RITCAS | REVISTA DE INVESTIGACIÓN TECNO-CIENTÍFICA ACADÉMICA SINERGIA

Ciencias de la Ingeniería Mecatrónica (CIMT)

Desarrollo de Sistemas de Control para Robots mediante ESP32 y Modelado Cinemático

Development of Control Systems for Robots using ESP32 and Kinematic Modeling

Juan Carlos Grijalva-Acuña [a, b, c *], Rogelio Acedo-Ruíz [a, b], Jorge Oswaldo Rivera-Nieblas [a, b, c], Eduardo Chávez-Mendiola [a, b, c], Vannia Gabriela Yanez-Guijarro [b]

[a] *Departamento de Eléctrica, Electrónica, Biomédica y Semiconductores, Tecnológico Nacional de México, Campus Hermosillo, Ave. Tecnológico 115, Col. Sahuaro, C.P. 83170 Hermosillo, Sonora, México.*

[b] *Carrera de Ingeniería Electrónica, Tecnológico Nacional de México Campus Hermosillo, Ave. Tecnológico 115, Col. Sahuaro, C.P. 83170 Hermosillo, Sonora, México.*

[c] *Carrera de Ingeniería Mecatrónica, Tecnológico Nacional de México Campus Hermosillo, Ave. Tecnológico 115, Col. Sahuaro, C.P. 83170 Hermosillo, Sonora, México.*

Correo electrónico: juan.grijalvaa@hermosillo.tecnm.mx

(Recibido: 22 de marzo 2025; Aceptado: 29 de abril 2025; Publicado: 01 de mayo 2025)

Resumen

El desarrollo de sistemas de control en robótica ha avanzado significativamente gracias a la integración de microcontroladores avanzados y modelos cinemáticos de alta precisión. Este estudio presenta el diseño e implementación de un sistema de control basado en el microcontrolador ESP32®, enfocado en la programación y control de actuadores en robótica. Se utilizaron servomotores, motores a pasos y motores de corriente continua con encoders de efecto Hall para garantizar precisión en el movimiento, implementando modelos de control como el Método de Denavit-Hartenberg para el modelado cinemático. Además, se desarrollaron algoritmos de control en tiempo real para optimizar la respuesta del sistema a diferentes tipos de cargas y condiciones operativas. Se incluyen códigos de programación tanto para la interfaz como para el control de los actuadores en Arduino IDE® y Visual Studio® con Windows Forms® en C#. Los resultados muestran que la combinación de estos elementos permite un control eficiente y adaptable en aplicaciones robóticas.

***Palabras claves**: Robótica, Control de actuadores, ESP32, Denavit-Hartenberg, Cinemática, Control en tiempo real.*

Abstract

The development of control systems in robotics has advanced significantly thanks to the integration of advanced microcontrollers and high-precision kinematic models. This study presents the design and implementation of a control system based on the ESP32 microcontroller, focused on the programming and control of actuators in robotics. Servo motors, stepper motors and direct current motors with Hall effect encoders were used to ensure precision in movement, implementing control models such as the Denavit-Hartenberg method for kinematic modeling. In addition, real-time control algorithms were developed to optimize the system's response to different types of loads and operating conditions. Programming code is included for both the interface and control of the actuators in the Arduino IDE and Visual Studio with Windows Forms in C#. The results show that the combination of these elements allows efficient and adaptable control in robotic applications.

***Keywords:** Robotics, Actuator Control, ESP32, Denavit-Hartenberg, Kinematics, Real-Time Control.*

1. Introducción

El desarrollo de robots con control preciso es un área clave en la ingeniería mecatrónica. La integración de microcontroladores de alto rendimiento, como el ESP32, permite un control eficiente de actuadores, lo que es fundamental para la implementación de sistemas robóticos en diversas aplicaciones [1].

En este estudio se exploran las técnicas de programación y modelado cinemático utilizadas para la gestión de movimientos en robots, abordando la implementación de motores de precisión y su control mediante software optimizado.

El uso de algoritmos de control en tiempo real, como el control PID adaptativo, permite una compensación dinámica de errores, mejorando la estabilidad y precisión del sistema [2]. Asimismo, la integración de sensores para la realimentación del sistema optimiza la respuesta del robot a estímulos externos, permitiendo una interacción más eficiente con su entorno.

2. Desarrollo

Para la implementación del sistema de control, se siguieron los siguientes pasos:

1) **Selección de actuadores:** Se emplearon servomotores de alto torque y motores a pasos para garantizar movimientos precisos en las articulaciones del robot.

