



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:

Nombre de actividad:

Clasificación de Posición en un Robot Seguidor de Línea Usando Árbol de
Decisión e Implementación en Arduino

Actividad realizada por:

Carlos Diego Dominguez López - 21030006

Guadalupe Victoria, Durango

Fecha de entrega:

05	04	2025
----	----	------

INDICE

Introducción.....	1
Objetivo.....	1
Marco Teórico	2
Aprendizaje Automático y Clasificación Supervisada	2
Árboles de Decisión	2
Seguidores de Línea.....	3
Implementación en Arduino	4
Importancia de la Clasificación en la Robótica Móvil.....	4
Desarrollo	5
Código para Árbol de Decisiones en Notebook.....	5
Código para Seguidor de Línea en Arduino.....	9
Diagrama para Simulación en Proteus.....	12
Link para Video de Simulación.....	12
Conclusión	12
Bibliografía.....	13

Introducción

En la presente práctica, se aplican técnicas de clasificación supervisada mediante árboles de decisión para optimizar el control de un robot seguidor de línea. El objetivo principal es desarrollar un modelo funcional que, a partir de un conjunto de datos históricos, permita predecir la posición del robot en relación con la trayectoria de una línea negra sobre una superficie blanca y generar un sistema de control eficiente para su implementación en Arduino.

Para ello, se dispone de un dataset que contiene combinaciones de estados de tres sensores digitales (ON/OFF), junto con la clasificación correspondiente de la posición del robot. A través del análisis de estos datos, se entrena un modelo de clasificación basado en árboles de decisión, el cual es evaluado para garantizar su precisión y posteriormente convertido a código en lenguaje C para su integración en Arduino.

La implementación física del modelo implica la programación del microcontrolador con una estructura condicional que ajuste la velocidad de los motores según la posición detectada. Finalmente, se realizan pruebas en una pista simulada para validar el desempeño del robot y realizar los ajustes necesarios, asegurando así un seguimiento estable de la línea. Esta práctica no solo fortalece el conocimiento en aprendizaje automático aplicado a la robótica, sino que también permite comprender la conversión de modelos de clasificación en sistemas embebidos de control.

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.

Marco Teórico

Aprendizaje Automático y Clasificación Supervisada

El aprendizaje automático es una rama de la inteligencia artificial que permite a los sistemas mejorar su desempeño en tareas específicas a partir de datos. Dentro de este campo, la clasificación supervisada es una técnica en la que un algoritmo aprende a asignar etiquetas a nuevas observaciones basándose en un conjunto de datos previamente etiquetado.



Figura 1. Modelo de Aprendizaje por Refuerzo

Uno de los algoritmos más utilizados en clasificación supervisada son los árboles de decisión, debido a su capacidad para modelar relaciones complejas en los datos de forma intuitiva y explicable.

Árboles de Decisión

Los árboles de decisión son modelos de aprendizaje supervisado que organizan la toma de decisiones en una estructura jerárquica de nodos. Cada nodo representa una condición sobre una variable de entrada y cada rama representa una posible decisión basada en esa condición. El proceso de clasificación se realiza recorriendo el árbol desde la raíz hasta una hoja, donde se obtiene la predicción final.

