Image Feature Detection & Count Using Computer Vision

MS Data Science and Business Analytics

Wayne State University

Team:
Kamrun Naher Sumi, Karim Al Zeer Alhusaini,
Matt Seraj, Mathai Paul, and Rajpal Virk

10 August 2020

Faculty Advisors:

Dr. Yanchao Liu | Dr. Sara Masoud | Dr. Murat Yildirim

Contents

1 Executive Summary

The team worked with ATCO Industries Inc whose primary business is quality inspection, quality sorting, and containment services working mainly with automotive manufacturers. The primary objective of this project is to use machine learning to identify and count the number of instances of a certain feature present in the object image. The expected value of this project is to improve upon the current process at ATCO, which is manual, time-consuming, and lacks real-time feedback on the inspection and detection results. Likewise, the current process lacks any comprehensive documentation and storing of the inspection results, which is limited to rejected parts. ATCO deems this project as a proof of concept to evaluate if an efficient and effective machine learning solution can help improve its operations. The company ultimately wants to automate its inspection process to provide real-time feedback on the inspection and results and provide a complete collection of every inspected part.
The team researched and tested multiple image processing solutions, including YOLO Object Detection, HSV Color Space Tracking and Detection, and One-Shot Learning. However, none of these techniques provided a dynamic and feasible solution that works on any industrial part with a user-friendly interface and with minimal user input. The team determined that Template Matching, an algorithm in the OpenCV library, is the best solution to the given problem. It computes the pixel to pixel match between a feature image and a source image of the entire inspect part. Though the method is rigid as a standalone algorithm, the team added multiple features to make the method more dynamic and flexible, such as rotation, scaling, warping adjustment, and threshold tuning. Most importantly, the designed program can process a batch of images using a single feature template image. The team tested the final product on multiple images while adjusting for different parameters. The team documented the limitations and recommended methods on how to overcome the limitations to produce the best detection accuracy. Lastly, the team details the future scope of this project which entails deployment of a production version on Android devices and leveraging an image database to create advanced machine learning and computer vision models.

Keywords: Computer Vision, OpenCV, Machine Learning, Template Matching

2 Project Overview

Quality control is essential to the success of any business. Increased levels of automation in manufacturing processes demand automation of quality control processes with little human intervention. To stay competitive, many businesses are turning to technologies such as Artificial Intelligence, Deep Learning, Computer Vision, etc. to automate quality inspection processes.

Computer Vision is recognized as one of the prominent technologies in the automation of quality control. It involves acquiring, processing, analyzing, and understanding of digital images. The image data is transformed into symbolic or numerical information that can then be easily analyzed by the computer.

2.1 The Sponsor

The sponsor of this project is ATCO Industries Inc. ATCO Industries was formed in 1980. It provides quality inspection and logistics services to Tier-1 automotive manufacturers in North America. With years of experience in the field of quality management, the company recognizes the importance of automation in quality inspections and control processes. The current project is part of a major automation endeavor in quality assurance delivery.

2.2 Problem Statement

The main objective of this project to research and eventually leverage a machine learning tool that can detect and count the instance of a certain feature present in the source image

of an object. The main aspect of this automation project is to develop a system capable of taking input as a source image of some part along with a template image focused on the feature to be detected and counted in the part in the source image.

The system should be capable of transforming and analyzing the source image data based on the given template, detecting features present in the source image that match with the feature present in the template image and produce a correct count of features present.

The current process of feature detection and counting is manual, slow, and prone to human errors. It also lacks real-time feedback to the inspection results and reports are only created when the inspected part is rejected. An automated system is believed to be helpful not only in expediting the process but also in increasing accuracy. Moreover, with the new automated system, an image database of each inspected part will be recorded for future work and reference.

2.3 Project Deliverables

Based on the project objective, the key project deliverables have been finalized and agreed upon by all three stakeholders:

1. A python code that takes a set of 2 images - one showing complete part and other showing the feature of interest - as input, and outputs the correct count of a certain feature.

2. Literature review and documentation related to the Python code application, including capabilities and limitations of all methods explored.

