UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
FACULTAD DE INGENIERÍA

DISEÑO Y CONSTRUCCIÓN DE UN ROBOT HEXÁPODO

TESIS

QUE PARA OBTENER EL TITULO DE:
INGENIERO ELÉCTRICO-ELECTRÓNICO

PRESENTA:
JOSÉ RAMÓN ROLDÁN RUIZ

DIRECTOR DE TESIS:
MTRO. EDUARDO E. RAMÍREZ SÁNCHEZ

MÉXICO D.F **OCTUBRE DE 2012**

Agradecimientos

A mis padres Martha y Hugo, por su amor, cariño, apoyo y comprensión, por siempre estar ahí y siempre confiar en mí. La culminación de esta meta es suya.

A mis hermanos Delia y Ernesto por ser parte importante de este logro y acompañarme en este largo caminar.

A la Universidad Nacional Autónoma de México y a la Facultad de Ingeniería por haberme abierto sus puertas y haberme dado la oportunidad de aprender.

A Texas Instruments y a la DGAPA proyecto PAPIIT 1N118509, por financiar de manera parcial el desarrollo de este proyecto.

CONTENIDO:

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

MARCO REFERENCIAL

INTRODUCCIÓN

Desde hace siglos, la robótica ha tenido un desarrollo muy importante hasta nuestros días abarcando desde robots móviles con sistemas muy simples hasta robots industriales con un mayor grado de complejidad. Hoy en día la tecnología permite crear dispositivos con gran grado de complejidad, permitiendo a los robots realizar actividades que serian imposibles realizar por un ser humano.
Debido al notable avance tecnológico de la robótica a nivel mundial, surge la inquietud de crear un sistema móvil capaz de controlarse de manera inalámbrica.

El diseño e implementación del sistema está basado en un robot tipo hexápodo similar a una araña, el cuerpo y las articulaciones se encuentran construidos de madera comprimida (MDF), cada extremidad está compuesta por 3 servomotores dando un total de 18 servomotores en todo el sistema.
El mando inalámbrico consta de 5 botones táctiles (*touch*) que controlarán las diferentes posiciones del robot: camina adelante, camina atrás, gira a la derecha, gira a la izquierda y cuatro posiciones diferentes de parado.

El sistema funciona a partir de un mando inalámbrico táctil, compuesto por un microcontrolador MSP430G2553, un módulo *touch* y un transceptor (transmisor) CC110L de Texas Instruments.
Por otra parte el robot se encuentra compuesto por un microcontrolador MSP430G2553 y un transceptor (receptor) CC110L de Texas Instruments, una tarjeta controladora de servomotores marca Pololu y una tarjeta convertidora de niveles lógicos marca Sparkfun.

El sistema de control y la programación del dispositivo están diseñados para utilizar un microcontrolador MSP430G2553 de Texas Instruments que es el encargado de recibir la posición de cada servomotor y enviarla de manera serial a la controladora de servos. Por otra parte el mando inalámbrico se encuentra diseñado con un microcontrolador que es el encargado de transmitir la posición del robot al receptor.

OBJETIVOS

Objetivo general

Diseño y construcción de un robot hexápodo (hardware y software) basado en el microcontrolador MSP430 de Texas Instruments, controlado de manera inalámbrica por medio de una interfaz táctil.

Objetivos específicos

1. Diseñar e implementar la estructura de un robot del tipo hexápodo.

2. Diseñar e implementar el sistema electrónico que será capaz de controlar al robot.

3. Diseñar e implementar el control a distancia del robot a partir de los módulos C*apacitive Touch BoosterPack* y CC110L *Anaren Air BoosterPack*.

4. Diseñar y Programar el sistema de control del robot.

5. Explotar las capacidades de hardware y software de las tarjetas y módulos de Texas Instruments: *Launchpad*, C*apacitive Touch BoosterPack* y CC110L *Anaren Air BoosterPack.*

JUSTIFICACIÓN

El proyecto que aquí se plantea, nace con la intención de diseñar un dispositivo lo más parecido a un robot hexápodo comercial, que sea capaz de controlarse de manera inalámbrica y que sea programado a partir del microcontrolador MSP430 de Texas Instruments.
Desde el punto de vista teórico, un dispositivo con estas características presenta un nivel elevado de programación, en contraparte un microcontrolador de la familia MSP430G se encuentra reducido en sus características y, realizar un sistema de esta índole hace más interesante el desarrollo del proyecto.

ALCANCE

El alcance del proyecto, es poder implementar un robot que pueda ser controlado de manera inalámbrica, además que sea capaz de funcionar adecuadamente con los algoritmos de caminar, girar y pararse.

CAPÍTULO 1

FUNDAMENTOS BÁSICOS

1.1 CINEMÁTICA Y DINÁMICA DE UN HEXÁPODO

El análisis cinemático y dinámico es de gran utilidad en este tipo de trabajos ya que ayuda a determinar el tipo de locomoción que se utiliza, así como planear trayectorias de locomoción que hagan más estable al dispositivo. Este análisis también nos ayuda a determinar la cantidad de patas que sostendrán el peso del robot en sus diferentes posiciones de locomoción.

Existen en Internet diferentes artículos con análisis bien desarrollados y bien estudiados acerca de la cinemática y dinámica de los robots hexápodos, en este trabajo solo mencionaremos los resultados de un artículo en especial el cual explica de manera clara y concisa el análisis cinemático y dinámico de un robot hexápodo

El articulo con el que trabajamos fue "Método de diagrama de cuerpo libre. Modelado cinemático y dinámico de un robot de seis patas". En este se analiza el robot como un cuerpo rígido aislado. Toma en cuenta las patas con respecto del cuerpo central en dos dimensiones, primero toma en cuenta la vista lateral desde la cual aprecia los eslabones de cada pata con respecto el cuerpo del hexápodo y después una vista superior analizando los ángulos comprendidos entre las patas, además de las fuerzas ejercidas por la fricción del suelo en casos diferentes.

1.1.1 Cinemática de un Robot Hexápodo

Los vehículos móviles que funcionan a base de patas, pueden caminar en superficies ásperas e irregulares con un alto grado de suavidad. En el articulo se desarrollan los modelos cinemáticos y dinámicos. Las ecuaciones cinemáticas se derivan con el método de Denavit-Hartenberg. Por otro lado se utilizó el método de diagrama del cuerpo libre, basado en las ecuaciones dinámicas de cuerpos rígidos para modelar dinámicamente. [1]

La locomoción de las patas en terreno natural, presenta varios problemas complejos (colocación del pie, evitación de obstáculos, distribución de carga, estabilidad general del vehículo, etc.) que se deben considerar en la construcción mecánica de vehículos y en el desarrollo de las estrategias del control.

El método de Denavit-Hartenberg se utiliza en derivar un modelo cinemático 3D de características de seis patas.

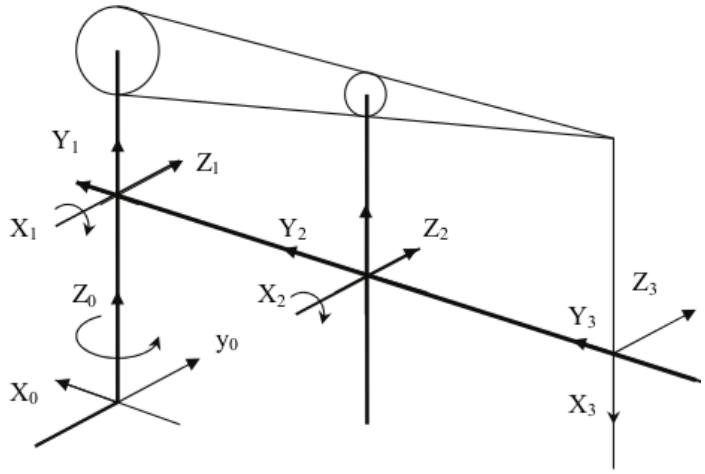

Figura 1.1 Parametrización de una pata de un robot hexápodo [1]

Partiendo del análisis de parametrización de Denavit Hartenberg se obtienen las matrices para cada una de las articulaciones de la pata del hexápodo. [1]

$$ {}^{0}A_1 = \begin{bmatrix} \cos\theta_1 & 0 & sen\theta_1 & 0 \\ sen\theta_1 & 0 & -\cos\theta_1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} {}^{1}A_2 = \begin{bmatrix} \cos\theta_2 & -sen\theta_2 & 0 & -a\cos\theta_2 \\ sen\theta_2 & \cos\theta_2 & 0 & -asen\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} {}^{2}A_3 = \begin{bmatrix} \cos\theta_3 & -sen\theta_3 & 0 & -a\cos\theta_3 \\ sen\theta_3 & \cos\theta_3 & 0 & -asen\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

1.2 SERVOMOTORES

Un servomotor es un motor eléctrico que tiene la capacidad de ser controlado, tanto en velocidad como en posición. Se encuentra constituido por un reductor de velocidad, un multiplicador de fuerza y un circuito de control que controla el grado de giro del eje. [2]
Dependiendo del servomotor este puede ser alimentado de 4.8 volts a 7 volts, en la parte externa del servomotor se encuentran tres cables, uno de alimentación (Rojo), uno de tierra (Negro) y un tercero para controlar la posición del servo (blanco, amarillo o naranja).

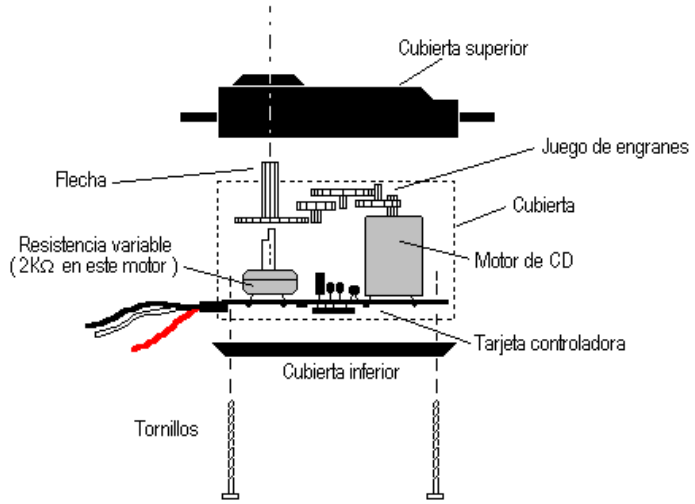

Figura 1.2 Parte interna de un servomotor

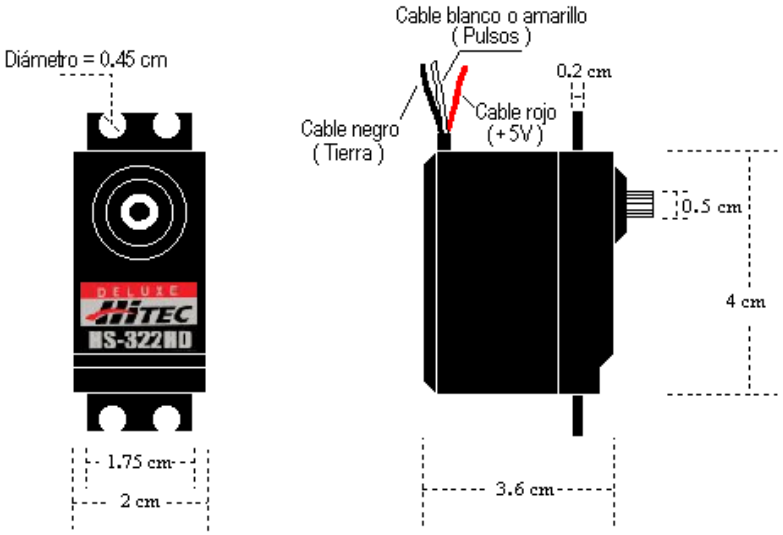

Figura 1.3 Vista frontal y lateral de un servomotor

1.2.1 Funcionamiento del servomotor

El eje del servomotor es desplazado al ángulo deseado, enviando señales de modulación por ancho de pulso (PWM) al circuito de control del servomotor.

El ancho de pulso de la señal indica el ángulo de posición: una señal con pulsos más anchos ubicará al servomotor en un ángulo mayor, en contraste un servomotor con un menor ángulo ubicará al servomotor en un ángulo menor.

Los tiempos más generales de la duración de pulso son 1 milisegundo a 2 milisegundos que corresponden a la posición de los extremos del servomotor (0° y 180°).

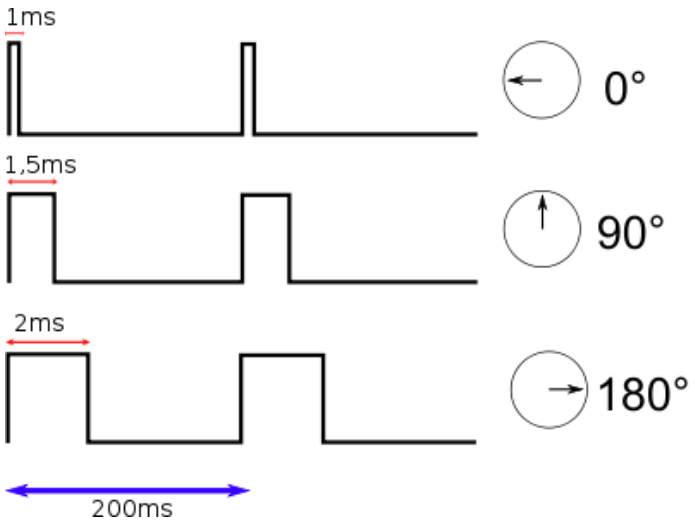

Figura 1.4 Generación de pulsos para servomotor

Es importante mencionar que, para que un servomotor se mantenga en una posición durante cierto tiempo, es necesario enviarle de manera constante el pulso correspondiente, ya que si se le dejará de enviar el pulso al servomotor, este dejaría de mantener su posición de manera que cualquier fuerza podría desplazarlo.

1.2.2 Servomotores Hitec HS-322HD

Para este trabajo se utilizaron 18 servomotores marca Hitec modelo HS-322HD, siendo una de sus ventajas su precio bajo pero teniendo como desventaja el torque tan reducido con el que cuenta.

Figura 1.5 servomotor Hitec HS-322

Características [10]:

- Voltaje de operación 4.8 volts a 6 volts.
- Engranaje de Karbonite.
- Rotación 180°.
- Máxima velocidad 316°/sec.
- Consumo de corriente 160mA sin carga.
- Corriente con carga máxima 430 mA.
- Par 3.0 kg.cm.
- Tamaño 40 x 20 x 37mm.
- Peso 43g.

1.3 TARJETA CONTROLADORA DE SERVOS

Una tarjeta controladora de servos o servocontroladora, es una tarjeta que tiene la capacidad de soportar varios servomotores, controlados al mismo tiempo ya que tiene la capacidad de generar diferentes modulaciones de PWM y transmitirlos por diferentes canales.

Cada servocontroladora cuenta con un microcontrolador propio, que se encarga de generar las señales de PWM necesarias, además de contar con un bus de comunicación que se encarga de comunicarse con otros dispositivos por medio de una línea serial (I2C, RS232, UART, etc.)

En el mercado existen un gran número de controladoras de servomotores, cada uno con características diferentes como son numero de servos soportados, tipo de señales empleadas, bus de comunicaciones usado, etcétera.

1.3.1 Tarjeta controladora Pololu mini maestro 18 CH

Para este trabajo se utilizó una controladora de servos marca Pololu modelo mini maestro de 18 canales.

Figura 1.6 Controladora de servomotores Pololu 18 ch

Características [11]:

- Tres métodos de control: USB, TTL (5 volts) y con scripts internos.
- 0.2µs de resolución en la salida de ancho de pulso.
- Configuración de pulso alto y ancho del pulso.
- Control de velocidad y aceleración individual para cada canal.
- Salidas digitales de propósito general (0 volts o 5 volts).

1.4 MICROCONTROLADORES

1.4.1 Familia MSP430

El MSP430 es una familia de microcontroladores producidos por Texas Instruments. Construido con una CPU de 16 bits, el MSP430 está diseñado para aplicaciones empotradas de bajo costo y bajo consumo de energía. El MSP430 es muy útil para aplicaciones inalámbricas y para aplicaciones de bajo consumo [15].

Este dispositivo tiene una variedad de configuraciones con los siguientes periféricos: oscilador interno, timer incluyendo un PWM, temporizador watchdog, USART (Universal Síncrono Asíncrono Receptor Transmisor) , bus SPI (Interfase Periférica Serial), bus I²C, 10, 12, 14 y 16-bit conversores ADC.

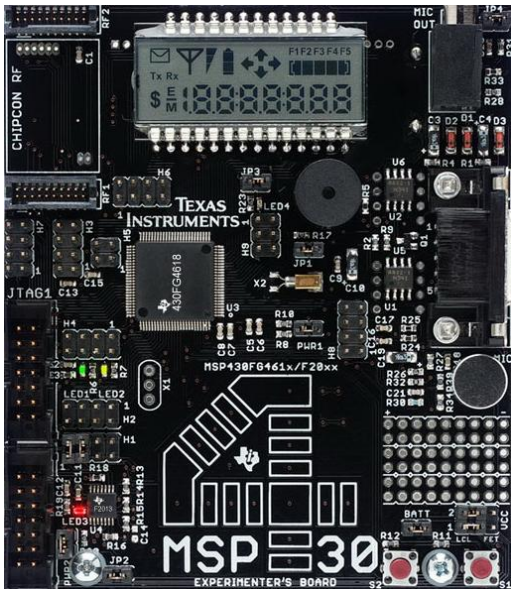

Figura 1.7 Tarjeta de experimentación MSP430FG4618 (ilustrativo)

1.4.2 Familias del MSP430

Se han desarrollado cuatro familias principales del procesador MSP430: La familia 1xx, la 2xx, la 3xx y la 4xx. Los números escritos después de la letra de cada familia identifican primero el modelo, segundo la cantidad de memoria integrada, y tercero alguna variante del modelo [3].

MSP430x1xx

Familia con un extenso rango de dispositivos, desde dispositivos de propósito general hasta dispositivos con un sistema completo para procesamiento digital de señales. Empaquetado de 20 a 64 pines.

MSP430x2xx

Es similar a la familia 1xx pero opera con un menor consumo, soporta velocidades de operación de hasta 16Mhz y son más precisas en la sincronía del reloj, lo que los hace más fáciles de operar sin un cristal externo.

MSP430x3xx

Es la familia más antigua de esta serie de microcontroladores, fue diseñada para instrumentos portátiles con controlador de LCD integrada. Incluye un oscilador que puede ser sincronizado automáticamente con cristales de baja velocidad (32Khz). Esta familia no soporta memorias EEPROM, solamente ROM y EPROM. Actualmente se encuentra obsoleta.

MSP430x4xx

Esta familia es muy similar a la 3xx, también incluye un controlador de LCD pero es mejor y más avanzada, y viene en versiones flash ROM.

1.4.3 Launchpad

El *Launchpad* es una herramienta de evaluación y de desarrollo para los microcontroladores de Texas Instruments de la familia MSP430G.

La tarjeta dispone de un socket DIP de 20 pines para alguno de los microcontroladores de 16 bits de la familia MSP430G y dispone además de una conexión USB que permite descargar y depurar programas directamente en el hardware.

Dispone de dos botones, un par de LEDs y unos headers para poder acceder a los pines del microcontrolador [12].

Características [16]:

- Socket DIP de 20 pines que soporta toda la familia de microcontroladores MSP430G.
- Emulación flash incluida para depuración y programación.
- 1 LED de encendido.
- 2 LEDs programables.
- 1 botón de reset.
- 1 botón programable.

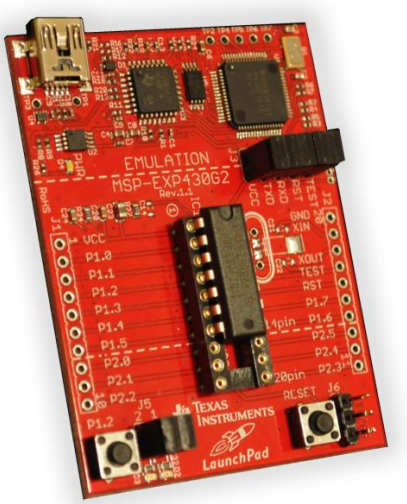

Figura 1.8 *Launchpad*

1.4.4 Microcontrolador MSP430G2553

En este trabajo de tesis se utiliza un microcontrolador MSP430G2553 el cual es uno de los más completos dentro de la familia MSP430G.

Características [13]:

- Rango de voltaje de 1.8 volts a 3.6 volts.
- Ultra bajo consumo de energía.
- Modo activo: 230 mA a 1 MHz, 2.2 volts.
- Modo Apagado (retención de memoria RAM): 0,1 mA.
- Cinco modos de ahorro de energía.
- Arquitectura de 16-Bit RISC, Tiempo de ciclo de instrucción de 62,5 ns.
- Cristal de 32-kHz.
- Fuente externa de reloj digital.
- Dos Timer_A de 16 bits Con tres modos de Captura / Comparación Registros.
- Interfaz de Comunicación Serial Universal (USCI).
- Comunicación serial SPI (Interfase Periférica Serial).
- I2C (Inter-Circuitos Integrados).

CAPÍTULO 2

DISEÑO, CONSTRUCCIÓN Y ENSAMBLADO

2.1 DISEÑO

Para el diseño del hexápodo se buscaron los diferentes modelos de hexápodos disponibles en el mercado y a partir de estos se empezó a esbozar el diseño de las partes mecánicas.

Figura 2.1 Robot Phoenix Lynxmotion hexápodo

Figura 2.2 Robot hexápodo MSR-H01

Después de tener un diseño mecánico esbozado, este se capturo en Autocad para tener una visión del diseño en 2D. Posteriormente este diseño se paso a *SketchUp* para observar las figuras en 3D.

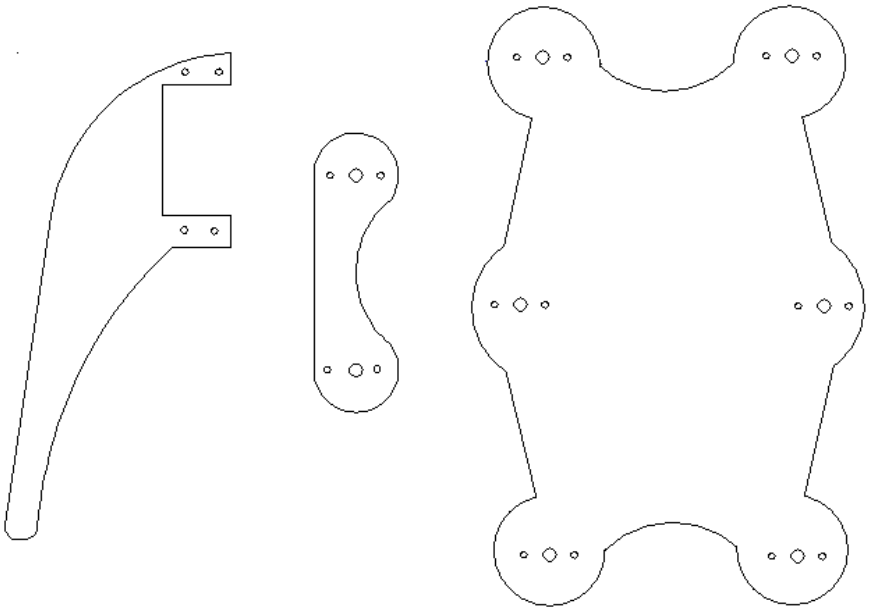

Figura 2.3 Piezas planas del cuerpo y extremidades del hexápodo

2.1.1 SketchUp

SketchUp es un programa de diseño y modelado en tres dimensiones basado en caras, para entornos arquitectónicos, ingeniería civil, ingeniería mecánica, diseño industrial, videojuegos o películas.

Este es un programa desarrollado y publicado por Trimble y una de sus principales ventajas es su fácil manejo y compatibilidad con otros programas de diseño [14].

2.1.2 Extremidad del robot

El robot dispondrá de un cuerpo que permitirá ensamblar seis extremidades, cada extremidad estará compuesta de tres servomotores, de una pata, un fémur y un hombro.

Dado a que es un robot articulado y que cada extremidad tiene tres servomotores se considera este un robot de tres grados de libertad por pata, dando en total dieciocho grados de libertad en todo el sistema.