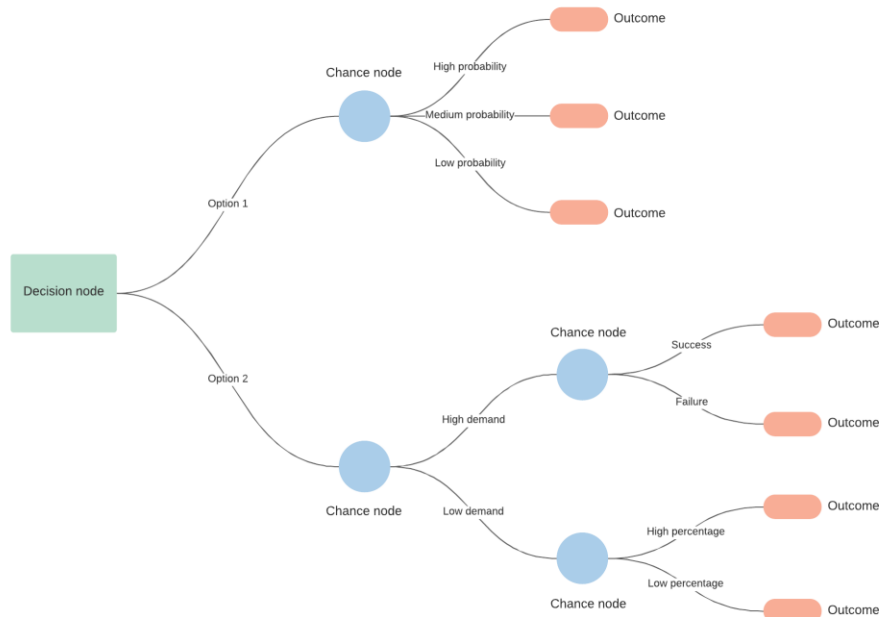


Figura 2. Ejemplo de Árbol de Decisiones

El algoritmo más común para generar árboles de decisión es CART (Classification and Regression Trees), el cual divide los datos en subconjuntos con base en la métrica de impureza de Gini o la entropía. En el contexto de esta práctica, los árboles de decisión se utilizan para determinar la posición del robot seguidor de línea en función del estado de sus sensores.

Seguidores de Línea

Un robot seguidor de línea es un sistema autónomo diseñado para desplazarse siguiendo una línea trazada sobre una superficie. Generalmente, estos robots utilizan sensores ópticos que detectan el contraste entre la línea y el fondo, permitiendo ajustar su trayectoria mediante el control de sus motores.

El control del robot se basa en la lectura de sensores que detectan la posición de la línea y en la ejecución de estrategias de corrección de trayectoria, como las siguientes:

- **Corrección simple:** Se ajusta la velocidad de los motores según la posición detectada.

- **Control PID:** Se aplica un control proporcional-integral-derivativo para mejorar la estabilidad del seguimiento.
- **Clasificación supervisada:** Se entrena un modelo de aprendizaje automático para identificar patrones en los datos y mejorar la toma de decisiones del robot.

Implementación en Arduino

Arduino es una plataforma de hardware y software de código abierto que facilita el desarrollo de sistemas embebidos. Para implementar el modelo entrenado en Python en un microcontrolador Arduino, se debe convertir el árbol de decisión a código C, representándolo mediante estructuras condicionales (if-else).

El código resultante debe controlar los motores del robot según la clase predicha, de manera que pueda corregir su trayectoria de forma eficiente. La implementación incluye pruebas en una pista física y ajustes en los parámetros de velocidad y respuesta para lograr un seguimiento estable de la línea.

Importancia de la Clasificación en la Robótica Móvil

El uso de técnicas de aprendizaje automático, como los árboles de decisión, en sistemas de robótica móvil permite mejorar la precisión y adaptabilidad de los robots en entornos dinámicos. La integración de modelos de clasificación en robots seguidores de línea no solo optimiza su desempeño, sino que también abre la puerta a aplicaciones más avanzadas, como navegación autónoma en almacenes, logística y agricultura de precisión.

Desarrollo

Código para Árbol de Decisiones en Notebook

Para poder crear el árbol de decisiones que usaremos en el seguidor de línea, primero tendremos que introducir los datos para la toma de decisiones en cada una de las situaciones probables en las que se pueda encontrar el seguidor, para empezar, usaremos las siguientes librerías:

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn.model_selection import train_test_split
```

Figura 3. Librerías usadas para la creación del árbol de decisiones

Posteriormente, agregamos la tabla con los datos necesarios para interpretar las posibles situaciones en las que se podría ver el seguidor con sus sensores y sus modos de acción:

```
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	2
2	0	0	0	0	1	0	5
3	0	0	0	0	1	1	4
4	0	0	0	1	0	0	8
..
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]

Figura 4. Tabla de datos para el seguidor de líneas

Después de cargar y explorar los datos, el siguiente paso es separar las variables de entrada y la salida objetivo para el entrenamiento del modelo:

```
X = data[['s_i_p','s_c_p','s_d_p','s_i','s_c','s_d']]
y_1 = data[['accion']]
```

Figura 5. Separación de variables de entrada y salida

- **X:** Se está creando una variable que contiene las características (variables de entrada) del conjunto de datos. Se seleccionan las columnas s_i_p, s_c_p, s_d_p, s_i, s_c y s_d, que representan los valores de los sensores del robot en el pasado (_p) y en el presente.
- **y_1:** Se está creando una variable con la columna acción, que representa la salida o el objetivo que el modelo debe predecir (por ejemplo, el movimiento del robot).

Luego confirmamos que todos los valores son numéricos para evitar problemas a la hora de crear el árbol de decisión.