The above project deliverables had to meet the following criteria:

1. The final script needed to be user-friendly and easily navigable with a simple GUI. It also must require minimal user input. This was necessary as the final product will be used by ATCO's on-floor operators.

2. It must use open-source python libraries, such as OpenCV.

3. It must be highly dynamic with the ability to process any image or any given part and detect any given feature.

4. It must work on different operating systems, mainly Android on which the script will eventually be deployed.

2.4 Researched Techniques

With the project objective and deliverables clearly defined, the team researched existing online literature on which techniques work best. Some of the methods explored include:

1. **YOLO Object Detection:** This is one of the most common object detection techniques. It provides fast real-time image processing and provides accurate results. While this method works well on generic images, it has two main limitations. First, it is built to detect objects in images, such as cars, and not certain features in images, such as defects or part components. Second, this algorithm does not generalize well on the industrial parts that ATCO handles.

2. **One-Shot Learning:** One-shot learning is a classification or object categorization task in which one or a few examples are used to classify many new examples. Historically, deep learning algorithms fail to work well if we have only one training example. This is because, in many computer vision problems like object recognition where we have only one or a few template images in the training set, the image may have different background variations and lighting conditions. The team tried to replicate an existing online script and ran it on one of the test images provided by ATCO. The team utilized Wayne State University's On-Demand Grid as it required significant computing power to run the model. Nonetheless, the code ran for over a day and produced insignificant results. This method, therefore, was not pursued any further.

In addition to the techniques above, the team research additional feature detection techniques such as Brute Force and FLANN (SIFT, ORB algorithms) but these methods did not provide a clear solution to the problem at hand.

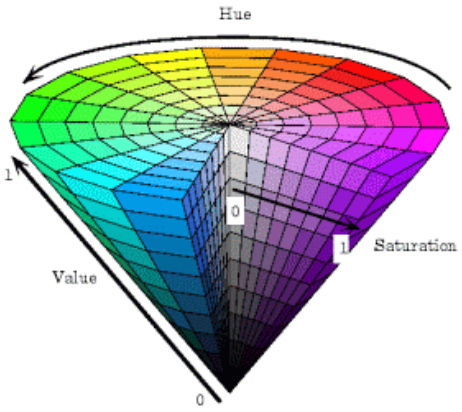
3 Explored Technique: HSV Color Tracking and Blob Count

The team researched and extensively worked on this method but dropped it as it ultimately did not meet the project criteria. This method relies on adjusting the image color space to highlight and isolate the features that need to be counted, then use Blob detection and counting to count the features. This method uses the HSV color space to isolate the feature by adjusting the HSV (Hue, Saturation, Value) of the image color space.

3.1 HSV Color Space

The HSV color space uses the Hue, Saturation, and Value representation of colors. The inverse conical chart below details each of these values.

- **Hue** corresponds to the primary color components or pigment. The value ranges from 0 - 360 degrees.
- **Saturation** is the amount of color (shade) or depth of the pigment. The value ranges from 0 - 100%.
- **Value** represents the brightness of the color. The value ranges from 0 - 100%.



The main reason behind using HSV over RGB color space is that RGB cannot separate color brightness from image luminance. HSV, on the other hand, can separate image luminance from color information to a certain degree.

3.2 HSV - OpenCV

OpenCV has a built-in function to create a color tracker GUI where the upper and lower bounds of the HSV values can be adjusted providing the ability to isolate and highlight any region of interest within the image.

Once the user selects the region of interest, the algorithm then calculates the most dominant colors within the region of interest. This can be done by simply averaging the numerical pixel values to produce the most dominant colors. However, a more accurate method uses **K-means clustering** to isolate the dominant colors. K-means clustering partitions **n** observations into **k** clusters. A parameter is used to define how many dominant colors need to be separated from all the colors in the image. We used Scikit-learn to perform the K-means clustering over OpenCV due to its flexibility and performance advantage.

Input Parameters

To demonstrate this method, we will use an example image of Lego pieces to isolate and count the number of features.