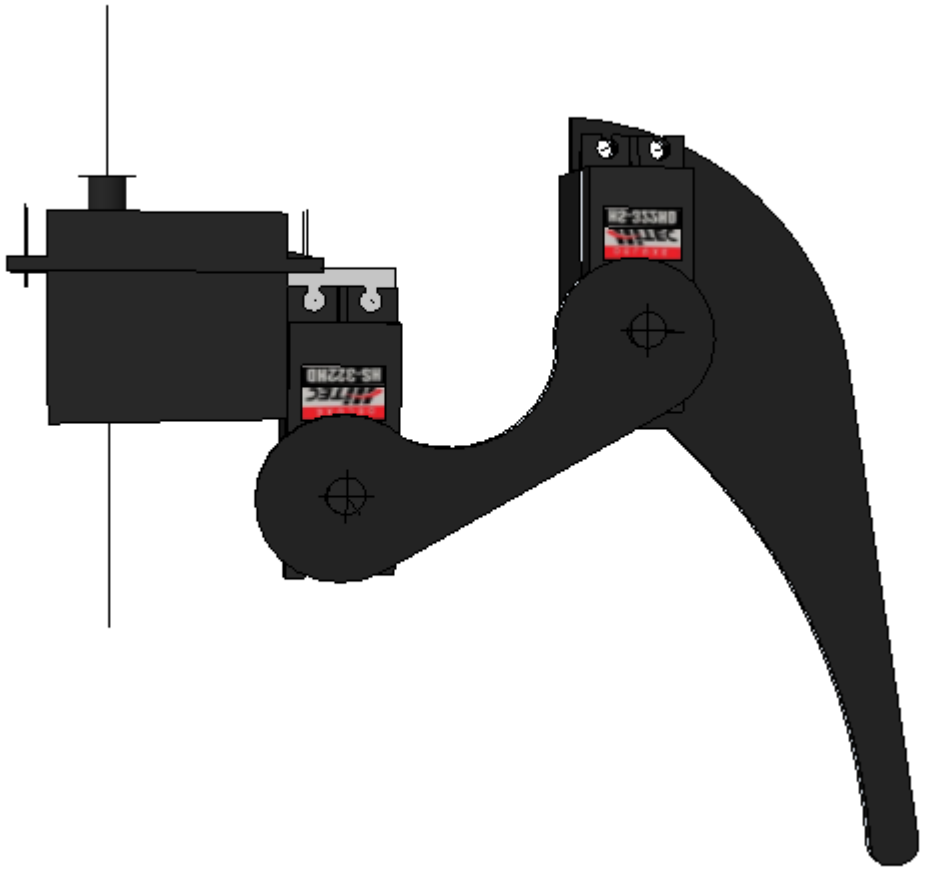

Figura 2.4 Extremidad del robot en SketchUp

2.1.3 Cuerpo del robot

El cuerpo del robot se ha diseñado de manera que se le puedan ensamblar hasta seis articulaciones complejas y es una de las partes principales del dispositivo ya que se encarga de cargar a las articulaciones así como también llevar toda la electrónica del mismo.

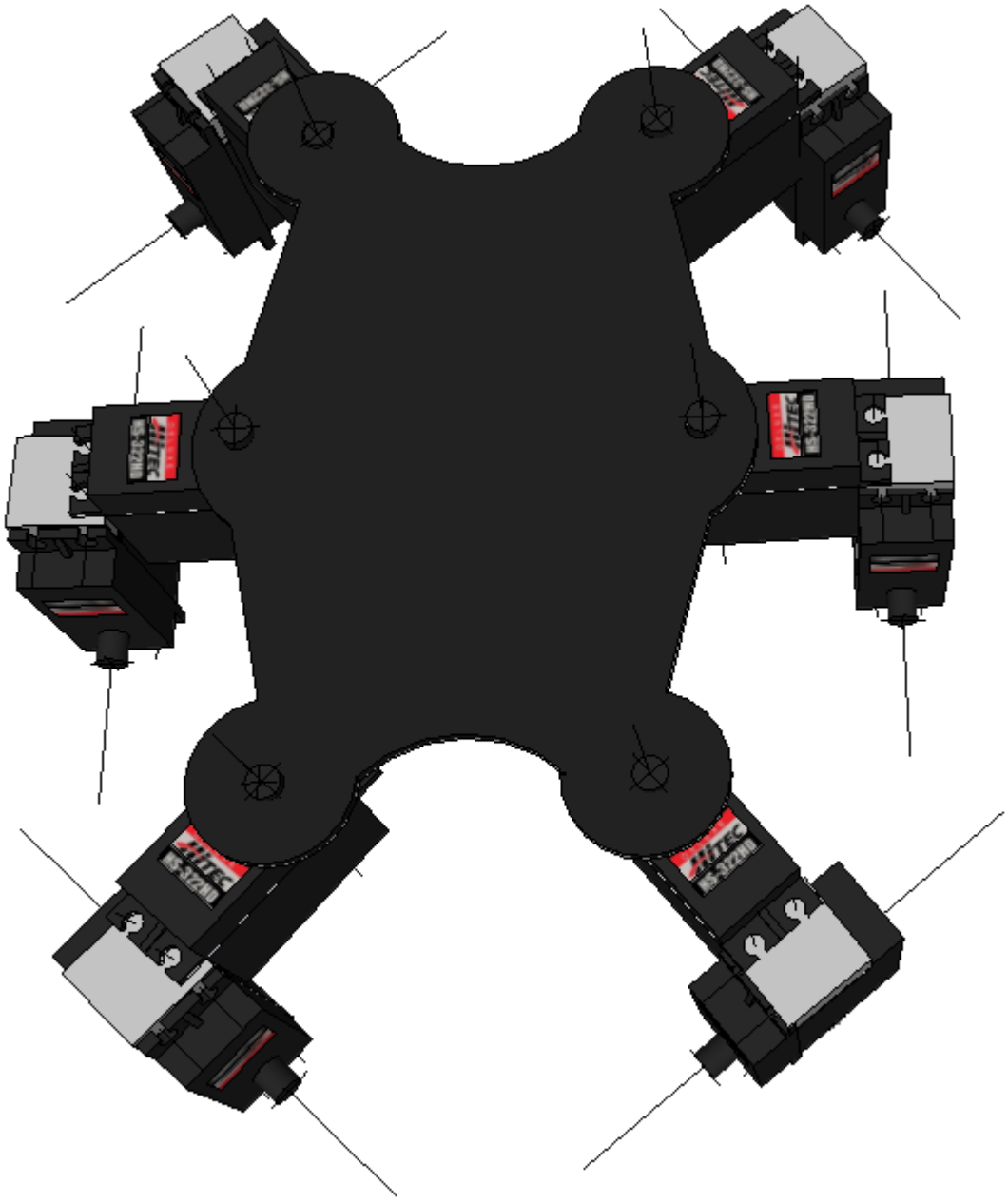

Figura 2.5 Vista superior del cuerpo del robot

2.1.4 Diseño ensamblado

El diseño del ensamblado final hecho en *SketchUp* considerando las extremidades, así como el cuerpo se puede observar en la figura 2.6, 2.7 y 2.8.

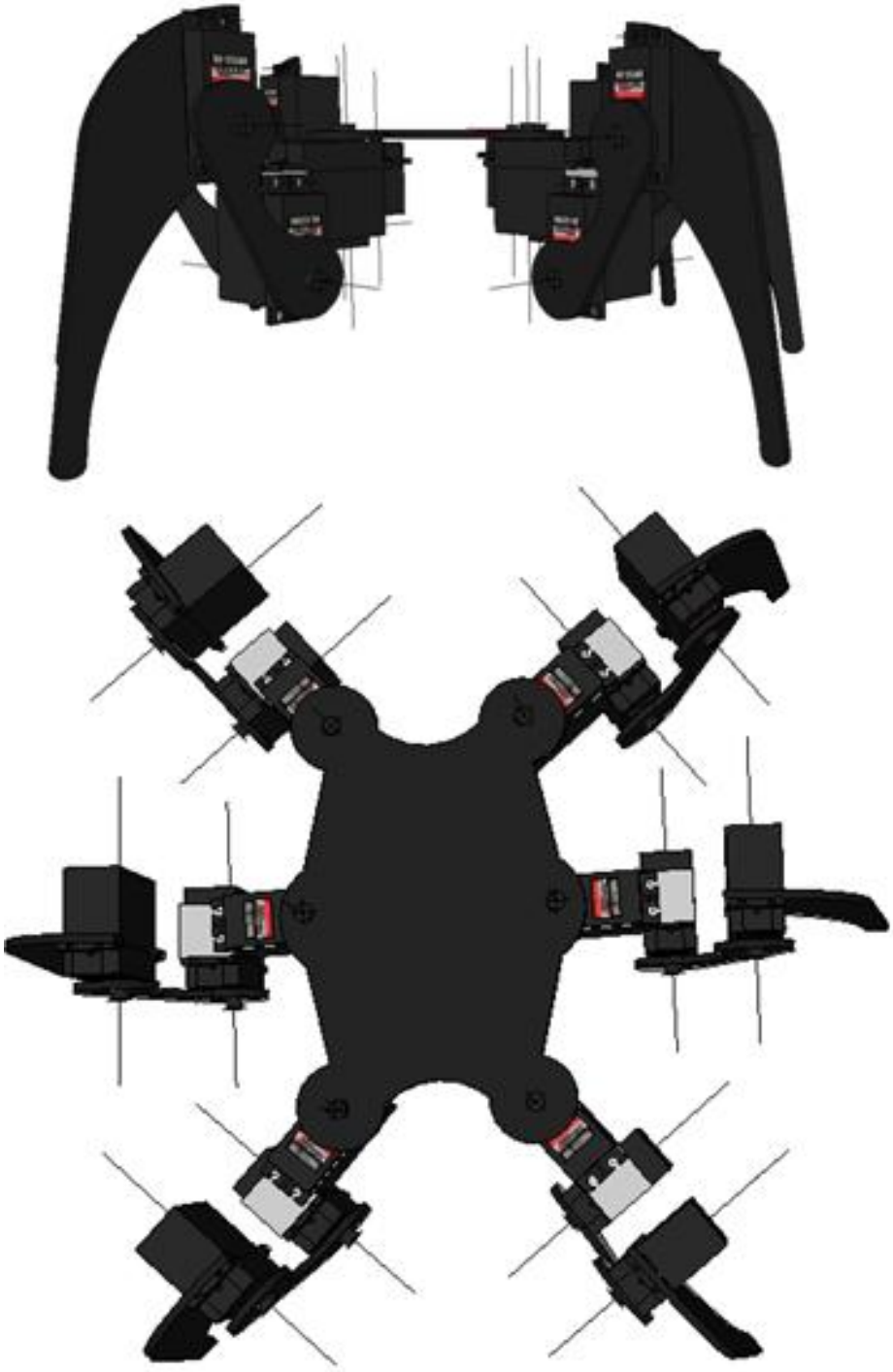

Figura 2.6 Vista frontal y superior

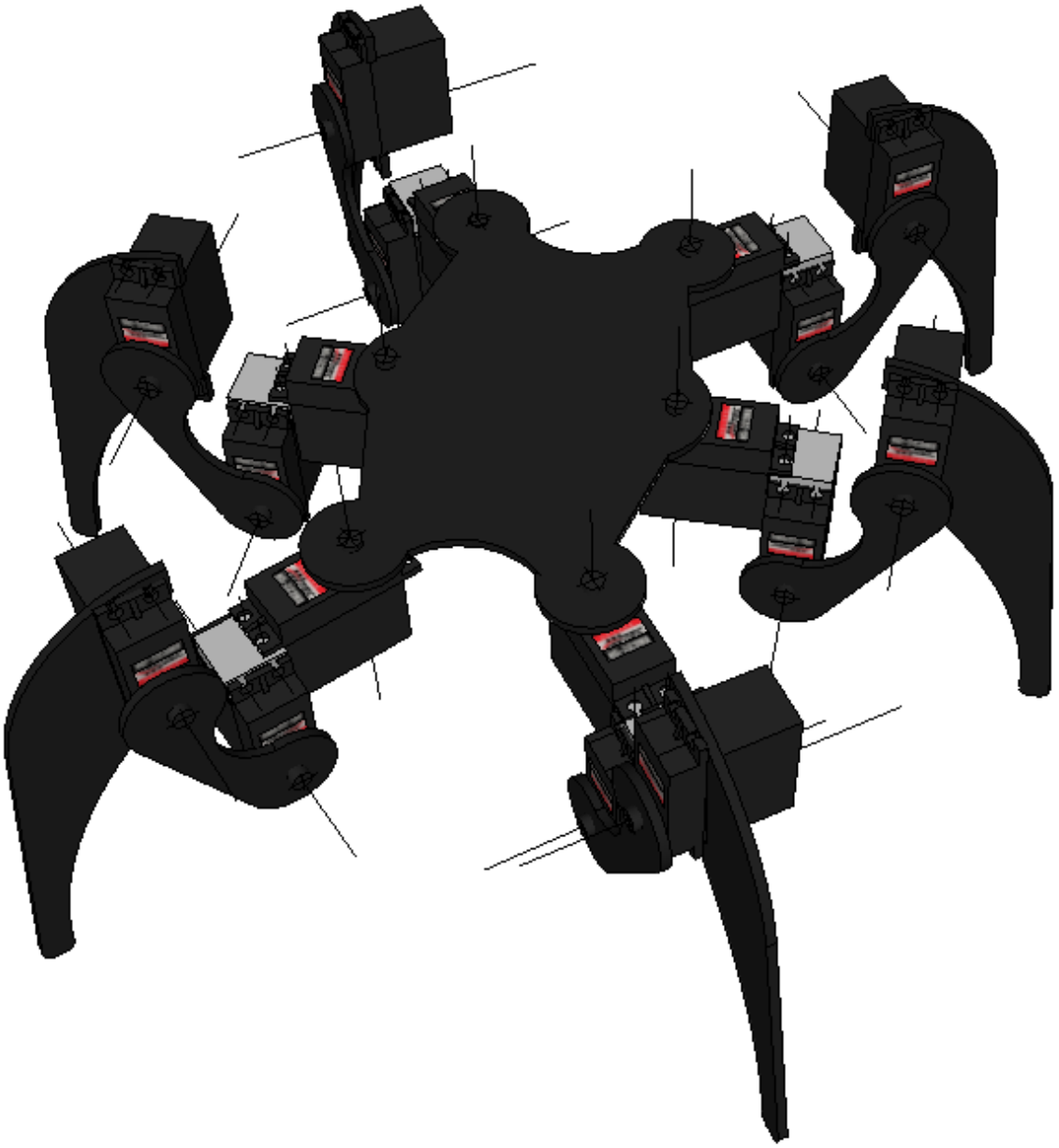

Figura 2.7 Diseño final hexápodo

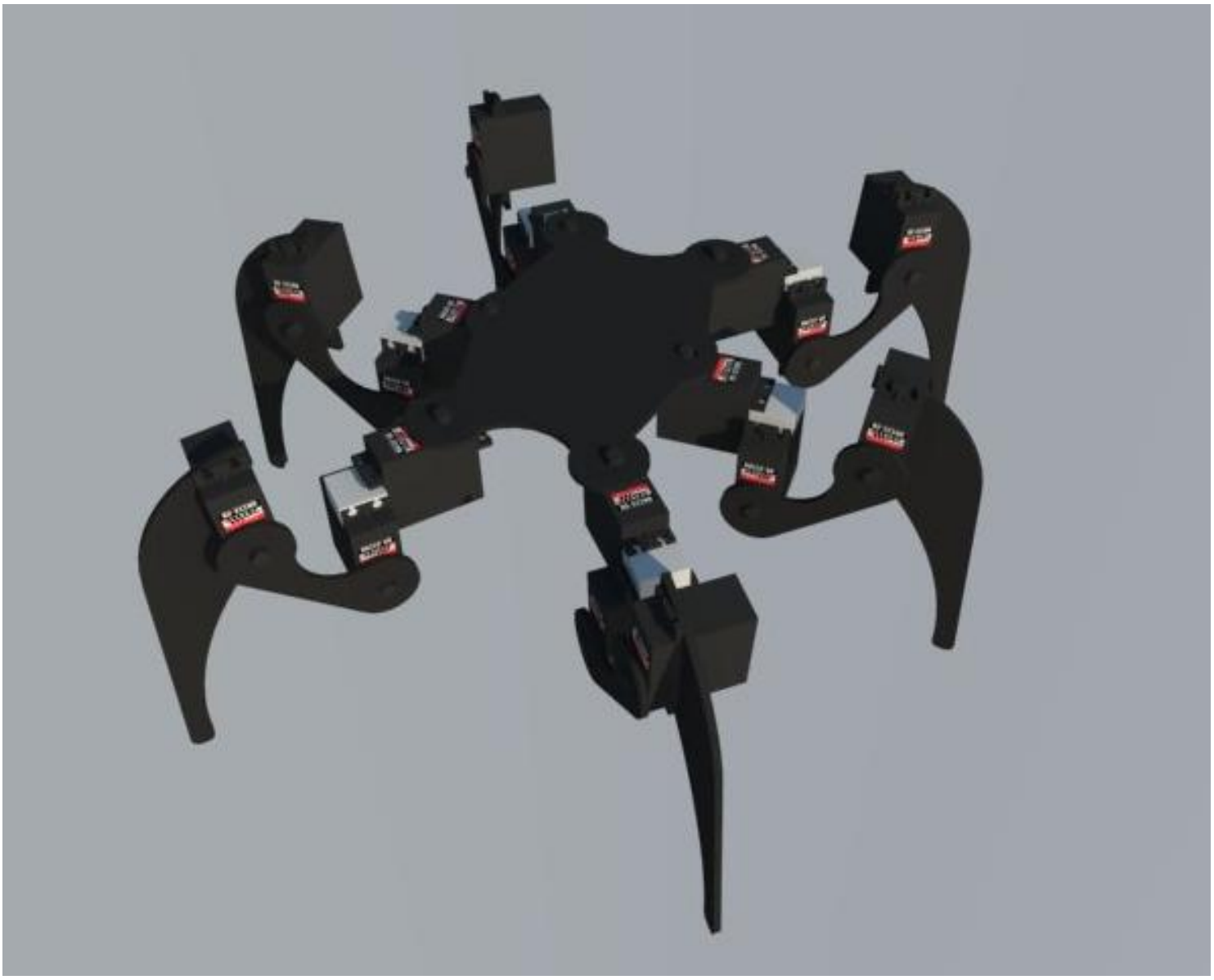

Figura 2.8 Diseño final hexápodo render

2.2 CONSTRUCCIÓN

Para la construcción del dispositivo se ha buscado un material que sea ligero, resistente y económico, por lo que se analizaron las opciones de hacerlo con aluminio, acrílico o madera, estos tres materiales cumplen con las características necesarias para realizar este tipo de trabajo. Sin embargo se llegó a la conclusión de utilizar madera del tipo MDF (*Medium Density Fibreboard*) la cual es muy barata y cumple con todos los requerimientos necesarios. En este caso se utilizo madera MDF con un grueso de 3 mm.

Figura 2.9 Madera MDF

Todas las piezas fueron cortadas por medio de una caladora

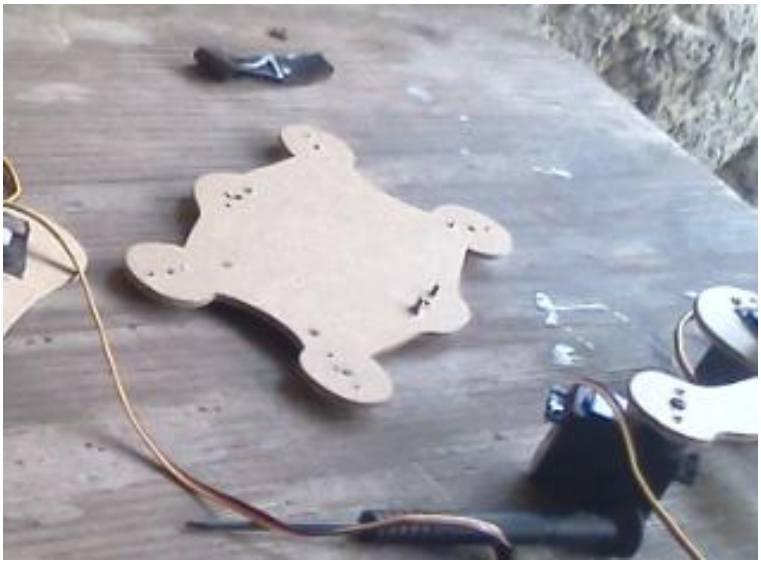

Figura 2.10 Cuerpo MDF cortado

Figura 2.11 Patas MDF cortadas

Figura 2.12 Fémur y hombro MDF cortados

2.3 ENSAMBLADO

Al terminar de cortar todas las piezas contamos con seis patas, seis hombros, seis fémures y el cuerpo principal del robot. Se ensamblaron todas las piezas y se pintaron de esmalte negro.

En la figura 2.13 y 2.14 se puede observar el montaje de las piezas antes de ser ensambladas y unidas finalmente.

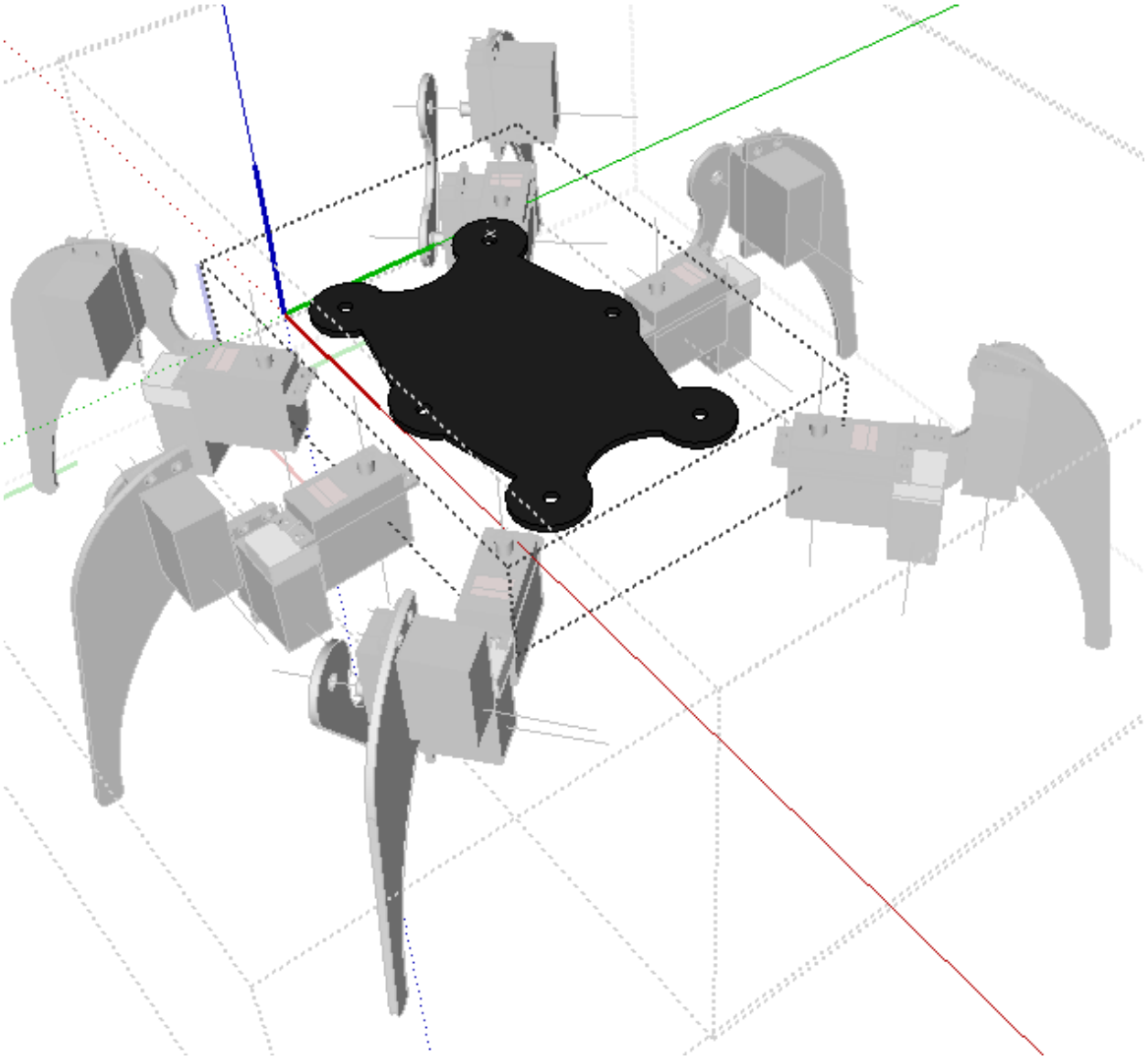

Figura 2.13 Montaje de las piezas

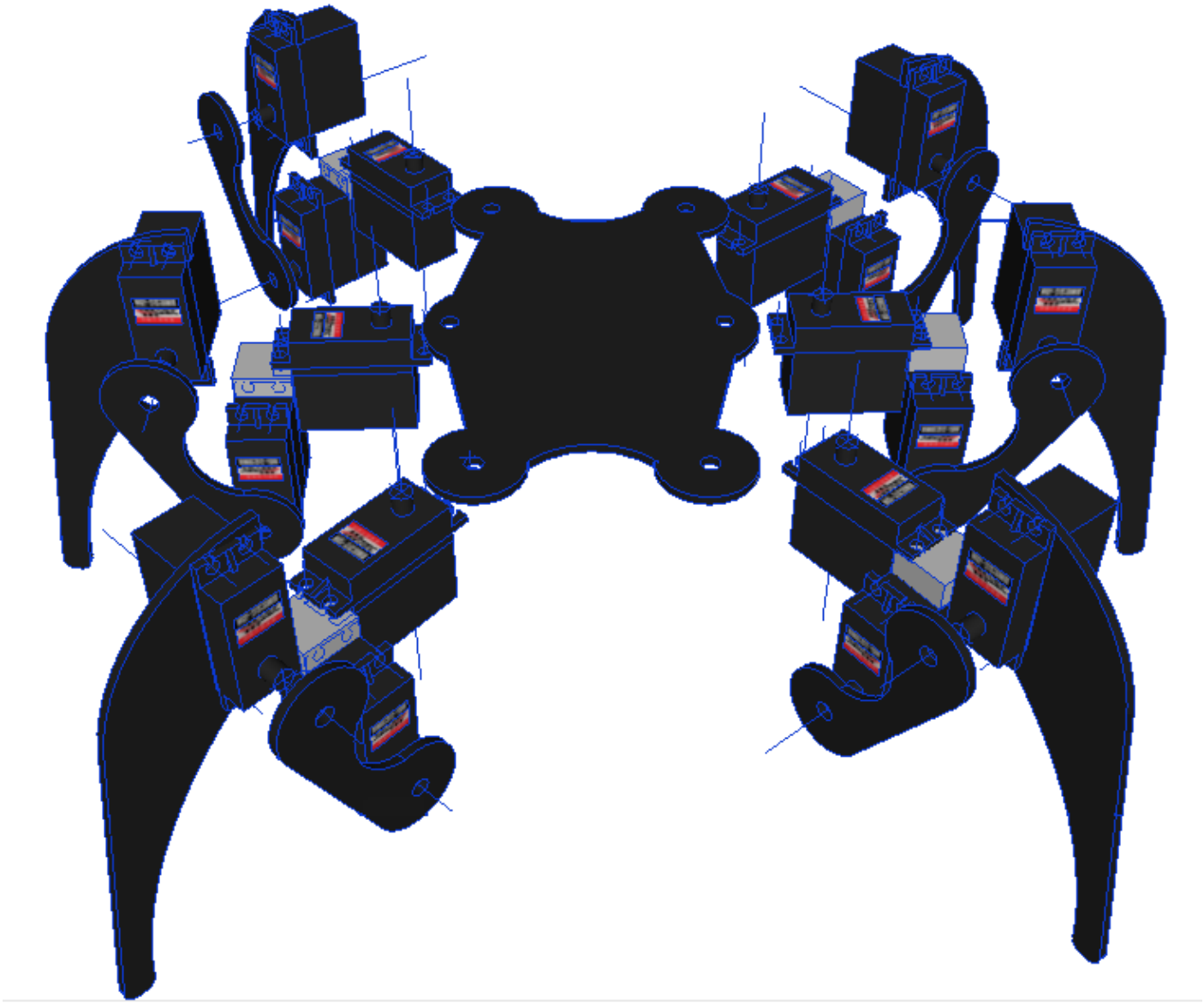

Figura 2.14 Montaje de las piezas vista frontal

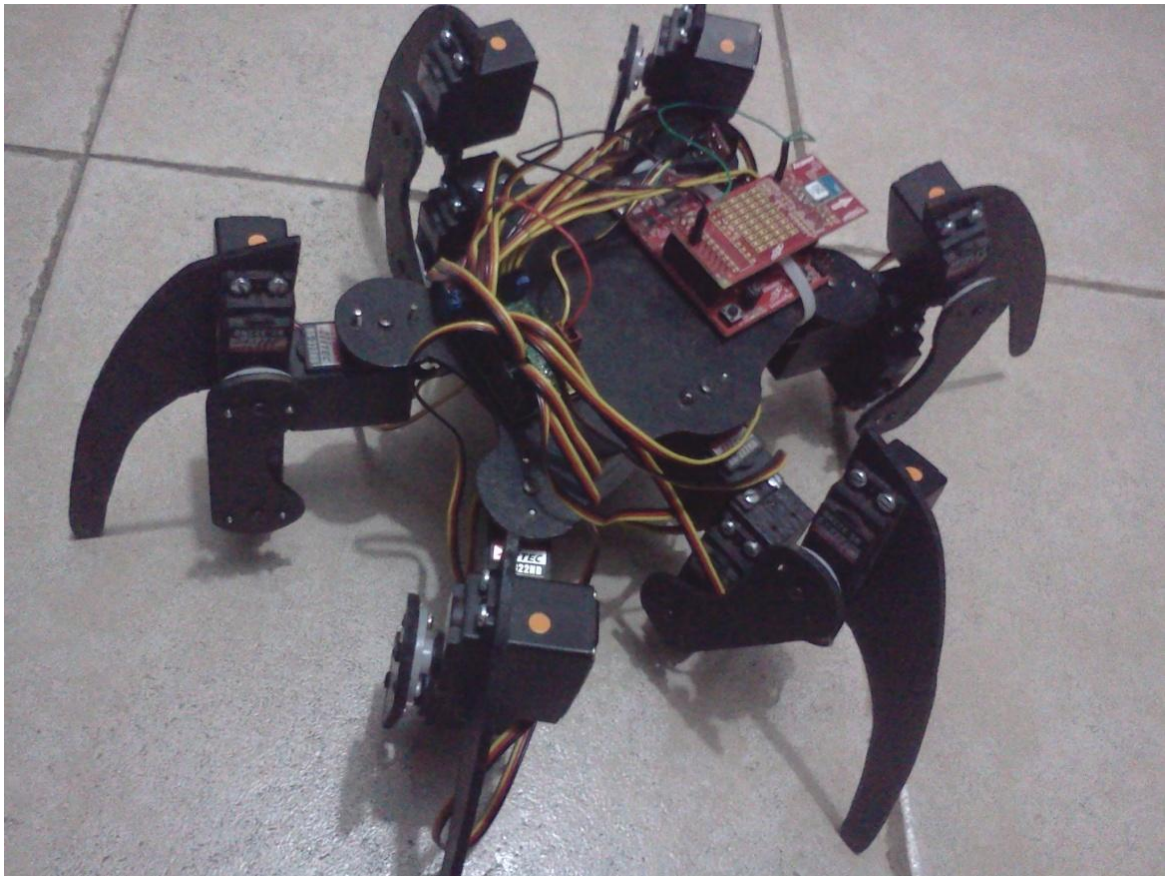

Figura 2.15 Ensamblado

CAPÍTULO 3

MONTAJE ELÉCTRICO

3.1 INTRODUCCIÓN

En este capítulo se detallan los dispositivos electrónicos con los que está compuesto el control remoto, el robot, así como sus diagramas de conexión. Como se mencionó en el primer capítulo, el microcontrolador utilizado para nuestra aplicación será un MSP430G2553 montado sobre una tarjeta de desarrollo *Launchpad*, el cual posee los pines de entrada y salida necesarios para nuestra aplicación.

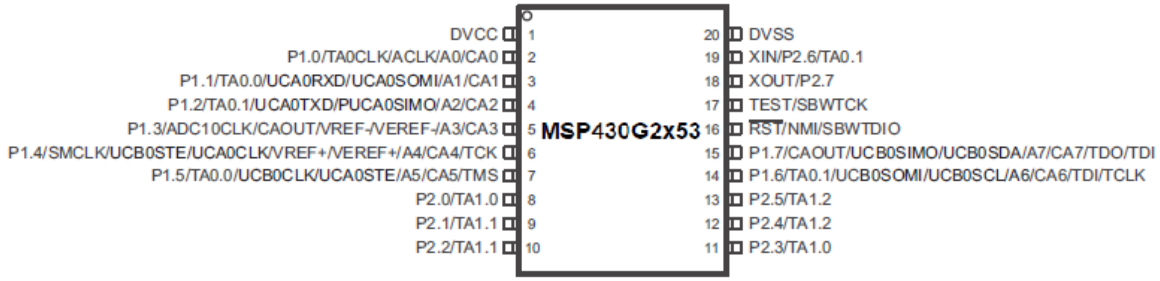

Figura 3.1 Pines de entrada salida del MSP430G2553

3.2 CONTROL REMOTO

El control remoto o radiocontrol nos permite el manejo de un objeto a distancia y de manera inalámbrica mediante una emisora de control remoto.

Tres de los aspectos fundamentales en el control remoto son:

1. La electrónica que transforma las órdenes dadas en ondas de radio en el transmisor y a la inversa en el receptor.
2. La electricidad que proporciona la energía necesaria a los dispositivos tanto al transmisor como el receptor.
3. La mecánica encargada del movimiento mecánico.

El control remoto en este dispositivo se encuentra formado por tres módulos encargados de la interfaz, la transmisión - recepción y el procesamiento de datos: *Touchpad* capacitivo, Transceptor Anaren CC110L y *Launchpad* con el microcontrolador MSP430G2553.

3.2.1 Touchpad capacitivo

El *Touchpad* capacitivo es una tecnología basada en el acoplamiento de capacitancia la cual sensa la capacitancia del cuerpo humano y la toma como una entrada al microcontrolador. Hoy en día el sensado capacitivo es usado en diferentes tipos de sensores como son: de proximidad, posición y desplazamiento, humedad, nivel de fluidos y aceleración.

Para el diseño del control remoto se utilizará un módulo C*apacitive Couch BoosterPack* de Texas Instruments. El módulo contiene tres diferentes tipos de sensado capacitivo: un botón en medio del módulo, una rueda con cuatro diferentes sensores capacitivos y un sensor de proximidad en la parte trasera.

Además el módulo contiene nueve leds para mostrar una retroalimentación con el uso humano. Los nueve leds montados sobre el módulo se encuentran multiplexados para eliminar el uso de varias entradas - salidas al microcontrolador [9].

Figura 3.2 *CapacitiveTouch BoosterPack* montado sobre *Launchpad*

3.2.2 Transceptor Anaren CC110L

El módulo de radiofrecuencia CC110L A*ir BoosterPack* es un transceptor inalámbrico de un bajo consumo energético, es un kit de extensión para ser usado con el módulo *Launchpad* de Texas Instruments. Su funcionamiento está basado en la antena CC110L de Anaren la cual puede ser configurada a diferentes frecuencias 433MHz, 868-870MHz y 902-928MHz [6].

Características:

- Voltaje de operación de1.8 volts a 3.6 volts.
- Bajo consumo energético.
- SPI (Interfase Periférica Serial).
- Área para soldar prototipo.
- Pines para agregar microcontrolador, pushbutton, y led para operación sin *Launchpad.*

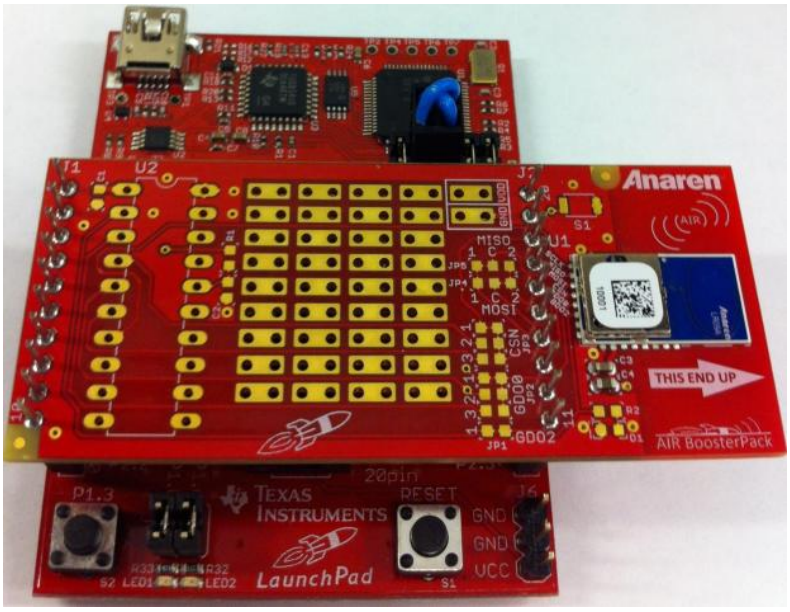

Figura 3.3 *Air BoosterPack* montado sobre *Launchpad*

3.2.3 Montaje del control remoto

El montaje del control remoto quedó constituido por los tres módulos antes mencionados.

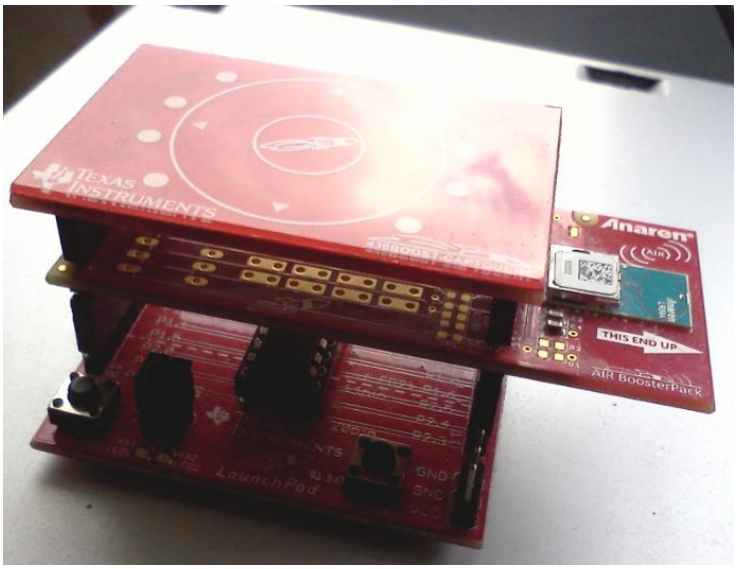

Figura 3.4 Control remoto ensamblado

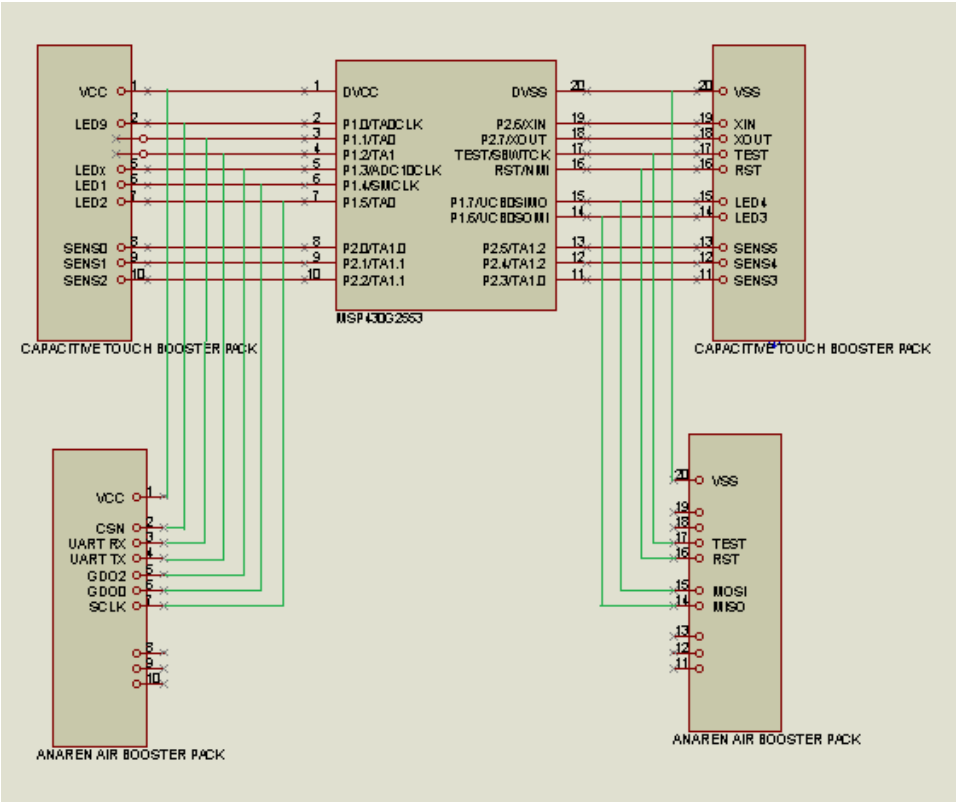

Figura 3.5 Esquemático de los tres módulos I/O

3.3 RECEPTOR - SERVOCONTROLADOR

El receptor (CC110L) estará montado sobre el *Launchpad* y este a su vez estará montado sobre el cuerpo del hexápodo.

Dado que el voltaje de salida del *Launchpad* es de 3.3 volts y el voltaje de funcionamiento de la controladora de servos es de 5 volts es necesario el uso de una interfaz de niveles lógicos para hacer funcionar ambos dispositivos, en nuestro caso utilizamos una interfaz marca Sparkfun.

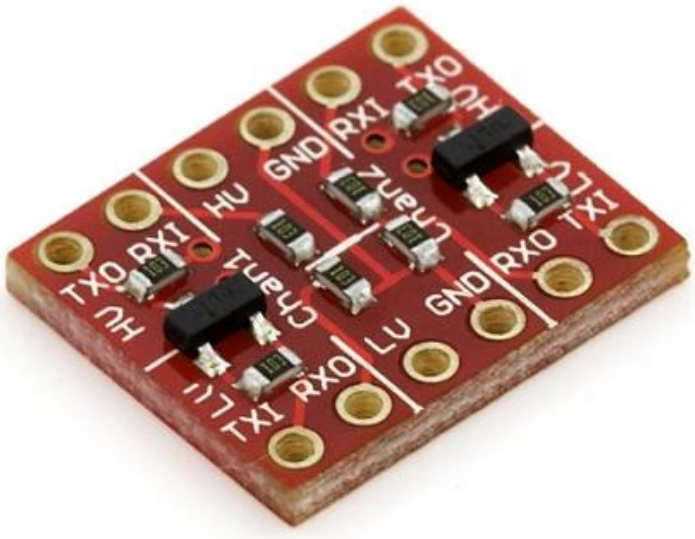

Figura 3.6 Convertidor de niveles lógicos Sparkfun

El orden de conexiones será el siguiente: del *Launchpad* se conecta el puerto serial UART a la interfaz de niveles lógicos y de la interfaz de niveles lógicos se conecta a la controladora de servomotores.

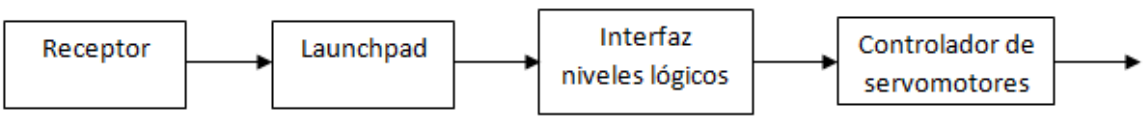

Figura 3.7 Diagrama de bloques de conexión receptor- servocontrolador

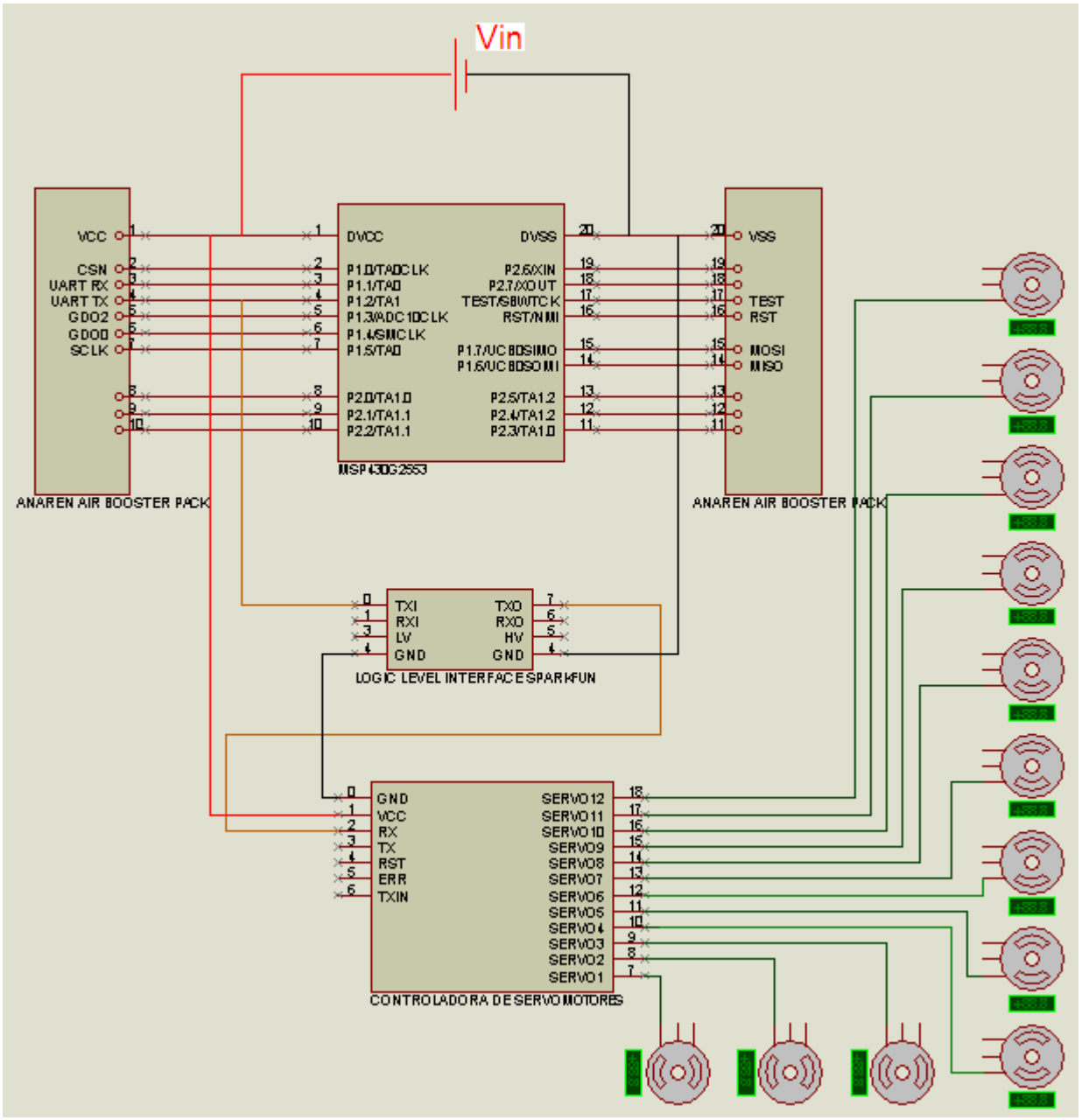

Figura 3.8 Esquemático de conexiones Receptor- Servo controlador

3.4 ALIMENTACIÓN

3.4.1 Alimentación servomotores

Para alimentar eléctricamente todo el dispositivo se buscaron alternativas de baterías de níquel-hidruro metálico (Ni-MH) y baterías de polímero de litio (LIPO) ambas recargables y que entregarán una carga mínima de 2.5 amperes, que pesarán lo menos posibles, y que fueron lo más costeables posible. Analizando ambas baterías resultaban más atractivas las baterías de polímero de litio pero la diferencia en precio era muy elevada, por lo que se decidió comprar unas pilas níquel-hidruro metálico marca GP.

Figura 3.9 Pilas Recargables marca GP

En su funcionamiento normal el robot llega a tener consumos pico de corriente de hasta 3 amperes, dependiendo la posición y esfuerzo de los servomotores. Durante el tiempo de descarga de las baterías el robot empieza a perder estabilidad en la locomoción, por esta razón se busco otra opción para mantener el dispositivo funcionando más tiempo y se opto por ocupar una fuente de computadora que tiene una salida que entrega 5 volts con 12 amperes constantes.

3.4.2 Alimentación microcontrolador y módulos

Para la alimentación del *Launchpad*, C*apacitive Touch BoosterPack* y CC110L A*ir BoosterPack* se probaron la alimentación con dos pilas diferentes: una pila de litio tipo botón de 3V que soporta una carga de 300 mAh y una pila de polímero de litio (LIPO) de 3 volts y que soporta una carga de hasta 600 mAh. En el caso de la pila tipo botón la carga llego a durar hasta una hora en uso continuo y en el caso de la pila de polímero de litio la carga llego a durar dos horas, siendo bastante eficientes ambas pilas para el consumo de corriente de nuestra aplicación.

Figura 3.10 Pilas Recargables tipo LIPO

CAPÍTULO 4

ALGORITMOS Y PROGRAMACIÓN

4.1 INTRODUCCIÓN

A continuación se presenta el software de control que permite el movimiento de nuestro dispositivo, el lenguaje de programación utilizado fue C y el compilador utilizado para correr y depurar los programas es el *Code Composer Studio* V 4.0 el cual es de libre distribución por parte de Texas Instruments.
Los programas se desarrollaron en dos módulos principales (transmisor y receptor).

El primer módulo está programado en el control remoto y funciona como transmisor, su función es la de enviar la posición del robot al receptor. El segundo módulo es el software que sirve como receptor y es el encargado de recibir y enviar la posición a cada uno de los servomotores.

4.2 PROGRAMACIÓN

La programación de un robot se puede definir como el proceso mediante el cual se le indica a este la secuencia de acciones que deberá llevar a cabo durante la realización de su tarea. Estas acciones consisten en su mayor parte en moverse a puntos predefinidos y mover objetos del entorno.

Durante la ejecución de un programa se trabaja con la memoria del sistema, leyendo y actualizando el contenido de las variables utilizadas en el programa.

El sistema de programación es por tanto la herramienta con que cuenta el usuario para acceder a las diversas prestaciones del robot, existiendo una relación directa entre las características y posibilidades del sistema de programación y las del robot en sí mismo [4].

4.2.1 Métodos de programación de robots

Existen diversos criterios para realizar una clasificación de los métodos de programación de robots. Algunas se derivan de la potencia del método, mientras que otras clasificaciones hacen referencia al sistema empleado para indicar la secuencia de acciones a realizar.

4.2.1.1 Programación por guiado

La programación por guiado o por aprendizaje consiste en hacer realizar al robot acciones manualmente por un guiado exterior al tiempo que se registran las configuraciones adoptadas para su posterior repetición de manera automática
Para guiar al robot por las trayectorias o puntos deseados se utilizan diferentes soluciones atendiendo a la potencia del sistema se pueden dividir en guiado básico y guiado extendido.

4.2.1.1.1 Guiado básico

El robot es guiado consecutivamente por los puntos por los que se quiere que pase durante la fase de ejecución automática del programa. Ocasionalmente no es posible incluir ningún tipo de estructuras de control dentro del programa, por lo que los puntos son recorridos siempre secuencialmente, en el mismo orden en que se programaron.

