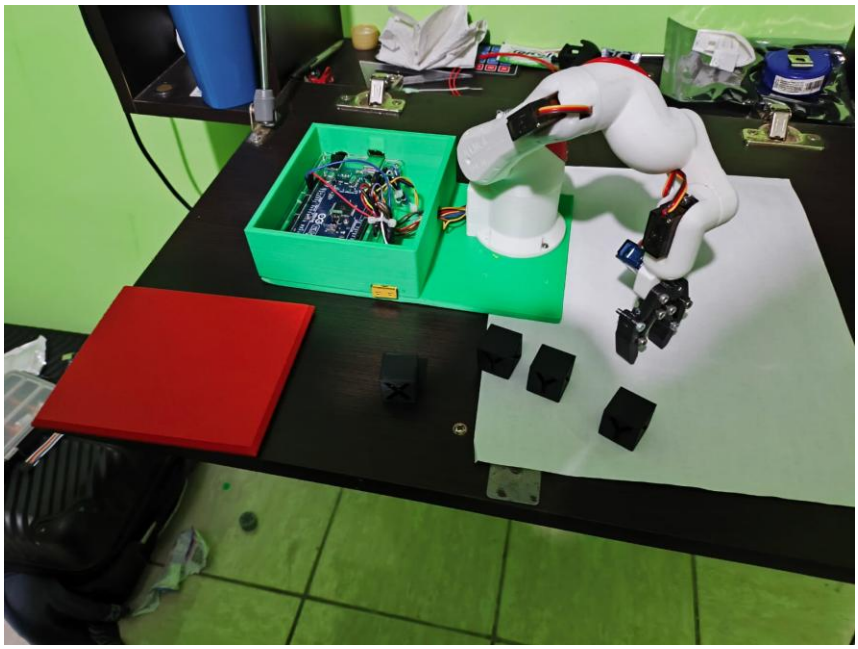
Control de un Brazo Robótico 3D con Interfaz Gráfica en Visual C#



Autores

Nombre: Miguel Angel Urbina Gonzalez
Universidad: Tecnológico Nacional de México, Campus Reynosa.
Carrera: Ingeniería en Mecatrónica, 2º semestre
Fecha de entrega: 12 de mayo de 2025

Resumen

En esta práctica se desarrolló un sistema de control para un brazo robótico impreso en 3D, compuesto por 4 grados de libertad más un gripper. El brazo fue manipulado mediante una interfaz gráfica desarrollada en Visual C# estilo HMI (Human Machine Interface), permitiendo enviar señales a cinco servomotores a través de comunicación serial con Arduino. Se implementaron funciones como "Origen", "Guardar posición" y "Reproducir posición". Los resultados demostraron que el sistema responde de manera precisa y confiable a los comandos enviados, logrando una experiencia real de control robótico básico.

Introducción

La robótica es una rama esencial de la mecatrónica. Integrar componentes electrónicos con impresión 3D permite desarrollar prototipos funcionales y económicos. En esta práctica, se diseñó un brazo robótico con 4 grados de libertad y un gripper, todos operados por servomotores. El sistema se controló mediante una interfaz en C# similar a un HMI industrial.

"Arduino es una plataforma abierta basada en hardware y software libre que facilita el aprendizaje de la programación y la electrónica" (Banzi, 2011).

Objetivo General:
Diseñar e implementar un sistema de control de un brazo robótico 3D de 5 ejes mediante una interfaz en Visual C#, usando Arduino como núcleo de control y servomotores para la articulación.

Materiales y Métodos

Materiales utilizados:

- 4 servomotores **MG90S** (para articulaciones M1 a M4)
- 1 servomotor **SG90** (para gripper)
- Estructura del brazo robótico impresa en **3D** (PLA)
- Placa **Arduino (MEGA o UNO)**
- Conector **Jack** para alimentación externa (5V 2A)
- Fuente de alimentación externa
- Cables Dupont
- PC con **Visual Studio** y **Arduino IDE**

Método y desarrollo:

Diseño del sistema

Se conectaron los 5 servomotores a los pines PWM de la placa Arduino. La alimentación se gestionó por un jack externo para evitar sobrecargas por USB.
La interfaz HMI se programó en Visual C# como se muestra a continuación:

Interfaz gráfica (Visual C#):

Contiene 5 sliders para controlar M1, M2, M3, Base y Gripper, además de botones funcionales:

- **Origen**: Lleva el brazo a su posición inicial.
- **Guardar Posición**: Registra la posición actual.
- **Reproducir Posición**: Ejecuta el movimiento guardado.
- **Salir**: Cierra la interfaz.

