ECE5242 - INTELLIGENT AUTONOMOUS SYSTEMS

PROJECT 1 - COLOR SEGMENTATION

FEBRUARY 10, 2020

EVA ESTEBAN
CORNELL TECH

ABSTRACT

The aim of this report is to provide an in-depth analysis of the different machine learning models trained to detect red barrels within an image and provide their location coordinates. This document describes the approaches implemented, the results obtained and a discussion on the performance of every model.

TABLE OF CONTENTS

1 INTRODUCTION

Nowadays, a common way to identify objects within a coloured image is to use color image segmentation. Color segmentation can be implemented by creating a model of the color in question using training images, developing an algorithm to learn this model and using it to segment the color in every new image from the test data. To provide additional information about the object's location, calculations of the object's centroid coordinates and the object's distance to the observer can be implemented using a camera model such as the pin-hole model.

2 PROBLEM STATEMENT

The problem tackled in this project is the detection of all the red barrels within a given image with the highest possible accuracy. The system must identify every red barrel in each new image, and determine its location relative to the world and to the observer i.e the camera. For this, a model was trained with pre-existing data using different hand-labelling and machine learning techniques, and tested with new unknown images to determine its accuracy.

3 APPROACHES

3.1 ITERATION 1: 2-CLASS MODEL

3.1.1 PIXEL CLASSIFICATION

The first step of the project consisted of labelling each of the 50 images. For this, the data was hand-labelled by drawing one or several Regions of Interest (ROIs) around every red barrel in every image using the library Roipoly [3] in the pre-processing script **Hand_Labelling.py**, which can be found in the project zip folder. The corresponding boolean masks, which are 'True' for every red barrel pixel and 'False' for every non barrel pixel, were saved.

The next step consisted of training the data and saving the trained models. In **epe26_project1_code.ipynb**, each image was read using the OpenCV library [1], and converted to YCRCB color space, where CB and CR are the chroma components representing blue-difference and red-difference, and Y is the luma component representing luminance [8]. Even though several other color spaces, such as Red, Green, Blue (RGB), Hue, Saturation, Lightness (HSL), and Hue, Saturation, Value(HSV) were considered, the YCRCB color space was chosen to account for the large variance in luminance accross images and be able to modify it, as well as to identify the strength of the red component of every pixel. The boolean masks were then applied to their corresponding images to separate the barrel pixels from the non barrel pixels, creating two classes for the first simple classifier: barrel and non-barrel.

The next step consisted of calculating the mean and covariance of each class, fitting a multivariate Gaussian function [4]. The equation for a multivariate Gaussian PDF can be found on Eq. 1, where D is the number of dimensions, x is the input vector or pixel, μ represents the mean, and Σ represents the covariance. The functionality for fitting a multivariate Gaussian function to the data can be found in the function **posterior_prob_gaussian()**.

$$f(x;\mu,\Sigma) = \frac{1}{\sqrt{(2\pi)^D|\Sigma|}} e^{-\frac{1}{2}*((x-\mu)^T*\Sigma^-1*(x-\mu)} \tag{1}$$

Furthermore, the prior probability of each class was calculated in the function **prior_prob()** following Eq. 2.

$$P(\text{class}) = \frac{\#\text{ items in class}}{\text{total }\#\text{ items}} \tag{2}$$

The results of the mean, covariance and prior probability calculations for each class were saved into the folder **Trained_Models** using the function **save_model()**.

To classify an image, first the image is read and converted to YCRCB using OpenCV [1]. The trained model data, which consists of the parameters describing each Gaussian distribution, is then loaded using **load_model()**. Taking the Gaussian distribution for one of the classes, for each pixel, this Probability Density Function (PDF) provides a value indicating the probability of that particular class given the pixel, which constitutes the likelihood or conditional probability. Finally, the marginal probability can be calculated as the expected value of the class, which is given by the sum of the prior probability of each class by the conditional probability of the pixel given that class. Bayes Theorem is then applied using Eq. 3 to obtain the probability that a pixel belongs to a particular class e.g the probability of pixel (180, 81, 27) belonging to the red barrel class. This is done in function **bayes_raw()**.

$$P(x|\text{class}) = \frac{P(\text{class}|x)P(x)}{P(\text{class})} \tag{3}$$

A threshold is established to determine whether the probability is high enough for the pixel to belong to the class. In this iteration of the project, the threshold was set to 0.85, so any pixel with a probability greater than or equal to 0.85 of being a red barrel pixel is considered a red barrel pixel.