4.2.1.1.2 Guiado extendido

Permite especificar, junto a los puntos por los que deberá pasar el robot, datos relativos a la velocidad, tipo de trayectoria, precisión, control de flujo del programa, etc [4].

4.2.1.2 Programación textual

El método de programación textual permite indicar la tarea al robot mediante el uso de un lenguaje de programación específico.
La programación textual puede ser clasificada en tres niveles: robot, objeto y tarea, dependiendo de las ordenes se refieran a los movimientos a realizar por el robot, al estado en que deben ir quedando los objetos manipulados o al objetivo a conseguir [4].

Figura 4.1 Robot de pintura de programación mediante guiado pasivo

4.3 CONTROL

Dado que nuestro sistema será controlado por medio de un control remoto y además no contará con ningún tipo de sensor que envié retroalimentación al microcontrolador, podemos afirmar que nuestro sistema de control es de lazo abierto.

4.3.1 Sistemas de control en lazo abierto

Los sistemas en los cuales la salida no tiene efecto sobre la acción de control se denomina de lazo abierto. En un sistema de lazo abierto no se mide la salida ni se realimenta para compararla con la entrada.

4.4 PROGRAMA DE TRANSMISIÓN

Nuestro robot será programado por un guiado básico con un sistema de control de lazo abierto, el control remoto será el encargado del programa de transmisión.

4.4.1 Programa control remoto

El control remoto contará con cinco botones táctiles que nos permitirán controlar ocho rutinas diferentes: cuatro posiciones de parado, caminar al frente, caminar atrás, girar a la izquierda y girar a la derecha.

Como se menciona en el tercer capítulo el control remoto estará constituido por tres módulos de hardware: C*apacitive Touch BoosterPack*, A*ir BoosterPack* CC110L y *Launchpad*. El programa así como la configuración de entrada y salidas de los módulos estarán programados en un microcontrolador MSP430G2553.

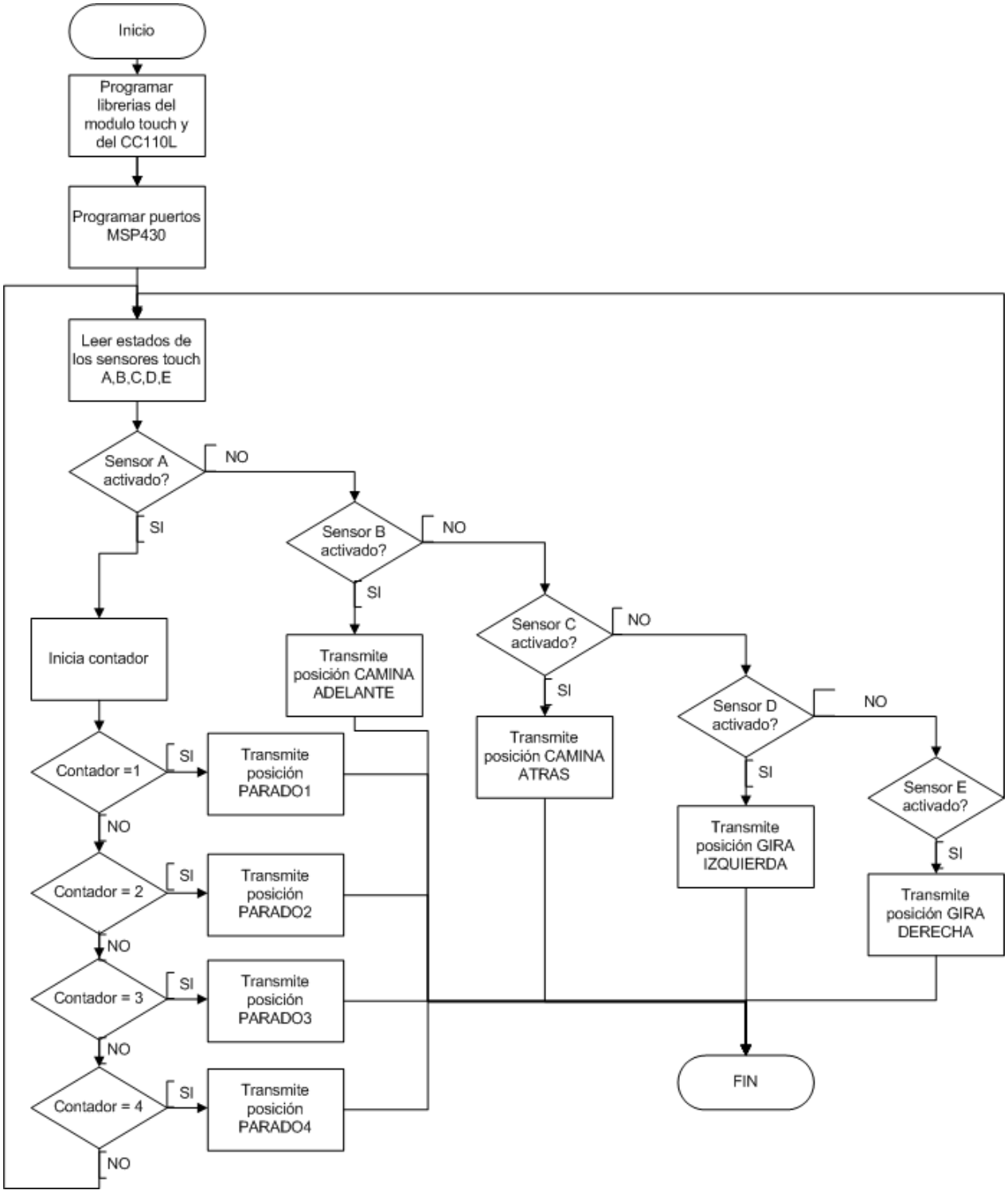

Figura 4.2 Diagrama de bloques del transmisor

4.4.2 Configuración del Capacitive Touch BoosterPack (interfaz táctil)

Como se menciono en el cuarto capítulo el *touchpad* capacitivo es un módulo desarrollado para ser usado exclusivamente con el *Launchpad* y con microcontroladores de la familia MSP430 de veinte pines.

El módulo *Capacitive Touch BoosterPack* contiene tres diferentes tipos de sensores: sensor de proximidad, sensor por botón y sensor por grupo de botones.

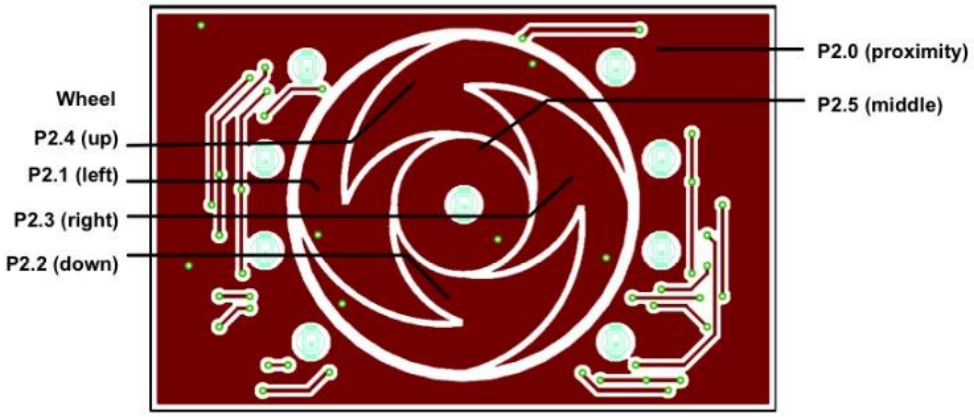

Figura 4.3 *Touch BoosterPack* I/O [7]

Cabe mencionar que Texas Instruments incluye una librería específica para el módulo táctil la cual permite desarrollar rápidamente aplicaciones de sensado capacitivo para ser implementados en microcontroladores de la familia MSP430 y en el *Launchpad*.

El principio de funcionamiento se basa en comparar dos diferentes dominios de tiempo. Un dominio se encontrará fijo y el otro será variable como una función de la capacitancia.

En este trabajo se utilizará el método de oscilador relajado para medir el número de ciclos de oscilación en un periodo fijo.

4.4.2.1 Oscilador relajado

El método del oscilador relajado puede ser aplicado mediante el uso de un comparador o mediante el uso del PinOsc el cual es una característica del MSP430G2553.

Cuando el PinOsc es habilitado, la estructura interna del microcontrolador combinada con un sensor capacitivo externo produce un oscilador interno. Esta oscilación será directamente proporcional a la capacitancia del sensor externo, por lo que ésta será la frecuencia que mediremos y posteriormente compararemos. [7]

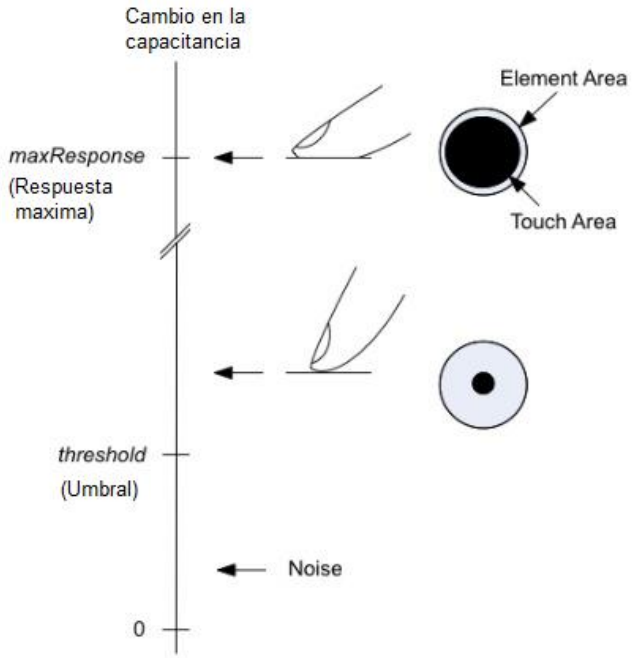

Figura 4.4 Parámetro de medición de un botón [9]

La librería del sensado capacitivo tiene que ser configurada definiendo los puertos, el método de sensado, el número de elementos y otros factores extra.

Dos parámetros importantes dentro de la configuración son el *threshold* (umbral) y *maxresponse* (respuesta máxima). El valor del *threshold* será el valor mínimo donde se puede sensar el uso del *touchpad*, este valor es calculado mediante la expresión; *treshold* = (TouchCnt - noTouchCnt)/2 pero para esta aplicación tomamos el valor estándar de Texas Instruments de *threshold* = 100 y *maxResponse* = *threshold* + 655.

Al configurar la librería tendremos que declarar algunas características del código structure.c como son la estructura de nuestro método de sensado, declaración de la estructura, declarar el número de elementos a sensar, declarar los botones a sensar, seleccionar el reloj para el WDT, entre otras más:

```
const struct Sensor wheel_buttons =                    //DECLARACION DE LA ESTRUCTURA
            {
               .halDefinition = RO_PINOSC_TA0_WDTp,    //DECLARACION DEL METODO DE SENSADO
               .numElements = 5,                       //DECLARA NUMERO DE ELEMENTOS A SENSAR
               .baseOffset = 0,                        //PRIMER ELEMENTO
               .arrayPtr[0] = &up_element,             // DECLARACION DEL BOTON ARRIBA
               .arrayPtr[1] = &down_element,           // DECLARACION DEL BOTON ABAJO
               .arrayPtr[2] = &left_element,           // DECLARACION DEL BOTON IZQUIERDA
               .arrayPtr[3] = &right_element,          // DECLARACION DEL BOTON DERECHA
               .arrayPtr[4] = &middle_element,         // DECLARACION DEL BOTON ENMEDIO
               // Timer Information
               .measGateSource= GATE_WDT_ACLK,      //  0->SMCLK, 1-> ACLK (SELECCIONA ACLK PARA EL WDT)
               .accumulationCycles= WDTp_GATE_64    //64 - Default
            };
```

Por otro lado se debe configurar cada botón que será sensado, se debe agregar el valor del umbral, el valor de la respuesta máxima, así como el número del pin y el puerto al que será asignado dicho botón:

```
#include "structure.h"
const struct Element middle_element = {

                .inputPxselRegister = (unsigned char *)&P2SEL,
                .inputPxsel2Register = (unsigned char *)&P2SEL2,
                .inputBits = BIT5,              //P2.5
                .maxResponse = 100+655,         //RESPUESTA MAXIMA
                .threshold = 100                //UMBRAL
};

const struct Element up_element = {

                .inputPxselRegister = (unsigned char *)&P2SEL,
                .inputPxsel2Register = (unsigned char *)&P2SEL2,
                .inputBits = BIT4,              //P2.4
                .maxResponse = 100+655,         //RESPUESTA MAXIMA
                .threshold = 100                //UMBRAL
};

const struct Element left_element = {

                .inputPxselRegister = (unsigned char *)&P2SEL,
                .inputPxsel2Register = (unsigned char *)&P2SEL2,
                .inputBits = BIT1,              //P2.3
                .maxResponse = 100+655,         //RESPUESTA MAXIMA
                .threshold = 100                //UMBRAL
};

const struct Element right_element = {

                .inputPxselRegister = (unsigned char *)&P2SEL,
                .inputPxsel2Register = (unsigned char *)&P2SEL2,
                .inputBits = BIT3,              //P2.2
                .maxResponse = 100+655,         //RESPUESTA MAXIMA
                .threshold = 100                //UMBRAL
};
```

Para finalizar la configuración de la librería se debe definir el nombre de cada una de las estructuras en el código del archivo structure.h:

```
#include "msp430.h"
#include <stdint.h>

/* DECLARACION DE ESTRUCTURAS */
extern const struct Element middle_element;
extern const struct Element up_element;
extern const struct Element down_element;
extern const struct Element left_element;
extern const struct Element right_element;
extern const struct Sensor wheel_buttons;
```

4.4.3 Configuración del módulo inalámbrico

Como se ha mencionado el módulo inalámbrico que utilizaremos será el CC110L A*ir BoosterPack*. Para programar este módulo el fabricante incluye una librería específica para poder configurar el dispositivo, además existen otras librerías que son de otros dispositivos y que son compatibles con el chip CC110L.

Para comenzar hay que mencionar que el CC110L es un módulo de RF que trabaja en tres frecuencias diferentes: 433 MHz, 868 MHz y 915 MHz Es controlado por el *Launchpad* por medio de una interfaz periférica serial SPI. Los pines del CC110L a controlar son: SCLK, SOMI, SIMO, CSN y GDO. El estado del dispositivo se conoce leyendo los registros del microcontrolador y se configura escribiendo en los mismos registros.

Port Mappings for LaunchPad / BoosterPack target board / MSP430G2553 and EZ430-RF2500 / EZ4x / MSP430F2274			
Signal Name	**EZ4X**	**BoosterPack target board**	**BoosterPack target board Notes**
GDO0	P2.6	P2.6	Set by JP2
GDO2	P2.7	P1.0	Set by JP1 -- Also connected to GDO2
UCB0SIMO/MOSI	P3.1	P1.7	Set by JP4
UCB0SOMI/MISO	P3.2	P1.6	Set by JP5
UCB0CLK/SCLK	P3.3	P1.5	
CSN	P3.0	P2.7	Set by JP3
LED1 (Red)	P1.0	P1.0	Also connected to GDO2
LED2 (Green)	P1.1	P1.4	Only when mated to LaunchPad. Requires jumper change. Disconnect P1.6 to LED2 Jumper. Connect Jumper from LED2 to J1 pin 6 on BoosterPack target board.
Switch	P1.2	P1.3	Marked as SW2 on LaunchPad
UCA0TXD	P3.4	P1.2	Requires Criss-Cross Jumpers to operate with USB interface on LaunchPad.
UCA0RXD	P3.5	P1.1	Requires Criss-Cross Jumpers to operate with USB interface on LaunchPad.

Tabla 4.1 Pines a controlar de CC110L [8]

Para nuestra aplicación tendremos que configurar en los registros de la biblioteca la frecuencia, la potencia de transmisión, así como asignar los pines que utilizaremos con el *Launchpad*.

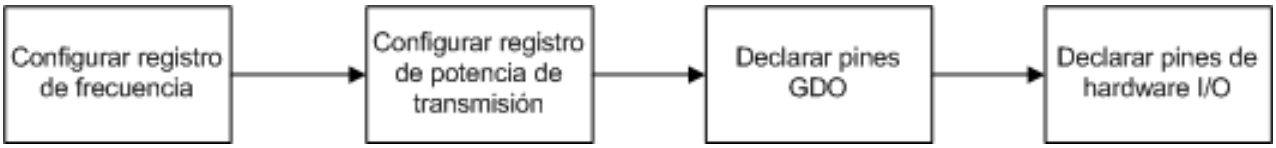

Figura 4.5 Algoritmo de programación de la librería del CC110L

Configuramos la frecuencia en la que trabajaremos en la librería CC1100-CC2500.c en este caso se escogió la frecuencia de 868 MHz:

```
#include "include.h"
#include "TI_CC_CC1100-CC2500.h"
#define TI_CC_RF_FREQ  868                    // 433, 868, 915
```

Además de esto el archivo CC1100-CC2500.c de la biblioteca nos da la facilidad de modificar los registros de la potencia a la que queremos que transmita para poder tener un mayor o menor rango de transmisión.

Power (dBm)	PA_Table(Hex)	Power (dBm)	PA_Table(Hex)
12	0xC0	4.2	0x84
11	0xC1	4	0x85
10.5	0xC2	3.6	0x86
10.3	0xC3	3.4	0xCF
10	0xC4	3	0x88
9.6	0xC5	2.5	0x8A
9.2	0xC6	2	0x8B
9	0xC7	1	0x8D
8.6	0xC8	0	0x8E
8.2	0xC9	-0.5	0x70
8	0xCA	-1	0x60
7.6	0xCB	-2	0x40
7.2	0xCC	-2.2	0x62
7	0xCD	-5	0x67
6.2	0xCE	-10	0x6D
5	0x80	-15	0x24
4.8	0x81	-20	0x22
4.6	0x82	-25	0x14
4.4	0x83	-30	0x03

Tabla 4.2 Algoritmo de programación de la librería del CC110L [6]

Para esta aplicación se configuró el registro para que diera una potencia de -1 dBm equivalente a 0.8 mW, lo cual nos da un rango aproximado de transmisión de 5-8 mts, suficiente para controlar nuestro dispositivo además de dar un mayor periodo de vida a las baterías:

```
// PATABLE (-1 dBm output power)
extern char paTable[] = {0x60};
extern char paTableLen = 1;
#endif
```

Finalmente definiremos el hardware de los pines de entrada/salida de nuestra tarjeta en el archivo de la biblioteca TI_CC_hardware_board_launchpad.h el cual es muy importante revisar al momento de analizar el programa main.c.

```
#define TI_CC_LED_PxOUT          P1OUT
#define TI_CC_LED_PxDIR          P1DIR
#define TI_CC_LED_PxIN           P1IN
#define TI_CC_LED_PxIE           P1IE
#define TI_CC_LED_PxIES          P1IES
#define TI_CC_LED_PxIFG          P1IFG
#define TI_CC_LED_PxDIR          P1DIR
#define TI_CC_LED1               BIT0

#define TI_CC_GDO0_PxOUT         P2OUT
#define TI_CC_GDO0_PxIN          P2IN
#define TI_CC_GDO0_PxDIR         P2DIR
#define TI_CC_GDO0_PxIE          P2IE
#define TI_CC_GDO0_PxIES         P2IES
#define TI_CC_GDO0_PxIFG         P2IFG
#define TI_CC_GDO0_PIN           BIT6

#define TI_CC_SW_PxIN            P1IN
#define TI_CC_SW_PxIE            P1IE
#define TI_CC_SW_PxIES           P1IES
#define TI_CC_SW_PxIFG           P1IFG
#define TI_CC_SW_PxDIR           P1DIR
#define TI_CC_SW_PxREN           P1REN
#define TI_CC_SW_PxOUT           P1OUT
#define TI_CC_SW1                BIT3
#define TI_CC_SW2                BIT4

#define TI_CC_GDO1_PxOUT         P3OUT
#define TI_CC_GDO1_PxIN          P3IN66
#define TI_CC_GDO1_PxDIR         P3DIR
#define TI_CC_GDO1_PIN           0x04

#define TI_CC_GDO2_PxOUT         P1OUT
#define TI_CC_GDO2_PxIN          P1IN
#define TI_CC_GDO2_PxDIR         P1DIR
#define TI_CC_GDO2_PIN           BIT0
```

El hardware del programa de recepción estará compuesto por un módulo CC110L, un *Launchpad* y la controladora de servomotores Pololu. Lo que hará el programa será habilitar una interrupción cada vez que se detecte un paquete de datos entrando por el pin GDO (P2.6) del módulo CC110L al *Launchpad*, al recibir un paquete de datos el *Launchpad* envía la posición de cada servomotor por medio del puerto serial UART del MSP430 a la controladora de servomotores.

Figura 4.6 Diagrama de flujo programa de recepción

4.5 PROGRAMACIÓN DE RECEPCIÓN

4.5.1 Programación de movimientos

El dispositivo tiene la capacidad de desplazarse hacia adelante y hacia atrás, girar a la izquierda y a la derecha, y además posee la capacidad de tener cuatro diferentes posiciones de parado.

4.5.2 Posiciones de parado

Se han programado cuatro posiciones diferentes de parado que son controladas por un mismo botón del módulo táctil: posición de reposo, parado bajo, parado y parado alto. Para alcanzar cada posición se ha programado un contador que suma el número de pulsaciones que se hacen al sensor táctil, cada número de pulsación tiene una posición programada que se repite indefinidamente.

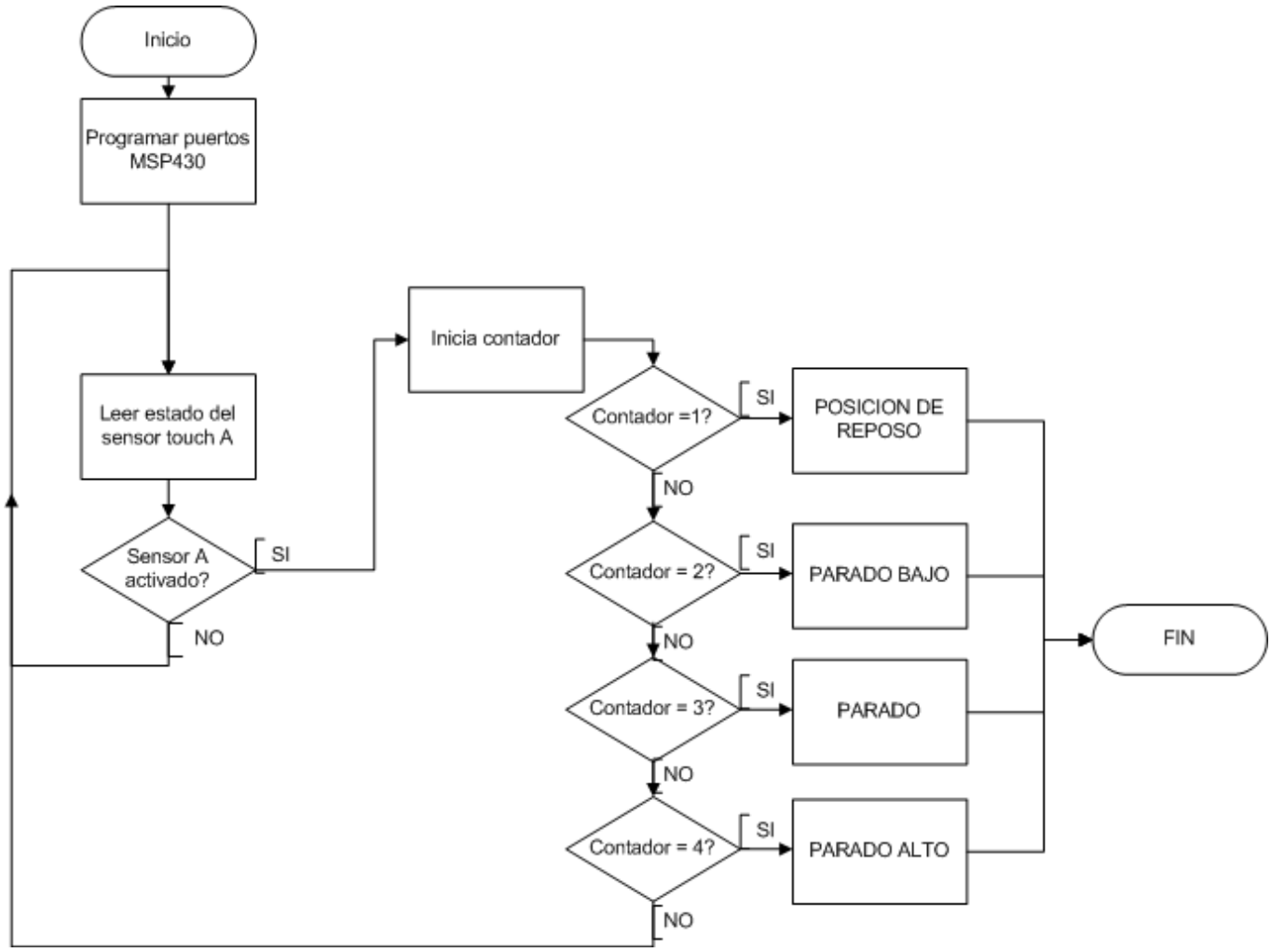

Figura 4.7 Diagrama de flujo posición de parado

Cabe mencionar que para la programación de los otros movimientos se parte siempre de una posición inicial que en nuestro caso es la posición de parado. En la figura 4.8 Podemos observar las cuatro posiciones programadas de parado: reposo, parado bajo, parado y parado alto.