(Ver imagen adjunta al final del reporte)

Código Arduino (fragmento):

```
cpp
CopiarEditar
#include <Servo.h>
Servo m1, m2, m3, base, gripper;

void setup() {
 m1.attach(3);
 m2.attach(5);
 m3.attach(6);
 base.attach(9);
 gripper.attach(10);
 Serial.begin(9600);
}

void loop() {
 if (Serial.available() >= 5) {
  m1.write(Serial.read());
```

```
  m2.write(Serial.read());
  m3.write(Serial.read());
  base.write(Serial.read());
  gripper.write(Serial.read());
 }
}
```

Código C# (envío serial):

```
csharp
CopiarEditar
private void EnviarPosiciones() {
  byte[] posiciones = new byte[5] {
    (byte)sliderM1.Value,
    (byte)sliderM2.Value,
    (byte)sliderM3.Value,
    (byte)sliderBase.Value,
    (byte)sliderGripper.Value
  };
  serialPort1.Write(posiciones, 0, 5);
}
```

Resultados

El sistema respondió satisfactoriamente a los comandos de la interfaz. Se logró una interacción fluida entre el operador y el brazo robótico.

Función	Resultado
Desplazamiento con sliders	Preciso y proporcional

Función	Resultado
Origen	Retorno exacto a posición base
Guardar y reproducir	Fiable, sin desfases
Comunicación serial	Estable y sin errores

Discusión

Durante la práctica, se identificó que el uso de servomotores MG90S fue acertado por su torque moderado y compatibilidad con estructuras 3D . Se requirió fuente externa por limitaciones de corriente del USB. La interfaz HMI simplificó el control y simula entornos industriales reales. La sincronización de los valores seriales fue un punto crítico que se resolvió agrupando los datos en un solo paquete.

Conclusiones

- Se logró implementar con éxito un sistema HMI para controlar un brazo robótico físico.
- Se reforzaron conocimientos sobre control por PWM, comunicación serial y diseño de interfaces gráficas.
- La experiencia permitió la validación de sistemas mecatrónicos funcionales a bajo costo.

Referencias

Banzi, M. (2011). *Getting Started with Arduino*. O'Reilly Media.
Monk, S. (2013). *Programming Arduino: Getting Started with Sketches*. McGraw-Hill.
De Souza, C. (2020). *Hands-On Robotics Programming with C#*. Packt Publishing.

Anexos

Código completo Arduino:

```
#include <Servo.h>

// Definición de los servos
Servo servoBase;     // ID = 1
Servo servoM1;       // ID = 2
Servo servoM2;       // ID = 3
Servo servoM3;       // ID = 4
Servo servoGripper;   // ID = 5
// Pines asignados (ajústalos si es necesario)
const int pinBase = 3;
const int pinM1 = 5;
const int pinM2 = 6;
const int pinM3 = 9;
const int pinGripper = 10;


void setup() {
 Serial.begin(115200);  // Velocidad para comunicación serial con C#


 // Conexión de servos a sus pines
 servoBase.attach(pinBase);
 servoM1.attach(pinM1);
 servoM2.attach(pinM2);
 servoM3.attach(pinM3);
 servoGripper.attach(pinGripper);
```

```
 // Posición inicial (opcional)
 servoBase.write(90);
 servoM1.write(90);
 servoM2.write(90);
 servoM3.write(90);
 servoGripper.write(45);  // Semicerrado
}

void loop() {
 // Esperar a que lleguen 2 bytes: [ID, Ángulo]
 if (Serial.available() >= 2) {
   byte id = Serial.read();
   byte angle = Serial.read();

   // Seguridad: limitar ángulo a 0–180
   angle = constrain(angle, 0, 180);

   // Ejecutar el movimiento del servo según el ID
   switch (id) {
     case 1:
       servoBase.write(angle);
       break;
     case 2:
       servoM1.write(angle);
       break;
     case 3:
```

```
      servoM2.write(angle);
      break;
    case 4:
      servoM3.write(angle);
      break;
    case 5:
      servoGripper.write(angle);
      break;
    default:
      // ID inválido, ignorar
      break;
  }
 }
}
```