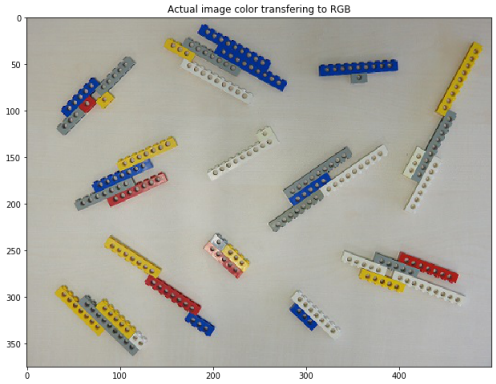
1. Source Image - Image of multiple Lego pieces or different colors and sizes.

2. Template Image - The cropped image which will be used to identify the region of interest for the HSV color range.

3.3 Methodology

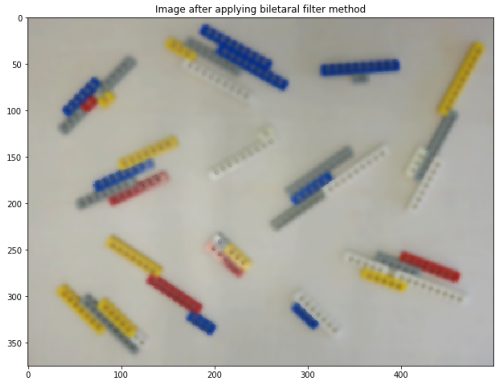
1. **Image Color Quantization** is applied in the pre-processing stage. Color quantization reduces the number of distinct colors in the image. This is done to reduce the color noise in the image and reduce the memory usage and processing time of the algorithm.

```
# quantize the image to reduce the number of colors
div = 6
img= real_image// div * div + div // 2
plt.figure(figsize =(20,8))
plt.title('Image after applying quantize method')
plt.imshow(img)
```

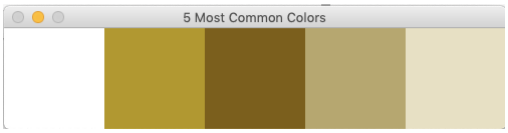


2. **Image Smoothening** is an important part of image processing. This process reduces the noise in the image. This is achieved by passing a 2D Convolutional filter to the image. Initially, we used the Gaussian filter (blur) to smoothen our image. This filter uses a Gaussian kernel to filter the image. The problem with the Gaussian filter is that it removes the noise from the image as well as the edges and shape. Since we are interested in counting the number of bricks, the edges must be preserved. Consequently, we used the **Bilateral Filter** which blurs the images but preserves the shapes of the objects in the image.

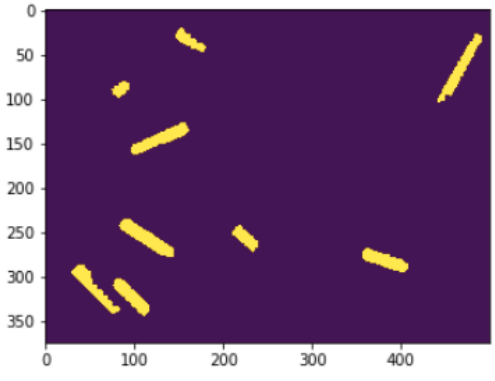
```
blurred_img = cv2.bilateralFilter(img, 8, 300, 300)
plt.figure(figsize =(20,8))
plt.title('Image after applying quantize method')
plt.imshow(blurred_img)
```



3. **Detect and Count** the features by converting the image to HSV color space. In this example, we are trying to segment and count the yellow Lego bricks. We used K-Means clustering to find the lower and upper HSV values of the yellow bricks so they can be isolated. Any colors outside the range are masked in black pixels. In this example, five dominant colors were extracted using K-Mean clustering as shown in the figure below.



After masking the image, the contours of the Lego pieces were drawn and were then used to count the number of pieces. Finally, if the size of the contour is outside the average size of the Lego piece, it is discarded. This results in the final count of the yellow Lego pieces.