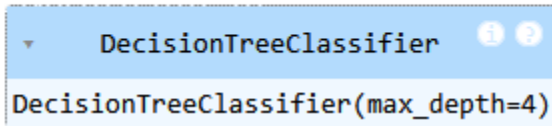
```
data.info()

<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
```

Figura 6. Información general de los datos del código

Después de definir las variables de entrada (X) y salida (y_1), el siguiente paso es entrenar el modelo utilizando un árbol de decisión:


```
clf = DecisionTreeClassifier(max_depth=4)
clf.fit(X,y_1)
```



```
DecisionTreeClassifier
```

```
DecisionTreeClassifier(max_depth=4)
```

Figura 7. Profundidad del árbol de decisión

- **DecisionTreeClassifier(max_depth=4)** crea un árbol de decisión con una profundidad máxima de 4 niveles, lo que ayuda a controlar la complejidad del modelo y evitar sobreajuste.
- **clf.fit (X, y_1)** entrena el modelo utilizando los datos de entrada (X) y las etiquetas (y_1), permitiéndole aprender las reglas para clasificar correctamente las acciones del robot en función de los sensores.

Una vez que el modelo ha sido entrenado, es importante evaluar su desempeño para verificar qué tan bien ha aprendido a clasificar las acciones del robot. Para ello, se mide su precisión con el siguiente código:

```
print('Precision:', clf.score(X,y_1))
```

```
Precision: 0.734375
```

Figura 8. Precisión del árbol de decisiones

Después de entrenar y evaluar el modelo, el siguiente paso es extraer las reglas de decisión generadas por el árbol para comprender cómo clasifica las acciones del robot:

```
rules = export_text(clf,feature_names=list(X.columns))
print(rules)
```

Figura 9. Extracción de reglas de decisión

export_text (clf, feature_names=list (X. columns)) convierte el árbol de decisión en una representación de texto que muestra las reglas utilizadas para clasificar los datos.

print(rules) muestra estas reglas en la consola, facilitando la interpretación del modelo.

```

|--- s_c <= 0.50
|   |--- s_i_p <= 0.50
|   |   |--- s_i <= 0.50
|   |   |   |--- s_d <= 0.50
|   |   |   |   |--- class: 10
|   |   |   |   |--- s_d > 0.50
|   |   |   |   |--- class: 2
|   |   |--- s_i > 0.50
|   |   |   |--- s_d <= 0.50
|   |   |   |   |--- class: 10
|   |   |   |   |--- s_d > 0.50
|   |   |   |   |--- class: 10
|   |--- s_i_p > 0.50
|   |   |--- s_d_p <= 0.50
|   |   |   |--- s_i <= 0.50
|   |   |   |   |--- class: 10
|   |   |   |   |--- s_i > 0.50
|   |   |   |   |--- class: 10
|   |   |--- s_d_p > 0.50
|   |   |   |--- class: 10
|--- s_c > 0.50
|   |--- s_c_p <= 0.50
|   |   |--- s_i <= 0.50
|   |   |   |--- s_i_p <= 0.50
|   |   |   |   |--- class: 4
|   |   |   |   |--- s_i_p > 0.50
|   |   |   |   |--- class: 10
|   |   |--- s_i > 0.50
|   |   |   |--- s_d_p <= 0.50
|   |   |   |   |--- class: 10
|   |   |   |--- s_d_p > 0.50
|   |   |   |   |--- class: 10
|   |--- s_c_p > 0.50
|   |   |--- s_i_p <= 0.50
|   |   |   |--- s_i <= 0.50
|   |   |   |   |--- class: 5
|   |   |   |   |--- s_i > 0.50
|   |   |   |   |--- class: 6
|   |   |--- s_i_p > 0.50
|   |   |   |--- s_d_p <= 0.50
|   |   |   |   |--- class: 6
|   |   |   |--- s_d_p > 0.50
|   |   |   |   |--- class: 0

```

Figura 10. Árbol de Decisión para el seguidor de línea

Código para Seguidor de Línea en Arduino

Se definen los pines de los sensores de línea y sus estados previos (s_i_p, s_c_p, s_d_p), los cuales permiten detectar la posición actual y pasada del robot sobre la línea. También se establecen los pines para controlar los motores, los cuales permitirán mover el robot en diferentes direcciones en función de la clasificación de la posición detectada.

```
// Definición de pines para los sensores
const int s_i = 2;    // Sensor izquierdo
const int s_c = 4;    // Sensor central
const int s_d = 6;    // Sensor derecho
const int s_i_p = 3;  // Sensor izquierdo previo
const int s_c_p = 5;  // Sensor central previo
const int s_d_p = 7;  // Sensor derecho previo

// Pines de los motores
const int motorIzqA = 9;
const int motorIzqB = 10;
const int motorDerA = 11;
const int motorDerB = 12;
```

