

Human-Robot Interaction and Autonomous Systems

AI 616

Master of Science in Artificial intelligence

Prepared by Dr. Khaled Eskaf

Lecture 9





Course Outlines

Module	Content	Simulation Lab
1. Introduction to Robotics	- Fundamentals of Robot Control - Classification of Robotics - Exploration of Robot Types	- Basic Robot Control - Modeling Robot Classes - Simulating Various Robot Types
2. Advanced HRI Studies	- Introductory to Advanced Concepts in HRI	- Basic to Complex HRI Scenarios
3. Social Dynamics in Robotics	- Architectures for Social Interaction - Case Studies and History of HRI	- Social Interaction Simulations - Historical Evolution in HRI
4. Strategic HRI System Design	- Core Principles and Communication Strategies in HRI	- Designing Interfaces - Communication Protocol Simulations
5. HRI Efficiency and Effectiveness	- Metrics for Mission Effectiveness and Human Factors	- Implementing and Analyzing HRI Metrics
6. Innovative HRI System Design	- Exploring Innovative Designs and Real-World Applications	- Testing Innovative Designs - Simulating Diverse Scenarios
7. HRI System Classification	- Developing Comprehensive Classifications	- Classification Application and Case Studies
8. Human Factors in HRI	- Analyzing Ergonomics and Human Factors	- Ergonomic Simulations - Scenario-Based Human Factor Analysis
9. Robotics and AI Integration	- Theoretical Concepts and Practical AI Applications	- AI Algorithm Implementation - AI-Driven HRI Modeling
10. HRI Case Studies	- Mobile Assistive Robot Studies	- Designing and Testing Mobile Assistive Robots



EXAMPLES



Webots

robot simulation

Analyze a 3x3 matrix around the central pixel and use the average color values for decision-making.

P1	P2	P3
P4	P5	P6
P7	P8	P9

-1	0	+1
-1	0	+1
-1	0	+1

We'll calculate the average red, green, and blue values of the nine pixels at the center of the image and use these averages to determine the robot's movement.

```
while robot.step(TIME_STEP) != -1:
    image = camera.getImage()
    width = camera.getWidth()
    height = camera.getHeight()

    # Calculate the central pixel's coordinates
    centerX = width // 2
    centerY = height // 2

    # Initialize sums for the color components
    sum_red = 0
    sum_green = 0
    sum_blue = 0
```

```
# Loop over a 3x3 matrix around the central pixel
for dx in range(-1, 2): # -1, 0, 1
    for dy in range(-1, 2): # -1, 0, 1
        # Calculate the pixel's coordinates
        x = centerX + dx
        y = centerY + dy

        # Sum the color components of the pixel
        sum_red += camera.imageGetRed(image, width, x, y)
        sum_green += camera.imageGetGreen(image, width, x, y)
        sum_blue += camera.imageGetBlue(image, width, x, y)
```

-1	0	+1
-1	0	+1
-1	0	+1

```
# Calculate the average color components
avg_red = sum_red // 9
avg_green = sum_green // 9
avg_blue = sum_blue // 9
```

```
# Adjust thresholds and decisions based on the average color
```

```
RED_THRESHOLD = 100
```

```
GREEN_THRESHOLD = 45
```

```
BLUE_THRESHOLD = 45
```

```
RED_THRESHOLD2 = 45
```

```
GREEN_THRESHOLD2 = 45
```

```
BLUE_THRESHOLD2 = 100
```

```
if avg_red > RED_THRESHOLD and avg_green < GREEN_THRESHOLD and avg_blue < BLUE_THRESHOLD:
```

```
    left_motor.setVelocity(5) # Turn right
```

```
    right_motor.setVelocity(-5)
```

```
elif avg_blue > BLUE_THRESHOLD2 and avg_green < GREEN_THRESHOLD2 and avg_red < RED_THRESHOLD2:
```

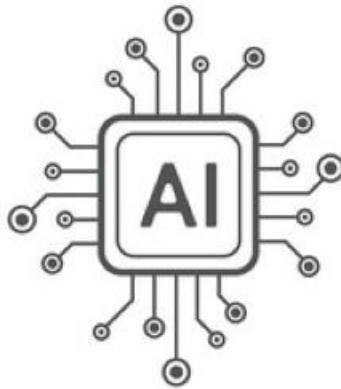
```
    left_motor.setVelocity(-5) # Move backward
```

```
    right_motor.setVelocity(-5)
```

```
else:
```

```
    left_motor.setVelocity(5) # Move forward
```

```
    right_motor.setVelocity(5)
```



Webots

robot simulation

Control the robot's speed based on color detection—moving faster if the color is green and very slowly if the color is red.

We are going to use Machine Learning Algorithm.
We are going to use Linear regression.

Let's assume:

- A high "color score" represents green, encouraging faster movement.
- A low "color score" represents red, dictating slower movement.

For simplicity, let's say our model has determined the relationship between color score and speed as $y=0.2x-1$,
where y is the robot's speed,
and x is the color score (ranging from 0 for red to 5 for green).

- Color Score: This is a conceptual representation of how "green" or "red" the camera's view is. (by averaging the RGB values).
- Linear Regression Model: The model $y = 0.2x - 1$ is a simple formula we're using to decide the robot's speed based on the detected color. The specific coefficients (0.2 and -1) are placeholders for this example and would be determined through actual model training in a real-world application.
- Speed Adjustment: The speed is adjusted based on the color score, with a higher score (more green) resulting in faster speed and a lower score (more red) resulting in slower speed or stopping.

```
from controller import Robot, Camera
```

```
# Initialization
```

```
robot = Robot()
```

```
timestep = int(robot.getBasicTimeStep())
```

```
# Assuming motor and camera initialization here
```

```
camera = robot.getDevice('camera')
```

```
camera.enable(timestep)
```

```
left_motor = robot.getDevice('left wheel motor')
```

```
right_motor = robot.getDevice('right wheel motor')
```

```
left_motor.setPosition(float('inf')) # Enable speed control
```

```
right_motor.setPosition(float('inf'))
```

```
left_motor.setVelocity(0) # Initially stop
```

```
right_motor.setVelocity(0)
```

```
while robot.step(timestep) != -1:
    # Simulate getting a color score from the camera
    # Let's say color score calculation logic has been applied here
    # For simplification, assume we get this score directly from the camera
    # A real implementation would involve analyzing the camera image
    color_score = 3 # Placeholder for the actual color score calculation (0 = red, 5 = green)

    # Apply the linear regression model
    # For our example, the model equation is  $y = 0.2x - 1$ 
    predicted_speed = 0.2 * color_score - 1 # Simplified model to control speed based on color
    score

    # Ensure the speed is within the robot's operational range
    predicted_speed = max(-10, min(10, predicted_speed))

    # Set the robot's motors to the predicted speed
    left_motor.setVelocity(predicted_speed)
    right_motor.setVelocity(predicted_speed)
```

A simple machine learning (ML) algorithm into the previous Webots robot navigation example introduces beginners to the intersection of robotics and ML. A straightforward approach is to use a decision tree classifier to decide the robot's movement based on the average color detected by the camera in a 3x3 pixel area. This example assumes you have basic data to train the classifier, representing average color values for different actions (move forward, turn right, move backward).

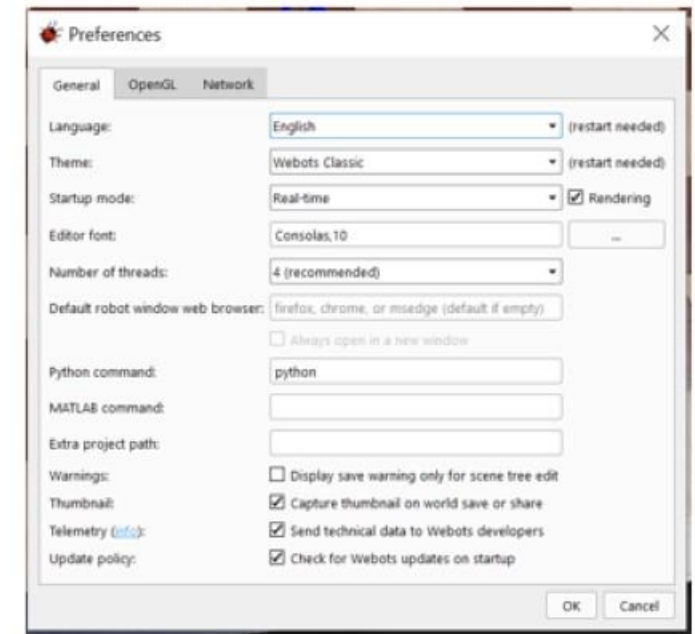
This example will be conceptual, as implementing a full ML pipeline involves steps (like data collection and model training) that go beyond the scope of running code directly in the Webots environment.

We are going to use a simple **decision tree classifier using scikit-learn**, a popular ML library for Python.

To install scikit-learn in Webots, or any Python environment within a simulation or application, you generally need to ensure that Python is correctly set up and that you have the necessary permissions to install additional libraries. Here's a step-by-step guide to installing scikit-learn for use with Webots:

Step 1: Identify Your Python Environment

- First, determine which Python environment Webots is using. Webots can use a global Python installation or a virtual environment. You can specify or check this in the Webots preferences:
- Open Webots.
- Go to Tools > Preferences (on macOS, Webots > Preferences).
- In the General tab, check the Python command setting to see which Python is being used.



Step 2: Install scikit-learn

Using a Global Python Installation

If Webots is configured to use your global Python installation, you can install scikit-learn using pip in your command line or terminal. Make sure you have pip installed and then run:

pip install scikit-learn

Using a Virtual Environment

If Webots is using a Python virtual environment:

Activate the Virtual Environment: First, you need to activate the virtual environment. Navigate to the virtual environment directory and activate it using the command line or terminal:

On Windows: `.\venv\Scripts\activate`

Error : `.\venv\Scripts\activate`
The system cannot find the path specified.

Solution:

1- Navigate to the Correct Directory

```
cd path\to\my_project
```

```
cd C:\Users\User\Documents\my_project2\controllers
```

2- Verify the Path to the Virtual Environment

```
dir
```

If you don't see the **venv** folder, or if the **Scripts** directory is missing inside **venv**, the virtual environment was not created correctly.

```
C:\Users\User\Documents\my_project2\controllers>dir
Volume in drive C has no label.
Volume Serial Number is CAE4-EDA3

Directory of C:\Users\User\Documents\my_project2\controllers

2024-03-19  01:47 AM    <DIR>          .
2024-03-19  01:47 AM    <DIR>          ..
2024-03-13  10:49 AM    <DIR>          Avid_Color_Obstacle_Using_Camera
2024-02-27  12:38 PM    <DIR>          Avoid_1
2024-03-12  12:17 PM    <DIR>          Avoid_Red_Object_usin_Camera
2024-03-19  01:47 AM    <DIR>          color_Based_Navigation_Using_Camera3X3
2024-02-20  11:24 AM    <DIR>          epuck_go_forward
2024-03-05  04:28 PM    <DIR>          multi_infrared_sensors_Avoid_Obstacle
2024-03-03  10:39 AM    <DIR>          my_camera
2024-03-05  12:32 PM    <DIR>          Obstacle_with_flash
               0 File(s)              0 bytes
               10 Dir(s)  39,366,397,952 bytes free
```

3- Recreate the Virtual Environment (If Necessary)

If the virtual environment doesn't exist or was not created properly, you'll need to recreate it. Make sure you have Python installed, and then run:

```
python -m venv venv
```

This command creates a new virtual environment named venv in the current directory. After creation, try activating it again:

```
.\venv\Scripts\activate
```

```
C:\Users\User\Documents\my_project2\controllers>.\venv\Scripts\activate
```

```
(venv) C:\Users\User\Documents\my_project2\controllers>cd path\to\my_project  
The system cannot find the path specified.
```

```
C:\Users\User\Documents\my_project2\controllers>python -m venv venv  
  
C:\Users\User\Documents\my_project2\controllers>  
C:\Users\User\Documents\my_project2\controllers>dir  
Volume in drive C has no label.  
Volume Serial Number is CAE4-EDA3  
  
Directory of C:\Users\User\Documents\my_project2\controllers  
  
2024-03-19  01:20 PM    <DIR>          .  
2024-03-19  01:20 PM    <DIR>          ..  
2024-03-13  10:49 AM    <DIR>          Avid_Color_Obstacle_Using_Camera  
2024-02-27  12:38 PM    <DIR>          Avoid_1  
2024-03-12  12:17 PM    <DIR>          Avoid_Red_Object_usin_Camera  
2024-03-19  01:47 AM    <DIR>          color_Based_Navigation_Using_Camera3X3  
2024-02-20  11:24 AM    <DIR>          epuck_go_forward  
2024-03-05  04:28 PM    <DIR>          multi_infrared_sensors_Avoid_Obstacle  
2024-03-03  10:39 AM    <DIR>          my_camera  
2024-03-05  12:32 PM    <DIR>          Obstacle_with_flash  
2024-03-19  01:20 PM    <DIR>          venv  
                0 File(s)                0 bytes
```

4- Install scikit-learn

Once your virtual environment is activated, you can install scikit-learn as previously mentioned:

pip install scikit-learn

```
(venv) C:\Users\User\Documents\my_project2\controllers>pip install scikit-learn
Collecting scikit-learn
  Obtaining dependency information for scikit-learn from https://files.pythonhosted.org/packages/76/d8/02fa68...
  Downloading scikit_learn-1.4.1.post1-cp312-cp312-win_amd64.whl.metadata (11 kB)
Collecting numpy<2.0,>=1.19.5 (from scikit-learn)
  Obtaining dependency information for numpy<2.0,>=1.19.5 from https://files.pythonhosted.org/packages/16/2e...
  Downloading numpy-1.26.4-cp312-cp312-win_amd64.whl.metadata (61 kB)
----- 61.0/61.0 kB 540.4 kB/s eta 0:00:00
Collecting scipy>=1.6.0 (from scikit-learn)
  Obtaining dependency information for scipy>=1.6.0 from https://files.pythonhosted.org/packages/f3/31/91a2a...
  Downloading scipy-1.12.0-cp312-cp312-win_amd64.whl.metadata (60 kB)
----- 60.4/60.4 kB 1.6 MB/s eta 0:00:00
Collecting joblib>=1.2.0 (from scikit-learn)
  Obtaining dependency information for joblib>=1.2.0 from https://files.pythonhosted.org/packages/10/40/d551...
  Downloading joblib-1.3.2-py3-none-any.whl.metadata (5.4 kB)
Collecting threadpoolctl>=2.0.0 (from scikit-learn)
  Obtaining dependency information for threadpoolctl>=2.0.0 from https://files.pythonhosted.org/packages/b1/...
  Downloading threadpoolctl-3.3.0-py3-none-any.whl.metadata (13 kB)
  Downloading scikit_learn-1.4.1.post1-cp312-cp312-win_amd64.whl (10.6 MB)
----- 10.6/10.6 MB 7.4 MB/s eta 0:00:00
Using cached joblib-1.3.2-py3-none-any.whl (302 kB)
  Downloading numpy-1.26.4-cp312-cp312-win_amd64.whl (15.5 MB)
----- 15.5/15.5 MB 7.4 MB/s eta 0:00:00
  Downloading scipy-1.12.0-cp312-cp312-win_amd64.whl (45.8 MB)
----- 45.8/45.8 MB 6.7 MB/s eta 0:00:00
  Downloading threadpoolctl-3.3.0-py3-none-any.whl (17 kB)
Installing collected packages: threadpoolctl, numpy, joblib, scipy, scikit-learn
Successfully installed joblib-1.3.2 numpy-1.26.4 scikit-learn-1.4.1.post1 scipy-1.12.0 threadpoolctl-3.3.0

[notice] A new release of pip is available: 23.2.1 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip
(venv) C:\Users\User\Documents\my_project2\controllers>
```

5- Verify Installation

To verify that scikit-learn has been successfully installed, you can run a small Python script to import scikit-learn and print its version:

First: Type python

Then use the following command

```
import sklearn  
print(sklearn.__version__)
```

```
(venv) C:\Users\User\Documents\my_project2\controllers>python  
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec 7 2023, 22:03:25) [MSC v.1937 64 bit (AMD64)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> import sklearn  
>>> print(sklearn.__version__)  
1.4.1.post1  
>>> _
```

Exit the Python Interactive Shell

6- Use scikit-learn in Your Webots Project

Able to use scikit-learn in your Webots projects. You can import scikit-learn in your Python controller scripts and use its functionalities as needed.

For simplicity, let's simulate training data. In a real scenario, you'd collect data based on the robot's observations and corresponding actions.

```
# Example training data: [avg_red, avg_green, avg_blue, action]
# Actions: 0 - Move Forward, 1 - Turn Right, 2 - Move Backward
training_data = [
    [150, 30, 30, 1], # Red-like colors prompt turning right.
    [30, 30, 150, 2], # Blue-like colors prompt moving backward.
    [80, 80, 80, 0],  # Neutral colors prompt moving forward.
    # Add more samples as needed
]
```

```
from sklearn.tree import DecisionTreeClassifier
import numpy as np

# Prepare training data
X = np.array([x[:-1] for x in training_data]) # Color components
y = np.array([x[-1] for x in training_data]) # Actions

# Train the classifier
clf = DecisionTreeClassifier()
clf.fit(X, y)
```

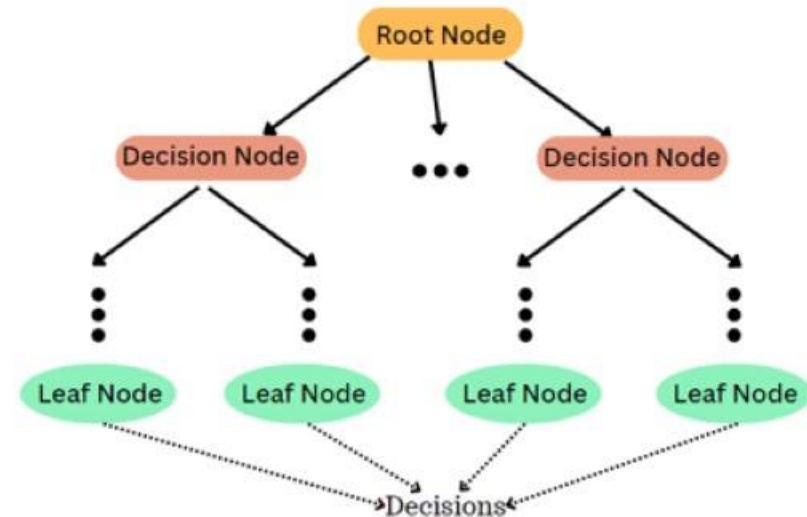
```
# Main simulation loop with ML decision-making
while robot.step(TIME_STEP) != -1:
    # Image processing to get the average color...
    avg_red, avg_green, avg_blue = get_average_color(camera)

    # Predict the action using the trained classifier
    action = clf.predict([[avg_red, avg_green, avg_blue]])[0]

    # Define robot movements based on the predicted action
    if action == 0: # Move Forward
        set_velocity(left_motor, right_motor, 5, 5)
    elif action == 1: # Turn Right
        set_velocity(left_motor, right_motor, 5, -5)
    elif action == 2: # Move Backward
        set_velocity(left_motor, right_motor, -5, -5)
```

Understanding Decision Trees

A **decision tree** is a flowchart-like tree structure where an internal node represents a feature (or attribute), the branch represents a decision rule, and each leaf node represents the outcome. In the context of our robot, the features are the average red, green, and blue (RGB) color values of a 3x3 pixel area in the camera's field of view, and the outcome is the action the robot should take (move forward, turn right, move backward).



This example assumes that we have a pretrained decision tree model. The training of the model is conceptually described, but in practice, you would need to collect data, label it, and train the model using a tool like scikit-learn in a Python environment outside of Webots.

First, you would collect data and train your model. This step is conceptual and would typically happen outside of your Webots project using scikit-learn or a similar library:

```
# Simulate training data collection
# Features: Average R, G, B values; Label: Action (0=Forward, 1=Right, 2=Backward)
training_data = [
    [120, 40, 40, 1], # Red objects mean turn right
    [40, 40, 120, 2], # Blue objects mean move backward
    [80, 110, 90, 0], # Neutral colors mean move forward
]

# Normally, here you would split your data into features (X) and labels (y),
# train a decision tree model using scikit-learn, and then save the model.
# For this example, assume we have a trained model ready to use.
```

Now, integrating the model into your Webots controller script. This part simulates loading a pretrained model and using it within the main robot control loop:

```
from controller import Robot, Camera, Motor
import numpy as np
```

```
# Define the robot's actions based on prediction
def robot_action(action, left_motor, right_motor):
    speed = 5
    if action == 0: # Move forward
        left_motor.setVelocity(speed)
        right_motor.setVelocity(speed)
    elif action == 1: # Turn right
        left_motor.setVelocity(speed)
        right_motor.setVelocity(-speed)
    elif action == 2: # Move backward
        left_motor.setVelocity(-speed)
        right_motor.setVelocity(-speed)
```

```
# Main robot control code
```

```
TIME_STEP = 64
```

```
robot = Robot()
```

```
# Initialize camera
```

```
camera = robot.getDevice('camera')
```

```
camera.enable(TIME_STEP)
```

```
# Initialize motors
```

```
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)
```

```
right_motor.setVelocity(0)
```

```
# Assume 'model' is our pretrained decision tree classifier
```

```
# For educational purposes, we will simulate its predictions  
based on color thresholds
```

```
while robot.step(TIME_STEP) != -1:
```

```
    image = camera.getImage()
```

```
    width = camera.getWidth()
```

```
    height = camera.getHeight()
```

```
# Center coordinates of the image
centerX = width // 2
centerY = height // 2

# Calculate the average color of the central 3x3 pixels
avg_color = np.zeros(3)
for dx in range(-1, 2):
    for dy in range(-1, 2):
        # Correcting the method to get RGB components for each pixel
        red = camera.imageGetRed(image, width, centerX + dx, centerY + dy)
        green = camera.imageGetGreen(image, width, centerX + dx, centerY + dy)
        blue = camera.imageGetBlue(image, width, centerX + dx, centerY + dy)
        avg_color += np.array([red, green, blue])
avg_color /= 9 # Average RGB values
```

Chapter

08

```
# Determine action based on average color
# Red detection
if avg_color[0] > 100 and avg_color[1] < 50 and avg_color[2] < 50:
    action = 1 # Turn right due to red
# Green detection
elif avg_color[1] > 100 and avg_color[0] < 50 and avg_color[2] < 50:
    action = 0 # Move forward due to green
else:
    action = 2 # Move backward for any other color as default

# Execute the action
robot_action(action, left_motor, right_motor)
```

CHAPTER 9

To integrate the machine learning model training and prediction directly into a Webots controller script, you'll have to prepare a slightly different setup. This **setup involves simulating the training data collection**, training a decision tree model, and using the model for real-time decision-making based on the robot's camera input.

However, it's important to note that for a real-world scenario or more complex simulations, you'd likely perform the model training outside of the Webots environment due to the computational costs and then load the trained model into your Webots controller for predictions.

Chapter

09

Step 1: Setting Up Your Environment

```
from controller import Robot
from sklearn.tree import DecisionTreeClassifier
import numpy as np
```

```
robot = Robot()
timestep = int(robot.getBasicTimeStep())
```

NOTE: You're importing necessary libraries (Robot class from Webots, DecisionTreeClassifier from scikit-learn, and numpy) and initializing your robot. The timestep variable stores the simulation timestep, which is essential for synchronizing your control commands with the simulation.

Step 2: Initializing Devices

```
camera = robot.getDevice('camera')
camera.enable(timestep)
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)
right_motor.setVelocity(0)
```

NOTE: You initialize the camera and motors attached to your robot. Setting the motor position to infinity (`float('inf')`) switches the motors to velocity control mode. Initially, the robot's motors are set to 0 speed, meaning the robot will not move.

Step 3: Preparing Training Data for the Decision Tree

```
training_data = np.array([  
    [120, 40, 40, 1], # Red objects mean turn right  
    [40, 40, 120, 2], # Blue objects mean move backward  
    [80, 110, 90, 0] # Neutral colors mean move forward  
])  
X_train = training_data[:, :3] # RGB values  
y_train = training_data[:, 3] # Actions
```

NOTE: This block creates an array of training data, where each row represents a different scenario. The first three numbers in each row are RGB values simulating the color the robot sees, and the fourth number is the action the robot should take (0 for forward, 1 for right, 2 for backward). This data is split into features (X_train) and labels (y_train).

Chapter

09

p 4: Training the Decision Tree Classifier

```
clf = DecisionTreeClassifier()  
clf.fit(X_train, y_train)
```

NOTE: A decision tree classifier is initialized and trained with the RGB values as features and the corresponding actions as labels. The .fit() method adjusts the model parameters to best fit the training data.

Step 5: Defining the Action Function

```
def perform_action(action, left_motor, right_motor):  
    # This function translates the predicted action into motor commands.
```

NOTE: This function maps the predicted action to actual motor speeds, controlling the robot's movement. Depending on the action, both motors may move forward, rotate in opposite directions (to turn), or both move backward.

Step 6: Main Control Loop

```
while robot.step(timestep) != -1:  
    current_rgb = np.array([[85, 90, 75]]) # Simulated current color  
    predicted_action = clf.predict(current_rgb)[0]  
    perform_action(predicted_action, left_motor, right_motor)
```

NOTE: Within an infinite loop that runs for the simulation's duration, you simulate getting RGB values from the camera and use your trained model to predict the appropriate action. The action is then performed by adjusting the motors' velocities.

- 1) Simulate Color Detection: For demonstration purposes, `current_rgb` is a placeholder for RGB values you might get from processing the camera's image.
- 2) Predict Action: The model predicts the best action (move forward, turn right, move backward) based on the current RGB values.
- 3) Execute Action: The predicted action is translated into motor movements, controlling the robot's behavior in the simulation.

Conclusion

This example demonstrates a basic application of machine learning within a robotics simulation. By training a decision tree to associate colors with movements, you've created a model that can guide a robot's navigation based on visual cues. While simplified, this process lays the foundation for more complex applications involving real-time data processing and decision-making in robotic systems.

```
# Iterate over a 3x3 matrix around the central pixel
for dx in range(-1, 2):
    for dy in range(-1, 2):
        index = ((centerY + dy) * width + (centerX + dx)) * 4
        r = image[index]
        g = image[index + 1]
        b = image[index + 2]
        sum_rgb += np.array([r, g, b])
        print(f"RGB at ({centerX+dx}, {centerY+dy}): ({r}, {g}, {b})") #

# Calculate average RGB values
avg_rgb = sum_rgb / 9
print(f"Average RGB: {avg_rgb}") # Print the average RGB values

# Predict the action based on the current RGB values
predicted_action = clf.predict([avg_rgb])[0]

# Set motor speeds based on the predicted action
speed = 5
if predicted_action == 0: # Move forward
    left_motor.setVelocity(speed)
    right_motor.setVelocity(speed)
elif predicted_action == 1: # Turn right
    left_motor.setVelocity(speed)
    right_motor.setVelocity(-speed)
elif predicted_action == 2: # Move backward
    left_motor.setVelocity(-speed)
    right_motor.setVelocity(-speed)
```

```
"""ML_DecissionTree_3 controller."""  
# Dr. Khaled Eskaf  
#Using ML Decision Tree 3  
from controller import Robot  
import numpy as np  
from sklearn.tree import DecisionTreeClassifier  
# Initialize the robot and simulation timestep  
robot = Robot()  
timestep = int(robot.getBasicTimeStep())  
# Initialize camera and motors  
camera = robot.getDevice('camera')  
camera.enable(timestep)  
left_motor = robot.getDevice('left wheel motor')  
right_motor = robot.getDevice('right wheel motor')  
left_motor.setPosition(float('inf')) # Enable speed control mode  
right_motor.setPosition(float('inf'))  
left_motor.setVelocity(0) # Initially stop the robot  
right_motor.setVelocity(0)
```

Simulated training data for decision making

```
training_data = np.array([  
    [120, 40, 40, 1], # Red-ish object  
    [40, 40, 120, 2], # Blue-ish object  
    [80, 110, 90, 0]  # Neutral object  
])
```

Split training data into features and labels

```
X_train = training_data[:, :3] # RGB values  
y_train = training_data[:, 3]  # Actions
```

Train the decision tree classifier

```
clf = DecisionTreeClassifier()  
clf.fit(X_train, y_train)
```

Main control loop

```
while robot.step(timestep) != -1:  
    image = camera.getImage()  
    width = camera.getWidth()  
    height = camera.getHeight()
```

```
# Calculate center pixel coordinates
```

```
centerX = width // 2
```

```
centerY = height // 2
```

```
# Initialize sum of RGB values
```

```
sum_rgb = np.zeros(3)
```

```
# Iterate over a 3x3 matrix around the central pixel
```

```
for dx in range(-1, 2):
```

```
    for dy in range(-1, 2):
```

```
        index = ((centerY + dy) * width + (centerX + dx)) * 4
```

```
        r = image[index]
```

```
        g = image[index + 1]
```

```
        b = image[index + 2]
```

```
        sum_rgb += np.array([r, g, b])
```

```
        print(f"RGB at ({centerX+dx}, {centerY+dy}): ({r}, {g}, {b})") # Print RGB values of each pixel
```

```
# Calculate average RGB values
```

```
avg_rgb = sum_rgb / 9
```

```
print(f"Average RGB: {avg_rgb}") # Print the average RGB values
```

```
# Predict the action based on the current RGB values
predicted_action = clf.predict([avg_rgb])[0]

# Set motor speeds based on the predicted action
speed = 5
if predicted_action == 0: # Move forward
    left_motor.setVelocity(speed)
    right_motor.setVelocity(speed)
elif predicted_action == 1: # Turn right
    left_motor.setVelocity(speed)
    right_motor.setVelocity(-speed)
elif predicted_action == 2: # Move backward
    left_motor.setVelocity(-speed)
    right_motor.setVelocity(-speed)
```

- 1) Initializes a simulated robot in Webots, equipping it with a camera and motors for navigation.
- 2) Sets up training data linking specific RGB values to actions (forward, right, backward) using a decision tree classifier.
- 3) Trains the classifier on this data, enabling the robot to make decisions based on color detected by the camera.
- 4) During operation, captures real-time camera images, extracting average RGB values from a central 3x3 pixel matrix.
- 5) Uses the trained model to predict actions based on these RGB values, guiding the robot's movement.
- 6) Demonstrates basic machine learning application in robotics for environment-responsive navigation based on visual cues.

1) Import Libraries:

- Robot from Webots allows you to control the simulated robot.
- numpy is used for handling arrays efficiently.
- DecisionTreeClassifier from scikit-learn is the machine learning model the robot will use to make decisions.

2) Initialize the Robot and Simulation Timestep:

- Create a Robot instance to interact with the simulation.
- timestep determines how often the robot's sensors are updated and actions are taken.

3) Set Up Camera and Motors:

- Access the camera and motors by their names, then enable the camera and set the motors to velocity control mode by assigning their positions to infinity. Initially, the motors are stopped.

4) Prepare Training Data:

- The training data simulates situations where different RGB color values are associated with specific actions (e.g., moving forward, turning, or moving backward). This data is represented in an array where each row contains RGB values followed by an action label.

5) Train the Decision Tree Classifier:

- The features (RGB values) and labels (actions) are extracted from the training data. The decision tree classifier is then trained on these features and labels, learning to associate specific color patterns with actions.

6) Main Control Loop:

- In an infinite loop, the robot continually captures images from its camera, calculates the average RGB values from a central 3x3 matrix, and uses the trained classifier to predict an action based on these values.

7) Calculate Average RGB Values:

- For each pixel in the 3x3 matrix centered around the camera's focal point, extract the RGB values, sum them, and then divide by 9 to get the average RGB values. This step involves iterating over the pixels, extracting their color, and accumulating the values.

8) Predict and Perform Actions:

The average RGB values are fed into the trained classifier, which predicts the most appropriate action (move forward, turn right, or move backward). The predicted action is then translated into motor velocities to move the robot accordingly.

9) Print Statements:

Throughout the process, print statements output the RGB values of each pixel in the 3x3 matrix and the overall average RGB values. This helps in understanding how the robot perceives its environment and decides on its actions.

Import necessary libraries:

- Robot: to control the simulated robot in Webots.

- numpy: for efficient array operations, crucial for handling data.

- DecisionTreeClassifier: a simple machine learning model from scikit-learn.

from controller import Robot

import numpy as np

from sklearn.tree import DecisionTreeClassifier

Create an instance of the Robot class to interact with the simulation.

robot = Robot()

Get the basic time step of the current world, which determines

how often the simulation updates the robot's sensors and actuators.

timestep = int(robot.getBasicTimeStep())

Access the camera and motor devices by their names in the robot's definition.

Then, enable the camera to start capturing images at the simulation's timestep.

camera = robot.getDevice('camera')

camera.enable(timestep)

Chapter

09

```
# For the motors, set their positions to infinity, which switches them  
# to velocity control mode, allowing us to specify speed rather than position.
```

```
left_motor = robot.getDevice('left wheel motor')  
right_motor = robot.getDevice('right wheel motor')  
left_motor.setPosition(float('inf'))  
right_motor.setPosition(float('inf'))
```

```
# Initially, stop the robot by setting both motors' speeds to 0.
```

```
left_motor.setVelocity(0)  
right_motor.setVelocity(0)
```

```
# Define the training data for the decision tree classifier.
```

```
# This data simulates scenarios where certain RGB color values of objects  
# seen by the robot are associated with specific actions (turning or moving).
```

```
training_data = np.array([  
    [120, 40, 40, 1], # Red objects make the robot turn right.  
    [40, 40, 120, 2], # Blue objects make the robot move backward.  
    [80, 110, 90, 0]  # Neutral (less distinctly colored) objects make the robot move forward.  
])
```

Chapter

09

```
# Split the training data into features (X) and labels (y).
# Features are the RGB values, and labels are the actions.
X_train = training_data[:, :3] # Extract the first three columns as features.
y_train = training_data[:, 3]  # Extract the last column as labels.

# Initialize the decision tree classifier and train (fit) it on the provided data.
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

# The main control loop of the robot.
while robot.step(timestep) != -1:
    # Retrieve the current image captured by the camera.
    image = camera.getImage()
    # Get the width and height of the camera image to calculate the center pixel.
    width = camera.getWidth()
    height = camera.getHeight()

    # Calculate the coordinates of the center pixel.
    centerX = width // 2
    centerY = height // 2
```

Prepare to calculate the average RGB values of the 3x3 area around the center pixel.

```
sum_rgb = np.zeros(3) # Initialize a sum for RGB values.
```

Loop over the pixels in a 3x3 area around the center.

```
for dx in range(-1, 2):
```

```
    for dy in range(-1, 2):
```

```
        # Calculate the index for each pixel in the flattened image array.
```

```
        index = ((centerY + dy) * width + (centerX + dx)) * 4
```

```
        # Extract the RGB values of each pixel and add them to the sum.
```

```
        r = image[index]
```

```
        g = image[index + 1]
```

```
        b = image[index + 2]
```

```
        sum_rgb += np.array([r, g, b])
```

```
        # Print the RGB values for educational purposes.
```

```
        print(f"RGB at ({centerX+dx}, {centerY+dy}): ({r}, {g}, {b})")
```

```
# Calculate the average RGB values by dividing the sum by the number of pixels (9).
avg_rgb = sum_rgb / 9
# Print the average RGB values for insight into the decision-making process.
print(f"Average RGB: {avg_rgb}")

# Use the trained classifier to predict the action based on the average RGB values.
predicted_action = clf.predict([avg_rgb])[0]

# Based on the predicted action, set the speeds of the motors to move the robot.
speed = 5 # Define a base speed.
if predicted_action == 0: # If the action is to move forward.
    left_motor.setVelocity(speed)
    right_motor.setVelocity(speed)
elif predicted_action == 1: # If the action is to turn right.
    left_motor.setVelocity(speed)
```

```
# If the action is to turn right, the left motor moves forward and the right motor moves backward.  
    left_motor.setVelocity(speed)  
    right_motor.setVelocity(-speed)  
elif predicted_action == 2: # If the action is to move backward.  
    # If the action is to move backward, both motors move in reverse.  
    left_motor.setVelocity(-speed)  
    right_motor.setVelocity(-speed)
```

Chapter

09



Integrations between robotics and AI are shaping the future of numerous industries, transforming how tasks are performed, from manufacturing and healthcare to agriculture and domestic chores. This synergy enhances the capabilities of robots, allowing them to operate more autonomously, make decisions in real-time, and adapt to new or changing environments without human intervention.

In the manufacturing sector, AI-enhanced robots are being used for a variety of tasks including assembly, painting, and quality control. AI algorithms can analyze data from the manufacturing process in real-time, enabling robots to identify and correct errors, optimize production processes, and predict maintenance needs.



Robots equipped with AI are revolutionizing the healthcare industry by assisting in surgeries with precision beyond human capability, managing patient care, and even providing companionship. These robots can learn from data collected during medical procedures, improving their performance over time.



AI-driven robotic systems are being deployed in agriculture for tasks such as planting seeds, harvesting crops, and monitoring field conditions. These robots can analyze data from various sensors and satellite images to optimize crop yield and reduce waste.



Robots, combined with AI, play a crucial role in search and rescue missions, especially in environments that are dangerous or inaccessible to humans. They can navigate through rubble, identify victims using heat sensors, and assess structures' stability.



In the domestic sphere, robotic vacuum cleaners and lawn mowers are becoming smarter through AI, learning the layout of homes and gardens to optimize their cleaning paths. Future developments may lead to more complex domestic robots capable of performing a wide range of household tasks.



Autonomous vehicles, essentially robots on wheels, rely heavily on AI for navigating and making decisions on the road. This integration is crucial for interpreting sensor data, predicting the actions of other road users, and choosing the safest course of action.



AI-powered robots are streamlining logistics and supply chain operations by automating warehousing tasks, such as sorting, packing, and shipping goods. These robots can adapt to changes in inventory levels or demand patterns, improving efficiency and reducing operational costs.





Thank You