INTRODUCCIÓN.

Este proyecto te propone diseñar, programar y simular un robot inteligente capaz de moverse por un entorno virtual, detectar un objeto mediante visión artificial y realizar una acción determinada (por ejemplo, recogerlo o marcar su ubicación). Para ello usarás herramientas profesionales como Webots, Python y librerías de visión artificial como OpenCV o Teachable Machine.

OBJETIVO GENERAL.

Diseñar un robot funcional, o seleccionar uno prediseñado desde la aplicación, que cumpla una tarea autónoma indicada por el equipo dentro de un entorno simulado creado por el alumnado, utilizando sensores, control cinemático, visión artificial y toma de decisiones programadas. Hay que poner al robot en una situación.

Ejemplos de gamificación:

El equipo de desarrollo de vuestra propia empresa debe crear un robot funcional para el pedido realizado por una gran empresa, para operar en un almacén automatizado, por lo tanto, nuestro robot debe identificar y transportar objetos de color rojo, los que corren más pris, desde un punto A hasta un punto B, esquivando obstáculos y optimizando su trayectoria.

Tu equipo forma parte de una misión de rescate en un planeta desconocido, por lo tanto, hay que diseñar un robot autónomo que localice una baliza de emergencia esquivando obstáculos y tomando decisiones inteligentes. Solo existen los planos del entorno, sensores básicos y una IA que debe entrenarse para reconocer la baliza visualmente.

Puntos a mencionar en el informe que no se deben olvidar.

1. Su modelado y control cinemático.
2. Problemas encontrados a la hora de diseñarlos y su solución.
3. La técnica de programación (Teach Pendant y la programación estructurada).
4. Opciones de diseño que se ha tenido en cuenta.
5. Cuál es la tipología física a la que pertenece.

PLAN DE TRABAJO.

FASE 1: Planificación del proyecto.

- Formad una pareja, identificad las tareas y distribuirlas entre los miembros de la pareja.
- Organizarse por sesiones y planificarse temporalmente.

FASE 2: Diseño del robot y del entorno.

- Elegir el tipo de robot en Webots: seleccionando o creado un robot base, móvil diferencial, SCARA, DELTA o manipulador simple.
- Modelarlo en Webots (se puede partir de un modelo base).
- Probad su funcionamiento básico y entenderlo.
- Añadir sensores simulados, por ejemplo:
    - Distancia (ultrasónico o infrarrojo).
    - Cámara (para visión artificial).
    - Encoders (opcional para movimiento controlado).

PISTA:

Webots tiene robots predefinidos como e-puck o TurtleBot3 ideales para empezar.

Se pueden clonar y añadir sensores desde el panel izquierdo *(PROTO)*.

Ejemplos de proyectos posibles:

Nombre del robot	¿Qué hace?
RoboPicker	Detecta y recoge objetos rojos y los lleva a una zona
LabBot	Navega un laberinto hasta encontrar una baliza
TrashSorter	Clasifica objetos según su color en zonas distintas
ExplorerBot	Crea un mapa simple del entorno (SLAM inicial)

FASE 3: Programación básica del comportamiento.

- Control de motores para avanzar, girar y detenerse.
- Lectura de sensores: reacción ante obstáculos.
- Programar desplazamientos, giros y esquiva de obstáculos.
- Control por condicionales (if, while, for).
- Control cinemático simple: velocidad lineal + rotacional.
- Algoritmo de toma de decisiones (lógica condicional o máquina de estados).

PISTA:

Usar el controlador de Python (my_robot.py).
Para mover el robot, por ejemplo:

```
left_motor.setVelocity(3.0)
right_motor.setVelocity(3.0)
```

Para leer un sensor de distancia, *por ejemplo:*

```
distance = distance_sensor.getValue()
if distance < 100:   # ¡Cuidado, obstáculo!
```

FASE 4: Comportamiento autónomo completo. Visión artificial e IA.

- Usad la cámara del robot: utilizando **OpenCV** en Webots.
- Detectar objetos de cierto color (por ejemplo, rojo) con OpenCV o un modelo entrenado.
- Tomar decisiones: si ve rojo → acercarse y "recoger" (simulado).
- Indicad al robot que reaccione al detectar el objeto (por ejemplo, detenerse o acercarse).
- Integrar resto de sensores (de visión, de movimiento, etc).
- Realizar tareas como:
    - Detectar el objeto.
    - Recogerlo (simulado).
    - Llevarlo a una zona objetivo.

PISTA:

Usar OpenCV para detectar color rojo, por ejemplo:

```
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
mask = cv2.inRange(hsv, (0,100,100), (10,255,255))  # Rango rojo
contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
if contours:
    # Objeto rojo detectado
```

*Crear una máquina de **estados finita**, por ejemplo:*

```
estado = "buscar"
if estado == "buscar":
    # Explorar
elif estado == "acercar":
    # Ir hacia el objeto
elif estado == "coger":
    # Ejecutar acción de recogida
```

FASE 5: Simulación final y presentación.