2) **Plataforma de control:** Se utilizó el ESP32 debido a su capacidad de procesamiento en tiempo real, comunicación Bluetooth y Wi-Fi, y bajo consumo energético.

3) **Programación de controladores:** Se desarrollaron algoritmos en lenguaje C para la gestión del movimiento de los actuadores mediante señales PWM y control de corriente. Se implementaron técnicas avanzadas como el control PID adaptativo para mejorar la estabilidad del sistema [3].

4) **Modelado cinemático:** Se implementó el método de Denavit-Hartenberg para describir matemáticamente los movimientos del robot y optimizar la trayectoria de sus articulaciones. [4] [5] [6] [7] [8] [9]

5) **Implementación de comunicación:** Se integraron protocolos de comunicación inalámbrica para la gestión remota del robot a través de dispositivos externos, permitiendo la supervisión en tiempo real.

6) **Sensores y retroalimentación:** Se incorporaron sensores de posición y corriente para la realimentación del sistema, mejorando la precisión y respuesta de los actuadores ante variaciones en la carga. [10] [11]

7) **Pruebas funcionales:** Se realizaron pruebas para evaluar la precisión, estabilidad y respuesta del sistema en diferentes condiciones operativas, incluyendo entornos de carga variable.

3. Metodología

El desarrollo técnico de esta investigación se centra en la implementación del código con la Arquitectura del Sistema en Visual Studio C#, siguiendo los siguientes apartados.

3.1 Arquitectura del Sistema en Visual Studio C#

Este proyecto de Windows Forms implementa un sistema de cálculo cinemático para robots 3DOF usando Visual Studio como IDE principal. Su estructura aprovecha características clave del ecosistema .NET:

3.1.1 Gestión de Paquetes y Dependencias

El código utiliza namespaces estándar de Windows Forms y cálculo básico:

- using System.Windows.Forms; // Para elementos de UI.
- using System.IO.Ports; // Control de puertos COM.
- using System; // Funciones matemáticas.

Los parámetros D-H se procesan con conversiones nativas de C# sin librerías externas.

3.1.2 Diseño de la Interfaz Gráfica

Creada con el Diseñador de Windows Forms de Visual Studio:

- TextBox para entrada de parámetros D-H.
- RadioButton para selección cinemática directa/inversa.
- Label dinámicos que cambian entre θ y XYZ.
- Button para ejecutar cálculos.

3.1.3 Flujo de Datos del Sistema

```
text
graph TD
    A[Interfaz Gráfica]--> B[Validación de Entradas]
    B --> C[Conversión a Radianes]
    C --> D[Construcción Matricial D-H]
    D --> E[Multiplicación Manual de Matrices 4x4]
    E --> F[Extracción Posición/Orientación]
    F --> G[Serialización para Puerto COM]
```

3.1.4 Detalles Clave del Código Original

El código original presenta cuatro detalles clave:

1) Configuración Serial (Puerto COM)

```
puertoSerial = new
SerialPort(txbxPuerto.Text, 115200); //Lee el nombre del puerto desde un TextBox y usa baud rate estándar para microcontroladores (115200)
```

2) Conversión Angular Automática

```
double alpha1 = Math.PI * Convert.ToDouble(txbxAlpha1.Text) / 180;
```

Implementa conversión grados→radianes in situ

Usa Math.PI de la librería estándar de C#

3) Matrices de Transformación Homogénea

Estructura manual de matrices 4x4:

```
double[,] matrizT1 = {
{ Math.Cos(theta1), -Math.Sin(theta1)*Math.Cos(alpha1), ... },
{ Math.Sin(theta1), Math.Cos(theta1)*Math.Cos(alpha1), ... },
// Elementos restantes...
};
```

Cada fila representa:

- Rotación y escalado (R)
- Traslación (P)
- Perspectiva (0)
- Escala (1)

4) Multiplicación Matricial Manual

Implementación "cruda" de multiplicación:

```
double A11 = matrizT1[0,0]*matrizT2[0,0] + matrizT1[0,1]*matrizT2[1,0] + ...;
```

Calcula cada elemento de la matriz resultante individualmente.

Sigue el algoritmo estándar $O(n^3)$ para matrices 4x4

3.1.5 Gestión de Eventos de UI

Mecanismo de Windows Forms para cambiar etiquetas:

```
private void rdbtCinematicaDirecta_CheckedChanged(object sender, EventArgs e)
{
  lbArticulacion1.Text = "θ1"; // Actualización dinámica
}
```

Usa el patrón observer-subject nativo de .NET

Esta implementación demuestra cómo desarrollar un núcleo matemático robusto para robótica usando solo las capacidades nativas de C# y Windows Forms, siguiendo prácticas estándar de desarrollo en Visual Studio.

