ITSRLL
INSTITUTO TECNOLÓGICO SUPERIOR DE LA REGIÓN DE LOS LLANOS

Ingeniería Mecatrónica

PROGRAMACIÓN AVANZADA

Enero – Junio 2025

M.C. Osbaldo Aragón Banderas

UNIDAD:

1	2	3	4	5

Actividad número: 3

Nombre de actividad:

U3A3. (20%) Clasificación de Posición en un Robot Seguidor de Línea Usando Árbol de Decisión e Implementación en Arduino

Actividad realizada por:

Muñiz Galvan Bryam 21030021

Guadalupe Victoria, Durango

Fecha de entrega:

05	04	2025

Objetivo

Aplicar técnicas de clasificación supervisada utilizando árboles de decisión para determinar la posición de un robot seguidor de línea a partir de un conjunto de datos históricos, generar un modelo funcional para su implementación en Arduino, y realizar las modificaciones necesarias para que el robot siga correctamente la trayectoria de una línea negra sobre superficie blanca.

Descripción del Problema

Se proporciona un dataset con datos históricos de un robot seguidor de línea que utiliza tres sensores digitales (ON/OFF) para detectar la posición de la línea. El conjunto de datos incluye las combinaciones de estados presentes y pasados de los sensores, así como la clase correspondiente a la posición del robot con respecto a la línea:

- Clases del 1 al 9: representan distintas posiciones relativas del robot respecto a la línea (por ejemplo, muy a la izquierda, centro, muy a la derecha, etc.)
- Clase 0: indica que el robot ha perdido completamente la línea.
- Clase 10: indica que el robot está en una posición no contemplada (error o transición anómala).

Instrucciones

Entrenamiento del Modelo de Clasificación:

- Cargar el dataset en Python utilizando pandas.
- Explorar y analizar los datos para entender la distribución de clases y características.
- Entrenar un modelo de clasificación basado en Árbol de Decisión (DecisionTreeClassifier de sklearn).

- Evaluar el modelo utilizando métricas como matriz de confusión, precisión general y precisión por clase.
- Exportar el árbol de decisión entrenado a formato visual y a código en texto plano (puede usarse sklearn.tree.export_text()).
- Conversión del Modelo a C para Arduino
- Convertir manualmente o con ayuda de herramientas el modelo a código C compatible con Arduino.
- Implementar la estructura de decisiones en código condicional (if-else) basado en las entradas actuales y anteriores de los sensores.
- Asignar acciones de control para los motores según la clase detectada:
- Por ejemplo, si la clase indica que el robot está a la izquierda de la línea, reducir la velocidad del motor izquierdo y aumentar la del derecho para girar.
- Si está al centro (clase ideal), ambos motores deben avanzar a velocidad constante.
- Implementación Física en el Robot
- Cargar el código al microcontrolador Arduino.
- Probar el robot en una pista con línea negra sobre fondo blanco.
- Realizar ajustes finos de las velocidades para lograr un seguimiento estable de la línea.

Dataset utilizado

s_i_p	s_c_p	s_d_p	s_i	s_c	s_d	accion
0	0	0	0	0	0	0
0	0	0	0	0	1	2
0	0	0	0	1	0	5
0	0	0	0	1	1	4
0	0	0	1	0	0	8
0	0	0	1	0	1	10
0	0	0	1	1	0	10
0	0	0	1	1	1	10
0	0	1	0	0	0	10
0	0	1	0	0	1	1
0	0	1	0	1	0	10
0	0	1	0	1	1	4
0	0	1	1	0	0	10
0	0	1	1	0	1	10
0	0	1	1	1	0	10
0	0	1	1	1	1	10

Este dataset representa un conjunto de lecturas de sensores y las acciones correspondientes que debe tomar un robot seguidor de línea en función de esas lecturas.

