Classification and Collection of Drinking Waste on a Conveyor Belt with UR3e and Computer Vision

Sosa Guzmán Mariana
Facultad de Ingeniería Eléctrica y Electrónica (FIEE)
Universidad Veracruzana
Veracruz, Mexico
zs19003110@estudiantes.uv.mx

***Abstract*—Collection and classification of waste for recycling is needed to decrease the amount of garbage in dumpsters and to avoid it from going to water bodies, also to save resources when manufacturing new products. This paper proposes a system for collecting cans from a conveyor band using computer vision and the UR3e robot. The aproach is based on the identification of drinking waste and location using a camera. For this first phase, the collection process is only implemented for cans.**

***Index Terms*—computer vision, UR3e, image classification, machine learning**

I. Introduction

Nowadays we face a huge environmental problem caused by the massive amount of garbage, that we as humans generate. Every year 2.12 billion tons of waste are thrown [1], unfortunately many objects end up in the oceans and cities. In 2004, the aluminium beverage can waste passed the 1 trillion units [2], and the time of desintegration for each is 10 years [3]. Therefore, it is imperative to implement technological systems in order to reduce and recycle the existing garbage. Aluminium cans can be recycle and used to create new products. The designed systems have to identify and be able to differentiate a can from other types of waste. For this reason, our system needs to locate the can for the robot to pick it. In this work, a system for classification and collection of cans is presented and used in collaboration with an UR3e robot.

II. State of the Art

The problem mentioned above has motivated researchers and engineers to develop garbage collector systems such as the one presented by Bofeng, et Al. in [4] with multiple waste collector robots working in parallel using the Multiple Travel Salesman Problem and a Genetic Algorithm to pick and place waste from a conveyor to a container; in [5], Kiyokawa et Al. propose a robotic waste sorter with agile manipulaton and a quickly trainable detector to replace the human role of robust detection and manipulation of waste, they provided three methodologies, one using a Deep Neural Network model (DNN) to identify the targets from images. Panyavaraporn et Al. designed a dual mode controlled water surface garbage collecting robot using embedded Deep Learning to identify waste in bodies of water and collect it [6].

III. The System

To tackle the problem, a small-scale collection system was designed with the intention of later being improved and adjusted for real life cases on a large scale, capable of collecting not only cans but also other waste. Despite being able to identify 4 types of drinking waste, the localization of the element was only implemented for aluminum cans.

A. System Description

The squematic of the system is presented in Figure 1, containing the UR3e robot, the location of the conveyor band and the camera.

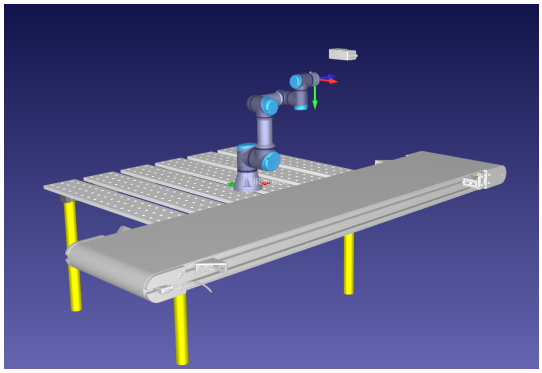

Figure 1. System Squematic

B. Block Diagram of the System

Figure 2 shows the block diagram of the computer vision system and how the X cartesian matrix can be obtained to get the q vector of joint positions for the robot by Inverse Kinematics, all of this just with one image.

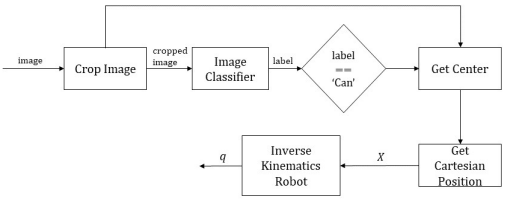

Figure 2. Block diagram

IV. Cropping the Input Image

Since the camera is at a height of 63cm, the images captured by it will have objects that might confuse the NN model; for this reason, the image captured by the camera must be cropped. For this first phase of the project, all images will be shoothed when the object is at a certain position in the conveyor, which will be equivalent to the square marked by the pixels [1040: 1760, 1330: 1750]. Figure 3 shows the result of this process.

Figure 3. Image Before and After Cropping

V. Image Classification

A Neural Network (NN) was built and trained using Transfer Learning [7], specifically using the pre-trained NN *'MobileNetV2'* [8] to classify and identify cans. The *drinking-waste-classification* dataset from Kaggle [9] was used to training the network.