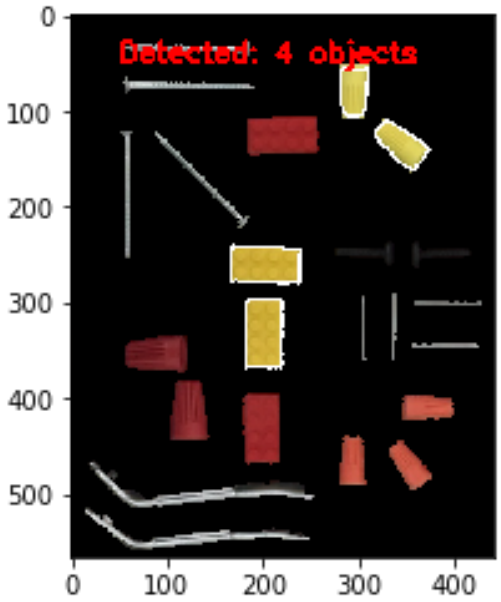


```
Detected 9 yellow pieces.
```

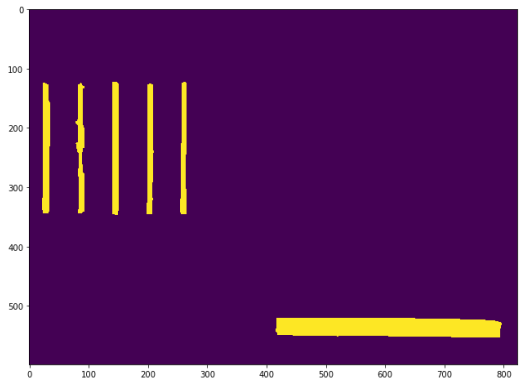
3.4 Limitations

There are some limitations to this method that need to be taken into consideration.

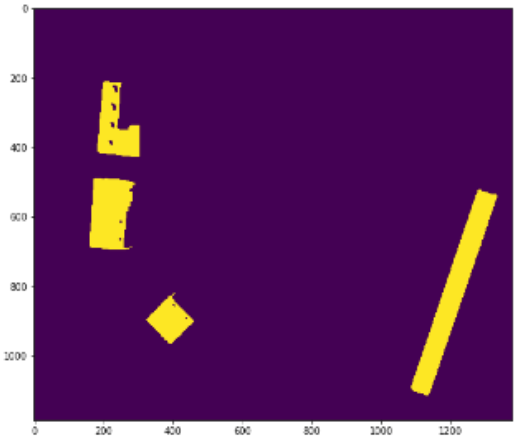
1. This method fails to account for the scenario when the dominant colors in the region of interest match with the colors of an undesired region. This results in a distorted count of the number of features in the image. In the image below, the two Lego pieces were detected but so were the caps which matched the feature's primary colors.



2. The method could miss subtle differences or details in the desired region of interest which may produce an inaccurate count of the feature. In the image below, two features (memory slots) were in close proximity to each other and the algorithm failed to identify the two features and grouped them into one feature.



3. Small color variations within the feature may be eliminated by the algorithm. This may lead to an erosion in the feature which results in a distorted detection. In the image below, the Lego piece had a light reflection that eroded the Lego piece.



As a result of the limitations outlined above, the team did not pursue this method any further as there was no feasible solution to overcome them.

4 Proposed Solution: Template Matching

Template matching is a technique in digital image processing to search and locate a patch or part, inside a larger image, that matches a template. It does a pixel to pixel match between the two images to detect features. This technique is useful in robotic navigation in manufacturing and quality control or as a way to detect edges in images.

4.1 Main Approaches

There are 2 main approaches to tackling the template matching method:

- Feature-based approach
- Template-based approach

4.1.1 Feature-based approach

This method is robust and works well when background clutter and illumination issues are present. It can be achieved through Neural

Networks or Deep learning classifiers such as AlexNet, Deep Convolutional Network, VGG, and others. This approach mainly extracts features such as shape, color, and texture from the template image and match it with similar features in the source image. The main drawback of this method is the lack of a large image dataset to train on parts that ATCO may use.