Columna	**Descripción**
s_i_p	Estado anterior del sensor izquierdo (0 = no detecta línea, 1 = detecta línea)
s_c_p	Estado anterior del sensor central
s_d_p	Estado anterior del sensor derecho
s_i	Estado actual del sensor izquierdo
s_c	Estado actual del sensor central
s_d	Estado actual del sensor derecho
accion	Código de acción determinada para el robot

Acciones (valor de acción):

Valor	**Descripción de la acción del robot**
0	Línea completamente perdida (default o parada)
1	Avanzar recto
2	Posiblemente un caso no contemplado o especial
4	Ajuste leve a la izquierda
5	Ajuste leve a la derecha
6	Giro fuerte a la derecha
8	Giro fuerte a la izquierda
10	Estado no reconocido o sin acción definida

Entrenamiento del modelo (Árbol de decisiones)

El dataset proporcionado, compuesto por las lecturas actuales y anteriores de los sensores de un robot seguidor de línea, fue utilizado como base para entrenar un modelo de árbol de decisión supervisado.

Objetivo:

El objetivo principal fue que el árbol de decisión aprenda la lógica de navegación del robot, prediciendo la acción correcta (movimiento) a partir de los valores binarios de los sensores.

Variables:

- **Entradas (features)**:
  - s_i_p, s_c_p, s_d_p: Lecturas anteriores de los sensores izquierdo, central y derecho.
  - s_i, s_c, s_d: Lecturas actuales de los sensores.
- **Salida (target)**:
  - acción: Código numérico que representa la acción a tomar (e.g., avanzar, girar, detenerse, etc.).

Proceso:

1. Se procesaron los datos como un conjunto de entrenamiento supervisado.
2. Se utilizó un algoritmo de árbol de decisión (por ejemplo, DecisionTreeClassifier de scikit-learn en Python).
3. El modelo fue entrenado con los datos etiquetados para aprender patrones lógicos entre las combinaciones de sensores y las acciones resultantes.
4. Se validó el modelo para comprobar su capacidad de reproducir decisiones coherentes con la lógica esperada del robot.

Ventajas del árbol de decisión:

- Genera reglas claras y fáciles de interpretar.
- Permite visualizar la lógica interna con un diagrama del árbol.
- Rápido para predecir en tiempo real en sistemas embebidos.

Notebook Jupyter

Clasificación de Posición en un Robot Seguidor de Línea Usando Árbol de Decisión

Objetivo

Aplicar técnicas de clasificación supervisada utilizando árboles de decisión para determinar la posición de un robot seguidor de línea a partir de un conjunto de datos históricos, generar un modelo funcional para su implementación en **Arduino**, y realizar las modificaciones necesarias para que el robot siga correctamente la trayectoria de una línea negra sobre superficie blanca.

Descripción del Problema

Se proporciona un dataset con datos históricos de un robot seguidor de línea que utiliza **tres sensores digitales (ON/OFF)** para detectar la posición de la línea. El conjunto de datos incluye combinaciones de estados **presentes y pasados** de los sensores, así como la clase correspondiente a la posición del robot respecto a la línea.

Las clases están definidas de la siguiente manera:

- **Clases 1 a 9:** Representan distintas posiciones relativas del robot respecto a la línea
  (por ejemplo: muy a la izquierda, ligeramente a la izquierda, centrado, ligeramente a la derecha, muy a la derecha, etc.)
- **Clase 0:** El robot ha perdido completamente la línea.
- **Clase 10:** El robot se encuentra en una posición no contemplada (error o transición anómala).

Descripción del Modelo de Entrenamiento

Carga y Análisis de Datos

El archivo `dataset_seguidor_3.csv` contiene **64 instancias**, cada una con **6 características** correspondientes a los valores de sensores infrarrojos (anteriores y actuales), y una columna objetivo llamada `accion`.

Columnas del Dataset:

- `s_i_p` : Sensor infrarrojo izquierdo posterior
- `s_c_p` : Sensor infrarrojo central posterior
- `s_d_p` : Sensor infrarrojo derecho posterior
- `s_i` : Sensor infrarrojo izquierdo actual
- `s_c` : Sensor infrarrojo central actual
- `s_d` : Sensor infrarrojo derecho actual
- `accion` : Clase objetivo que representa la acción o posición del robot

Preparación de los Datos

Se seleccionan las columnas de entrada (`X`) con los sensores, y la columna objetivo (`y`):

```
X = data[['s_i_p', 's_c_p', 's_d_p', 's_i', 's_c', 's_d']]
y = data['accion']
```

[1]:
```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier,export_text
from sklearn.model_selection import train_test_split
```