This dataset contains 4828 images of four different categories of drinking waste: Cans (1060 images), Glass (1232 images), HDPEM (1028 images), PET (1508 images); separated in four folders, each one corresponding to a category. Figure 4 shows the structure of the dataset.

```
rawimgs
|__ AluCan
    |_____ AluCan1,000.jpg
    |_____ AluCan1,001.jpg
    |_____ AluCan1,002.jpg
    ....
|__ Glass
    |_____ Glass1,000.jpg
    |_____ Glass1,001.jpg
    |_____ Glass1,002.jpg
    ....
|__ HDPEM
    |_____ HDPEM1,000.jpg
    |_____ HDPEM1,001.jpg
    |_____ HDPEM1,002.jpg
    ....
|__ PET
    |_____ PET1,000.jpg
    |_____ PET1,001.jpg
    |_____ PET1,002.jpg
    ....
```

Figure 4. Dataset structure

All these images are equivalent to 2GB of data, therefore, to avoid using local computational resources, the project was developed using Google Colab Notebooks [10].

A. Data Preparation

The first task was Data Preparation, since this dataset is only divided by category it was necessary to restructure it to have three main folders, *training, validation, testing* sets, each one containing subfolders for each category as in Figure 5.

```
output/
|__train/
|___ AluCan/
    |_____ alucan1.jpg
    |_____ alucan2.jpg
    ...
|___ Glass/
    |_____ glass1.jpg
    ...
|___ ...
|__val/
|___ AluCan/
    |_____ alucan8.jpg
    |_____ alucan10.jpg
    ...
|___ Glass/
    |_____ glass11.jpg
    ...
|___ ...
|__test/
|___ AluCan/
    |_____ alucan32.jpg
    |_____ alucan98.jpg
    ...
|___ Glass/
    |_____ glass78.jpg
    ...
|___ ...
```

Figure 5. Splited dataset structure

To solve this problem, the *split-folders* [11] Python library was used. With this library, a dataset splited by class name can be restructured to a dataset with train, validation and test sets and the percentage of data in each set can be specified by the user. In this particular case, the percentages were: *training: 80%, validation: 10%, testing: 10%*.

After the split, the number of images in each set is presented in Figure 6.

```
Number of AluCan images in the dataset: 1060
• 848 training set
• 106 validation set
• 106 testing set

Number of Glass images in the dataset: 1232
• 985 training set
• 123 validation set
• 124 testing set

Number of HDPEM images in the dataset: 1028
• 822 training set
• 102 validation set
• 104 testing set

Number of PET images in the dataset: 1508
• 1,206 training set
• 150 validation set
• 152 testing set
```

Figure 6. Number of images in each set after split

B. Pipeline: Image Transformations and Labels

When training a Neural Network it is useful to create randomness in the training data, so the network can classify correctly an image even if it's rotated, fliped, scaled, etc. This transformations can be done with the *ImageDataGenerator* class [12].

By using the *ImageDataGenerator* the training images were rotated, fliped vertically and horizontally, zoomed, etc, Figure7. Additionally, the class helped by rescaling each image and resizing to (224, 224) since it's the size needed by the *MobileNetV2*.

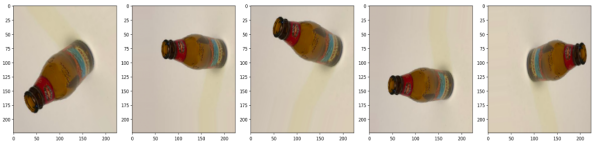

Figure 7. One image of the training set after transformations

For the validation and testing sets, it is not necessary to apply all the augmentations that were applied for the training set, but a rescale is necessary to pass the images to the NN model.

One of the advantages of using the *ImageDataGenerator* is that it automatically labels the images according to the folder names corresponding to the class names with the method `image_gen.flow_from_directory()`, in this method, if the parameter `class_mode='binary'` means that it will consider the four classes as an array of dimensions (1, n), where n is the number of classes in the dataset.

C. Neural Network Model

As previously mentioned, the *MobileNetV2* pre-trained network was used for this approach. The pre-trained model is responsible for extracting the features of the images, therefore this part of our model is called the `feature_extractor`. The weights and biases of the pre-trained model were freezed to avoid modifying them during training.

The model was built by creating a `tf.keras Sequential` model with the `feature_extractor` and a new classification layer. Since the dataset has 4 classes the output layer has 4 units, Figure 8.

```
my_model = tf.keras.Sequential([
    feature_extractor,
    tf.keras.layers.Dense(4, activation='softmax')
])
```

```
my_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 keras_layer (KerasLayer)    (None, 1280)              2257984

 dense (Dense)               (None, 4)                 5124

=================================================================
Total params: 2,263,108
Trainable params: 5,124
Non-trainable params: 2,257,984
_________________________________________________________________
```

Figure 8. Model structure and summary

For this NN a softmax activation function was selected since the problem is a multi-class classificacion problem, and the softmax function is used to transform results to continuous probabilities that will add up to 1.