Código C# (Form1.cs):

```
using System;
using System.IO.Ports;
using System.Windows.Forms;


namespace _2
{
  public partial class Form1 : Form
  {
    //  Declaración global de arduino
    SerialPort arduino;
```

```
List<byte[]> posicionesGuardadas = new List<byte[]>();


public Form1()
{
   InitializeComponent();

   // Inicializa el puerto serial
   arduino = new SerialPort("COM8", 115200);
   try
   {
      arduino.Open();
   }
   catch (Exception ex)
   {
      MessageBox.Show("Error al abrir el puerto COM8: " + ex.Message);
   }
}

private void trackBarBase_Scroll(object sender, EventArgs e)
{
   SendServoCommand(1, (byte)trackBarBase.Value);
}

private void trackBarArt1_Scroll(object sender, EventArgs e)
{
   SendServoCommand(2, (byte)trackBarArt1.Value);
```

```
}

private void trackBarArt2_Scroll(object sender, EventArgs e)
{
   SendServoCommand(3, (byte)trackBarArt2.Value);
}

private void trackBarArt3_Scroll(object sender, EventArgs e)
{
   SendServoCommand(4, (byte)trackBarArt3.Value);
}

private void trackBargripper_Scroll(object sender, EventArgs e)
{
   SendServoCommand(5, (byte)trackBargripper.Value);
}

// Método que envía comando al Arduino
private void SendServoCommand(byte id, byte angle)
{
   if (arduino != null && arduino.IsOpen)
   {
      try
      {
         byte[] data = new byte[] { id, angle };
         arduino.Write(data, 0, data.Length);
```

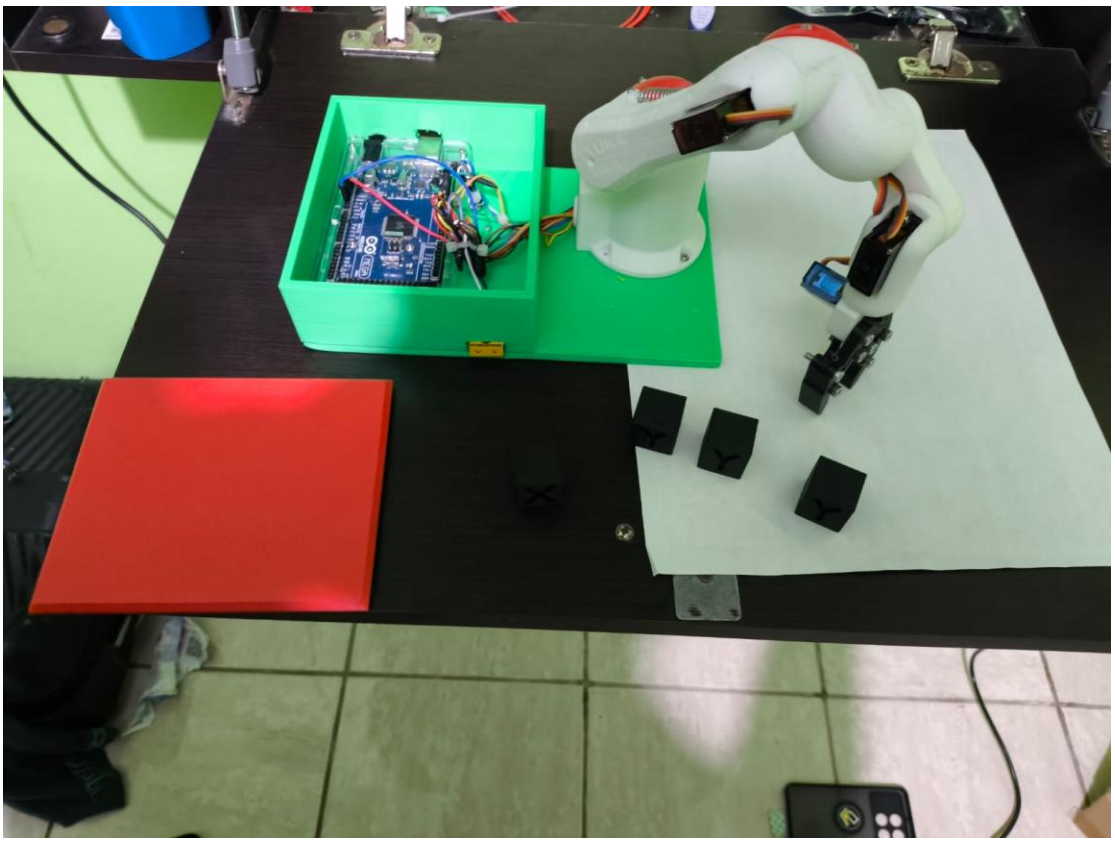
```
        }
        catch (Exception ex)
        {

        }
    }
}

private void button1_Click(object sender, EventArgs e)
{
    // Posiciones de origen sugeridas:
    byte baseStop = 90;      // Detener servo continuo
    byte artCenter = 90;     // Centro articulaciones
    byte gripperStart = 45;   // Semicerrado

    // Mueve los sliders visualmente
    trackBarBase.Value = baseStop;
    trackBarArt1.Value = artCenter;
    trackBarArt2.Value = artCenter;
    trackBarArt3.Value = artCenter;
    trackBargripper.Value = gripperStart;

    // Enviar comandos al Arduino
    SendServoCommand(1, baseStop);
    SendServoCommand(2, artCenter);
    SendServoCommand(3, artCenter);
```

```
        SendServoCommand(4, artCenter);
        SendServoCommand(5, gripperStart);
    }
    // BOTÓN: GUARDAR POSICIÓN ACTUAL
    private void button2_Click(object sender, EventArgs e)
    {
        byte[] nuevaPosicion = new byte[]
{
    (byte)trackBarBase.Value,
    (byte)trackBarArt1.Value,
    (byte)trackBarArt2.Value,
    (byte)trackBarArt3.Value,
    (byte)trackBargripper.Value
};

        posicionesGuardadas.Add(nuevaPosicion);
        MessageBox.Show("¡Posición guardada!");
    }
    // BOTÓN: REP POSICIÓN ACTUAL
    private void buttonreprovalores_Click(object sender, EventArgs e)
    {

        if (posicionesGuardadas.Count == 0)
        {
            MessageBox.Show("No hay posiciones guardadas.");
            return;
```