3.1.2 Connected Components

Once the pixels have been labelled appropriately, in order to group the pixels belonging to the same connected component, the **measure** module from the library **scikit-image** [7] is used. First, a different label is extracted for each connected component in the image using the **label()** function. Following this, the pixels with labels corresponding to the same region are grouped into a **RegionProperties** object using the module **measure.regionprops**. The regions in the barrel class constituted by more than 1,000 pixels are considered a barrel, and the rest discarded. The bounding box coordinates (min row, min col, max row, max col) are then extracted for each barrel. Using these coordinates and the **line()** function from OpenCV [1], four straight white lines are drawn on the image, changing the pixel data to show a bounding box around each detected barrel. The resulting image is then displayed on screen. The functions in the code containing this functionality are **find_barrels()**, **find_centroid()**, **create_bbox** and **locate_barrels()**.

3.1.3 Localization

The localization of the barrels detected in the images consists of finding their Centroid X and Centroid Y coordinates, as well as their distance from the camera. To calculate the centroid of each barrel, the property **bbox** from the **measure.regionprops** module is used for every region identified as a barrel in the image. The centroid data is then printed on screen.

In addition, to calculate the distance from the barrel to the camera, a pin-hole camera model is applied. This model follows the equation presented on Eq. 4, where f is the focal length of the camera, z is the distance from the object to the camera, $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ are the coordinates of a point in the object, and $\begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$ are the coordinates of that same point projected on the image plane. The functions in the code that carry out this functionality are **find_barrels()**, **locate_barrels()**, **find_centroid()**, **find_barrel_image_height()**, and **get_z()**.

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = -\frac{f}{z}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \tag{4}$$

Taking the height of the real barrel as y, and the height of the barrel in the image plane as y_I, we know that $y_I = \frac{-f}{z}y$, giving Eq. 5. We know that $y * f$ is always constant and equal across images, since the

same barrel and camera are being used. Hence $y_I * z$ must remain constant and equal to $y * f$.

$$y * f = y_I * z \tag{5}$$

The provided values for z were extracted from the filenames for each image, and used together with 4 to extract a height coefficient $y * f$ for each image. These coefficients were then averaged to get a single height coefficient. For every test image, the barrel image height y_I is measured and multiplied by the height coefficient to obtain z. The value for z was found to be more accurate when rounded for the training images, so the functionality to round it is included in the algorithm. The value of z represents the distance from the barrel to the camera in meters [m].

3.2 Iteration 2: 11-Class Model and Region Grouping

To reduce the number of false positive results output by the classifier, additional classes were created for the red objects in the images that were being wrongly classified as red barrels. The procedure to label the regions and train the model was analogous to the one described in section 3.1. The classes are: red chair, red cone, red floor, red ball, red exit sign, red fire extinguisher, red bike, red baxter, and red coke machine.

Additionally, the region detection part of the algorithm was improved in this second iteration of the project. A new function **_group_pairs()** was included to group regions within a certain radius. This function was modified from the open source **skimage.measure.regionprops** examples page in [6]. Additional functionality was incorporated in the **locate_barrels()** function to find the centroid of the largest combined region formed by the pairs detected by **_group_pairs()**.

The distance calculation of z procedure stayed the same as in Iteration 1 of the algorithm.

3.3 Iteration 3: Post-Submission

After the official submission of the code, changes were implemented to improve the accuracy of the classifier. To begin with, the threshold on the minimum number of pixels filled for a detected object to be considered a barrel was decreased to 50 in order to detect the barrels that were farthest away, which previously were not detected. In addition, the prior probabilities of the classes, such as the baxter class, were modified to improve performance, and the threshold distance used to determine if several regions are part of the same object was tuned appropriately. Additional techniques such as checking the ratio between the major axis and the minor axis of the object's detected were implemented and tested. The final results and techniques selected for this iteration can be found in section 4 of this document.

4 Testing and Results

4.1 Iteration 1 Outcome

The model was tested both at the pixel level and at the global level. At the pixel level, the training images were split into a training and a testing subset by using the **train_test_split()** function from the library **sklearn** [2]. For each pixel in the testing subset, a 'True' value was assigned if it belonged to the red barrel class, and a 'False' value was assigned otherwise. The probability threshold used for Bayes was 0.85. The accuracy obtained when comparing to the labels was 99%.

At the global level, when grouping regions of pixels belonging to the same class, the accuracy decreased. One of the reasons for this were the constraints on the minimum number of pixels needed for a region to be considered a barrel, since barrels located far away could have the same number of pixels as a ball located close to the camera. Another reason was the inadequate grouping of regions belonging to

the same barrel, if there was an obstacle between the barrel and the camera e.g a metal bar in front of the barrel cutting its picture by half. A second iteration was performed on the algorithm to solve this.

4.2 ITERATION 2 OUTCOME

When tested using **train_test_split()** on the training data, this approach reduced the number of false positives but did not eliminate them completely. While the EXIT sign regions were no longer selected by the classifier, the Baxter robot regions were still resulting in false positives. This is consistent with the fact that the training dataset includes 13 images with an EXIT sign, while only including 1 image with a Baxter robot in it. Since the only data available for testing prior to the release of the official testing data were different subsets of the training data, if the Baxter image was used for training, it could not be used for testing and vice versa, making the elimination of its false positives an especially challenging part of the project.

In regards to connecting regions from the same barrel, the classifier was able to connect the regions from the barrel in Image 2.6 of the training set when using a distance threshold of 20 pixels between neighbors. However, this threshold was not high enough for the algorithm to connect the regions in Image 3.11 of the training data. Increasing the threshold above 20 resulted in a correct classification of Image 3.11 but led to the barrel regions connecting with regions from false positives in Image 2.6 and others, so it was determined that the false positive problem had to be addressed before making more improvements regarding region grouping. This challenge was addressed in Iteration 3.

The classifier from Iteration 2 was trained on the full set of training images and tested on the official ten tested images once they were released. The results of the classification can be found below:

Image 001

- Centroid X = 806, Centroid Y = 229, Distance = 7.
- Centroid X = 918, Centroid Y = 918, Distance = 6.
- Centroid X = 1110.5, Centroid Y = 278, Distance = 9.
- Centroid X = 713, Centroid Y = 321, Distance = 12.
- Centroid X = 703, Centroid Y = 329, Distance = 11.
- Centroid X = 660, Centroid Y = 503, Distance = 2.

Image 002

- Centroid X = 617, Centroid Y = 376.5, Distance = 3.
- Centroid X = 619, Centroid Y = 498, Distance = 4.

Image 003

- Centroid X = 548, Centroid Y = 385.5, Distance = 4.
- Centroid X = 694.5, Centroid Y = 450, Distance = 3.

Image 004

- Centroid X = 631, Centroid Y = 437.5, Distance = 2.
- Centroid X = 712.5, Centroid Y = 575, Distance = 7.

Image 005

- Centroid X = 643.13, Centroid Y = 468.93, Distance = 6.

Image 006

- Centroid X = 633.17, Centroid Y = 425.92, Distance = 10.

Image 007

No barrel deteted

Image 008

No barrel deteted

Image 009

No barrel deteted

Image 010

- Centroid X = 664.06, Centroid Y = 395.13, Distance = 7.

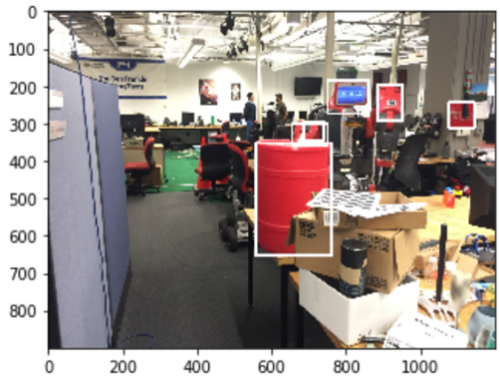

Image 001 result.

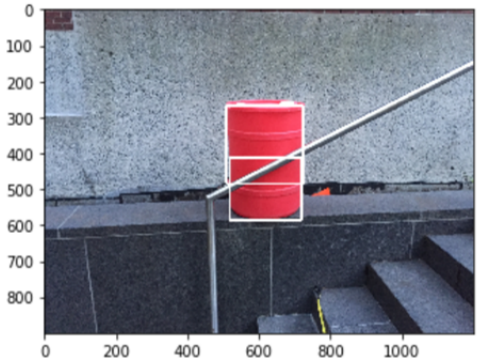

Image 002 result.

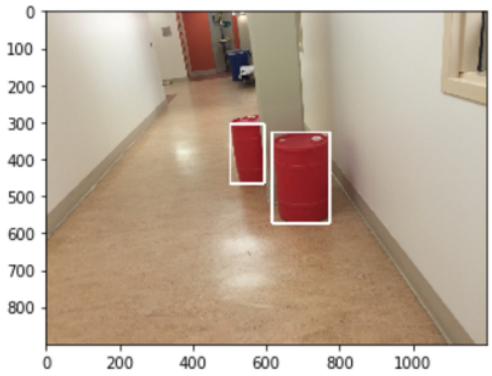

Image 003 result.

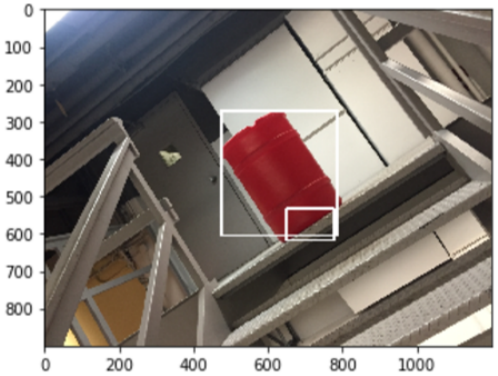

Image 004 result.

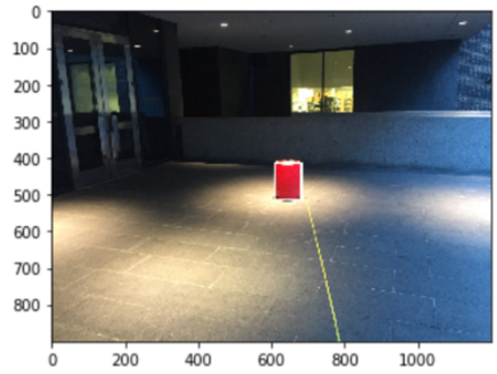

Image 005 result.

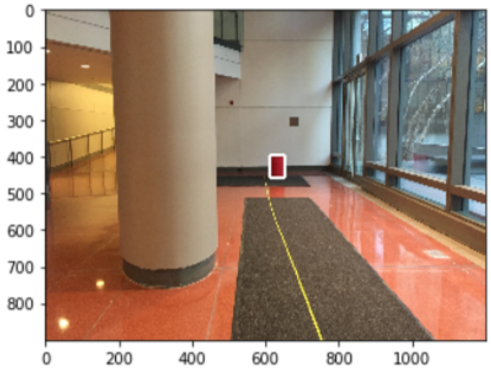

Image 006 result.

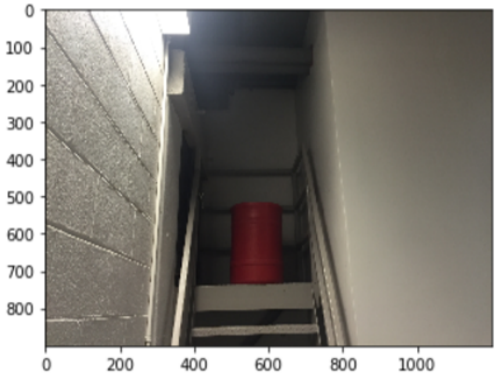

Image 007 result.

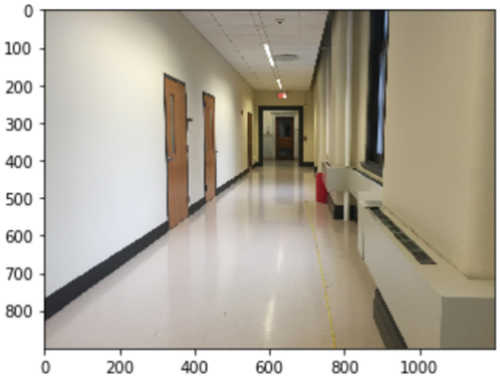

Image 008 result.

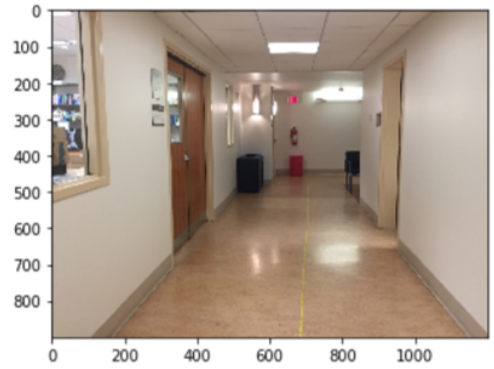

Image 009 result.

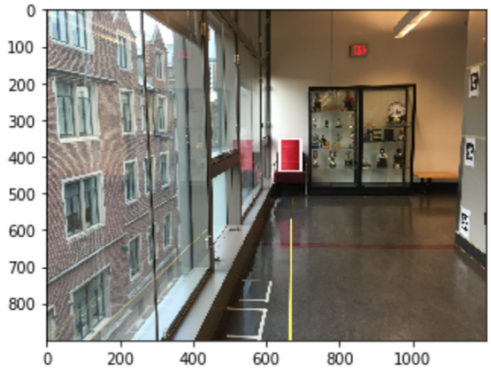

Image 010 result.

4.3 ITERATION 3 OUTCOME

Lowering the threshold on the minimum number of filled pixels needed for a region to be considered a barrel from 1,000 to 150 resulted in the algorithm correctly identifying the barrels in Image 008 and Image 009 from the testing set, which previously were not identified.

In addition to this, functionality for the algorithm to detect the two largest barrel regions in every image was incorporated in the function **find_barrels()**. This dramatically reduced the number of false positives in every image, especially in Image 001 of the test set, and, in turn, allowed to increase the threshold for the maximum distance between two regions to be joined into one to 80. This resulted in the barrel of Image 002 being classified correctly as a single region.

Moreover, by lowering the threshold for a pixel to be considered a barrel pixel from 0.85 to 0.75 allowed the algorithm to detect a small region of pixels on Image 007, whereas previously it was not detecting anything. The next step is to identify the full barrel.The final classification of the testing images can be found below:

Image 001

- Centroid X = 660, Centroid Y = 478, Distance = 2.
- Centroid X = 111.50, Centroid Y = 278, Distance = 9.

Image 002

- Centroid X = 617, Centroid Y = 426, Distance = 2.

Image 003

- Centroid X = 696, Centroid Y = 451.5, Distance = 2.
- Centroid X = 550, Centroid Y = 376.5, Distance = 3.

Image 004

- Centroid X = 635.5, Centroid Y = 445.5, Distance = 2.

Image 005

- Centroid X = 645.5, Centroid Y = 464.5, Distance = 6.

Image 006

- Centroid X = 632.5, Centroid Y = 472.5, Distance = 10.

Image 007

- Centroid X = 569, Centroid Y = 697.5, Distance = 29.

Image 008

- Centroid X = 741, Centroid Y = 473.5, Distance = 8.

Image 009

- Centroid X = 674.5, Centroid Y = 427.5, Distance = 11.

Image 010

- Centroid X = 665, Centroid Y = 396.5, Distance = 7.

