

ME 2041: Fundamentals of Mechatronics

Final Project Report

Smart Waste Management System

Group Members

Index No	Name
210021D	Adikaram W.K.D.N.E.
210139V	Dissanayaka D.A.L.D.
210227N	Igalagama D.P.
210286R	Kelaart D.J.
210375N	Mandis P.L.P.M.
210448R	Pathiraja P.M.J.
210546P	Rubasinghe L.K.
210674F	Vishwanath B.A.M.P.
210445F	Pasandul M.W.M.

Department of Mechanical Engineering

University of Moratuwa.

Abstract

‘Smart Waste Management System’ project aimed to develop an efficient and automated waste sorting system capable of accurately identifying and segregating waste items such as food, polythene, and paper.

The proposed system utilizes image classification technology to analyse the visual characteristics of waste items. An integrated camera captures images of the waste, which are then processed using sophisticated algorithms. These algorithms identify specific features and patterns, enabling the system to accurately classify the waste into different categories.

Once the waste is classified, the system incorporates a smart compressing and sealing mechanism. This feature aims to optimize waste management by reducing the volume of waste, thereby maximizing storage efficiency.

Additionally, the project includes a real-time monitoring system and an innovative waste generation pattern analysis that provides vital information about the waste management process. This monitoring system allows waste management authorities to access real time data regarding the waste volume. By analysing historical data and real-time inputs, the system identifies patterns in waste generation, such as specific waste types, and fluctuations in waste volume. This analysis enables waste management authorities and facility operators to anticipate waste generation patterns and adapt their strategies accordingly.

Content

Abstract.....	2
Content.....	3
Acknowledgement	4
Introduction.....	5
Literature Review.....	6
Image classification Unit	9
Conveyor belt and Sorting Unit	20
Compressing Unit	24
Sealing Unit	27
Real Time Monitoring System.....	30
Analysing Waste Generation Patterns.....	41
Conclusion	46

Acknowledgement

We would like to convey our heartfelt gratitude and appreciation to everyone who helped us to finish our final project report on the subject of "Smart Waste Management System".

First and foremost, we would like to express our gratitude to Prof. Amarasinghe Y.W.R., who served as our project supervisor, for his important direction, encouragement, and support. His knowledge and opinions were very helpful in determining the course of this study.

Our sincere appreciation goes out to Mr. Jayathilaka W.A.D.M. Dumith for his outstanding leadership and assistance during the whole project development process. His insightful feedback and persistent support were extremely important for the success of our project.

We would also like to express our gratitude to the Mechanical Engineering department instructors for their valuable comments and recommendations made during the project development phase. Their contributions significantly raised the overall standard of this study.

We are appreciative of the help and information that our classmates and friends shared with us when the project was still in the planning stages. In order to overcome obstacles and accomplish the project's goals, their cooperation and support have been essential.

Finally, we would want to express our gratitude to everyone who helped this project be completed successfully, whether they were directly involved or not. We appreciate the chance to work on this important project with their help and advice, which have been helpful.

Introduction

Smart waste management system is an innovative approach to managing waste more efficiently and sustainably. It combines advanced technologies, such as, Internet of Things (IoT) connectivity, data analytics, automation, and sorting.

The primary objective of a smart waste management system is to improve waste management practices, reduce environmental impact, and enhance overall efficiency. By incorporating intelligent features, it enables better monitoring, control, and decision-making in waste management operations.

Here are some key components and features of a smart waste management system:

- 1. Waste monitoring:** Sensors and IoT devices are deployed in waste bins or containers to monitor their fill levels in real-time. This data helps optimize waste collection routes and schedules, minimizing unnecessary pickups and reducing fuel consumption.
- 2. Waste sorting and classification:** Smart waste management systems often include mechanisms to sort and classify different types of waste. Image classification is used to identify and separate waste into categories such as paper, plastic, glass, or organic waste.
- 3. Waste compaction:** To maximize the capacity of waste bins, smart systems incorporate compaction mechanisms. This helps compress waste, reducing its volume and extending the time between collections.
- 4. Data analytics and optimization:** Collected data from sensors and other sources are analysed to identify patterns, optimize waste management processes, and make informed decisions. This includes route optimization, demand forecasting, and resource allocation.

Smart waste management systems are gaining traction globally as cities and organizations recognize the need for more sustainable waste management practices. By leveraging technology and data-driven approaches, these systems aim to transform traditional waste management into a smarter, greener, and more efficient process.

Literature Review

Waste management is crucial for protecting the environment, conserving resources, and ensuring public health and safety. Proper waste management practices minimize pollution and prevent hazardous materials from contaminating ecosystems. The scale of the waste management challenge is significant, as improper waste disposal can lead to pollution, hazardous materials contamination, and various environmental and health risks. However, by implementing smart waste management systems, we can address these challenges effectively. These systems integrate technologies such as waste sorting, compression, real-time monitoring, and analysis of waste generation patterns to enhance the efficiency and sustainability of waste management processes.

Sorting

Waste sorting, also known as waste segregation, is a crucial aspect of waste management with several important benefits such as increasing the efficiency of recycling processes, reducing landfill impact, reducing waste disposal costs, and creating potential revenue streams.

There are various existing sorting methods that are utilized in waste management systems.

1. Magnetic separation utilizes powerful magnets to separate ferrous metals from waste materials, allowing for their recovery and recycling.
2. Waste screening systems involve the use of screens or sieves to separate waste based on size or particle characteristics.
3. Sensor-based sorting utilizes technologies such as optical scanners, sensors, and X-rays to identify and sort waste based on specific properties or compositions.
4. X-Ray sorting utilizes X-ray technology to identify and separate waste based on material density or composition.
5. Induction sorting relies on electromagnetic fields to sort conductive materials such as metals.
6. Color sorting uses cameras and optical sensors to detect and sort waste based on colour variations.
7. Near infra-red (NIR) sorting utilizes near infra-red light to analyse waste materials and sort them based on their composition.

8.Laser-Induced Breakdown Spectroscopy (LIBS) sorting uses laser technology to analyse waste materials and identify specific elements for sorting purposes.

The method we implemented in our system is image classification using a deep learning model. Open CV was used to capture images and then they were classified as paper, polythene and waste food.

Compression

Waste compression, which involves compacting waste materials to reduce their size and volume, holds significant importance in waste management for several reasons such as Efficient Space Utilization, Reduced Transportation Costs and Environmental Benefits.

There are several existing waste compressing systems that are commonly used in waste management operations.

1. Compactors, designed to compress waste materials, such as general waste, cardboard, and plastics, into smaller and denser packages, typically using hydraulic or mechanical mechanisms to exert pressure and reduce the volume of waste.
2. Balers that compress waste materials, particularly recyclables like paper, cardboard, plastics, and metal cans, into dense bales.
3. Briquetting Machines, used to compress biomass or organic waste, such as wood chips, sawdust, agricultural residues, or biomass pellets, into compact briquettes.
4. Shredders with compaction which shred the waste into smaller pieces and then compact it, reducing its volume significantly.
5. Pneumatic waste conveying systems which incorporate waste compressors that compress the waste as it moves through the pipes, reducing its volume and facilitating efficient waste transportation and disposal.

The compressing process we used in our system rely on lead screw mechanism which allows a wooden plate to move downwards by actuating a stepper motor. Once the compression happens, it goes upwards back to its original position.

Real time monitoring and analysing waste generation patterns

Real-time monitoring of waste, which allows waste management authorities to track the fill levels of waste bins or containers in real-time, offers numerous advantages in waste management. It facilitates efficient scheduling of waste collection services, provides valuable

insights into waste generation patterns, and effectively reduces the risk of overflow and littering. Many of the existing real-time monitoring systems rely on Internet of Things (IoT)-based solutions that are connected to bin fill level sensors.

Analysing waste generation patterns is significant in waste management for several reasons including ability to identify waste reduction strategies, optimizing resource allocation, decision-making for waste management infrastructure planning etc. These systems rely on a database that functions as a centralized hub for collecting, storing, and effectively managing the data generated within waste management systems. Time series database InfluxDB was used in our system to graphically represent waste generating pattern.

By combining waste sorting, compression, real-time monitoring, and analysis of waste generation patterns, waste management systems can become more efficient, sustainable, and responsive to the needs of communities and the environment.

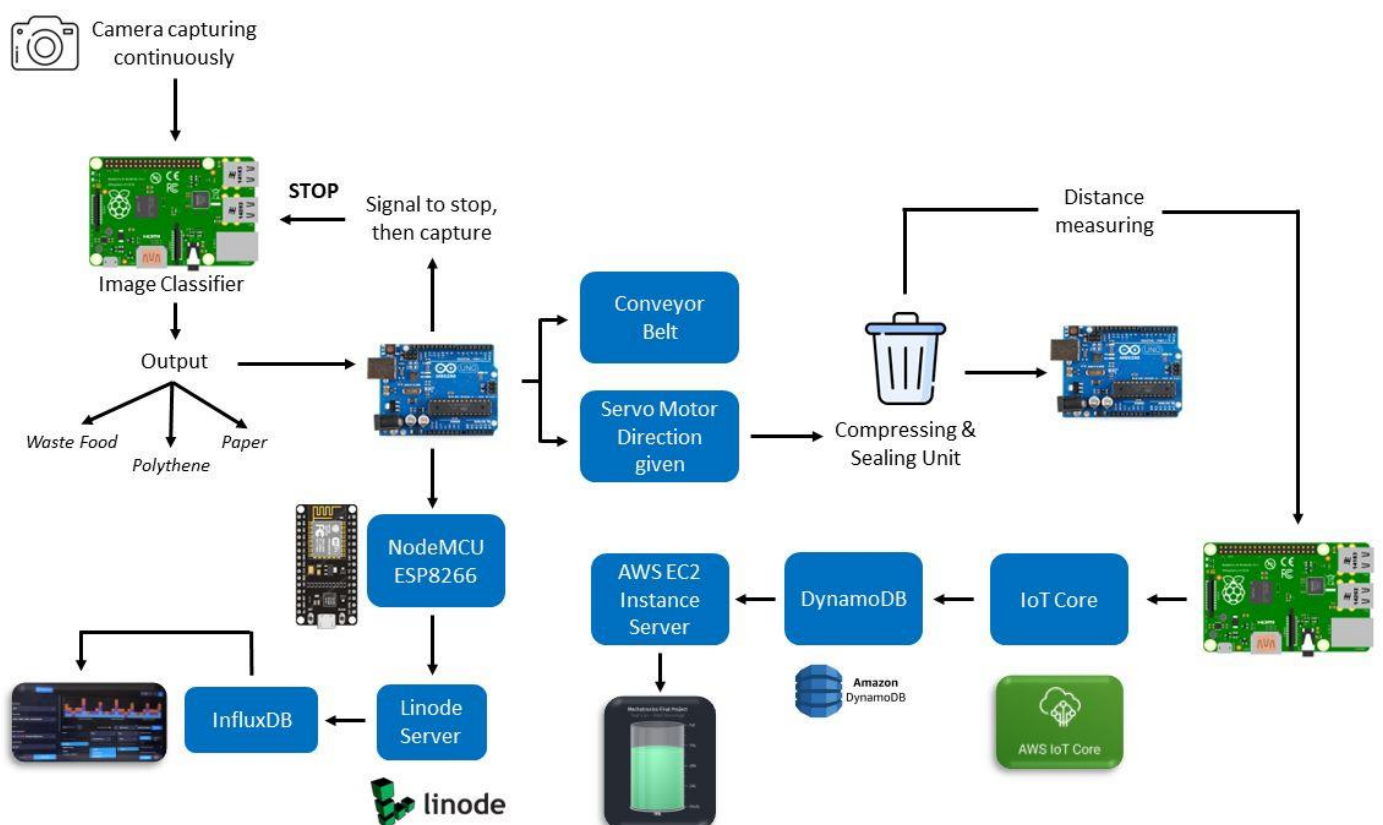


Image Classification Unit

The waste management system includes an image classification unit consisting of a Raspberry Pi single-board computer and a camera module. The objective of this unit is to identify the type of waste material, specifically whether it is paper, polythene, or food waste.

The waste material moves along the conveyor belt until it reaches a position under the camera module, which then captures an image. The Raspberry Pi captures the image using Open CV to identify the waste type, such as paper, polythene, or food waste. The waste type data is sent to the Arduino board, which directs the waste material to the correct bin using a servo motor. This integrated system automates waste type identification and sorting, streamlining the process, reducing manual effort, and ensuring effective waste disposal practices.

CNN model was used for classification since it is parameter efficient compared to MLP (Multilayer Perceptron), PCA (Principal Component Analysis).

Steps followed:

1. TensorFlow, OpenCV-Python, and Matplotlib libraries were installed in the Python virtual environment in Raspberry Pi OS. Then NumPy library was imported into Python script. NumPy is a powerful numerical computing library that provides support for large, multi-dimensional arrays and matrices.
2. Then images for each category: 128 images of waste food, 128 images for polythene, 64 images of paper and 32 null images using Raspberry Pi Camera Module V2 using OpenCV library.
3. The images were divided into 4 classes: 0=null, 1=Paper, 2=Polythene, 3=Waste food.
4. Next, by using the lambda function to normalize image data by scaling the pixel values from the original range of 0 to 255 to the normalized range of 0 to 1.
5. There are 11 batches which each consists of 32 images, and they were divided into 7 batches of train data, 3 batches of validation data, and 1 batch of test data.
6. Keras Sequential API was used and Conv2D, MaxPooling2D, Dense, Flatten, was imported to build the deep learning model.

7. Then, we used Keras Conv2D layer to perform the convolution operation in a two-dimensional space with 32 filters of (3,3). Initial input size of the image had height of 256, width of 256 and 32 channels with the activation of 'relu'.
8. Afterwards, we added Keras MaxPooling2D class to select the maximum value from each pooling window and discards the other values. Then that process was repeated 2 times after that.
9. The output of the last pooling layer, or the final pooled feature map, is fed into a CNN's Multilayer Perceptron (MLP), which can categorize the data into a class label.
10. Next, we used two Keras Dense classes with the objective of creating a class label from the image's observed features. One class was used for 'relu' activation and the other one for 'softmax' activation.
11. Then, in order to reduce the loss function and enhance performance, we applied the Adam optimizer.
12. Subsequently, categorical cross-entropy was used as the loss function. Categorical cross-entropy is commonly used for multi-class classification problems.
13. The accuracy metric was used to measure the percentage of correctly classified samples. After that, the model was compiled.
14. After that the model was trained with the use of validation batches. 20 epoches (entire passing of training data through the algorithm) were used for that.
15. By using 'matplotlib', the loss and the accuracy were represented.
16. Then the evaluation process was done with 'Precision:1.0, Recall:1.0, Accuracy:1.0'.
17. Then the testing process was done, and it classified the images successfully.
18. Finally, the created deep learning model was saved as **.h5** file to store multidimensional arrays of classification data.

1. Install Dependencies and Setup

```
In [1]: !pip install tensorflow opencv-python matplotlib
```

```
In [2]: !pip list
```

Package	Version
absl-py	1.4.0
asttokens	2.2.1
astunparse	1.6.3
backcall	0.2.0
cachetools	5.3.1
certifi	2023.5.7
charset-normalizer	3.1.0
colorama	0.4.6
comm	0.1.3
contourpy	1.1.0
cycler	0.11.0
debugpy	1.6.7
decorator	5.1.1
executing	1.2.0
flatbuffers	23.5.26
fonttools	4.40.0
gast	0.4.0
google-auth	2.21.0
google-auth-oauthlib	1.0.0
google-pasta	0.2.0
grpcio	1.56.0
h5py	3.9.0
idna	3.4
ipykernel	6.24.0
ipython	8.14.0
jax	0.4.13
jedi	0.18.2
jupyter_client	8.3.0
jupyter_core	5.3.1
keras	2.12.0
kiwisolver	1.4.4
libclang	16.0.0
Markdown	3.4.3
MarkupSafe	2.1.3
matplotlib	3.7.1
matplotlib-inline	0.1.6
ml-dtypes	0.2.0
nest-asyncio	1.5.6
numpy	1.23.5
oauthlib	3.2.2
opencv-python	4.8.0.74
opt-einsum	3.3.0
packaging	23.1
parso	0.8.3
pickleshare	0.7.5
Pillow	10.0.0
pip	23.1.2
platformdirs	3.8.0
prompt-toolkit	3.0.39
protobuf	4.23.3
psutil	5.9.5
pure-eval	0.2.2
pyasn1	0.5.0
pyasn1-modules	0.3.0
Pygments	2.15.1
pyparsing	3.1.0

python-dateutil	2.8.2
pywin32	306
pyzmq	25.1.0
requests	2.31.0
requests-oauthlib	1.3.1
rsa	4.9
scipy	1.11.1
setuptools	63.2.0
six	1.16.0
stack-data	0.6.2
tensorboard	2.12.3
tensorboard-data-server	0.7.1
tensorflow	2.12.0
tensorflow-estimator	2.12.0
tensorflow-intel	2.12.0
tensorflow-io-gcs-filesystem	0.31.0
termcolor	2.3.0
tornado	6.3.2
traitlets	5.9.0
typing_extensions	4.7.1
urllib3	1.26.16
wcwidth	0.2.6
Werkzeug	2.3.6
wheel	0.40.0
wrapt	1.14.1

```
In [1]: import tensorflow as tf
import os
```

2. OpenCV Import

```
In [2]: import cv2
```

3. Load Data

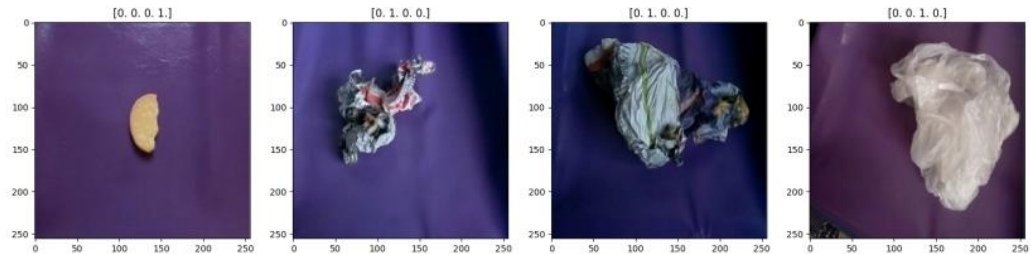
```
In [3]: import numpy as np
from matplotlib import pyplot as plt
```

```
In [47]: data = tf.keras.utils.image_dataset_from_directory('data',label_mode='categorical')
Found 352 files belonging to 4 classes.
```

```
In [46]: data_iterator = data.as_numpy_iterator()
```

```
In [38]: batch = data_iterator.next()
```

```
In [19]: fig, ax = plt.subplots(ncols=4, figsize=(20,20))
         for idx, img in enumerate(batch[0][:4]):
             ax[idx].imshow(img.astype(int))
             ax[idx].title.set_text(batch[1][idx])
```



4. Scale Data

```
In [48]: data = data.map(lambda x,y: (x/255, y))
```

```
In [49]: data.as_numpy_iterator().next()
         [0.30720073, 0.32037377, 0.6028952 ],
         [0.34135816, 0.35704446, 0.6037662 ],
         [0.46623775, 0.47905943, 0.621875  ]],
         ...,
         [[0.14803156, 0.13234529, 0.33234528],
          [0.1576691 , 0.14198282, 0.3419828 ],
          [0.16078432, 0.14509805, 0.34509805],
          ...,
          [0.04976923, 0.09819142, 0.302113 ],
          [0.0363511 , 0.08958333, 0.2658854 ],
          [0.26147038, 0.30460766, 0.41848958]],
         [[0.14613537, 0.13680632, 0.3336277 ],
          [0.15625231, 0.14570644, 0.34313625],
          [0.16285786, 0.14095093, 0.3430245 ],
          ...,
          [0.05193652, 0.1003587 , 0.30428028],
          [0.04601818, 0.09925041, 0.27555248],
```

5. Split Data

```
In [50]: print(len(data))
         print(int(len(data)*.7))
         print(int(len(data)*.3))
         print(int(len(data)*.1))
```

```
11
7
3
1
```

```
In [59]: model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
=====		
conv2d_6 (Conv2D)	(None, 254, 254, 32)	896
max_pooling2d_5 (MaxPooling 2D)	(None, 127, 127, 32)	0
conv2d_7 (Conv2D)	(None, 125, 125, 64)	18496
max_pooling2d_6 (MaxPooling 2D)	(None, 62, 62, 64)	0
conv2d_8 (Conv2D)	(None, 60, 60, 64)	36928
max_pooling2d_7 (MaxPooling 2D)	(None, 30, 30, 64)	0
flatten_2 (Flatten)	(None, 57600)	0
dense_4 (Dense)	(None, 64)	3686464
dense_5 (Dense)	(None, 4)	260
=====		
Total params: 3,743,044		
Trainable params: 3,743,044		
Non-trainable params: 0		

7. Train

```
In [60]: logdir='logs'
```

```
In [61]: tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)
```

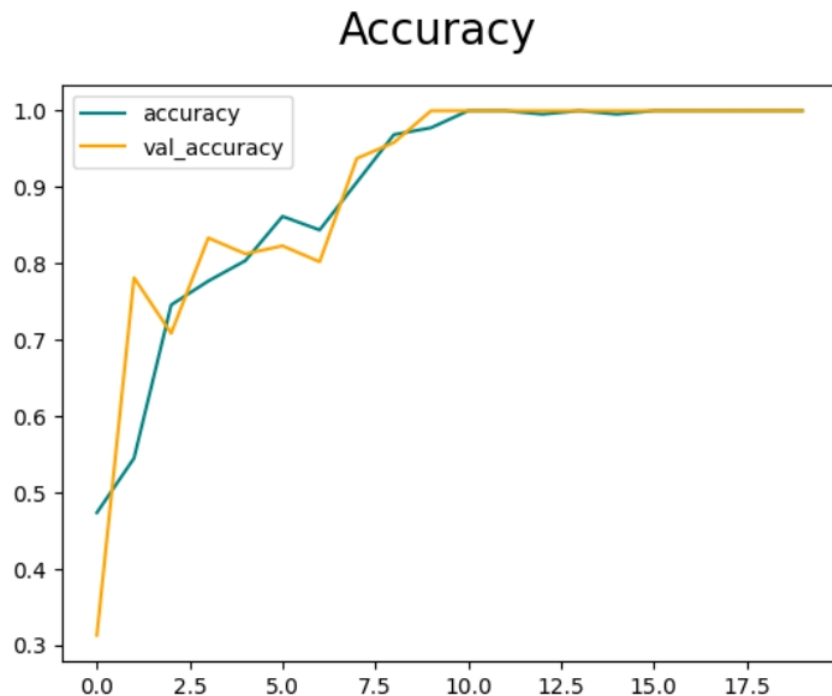
```
In [62]: hist = model.fit(train, epochs=20, validation_data=val, callbacks=[tensorboard_cal.
```

```

Epoch 1/20
7/7 [=====] - 15s 2s/step - loss: 1.5086 - accuracy: 0.4
732 - val_loss: 1.2389 - val_accuracy: 0.3125
Epoch 2/20
7/7 [=====] - 16s 2s/step - loss: 1.0517 - accuracy: 0.5
446 - val_loss: 0.7685 - val_accuracy: 0.7812
Epoch 3/20
7/7 [=====] - 17s 2s/step - loss: 0.7047 - accuracy: 0.7
455 - val_loss: 0.7329 - val_accuracy: 0.7083
Epoch 4/20
7/7 [=====] - 18s 2s/step - loss: 0.5625 - accuracy: 0.7
768 - val_loss: 0.3939 - val_accuracy: 0.8333
Epoch 5/20
7/7 [=====] - 17s 2s/step - loss: 0.4252 - accuracy: 0.8
036 - val_loss: 0.3536 - val_accuracy: 0.8125
Epoch 6/20
7/7 [=====] - 17s 2s/step - loss: 0.2791 - accuracy: 0.8
616 - val_loss: 0.3142 - val_accuracy: 0.8229
Epoch 7/20
7/7 [=====] - 17s 2s/step - loss: 0.2726 - accuracy: 0.8
438 - val_loss: 0.3380 - val_accuracy: 0.8021
Epoch 8/20
7/7 [=====] - 17s 2s/step - loss: 0.2079 - accuracy: 0.9
062 - val_loss: 0.1587 - val_accuracy: 0.9375
Epoch 9/20
7/7 [=====] - 17s 2s/step - loss: 0.1427 - accuracy: 0.9
688 - val_loss: 0.1356 - val_accuracy: 0.9583
Epoch 10/20
7/7 [=====] - 17s 2s/step - loss: 0.0956 - accuracy: 0.9
777 - val_loss: 0.1036 - val_accuracy: 1.0000
Epoch 11/20
7/7 [=====] - 17s 2s/step - loss: 0.0721 - accuracy: 1.0
000 - val_loss: 0.0538 - val_accuracy: 1.0000
Epoch 12/20
7/7 [=====] - 18s 3s/step - loss: 0.0372 - accuracy: 1.0
000 - val_loss: 0.0149 - val_accuracy: 1.0000
Epoch 13/20
7/7 [=====] - 17s 3s/step - loss: 0.0178 - accuracy: 0.9
955 - val_loss: 0.0138 - val_accuracy: 1.0000
Epoch 14/20
7/7 [=====] - 17s 2s/step - loss: 0.0072 - accuracy: 1.0
000 - val_loss: 0.0020 - val_accuracy: 1.0000
Epoch 15/20
7/7 [=====] - 17s 2s/step - loss: 0.0112 - accuracy: 0.9
955 - val_loss: 0.0211 - val_accuracy: 1.0000
Epoch 16/20
7/7 [=====] - 17s 3s/step - loss: 0.0145 - accuracy: 1.0
000 - val_loss: 0.0029 - val_accuracy: 1.0000
Epoch 17/20
7/7 [=====] - 17s 2s/step - loss: 0.0068 - accuracy: 1.0
000 - val_loss: 0.0028 - val_accuracy: 1.0000
Epoch 18/20
7/7 [=====] - 17s 2s/step - loss: 0.0042 - accuracy: 1.0
000 - val_loss: 7.7573e-04 - val_accuracy: 1.0000
Epoch 19/20
7/7 [=====] - 17s 2s/step - loss: 0.0017 - accuracy: 1.0
000 - val_loss: 0.0013 - val_accuracy: 1.0000
Epoch 20/20

```

```
In [64]: fig = plt.figure()
plt.plot(hist.history['accuracy'], color='teal', label='accuracy')
plt.plot(hist.history['val_accuracy'], color='orange', label='val_accuracy')
fig.suptitle('Accuracy', fontsize=20)
plt.legend(loc="upper left")
plt.show()
```



9. Evaluate

```
In [69]: from tensorflow.keras.metrics import Precision, Recall, CategoricalAccuracy
```

```
In [70]: pre = Precision()
re = Recall()
acc = CategoricalAccuracy()
```

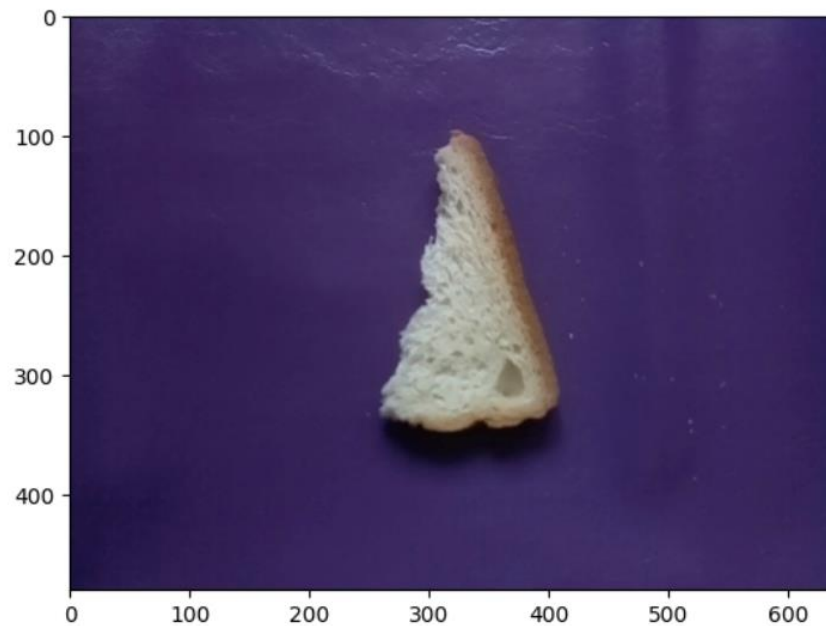
```
In [71]: for batch in test.as_numpy_iterator():
X, y = batch
yhat = model.predict(X)
pre.update_state(y, yhat)
re.update_state(y, yhat)
acc.update_state(y, yhat)
```

1/1 [=====] - 0s 422ms/step

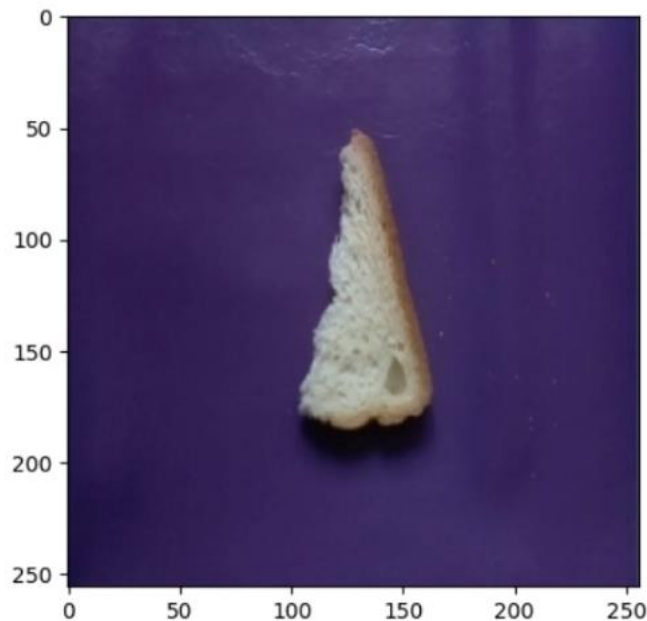

```
In [72]: print(f'Precision:{pre.result().numpy()},Recall:{re.result().numpy()} , Accuracy:{a  
Precision:1.0,Recall:1.0 , Accuracy:1.0
```

10. Test

```
In [107]: img = cv2.cvtColor(cv2.imread('103.jpg'),cv2.COLOR_BGR2RGB)  
plt.imshow(img)  
plt.show()
```



```
In [108]: resize = tf.image.resize(img, (256,256))
plt.imshow(resize.numpy().astype(int))
plt.show()
```



```
In [109]: yhat = model.predict(np.expand_dims(resize/256, axis=0))
1/1 [=====] - 0s 34ms/step
```

```
In [110]: yhat
```

```
Out[110]: array([[1.8166994e-06, 5.2435706e-08, 4.3512041e-06, 9.9999380e-01]],
dtype=float32)
```

```
In [111]: predicted_class_index = np.argmax(yhat)
```

```
In [112]: predicted_class_index
```

```
Out[112]: 3
```

```
In [113]: if predicted_class_index == 0:
print(f'Null')
elif predicted_class_index == 2:
print(f'Polythene')
elif predicted_class_index == 3:
print(f'Wastefood')
else:
print(f'Paper')
```

Wastefood

11. Save the Model

```
In [114]: from tensorflow.keras.models import load_model
```

```
In [115]: model.save(os.path.join('models', 'imageclassifier.h5'))
```

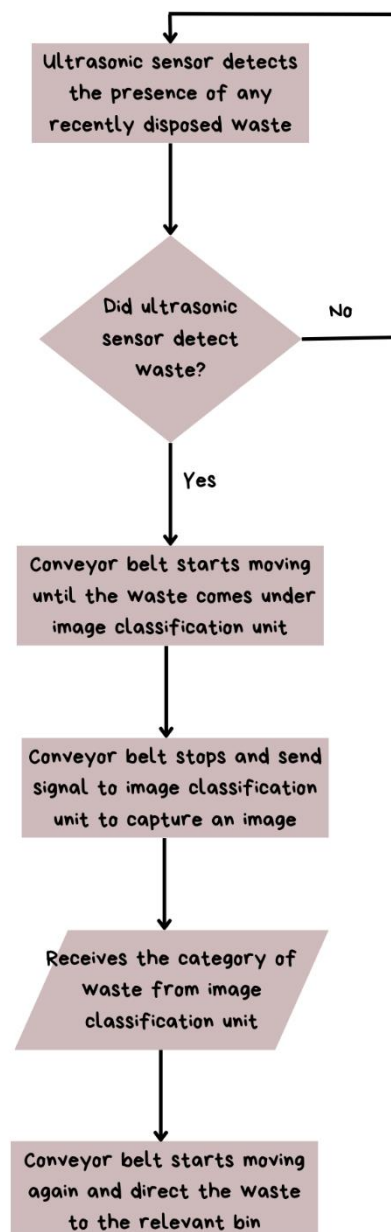
Classification Python code

This is the main code runs in raspberry pi single board computer, all other codes are connected through this code.

```
D: > img.py > ...
1  import tensorflow as tf
2  import numpy as np
3  import cv2
4  import time
5  import serial
6
7  # Load the trained model
8  model = tf.keras.models.load_model('/home/pi/Desktop/imageclassifier.h5')
9
10 # Define the class labels
11 class_labels = ['Null', 'Paper', 'Polythene', 'Waste food',]
12
13 # Load and preprocess the image
14 def preprocess_image(image):
15     #image = cv2.cvtColor(cv2.imread(image), cv2.COLOR_BGR2RGB)
16     image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
17     image = tf.image.resize(image, (256, 256)) # Resize to match the input size of the model
18     image = tf.cast(image, tf.float32) / 255.0
19     #image = image.astype('float32') / 255.0 # Normalize the pixel values
20     image = np.expand_dims(image, axis=0) # Add batch dimension
21     return image
22
23 # Capture and classify a single photo
24 def classify_photo():
25     # Open the video capture
26     cap = cv2.VideoCapture(0)
27
28     # Set up Arduino serial communication
29     arduino = serial.Serial('/dev/ttyACM0', 9600) # Replace with the correct port and baud rate
30
31     while True:
32         # Read a frame from the video capture
33         ret, frame = cap.read()
34
35         # Display the frame
36         cv2.imshow('Camera', frame)
37
38         # Check for incoming signal from Arduino
39         if arduino.in_waiting > 0:
40             signal = arduino.readline().decode().strip()
41             if signal == 'STOP':
42                 # Stop the conveyor belt
43
44                 # Capture photo
45                 # Preprocess the captured photo
46                 preprocessed_frame = preprocess_image(frame)
47
48                 # Make predictions
49                 predictions = model.predict(preprocessed_frame)
50                 predicted_class = np.argmax(predictions)
51                 class_label = class_labels[predicted_class]
52
53                 # Display the predicted class on the frame
54                 cv2.putText(frame, class_label, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
55                 cv2.imshow('Classification', frame)
56
57                 # Send the class label to Arduino
58                 arduino.write(class_label.encode())
59
60                 # Wait for 2 seconds
61                 time.sleep(2)
62
63                 # Start the conveyor belt
64                 arduino.write(b'START\n')
65
66             # Exit if 'q' is pressed
67             if cv2.waitKey(1) == ord('q'):
68                 break
69
70     # Release the video capture
71     cap.release()
72     cv2.destroyAllWindows()
73 # Start capturing and classifying photos
74 classify_photo()
```

Conveyor belt and Sorting Unit

After the waste materials are disposed onto the conveyor belt, the ultrasonic sensor detects their presence. Once detected, the conveyor belt starts moving using a 12V DC motor. The waste is then directed to an image classification unit for waste type identification. After the waste type is identified by the image classification unit, the data is sent to an Arduino board, which subsequently directs the waste towards the relevant bin by actuating a servo motor.



```

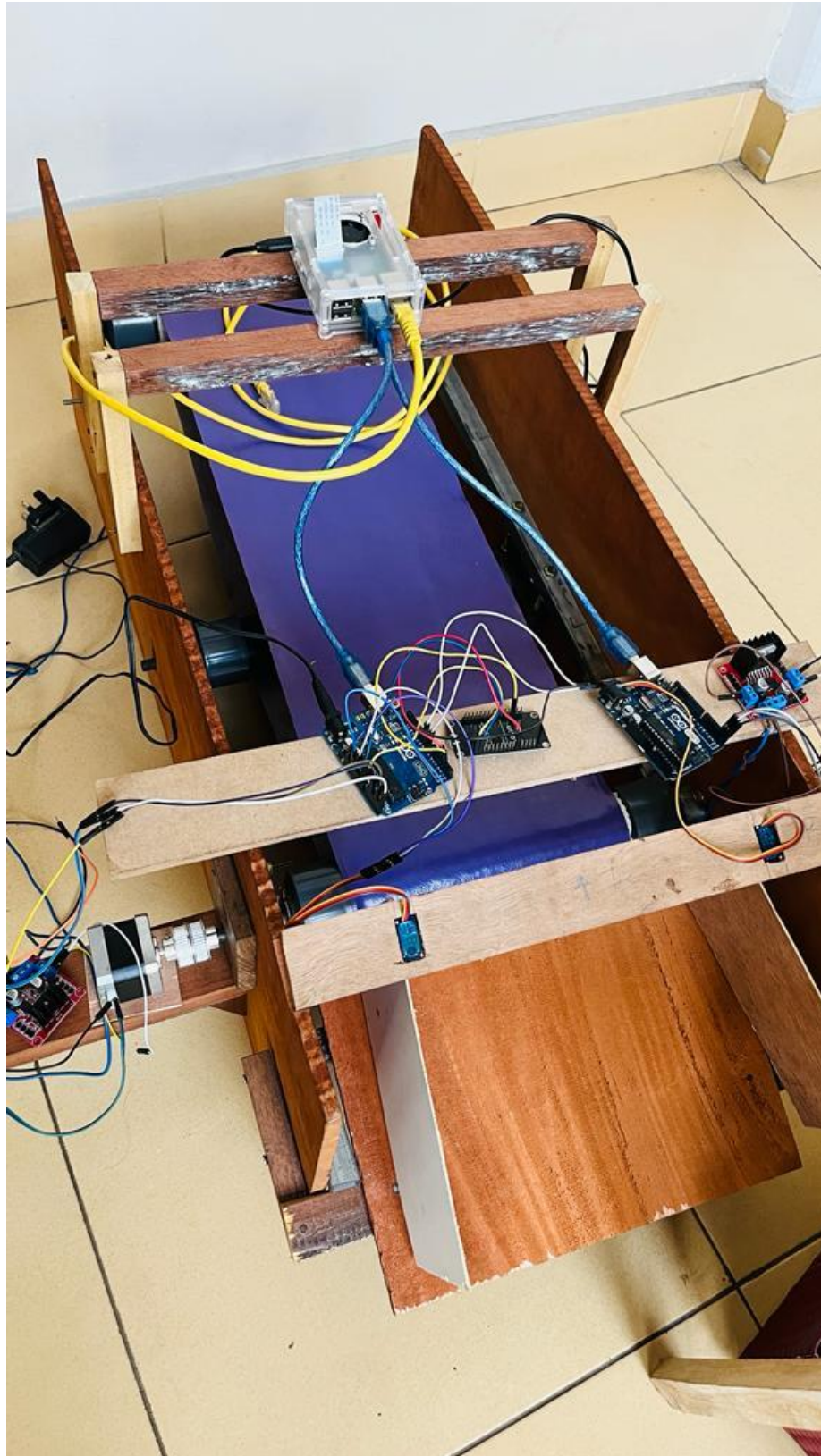
1  #include <Servo.h>
2  Servo servol; // create servo object for left servo
3  Servo servor; // create servo object for right servo
4
5  // Define pins for ultrasonic sensor
6  int trigPin = 3;
7  int echoPin = 4;
8  int in_1 = 8 ;
9  int in_2 = 9 ;
10 int enA = 5;
11 String data;
12
13 // Define variables for measuring distance
14 long duration;
15 float distance;
16
17 void setup() {
18
19
20 servol.attach(6); // attach first servo to pin 6
21 servor.attach(7); // attach second servo to pin 7
22
23 // Initialize serial communication at 9600 bits per second
24 Serial.begin(9600);
25
26 // Set trigPin as output and echoPin as input
27 pinMode(trigPin, OUTPUT);
28 pinMode(echoPin, INPUT);
29 pinMode(enA, OUTPUT);
30 pinMode(in_1, OUTPUT);
31 pinMode(in_2, OUTPUT);
32
33
34 // Turn off the motor - Initial state
35
36 digitalWrite(in_1, LOW);
37 digitalWrite(in_2, LOW);
38
39
40
41 // get output to light the led
42 pinMode(13, OUTPUT);
43 // Allow time for sensor to settle
44 delay(1000);
45 }
46
47 void loop() {
48 // Send a 10 microsecond pulse to trigger the sensor
49 digitalWrite(trigPin, LOW);
50 delayMicroseconds(5);
51 digitalWrite(trigPin, HIGH);
52 delayMicroseconds(10);
53 digitalWrite(trigPin, LOW);
54
55 // Measure the duration of the echo pulse in microseconds
56 duration = pulseIn(echoPin, HIGH);
57
58 // Calculate the distance in centimeters
59 distance = duration * 0.034 / 2;
60
61 // Print the distance to the serial monitor
62 Serial.print("Distance: ");
63 Serial.print(distance);
64 Serial.println(" cm");
65

```

```

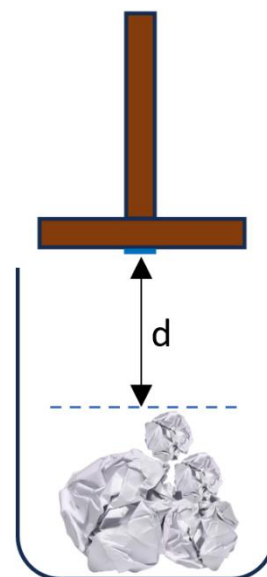
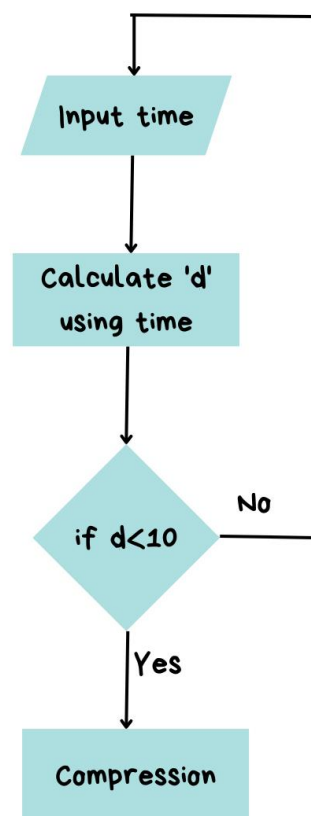
66 // 30cm = Width of the walls of the conveyor belt
67 if (distance < 30) {
68     digitalWrite(in_1 , HIGH);
69     digitalWrite(in_2 , LOW);
70     delay(1000); // The time it takes to take the garbage from initial position to the IMGC unit
71     digitalWrite(in_1, LOW);
72     digitalWrite(in_2 , LOW);
73
74     String data = "STOP"; // The string you want to send
75     Serial.println(data); // Send the string data through the serial port
76     delay(1000); // Delay between sending each string (optional)
77 // Garbage is positioned below the camera module
78
79 if (Serial.available()>0) { // check if there's incoming data
80     String data = Serial.readString(); // read the incoming data until the terminating character
81     Serial.println(data); // print the received data
82 }
83
84
85 // If the servo rotates counter clock wise, initial (0) angle should be pointing to right hand side
86 // Then initial position is angled 45 degrees to right hand side (Paper)
87 // if data == Wastefood, the servo should be angled at 45 degrees from servo motor's initial position
88 // if data == plastic, the servo should be angled at 90 degrees from servo motor's initial position
89 if (data == "Paper") {
90     digitalWrite(in_1 , HIGH);
91     digitalWrite(in_2 , LOW); // Drives the conveyor belt forward
92     int angle1 = 0; // set the angle required for left servo
93     // Flap is angled at 45 degrees to right hand side (From forward direction)
94     int angle2 = 0; // set the angle required for right servo
95     servoL.write(angle1); // move first servo to the desired angle
96     servoR.write(angle2); // move second servo to the desired angle
97     delay(3000);}
98
99
100 if (data == "Wastefood") {
101     digitalWrite(in_1 , HIGH);
102     digitalWrite(in_2 , LOW); // Drives the conveyor belt forward
103     int angle1 = 45; // set the angle required for left servo
104     // Flap is pointed parallel to the forward direction
105     int angle2 = 45; // set the angle required for right servo
106     servoL.write(angle1); // move first servo to the desired angle
107     servoR.write(angle2); // move second servo to the desired angle
108     delay(3000); }
109
110 if (data == "Polythene") {
111     digitalWrite(in_1 , HIGH);
112     digitalWrite(in_2 , LOW); // Drives the conveyor belt forward
113     int angle1 = 90; // set the angle required for left servo
114     // Flap is angled at 45 degrees to left hand side
115     int angle2 = 90; // set the angle required for right servo
116     servoL.write(angle1); // move first servo to the desired angle
117     servoR.write(angle2); // move second servo to the desired angle
118     delay(3000); }
119
120 else {
121     digitalWrite(in_1 , LOW);
122     digitalWrite(in_2 , HIGH);
123     delay(1000); // The time it takes to take the garbage from initial position to the IMGC unit (Same time as mentioned above)
124     digitalWrite(in_1, LOW);
125     digitalWrite(in_2 , LOW);}
126 }
127
128 else {
129     digitalWrite(in_1, LOW);
130     digitalWrite(in_2 , LOW);
131 }
132
133
134
135

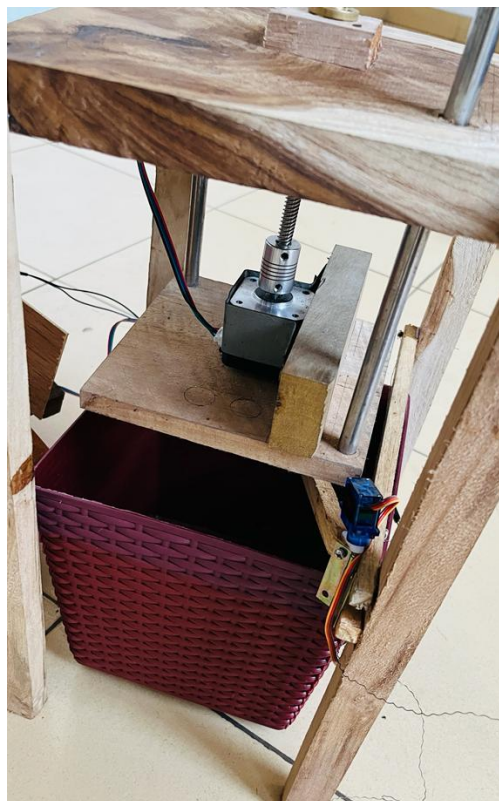
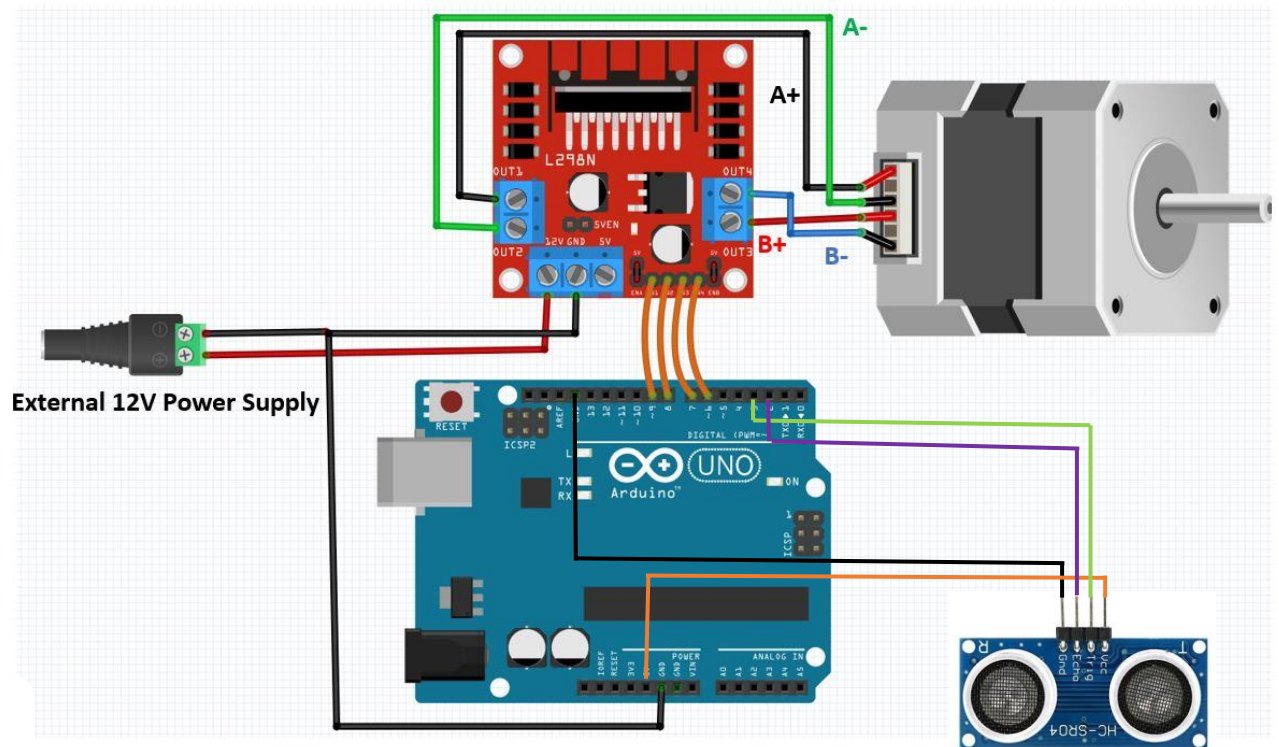
```

Compressing Unit

The objective of the compressing unit in your waste management system is to automate the compression process to maximize the capacity inside the waste bin. The unit comprises a **stepper motor**, **L298N dual H-bridge motor driver**, and an **ultrasonic sensor**, all connected to an **Arduino Uno microcontroller**. The ultrasonic sensor measures the height of the waste load inside the container, and once it reaches 10cm, the compressing action is triggered by stepper motor connected through L298N dual H-bridge motor driver. It uses lead screw mechanism to move the compressing plate downwards and then go upwards to its initial position.





```

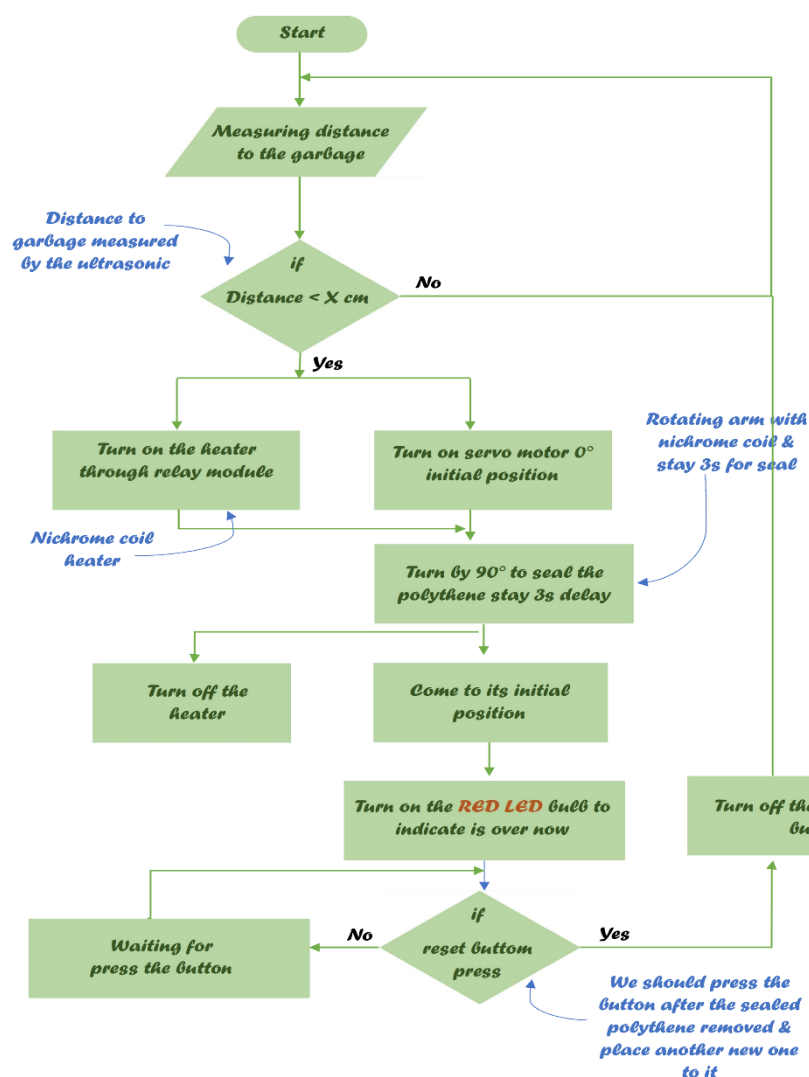
1  #include <Stepper.h>
2
3  // Define the pins for the ultrasonic sensor
4  const int trigPin = 2;
5  const int echoPin = 3;
6
7  // Define the maximum distance for the ultrasonic sensor to detect objects
8  const int maxDistance = 10; // in centimeters
9
10 // Define the number of steps per revolution
11 const int stepsPerRevolution=200;
12
13 // Create a Stepper object with the number of steps per revolution and the motor control pins
14 Stepper myStepper(stepsPerRevolution,8,9,10,11);
15
16 void setup() {
17     // Set the speed of the stepper motor
18     myStepper.setSpeed(60);
19
20     // Start the serial communication for debugging purposes
21     Serial.begin(9600);
22
23     // Set up the ultrasonic sensor pins
24     pinMode(trigPin, OUTPUT);
25     pinMode(echoPin, INPUT);
26 }
27
28 void loop() {
29     // Measure the distance with the ultrasonic sensor
30     long duration, distance;
31     digitalWrite(trigPin, LOW);
32     delayMicroseconds(2);
33     digitalWrite(trigPin, HIGH);
34     delayMicroseconds(10);
35     digitalWrite(trigPin, LOW);
36     duration = pulseIn(echoPin, HIGH);
37     distance = duration / 58; // Calculate 'd'
38
39     // Debug output
40     Serial.print("Distance: ");
41     Serial.print(distance);
42     Serial.println(" cm");
43
44     // Check if an object is within range
45     if (distance<10){
46
47         myStepper.step(-stepsPerRevolution*10); // Compressing plate comes down
48         delay(1000);
49
50         myStepper.step(stepsPerRevolution*10); // Compressing plate goes up
51         delay(1000);
52     }
53 }
54

```

Sealing Unit

The sealing unit is typically equipped with a mechanism that tightly seals the compressed waste, preventing any potential spills or leaks during storage or transportation. This helps maintain cleanliness and hygiene in waste management processes.

The sealing process is automated and activated based on predetermined parameters, ensuring consistent and reliable sealing of the waste. Heat sealing mechanism is used here.



C: > Users > Lochinie > OneDrive > Desktop >  sealing

```
1  #include <Servo.h>
2
3  // Constants
4  // (1)-Ultra Sonic & other pins
5
6  const int trigPin = 2;
7  const int echoPin = 3;
8  const int buttonPin = 4;      // to take the input of the button
9  const int endLED = 6;        // to indicate that the sealing process is over by red LED
10 const int coilPin = 7;        // power to coil
11 const int servoPin = 9;       // servo motor
12
13 // (2)-Other measurements
14
15 const int garbageDistance = 10; // Distance to garbage in cm
16 const int sealingtime = 3000;   // sealing time in ms
17
18 // for if loop
19 bool isLoopStopped = false;     // Variable to track if the loop is stopped
20
21 // Variables
22 long duration;
23 int distance;
24
25 // Create servo object
26 Servo servo;
27
28 void setup() {
29
30     // Initialize serial communication
31     Serial.begin(9600);
32
33     // Set trigPin as output and echoPin as input
34     pinMode(trigPin, OUTPUT);
35     pinMode(echoPin, INPUT);
36
37     // other
38     pinMode(buttonPin, INPUT_PULLUP);
39     pinMode(endLED, OUTPUT);
40     pinMode(coilPin, OUTPUT);
41
42     // Attach servo to the servoPin
43     servo.attach(servoPin);
44 }
45
46 void loop() {
47
48     if (!isLoopStopped) {
49
50         // Measure distance
51         digitalWrite(trigPin, LOW);
52         delayMicroseconds(2);
53         digitalWrite(trigPin, HIGH);
54         delayMicroseconds(10);
55         digitalWrite(trigPin, LOW);
56
57         //distance calculator
58         duration = pulseIn(echoPin, HIGH);
59         distance = duration * 0.034 / 2;
60
61         // Print distance to the serial monitor
62         Serial.print("Distance: ");
63         Serial.print(distance);
64         Serial.println(" cm");           //println is for moves cursor to a new line Distance: 5 cm like
65
66         servo.write(0);                  //initial position
67     }
```

```

68 // Check if distance is less than the garbage distance
69 if (distance < garbageDistance) {
70
71     servo.write(90);           // Turn the servo motor by 90 degrees
72     digitalWrite(coilPin,HIGH); // supply current to coil through relay module
73     delay(sealingtime);       // wait for the servo to reach the desired position (Sealing time)
74     digitalWrite(coilPin,LOW); // off the power supply to coil
75     servo.write(0);           // Reset the servo to the initial position (0 degrees)
76     digitalWrite(endLED,HIGH); // light the red LED to indicate the process is over
77
78     Serial.println("Distance is below 10 cm. Stopping the loop.");
79
80     isLoopStopped = true;
81
82 }
83
84
85 // Check if the button is pressed to restart the loop
86 if (digitalRead(buttonPin) == LOW) {
87
88     digitalWrite(endLED,LOW); // Wait for the servo to reach the desired position
89     Serial.println("Button pressed. Restarting the loop.");
90     isLoopStopped = false;
91
92 }
93 delay(100); // Delay between measurements
94
95 }

```

Real Time Monitoring System

Transferring Distances Measured by Ultrasonic Sensor to the Raspberry pi

We transmitted the distance measurements obtained from the ultrasonic sensor of the compressing unit to AWS IoT Core via Raspberry Pi for real-time display in the monitoring system.

Below is the Python program written in the Raspberry Pi IDE to read the data transmitted by the Arduino via the serial port:

```

C:\Users\Lochinie> OneDrive\Desktop> G sealing
1  import serial
2
3  # Configure the serial port
4  ser = serial.Serial('/dev/ttyACM0', 9600, timeout=1) # Use the appropriate serial port
5
6  while True:
7      line = ser.readline().decode().strip()
8      if line.startswith("Distance:"):
9          distance = int(line.split()[1])
10         print("Distance:", distance, "cm")
11

```

Then, distances were uploaded to the AWS IoT core.

Transferring Distances to the AWS IoT core from Raspberry pi

Steps followed:

Set up AWS IoT Core:

1. Created an AWS IoT Core Resource: In the AWS Management Console, accessed the AWS IoT Core service. Created an IoT Core resource, which serves as the central hub for managing devices, communication, and data processing within the IoT ecosystem.
2. Created a Thing: Created a Thing in AWS IoT Core to represent the raspberry pi. This was done by providing a unique name for the Thing and additional metadata that describes the raspberry pi.

3. Defined a Device Certificate: Within AWS IoT Core, generated a device certificate for the Thing we created. This certificate consists of a public key and a private key that the device uses to securely connect and communicate with the IoT Core service.
4. Created an IoT Policy: Created an IoT policy that specifies the allowed operations, such as publishing or subscribing to MQTT topics, connecting to AWS IoT Core, or accessing relevant resources.
5. Attached the Certificate and Policy to the Thing: Associated the previously created device certificate and IoT policy with the corresponding Thing in AWS IoT Core.

Installed AWS SDK:

Opened a terminal on the Raspberry Pi and execute the command **`pip install AWSIoTPythonSDK`**.

Device Connection:

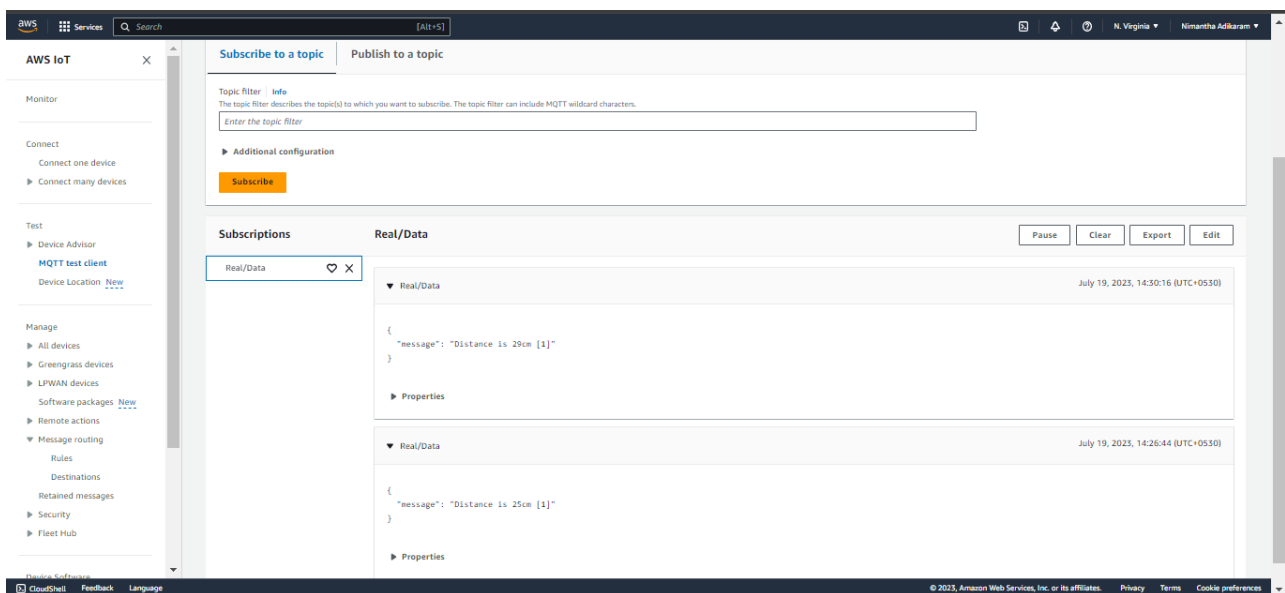
1. AWS IoT Core Endpoint: Obtained the endpoint for AWS IoT Core instance.
2. Client ID: Generated a unique client ID to identify the Raspberry Pi device when connecting to AWS IoT Core.
3. Security Credentials: Obtained necessary security credentials, including the device certificate, private key, and Amazon Root CA certificate, which were previously generated during the AWS IoT Core setup process.
4. Connection Object: In the Python code running on the Raspberry Pi, used the AWS IoT Device SDK to create a connection object. This was done using the **`mqtt_connection_builder`** module and the **`mtls_from_path`** method.
5. Configure Connection Parameters: Set the connection parameters for the MQTT connection object by specifying the AWS IoT Core endpoint, client ID, and other optional parameters such as port number, clean session, and keep-alive time.
6. Established the Connection: Call the **`connect ()`** method on the MQTT connection object to initiate the connection to AWS IoT Core.
7. Connection Confirmation: After calling **`connect ()`**, used the **`result ()`** method to wait until the connection is successfully established. This ensures that the Raspberry Pi has successfully connected to AWS IoT Core before proceeding with further actions.

Data Publishing:

Wrote the code to publish data to AWS IoT Core. It involved creating a message payload with the data we wanted to publish and using the AWS SDK to publish the message to the relevant topic on AWS IoT Core.

Subscribed to Topics:

1. MQTT Connection: Activated MQTT connection object established with AWS IoT Core. This connection object was created using the **mqtt_connection_builder** module from the AWS IoT Device SDK.
2. Topic Subscription: The `subscribe ()` method was called on the MQTT connection object and took the relevant topic as input.
3. Quality of Service (QoS): Specified the Quality of Service (QoS) level as 1 for the topic subscription.
4. Message Handler: After subscribing to a topic, implemented a message handler function.
5. Subscription Confirmation: Used the **result ()** method to wait for the subscription confirmation.



Python code on the Raspberry pi:

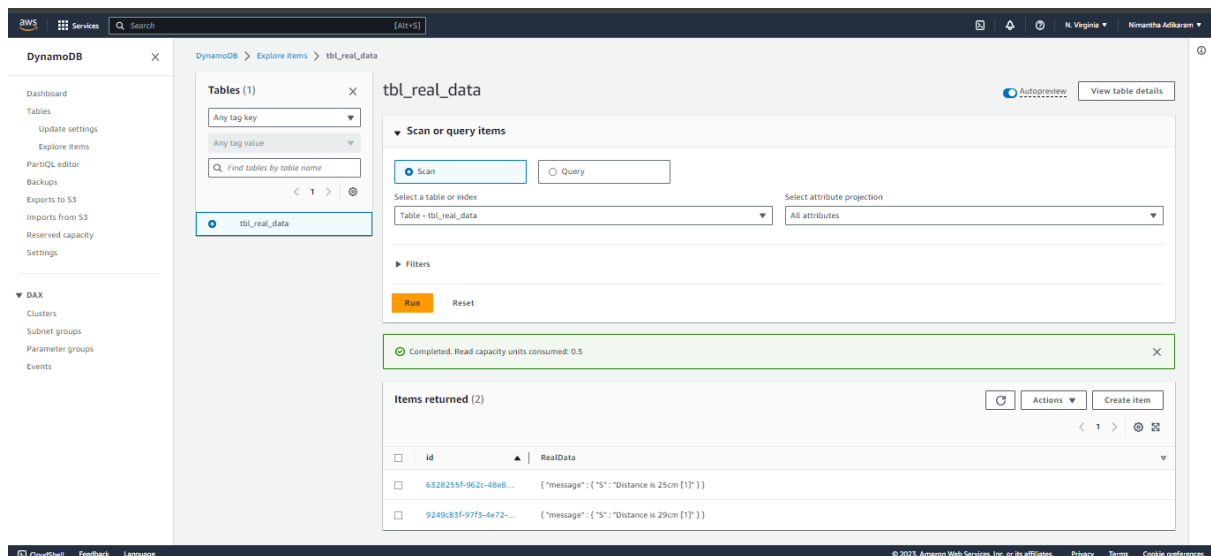
```
C: > Users > Lochinie > OneDrive > Desktop > aws.py > ...
1  from awscrt import io, mqtt, auth, http
2  from awsiot import mqtt_connection_builder
3  import time as t
4  import json
5
6
7  # Define ENDPOINT, CLIENT_ID, PATH_TO_CERTIFICATE, PATH_TO_PRIVATE_KEY, PATH_TO_AMAZON_ROOT_CA_1, MESSAGE, TOPIC, and RANGE
8  ENDPOINT = "a5jcw2pa8d9m-ats.iot.us-east-1.amazonaws.com"
9  CLIENT_ID = "G5Mech"
10 PATH_TO_CERTIFICATE = "/home/pi/Desktop/RaspberryFiles/certificate.pem.crt"
11 PATH_TO_PRIVATE_KEY = "/home/pi/Desktop/RaspberryFiles/private.pem.key"
12 PATH_TO_AMAZON_ROOT_CA_1 = "/home/pi/Desktop/RaspberryFiles/root-ca.pem"
13 DISTANCE = distance
14 TOPIC = "Real/Data"
15 RANGE = 10
16
17 # Spin up resources
18 event_loop_group = io.EventLoopGroup(1)
19 host_resolver = io.DefaultHostResolver(event_loop_group)
20 client_bootstrap = io.ClientBootstrap(event_loop_group, host_resolver)
21 mqtt_connection = mqtt_connection_builder.mtls_from_path(
22     endpoint=ENDPOINT,
23     cert_filepath=PATH_TO_CERTIFICATE,
24     pri_key_filepath=PATH_TO_PRIVATE_KEY,
25     client_bootstrap=client_bootstrap,
26     ca_filepath=PATH_TO_AMAZON_ROOT_CA_1,
27     client_id=CLIENT_ID,
28     clean_session=False,
29     keep_alive_secs=6
30 )
31 print("Connecting to {} with client ID '{}'...".format(
32     ENDPOINT, CLIENT_ID))
33 # Make the connect() call
34 connect_future = mqtt_connection.connect()
35 # Future.result() waits until a result is available
36 connect_future.result()
37 print("Connected!")
38 # Publish message to server desired number of times.
39 print('Begin Publish')
40 for i in range (RANGE):
41     data = "{} [{}]" .format(DISTANCE, i+1)
42     message = {"message" : data}
43     mqtt_connection.publish(topic=TOPIC, payload=json.dumps(message), qos=mqtt.QoS.AT_LEAST_ONCE)
44     print("Published: " + json.dumps(message) + " to the topic: " + "'test/testing'")
45     t.sleep(0.1)
46 print('Publish End')
47 disconnect_future = mqtt_connection.disconnect()
48 disconnect_future.result()
49
```

Visualizing the filled percentage of the trash bin

The trash bin is filled with time accordingly. The filled distance is measured by an ultrasonic sensor and the distance value is sent through raspberry pi to the AWS IoT Core. It creates a JSON file with key-value pairs and from the “payload” key the distance value is stored. This JSON file is stored in the DynamoDB which is a NoSQL database service provided by AWS. The below code connects a front-end application to a back-end server built with Express.js and establishes communication between the server and a DynamoDB database. It allows clients to retrieve and update the “**trashCanFilledPercentage**” value while keeping all connected clients in sync with the latest value using Socket. IO.

Here's a breakdown of how these components is connected:

1. Creating an AWS DynamoDB



Created a table in the DynamoDB console, where I specified various table details such as the table name, primary key, and table settings.

To ensure optimal performance and scalability, made decisions regarding the table settings.

With all the necessary details specified, proceeded to create the table by clicking the "Create" button within the DynamoDB console. This initiated the creation process, and upon completion, DynamoDB table was successfully created.

2. **Database (AWS DynamoDB):** The code uses the AWS SDK to interact with DynamoDB, a NoSQL database service. It creates an instance of the DynamoDB DocumentClient, which provides an easy-to-use API for working with DynamoDB tables. The code includes functions to retrieve and update data in DynamoDB.

a. Install the AWS SDK for JavaScript - **npm install aws-sdk.**

b. Import the necessary modules and configure AWS SDK

```
1 // Import the necessary modules and configure AWS SDK
2 const AWS = require('aws-sdk');
3
4 // Set the AWS region
5 AWS.config.update({ region: 'us-east-1' });
6
7 // Create an instance of DynamoDB DocumentClient
8 const docClient = new AWS.DynamoDB.DocumentClient();
9
```

c. Retrieve data from DynamoDB

```
//Retrieve data from DynamoDB
const getTrashCanFilledPercentage = () => {
  const params = {
    TableName: 'YOUR_TABLE_NAME',
    Key: {
      id: 'YOUR_ITEM_ID' // Replace with the actual item ID
    }
  };

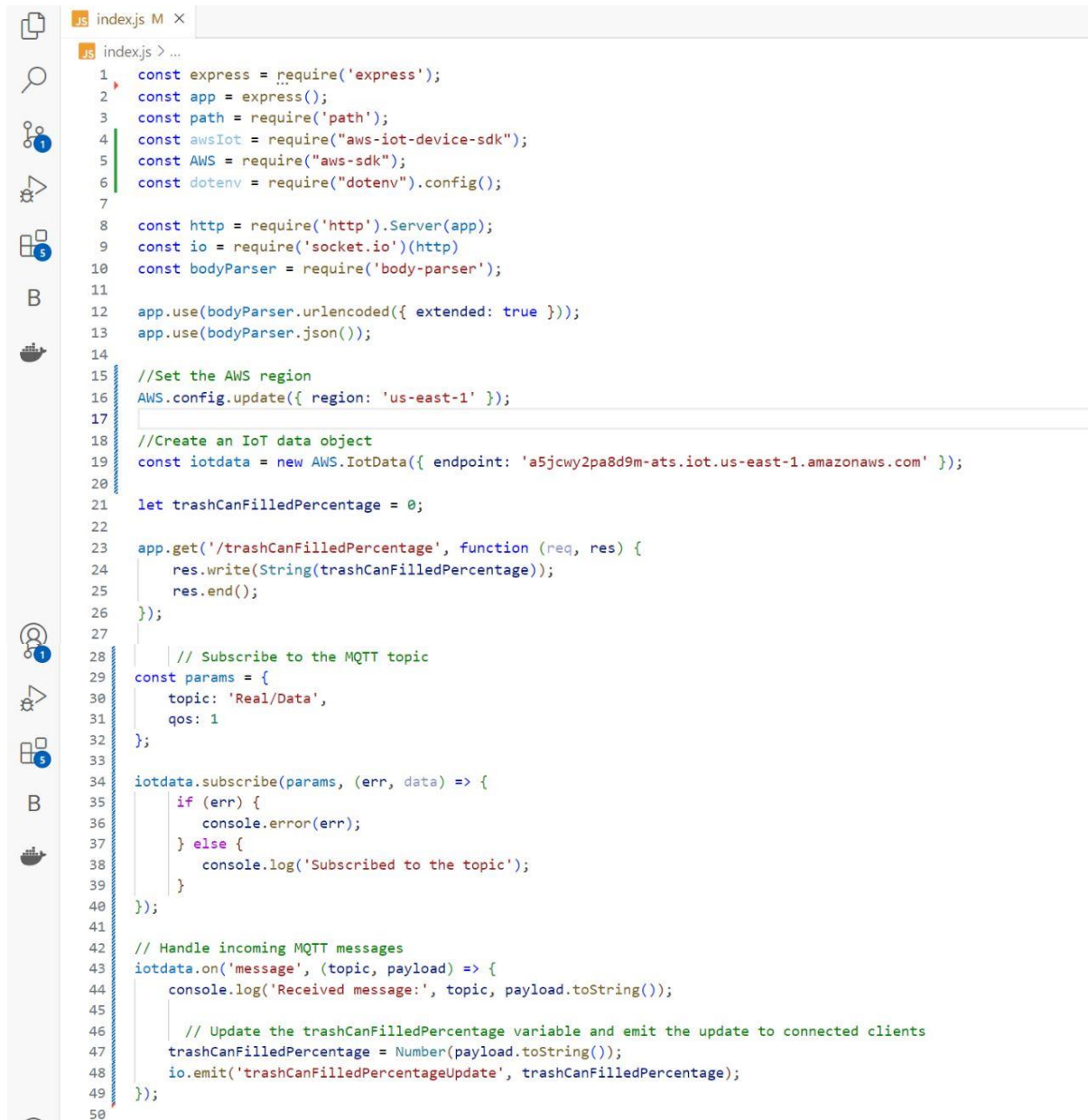
  docClient.get(params, (err, data) => {
    if (err) {
      console.error('Failed to get data from DynamoDB:', err);
    } else {
      if (data.Item && data.Item.filledPercentage) {
        const trashCanFilledPercentage = Number(data.Item.filledPercentage);
        // Use the retrieved value as needed
        console.log('Trash can filled percentage:', trashCanFilledPercentage);
      }
    }
  });
};
```

d. Update data in DynamoDB

```
//Update data in dynamoDB
const updateTrashCanFilledPercentage = (newPercentage) => {
  const params = {
    TableName: 'YOUR_TABLE_NAME',
    Key: {
      id: 'YOUR_ITEM_ID' // Replace with the actual item ID
    },
    UpdateExpression: 'set filledPercentage = :p',
    ExpressionAttributeValues: {
      ':p': newPercentage
    }
  };

  docClient.update(params, (err) => {
    if (err) {
      console.error('Failed to update data in DynamoDB:', err);
    } else {
      console.log('Updated trash can filled percentage:', newPercentage);
    }
  });
};
```

3. **Back-end Server (Express.js):** The code utilizes Express.js, a popular web application framework for Node.js, to create the back-end server. Express handles incoming HTTP requests and defines the server's routes and behavior. It listens for requests on a specified port and responds with appropriate data.



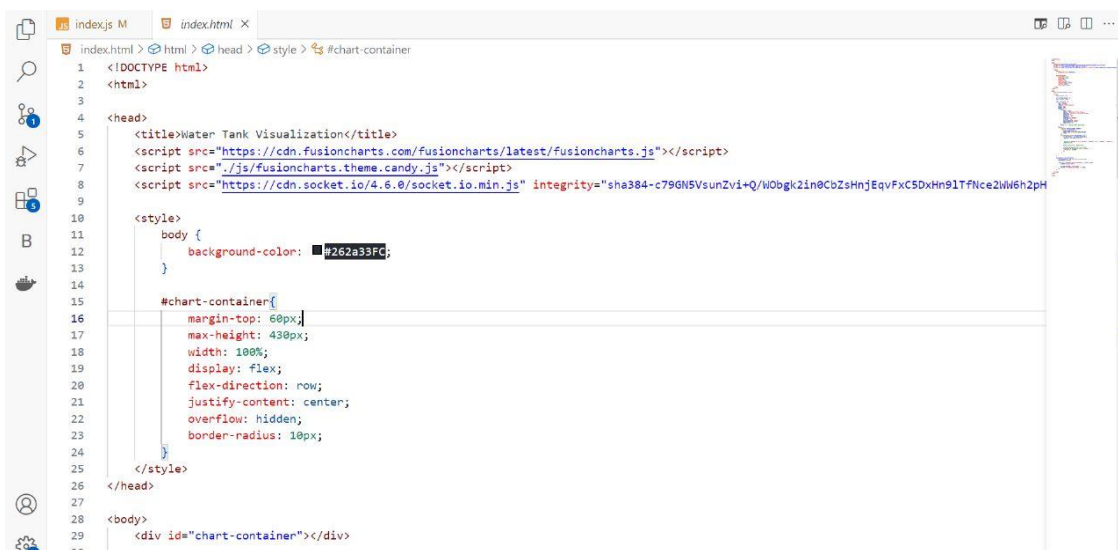
```
1  const express = require('express');
2  const app = express();
3  const path = require('path');
4  const awsIot = require("aws-iot-device-sdk");
5  const AWS = require("aws-sdk");
6  const dotenv = require("dotenv").config();
7
8  const http = require('http').Server(app);
9  const io = require('socket.io')(http);
10 const bodyParser = require('body-parser');
11
12 app.use(bodyParser.urlencoded({ extended: true }));
13 app.use(bodyParser.json());
14
15 //Set the AWS region
16 AWS.config.update({ region: 'us-east-1' });
17
18 //Create an IoT data object
19 const iotdata = new AWS.IotData({ endpoint: 'a5jcw2pa8d9m-ats.iot.us-east-1.amazonaws.com' });
20
21 let trashCanFilledPercentage = 0;
22
23 app.get('/trashCanFilledPercentage', function (req, res) {
24   res.write(String(trashCanFilledPercentage));
25   res.end();
26 });
27
28 // Subscribe to the MQTT topic
29 const params = {
30   topic: 'Real/Data',
31   qos: 1
32 };
33
34 iotdata.subscribe(params, (err, data) => {
35   if (err) {
36     console.error(err);
37   } else {
38     console.log('Subscribed to the topic');
39   }
40 });
41
42 // Handle incoming MQTT messages
43 iotdata.on('message', (topic, payload) => {
44   console.log('Received message:', topic, payload.toString());
45
46   // Update the trashCanFilledPercentage variable and emit the update to connected clients
47   trashCanFilledPercentage = Number(payload.toString());
48   io.emit('trashCanFilledPercentageUpdate', trashCanFilledPercentage);
49 });
50
```

```

51 app.post('/trashCanFilledPercentage', function (req, res) {
52   trashCanFilledPercentage = Number(req.body.value);
53
54   io.emit('trashCanFilledPercentageUpdate', trashCanFilledPercentage);
55
56   //send data to influxDB(Install influxdb to server)
57
58   //make a series using influxDB
59
60   console.log('Updated trashfill percentage: ' + trashCanFilledPercentage);
61
62   res.write('updated the percentage');
63   res.end();
64 });
65
66 app.get('/', function (req, res) {
67   res.sendFile(path.join(__dirname, 'index.html'));
68 });
69
70 app.use('/js', express.static('js'))
71
72 const port = process.env.PORT || 3000;
73
74 const server = http.listen(port, () => {
75   const { port } = server.address();
76   console.log(`Server running on http://3.91.54.203:${port}`);
77 });
78
79
80 io.on('connection', function (socket) {
81   socket.emit('trashCanFilledPercentageUpdate', trashCanFilledPercentage);
82   console.log('Client connected...')
83 });

```

4. **Front-end (HTML, JavaScript, and Socket.IO):** The code serves static files, including an index.html file and client-side JavaScript files, to the front-end application. The front-end files are stored in the js folder and can be accessed via the /js route. Socket.IO is used to enable real-time communication between the server and the client, facilitating updates without requiring page refreshes.




```

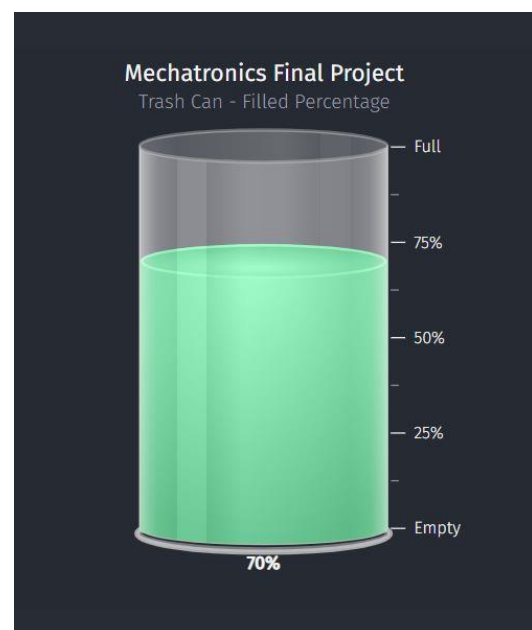
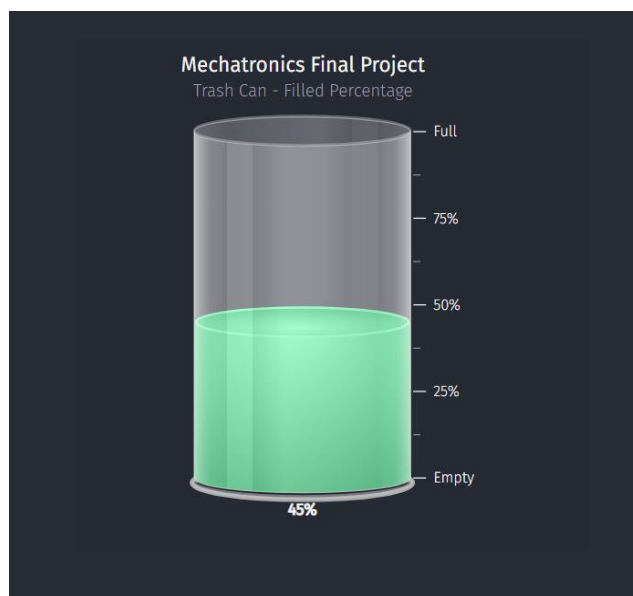
index.js M  index.html x
index.html > html > head > style > #chart-container

28 <body>
29   <div id="chart-container"></div>
30
31   <script>
32     const socket = io();
33
34     let filledPercentage = 0;
35     const chartRef = null;
36
37     const chartConfig = {
38       id: 'trashbin',
39       type: 'cylinder',
40       renderAt: 'chart-container',
41       width: '450',
42       height: '500',
43       dataSource: {
44         chart: {
45           theme: "candy",
46           caption: 'Mechatronics Final Project',
47           subcaption: "Trash Can - Filled Percentage",
48           lowerLimit: '0',
49           upperLimit: '100',
50           showValue: '1',
51           cylFillColor: '#44FF98',
52           cylHeight: '300',
53           lowerLimitDisplay: "Empty",
54           upperLimitDisplay: "Full",
55           numberSuffix: "%",
56         },
57       },
58       events: {
59         rendered: function(evtObj, argObj) {
60           setInterval(function() {
61             evtObj.sender.getData();
62             evtObj.sender.setData(filledPercentage);
63           }, 500);
64         },
65         realTimeUpdateComplete: function(evt, arg) {
66           var annotations = evt.sender.annotations,
67             dataVal = evt.sender.getData(),
68
69           colorVal = (dataVal >= 70) ? "#6caa03" : ((dataVal <= 35) ? "#e44b02" : "#f8bd1b");
70           //Updating value
71
72           console.log("dataVal", annotations)
73
74           //Changing background color as per value
75           annotations && annotations.update('rangeBg', {
76             "fillcolor": colorVal
77           });
78         }
79       },
80     };
81
82   </script>
83
84   FusionCharts.ready(function(){
85     var fusioncharts = new FusionCharts(chartConfig);
86     fusioncharts.render();
87
88     socket.on('trashCanFilledPercentageUpdate', function (data) {
89       if(data == null) return;
90
91       filledPercentage = Number(data);
92       console.log("Updated percentage: " + data);
93     });
94   });
95
96   </script>
97 </body>
98 </html>

```

Here's a step-by-step overview of how the components interact:

- When the server starts, it establishes a connection to DynamoDB using the AWS SDK and creates a DynamoDB DocumentClient.
- It then calls the **getTrashCanFilledPercentage** function to retrieve the initial data from DynamoDB. This data is the **trashCanFilledPercentage**.
- The server sets up various routes using Express.js. For example:
 - The **/trashCanFilledPercentage** route handles GET and POST requests related to the **trashCanFilledPercentage**. GET requests return the current value, while POST requests update the value and store it in DynamoDB.
 - The **/** route serves the **index.html** file, which represents the front-end user interface.
 - The **/** route serves the client-side JavaScript files.
- The server listens for incoming requests on a specified port.
- When a client connects to the server via Socket.IO, the server emits the current **trashCanFilledPercentage** value to the client.
- When a client updates the **trashCanFilledPercentage** value and sends a POST request, the server updates the value and stores it in DynamoDB.
- The server emits the updated **trashCanFilledPercentage** value to all connected clients via Socket. IO.

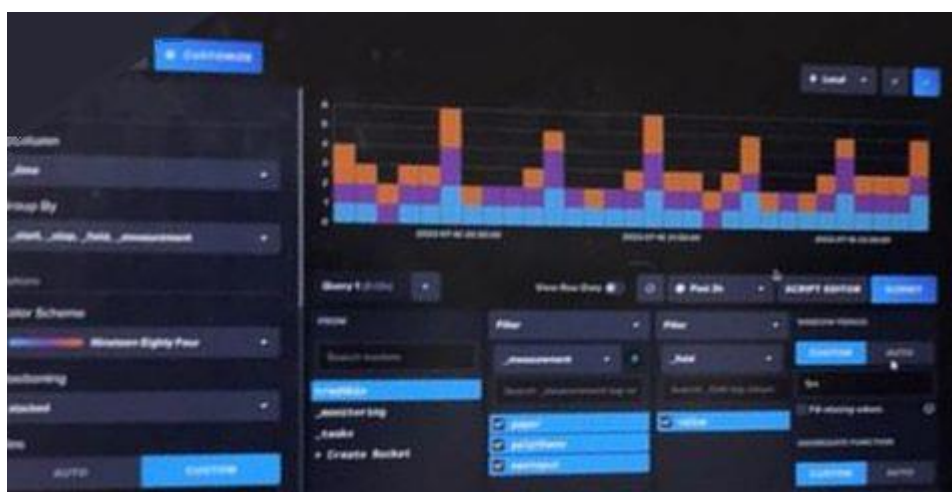
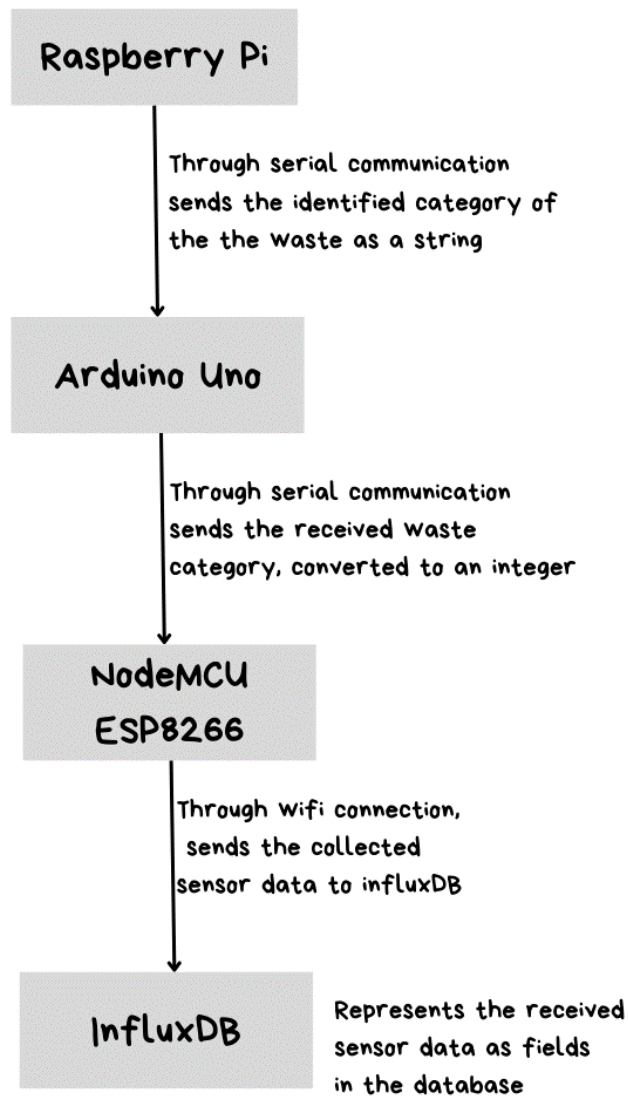


Analysing Waste Generation Patterns

Every time Raspberry Pi classifies the type of the disposed waste, that data is sent to Arduino Uno through serial communication. Since the Arduino Uno cannot directly connect to internet, it is connected to a NodeMCU ESP8266 which has ability to send the data to a web server.

To connect the Arduino Uno and NodeMCU ESP8266, serial communication was used and data from Arduino to NodeMCU was sent using SoftwareSerial protocol. The SoftwareSerial protocol uses two digital pins on the Arduino Uno to create a virtual serial port, allowing the Arduino Uno to communicate with other devices that use the serial communication protocol. Though the data received by Arduino are strings, they had to be converted into numerical values before sending to NodeMCU, since the `parseFloat ()` function we used, is not applicable to string data type.

Then the classification data is sent from NodeMCU to InfluxDB Cloud, a time series database that can be used to store and analyze data. Those data (called sensor data on InfluxDB environment) is used to create graphical representations of the disposal of each waste category over a period. This information can be used to track trends in waste disposal, identify areas where waste disposal could be improved, and make better decisions about waste management.



Code on Arduino Uno

Arduino.ino

```
1  #include <SoftwareSerial.h>
2  SoftwareSerial s(3,2);
3  int cat; // to store the waste catogery as 1,2,3
4
5  void setup(){
6      Serial.begin(9600);
7      s.begin(4800);
8      pinMode(2,OUTPUT);
9      pinMode(3,INPUT);
10 }
11
12 void loop(){
13     String data = "STOP"; // The string you want to send
14     Serial.println(data); // Send the string data through the serial port
15     delay(1000); // Delay between sending each string (optional)
16     // Garbage is positioned below the camera module
17
18     if (Serial.available()>0) { // check if there's incoming data
19         String data = Serial.readString(); // read the incoming data until the terminating character
20         Serial.println(data); // print the received data
21     }
22
23     // converting the types of waste catogery from string to int
24     if (data=="Paper"){
25         cat=1;
26     }
27     if (data=="Wastefood"){
28         cat=2;
29     }
30     if (data=="Polythene"){
31         cat=3;
32     }
33     else {
34         cat=5;
35     }
36
37     s.print(cat);
38     s.println("\n");
39     delay(30);
40
41
42 }
```

Code on NodeMCU ESP8266

sketch_jul18a.ino

```
1  #if defined(ESP32)
2  #include <WiFiMulti.h>
3  WiFiMulti wifiMulti;
4  #define DEVICE "ESP32"
5  #elif defined(ESP8266)
6  #include <ESP8266WiFiMulti.h>
7  ESP8266WiFiMulti wifiMulti;
8  #define DEVICE "ESP8266"
9  #endif
10 #include <SoftwareSerial.h>
11 #include <ESP8266WiFi.h>
12 SoftwareSerial es(D2,D3);
13
14 float val;
15
16 #include <InfluxDbClient.h>
17 #include <InfluxDbCloud.h>
18
19 // WiFi AP SSID
20 #define WIFI_SSID "Mi A3"
21 // WiFi password
22 #define WIFI_PASSWORD "chamindu2000"
23
24 #define INFLUXDB_URL "http://139.177.191.214:8086"
25 #define INFLUXDB_TOKEN "8-fdYV_Iwyh-eKSBWOQd04Zmrds77-PVCMLPNklqYFnaAhdS0jqGncuhs_MwcUIhhPG4rj0qStz1Nymn50ZKWA=="
26 #define INFLUXDB_ORG "134dda162afa3554"
27 #define INFLUXDB_BUCKET "trashbin"
28
29 // Time zone info
30 #define TZ_INFO "UTC5.5"
31
32 // Declare InfluxDB client instance with preconfigured InfluxCloud certificate
33 InfluxDBClient client(INFLUXDB_URL, INFLUXDB_ORG, INFLUXDB_BUCKET, INFLUXDB_TOKEN, InfluxDbCloud2CACert);
34
35 // Declare Data point
36 Point sensor("Cat");
37
38 int polytheneCount = 120;
39 int wastefoodCount = 100;
40 int paperCount = 111;
41 String data;
42 void setup() {
43
44
45
46 //Serial.begin(115200);
47 Serial.begin(9600);
48 es.begin(4800);
49 pinMode(D2,INPUT);
50 pinMode(D3,OUTPUT);
51
52
53 // Setup wifi
54 WiFi.mode(WIFI_STA);
55 wifiMulti.addAP(WIFI_SSID, WIFI_PASSWORD);
56
57 Serial.print("Connecting to wifi");
58 while (wifiMulti.run() != WL_CONNECTED) {
59   Serial.print(".");
60   delay(100);
61 }
62 Serial.println();
63
64 // Accurate time is necessary for certificate validation and writing in batches
65 // We use the NTP servers in your area as provided by: https://www.pool.ntp.org/zone/
66 // Syncing progress and the time will be printed to Serial.
67 timeSync(TZ_INFO, "pool.ntp.org", "time.nis.gov");
```

MANDIS P.L.P.M 210375N

PASANDUL M.W.M 210445F

```

70 // Check server connection
71 if (client.validateConnection()) {
72     Serial.print("Connected to InfluxDB: ");
73     Serial.println(client.getServerUrl());
74 } else {
75     Serial.print("InfluxDB connection failed: ");
76     Serial.println(client.getLastErrorMessage());
77 }
78 // ... code in setup() from Initialize Client
79
80 // Add tags to the data point
81 sensor.addTag("device", DEVICE);
82 }
83 void loop() {
84     while(es.available()>0){
85         val = es.parseFloat();
86         if (es.read()=='\n'){
87             }}
88
89 delay(1000);
90 // Clear fields for reusing the point. Tags will remain the same as set above.
91 sensor.clearFields();
92
93 if (val == 1.00){
94     paperCount = paperCount + 1;
95 }
96 if (val == 2.00){
97     wastefoodCount = wastefoodCount + 1;
98 }
99 if (val ==3.00){
100     polytheneCount = polytheneCount + 1;
101 }
102 // Store measured value into point
103 // Report RSSI of currently connected network
104 sensor.addField("polythene", polytheneCount);
105 sensor.addField("paper", paperCount);
106 sensor.addField("wastefood", wastefoodCount);
107 //sensor.addField("rssi", WiFi.RSSI());
108
109 // Print what are we exactly writing
110 Serial.print("Writing: ");
111 Serial.println(sensor.toLineProtocol());
112
113 // Check WiFi connection and reconnect if needed
114 if (wifiMulti.run() != WL_CONNECTED) {
115     Serial.println("wifi connection lost");
116 }
117 Serial.println("Waiting 1 second");
118 delay(1000);
119 }
120 }
121

```

Conclusion

In conclusion, the smart waste management mechatronic project has successfully demonstrated the integration of various technologies to address the challenges of waste management. By utilizing image classification technology, the system achieved efficient and accurate sorting of waste items, including food, polythene, and paper. This sorting capability significantly contributes to improving waste segregation and recycling efforts.

The project's implementation of a smart compressing and sealing system further enhances waste management efficiency. By reducing the volume of waste through effective compression, the system optimizes storage space. The sealing mechanism ensures the containment of compressed waste, mitigating odour and leakage issues. The real-time monitoring system implemented in the project offers valuable insights into waste management processes.

However, it is important to note that continuous research and development efforts are necessary to further enhance the system's accuracy, reliability, and scalability. Future iterations of the project can focus on incorporating additional waste sorting capabilities, expanding the range of waste items that can be processed, and integrating machine learning algorithms to improve classification accuracy.