| Sl61 | PxIMS2-03 |
| --- | --- |

Smart Waste Sorting Systems

**Data Collection**

1.Waste Types: Collect images and data for different waste types (plastic, paper, metal, organic, etc.). (Search it online or create according to maintain the topic structure )

2.Data Sources: Public datasets, waste management facilities, and manually collected data.

Data Annotation: Label the data for training machine learning models.

**Algorithm Development**

1.Machine Learning: Develop classification models using Python libraries (TensorFlow, Keras, scikit-learn).

2.Computer Vision: Implement image processing techniques with OpenCV.

Integration: Combine machine learning and computer vision for sorting algorithms.

**Simulation Environment**

1.Modelling: Create a simulation environment using SimPy to replicate the waste sorting process.

2.Parameters: Define parameters such as waste input rate, sorting speed, and accuracy.

3.Scenarios: Simulate different scenarios (e.g., mixed waste streams, varying waste volumes).

**Optimization Techniques**

1.Optimization Algorithms: Apply genetic algorithms, particle swarm optimization, or other techniques to improve sorting efficiency.

2.Performance Metrics: Measure sorting accuracy, speed, and throughput.

Iterative Improvement: Continuously refine algorithms based on simulation results.

**Performance Evaluation**

1.Benchmarking: Compare the simulation results with real-world data.

2.Validation: Validate the model’s accuracy and reliability.

Analysis: Analyse the impact of optimised sorting on waste management efficiency.

**Environmental Impact Assessment**

1.Reduction in Contamination: Evaluate how improved sorting reduces contamination in recycling streams.

2.Resource Recovery: Assess the increase in resource recovery rates.

3.Environmental Benefits: Analyse the overall environmental benefits, such as reduced landfill use and lower greenhouse gas emissions.

**Implementation and Testing**

1.Prototype Development: Build a prototype system based on the optimised algorithms.

2.Testing: Test the prototype in a controlled environment with real waste samples.

**Make necessary adjustments based on testing outcomes.**

**Belows codes are just for demo make necessary adjustments**

**Data Collection and Preparation**

import os

import cv2

import numpy as np

from sklearn.model\_selection import train\_test\_split

# Directory paths

data\_dir = 'path\_to\_waste\_images'

categories = ['recyclable', 'compostable', 'landfill']

# Data preparation

data = []

labels = []

for category in categories:

category\_path = os.path.join(data\_dir, category)

class\_num = categories.index(category)

for img in os.listdir(category\_path):

try:

img\_array = cv2.imread(os.path.join(category\_path, img))

resized\_img = cv2.resize(img\_array, (128, 128))

data.append(resized\_img)

labels.append(class\_num)

except Exception as e:

pass

data = np.array(data)

labels = np.array(labels)

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data, labels, test\_size=0.2, random\_state=42)

**Machine Learning Model Developmen**t

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

# Model creation

model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(128, 128, 3)),

MaxPooling2D((2, 2)),

Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Flatten(),

Dense(128, activation='relu'),

Dense(3, activation='softmax')

])

# Compile the model

model.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Train the model

history = model.fit(X\_train, y\_train, epochs=10, validation\_data=(X\_test, y\_test))

**Simulation Environment**

import simpy

def waste\_sorting(env, waste\_bin):

while True:

# Simulate the sorting process

yield env.timeout(1) # Sorting takes 1 time unit

sorted\_waste = model.predict(np.array([waste\_bin.pop()])) # Sort one waste item

print(f"Sorted waste type: {np.argmax(sorted\_waste)}")

# Simulation setup

env = simpy.Environment()

waste\_bin = [X\_test[i] for i in range(50)] # Simulated waste items

env.process(waste\_sorting(env, waste\_bin))

env.run(until=50)

**Optimization Techniques**

from scipy.optimize import differential\_evolution

# Define an optimization function

def optimize\_sorting(params):

# Adjust sorting parameters based on optimization

accuracy, speed = params

model.optimizer.learning\_rate = speed

sorted\_waste = model.predict(X\_test)

accuracy\_score = np.mean(np.argmax(sorted\_waste, axis=1) == y\_test)

return 1 - accuracy\_score # Minimise the error

# Apply differential evolution optimization

result = differential\_evolution(optimize\_sorting, bounds=[(0.001, 0.1), (0.001, 0.01)])

optimized\_params = result.x

print(f"Optimised Parameters: {optimized\_params}")

**Performance Evaluation and Environmental Impact**

# Evaluate model performance

test\_loss, test\_acc = model.evaluate(X\_test, y\_test)

print(f"Test Accuracy: {test\_acc}")

# Environmental impact analysis

contamination\_reduction = (initial\_contamination\_rate - final\_contamination\_rate) / initial\_contamination\_rate \* 100

resource\_recovery\_increase = (final\_recovery\_rate - initial\_recovery\_rate) / initial\_recovery\_rate \* 100

print(f"Contamination Reduction: {contamination\_reduction}%")

print(f"Resource Recovery Increase: {resource\_recovery\_increase}%")