Figura 11. Definición de pines para sensores y motores

En la función setup (), se configuran los sensores como entradas y los motores como salidas, lo que permite recibir datos de la línea y controlar el movimiento del robot. Además, se inicializa la comunicación serie a 9600 baudios, lo que permite monitorear la clase detectada en el monitor serie para depuración y análisis del comportamiento del robot.

```
void setup() {
  pinMode(s_i, INPUT);
  pinMode(s_c, INPUT);
  pinMode(s_d, INPUT);
  pinMode(s_i_p, INPUT);
  pinMode(s_c_p, INPUT);
  pinMode(s_d_p, INPUT);
  pinMode(motorIzqA, OUTPUT);
  pinMode(motorIzqB, OUTPUT);
  pinMode(motorDerA, OUTPUT);
  pinMode(motorDerB, OUTPUT);
  Serial.begin(9600);
}
```

Figura 12. Configuración de pines y comunicación serie

La función `clasificarPosicion()` lee los valores de los sensores actuales y previos para determinar la posición del robot en relación con la línea. Se emplea una estructura de árbol de decisión anidado para clasificar la posición en diferentes categorías, donde los valores de los sensores determinan si el robot debe avanzar, girar o detenerse. La función devuelve un número entero que representa una clase específica, facilitando la toma de decisiones en la función de control de movimiento.

```
int clasificarPosicion() {
    int si = digitalRead(s_i);
    int sc = digitalRead(s_c);
    int sd = digitalRead(s_d);
    int si_p = digitalRead(s_i_p);
    int sc_p = digitalRead(s_c_p);
    int sd_p = digitalRead(s_d_p);

    if (sc == 0) {
        if (si_p == 0) {
            if (si == 0) {
                return (sd == 0) ? 10 : 2;
            } else {
                return 10;
            }
        } else {
            return 10;
        }
    } else {
        if (sc_p == 0) {
            return (si_p == 0) ? 4 : 10;
        } else {
            if (si_p == 0) {
                return (si == 0) ? 5 : 6;
            } else {
                return (sd_p == 0) ? 6 : 0;
            }
        }
    }
}
```

Figura 13. Clasificación de la posición del robot usando un árbol de decisión

La función `moverRobot(int clase)` recibe como entrada la clase determinada por `clasificarPosicion()` y activa o desactiva los motores de acuerdo con la estrategia definida. Si la clase corresponde a 4 o 5, el robot avanza recto; si es 6, gira a la izquierda; si es 2, gira a la derecha; y si es 0 o 10, el robot se detiene. Esta estructura permite que el robot siga la línea de manera lógica según la información proporcionada por los sensores.

```

void moverRobot(int clase) {
  switch (clase) {
    case 4:
    case 5:
      digitalWrite(motorIzqA, HIGH);
      digitalWrite(motorIzqB, LOW);
      digitalWrite(motorDerA, HIGH);
      digitalWrite(motorDerB, LOW);
      break;
    case 6:
      digitalWrite(motorIzqA, LOW);
      digitalWrite(motorIzqB, HIGH);
      digitalWrite(motorDerA, HIGH);
      digitalWrite(motorDerB, LOW);
      break;
    case 2:
      digitalWrite(motorIzqA, HIGH);
      digitalWrite(motorIzqB, LOW);
      digitalWrite(motorDerA, LOW);
      digitalWrite(motorDerB, HIGH);
      break;
    case 0:
    case 10:
      digitalWrite(motorIzqA, LOW);
      digitalWrite(motorIzqB, LOW);
      digitalWrite(motorDerA, LOW);

      digitalWrite(motorDerB, LOW);
      break;
  }
}

```

Figura 14. Movimiento del robot según la clasificación

En la función `loop()`, el robot ejecuta continuamente la secuencia de lectura de sensores, clasificación de posición y ejecución del movimiento. Se llama a `clasificarPosicion()` para determinar la clase, se imprime el resultado en el monitor serie y se ejecuta `moverRobot(clase)` para realizar la acción correspondiente. Se añade un pequeño `delay(100)` para estabilizar las lecturas y evitar cambios bruscos en el movimiento del robot.

```

void loop() {
  int clase = clasificarPosicion();
  Serial.print("Clase detectada: ");
  Serial.println(clase);
  moverRobot(clase);
  delay(100);
}