Figure 9 shows the training process, first the model was compiled using the *adam* optimizer, the *sparse categorical crossentropy* loss and accuracy as metric. Then an early stopping technique was applied to avoid overfitting and extra use of computing resources. After finding the best model, the model was saved as a Keras model for later use.

```
# Compile the model
my_model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

```
# Early Stopping
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                                  patience=7)
# Save the best model
save_best_model = tf.keras.callbacks.ModelCheckpoint('./best_model.h5',
                                                     monitor='val_loss',
                                                     save_best_only=True)
```

```
# Train model
EPOCHS = 15
history = my_model.fit(train_data_gen,
                       epochs=EPOCHS,
                       validation_data=val_data_gen)
```

Figure 9. Model training

After 15 epochs, the model got the results shown in TableI.

Epochs	Loss	Accuracy	Val-Loss	Val-Accuracy
15	0.1597	0.9476	0.1631	0.9332

Table I
RESULTS AFTER TRAINING

Figure 10 shows the loss and accuracy values achieved during training for the training and validation set.

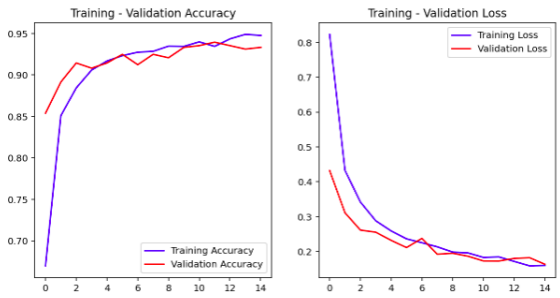

Figure 10. Training-Validation Accuracy & Training-Validation Loss

D. Model Test

After getting the results presented before, the model was evaluated using the testing set. Figure 11 shows the resulting accuracy and loss.

```
loss, accuracy = my_model.evaluate(test_data_gen)
print('Loss and Accuracy on Testing Data')
print('Loss: {}'.format(loss))
print('Accuracy: {}'.format(accuracy))
```