- Realizad pruebas completas.
- Entregables:
    - **Presentación final:** Vídeo demostrativo o simulación en directo y las explicaciones necesarias
    - Los **archivos requeridos**:
        - Informe técnico completado (plantilla colgada en Aules) .
        - Código fuente comentado (en .py o .cpp).
        - Archivos de Webots: robot y entorno (.WBT)

PISTA:

Graba la simulación con OBS o el propio grabador de Webots *(Ctrl + Shift + V)*.

Exporta tu *mundo .wbt* y el controlador *.py*.

EVALUACIÓN (RÚBRICA).

Criterio	Puntos
Robot funcional (movimiento, esquiva)	2
Detección de baliza mediante IA	3
Entorno de simulación bien diseñado	2
Trabajo en equipo y roles definidos	1
Presentación clara y justificada	2
Total	**10**

RECURSOS:

→ ENLACES ÚTILES.

- Tutorial Webots con Python: https://cyberbotics.com/doc/guide/tutorial-4-simple-robot-python
- Documentación de Webots: https://cyberbotics.com/doc/guide/index
- Tutoriales de OpenCV: https://docs.opencv.org/master/d6/d00/tutorial_py_root.html
- OpenCV detección de colores: https://docs.opencv.org/4.x/da/d97/tutorial_threshold_inRange.html
- Teachable Machine (opcional): https://teachablemachine.withgoogle.com/
- GitHub ejemplos Webots + Python: https://github.com/cyberbotics/webots/tree/master/projects/samples

→ ENTORNO DE SIMULACIÓN EN WEBOTS.

1. Creación del Mundo en Webots:

- **Descarga e instalación de Webots:**
  - Accede a la página oficial de Webots y descarga la versión correspondiente a tu sistema operativo: https://cyberbotics.com.
- **Configuración del entorno:**
  - Abre Webots y crea un nuevo mundo.
  - Añade un suelo (Floor) y define sus dimensiones.
  - Incorpora obstáculos simples (Box) que el robot deberá esquivar.

2. Integración del Robot:

- **Selección del robot:**
  - Utiliza el robot e-puck disponible en la biblioteca de Webots. Este robot es adecuado para proyectos educativos y cuenta con sensores y actuadores básicos.
- **Configuración de sensores:**
  - Activa los sensores de proximidad (ps0 a ps7) para la detección de obstáculos.Wikipedia, la enciclopedia libre+1Wikipedia+1
  - Añade una cámara para la detección de colores u objetos.

→ CÓDIGO BASE EN PYTHON PARA EL ROBOT

A continuación, se proporciona un código base de ejemplo en Python que permite al robot moverse y evitar obstáculos utilizando sus sensores de proximidad:

Descripción del código:

- **Inicialización:** Se configura el robot, los motores y los sensores de proximidad. Wikipédia, l'encyclopédie libre
- **Lectura de sensores:** Se obtienen los valores de los sensores de proximidad.
- **Detección de obstáculos:** Se determina si hay obstáculos a la izquierda o derecha del robot.
- **Lógica de control:** Si se detecta un obstáculo a la izquierda, el robot gira a la derecha y viceversa. Si no hay obstáculos, el robot avanza recto.

```
from controller import Robot

# Tiempo de paso de la simulación
TIME_STEP = 64

# Inicialización del robot
robot = Robot()

# Inicialización de los motores
left_motor = robot.getDevice('left wheel motor')
right_motor = robot.getDevice('right wheel motor')
left_motor.setPosition(float('inf'))
right_motor.setPosition(float('inf'))
left_motor.setVelocity(0.0)
right_motor.setVelocity(0.0)

# Inicialización de los sensores de proximidad
prox_sensors = []
for i in range(8):
    sensor_name = f'ps{i}'
    sensor = robot.getDevice(sensor_name)
    sensor.enable(TIME_STEP)
    prox_sensors.append(sensor)

# Función para leer los valores de los sensores
def read_sensors():
    return [sensor.getValue() for sensor in prox_sensors]



# Bucle principal de control
while robot.step(TIME_STEP) != -1:
    sensor_values = read_sensors()
```

```
# Detección de obstáculos
right_obstacle = sensor_values[0] > 80.0 or sensor_values[1] > 80.0 or sensor_values[2] > 80.0
left_obstacle = sensor_values[5] > 80.0 or sensor_values[6] > 80.0 or sensor_values[7] > 80.0


# Lógica de evitación de obstáculos
if left_obstacle:
    left_speed = 0.5
    right_speed = -0.5
elif right_obstacle:
    left_speed = -0.5
    right_speed = 0.5
else:
    left_speed = 1.0
    right_speed = 1.0

# Establecer velocidades de los motores
left_motor.setVelocity(left_speed)
right_motor.setVelocity(right_speed)
```

→ AMPLIACIÓN

- Implementar SLAM o navegación autónoma con LIDAR.
- Crear un sistema multitarea con planificación de tareas.
- Usar ROS para separar nodos (control motor, visión, lógica, etc.)
- Realizar pruebas en Jetson o RPi con entorno físico mínimo.
- Medir eficiencia: tiempo de tarea, tasa de éxito, consumo energético simulado.
- Integrar un comportamiento diferente si detecta múltiples señales.
- Agregar una pantalla en la que el robot “hable” o dé feedback textual.