```

Figura 15. Bucle principal del programa

Diagrama para Simulación en Proteus

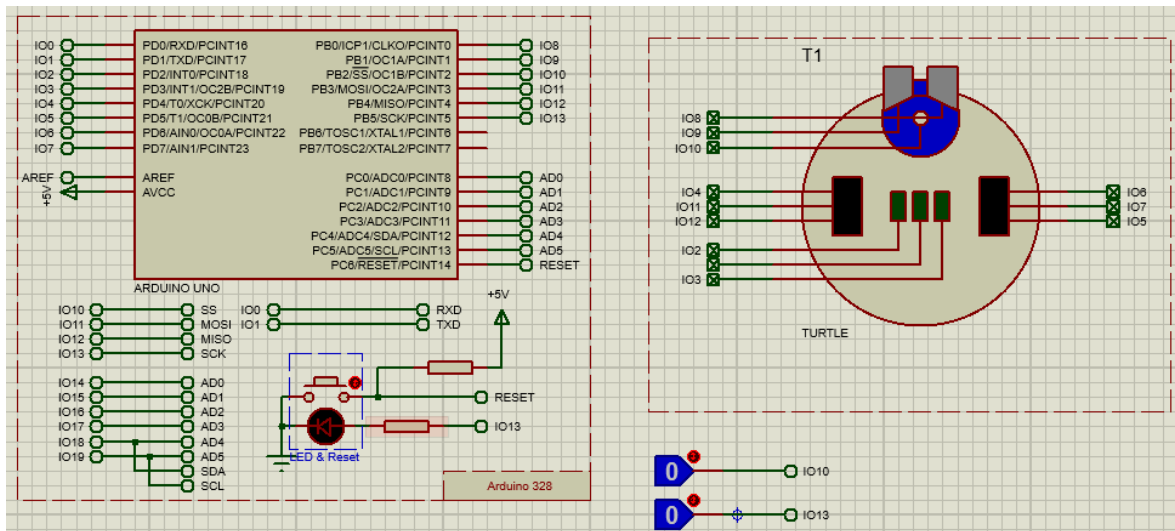


Figura 16. Arduino UNO con Robot Arduino Turtle

Link para Video de Simulación

https://drive.google.com/drive/folders/1KbdllfiNRTEWLYVna9yR_dUDSII6IMLB

Conclusión

En esta práctica se logró aplicar técnicas de clasificación supervisada utilizando árboles de decisión para mejorar el control de un robot seguidor de línea. A partir de un conjunto de datos históricos, se entrenó un modelo capaz de predecir la posición del robot con base en las lecturas de sus sensores, lo que permitió generar una estrategia eficiente para su desplazamiento sobre una línea negra en un fondo blanco.

El proceso comenzó con la carga y exploración del dataset, donde se identificaron las variables de entrada y la salida objetivo. Posteriormente, se entrenó un modelo de árbol de decisión con una profundidad controlada para evitar sobreajuste, y se evaluó su precisión en los datos de entrenamiento. La extracción de las reglas del árbol permitió comprender la lógica detrás de las decisiones del modelo, lo que

facilitó su conversión a código C para implementarlo en un microcontrolador Arduino.

Este ejercicio no solo permitió aplicar el aprendizaje automático en la robótica móvil, sino que también demostró la importancia de integrar modelos de clasificación en sistemas embebidos para optimizar su desempeño. La combinación de árboles de decisión y Arduino es una herramienta poderosa para desarrollar robots autónomos con mayor precisión y adaptabilidad.

En futuras implementaciones, se podrían explorar otros modelos de clasificación, como redes neuronales o máquinas de soporte vectorial, para comparar su desempeño con los árboles de decisión. También sería interesante evaluar el modelo en entornos con distintas condiciones de iluminación o variaciones en la superficie, para analizar su robustez y capacidad de generalización.

Bibliografía

- Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.
- Mitchell, T. M. (1997). *Machine learning*. McGraw-Hill.
- Russell, S., & Norvig, P. (2020). *Artificial intelligence: A modern approach* (4th ed.). Pearson.
- Szeliski, R. (2022). *Computer vision: Algorithms and applications* (2nd ed.). Springer.
- Tang, J. (2017). *Arduino programming with .NET and Sketch*. Apress.
- Wolf, M. (2017). *Embedded system interactions*. Morgan & Claypool Publishers.