```
8/8 [==============================] - 18s 2s/step - loss: 0.1477 - accuracy: 0.9505
Loss and Accuracy on Testing Data
Loss: 0.14770400524139404
Accuracy: 0.9505154490470886
```

Figure 11. Loss and Accuracy of Testing set

Finally, the model was tested using some images from the internet and some pictures taken with a camera, Figure 12.

Figure 12. Predictions

VI. Image Location

After classifying images, and identifying cans, which in the scope of this research, is the goal; it was imperative to find the location of the objective with the camera and transform it to a cartesian position for the robot. For this task, we took a picture from the established height (63cm) and measure with a flexometer the length of the perimeter captured by the camera as shown in Figure 13. The obtained distances were 56cm of width and 65cm of height.

Figure 13. Measure of the frame

After getting the lengths, two approches were tested to get the center of the object to later obtain its distance from the robot.

1) Getting the borders of the can and then drawing a line that goes from the top-left corner to the bottom-right corner, and finally getting the center of that line which will correspond to the center of the can.
2) Getting the center of the cropped image as a global center for the robot's gripper.

For simplicity, the second approach was the one used in the final implementation. But both approaches will be described in this paper.

A. First Approach

An *autocanny* [citaCannyFilter] function was defined to get the borders of the cans, to use this function we can specify a *sigma* value to that works as a noise canceling parameter in the image, if not specified, this parameter takes a default value of 0.33, Figure 14.

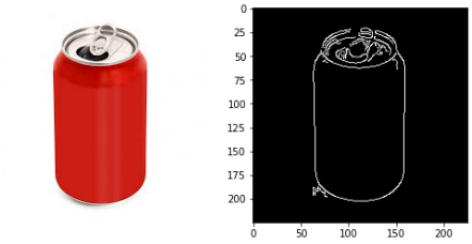

Figure 14. Image borders after autocanny

Then, two functions were defined. One to get a rectangle of the largest contour found on the image, and the other one to get the line that goes from the top-left corner to the bottom-right corner of the largest contour, the results are shown in Figure 15. Both functions use the perimeter of the contours to identify the largest one.

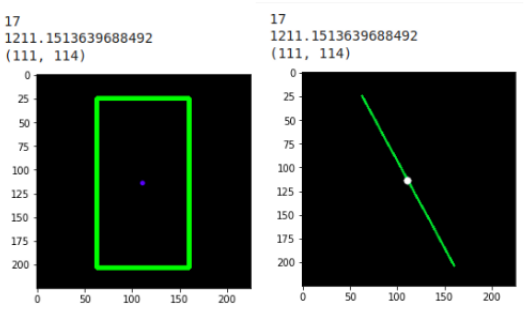

Figure 15. Image after drawing rectangle and line

B. Second Approach

This approach is simpler than the previous one; to get the center we only need to get the centroid of the rectangle gotten after the cropping, using the pixel coordenates used to crop the image.

After getting the center of the object, to get the cartesian coordinates from the pixel coordinates, we pass the center pixels to a function called `pixel_to_cartesian` that uses a proportionality rule, the pixel dimensions of the original image (before cropping) and the lengths mentioned at the beginning of this section, to get the cartesian values.

VII. Results

Results are shown in Figure 16. The model detected the can with a 73.54% of certainty, and the location found was: (28.52cm, 22.57cm, 63cm) in axis (x, y, z) respectively.

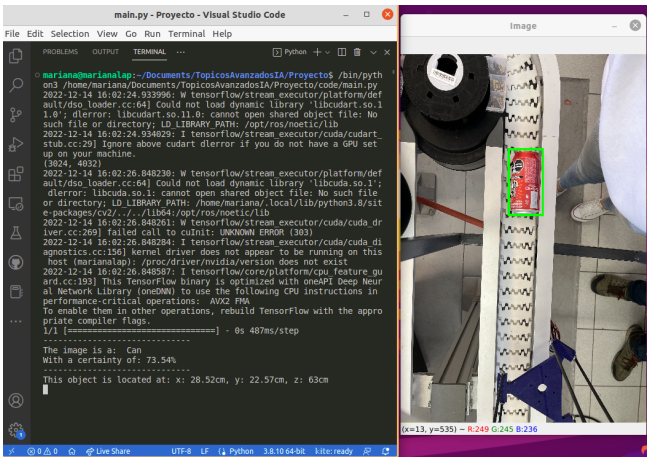

Figure 16. Results

VIII. Conclusions and Future Work

In this research, to achive the goal, first we cropped the image took by the camera and then we trained a NN model to classify 4 different kinds of drinking waste, this model got a 94.76% of accuracy, then we only focused on cans to identify their cartesian location for the robot to pick it, for the localization process two methods were proposed, the first one involved getting the contours of the image and get the centroid of the object, the second was simplier because the process only required to get the centroid of the cropped image. After joining the processes we got the results shown in Figure 16.

After finishing the first phase of this project, the next step is to identify the position and rotation of the object, for example, if the can is standing, upside down, laying down, and the rotation respect a reference frame so we can plan a grasping rutine for the robot; for this task, the dataset is currently being generated and prepared to train a NN for the possible outputs. Another point to develop is obtaining the Inverse Kinematics of the robot using a NN. grasping

Last but not least, as a future task after completing the ones mentioned above, we want to be able to obtain the position of the objects when the conveyor belt is in continuous motion and not just for a specific point.

Acknowledgment

Thanks to MIA Julio Antonio Caballero Mora for helping with the revision of sections I, III and part of section V.

References

[1] The World Counts. How many tons of waste is produced each year? https://www.theworldcounts.com/challenges/planet-earth/state-of-the-planet/world-waste-facts.

[2] J. Gitlitz and P. Franklin. Aluminum beverage can waste passes the one trillion mark recycling rate drops to lowest point in 25 years, 2004. shorturl.at/frzJW.

[3] Ayuntamiento de Alpedrete. 1 lata de aluminio tarda 10 años en degradarse, 2018. shorturl.at/fHJNU.

[4] Bofeng Qi, Lishen Pu, Chunquan Xu, and Aiqun Zheng. Multi-robot task assignment method in the construction waste sorting system. *International Conference on Mechatronics and Automation*, pages 1364–1369, 2022.

[5] Takuya Kiyokawa, Hiroki Katayama, Yuya Tatsuta, Jun Takamatsu, and Tsukasa Ogasawara. Robotic waste sorter with agile manipulation and quickly trainable detector. *IEEE Access*, pages 124616–124631, 2021.

[6] Jantana Panyavaraporn, Natthawat Chaimongkol, Nattasit Limsomnuek, Wichai Wasayangkul, Nathamon Charoenwattana, and Paramate Horkaew. Dual mode controlled water surface garbage collecting robot by using embedded deep learning. *2021 18th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, pages 14–17, 2021.

[7] Nerd For Tech. Transfer learning, 2021. https://medium.com/nerd-for-tech/transfer-learning-7914c6ab2b56.

[8] Keras. Mobilenetv2. https://keras.io/api/applications/mobilenet/.

[9] Arkadiy Serezhkin. Drinking waste classification, 2019. https://www.kaggle.com/datasets/arkadiyhacks/drinking-waste-classification.

[10] Google. Google colaboratory. https://colab.research.google.com/.

[11] Johannes Filter. split-folders. https://pypi.org/project/split-folders/.

[12] TensorFlow. Imagedatagenerator. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator.