros.

pin es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..15.

estado es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..1.

variable es una variable tipo bit, nibble, byte o word.

READ

READ ubicación, variable

Lee un byte de la EEPROM y lo almacena en variable.

ubicación es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..2047.

variable es una variable tipo bit, nibble, byte o word.

RETURN

Regresa de una subrutina. Regresa el programa a la dirección (instrucción) inmediatamente siguiente al
GOSUB más reciente.

REVERSE

REVERSE pin

Si el pin es de salida, lo hace de entrada. Si el pin es de entrada, lo hace de salida.

pin es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..15.

SERIN

SERIN rpin{\ fpin}, baudmodo, { petiqueta,} { tiempoespera, tetiqueta,} [datosentrada]

Recibe datos asincrónicamente (como en RS-232).

rpin es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..16.

fpin es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..15.

baudmodo es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..65535.

petiqueta es una etiqueta a la cual saltar en caso de error de paridad.

tiempoespera es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..65535 que representa el número de milisegundos a esperar por un mensaje entrante.

tetiqueta es una etiqueta a la cual saltar en caso de demora.

Datosentrada es un conjunto de nombres de variables, expresiones o constantes, separados por comas que pueden tener los formatos disponibles para el comando DEBUG, excepto los formatos ASC y REP.

SEROUT

SEROUT tpin{\ fpin}, baudmodo, { pausa,} { tiempoespera, tetiqueta,} [datossalida]

Envía datos asincrónicamente (como en RS-232).

tpin es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..16.

fpin es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..15.

baudmodo es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..60657.

pausa es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..65535 que especifica un tiempo (en milisegundos) de retardo entre los bytes transmitidos. Este valor sólo puede ser especificado si no se da un valor a fpin.

tiempoespera es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..65535 representa el número de milisegundos a esperar para la señal de transmisión del mensaje. Este valor sólo puede ser especificado si se da un valor a fpin.

tetiqueta es una etiqueta a la cual saltar si se sobrepasa el tiempoespera. Se especifica con fpin.

datossalida es un conjunto de constantes, expresiones y nombres de variables separados por comas y opcionalmente precedido por modificadores disponibles en el comando DEBUG.

SHIFTIN

SHIFTIN dpin, cpin, modo, [resultado{\ bits} { , resultado{\ bits}... }]

Convierte los bits recibidos de serie a paralelo y los almacena.

dpin es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..15 que especifica el pin de datos.

cpin es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..15 que especifica el pin del reloj (clock).

modo es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..4 que especifica el modo de transmisión. 0 o MSBPRE = msb primero, pre-reloj, 1 o LSBPRE = lsb primero, prereloj, 2 o MSBPOST = msb primero, post-reloj, 3 o LSBPOST = lsb primero, post-reloj.

resultado es una variable tipo bit, nibble, byte o word donde el dato recibido es guardado.

bits es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 1..16 que especifica el número de bits a recibir en resultado. El valor predeterminado es 8.

SHIFTOUT

SHIFTOUT dpin, cpin, modo, [datos{\ bits} {, datos{\ bits}... }]

Envía los datos en forma serial.

- **dpin** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..15 que especifica el pin de datos.
- **cpin** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..15 que especifica el pin del reloj (clock).
- **Modo** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..1 que especifica el modo de transmisión, 0 o LSBFIRST = lsb primero, 1 o MSBFIRST = msb primero.
- **datos** es una constante, expresión o variable tipo bit, nibble, byte o word que contiene el dato a ser enviado.
- **bits** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 1..16 que especifica el número de bits a enviar. El valor predeterminado es 8.

SLEEP

SLEEP segundos

Duerme por 1-65535 segundos. El consumo de energía es reducido a 50 uA.

- **segundos** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..65535 que especifica el número de segundos a dormir.

TOGGLE

TOGGLE pin

Invierte el estado de un pin.

- **pin** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..15.

WRITE

WRITE ubicación, datos

Escribe un byte en la EEPROM.

- **ubicación** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..2047.
- **datos** es una constante, expresión o variable tipo bit, nibble, byte o word.

XOUT

XOUT mpin, zpin, [casa\ clavecomando{\ ciclos} {, casa\ clavecomando{\ ciclos}... }]

Genera códigos de control de línea X-10. Se usa con los módulos de interfase TW523 o TW513.

- **mpin** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..15 que especifica el pin de modulación.
- **zpin** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..15 que especifica el pin de cruce por cero.
- **casa** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..15 que especifica el código de casa A..P.
- **clavecomando** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 0..15 que especifica las claves 1..16 o es uno de los comandos mostrados en BASIC Stamp Manual Version 1.9. Estos comandos incluyen encender y apagar luces.
- **ciclos** es una constante, expresión o variable tipo bit, nibble, byte o word en el rango de 2..65535 que especifica el número de veces que se transmite un comando. (Predeterminado es 2).