[2]:
```
data = pd.read_csv("dataset_seguidor_3.csv")
print(data)
```

```
    s_i_p  s_c_p  s_d_p  s_i  s_c  s_d  accion
0       0      0      0    0    0    0       0
1       0      0      0    0    0    1       3
2       0      0      0    0    1    0       5
3       0      0      0    0    1    1       4
4       0      0      0    1    0    0       0
..    ...    ...    ...  ...  ...  ...     ...
59      1      1      1    0    1    1       4
60      1      1      1    1    0    0      10
61      1      1      1    1    0    1      10
62      1      1      1    1    1    0       6
63      1      1      1    1    1    1       0

[64 rows x 7 columns]
```

Análisis de datos

[3]:
```
data.info()
```

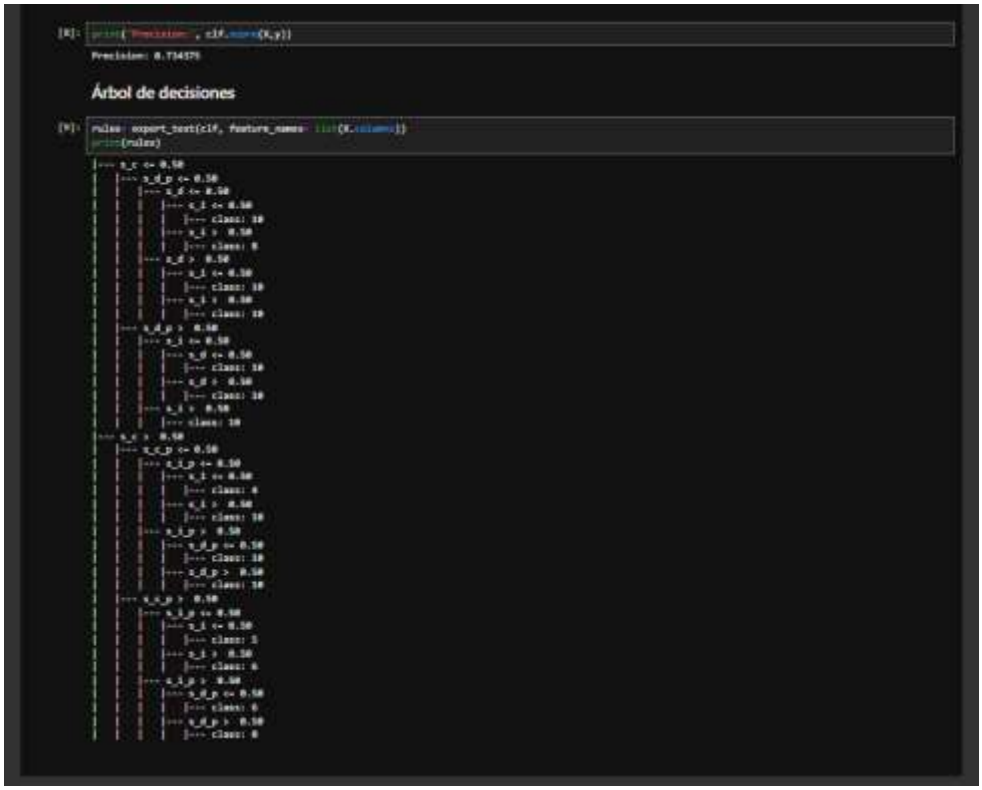
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 64 entries, 0 to 63
Data columns (total 7 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   s_i_p    64 non-null     int64
 1   s_c_p    64 non-null     int64
 2   s_d_p    64 non-null     int64
 3   s_i      64 non-null     int64
 4   s_c      64 non-null     int64
 5   s_d      64 non-null     int64
 6   accion   64 non-null     int64
dtypes: int64(7)
memory usage: 3.6 KB
```

Preparación de los datos

[4]:
```
X = data[['s_i_p', 's_c_p', 's_d_p', 's_i', 's_c', 's_d']]
y = data['accion']
```

[7]:
```
clf= DecisionTreeClassifier(max_depth=4)
clf.fit(X,y)
```

[7]:
```
DecisionTreeClassifier(max_depth=4)
```



Programa Arduino utilizado

```
const int pinSensorIzq = 11;

const int pinSensorCen = 12;

const int pinSensorDer = 10;

const int motorIzq = 3;

const int motorDer = 5;