4.1.2 Template-based approach

It is more suitable for templates that are missing strong features or where the template image consists mainly of the key feature to detect. In this approach, the template image simply slides over the source image as in a 2D convolutional layer and a comparison is done to match data points of template image with the patch of input source image data.

Threshold

The main input metric used in the Template-based approach is the Threshold. The higher the threshold, the more stringent the matching comparison will be. A higher threshold value does not always constitute higher accuracy in detecting the template. For example, while detecting eyes in facial images, one must consider the variations in eye color, shape, etc. In this case, high threshold values will fail to detect eyes with variations from the original template. A threshold of 50% or 60% will yield better results than a threshold of 90% or 100%.

4.2 Template Matching - OpenCV

In Python, OpenCV has a built-in function for template matching. There are 6 primary algorithms used to perform template matching.

1. Template Matching Correlation Coefficient: this algorithm computes the correlation coefficient between the greyscale pixels of both images. It adjusts for the width and height of the template image size.

2. Template Matching Correlation Coefficient Normalized: this algorithm computes the correlation coefficient between the pixels of both images. It then normalizes the coefficient by dividing the root square sum of all elements in the matrix. This helps with removing any adverse effect of brightness and illumination in the image.

3. Template Matching Cross-Correlation: it is a similar algorithm to the previous algorithm, but it does not adjust for the width and height of the template image.

4. Template Matching Cross-Correlation Normalized: this algorithm is a normalized version of the previous algorithm.

5. Template Matching Square Difference: computes the square difference between the greyscale pixels of the two images.

6. Template Matching Square Difference Normalized: this algorithm is identical to the previous algorithm but is normalized to remove the brightness effect.

Appendix A shows the results of running each of the algorithms. In this project, we are concentrating on the **Template Matching Correlation Coefficient Normed (TM_CCOEFF_NORMED)** algorithm. We chose this method mainly because it produced the best results when tested on multiple images. It also works comparatively better than the remaining 5 algorithms in images with high brightness.

4.2.1 Template Matching Correlation Coefficient Normalized

This function slides through the image, compares the overlapped patches of size w * h against the template using the specified method then stores the comparison results. The comparison algorithm used in this function is:

$$R(x,y) = \frac{\sum_{x',y'}\left(T'(x',y').I'(x+x',y+y')\right)}{\sqrt{\sum_{x',y'} T'(x',y')^2 . \sum_{x',y'} I'(x+x',y+y')^2}}$$

(I - source Image; T - template; R - Result; $x^{'} = 0...w-1; y^{'} = 0...h-1$)

After completing the comparison, the best matches are stored as global maxima using *Threshold* measurement for source images which contain multiple template features.

Input Parameters

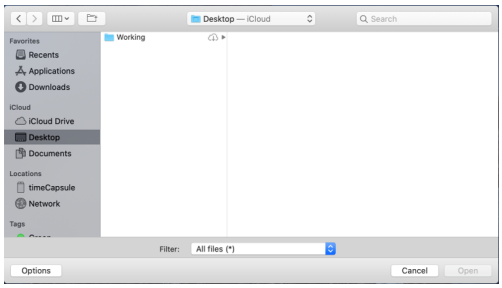
1. Source Image - The image in which we expect to find a match to the template image. In this case, it is the part that ATCO wants to find a certain feature in then count the occurrences of this feature.

2. Template Image - The patch image, which will be compared to the source image which is the selected or highlighted feature that is to be detected and counted.

4.3 Methodology

At the core of the proposed solution is the template matching algorithm in OpenCV. Nonetheless, to make the algorithm more flexible, the team added multiple processing steps and features. The process flow can be summarized in 5 key steps.

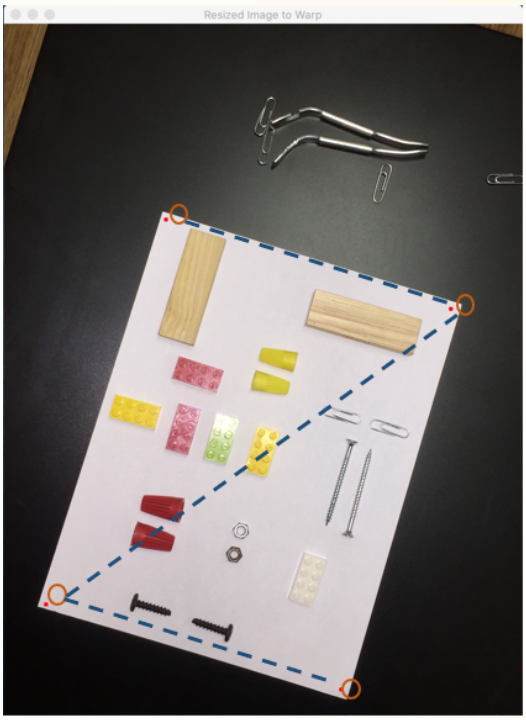
1. **Image Data Input:** In this step, the operator selects the image of the part that needs to be processed. To make it user-friendly, we added a GUI that allows the operator to select the image instead of having to alter the code manually. The program has a built-in graphical user interface for image data selection. Currently, the code is designed to input and process a single image. However, batch image input can be added for the production-ready version of the application. The image below shows the GUI that is displayed to the user.



2. **Image Pre-Processing:** This step includes 4 main sub-processes, some of which are optional.

- **Image Resizing:** OpenCV does not have a built-in fit-to-screen function so if the source image is too large to be displayed on the screen, the image can be resized so it can be displayed properly on the screen. The user is prompted to enter a float value for the resizing multiplier which can either enlarge or shrink the size of the image. If the user wishes to resize then they can enter (y) followed by a float value for the multiplier. If resizing is not needed, the user can enter (n).

- **Image Warping:** This optional step involves straightening the image if the image input is taken at an angle. The program will prompt the user to choose if warping is needed. The user can input (y) when prompted. If selected, the user needs to select 4 points on the image in a (Z) pattern and press s to save. This is demonstrated in the image below.



The output of warping is shown in the image below. The new image will be used for the remainder of the process flow.

If the user runs into an error while selecting the 4 points, they can revert by inputting (r). If the selection is successful, the user enters (s) to save. If warping is not necessary, the user enters 'n' to proceed. This option was presented to ATCO, and while it addresses an important issue regarding the image angle, ATCO may not utilize it as it requires user input. Nevertheless, it is a useful feature to keep for future use.

- **Apply Filter:** This step runs on all images in the pre-processing stage. Filtering or smoothening reduces the noise in the image and sharpens the edges for better detection during the template matching process. There are a plethora of filters that can be added but the team chose the **Bilateral Filter** as it works best with template matching. The syntax of the filter is: *bilateralFilter(src, dst, d, sigmaColor, sigmaSpace, borderType)* Where **src**: source or input image, **dst**: output image, **d**: an integer representing the diameter of the pixel neighborhood, **sigmaColor**: an integer representing the filter sigma in the color space, **sigmaSpace**: an integer representing the filter sigma in the coordinate space, **borderType**: an integer object representing the type of the border used.
- **Convert to Grey scale:** Lastly, the filtered image is converted to greyscale. This process converts the 3 color channels to 1 channel. This step is necessary as Template Matching processes single-channel images to reduce processing time.

3. **Template Selection:** Once the image has been processed, the updated processed image is then used to select the template. The user can choose from two options:

   - **Select an Existing Template:** The program checks the database for any stored existing template. If a template is present, it will be displayed, and the user will be asked for a decision on whether to use this existing template or select a new template. In the case of batch processing, using an existing template automates this process and a batch of hundreds of images can be processed without the need to change the template. However, a new template is required if either the existing batch job requires a different feature to be detected or if a different input image with dif-

ferent characteristics is processed.