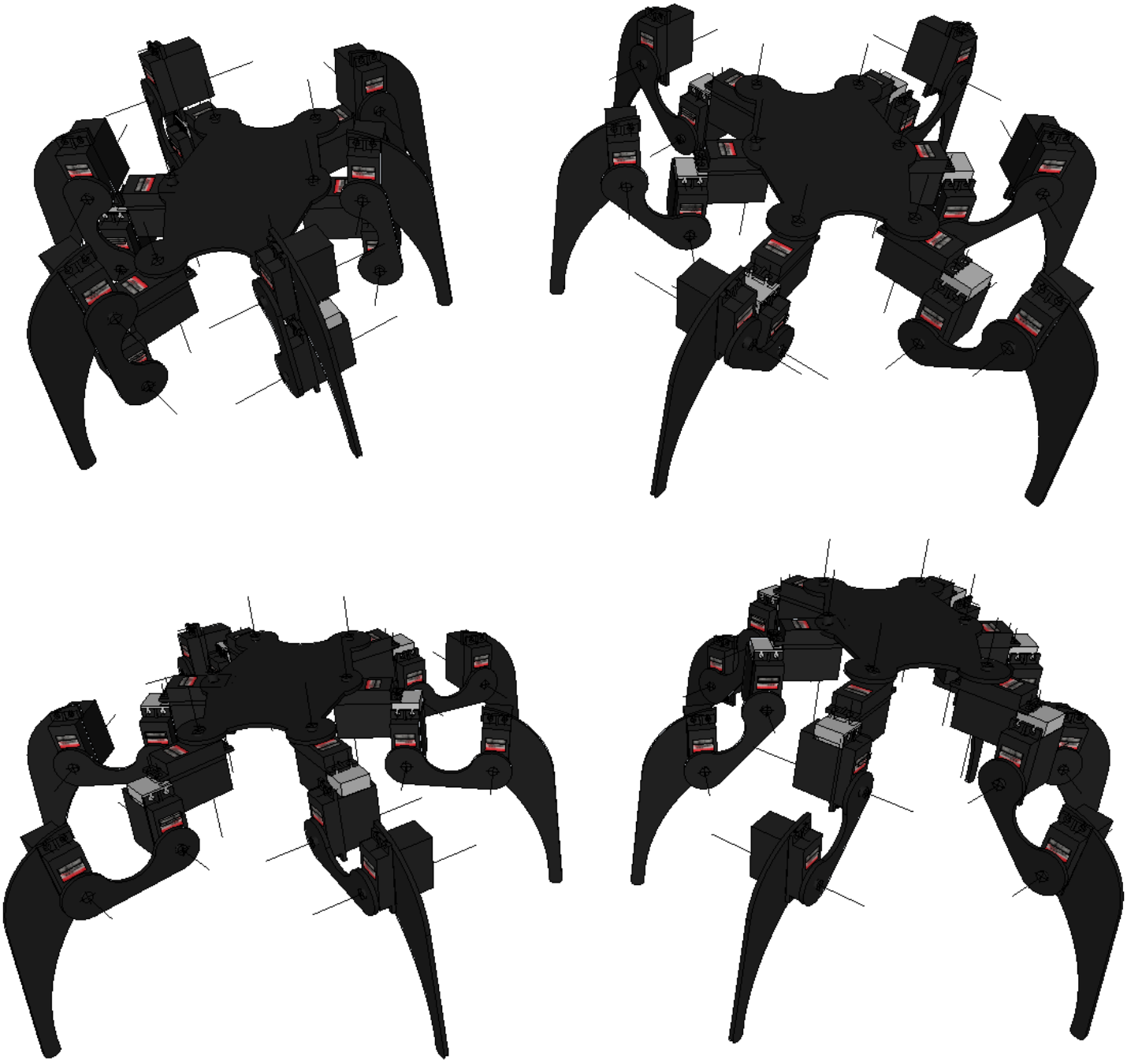

Figura 4.8 Posiciones de parado

4.6 LOCOMOCIÓN

Existen dos tipos básico de locomoción para robots hexápodos: movimiento cuadrúpedo y movimiento trípode.

4.6.1 Movimiento trípode

En el ciclo de movimiento trípode el robot siempre mantiene tres patas en el suelo. Estas se mueven hacia atrás para impulsar el robot hacia adelante, mientras que las que están al aire buscan la posición adecuada hacia adelante para después poder impulsar al robot.

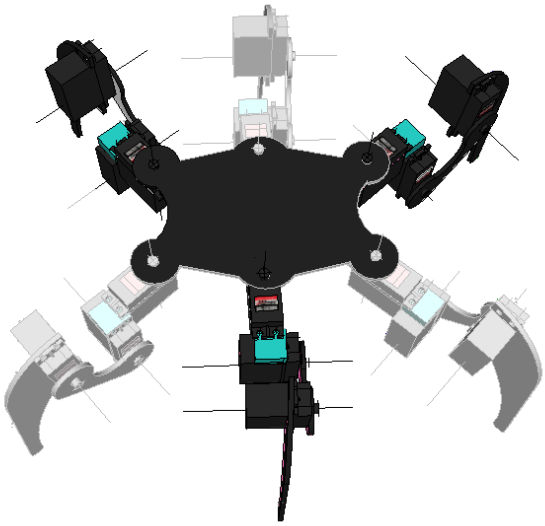

Figura 4.9 Movimiento trípode

Para programar los distintos algoritmos de caminar y girar se enumeraron cada extremidad del hexápodo como se observa en la figura 5.

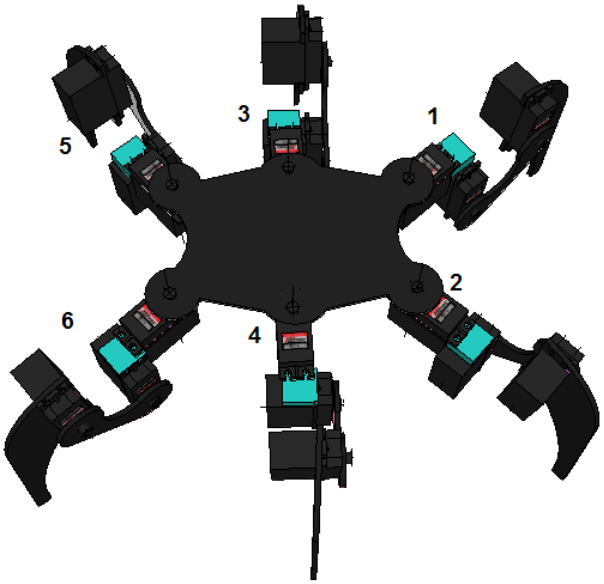

Figura 4.10 Posición de patas

4.6.1.1 Algoritmo caminata hacia adelante trípode

El algoritmo para caminar hacia adelante en locomoción trípode consta de 7 pasos que se repiten para tener una secuencia de locomoción hacia delante [5]:

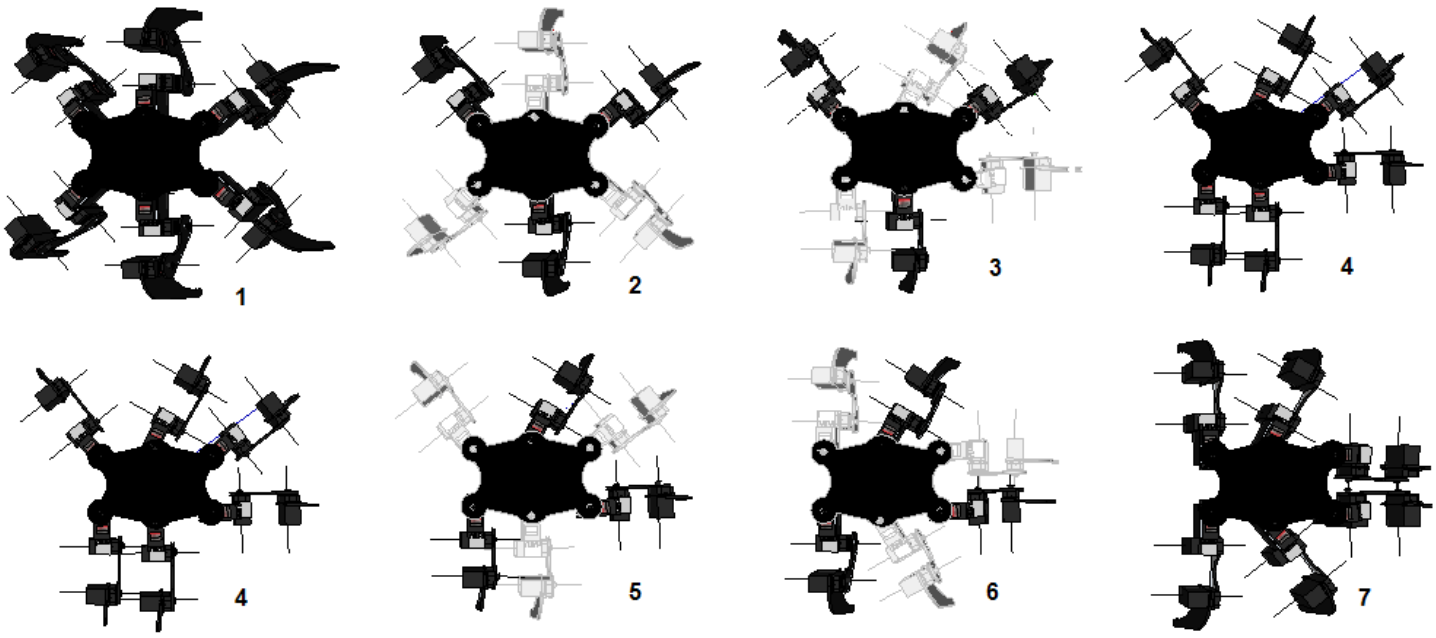

Figura 4.11 Algoritmo caminata trípode adelante

1. Posición de parado.
2. De la posición de parado se levantan la patas 2, 3 y 6.
3. Se mueven hacia adelante las patas 2, 3 y 6 aun levantadas.
4. Se bajan las patas 2, 3 y 6.
5. De la última posición se levantan las patas 1, 4 y 5.
6. Se mueven hacia adelante las patas 1, 4 y 5 aun levantadas.
7. Se bajan las patas 1, 4 y 5.

El siguiente paso sería volver a la posición inicial y realizar el algoritmo nuevamente.

4.6.2 Movimiento cuadrúpedo

El movimiento cuadrúpedo mantiene cuatro patas en el suelo mientas solo desplaza dos hacia adelante para poder impulsar al robot hacia adelante o atrás.

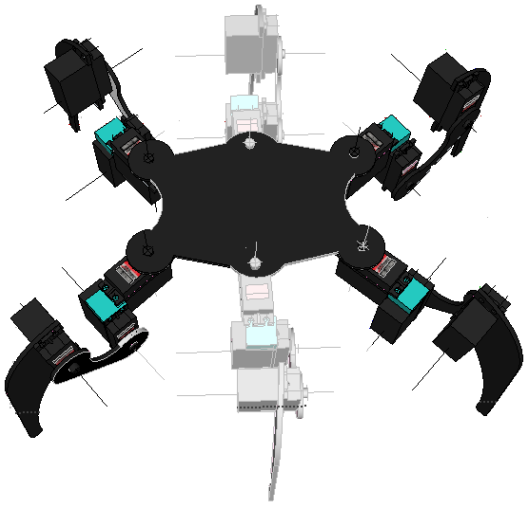

Figura 4.12 Movimiento cuadrúpedo

4.6.2.1 Algoritmo caminata hacia adelante cuadrúpedo

El algoritmo para caminar hacia adelante en locomoción cuadrúpeda consta de 10 pasos que se repiten para tener una secuencia de locomoción hacia adelante:

1. Posición de parado.
2. De la posición de parado se levantan la patas 3 y 4.
3. Se mueven hacia adelante las patas 3 y 4.
4. Se bajan las patas 3 y 4.
5. De la última posición se levantan las patas 2 y 5.
6. Se mueven hacia adelante las patas 2 y 5 aun levantadas.
7. Se bajan las patas 2 y 5.
8. De la última posición se levantan las patas 1 y 6.
9. Se mueven hacia adelante las patas 1 y 6 aun levantadas.
10. Se bajan las patas 1 y 6

El siguiente paso sería volver a la posición inicial y realizar nuevamente los 10 pasos en un algoritmo repetitivo.

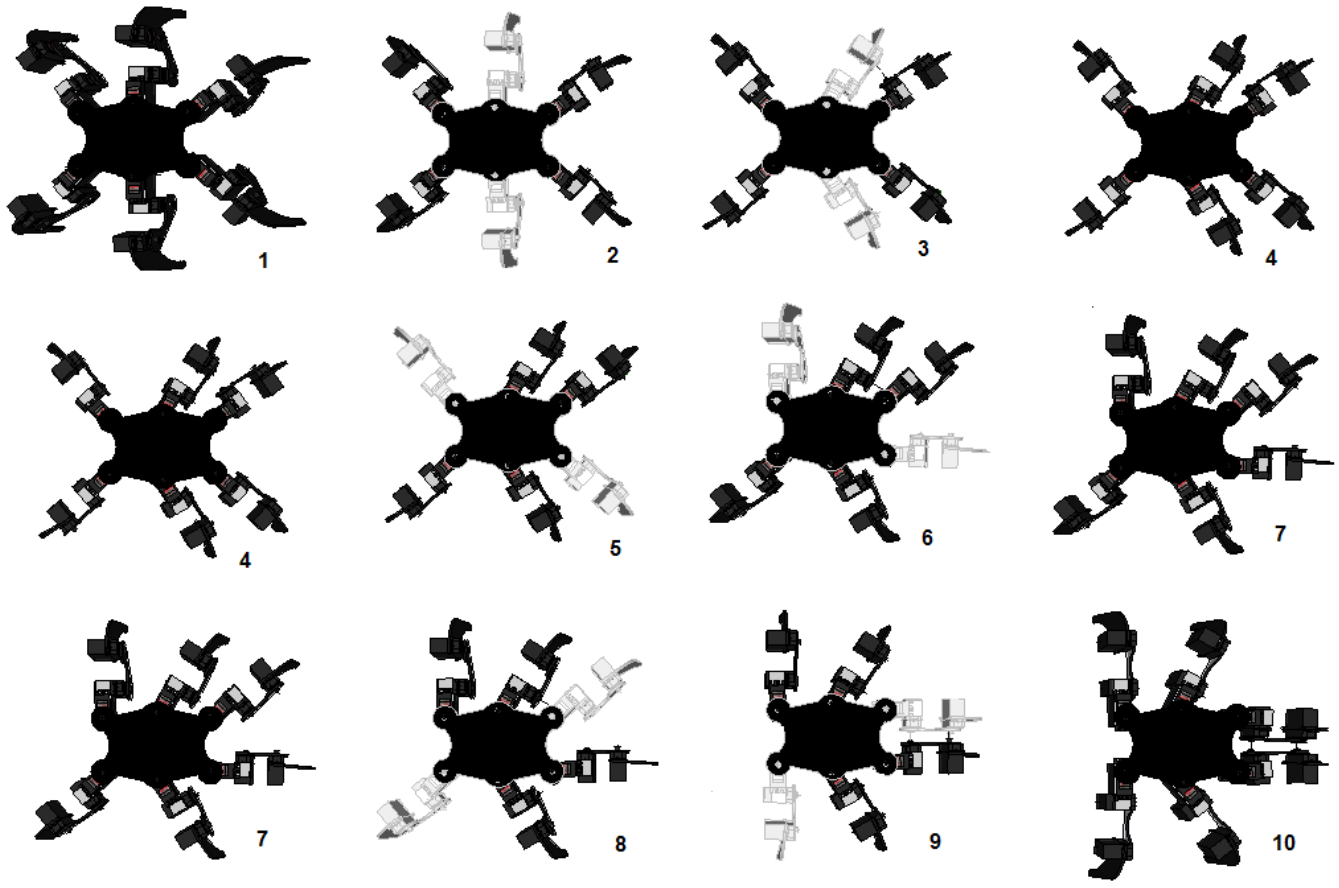

Figura 4.13 Algoritmo caminata cuadrúpedo adelante

El control remoto tiene un botón táctil para desplazar hacia adelante y uno más para desplazar hacia atrás al robot, el algoritmo para desplazarse hacia ambos lados es el mismo solo cambia la dirección. En el desarrollo del proyecto se probaron el modo de desplazamiento cuadrúpedo y el desplazamiento trípode, siendo el desplazamiento trípode más inestable para el robot se decidió por utilizar únicamente el algoritmo de locomoción cuadrúpedo, siendo este más estable aunque este incluye tres movimientos que le quitan tiempo al desplazarse en comparación con la locomoción trípode.

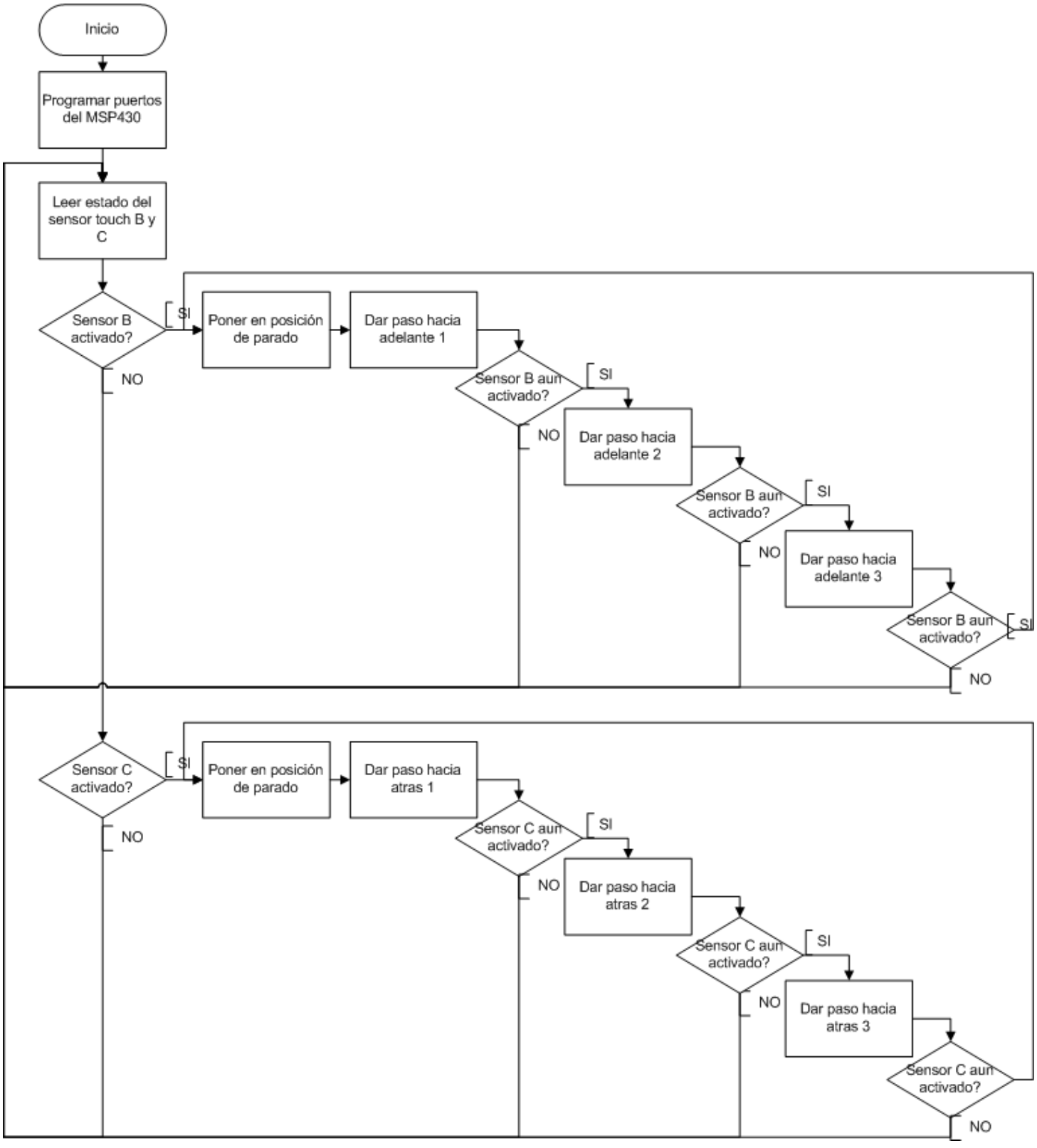

Figura 4.14 Diagrama de flujo de movimiento adelante - atrás

4.6.3 Algoritmo de giro izquierda – derecha

El algoritmo para girar hacia la izquierda y la derecha es muy similar al algoritmo de locomoción cuadrúpedo solo que en este caso una extremidad gira hacia adelante mientras la otra gira hacia atrás, al igual que en la locomoción cuadrúpeda el algoritmo consta de 10 pasos:

1. Posición de parado.
2. De la posición de parado se levantan la patas 3 y 4.
3. Se mueve hacia adelante la patas 4 y al mismo tiempo se mueve hacia atrás la pata 3.
4. Se bajan las patas 3 y 4.
5. De la última posición se levantan las patas 1 y 6.
6. Se mueve hacia adelante la pata 6 y al mismo tiempo se mueve hacia atrás la pata 1.
7. Se bajan las patas 1 y 6.
8. De la última posición se levantan las patas 2 y 5.
9. Se mueve hacia adelante la pata 2 y al mismo tiempo se mueve hacia atrás la pata 5.
10. Se bajan las patas 2 y 5.

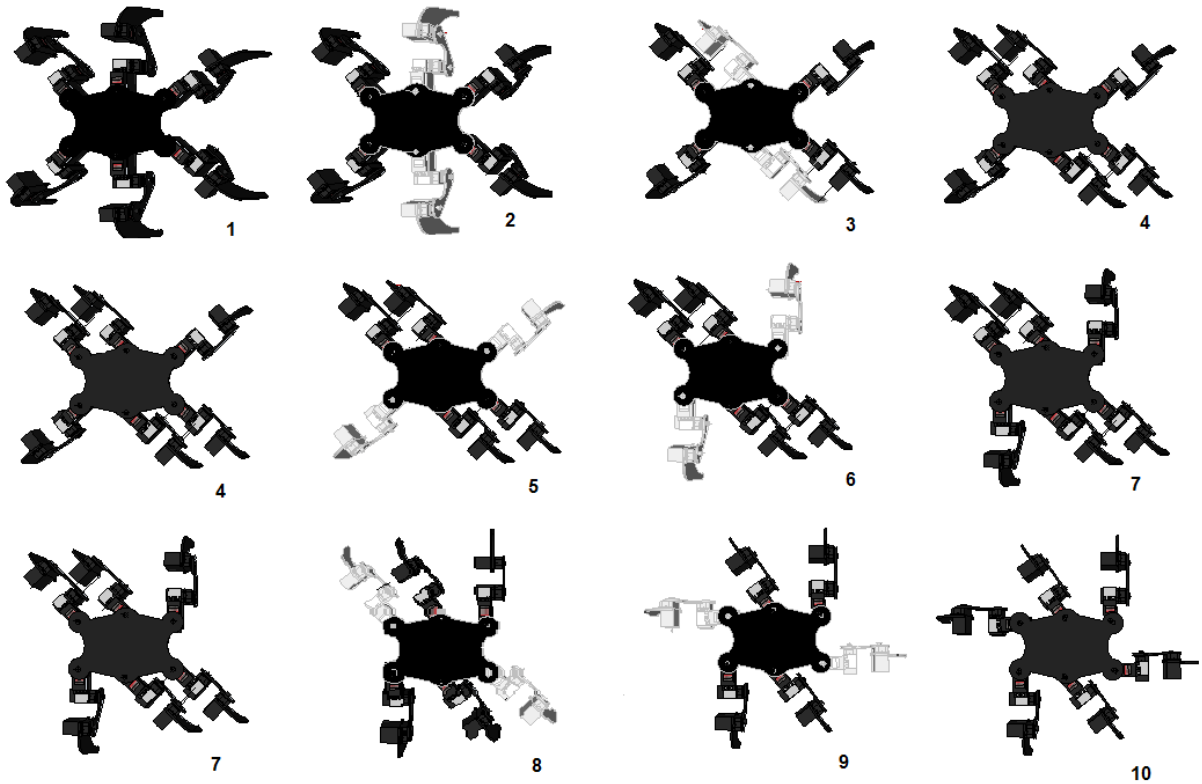

Figura 4.15 Algoritmo de giro izquierda – derecha

Al igual que en la locomoción delantera – trasera, el control remoto tiene un botón táctil para girar hacia la izquierda y uno más para girar hacia la derecha, el algoritmo para desplazarse hacia ambos lados es el mismo, cambiando solo la dirección. Al igual que en la locomoción se programo el algoritmo de locomoción cuadrúpeda debido a que es el que tiene una mayor estabilidad al momento de girar.

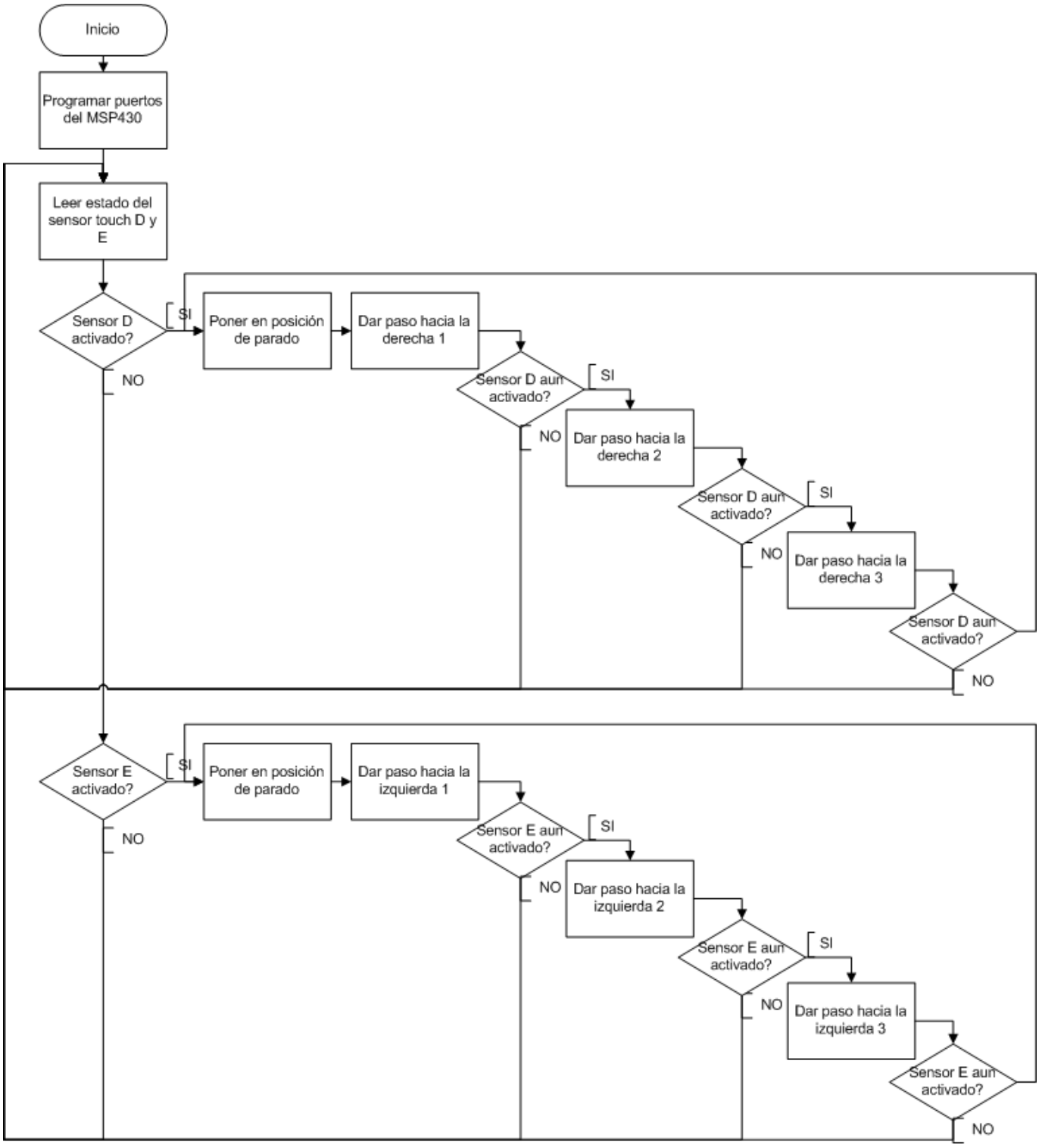

Figura 4.16 Diagrama de flujo de giro izquierda – derecha

CAPÍTULO 5

PRUEBAS Y RESULTADOS

5.1 INTRODUCCIÓN

En este capítulo se detalla las pruebas y resultados obtenidos tras la fabricación e implementación de nuestro hexápodo.

5.2 CONTROL REMOTO

El comportamiento de la interfaz táctil es buena y cumple con los objetivos de hacer funcionar los cinco botones táctiles así como interactuar con el módulo inalámbrico para poder transmitir la posición de los botones.

El módulo inalámbrico CC110L funciona con la configuración antes hecha teniendo una buena recepción por parte del receptor, teniendo un alcance de aproximadamente 7 metros y funcionando a una frecuencia de 868 MHz

La duración de la batería en un uso continuo es de aproximadamente 2 horas.

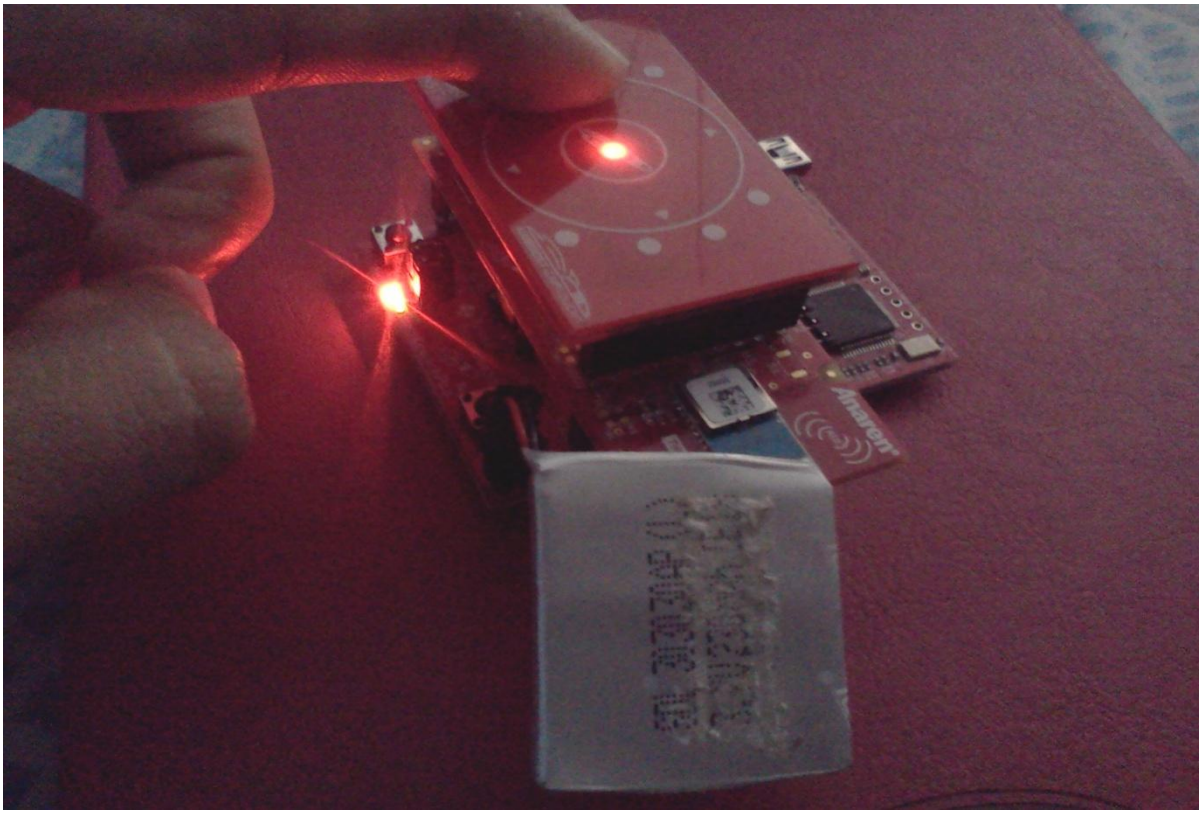

Figura 5.1 Control remoto

5.3 CAMINATA ADELANTE – ATRÁS

La locomoción del robot en la caminata sobre superficies planas es aceptable tomando en cuenta la estabilidad y fluidez de la locomoción.

La velocidad de desplazamiento es lenta debido al tipo de locomoción cuadrúpeda la cual contempla tres pasos para que el robot se desplace aproximadamente 10 centímetros y vuelva a su posición inicial. Sin embargo esto le da mayor estabilidad al momento de desplazarse.

En las figuras 5.2, 5.3 y 5.4 se muestra al robot realizando su caminata en una superficie lisa vista desde tres diferentes posiciones.

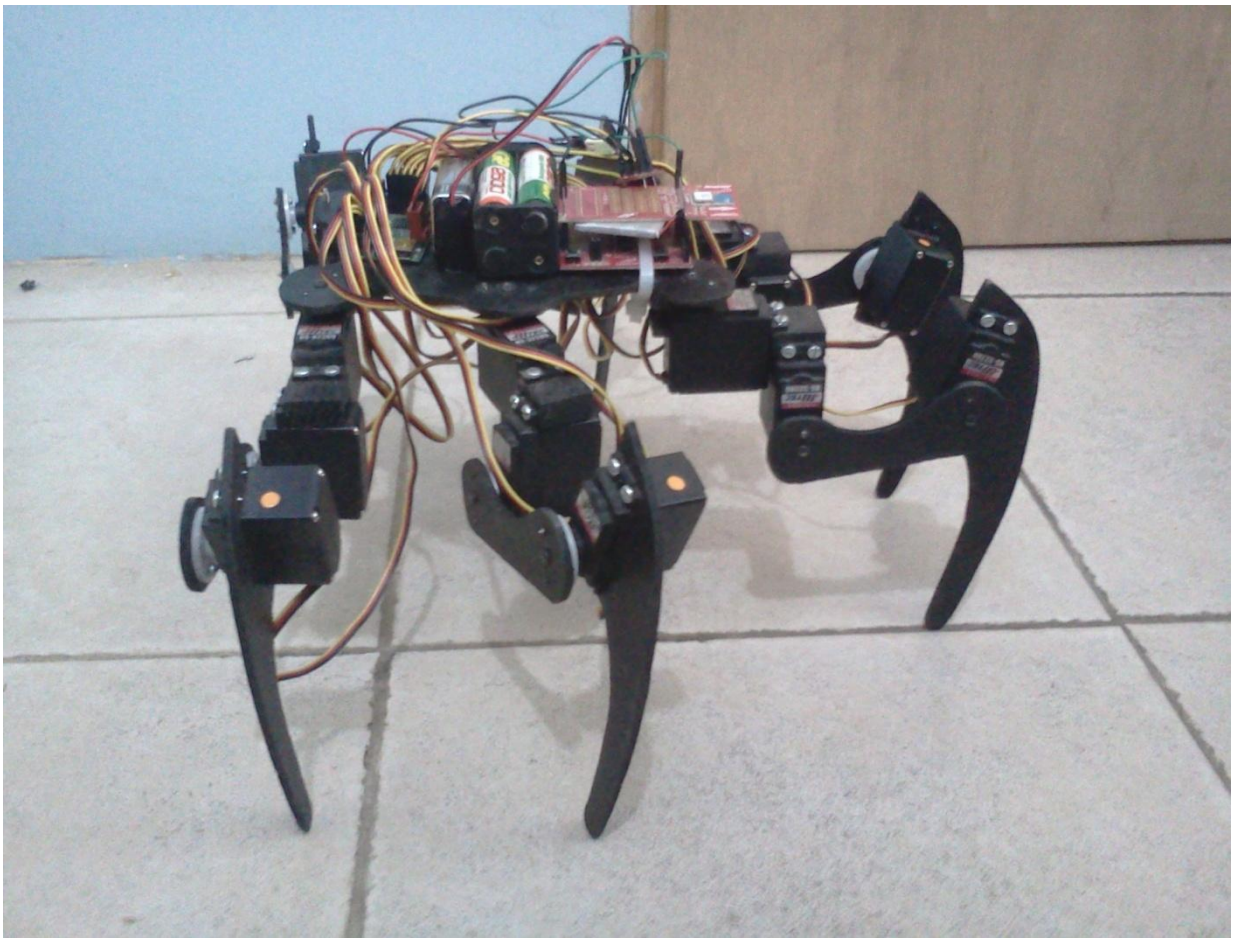

Figura 5.2 Locomoción delantera vista lateral

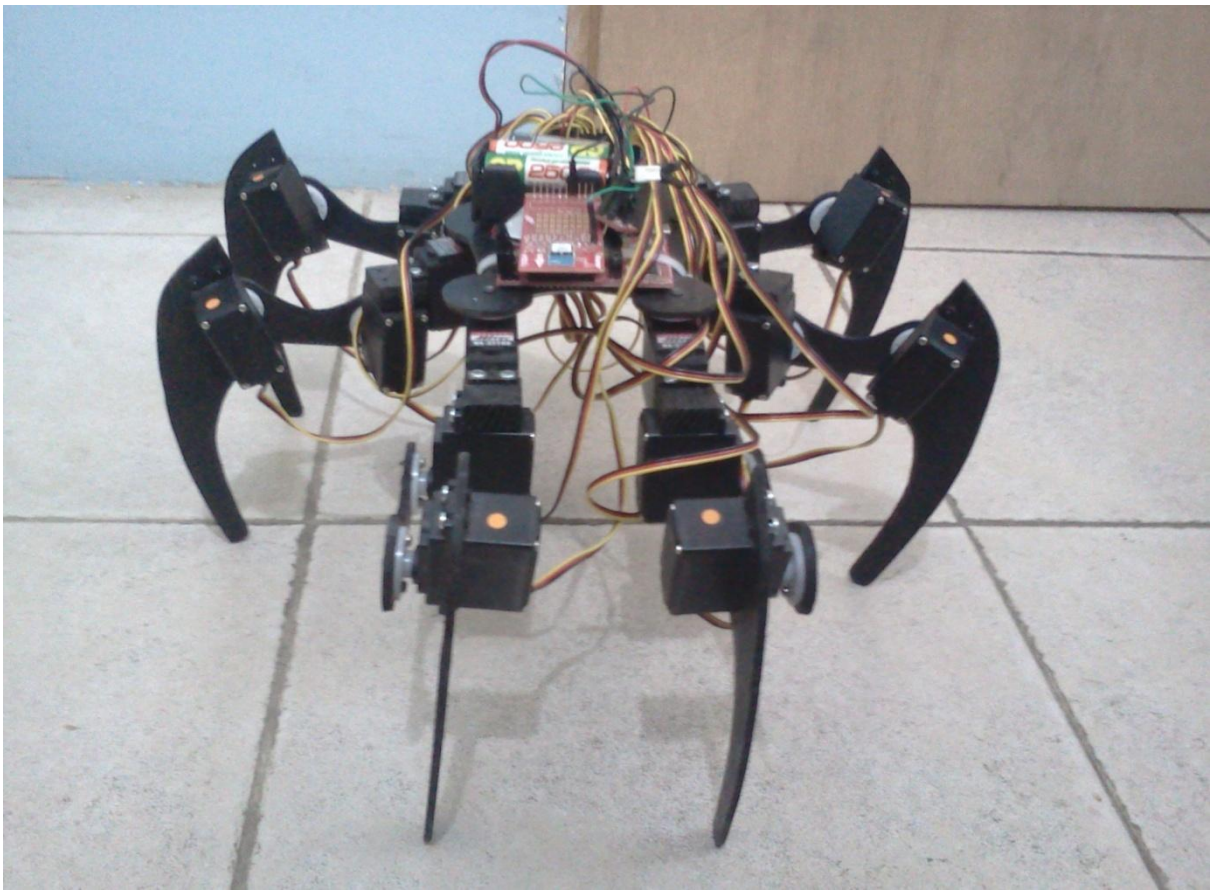

Figura 5.3 Locomoción delantera vista frontal

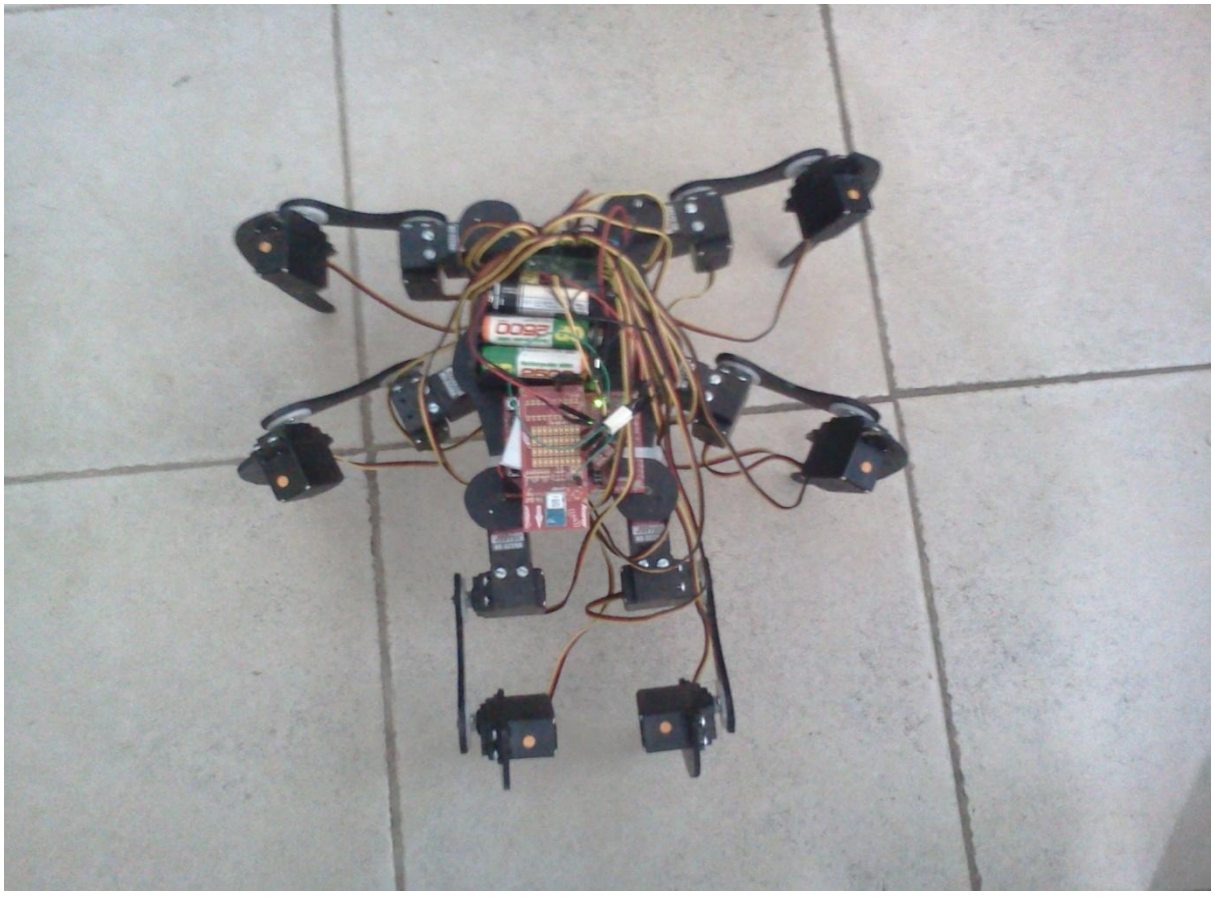

Figura 5.4 Locomoción delantera vista superior

5. 4 POSICIONES DE PARADO

Las cuatro posiciones de parado tiene un comportamiento bastante estable y aceptable, el único inconveniente destacable es que en superficies muy lisas el robot ocasionalmente tiende a resbalar debido al material de las patas y a que los servomotores soportan muy poco peso.

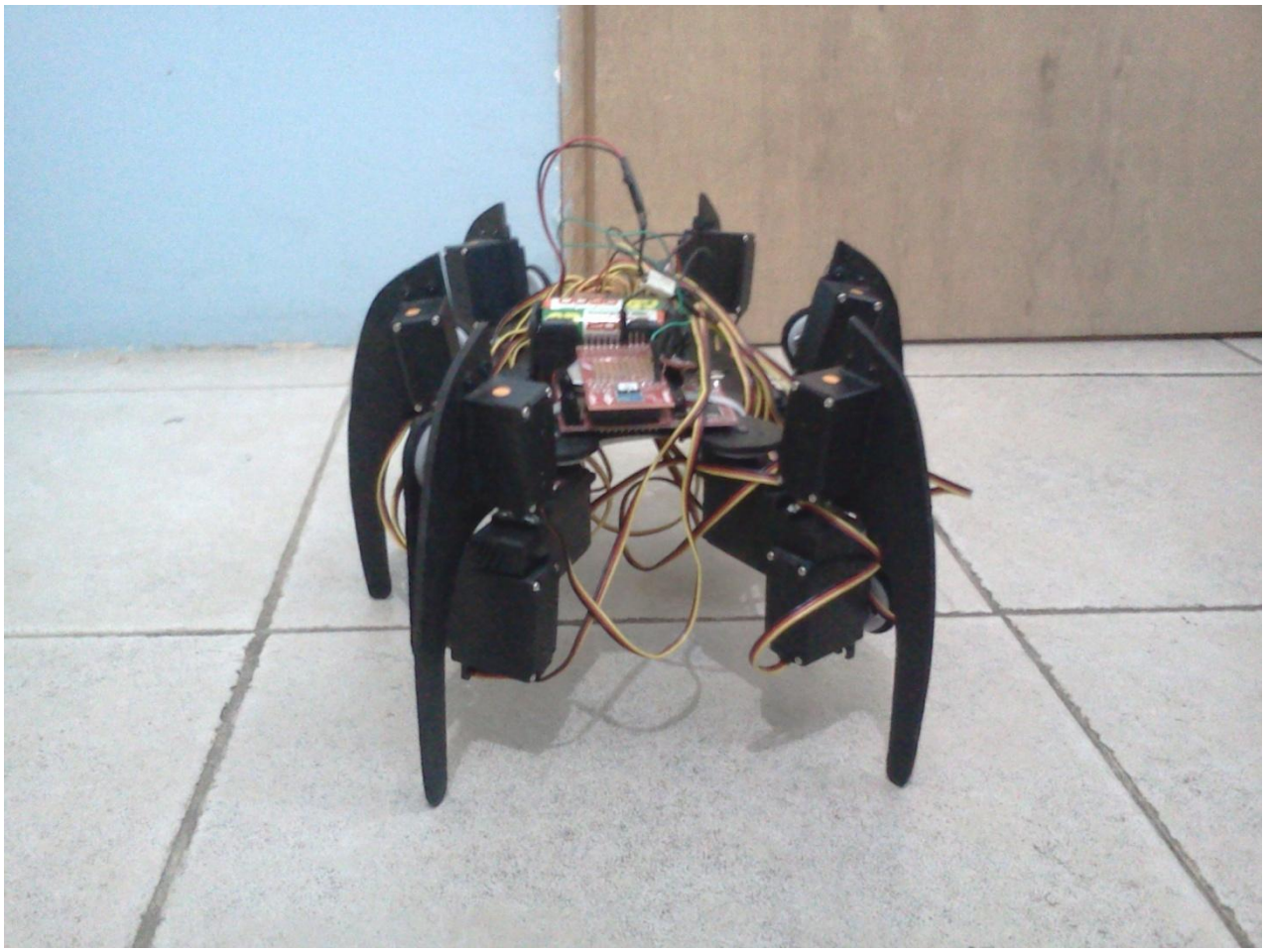

Figura 5.5 Posición de reposo

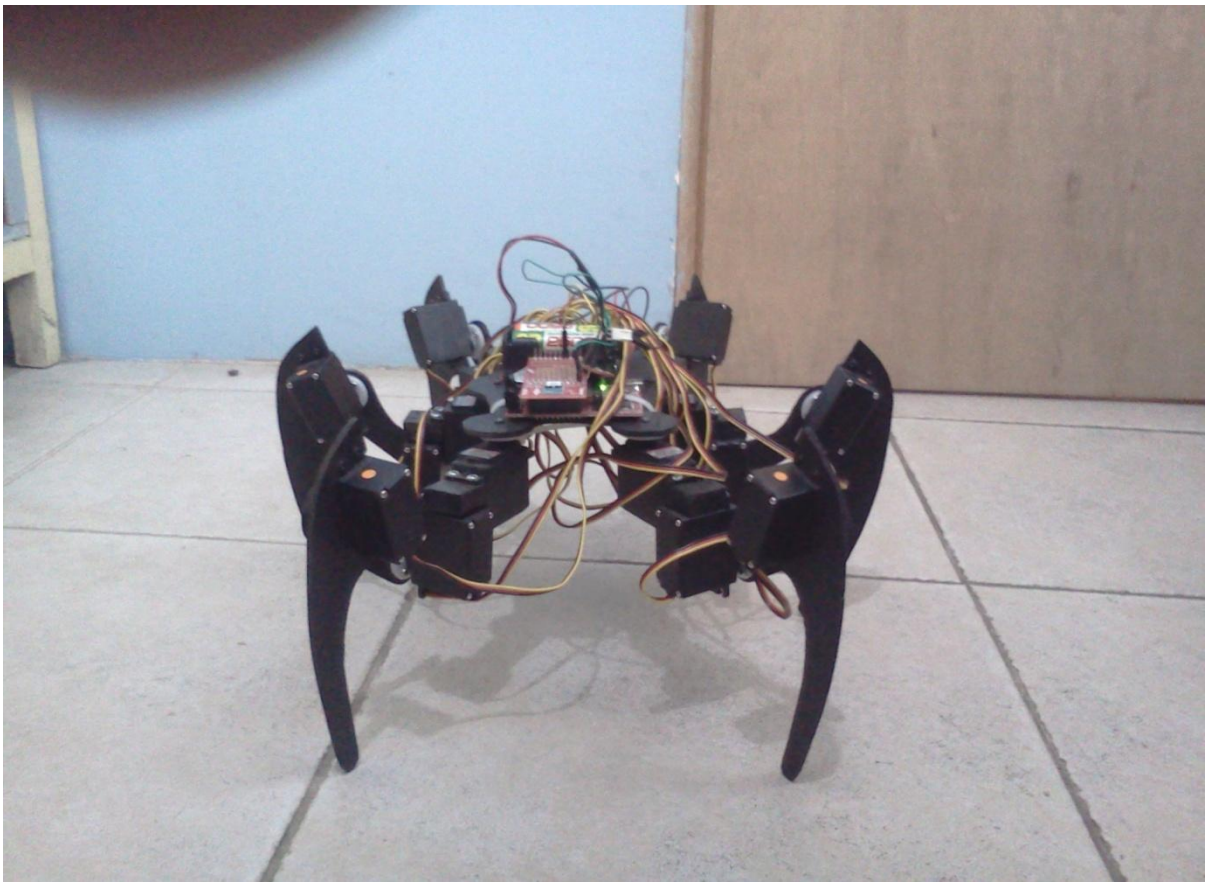

Figura 5.6 Posición parado bajo

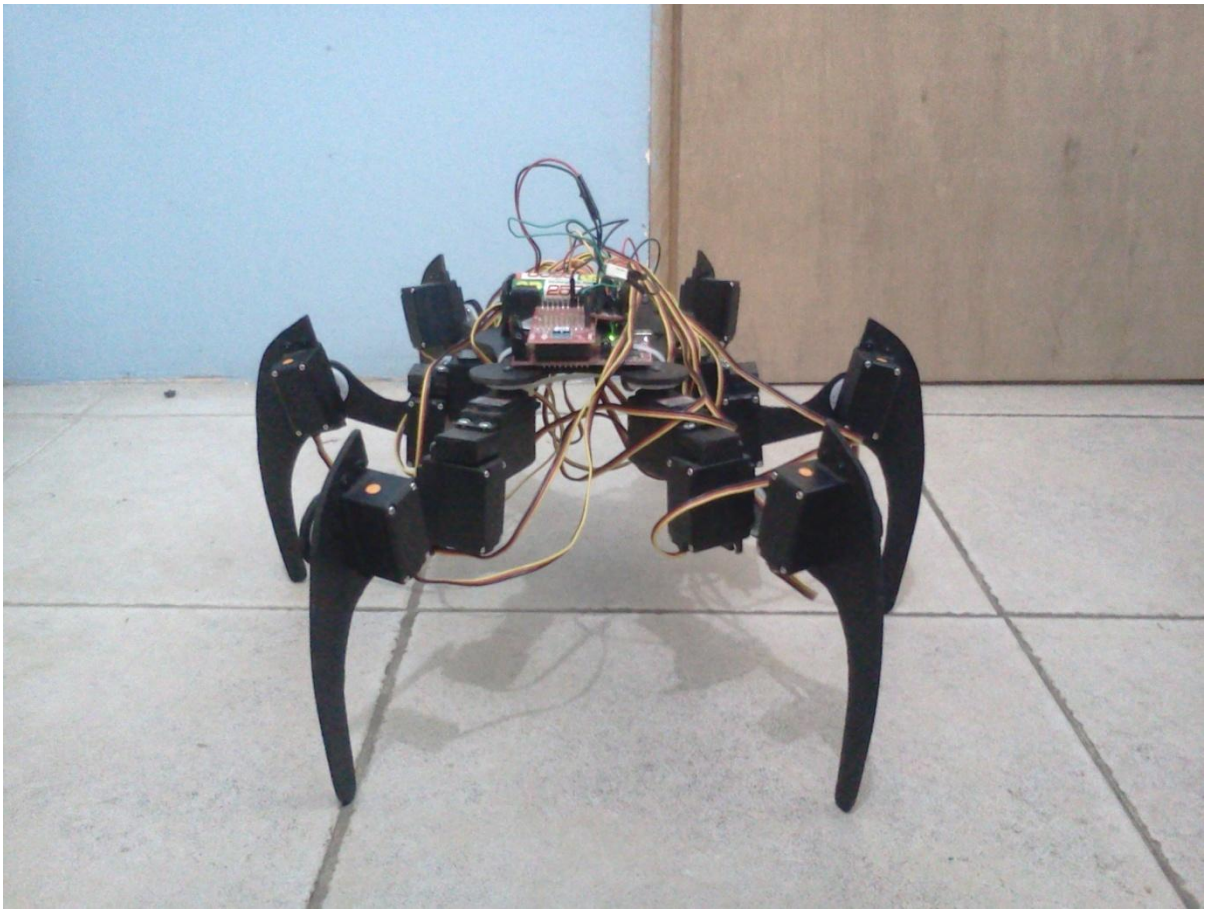

Figura 5.7 Posición parado

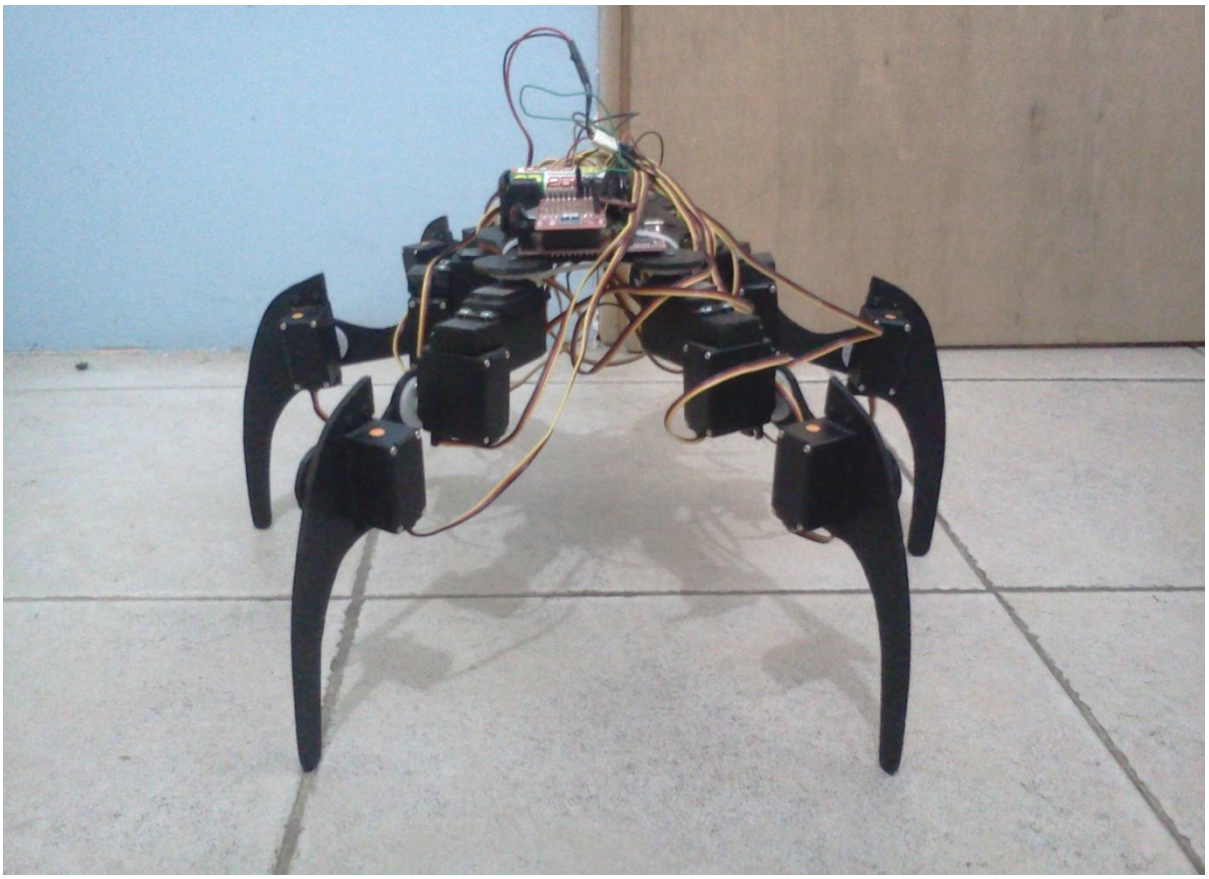

Figura 5.8 Posición parado alto

5. 5 GIRO IZQUIERDA - DERECHA

El algoritmo programado para girar a la izquierda y a la derecha es bastante bueno considerando que se controla de manera remota, ya que el robot no sufre de inestabilidad y lo hace con una locomoción bastante fluida.

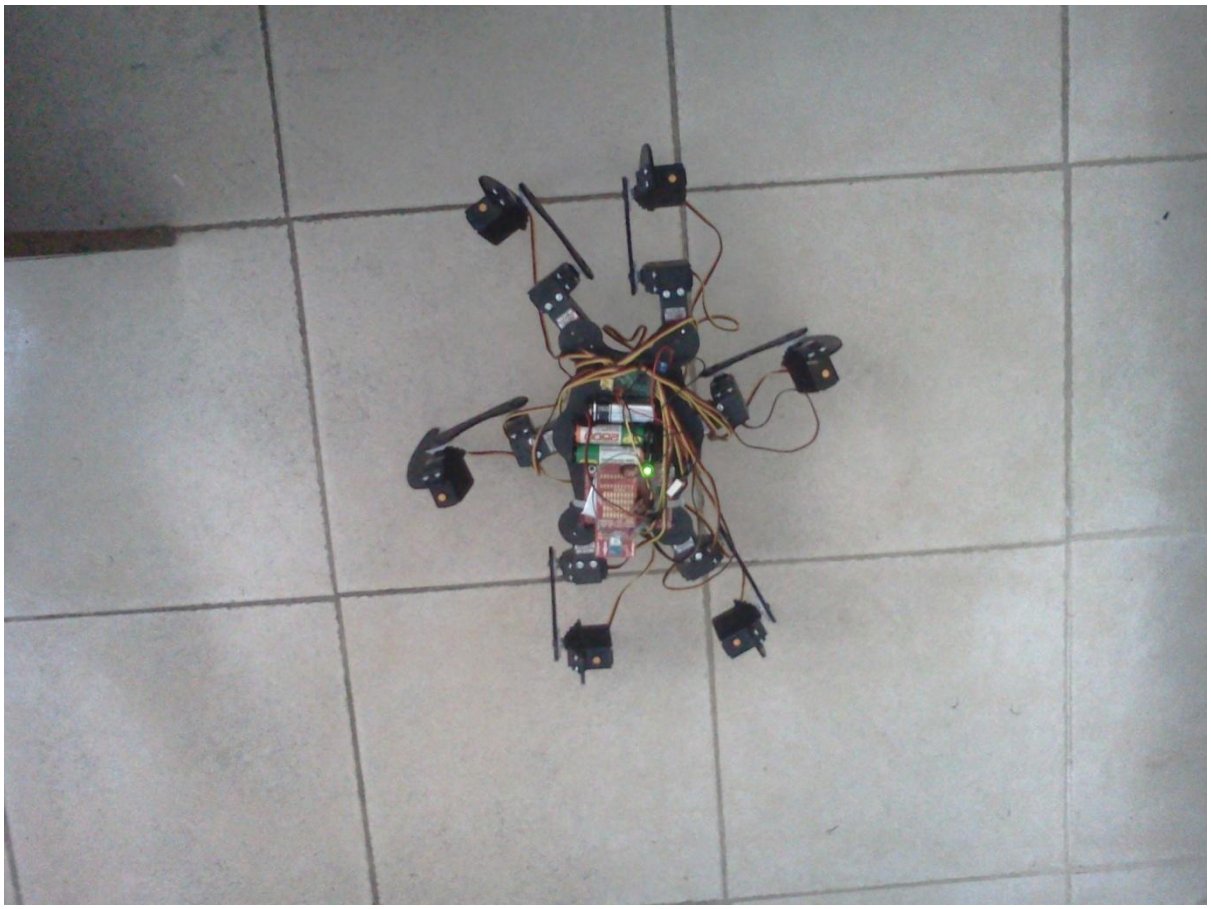

Figura 5.9 Girando a la izquierda paso 1

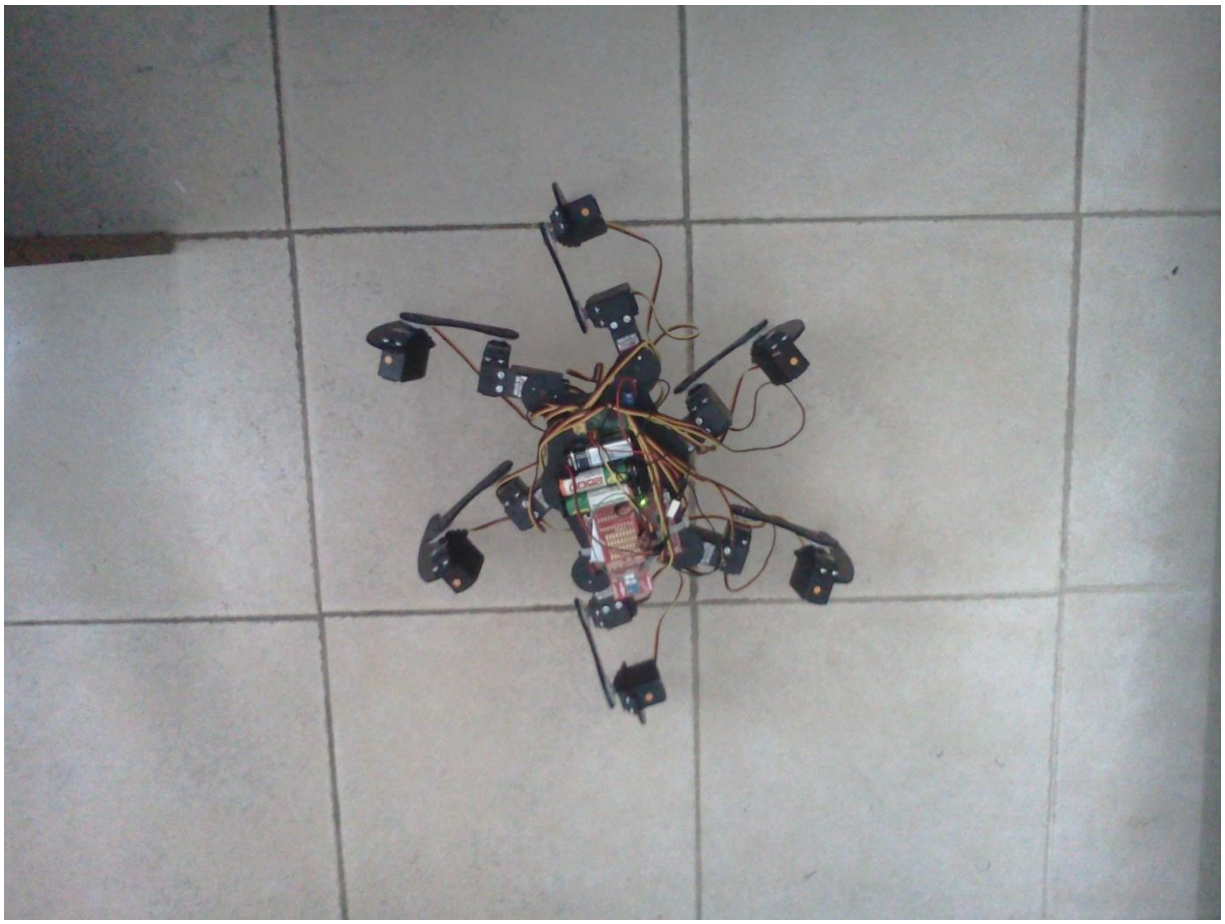

Figura 5.10 Girando a la izquierda paso 2

CONCLUSIONES

CONCLUSIONES

Se logró diseñar, construir y programar un robot tipo hexápodo con dieciocho grados de libertad controlado de manera inalámbrica por medio de un control remoto con interfaz táctil. Cabe mencionar que todo se hizo con módulos ensamblados y no fue requerido componente externo alguno, el rango de alcance del control remoto es de aproximadamente 7 metros y funciona a una frecuencia de 868 MHz

La programación del robot se hizo utilizando el compilador *Code composer studio v4.0* y se programo en lenguaje C, para configurar y programar los diversos módulos se hizo necesario el apoyo de las diferentes bibliotecas que facilitaron y redujeron la cantidad de código. La programación de cada movimiento del hexápodo fue hecha de manera secuencial tomando como base una primera posición y a partir de esta posición se fueron programando las posiciones subsecuentes, cabe mencionar que no se hizo uso de simulador alguno por lo que no se simulo ningún algoritmo o secuencia de movimientos lo cual le resta estabilidad al dispositivo pero aun así lo hace lo suficientemente funcional para ser controlado de manera remota.

Como se mencionó en su momento el robot no contiene ningún sensor que le de retroalimentación al control y que lo pueda hacer más estable, ya que esto implicaba realizar un análisis de cinemática inversa para poder conocer el comportamiento mecánico de cada pieza y poder establecer un modelo matemático que mejorará la estabilidad del sistema, sin embargo esto fue poco viable ya que implicaba un gasto mayor en sensores y servomotores, además hubiese sido necesario utilizar otro microcontrolador con mayores características (software y hardware) en comparación con el MSP430G2553 para así poder programar el modelo matemático que estabilizará al sistema.

Para el diseño del dispositivo se utilizo principalmente el software S*ketchup* el cual es un programa para desarrollo de imágenes CAD 3D, y a partir de este se diseñaron cada una de las partes implementadas en el cuerpo del robot. Por otra parte para la construcción se utilizo madera tipo MDF de 3 mm y después se hicieron los cortes con caladora cabe mencionar que se buscaron alternativas como el aluminio o el acrílico pero resulto mas económico y fácil construirlo con madera comprimida y después pintarlo con pintura acrílica.

Uno de los puntos débiles en el trabajo son las baterías que controlan los servomotores ya que todos los servomotores funcionando en conjunto llegan a tener un consumo pico de 3 amperes lo cual acaba demasiado rápido con la carga de las pilas. Otro problema importante en el desarrollo de este trabajo fueron los servomotores que utilizamos, ya que estos servomotores tienen un torque reducido y aunado al problema de las baterías hacen tener problemas de inestabilidad a nuestro dispositivo. Todos estos problemas son solucionados fácilmente invirtiendo en baterías que soporten una mayor carga de corriente y en servomotores que tengan un mayor torque, sin embargo hacer este tipo de modificaciones implicaba una inversión muy superior que no era costeable.

TRABAJO FUTURO

- Realizar un análisis dinámico y cinemático inverso, para poder darle un modelo matemático al robot que le de retroalimentación y por lo tanto estabilidad.

- Realizar un análisis estructural de las piezas diseñadas y de ser necesario ocupar otro material que elimine posibles deformaciones o vibraciones mecánicas al sistema.

- Adaptarle al sistema sensores de posición, acelerómetro, infrarrojos, ultrasónico.

- Utilizar un microcontrolador con mayores características en software y hardware que sea capaz de procesar la información del análisis de cinemática inversa y de los sensores.

- Utilizar pilas de polímero de litio (LIPO), que soportan un consumo mayor de corriente eléctrica y son más delgadas en comparación con las pilas de níquel-hidruro metálico.

- Equipar al dispositivo con servomotores con un mayor torque, que le den mayor movilidad al sistema.

APÉNDICE A

CÓDIGO TRANSMISOR

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++
//CODIGO TRANSMISOR
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++
#include "include.h"
#include "CTS_Layer.h"                          //Libreria modulo tactil
#define DELAY 50                            //Tiempo de retraso  (50*0.1msec = 5
milisegundos)
struct Element * keyPressed;                // Puntero a la estrutura Element

#ifdef ELEMENT_CHARACTERIZATION_MODE      //Inclusion condicional
unsigned int wheelCnt[5];
#endif                                                      //Fin de condicional



//Configura el Timer A para trabajar apagado en ACLK, cuenta en modo acendente,
//pone el procesador en modo de ahorro de energia, ademas habilita el vector de
interrupcion para entrar en ISR
void sleep(unsigned int time)              //Declaracion funcion dormir
{
    TA0CCR0 = time;                       //Declara vector de interrupcion en modo
comparacion
    TA0CTL = TASSEL_1+MC_1+TACLR;         //Declara ACLK, modo ascendente, bit
TACLR siempre es leido como cero
    TA0CCTL0 &= ~CCIFG;                         //espera el bit de la bandera de
interrupcion en modo de comparacion
    TA0CCTL0 |= CCIE;                           //espera el bit de la bandera de
interrupcion
    __bis_SR_register(LPM3_bits+GIE);     //Cuando termina la interrupcion entra
en modo de ahorro LPM3
    __no_operation();
}


//char rxBuffer[CC110L_BUFFER_LENGTH];



//Declaracion de Variables
int d;
int conta = 0;
char txBuffer[12];
char rxBuffer[12];
unsigned int i = 0;
unsigned char TxByte=0;
unsigned int result;
unsigned int del;
extern char paTable[];
extern char paTableLen;

//Declaracion de FUNCIONES
void putr(char c);
void in(unsigned int servo, unsigned int angle);
unsigned posinicial(void);
unsigned paradobajo(void);
unsigned parado (void);
unsigned paradoalto (void);
unsigned caminadelante (void);
unsigned camina (void);
unsigned caminadelante1(void);
unsigned caminadelante2(void);
```

```
//Funciones de caminata ADELANTE
unsigned delante(void);
unsigned delantea(void);
unsigned delanteb(void);
unsigned delantec(void);
unsigned delanted(void);
unsigned delantee(void);

//Funciones de caminata ATRAS
unsigned atras(void);
unsigned atrasa(void);
unsigned atrasb(void);
unsigned atrasc(void);

//Funciones de giro DERECHA
unsigned derechaa(void);
unsigned derechab(void);
unsigned derechac(void);
unsigned derecha(void);

//Funciones de giro IZQUIERDA
unsigned izquierda (void);
unsigned izquierdaa (void);
unsigned izquierdab (void);
unsigned izquierdac (void);

//Funcion de enviar byte por el UART
void envial0(char c);

void main (void)
{
WDTCTL = WDTPW + WDTHOLD;               // Detiene WDT

//Configuracion de registros del reloj MODULO TACTIL
  BCSCTL1 = CALBC1_1MHZ;                // Se fija el registro de control del
reloj a 1 MHz
  DCOCTL = CALDCO_1MHZ;                       // Se fija el registro de control
del DCO fijo a 1 MHz
  BCSCTL2 |= DIVS_2;                    // Divide SMCLK entre 4
  BCSCTL3 |= LFXT1S_2;                  // Fija LFXT1 = VLO

//Configuracion del UART
  P1SEL = BIT1 + BIT2 ;                 // Fija bits UART P1.1 = RXD, P1.2=TXD
  P1SEL2 = BIT1 + BIT2 ;                // Fija bits UART P1.1 = RXD, P1.2=TXD
  P2SEL &= ~(BIT6 + BIT7);
  UCA0CTL1 |= UCSSEL_2;                 // SMCLK
  //UCA0BR0 = 104;                            // 1MHz 9600
  //UCA0BR1 = 0;                              // 1MHz 9600
  UCA0BR0 = 52;                         // 1MHz 9600
  UCA0BR1 = 0;                          // 1MHz 9600
  UCA0MCTL = UCBRS0;                    // Modulacion
  UCA0CTL1 &= ~UCSWRST;                 // Inicia USCI
  //IE2 |= UCA0RXIE;//+UCA0TXIE;        // Habilita USCI_A0 RX interrupt

//Configuracion del transmisor CC110L
//Contiene funciones que permiten al MSP430 accesar a la interfaz del CC110L
(funciones de la biblioteca CC110L)
  TI_CC_SPISetup();                     // Inicializa puertos SPI
  TI_CC_PowerupResetCCxxxx();           // Reset al CC110L
  writeRFSettings();                    // Write RF settings to config reg
  TI_CC_SPIWriteBurstReg(TI_CCxxx0_PATABLE, paTable, paTableLen);//Write PATABLE

  TI_CAPT_Init_Baseline(&wheel_buttons);
  TI_CAPT_Update_Baseline(&wheel_buttons,5);
```

```
  TI_CC_LED_PxOUT &= ~(TI_CC_LED1);       // Fija Bit 0 como dato de salida (led
mando tactil)
  TI_CC_LED_PxDIR |= TI_CC_LED1;          // Fija direccion de salida de led Bit
0
  TI_CC_GDO0_PxIES |= TI_CC_GDO0_PIN;    // Fija Interrupcion en flanco de bajada
(end of pkt)
  TI_CC_GDO0_PxIFG &= ~TI_CC_GDO0_PIN;  // Limpia bandera
  TI_CC_GDO0_PxIE |= TI_CC_GDO0_PIN;     // Interrupcion habilitada
  TI_CC_SPIStrobe(TI_CCxxx0_SRX);        // Inicializa CC110L en modo RX
   __bis_SR_register( GIE);               // GIE=Interrupcion general habilitada

while (1)
  {
      #ifdef ELEMENT_CHARACTERIZATION_MODE
      // Get the raw delta counts for element characterization
      TI_CAPT_Custom(&wheel_buttons,wheelCnt);
      __no_operation();                               // Set breakpoint here
      #endif

      #ifndef ELEMENT_CHARACTERIZATION_MODE
      keyPressed = (struct Element *)TI_CAPT_Buttons(&wheel_buttons);
      if(keyPressed)
        {

            //BOTON HACIA ABAJO
            if(keyPressed == &down_element)     //Si el boton de abajo a sido
presionado:
            {
                        conta++;          //Inicia contador
                        P1OUT |= BIT0;    //Enciende led modulo tactil cada vez
que se preciona el boton
                        switch(conta) {
                        case 1:           //Boton presionado una vez:
                        default:
                        conta=1;
                        parado();         //Se va a posicion de parado
                       __delay_cycles(500000); //retraso de tiempo
                                  break;

                                  case 2:                 //Boton presionado
segunda vez:
                        delantea();       //Se va a posicion de delantea(DA
PRIMER PASO)
                        break;

                        case 3:                 //Boton presionado tercera vez:
                        delanteb();     //Se va a posicion de delanteb(DA
SEGUNDO PASO)
                        break;

                        case 4:                 //Boton presionado cuarta vez:
                        delantec();       //Se va a posicion de delantec(DA
TERCER PASO)
                        break;

                        //Se repite el algoritmo ahora con las piernas del lado
contrario
                        case 5:                 //Boton presionado quinta vez:
                        parado();         //Se va a posicion de parado
                  __delay_cycles(500000); //retraso de tiempo
                        break;

                        case 6:                 //Boton presionado sexta vez:
```

```
                        delantea();       //Se va a posicion de delantea(DA
PRIMER PASO)
                        break;

                        case 7:                 //Boton presionado septima vez:
                        delanted();       //Se va a posicion de delanted(DA
SEGUNDO PASO)
                        break;

                        case 8:                 //Boton presionado octava vez:
                        delantee();       //Se va a posicion de delantee(DA
TERCER PASO)
                        break;                  //Sale del contador y se va a
default (reinicia  ciclo)
                        }

            }
            //BOTON HACIA ARRIBA
            if(keyPressed == &up_element) //Si el boton de arriba a sido
presionado:
                {
                        conta++;          //Inicia contador
                        P1OUT |= BIT0;    //Enciende led modulo tactil cada vez
que se preciona el boton
                        switch(conta) {

                        case 1:           //Boton presionado primera vez:
                        default:
                        conta=1;
                        parado();         //Se va a posicion de parado
                    __delay_cycles(500000); //retraso de tiempo
                                    break;

                                    case 2:                 //Boton presionado
segunda vez:
                        atrasa();         //Se va a posicion de atrasa(DA PRIMER
PASO ATRAS)
                        break;

                        case 3:                 //Boton presionado tercera vez:
                        atrasb();       //Se va a posicion de atrasb(DA SEGUNDO
PASO ATRAS)
                        break;

                        case 4:                 //Boton presionado cuarta vez:
                        atrasc();         //Se va a posicion de atrasb(DA TERCER
PASO ATRAS)
                        break;                  //Sale del contador y se va a
default (reinicia  ciclo)
                         }

            }
            //BOTON HACIA IZQUIERDA
            if(keyPressed == &left_element)     //Si el boton de la izquierda a
sido presionado:
            {
                                    conta++;          //Inicia contador
                                    P1OUT |= BIT0;    //Enciende led modulo
tactil cada vez que se preciona el boton
                                    switch(conta) {

                        case 1:           //Boton presionado primera vez:
                        default:
                        conta=1;
```

```
                        parado();         //Se va a posicion de parado
                  __delay_cycles(500000); //retraso de tiempo
                                    break;

                                    case 2:                 //Boton presionado
segunda vez:
                        izquierdaa();     //Se va a posicion de izquierdaa(DA
PRIMER PASO A LA IZQUIERDA)
                        break;

                        case 3:                 //Boton presionado tercera vez:
                        izquierdab();   //Se va a posicion de izquierdab(DA
SEGUNDO PASO A LA IZQUIERDA)
                        break;

                        case 4:                 //Boton presionado cuarta vez:
                        izquierdac();     //Se va a posicion de izquierdac(DA
TERCER PASO A LA IZQUIERDA)
                        break;                  //Sale del contador y se va a
default (reinicia  ciclo)
                         }
                                          P1OUT |= BIT0;
            }
            //BOTON DERECHA
            if(keyPressed == &right_element)//Si el boton de la izquierda a sido
presionado:
            {
                              conta++;          //Inicia contador
                                    P1OUT |= BIT0;    //Enciende led modulo
tactil cada vez que se preciona el boton
                                    switch(conta) {

                        case 1:           //Boton presionado primera vez:
                        default:
                        conta=1;
                        parado();         //Se va a posicion de parado
                  __delay_cycles(500000); //retraso de tiempo
                        break;

                        case 2:                 //Boton presionado segunda vez:
                        derechaa();     //Se va a posicion de derechaa(DA PRIMER
PASO A LA DERECHA)
                        break;

                        case 3:                 //Boton presionado tercera vez:
                        derechab();       //Se va a posicion de derechab(DA
SEGUNDO PASO A LA DERECHA)
                        break;

                        case 4:                 //Boton presionado cuarta vez:
                        derechac();       //Se va a posicion de derechac(DA
TERCER PASO A LA DERECHA)
                        break;
                         }
            }

            //BOTON DEL CENTRO
            if(keyPressed == &middle_element)//Si el boton del centro a sido
presionado:
                {
                        P1OUT |= TI_CC_LED1; //Enciende led modulo tactil cada
vez que se preciona el boton
                        conta++;          //Inicia contador
```

```
                        switch(conta) {

                        case 1:           //Boton presionado primera vez:
                        default:
                        conta=1;
                        posinicial();     //Se va a POSICION INICIAL
                __delay_cycles(300000);  //retraso de tiempo
                        break;

                        case 2:                 //Boton presionado segunda vez:
                        paradobajo();   //Se va a posicion PARADO BAJO
                __delay_cycles(300000);   //retraso de tiempo
                        break;

                        case 3:                 //Boton presionado tercera vez:
                        parado();         //Se va a posicion PARADO
                __delay_cycles(300000);
                        break;

                        case 4:                 //Boton presionado cuarta vez:
                        paradoalto();     //Se va a posicion PARADO ALT0
                __delay_cycles(300000);   //retraso de tiempo
                        break;

                        case 5:                 //Boton presionado quinta vez:
                        parado();         //Se va a posicion PARADO
                __delay_cycles(300000);   //retraso de tiempo
                        break;

                        case 6:                 //Boton presionado sexta vez:
                        paradobajo();     //Se va a posicion PARADO BAJO
                __delay_cycles(300000);   //retraso de tiempo
                        break;

                             }
                              }
                               }
                 else
                           {
                   P1OUT &= ~TI_CC_LED1;  //No encender led si no es precionado
                             }

       sleep(DELAY);
    #endif

  }
}

//Interrupcion del modulo tactil
#pragma vector=TIMER0_A0_VECTOR
__interrupt void ISR_Timer0_A0(void)
{
  TA0CTL &= ~(MC_1);
  TA0CCTL0 &= ~(CCIE);
  __bic_SR_register_on_exit(LPM3_bits+GIE);
}

//Funcion que envia un caracter de manera serial
void putr(char c) {
    while(!(IFG2 & UCA0TXIFG));          //Ciclo que se ejecuta hasta que el
buffer de TX se encuentra listo para enviar un nuevo byte
    UCA0TXBUF = c;                                  //pone el caracter en el
buffer y lo envia
```

```
    envial0(c);                                          //Envia cada caracter del
buffer por MEDIO DEL TRANSMISOR
}

//Funcion que declara el numero de servomotor y el angulo a ser enviado
posteriormente de manera serial
void in(unsigned int servo, unsigned int angle)
{
//bytes al servocontrolador
putr(0xAA);                                           //envia byte de inicio
(protocolo envio datos servocontroladora)
putr(0x0C);                                          //numero de dispositivo
(protocolo envio datos servocontroladora)
putr(0x04);                                           //byte de comando que
muktiplica por 4 la duracion del pulso
putr(servo);                                          //numero de servo

//el destino representa el ancho de pulso a transmitir en cuartos de
microsegundo:
//1000 µs para 0° y 2000 µs para 180°
putr(angle&0x7f);                                 //los 7 bits bajos del destino
putr(((angle>>7)&0x3f));                          //los 7 bits altos del destino
}

//Funcion de posinicial indicando el numero de servomotor y ancho de pulso
unsigned posinicial(void) { //posicion inicial
in(2,5000);
in(5,7000);
in(8,6000);
in(11,6000);
in(14,7000);
in(17,5000);

in(0,4000);
in(3,8000);
in(6,4000);
in(9,8000);
in(12,4000);
in(15,8000);

in(1,4000);
in(4,8000);
in(7,4000);
in(10,8000);
in(13,4000);
in(16,8000);
return 0;
}
//Funcion de paradobajo indicando el numero de servomotor y ancho de pulso
unsigned paradobajo(void) { //parado bajo
in(0,5500);
in(3,6500);
in(6,5500);
in(9,6500);
in(12,5500);
in(15,6500);

in(1,6000);
in(4,6000);
in(7,6000);
in(10,6000);
in(13,6000);
in(16,6000);
```

```
in(2,5000);
in(5,7000);
in(8,6000);
in(11,6000);
in(14,7000);
in(17,5000);
return 0; }
//Funcion de parado indicando el numero de servomotor y ancho de pulso
unsigned parado (void) { //parado
in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);

in(1,7000);
in(4,5000);
in(7,7000);
in(10,5000);
in(13,7000);
in(16,5000);

in(2,5000);
in(5,7000);
in(8,6000);
in(11,6000);
in(14,7000);
in(17,5000);
return 0;
}
//Funcion de paradoalto indicando el numero de servomotor y ancho de pulso
unsigned paradoalto (void) { //parado alto
in(2,5000);
in(5,7000);
in(8,6000);
in(11,6000);
in(14,7000);
in(17,5000);

in(0,8000);
in(3,4000);
in(6,8000);
in(9,4000);
in(12,8000);
in(15,4000);

in(1,8000);
in(4,4000);
in(7,8000);
in(10,4000);
in(13,8000);
in(16,4000);
return 0;
}
//Funcion de caminar adelante indicando todas las funciones por las que esta
compuesta
unsigned delante (void){
delantea();
delanteb();
delantec();
delanted();
delantee();
return 0; }
```

```
//Funcion de caminar atras indicando todas las funciones por las que esta
compuesta
unsigned atras (void){
atrasa();
atrasb();
atrasc();
return 0; }
//Funcion de girara a la derecha indicando todas las funciones por las que esta
compuesta
unsigned derecha(void){
derechaa();
derechab();
derechac();
return 0;
}
//Funcion de girara a la izquierda indicando todas las funciones por las que
esta compuesta
unsigned izquierda(void) {
izquierdaa();
izquierdab();
izquiedac();
return 0; }
//Funcion de delantea indicando el numero de servomotor y ancho de pulso
unsigned delantea(void) {
in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);

in(1,7000);
in(4,5000);
in(7,6000);
in(10,6000);
in(13,7000);
in(16,5000);

in(2,5000);
in(5,7000);
in(8,5500);
in(11,6500);
in(14,7000);
in(17,5000);

in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);

in(1,7000);
in(4,5000);
in(7,7000);
in(10,5000);
in(13,7000);
in(16,5000);

in(2,5000);
in(5,7000);
in(8,5500);
in(11,6500);
in(14,7000);
```

```
in(17,5000);
return 0;
}
//Funcion de delanteb indicando el numero de servomotor y ancho de pulso
unsigned delanteb(void){
in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);

in(1,7000);
in(4,6000);
in(7,7000);
in(10,5000);
in(13,6000);
in(16,5000);

in(2,5000);
in(5,8000);
in(8,5500);
in(11,6500);
in(14,5500);
in(17,5000);

in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);

in(1,7000);
in(4,5000);
in(7,7000);
in(10,5000);
in(13,7000);
in(16,5000);

in(2,5000);
in(5,8000);
in(8,5500);
in(11,6500);
in(14,5500);
in(17,5000);
return 0;
}
//Funcion de delantec indicando el numero de servomotor y ancho de pulso
unsigned delantec(void){
in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);

in(1,6000);
in(4,5000);
in(7,7000);
in(10,5000);
in(13,7000);
in(16,6000);
```

```
in(2,4000);
in(5,8000);
in(8,5500);
in(11,6500);
in(14,5500);
in(17,6500);

in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);


in(1,7000);
in(4,5000);
in(7,7000);
in(10,5000);
in(13,7000);
in(16,5000);

in(2,4000);
in(5,8000);
in(8,5500);
in(11,6500);
in(14,5500);
in(17,6500);
return 0;
}
//Funcion de delanted indicando el numero de servomotor y ancho de pulso
unsigned delanted(void) {
in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);

in(1,6000);
in(4,5000);
in(7,7000);
in(10,5000);
in(13,7000);
in(16,6000);

in(2,4000);
in(5,7000);
in(8,5500);
in(11,6500);
in(14,7000);
in(17,6500);

in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);

in(1,7000);
in(4,5000);
in(7,7000);
in(10,5000);
```

```
in(13,7000);
in(16,5000);

in(2,4000);
in(5,7000);
in(8,5500);
in(11,6500);
in(14,7000);
in(17,6500);
return 0;
}
//Funcion de delantee indicando el numero de servomotor y ancho de pulso
unsigned delantee(void) {
in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);

in(1,7000);
in(4,6000);
in(7,7000);
in(10,5000);
in(13,6000);
in(16,5000);

in(2,4000);
in(5,8000);
in(8,5500);
in(11,6500);
in(14,5500);
in(17,6500);

in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);

in(1,7000);
in(4,5000);
in(7,7000);
in(10,5000);
in(13,7000);
in(16,5000);

in(2,4000);
in(5,8000);
in(8,5500);
in(11,6500);
in(14,5500);
in(17,6500);
return 0;
}
//Funcion de atrasa indicando el numero de servomotor y ancho de pulso
unsigned atrasa (void){
in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);
```

```
in(1,7000);
in(4,5000);
in(7,5000);
in(10,7000);
in(13,7000);
in(16,5000);

in(2,5000);
in(5,7000);
in(8,7000);
in(11,5000);
in(14,7000);
in(17,5000);

in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);

in(1,7000);
in(4,5000);
in(7,7000);
in(10,5000);
in(13,7000);
in(16,5000);

in(2,5000);
in(5,7000);
in(8,7000);
in(11,5000);
in(14,7000);
in(17,5000);
return 0;
}
//Funcion de atrasb indicando el numero de servomotor y ancho de pulso
unsigned atrasb (void){
in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);

in(1,7000);
in(4,6000);
in(7,7000);
in(10,5000);
in(13,6000);
in(16,5000);

in(2,5000);
in(5,5000);
in(8,7000);
in(11,5000);
in(14,8000);
in(17,5000);

in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
```

```
in(12,6500);
in(15,5500);

in(1,7000);
in(4,5000);
in(7,7000);
in(10,5000);
in(13,7000);
in(16,5000);

in(2,5000);
in(5,5000);
in(8,7000);
in(11,5000);
in(14,8000);
in(17,5000);
return 0;
}
//Funcion de atrasc indicando el numero de servomotor y ancho de pulso
unsigned atrasc (void){
in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);

in(1,6000);
in(4,5000);
in(7,7000);
in(10,5000);
in(13,7000);
in(16,6000);

in(2,7000);
in(5,5000);
in(8,7000);
in(11,5000);
in(14,8000);
in(17,4000);

in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);

in(1,7000);
in(4,5000);
in(7,7000);
in(10,5000);
in(13,7000);
in(16,5000);

in(2,7000);
in(5,5000);
in(8,7000);
in(11,5000);
in(14,8000);
in(17,4000);
return 0;
}
//Funcion de derechaa indicando el numero de servomotor y ancho de pulso
```

```
unsigned derechaa(void) {
in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);

in(1,7000);
in(4,5000);
in(7,6000);
in(10,6000);
in(13,7000);
in(16,5000);

in(2,5000);
in(5,7000);
in(8,7000);
in(11,7000);
in(14,7000);
in(17,5000);

in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);

in(1,7000);
in(4,5000);
in(7,7000);
in(10,5000);
in(13,7000);
in(16,5000);

in(2,5000);
in(5,7000);
in(8,7000);
in(11,7000);
in(14,7000);
in(17,5000);
return 0;
}
//Funcion de derechab indicando el numero de servomotor y ancho de pulso
unsigned derechab (void){
in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);

in(1,7000);
in(4,6000);
in(7,7000);
in(10,5000);
in(13,6000);
in(16,5000);

in(2,5000);
in(5,8000);
in(8,7000);
in(11,7000);
```

```
in(14,8000);
in(17,5000);

in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);

in(1,7000);
in(4,5000);
in(7,7000);
in(10,5000);
in(13,7000);
in(16,5000);

in(2,5000);
in(5,8000);
in(8,7000);
in(11,7000);
in(14,8000);
in(17,5000);
return 0;
}
//Funcion de derechac indicando el numero de servomotor y ancho de pulso
unsigned derechac(void){
in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);

in(1,6000);
in(4,5000);
in(7,7000);
in(10,5000);
in(13,7000);
in(16,6000);

in(2,6000);
in(5,8000);
in(8,7000);
in(11,7000);
in(14,8000);
in(17,6000);

in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);

in(1,7000);
in(4,5000);
in(7,7000);
in(10,5000);
in(13,7000);
in(16,5000);

in(2,6000);
in(5,8000);
```

```
in(8,7000);
in(11,7000);
in(14,8000);
in(17,6000);
return 0;
}
//Funcion de izquierdaa indicando el numero de servomotor y ancho de pulso
unsigned izquierdaa (void){
in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);

in(1,7000);
in(4,5000);
in(7,6000);
in(10,6000);
in(13,7000);
in(16,5000);

in(2,5000);
in(5,7000);
in(8,5000);
in(11,5000);
in(14,7000);
in(17,5000);

in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);

in(1,7000);
in(4,5000);
in(7,7000);
in(10,5000);
in(13,7000);
in(16,5000);

in(2,5000);
in(5,7000);
in(8,5000);
in(11,5000);
in(14,7000);
in(17,5000);
return 0;
}
//Funcion de izquierdab indicando el numero de servomotor y ancho de pulso
unsigned izquierdab(void) {
in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);

in(1,6000);
in(4,5000);
in(7,7000);
in(10,5000);
```

```
in(13,7000);
in(16,6000);

in(2,4000);
in(5,7000);
in(8,5000);
in(11,5000);
in(14,7000);
in(17,4000);

in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);

in(1,7000);
in(4,5000);
in(7,7000);
in(10,5000);
in(13,7000);
in(16,5000);

in(2,4000);
in(5,7000);
in(8,5000);
in(11,5000);
in(14,7000);
in(17,4000);
return 0;
}
//Funcion de izquierdac indicando el numero de servomotor y ancho de pulso
unsigned izquierdac(void){
in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);

in(1,7000);
in(4,6000);
in(7,7000);
in(10,5000);
in(13,6000);
in(16,5000);

in(2,4000);
in(5,6000);
in(8,5000);
in(11,5000);
in(14,6000);
in(17,4000);

in(0,6500);
in(3,5500);
in(6,6500);
in(9,5500);
in(12,6500);
in(15,5500);

in(1,7000);
in(4,5000);
```

```
in(7,7000);
in(10,5000);
in(13,7000);
in(16,5000);

in(2,4000);
in(5,6000);
in(8,5000);
in(11,5000);
in(14,6000);
in(17,4000);
return 0;
}
//Funcion que envia el paquete de datos por RF
void envial0(char c)
{
char Buffer[3];
Buffer[0] = 2;                          // longitud del paquete
Buffer[1] = 0x01;                       // direccion del paquete
Buffer[2] = c;                                      // dato a enviar
RFSendPacket(Buffer, 3);                // Envia dato por RF
}
```

APÉNDICE B

CÓDIGO RECEPTOR

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++
//CODIGO DEL RECEPTOR::
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++
#include "include.h"
#define CC110L_BUFFER_LENGTH 64

//Declaracion de Variables
extern char paTable[];
extern char paTableLen;
char txBuffer[12];
char rxBuffer[12];
unsigned int i = 0;
int conta = 0;
unsigned char TxByte=0;
unsigned int result;
unsigned int del;
int d;

//Declaracion de Funciones
void putr(char c);
void in(unsigned int servo, unsigned int angle);
unsigned posinicial(void);
unsigned paradobajo(void);
unsigned parado (void);
unsigned paradoalto (void);

void main (void)
{
WDTCTL = WDTPW + WDTHOLD;                 // Detiene WDT
///   __delay_cycles(500000);
BCSCTL1 = CALBC1_1MHZ;                    // Se fija el registro de control del
reloj a 1 MHz
DCOCTL = CALDCO_1MHZ;                          // Se fija el registro de control
del DCO fijo a 1 MHz
P1SEL = BIT1 + BIT2 ;                     // Fija bits UART P1.1 = RXD, P1.2=TXD
P1SEL2 = BIT1 + BIT2 ;                    // Fija bits UART P1.1 = RXD, P1.2=TXD
UCA0CTL1 |= UCSSEL_2;                     // SMCLK
UCA0BR0 = 52;                             // 1MHz 9600
UCA0BR1 = 0;                              // 1MHz 9600
UCA0MCTL = UCBRS0;                        // Modulacion
UCA0CTL1 &= ~UCSWRST;                     // Inicia USCI
IE2 |= UCA0RXIE;//+UCA0TXIE;              // Habilita USCI_A0 RX interrupcion

//Configuracion del transmisor CC110L
//Contiene funciones que permiten al MSP430 accesar a la interfaz del CC110L
(funciones de la biblioteca CC110L)
TI_CC_SPISetup();                         // Inicializa puertos SPI
TI_CC_PowerupResetCCxxxx();               // Reset al CC110L
writeRFSettings();                        // Write RF settings to config reg
TI_CC_SPIWriteBurstReg(TI_CCxxx0_PATABLE, paTable, paTableLen);//Write PATABLE

TI_CC_LED_PxOUT &= ~(TI_CC_LED1);          // Fija Bit 0 como dato de salida
(led mando tactil)
TI_CC_LED_PxDIR |= TI_CC_LED1;                  // Fija direccion de salida de
led Bit 0
TI_CC_GDO0_PxIES |= TI_CC_GDO0_PIN;       // Fija Interrupcion en flanco de
bajada (end of pkt)
TI_CC_GDO0_PxIFG &= ~TI_CC_GDO0_PIN;      // Limpia bandera
TI_CC_GDO0_PxIE |= TI_CC_GDO0_PIN;        // Interrupcion habilitada
TI_CC_SPIStrobe(TI_CCxxx0_SRX);           // Inicializa CC110L en modo RX
                                          // Cuando un paquete es recibido, el
pin GDO0 se pone en alto
```

```
                                          // y el cpu es activado
__bis_SR_register(LPM0_bits + GIE);       // Habilita interrupcion general y
activa modo LMP0
}

//Declaracion de interrupcion del pin GDO0
#pragma vector=PORT2_VECTOR
__interrupt void Port2_ISR(void)
{

//Cuando se activa la interrupcion del puerto 2:
    char len = CC110L_BUFFER_LENGTH;       //se declara variable len
  if(TI_CC_GDO0_PxIFG & TI_CC_GDO0_PIN);  //Si la bandera del P2 Bit 6 se pone
en alto
  {
if (RFReceivePacket(rxBuffer,&len))        //Y si ademas recibe un paquete de
longitud variable:
    {
         P1OUT ^= TI_CC_LED1;             //enciende el led p1.1
     putr(rxBuffer[1]); //Transmite por el UART el paquete de datos hacia la
ctrladora de servos

     rxBuffer[0]=0;                      //Limpia el buffer de datos recibidos
     rxBuffer[1]=0;
     }
    }
TI_CC_GDO0_PxIFG &= ~TI_CC_GDO0_PIN;      //Limpia la bandera que causo la
interrupcion
}
//Funcion que envia cada byte recibido a la servoctroladora por medio del UART
void putr(char c) {
while(!(IFG2 & UCA0TXIFG)); //Ciclo que se ejecuta hasta que el buffer de TX se
encuentra listo para enviar un nuevo byte
UCA0TXBUF = c;          //pone el caracter en el buffer y lo envia por el UART

}

//Funcion que declara el numero de servomotor y el angulo a ser enviado
posteriormente de manera serial
void in(unsigned int servo, unsigned int angle)
{
//bytes al servocontrolador
putr(0xAA); //envia byte de inicio (protocolo envio datos servocontroladora)
putr(0x0C); //numero de dispositivo (protocolo envio datos servocontroladora)
putr(0x04); //byte de comando que muktiplica por 4 la duracion del pulso
putr(servo);//numero de servo

//el destino representa el ancho de pulso a transmitir en cuartos de
//microsegundo:
//1000 µs para 0° y 2000 µs para 180°
putr(angle&0x7f);                              //los 7 bits bajos del destino
putr(((angle>>7)&0x3f));                       //los 7 bits altos del destino
}
```

REFERENCIAS

[1] Joao Barreto, “FBD The free body diagram method. Kinematics and dynamic modeling of a six leg robot”, IEEE The institution of electrical enginers

[2] Irving L. Kosow, “Máquinas eléctricas y transformadores”, Segunda edición, Editorial Pearson Educación

[3] John H. Davies, “MSP430 Microcontroller basics”. Primera edición, Editorial Newnes

[4] Antonio Barrientos, “Fundamentos de robótica” Segunda edición, Editorial MacGraw hill

[5] Anibal Ollero Baturon, “Robótica: Manipuladores y robots móviles” Primera edición, Editorial Marcombo

[6] BoosterPack_Users_Manual.pdf (Manual del Air BoosterPack)

[7] slaa491b.pdf (Getting Started With Capacitive Touch Software Library)

[8] slau227e.pdf (eZ430-RF2500 Development Tool)

[9] slau337a.pdf (430BOOST-SENSE1 - Capacitive Touch user manual)

[10] http://www.hitecrcd.com/products/analog/standard-sport/hs-322hd.html

[11] http://www.pololu.com/catalog/product/1354

[12] http://www.processors.wiki.ti.com/index.php/MSP430_LaunchPad_(MSP-EXP430G2)

[13] http://www.ti.com/product/msp430g2553

[14] http://www.sketchup.google.com

[15] http://es.wikipedia.org/wiki/MSP430

[16] http://www.ti.com/tool/msp-exp430g2