// Ajuste de velocidades

int rapido = 220, muyRapido = 235, medioAlto = 210, medio =
200, lento = 140, muyLento = 90, stop = 0;


int s_i = 0, s_c = 0, s_d = 0;

int s_i_p = 0, s_c_p = 0, s_d_p = 0;


void setup() {

  pinMode(pinSensorIzq, INPUT);
```

```
  pinMode(pinSensorCen, INPUT);
  pinMode(pinSensorDer, INPUT);
  pinMode(motorIzq, OUTPUT);
  pinMode(motorDer, OUTPUT);
}

void loop() {
  // Leer sensores actuales
  s_i = digitalRead(pinSensorIzq);
  s_c = digitalRead(pinSensorCen);
  s_d = digitalRead(pinSensorDer);

  int classLabel = 10; // Por defecto: detenerse

  // === CASOS NORMALES DE DETECCIÓN DE LÍNEA ===
  if (s_i == 1 && s_c == 0 && s_d == 0) {
    classLabel = 8; // Solo izquierda ? giro izquierda
  } else if (s_i == 0 && s_c == 0 && s_d == 1) {
    classLabel = 6; // Solo derecha ? giro derecha
  } else if (s_i == 0 && s_c == 1 && s_d == 0) {
    classLabel = 1; // Solo centro ? avanzar recto
  } else if (s_i == 1 && s_c == 1 && s_d == 0) {
    classLabel = 4; // Centro + izquierda ? leve izquierda
  } else if (s_i == 0 && s_c == 1 && s_d == 1) {
    classLabel = 5; // Centro + derecha ? leve derecha
  }

  // === CASO: LÍNEA PERDIDA ===
  else if (s_i == 0 && s_c == 0 && s_d == 0) {
```

```
  if (s_i_p == 1 && s_c_p == 0 && s_d_p == 0) {
    classLabel = 8; // Última dirección fue izquierda
  } else if (s_i_p == 0 && s_c_p == 0 && s_d_p == 1) {
    classLabel = 6; // Última dirección fue derecha
  } else {
    classLabel = 10; // Sin información clara ? detenerse
  }
}

// === CONTROL DE MOTORES ===
switch (classLabel) {
  case 1: // Avanzar recto
    analogWrite(motorIzq, muyRapido);
    analogWrite(motorDer, muyRapido);
    break;
  case 4: // Leve izquierda
    analogWrite(motorIzq, medio);
    analogWrite(motorDer, muyRapido);
    break;
  case 5: // Leve derecha
    analogWrite(motorIzq, muyRapido);
    analogWrite(motorDer, medio);
    break;
  case 6: // Giro derecha
    analogWrite(motorIzq, muyRapido);
    analogWrite(motorDer, muyLento);
    break;
  case 8: // Giro izquierda
    analogWrite(motorIzq, muyLento);
```

```
      analogWrite(motorDer, muyRapido);
      break;
    case 10: // Detenerse (sin dirección clara)
    default:
      analogWrite(motorIzq, stop);
      analogWrite(motorDer, stop);
      break;
  }

  // Guardar última lectura
  s_i_p = s_i;
  s_c_p = s_c;
  s_d_p = s_d;

  delay(20); // Tiempo de respuesta óptimo
}
```

Demostración de la simulación

https://drive.google.com/drive/folders/1ua7kTurUh0o5i2NyB4ML8UUlpaV5RbLJ?usp=sharing

Conclusión

El uso de árboles de decisión para la clasificación de la posición de un robot seguidor de línea nos permitió demostrar que bien implementado puede llegar a ser una solución eficiente, fácil de interpretar y adecuada para poder implementarlo en Arduino. A partir de un conjunto de datos históricos con valores binarios provenientes de sensores digitales, se logró entrenar un modelo capaz de predecir correctamente la acción que el robot debe ejecutar para mantener su trayectoria sobre una línea negra en un fondo blanco.

La simplicidad del modelo permite traducir sus reglas directamente a estructuras condicionales de programación, lo que facilita su implementación en sistemas embebidos con recursos limitados. Además, el comportamiento basado en sensores ON/OFF resulta suficiente para lograr una respuesta rápida y precisa en entornos controlados.

Este enfoque no solo mejora el seguimiento de línea en tiempo real, sino que también sienta las bases para futuras mejoras, como el uso de sensores analógicos, la implementación de aprendizaje adaptativo, o la integración con otros sistemas de navegación autónoma más complejos.