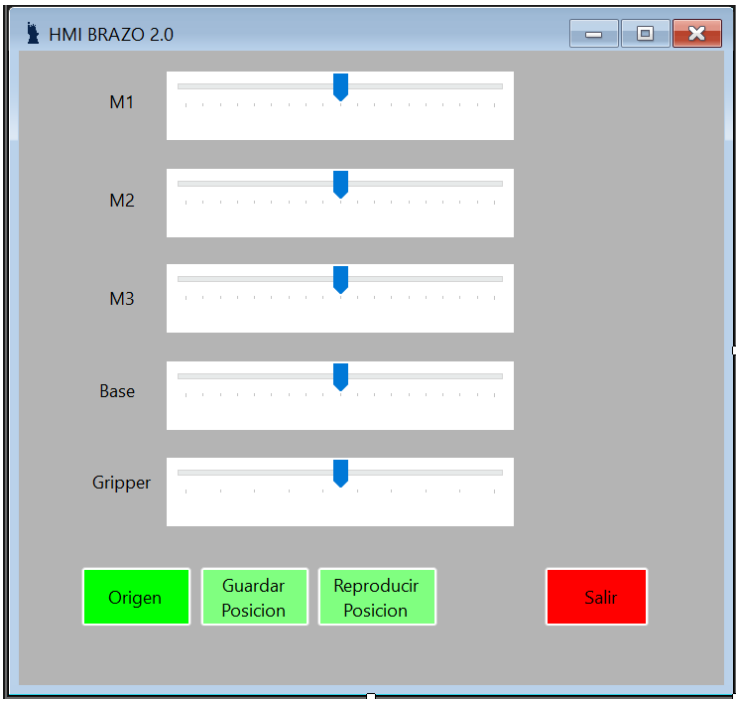
```
    }

    // Ejemplo: reproducir la PRIMERA posición guardada
    byte[] posicion = posicionesGuardadas[0];

    trackBarBase.Value = posicion[0];
    trackBarArt1.Value = posicion[1];
    trackBarArt2.Value = posicion[2];
    trackBarArt3.Value = posicion[3];
    trackBargripper.Value = posicion[4];

    SendServoCommand(1, posicion[0]);
    SendServoCommand(2, posicion[1]);
    SendServoCommand(3, posicion[2]);
    SendServoCommand(4, posicion[3]);
    SendServoCommand(5, posicion[4]);

}

private void Form1_Load(object sender, EventArgs e)
{
}
private void button2_Click_1(object sender, EventArgs e)
{
    Application.Exit();
}
```

```
        private void label6_Click(object sender, EventArgs e)
        {


        }
    }
}
```

Imágenes del proyecto:



Interfaz HMI utilizada:



Link a evidencia de practica:

https://drive.google.com/file/d/1rwjepGEJHf8ICzZk7gZEbTiHec4q-BNx/view?usp=drive_link