- **Select a New Template:** The program displays the pre-processed image on the screen and waits for user input. The user is required to draw a rectangle from the top-left corner to the bottom-right corner of the feature to encapsulate the new template of the feature. If the selection is correct, the user must press (s) to save the template to the existing database. This process overrides the earlier stored template in the database. If the selection is incorrect, the user must press (r) to refresh the pre-processed image. This will erase any earlier drawn points or rectangles from the pre-processed image and a refreshed pre-processed image will appear on the screen.

4. **Template Matching:** The pre-processed image from Step 2 and the template selected in Step 3 are then used as input to run the template matching algorithm using the previously described Correlated Coefficient Normalized algorithm of template matching process. The script used is:

```
res=cv2.matchTemplate(rotated,template, cv2.TM_CCOEFF_NORMED)
```

Where **res**: results of template matching process, **img_gray**: pre-processed image, **template**: template of feature, **TM_CCOEFF_NORMED**: normalized correlation coefficient

The output of this step is a float value which is compared against the **Threshold** value described earlier. The user is asked to enter a threshold value, typically between 0.5 and 1.0. If the output value of the algorithm is greater than or equal to the threshold, then the area is selected as a **Region of Interest (ROI)**. The ROI corresponds to the feature that is detected. The coordinates of the ROIs are then stored as two vertices: top-left corner, and bottom-right corner. These vertices are used to draw a rectangle covering the ROI. The template matching algorithm only detects ROIs that match the scale and orientation of the provided template. However, in most cases, the template image needs to match with an ROI that exists at a different scale or orientation. To overcome this concern, we added two optional steps to increase the detection accuracy.

- **Image Scaling:** This process gives the program flexibility to select template features present in the source image at different scales. In this process, we have scaled the source image rather template as this produced the best results. The user is prompted to enter a float scaling multiplier. If the feature selected is the smallest in the source image and we need to detect features of this smallest to a new largest scale, we can input the scaling parameters from 1 onwards. But if the largest feature is selected, then we must input the scaling parameter from a float value less than 1 to an integer value of 1. However, if the selected template size is neither the smallest nor largest in the source image, then we must select input parameters from a float value less than 1 to an integer or a float value greater than 1. The scaling multiplier is applied to both the height and width (or area) of the image so it will scale up or down while preserving the aspect ratio. In addition to the scaling multiplier, the user is asked to enter an integer value which determines how long the sequence needs to be. For example, if the user enters a minimum scale value of 1 and a maximum scale value of 2 with an interval of 5, then the program with loop through 5 different scaling multiplier values: $[1, 1.25, 1.50, 1.75, 2.0]$.
- **Image Rotation:** Much like scal-

ing, rotation is another optional feature that lets the user find the features present in the image at different rotational angles. The program prompts the user to select whether to rotate the image or not. If rotation is selected, then the user must input the following 2 values:

- **Maximum Rotation Angle:** The maximum angle to which the image will be rotated. Please note the starting angle is set at 0 degrees.

- **Rotation Interval:** The interval angle between 0 and the maximum rotation angle. For example, if the maximum rotation angle is 360 degrees and the rotation interval angle is 90 degrees, then the template matching algorithm will compare and store results at every 90 degrees rotation of the input image. The image below shows a part with 5 ridges around the circle to be detected.

To detect all 5 ridges, the image needs to be rotated to find a match. The image below shows the rotated image at 45 degrees counterclockwise.

5. **Result Output:** Eventually, the final output of the template matching process is a set of coordinate values related to regions of interest on the source image. The program takes these coordinates as input and draws rectangles on these regions of interest with a counter set on it. The final output of the program is bounding boxes (rectangles) shown along with the count of these bounding boxes on the source image. These bounding boxes are the areas where features that match input template are present in the source (input) image. The current program eliminates duplicate counting by masking ROIs that are detected so they do not get double-counted. The image below shows the final bounding boxes and count on a gasket image.

If the user deems the detection results as unsatisfactory, the program prompts the user to enter a new Threshold value which will be used to measure how strict the matching needs to be. The program will run using the new Threshold value. If a satisfactory result is reached, the user can end the program by entering a Threshold of 0.

