INGENIERÍA MECATRÓNICA
MEC

Universidad Católica Boliviana "San Pablo"
Departamento de Ciencias de la Tecnología e Innovación – Tarija

Informe de Proyecto Robot Bípedo

Asignatura: IMT- 222 Sistemas Embebidos I
Docente: Ing. Elmer Alan Cornejo Q.
ecornejo@ucb.edu.bo

Sergio Alejandro Mercado Vallejos
Valeria Alejandra Hoyos Tovar
Roberth Williams Ruiz Condori

Tarija, 09 de diciembre de 2025

Contenido

RESUMEN

El presente trabajo desarrolla el diseño, modelado, implementación y validación de un robot bípedo de seis grados de libertad (6 GDL) controlado mediante un sistema embebido basado en el microcontrolador ESP32, utilizando el sistema operativo de tiempo real FreeRTOS.

El objetivo principal del proyecto es integrar conceptos de robótica, mecanismos, sistemas embebidos y control en tiempo real, aplicados al desarrollo de un prototipo funcional de marcha bípedo, orientado a fines académicos y experimentales.

El robot está compuesto por dos piernas simétricas, cada una con tres grados de libertad, permitiendo movimientos básicos de caminata y estabilidad. Para el control de movimiento se emplean servomotores MG995, accionados mediante un controlador PWM PCA9685, mientras que la orientación y equilibrio del sistema se obtiene a través de una unidad de medición inercial (IMU MPU9250).

El software se estructura mediante múltiples tareas bajo FreeRTOS, permitiendo una ejecución concurrente y determinista de procesos como lectura de sensores, planificación de trayectoria, control de actuadores y gestión del sistema.

Adicionalmente, se realiza el modelado cinemático del robot, incluyendo el análisis de grados de libertad, cinemática directa e inversa, y generación de trayectorias de marcha. Dichos modelos son validados mediante simulaciones en MATLAB, donde se evalúan posiciones, trayectorias y requerimientos de par en las articulaciones.
Los resultados obtenidos demuestran la viabilidad del sistema propuesto y su utilidad como plataforma didáctica para el estudio de robótica bípedo y sistemas embebidos en tiempo real.

INTRODUCCIÓN

Planteamiento del problema

La robótica bípeda representa uno de los mayores desafíos dentro del campo de la robótica móvil debido a su complejidad mecánica, cinemática y de control. A diferencia de los robots con ruedas, los robots bípedos deben mantener equilibrio dinámico, coordinar múltiples articulaciones y responder en tiempo real a perturbaciones externas.

En el ámbito académico, especialmente en asignaturas como Sistemas Embebidos I, se identifica una limitación frecuente: la dificultad de integrar teoría y práctica en proyectos reales que involucren sistemas de tiempo real, control concurrente y modelado matemático del movimiento. Muchos proyectos se limitan a aplicaciones simples que no reflejan la complejidad de un sistema embebido real.

Por esta razón, surge la necesidad de desarrollar un robot bípedo didáctico, capaz de integrar:

- Control de múltiples actuadores.
- Lectura y procesamiento de sensores en tiempo real.
- Planificación de movimiento.
- Uso de un sistema operativo de tiempo real.

- Análisis mecánico y matemático del movimiento.

Antecedentes y trabajos relacionados

Diversos proyectos académicos han abordado el desarrollo de robots bípedos utilizando plataformas comerciales o de investigación. Sin embargo, muchos de estos sistemas presentan alguna de las siguientes limitaciones:

- Alto costo de componentes.
- Arquitecturas cerradas o poco flexibles.
- Uso de sistemas sin control en tiempo real formal.
- Escasa documentación del modelado matemático.

En contraste, el uso de microcontroladores como el ESP32, combinado con FreeRTOS, permite la implementación de sistemas de control avanzados a bajo costo, manteniendo un alto grado de flexibilidad. Asimismo, el uso de herramientas como MATLAB facilita el análisis cinemático y la validación previa al montaje físico.

Justificación del proyecto

La realización de este proyecto se justifica desde los siguientes enfoques:

- **Justificación académica:** Permite aplicar de forma práctica los contenidos de sistemas embebidos, programación en tiempo real, robótica y mecanismos, fortaleciendo las competencias del estudiante en integración hardware-software.
- **Justificación técnica:** El uso de FreeRTOS permite demostrar la importancia de la concurrencia, sincronización y determinismo temporal en un sistema complejo como un robot bípedo.
- **Justificación formativa:** El proyecto actúa como una plataforma de aprendizaje para futuros desarrollos más avanzados, como control balanceado, visión artificial o aprendizaje automático.

OBJETIVO GENERAL

Diseñar e implementar un robot bípedo de seis grados de libertad, controlado mediante un sistema embebido basado en ESP32 y FreeRTOS, incorporando modelado cinemático y simulación de movimientos en MATLAB, con fines académicos y experimentales.

OBJETIVO ESPECIFICO

Diseñar la estructura mecánica del robot bípedo con 6 grados de libertad.

- Realizar el análisis cinemático del sistema.
- Implementar el control de movimiento utilizando FreeRTOS.
- Simular los movimientos de marcha en MATLAB.
- Integrar sensores y actuadores en un sistema embebido funcional.
- Validar el comportamiento del robot mediante pruebas experimentales.

ALCANCES Y LIMITACIONES

Alcances:

- Desarrollo de un prototipo funcional.
- Implementación de marcha básica en superficie plana.
- Análisis cinemático y simulación.

Limitaciones:

- No se implementa control dinámico avanzado (ZMP completo).
- El sistema se limita a 6 GDL.
- No se incluye visión artificial.

Metodología de desarrollo

El proyecto se desarrolla siguiendo una metodología incremental:

1. Análisis teórico.
2. Diseño mecánico.
3. Modelado matemático.
4. Simulación en MATLAB.
5. Implementación embebida.
6. Pruebas y validación.

Estructura del documento

El documento se organiza, comenzando con fundamentos teóricos, seguido del diseño mecánico, modelado cinemático, implementación embebida, simulación, pruebas y conclusiones finales.

FUNDAMENTOS TEÓRICOS

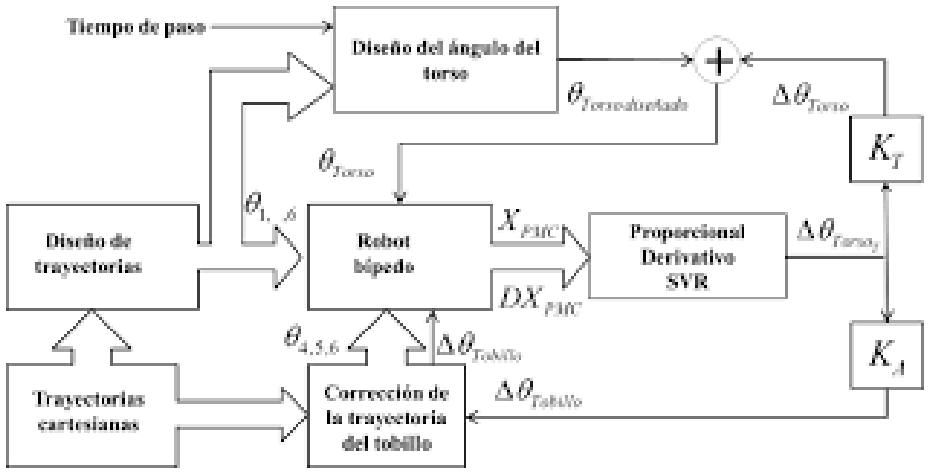
Sistemas embebidos aplicados a la robótica bípedo

Un sistema embebido es un sistema computacional dedicado a realizar una o varias funciones específicas dentro de un sistema mayor. A diferencia de un computador de propósito general, un sistema embebido está diseñado para ejecutar tareas concretas con restricciones bien definidas de tiempo, consumo energético, costo y recursos.

En el contexto de la robótica bípedo, los sistemas embebidos cumplen un papel fundamental, ya que deben:

- Leer datos de múltiples sensores en tiempo real.
- Procesar información para la toma de decisiones.
- Controlar actuadores con precisión temporal.
- Responder de forma determinista ante eventos externos.

Un robot bípedo puede ser considerado un sistema embebido complejo, debido a la integración simultánea de subsistemas mecánicos, eléctricos y de software, los cuales deben operar de manera coordinada para garantizar estabilidad y movimiento adecuado.



Características de un sistema embebido de tiempo real

Un sistema embebido de tiempo real se caracteriza por la necesidad de cumplir **restricciones temporales estrictas**, conocidas como *deadlines*. En el caso de un robot bípedo, el incumplimiento de dichas restricciones puede provocar:

- Pérdida de equilibrio.
- Movimientos erráticos.
- Caídas del robot.
- Daños mecánicos o eléctricos.

Las principales características incluyen:

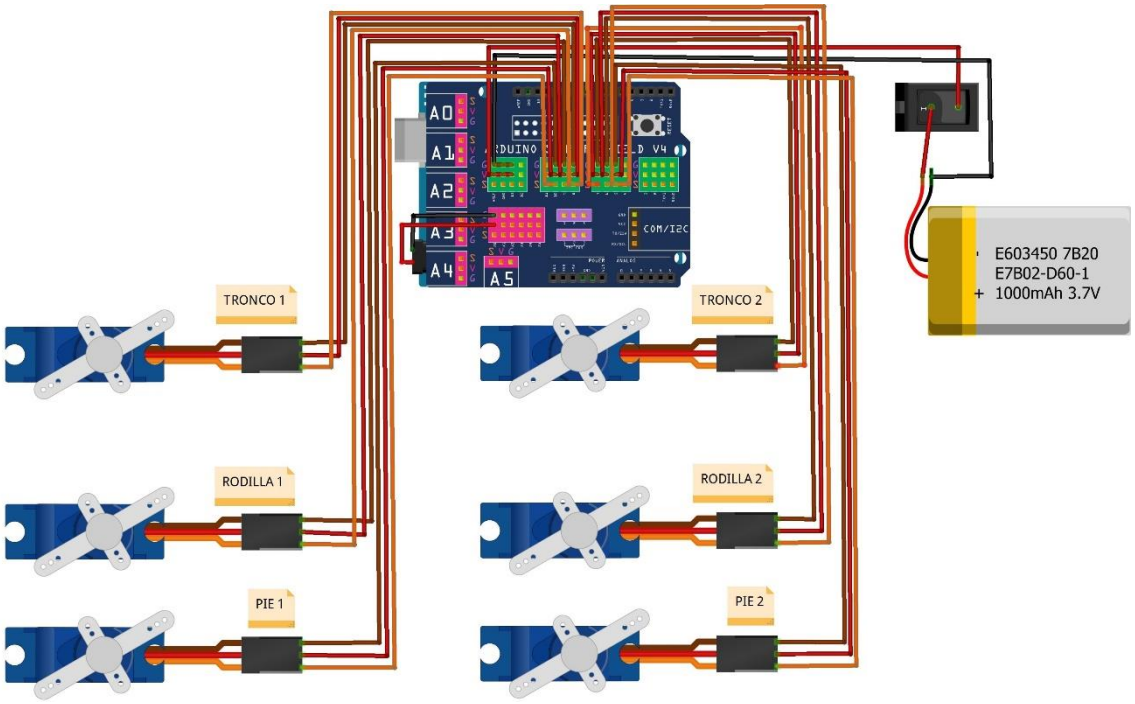
- **Respuesta determinista:** el sistema debe responder dentro de un intervalo de tiempo conocido.
- **Concurrencia:** ejecución simultánea de múltiples tareas.
- **Bajo consumo de recursos:** uso eficiente de memoria y CPU.
- **Alta confiabilidad:** operación continua sin fallos críticos.

Arquitectura general de un robot bípedo embebido

La arquitectura de un robot bípedo puede dividirse en los siguientes bloques funcionales:

1. **Unidad de procesamiento:** microcontrolador ESP32.
2. **Sensores:** IMU (acelerómetro, giroscopio, magnetómetro).
3. **Actuadores:** servomotores para las articulaciones.
4. **Sistema de control:** software basado en FreeRTOS.
5. **Fuente de energía:** baterías y reguladores de voltaje.

Cada uno de estos bloques interactúa mediante buses de comunicación y señales de control, formando un sistema integrado.



Microcontrolador ESP32

El ESP32 es un microcontrolador de 32 bits desarrollado por Espressif Systems, ampliamente utilizado en aplicaciones de IoT, robótica y sistemas embebidos avanzados. Se caracteriza por ofrecer:

- Alto poder de procesamiento.
- Amplia variedad de periféricos.
- Compatibilidad con sistemas operativos de tiempo real como FreeRTOS.

Arquitectura interna del ESP32

El ESP32 posee una arquitectura dual-core, basada en procesadores Xtensa LX6, lo que permite distribuir tareas críticas entre núcleos.

Principales características:

- Frecuencia de reloj hasta 240 MHz.
- Memoria SRAM integrada.
- Múltiples timers de hardware.
- Interfaces de comunicación: UART, I2C, SPI, PWM.
- Soporte nativo para FreeRTOS.

En este proyecto, uno de los núcleos se dedica principalmente a tareas de control y el otro a tareas auxiliares, permitiendo una mejor administración del tiempo real.

Justificación del uso del ESP32

El ESP32 fue seleccionado debido a:

- Capacidad de ejecutar FreeRTOS de forma nativa.
- Disponibilidad de múltiples buses I2C para sensores.
- Soporte para control de servomotores mediante PWM.
- Bajo costo y amplia documentación.
- Facilidad de programación en entorno Arduino/C++.

Estas características lo convierten en una plataforma ideal para proyectos académicos de robótica bípedo.

Sistemas Operativos de Tiempo Real (RTOS)

Un Sistema Operativo de Tiempo Real (RTOS) es un sistema operativo diseñado para gestionar tareas con restricciones temporales estrictas. A diferencia de un sistema operativo tradicional, el objetivo principal de un RTOS no es maximizar el rendimiento promedio, sino garantizar que las tareas se ejecuten dentro de plazos predecibles.

En un robot bípedo, el control de marcha requiere que cada cálculo y movimiento se ejecute en un intervalo de tiempo determinado.

FreeRTOS

FreeRTOS es un RTOS liviano y de código abierto, ampliamente utilizado en microcontroladores. Proporciona:

- Gestión de tareas (*tasks*).
- Planificador (*scheduler*).
- Comunicación entre tareas mediante colas.
- Sincronización mediante semáforos y mutex.
- Temporizadores de software.

Ventajas de FreeRTOS en robótica bípedo

El uso de FreeRTOS en este proyecto aporta múltiples ventajas:

- Separación lógica de funcionalidades.
- Ejecución concurrente de sensores y actuadores.
- Mayor estabilidad del sistema.
- Escalabilidad del software.
- Mejor depuración y mantenimiento.

En un robot bípedo, FreeRTOS permite ejecutar, por ejemplo, la lectura de la IMU y el control de los servos de manera independiente y sincronizada.

Tareas, prioridades y planificación

Cada tarea en FreeRTOS se ejecuta con una prioridad asignada.
Las tareas críticas, como el control de equilibrio, deben tener mayor prioridad que tareas secundarias como comunicaciones.

Ejemplo de jerarquía:

- Alta prioridad: control de marcha.
- Media prioridad: lectura de sensores.
- Baja prioridad: comunicación y monitoreo.

Comunicación digital en el sistema

Protocolo I2C

El protocolo **I2C (Inter-Integrated Circuit)** es un bus de comunicación serial síncrono que utiliza dos líneas:

- SDA: datos.
- SCL: reloj.

Características principales:

- Comunicación maestro-esclavo.
- Direccionamiento por hardware.
- Soporte para múltiples dispositivos en el mismo bus.

Uso de I2C en el proyecto

En este proyecto, el bus I2C se utiliza para:

- Comunicación con la IMU MPU9250.
- Control del módulo PCA9685.

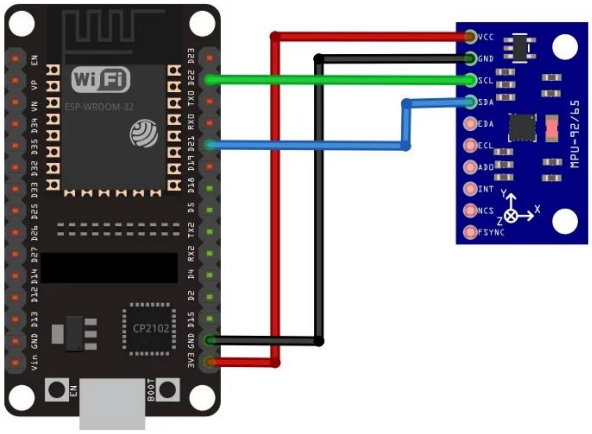
El ESP32 actúa como maestro, mientras que los periféricos actúan como esclavos.

PWM y control de servomotores

Los servomotores requieren una señal PWM con:

- Frecuencia típica: 50 Hz.
- Pulso variable entre 1 ms y 2 ms.

El controlador PCA9685 permite generar múltiples señales PWM de manera precisa, descargando al microcontrolador de esta tarea.



FUNDAMENTOS DE ROBOTICA Y MECANISMOS BIPEDOS

Un robot bípedo es un sistema mecánico que imita la locomoción humana mediante dos extremidades inferiores articuladas. Cada pierna se modela como una cadena cinemática serial.

Articulaciones y eslabones

- **Eslabón:** parte rígida del mecanismo.
- **Articulación:** unión entre eslabones que permite movimiento relativo.

En el presente proyecto se emplean articulaciones rotacionales accionadas por servomotores.

Grados de libertad (GDL)
El número de grados de libertad define la cantidad de movimientos independientes que puede realizar un sistema.

El robot bípedo cuenta con:

- 3 GDL por pierna.
- 6 GDL totales.

Estos permiten movimientos de flexión y extensión necesarios para la marcha.

Importancia de los GDL en la marcha

Los 6 GDL permiten:

- Levantar el pie.
- Avanzar la pierna.
- Mantener estabilidad básica.

Un número menor limitaría severamente la locomoción, mientras que un número mayor incrementaría la complejidad del control.

Descripción general del sistema

El robot bípedo desarrollado se compone de dos subsistemas principales:

- **Subsistema mecánico:** estructura del robot, eslabones, articulaciones y servomotores.
- **Subsistema embebido:** microcontrolador, sensores, actuadores, fuente de energía y software de control.

Ambos subsistemas se integran para conformar un sistema mecatrónico completo.

Arquitectura de hardware

El hardware del robot bípedo incluye los siguientes bloques:

- **Microcontrolador ESP32:** unidad central de procesamiento del sistema.
- **Controlador PWM PCA9685:** generación de señales PWM para los servomotores.
- **Servomotores MG995:** actuadores de las articulaciones.
- **IMU MPU9250:** adquisición de datos de orientación y aceleración.
- **Sistema de alimentación:** baterías 18650, regulador XL4016 y BMS 2S.

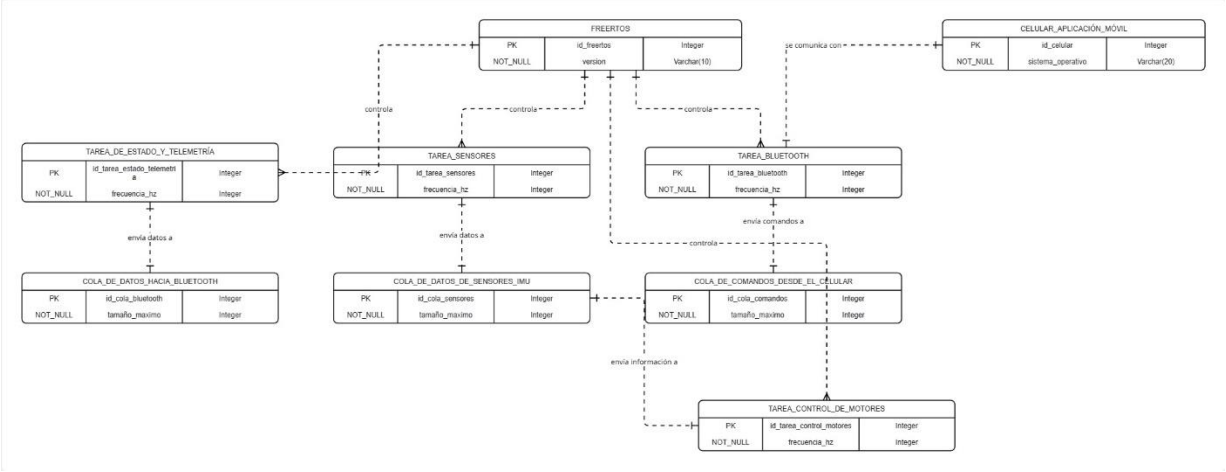
La interconexión entre los módulos se realiza principalmente mediante el bus **I2C**, mientras que la alimentación se distribuye desde el regulador principal.

Arquitectura de software

El software del sistema está estructurado de la siguiente manera:

- **Sistema operativo:** FreeRTOS.
- **Tareas principales:**
  - Lectura de sensores.
  - Procesamiento de datos inerciales.
  - Generación de trayectorias de marcha.
  - Control de servomotores.
  - Gestión del sistema.

Cada tarea se ejecuta de manera concurrente y sincronizada, respetando prioridades y períodos de ejecución.



Especificación de hardware

ESP32 Dev Module

- Arquitectura: 32 bits.
- Núcleos: Dual-core.
- Frecuencia máxima: 240 MHz.
- Memoria: SRAM integrada.
- Periféricos: I2C, PWM, UART, SPI.

Sensores

MPU9250 (IMU):

- Acelerómetro de 3 ejes.
- Giroscopio de 3 ejes.
- Magnetómetro de 3 ejes.
- Comunicación I2C.

Su función principal es proporcionar información de orientación del cuerpo del robot, indispensable para análisis de estabilidad.

Actuadores

Servomotores MG995:

- Tipo: Servomotor de rotación limitada.
- Torque nominal: suficiente para prototipos educativos.
- Alimentación: 6 V.
- Control mediante señal PWM.

Controlador PWM

PCA9685:

- 16 canales PWM independientes.
- Resolución: 12 bits.
- Comunicación I2C.

- Frecuencia configurable.

Este módulo permite controlar múltiples servomotores sin comprometer los recursos del microcontrolador.

Sistema de alimentación

El sistema de alimentación está compuesto por:

- Baterías 18650 en configuración 2S.
- BMS para protección contra sobrecarga y descarga.
- Regulador DC-DC XL4016 para estabilizar el voltaje.

Este sistema garantiza una alimentación estable tanto para la electrónica como para los actuadores.

ESPECIFICACION DE SOFTWARE

Lenguaje de programación

El software se desarrolla en C/C++, utilizando el entorno Arduino para ESP32, compatible con FreeRTOS.

Estructura del software

El código se organiza en módulos:

- Módulo de sensores.
- Módulo de actuadores.
- Módulo de control de marcha.
- Módulo de gestión del sistema.

Esta estructura mejora la mantenibilidad y escalabilidad del software.

Condiciones de operación

El robot bípedo está diseñado para operar bajo las siguientes condiciones:

- Superficies planas y rígidas.
- Temperatura ambiente.
- Carga limitada al peso propio del robot.

Criterios de validación

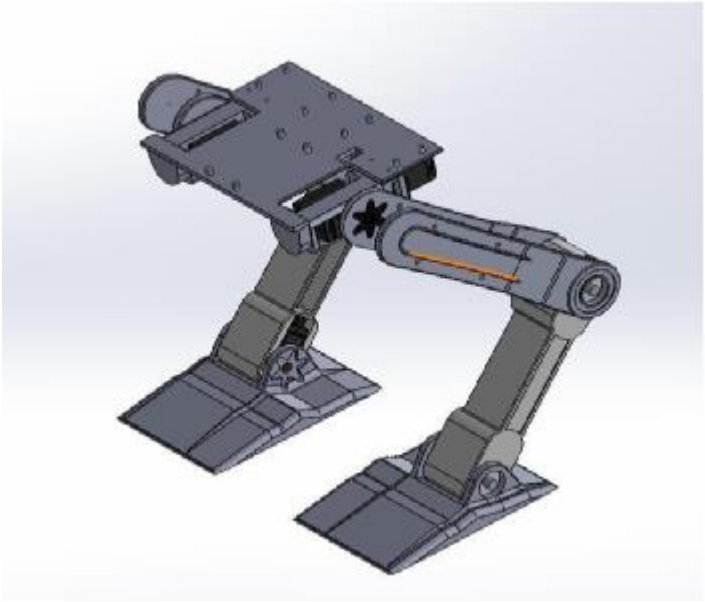
Los criterios de validación del sistema incluyen:

- Correcta ejecución de tareas en FreeRTOS.
- Movimientos coordinados sin colisiones mecánicas.
- Estabilidad del sistema durante la marcha.
- Coherencia entre simulación y comportamiento real.

DISEÑO MECÁNICO DEL ROBOT BÍPEDO Y ANÁLISIS DE GRADOS DE LIBERTAD

El diseño mecánico de un robot bípedo constituye uno de los aspectos más críticos del proyecto, ya que determina directamente la capacidad de movimiento, estabilidad y viabilidad del control. A diferencia de otros robots móviles, la locomoción bípeda exige una coordinación precisa entre múltiples articulaciones, así como una correcta distribución de masas para evitar caídas.

En este proyecto se opta por un diseño simplificado pero funcional, orientado a fines académicos, que permite implementar conceptos de cinemática y control sin incurrir en una complejidad excesiva.



Configuración general del robot bípedo

El robot está compuesto por dos extremidades inferiores simétricas unidas a un torso rígido. Cada pierna se modela como una cadena cinemática serial compuesta por eslabones rígidos y articulaciones rotacionales accionadas por servomotores.

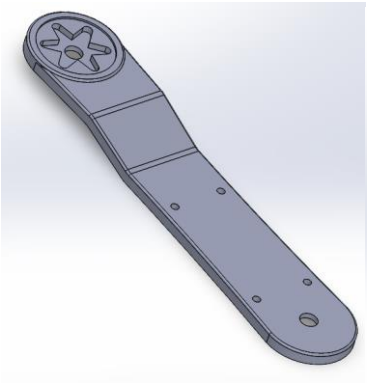
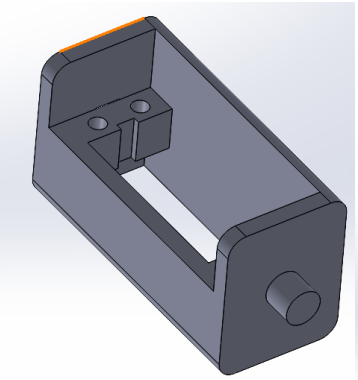
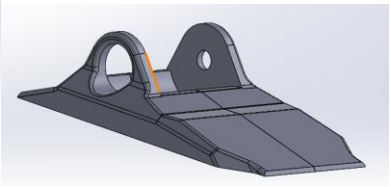
Estructura general

Cada pierna está formada por los siguientes segmentos:

- Pie
- Pierna inferior (tibia)
- Pierna superior (muslo)

Las articulaciones se localizan en:

- Tobillo
- Rodilla
- Cadera

Definición de grados de libertad

Tabla – Identificación de eslabones y articulaciones del robot bípedo

Pierna	Articulación	Eslabón superior (SolidWorks)	Eslabón inferior (SolidWorks)	Tipo de articulación	GDL
Izquierda	Cadera	base	femur	Rotacional	1
Izquierda	Rodilla	femur	tibia	Rotacional	1
Izquierda	Tobillo	tibia	p_izq	Rotacional	1
Derecha	Cadera	base2	femur2	Rotacional	1
Derecha	Rodilla	femur2	tibia	Rotacional	1
Derecha	Tobillo	tibia	p_der	Rotacional	1

Identificación funcional de piezas estructurales

Tabla – Descripción de componentes mecánicos

Nombre (SolidWorks)	Función mecánica	Descripción
base	Soporte izquierdo	Unión estructural entre el torso y la pierna izquierda
base2	Soporte derecho	Unión estructural entre el torso y la pierna derecha
base22	Refuerzo central	Aumenta rigidez estructural del cuerpo
femur	Muslo izquierdo	Eslabón superior de la pierna izquierda
femur2	Muslo derecho	Eslabón superior de la pierna derecha
tibia	Pierna inferior	Eslabón inferior común
p_izq	Pie izquierdo	Soporte durante fase de apoyo
p_der	Pie derecho	Soporte durante fase de apoyo

apoyo	Base auxiliar	Incrementa estabilidad en reposo

MATERIALES Y FABRICACION

La estructura del robot se diseña para ser fabricada mediante impresión 3D en PLA, debido a:

- Bajo costo.
- Facilidad de fabricación.
- Suficiente rigidez para cargas moderadas.

Las piezas se ensamblan mediante tornillos métricos, permitiendo desmontaje y mantenimiento.

MODELADO CINEMÁTICO DEL ROBOT BÍPEDO DE 6 GDL

El modelado cinemático permite establecer la relación matemática entre las variables articulares del robot y la posición del extremo final, en este caso el pie. Este análisis es esencial para generar trayectorias de marcha y transformar posiciones deseadas en ángulos de los servomotores.

Preparación para simulación en MATLAB

Las ecuaciones de cinemática directa e inversa sirven como base para:

- Simular la marcha.
- Verificar ángulos límites.

- Calcular velocidades articulares.

Mathlab

```
/* definición de parámetros*/
Lf = 0.10;  % femur (m)
Lt = 0.10;  % tibia (m)
Lp = 0.05;  % pie (m)
/* Generación de trayectoria*/
T = 2;          % tiempo del paso
t = linspace(0,T,100);
v = 0.04;       % velocidad
h = 0.03;       % altura del paso
x = v*t;
z = h*sin(pi*t/T);
/*Cinemática inversa*/
theta1 = zeros(size(t));
theta2 = zeros(size(t));
theta3 = zeros(size(t));
for i = 1:length(t)
    D = (x(i)^2 + z(i)^2 - Lf^2 - Lt^2) / (2*Lf*Lt);
    theta2(i) = acos(D);
    theta1(i) = atan2(z(i), x(i)) - atan2(Lt*sin(theta2(i)), ...
                                    Lf + Lt*cos(theta2(i)));
```

DISEÑO DEL SISTEMA EMBEBIDO BASADO EN FreeRTOS

El control de un robot bípedo requiere la ejecución simultánea de múltiples procesos que deben operar de manera coordinada y en tiempo real. Para este propósito se utiliza FreeRTOS, un sistema operativo de tiempo real que permite estructurar el software de manera modular, escalable y determinista.

Arquitectura de software del sistema

El software se estructura en capas:

1. Capa de hardware (drivers).
2. Capa de servicios (sensores, actuadores).
3. Capa de control.
4. Capa de aplicación.

Esta arquitectura favorece el mantenimiento y la expansión del sistema.

Definición de tareas (Tasks)

Tarea de lectura de IMU

Nombre: `Task_IMU`
Prioridad: Media
Periodo: 10 ms

Funciones:

- Lectura del MPU9250 vía I2C.
- Filtrado básico de datos.
- Envío de datos a una cola.

Tarea de planificación de marcha

Nombre: `Task_GaitPlanner`
Prioridad: Alta
Periodo: 20 ms

Funciones:

- Generación de trayectorias de paso.
- Cálculo de referencias articulares.
- Envío de ángulos deseados.

Tarea de control de servomotores

Nombre: `Task_ServoControl`
Prioridad: Alta
Periodo: 20 ms

Funciones:

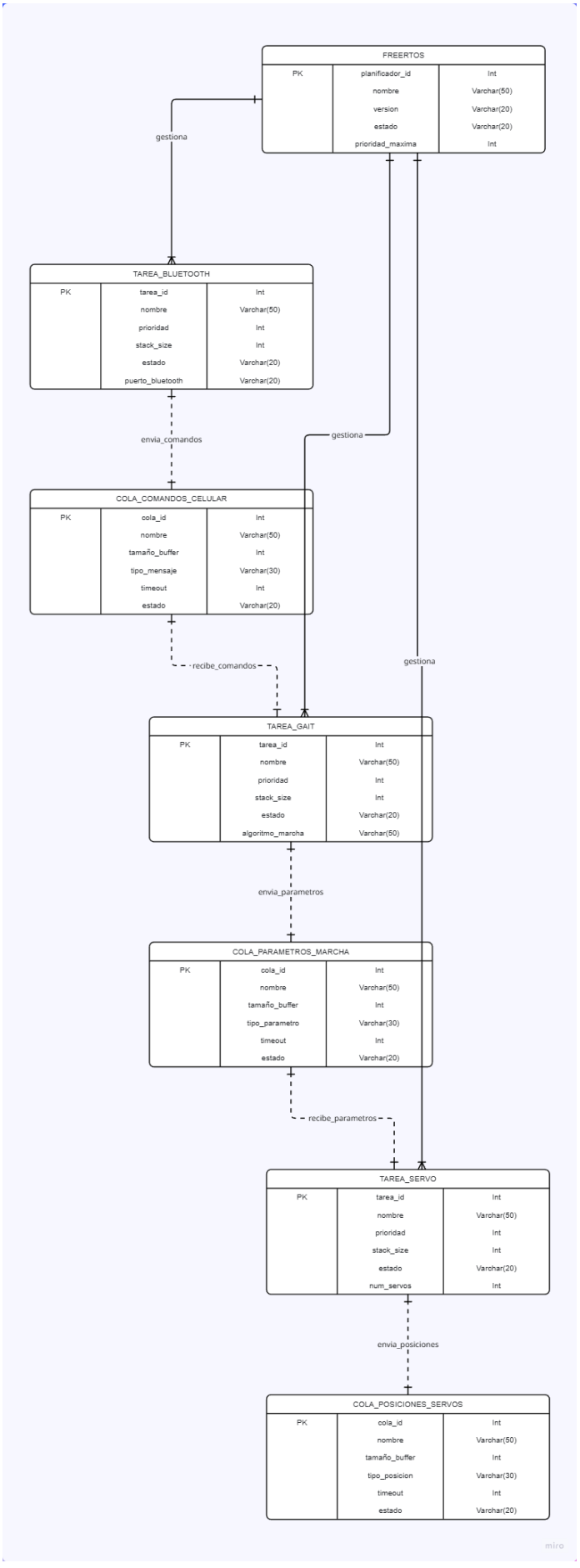
- Conversión de ángulos a PWM.
- Control de PCA9685.
- Verificación de límites mecánicos.

Tarea de gestión del sistema

Nombre: `Task_System`
Prioridad: Baja

Funciones:

- Inicialización del sistema.
- Supervisión de errores.
- Gestión de estados.


FREERTOS
PK
planificador_id
nombre
version
estado
prioridad_maxima
Int
Varchar(50)
Varchar(20)
Varchar(20)
Int
gestiona
TAREA_BLUETOOTH
PK
tarea_id
nombre
prioridad
stack_size
estado
puerto_bluetooth
Int
Varchar(50)
Int
Int
Varchar(20)
Varchar(20)
envia_comandos
gestiona
COLA_COMANDOS_CELULAR
PK
cola_id
nombre
tamaño_buffer
tipo_mensaje
timeout
estado
Int
Varchar(50)
Int
Varchar(30)
Int
Varchar(20)
recibe_comandos
gestiona
TAREA_GAIT
PK
tarea_id
nombre
prioridad
stack_size
estado
algoritmo_marcha
Int
Varchar(50)
Int
Int
Varchar(20)
Varchar(50)
envia_parametros
COLA_PARAMETROS_MARCHA
PK
cola_id
nombre
tamaño_buffer
tipo_parametro
timeout
estado
Int
Varchar(50)
Int
Varchar(30)
Int
Varchar(20)
recibe_parametros
TAREA_SERVO
PK
tarea_id
nombre
prioridad
stack_size
estado
num_servos
Int
Varchar(50)
Int
Int
Varchar(20)
Int
envia_posiciones
COLA_POSICIONES_SERVOS
PK
cola_id
nombre
tamaño_buffer
tipo_posicion
timeout
estado
Int
Varchar(50)
Int
Varchar(30)
Int
Varchar(20)
miro

Simulación Código

ProyectoBipedo.h

```
#ifndef PROYECTO_BIPEDO_H
#define PROYECTO_BIPEDO_H
#include <Arduino.h>
#include <Wire.h>
#include <freertos/FreeRTOS.h>
#include <freertos/task.h>
#include <freertos/queue.h>
#include <Adafruit_PWMServoDriver.h>
#include "BluetoothSerial.h"
#define SERVO_FREQ 50
#define HIP_L   0
#define KNEE_L  1
#define ANKLE_L 2
#define HIP_R   3
#define KNEE_R  4
#define ANKLE_R 5
typedef struct {
  float h1, k1, a1;
  float h2, k2, a2;
} JointAngles;
extern QueueHandle_t cmdQueue;
extern QueueHandle_t servoQueue;
void TaskBluetooth(void *pvParameters);
void TaskGait(void *pvParameters);
void TaskServo(void *pvParameters);
int angleToPulse(float angle);
```

ProyectoBipedo.cpp

```
#include "ProyectoBipedo.h"


Adafruit_PWMServoDriver pwm(0x40);
QueueHandle_t servoQueue;
QueueHandle_t cmdQueue;
int angleToPulse(float angle) {
  return map(angle, 0, 180, 100, 500);
}
void TaskServo(void *pvParameters) {
  JointAngles j;
  while (1) {
    if (xQueueReceive(servoQueue, &j, portMAX_DELAY)) {
      pwm.setPWM(HIP_L, 0, angleToPulse(j.h1));
      pwm.setPWM(KNEE_L, 0, angleToPulse(j.k1));
      pwm.setPWM(ANKLE_L, 0, angleToPulse(j.a1));
      pwm.setPWM(HIP_R, 0, angleToPulse(j.h2));
      pwm.setPWM(KNEE_R, 0, angleToPulse(j.k2));
      pwm.setPWM(ANKLE_R, 0, angleToPulse(j.a2));
    }
  }
}
void TaskGait(void *pvParameters) {
  char cmd;
  JointAngles walk = {90, 70, 30, 90, 70, 30};
  while (1) {
    if (xQueueReceive(cmdQueue, &cmd, portMAX_DELAY)) {
      if (cmd == 'F') {
        xQueueSend(servoQueue, &walk, portMAX_DELAY);
      }
    }
  }
}
```

ComunicacionBT.cpp

```
#include "ProyectoBipedo.h"


BluetoothSerial SerialBT;
void TaskBluetooth(void *pvParameters) {
  SerialBT.begin("Robot_Bipedo");
  char cmd;
  while (1) {
    if (SerialBT.available()) {
      cmd = SerialBT.read();
      xQueueSend(cmdQueue, &cmd, portMAX_DELAY);
    }
    vTaskDelay(pdMS_TO_TICKS(20));
  }
}
```

main.ino

```
#include "ProyectoBipedo.h"


void setup() {
  Wire.begin();
  pwm.begin();
  pwm.setPWMFreq(SERVO_FREQ);
  servoQueue = xQueueCreate(5, sizeof(JointAngles));
  cmdQueue = xQueueCreate(5, sizeof(char));
  xTaskCreate(TaskBluetooth, "BT", 4096, NULL, 2, NULL);
  xTaskCreate(TaskGait, "Gait", 4096, NULL, 2, NULL);
  xTaskCreate(TaskServo, "Servo", 4096, NULL, 3, NULL);
}
void loop() {}
```

Comunicación entre tareas

La comunicación entre tareas se realiza mediante:

- **Colas:** paso de datos de sensores.
- **Mutex:** protección del bus I2C.
- **Semáforos:** sincronización de eventos.

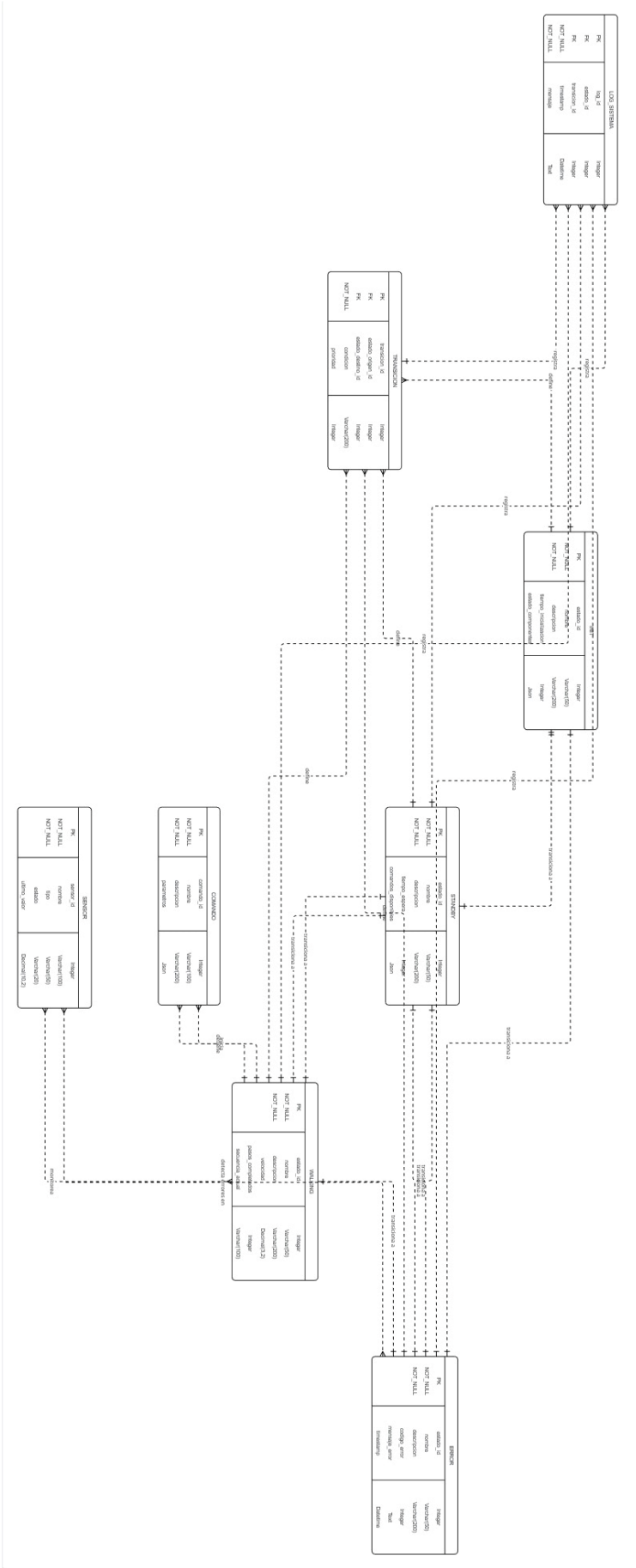
Esto garantiza integridad de datos y operación segura.

Máquina de estados del robot bípedo

El sistema implementa una **máquina de estados finitos (FSM)** con los siguientes estados:

1. INIT
2. CALIBRATION
3. STANDBY
4. WALKING
5. ERROR

Cada estado define claramente qué tareas están habilitadas.



Asignación de prioridades y temporización

La correcta asignación de prioridades permite:

- Evitar bloqueos.
- Garantizar ejecución crítica.
- Mantener estabilidad del robot.

Las tareas de control tienen mayor prioridad que las de comunicación.

Integración con el ESP32

FreeRTOS se ejecuta de forma nativa sobre el ESP32, aprovechando:

- Núcleos múltiples.
- Timers de hardware.
- Interrupciones.

Esto mejora la respuesta en tiempo real del sistema.

Seguridad y manejo de errores

El sistema incluye:

- Detección de fallos de comunicación I2C.
- Límites de ángulo.
- Parada de emergencia mediante cambio de estado.

CONCLUSIONES

- Se logró diseñar e implementar un robot bípedo de seis grados de libertad utilizando un sistema embebido basado en el microcontrolador ESP32 y el sistema operativo de tiempo real FreeRTOS.
- El uso de FreeRTOS permitió una correcta organización del software mediante tareas concurrentes, mejorando la estabilidad, escalabilidad y mantenimiento del sistema.
- La implementación de comunicación Bluetooth posibilitó el control remoto del robot desde un teléfono móvil, demostrando la integración efectiva entre sistemas embebidos y dispositivos externos.
- La simulación previa en MATLAB permitió validar el modelo cinemático y las trayectorias de movimiento antes de la implementación física, reduciendo errores mecánicos y de control.
- El proyecto cumplió satisfactoriamente los objetivos académicos de la asignatura Sistemas Embebidos I, integrando hardware, software y modelado matemático en un sistema funcional.

RECOMENDACIONES

- Implementar algoritmos de control dinámico más avanzados para mejorar el equilibrio del robot durante la marcha.
- Sustituir los servomotores por actuadores de mayor precisión para mejorar la repetibilidad del movimiento.
- Incorporar sensores adicionales (como sensores de presión en los pies) para mejorar la estabilidad.
- Desarrollar una aplicación móvil dedicada para un control más intuitivo del robot.
- Ampliar la simulación a modelos dinámicos en MATLAB/Simulink.

BIBLIOGRAFIA

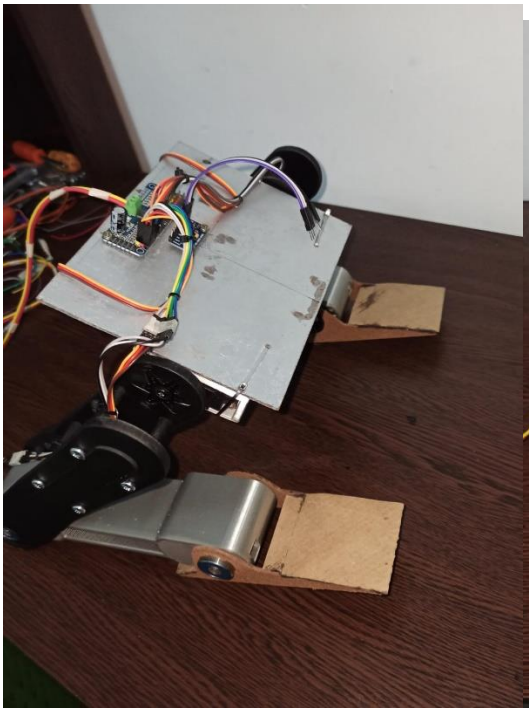
[1] John J. Craig, *Robótica*, Editorial Pearson, 3era Edición, 2009.

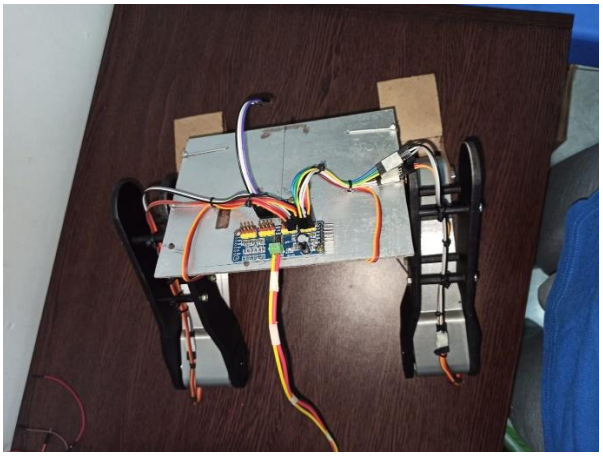
[2] M. W. Spong, S. Hutchinson, M. Vidyasagar, *Robot Modeling and Control*, Editorial Wiley, 1era Edición, noviembre 2005.

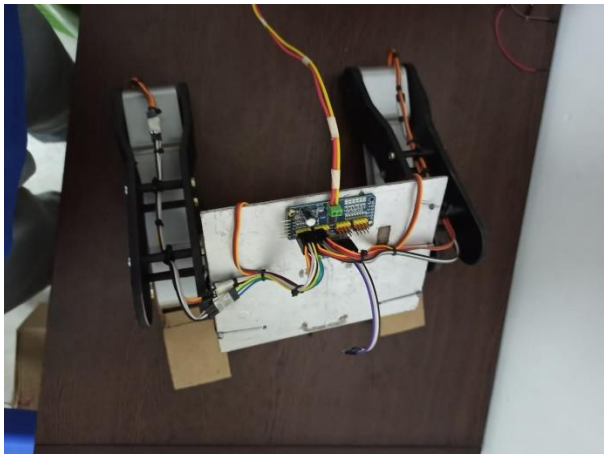
[3] Robert L. Boylestad y Louis Nashelsky, *Teoría de circuitos y dispositivos electrónicos*, Editorial Pearson, 2010.

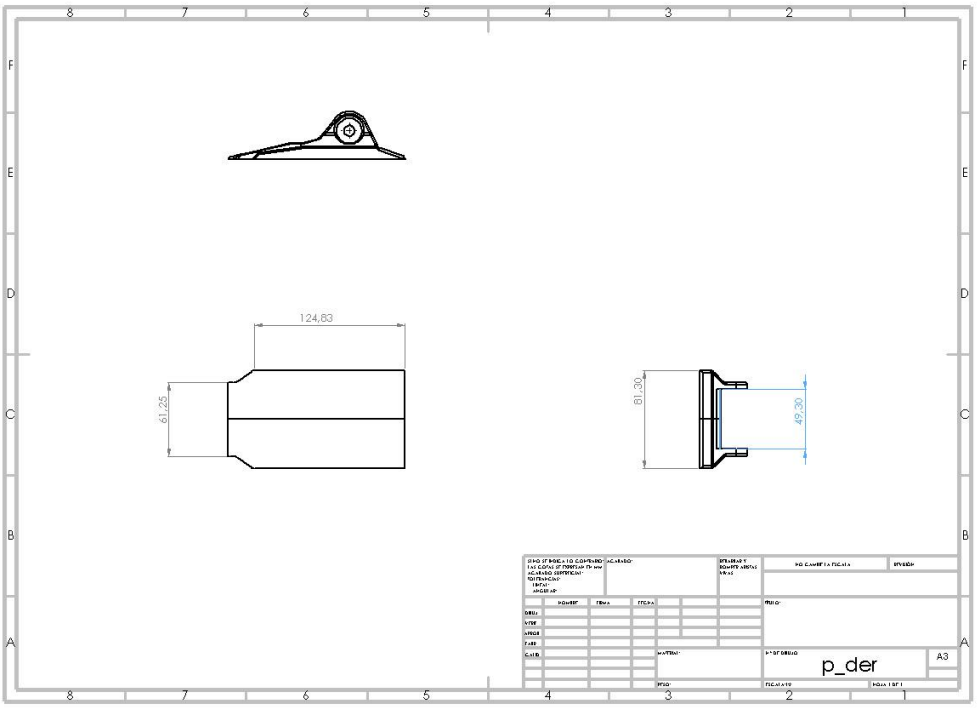
[4] William H. Hayt Jr., Jack E. Kemmerly y Steven M. Durbin, *Análisis de circuitos en ingeniería*, Sexta edición, Editorial McGraw-Hill.

Anexos








124,83
61,25
81,30
49,30
p_der
A3