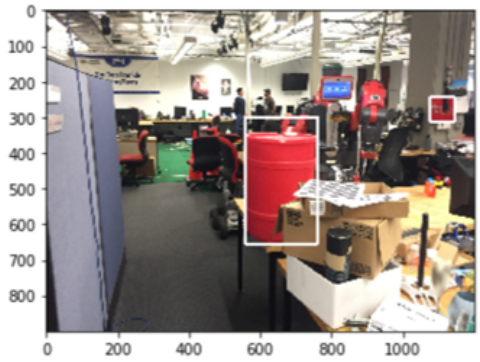

Image 001 result after algorithm improvement.

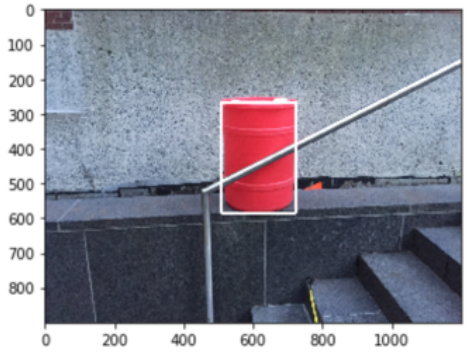

Image 002 result after algorithm improvement.

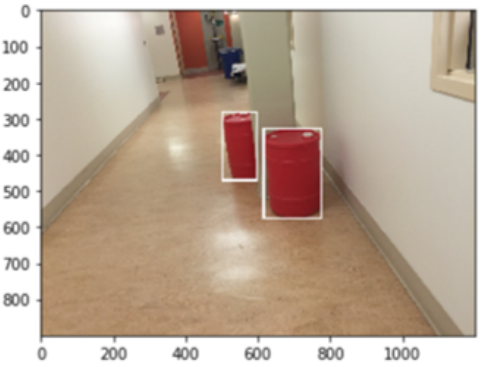

Image 003 result after algorithm improvement.

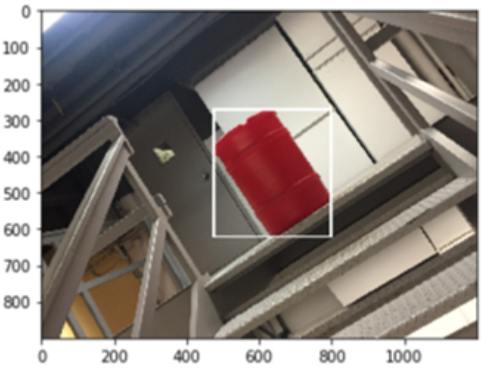

Image 004 result after algorithm improvement.

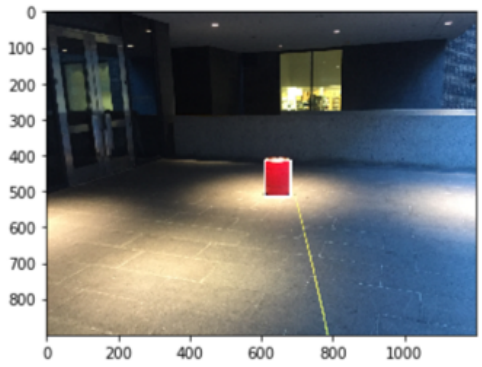

Image 005 result after algorithm improvement.

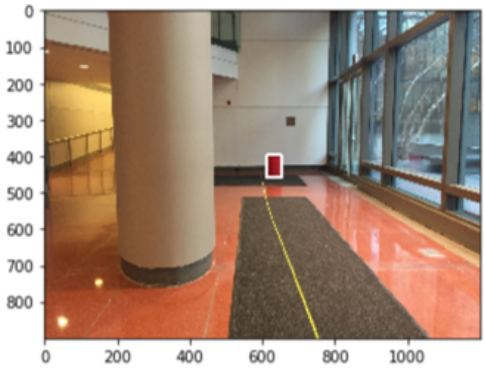

Image 006 result after algorithm improvement.

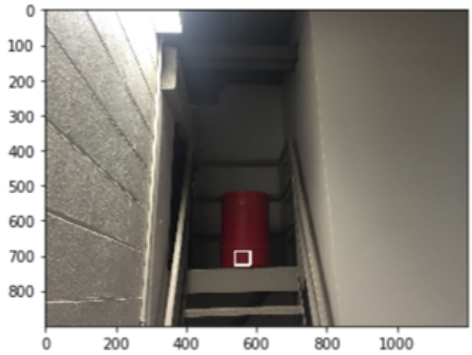

Image 007 result after algorithm improvement.

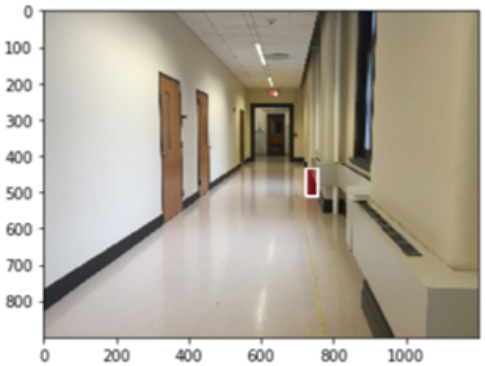

Image 008 result after algorithm improvement.

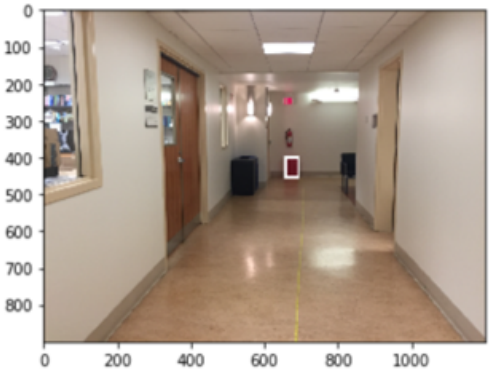

Image 009 result after algorithm improvement.

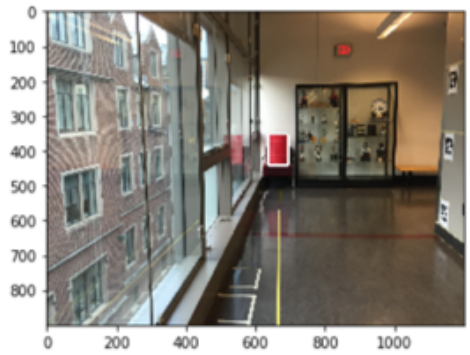

Image 010 result after algorithm improvement.

5 In Progress and Future Work

To improve the accuracy of the system, a Gaussian Mixture Model with 3 multivariate Gaussian functions algorithm was implemented and can be found on the last section of the code. However, due to memory constrains of the laptop used to complete the project, GMM did not finish running and threw several Memory Errors despite multiple attempts. Moreover, in order to choose the most optimal value for each parameter, a cross-validation algorithm was implemented. For the same reasons as GMM and due to time limitation, cross-validation was not ran completely before the submission of the project.

Future work for this project includes finishing running the GMM and cross-validation sections of the code in order to improve the accuracy of the model, especially for the darker images such as Image 007 of the test set. This could also be improved by incorporating some pre-processing to the illuminance data of the images when reading them in. Finally, a lookup table will be implemented to save the probability of every YCRCB pixel value of belonging to each class, hence reducing the run time of the program significantly.

References

[1] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[2] Alexandre Gramfort et al. Fabian Pedregosa, Gaël Varoquaux. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 2011.

[3] jdoepfert. Roipoly, 2019. "`https://github.com/jdoepfert/roipoly.py`"[Accessed 01-26-2020].

[4] Dr Lee. ECE5242 - Intelligent Autonomous Systems - Notes, 2019.

[5] OpenCV. Opencv, 2019. "`https://docs.opencv.org/2.4/modules/refman.html`"[Accessed 01-26-2020].

[6] ProgramCreek. Scikit-image examples, 2019. "`https://www.programcreek.com/python/example/88831/skimage.measure.regionprop`"[Accessed 02-06-2020].

[7] scikit-image development team. Scikit-image, 2019. "`https://scikit-image.org/docs/dev/api/skimage.measure.html`"[Accessed 02-05-2020].

[8] Wikipedia. Ycrcb, 2019. "`https://en.wikipedia.org/wiki/YCbCr`"[Accessed 02-03-2020].