El código sirve tanto para fines educativos como de prototipado rápido en aplicaciones mecatrónicas.

3.2 Arquitectura General del Código

Este *sketch* implementa un sistema de control multi-actuador para robots 3DOF, diseñado para integrarse con una interfaz C# mediante comunicación serial. Su estructura combina tres tecnologías de actuación en un mismo microcontrolador. Esta arquitectura se puede desglosar por Bloques Funcionales de la siguiente manera.

3.2.1 Inclusión de Librerías y Definición de Pines

```
#include <Servo.h>  // Control de servomotores

// Definición de pines para actuadores
#define ENCODER_A1 34  // Entrada A del encoder óptico (GPIO34)
#define ENCODER_B1 35  // Entrada B del encoder (GPIO35, entrada analógica)
#define PWM_DC1 25     // Salida PWM para motor DC (GPIO25, DAC1)
#define DIR_DC1 26     // Dirección motor DC (GPIO26)

#define STEP_PIN1 14  // Paso motor pasos (GPIO14)
#define DIR_PIN1 27     // Dirección motor pasos (GPIO27)
#define ENABLE_PIN1 33 // Habilitación driver (GPIO33)

#define SERVO_PIN1 32   // PWM servomotor (GPIO32, salida PWM)
```

- **Servo.h:** Permite generar señales PWM compatibles con servomotores estándar3
- **GPIO ESP32:** Selección estratégica de pines:
  - **Pines 34-35:** Entradas analógicas para encoder.
  - **Pines 25-27:** Salidas digitales/PWM de propósito general.
  - **Pin 32:** Canal PWM independiente para servos. [12]

3.2.2 Configuración Inicial (Setup)

```
void setup() {
  Serial.begin(115200); // Configuración UART a 115200 baudios

  // Configuración motor DC
  pinMode(PWM_DC1, OUTPUT);
  pinMode(DIR_DC1, OUTPUT);
```

```
// Configuración motor pasos
pinMode(STEP_PIN1, OUTPUT);
pinMode(DIR_PIN1, OUTPUT);
pinMode(ENABLE_PIN1, OUTPUT);
digitalWrite(ENABLE_PIN1, LOW); // Activa driver ULN2003

// Inicialización servo
servo1.attach(SERVO_PIN1); // Asigna pin PWM al servo
}
```

- **Serial.begin():** Velocidad fija para sincronía con software C#. [13]
- **Habilitación driver:** LOW activa el driver ULN2003 (lógica activa baja).
- **attach():** Crea canal PWM dedicado para el servo (frecuencia ~50Hz). [12]

3.2.3 Recepción de Datos Seriales

```
void loop() {
  if (Serial.available() > 0) {
    receivedData = Serial.readStringUntil('\n');
    processSerialData(receivedData);
  }
}
```

- **readStringUntil('\n'):** Lee hasta salto de línea (formato requerido por C#).
- **Buffer String:** Almacena hasta 256 caracteres por defecto (modificable).

3.2.4 Procesamiento de Comandos

```
void processSerialData(String data) {
  int firstComma = data.indexOf(',');
  //... (búsqueda de comas)

  theta1 = data.substring(0, firstComma).toFloat();
  //... (extracción de valores)

  actuationMode = data.substring(thirdComma + 1).toInt();

  controlActuators();
}
```

- **Formato requerido:** "θ1,θ2,θ3,modo\n" (ej: "45.0,30.0,15.0,0\n").
- **indexOf():** Localiza separadores para dividir la cadena.
- **toFloat()/toInt():** Conversión a tipos numéricos.

3.2.5 Control de Actuadores

1) Servomotor (Modo 0)

```
void controlServo(float angle) {
  int servoPos = map(angle, -90, 90, 0, 180);
  servo1.write(servoPos);
}
```

- **map():** Adapta rango angular (-90° a 90°) a pulsos PWM (0-180)
- **write():** Genera pulso de 0.5-2.5ms (estándar servos). [12]

2) Motor DC (Modo 1)

```
void controlDCMotor(float speed) {
  int pwmValue = map(speed, -100, 100, 0, 255);

  digitalWrite(DIR_DC1, (speed >= 0) ? HIGH: LOW);
  analogWrite(PWM_DC1, abs(pwmValue));
}
```

- **PWM 8-bit:** Rango 0-255 para control L298N. [13]
- **DIR pin:** Determina dirección de giro (HIGH/LOW)
- **analogWrite():** Usa canales LEDC del ESP32 (8, 15, 20 bits)

3) Motor a Pasos (Modo 2)

```
void controlStepperMotor(float steps) {
  digitalWrite(DIR_PIN1, (steps >= 0) ? HIGH: LOW);

  for(int i=0; i<stepsToMove; i++) {
    digitalWrite(STEP_PIN1, HIGH);
    delayMicroseconds(500);
    digitalWrite(STEP_PIN1, LOW);
    delayMicroseconds(500);
  }
}
```

- **Lógica de paso:** Genera pulsos cuadrados para driver ULN2003.
- **delayMicroseconds(500):** Velocidad fija de ~1000 pasos/segundo.
- **Bloqueante:** Mantiene ocupado el bucle durante el movimiento.

3.2.6 Protocolo de Comunicación C#-ESP32

```
text
Formato: θ1,θ2,θ3,modo\n
Ejemplo: 45.5,-30.0,15.75,2\n
```

- **θ1-θ3:** Valores float (ángulos en grados).
- **modo:** Entero (0=servo, 1=DC, 2=stepper).
- **Terminador:** '\n' obligatorio para finalizar trama.

Este código representa una **base funcional** para sistemas robóticos educativos o prototipos, permitiendo rápida integración con software de control superior (C#). Su diseño modular facilita la expansión a más articulaciones o tipos de actuadores

4. Resultados

Los resultados obtenidos confirman que el uso del ESP32 en el control de actuadores permite una ejecución fluida y precisa de movimientos robóticos. La programación optimizada de señales PWM y controladores PID mejoró la estabilidad del sistema, reduciendo errores en la trayectoria de los actuadores. Además, el uso de retroalimentación de sensores permitió ajustar dinámicamente los parámetros de control, optimizando la eficiencia del sistema en tiempo real.

El método de Denavit-Hartenberg facilitó el modelado y control cinemático del robot, permitiendo calcular trayectorias eficientes y optimizar el rendimiento del sistema, como se ve en la figura 1.

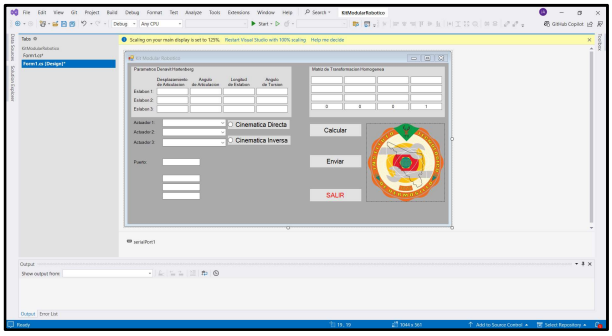

Fig. 1 Desarrollo de Interfaz de Control.

En la figura 2 se muestra la interfaz de Resultados al método de Denavit-Hartenberg.

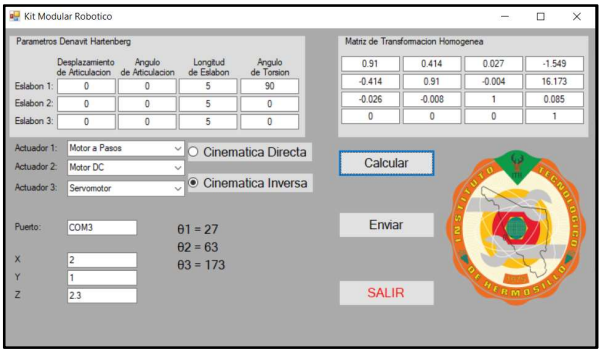

Fig. 2 Interfaz de Resultados

Además, la integración de comunicación posibilitó el monitoreo y ajuste remoto del robot, incrementando su versatilidad en entornos industriales y educativos. Se observó que la implementación de sensores de realimentación mejoró la precisión en la ejecución de tareas repetitivas, permitiendo compensar perturbaciones externas.

Consideraciones de Performance en Visual Studio C#:

- Método Convert.ToDouble() vs double.TryParse().
- Multiplicación matricial manual vs algoritmos optimizados.
- Uso de tipos double para precisión en cálculos trigonométricos.

Consideraciones de Implementación en Arduino IDE®:

1) Limitaciones Actuales:

a) **Sincronización Serial:** No hay ACK/NACK para confirmar recepción.
b) **Control no Bloqueante:** El motor a pasos detiene otros procesos.
c) **Seguridad Eléctrica:** Faltan protecciones contra cortos.
d) **Precisión Stepper:** No considera micropasos o aceleración.

2) Optimizaciones Sugeridas:

```
// Ejemplo: Uso de interrupciones para encoder
void IRAM_ATTR encoderISR() {
  // Lógica de conteo aquí
}
attachInterrupt(digitalPinToInterrupt(ENCODER_A1), encoderISR, CHANGE);
```

a) **LEDC para PWM:** Mayor resolución en motores DC.
b) **Protocolo binario:** Mejor eficiencia que strings.
c) **Colas de movimiento:** Para control asíncrono.

5. Conclusiones

Este estudio demuestra que el uso del ESP32 en el control de actuadores robóticos es una solución eficiente para el desarrollo de sistemas mecatrónicos avanzados. La combinación de motores de precisión, modelado cinemático, algoritmos de control adaptativo y sensores de realimentación permite optimizar el desempeño del sistema, facilitando su aplicación en diversas áreas de la robótica. Además, la

implementación de una interfaz gráfica en Visual Studio Windows Forms® permite una interacción más intuitiva y una comunicación eficiente con el sistema embebido.

Futuras investigaciones pueden centrarse en la integración de Inteligencia Artificial para mejorar la adaptabilidad del robot, así como en la implementación de sensores avanzados para incrementar la autonomía y capacidad de percepción del sistema.

Características de Visual Studio® Explotadas:

- **Auto-generated code:** InitializeComponent() gestionado por el IDE.
- **Control de errores:** Conversiones con Convert.ToDouble() sin try/catch.
- **Depuración:** Breakpoints posibles en cálculos matriciale.s
- **Diseñador de UI:** Alineación automática de controles.

6. Agradecimientos

Agradecemos al Tecnológico Nacional de México Campus Hermosillo por todas las facilidades prestadas y al PRODEP por el Fortalecimiento a Cuerpos Académicos.

7. Referencias

[1] Espressif Inc., «ESP32 Series Datsheet,» Espressif Systems, 2019. [En línea]. Available: https://www.alldatasheet.com/datasheet-pdf/view/1148023/ESPRESSIF/ESP32.html.

[2] M. W. Spong, S. Hutchinson y M. Vidyasagar, Robot Modelling and Control, First ed., John Wiley & Sons Inc, 2005.

[3] W. Bolton, Mecatrónica: sistemas de control electrónico en la ingenería mecánica y eléctrica, 6ta ed., Alfaomega, 2017.

[4] J. Denavit y R. S. Hartenberg, «A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices,» Journal of Applied Mechanics, vol. 22, nº 2, pp. 215-221, 01 06 1955.

[5] J. J. Craig, Introduction to Robotics: Mechanics and Control, Fourth ed., Harlow: Pearson Education, 2022.

[6] B. Siciliano y O. Khatib, Edits., Springer Handbook of Robotics, 2nd ed., Springer, 2017.

[7] A. Barrientos, L. F. Peñin, C. Balaguer y R. Aracil, Fundamentos de Robótica, 2da Edición ed., Aravaca: McGraw-Hill/Interamericana de España, 2007.

[8] S. K. Saha, Introducción a la Robótica, Ciudad de México: McGraw-Hill/Interamericana, 2010.

[9] P. I. Corke, «A simple and systematic approach to assigning Denavit–Hartenberg parameters,» IEEE transactions on robotics, vol. 23, nº 3, pp. 590-594, 01 06 2007.

[10] A. A. Hayat , R. G. Chittawadigi, A. D. Udai y S. K. Saha, «Identification of Denavit-Hartenberg Parameters of an Industrial Robot,» AIR, pp. 1-6, 04 07 2013.

[11] C. E. Conejo Benitez, Tesis de Maestria: Modelado y control de un brazo Robótico de 3 grados de libertad, Salamanca, Guanajuato, 2021.

[12] MertArduino, «instructables.com,» AUTODESK, 2023. [En línea]. Available: https://www.instructables.com/ESP32-Servo-Motor-Controller-Board-Wireless-Contro/. [Último acceso: febrero 2025].

[13] Random Nerd Tutorials, «randomnerdtutorials.com,» 11 06 2024. [En línea]. Available: https://randomnerdtutorials.com/esp32-dc-motor-l298n-motor-driver-control-speed-direction/. [Último acceso: 05 02 2025].