4.4 Limitations & Recommendations

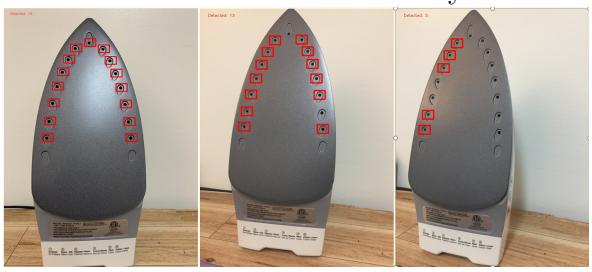
The final script was tested on a set of images of common household parts, in addition to a few images of two parts that ATCO provided. The main purpose was to understand how external parameters and conditions impact the detection accuracy.

4.4.1 Image Resolution Variations

This limitation has to do with the resolution of the camera through which the image was taken. When processing a batch of images of the same part, using a single template image of a different resolution will reduce the detection accuracy. To overcome this, the operator may re-select the template to match the resolution of the processed image. Alternatively, the operator may adjust the **scaling** and **threshold** parameters to increase the detection accuracy.

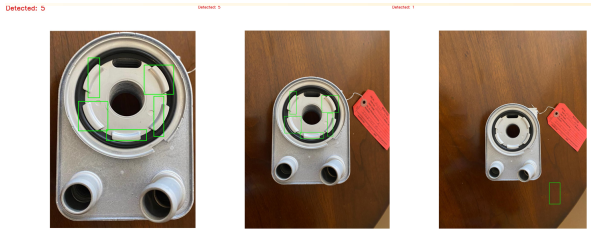
4.4.2 Camera Angle Variations

This limitation has to do with the angle at which the image was taken, relative to the part. If the operator is processing a batch of images, then using a single template image taken at a different angle from the source image may reduce the detection accuracy. To overcome this, the operator may re-select the template to match the angle of the source image. Alternatively, the operator may adjust the **rotation**, **warping**, and **threshold** parameters to increase the detection accuracy. The image below shows how using one template to process images taken at different angles can reduce the detection accuracy.



4.4.3 Camera Distance Variations

This limitation has to do with the distance at which the image was taken, relative to the part. If the operator is processing a batch of images, then using a single template image taken at a different distance from the source image may reduce the detection accuracy. To overcome this, the operator may re-select the template to match the distance of the source image. Alternatively, the operator may adjust the **scaling** and **threshold** parameters to increase the detection accuracy. The image below shows how using a single template to process multiple images taken at different distances can reduce the detection accuracy.



4.4.4 Lighting & Visual Obstructions

If the source image contains visual obstructions or lighting problems such as flash or glare, the algorithm fails to detect the correct number of features. The operator will need to take an image with minimal lighting and visual obstructions to detect the correct number of features.

In the image above, brightness and visual distortions prevented the detection of all the box labels.

5 Future Scope

1. **Android Application:** The script used in this project is portable to an Android environment. The script can be deployed to a production environment and can later be installed on the Panasonic FZ-X1 device which ATCO uses to write reports of the rejected parts. ATCO has a team of developers who can transfer the script into Android.

2. **ATCO 2019 Project Integration**: This project can be integrated with the ATCO 2019 project which dealt with image orientation adjustment. This ensures that all the images taken meet certain standards and are within an acceptable degree of variance from the correct template. By merging both scripts, the limitations described above can be mostly eliminated.

3. **Part Image Database**: If this project is implemented, ATCO can create a database of all the parts inspected. This helps with documentation and data warehousing of the parts. It also provides a database of images that can be used to train future computer vision projects which use pre-trained models, such as Convolutional Neural Networks, YOLO, and others. These models can learn features about the provided images which helps detect certain defects or components within the source image.

6 Conclusion

As a team, we have learned valuable skills in computer vision and project management. Once we had the deliverables clearly defined, the team did extensive research to provide the best solution to the problem. The team started with an existing algorithm - Template Matching - which is quite rigid and restrictive on its own, then improved upon the method by adding many features that make it flexible and fit for the problem given. The solution provided is unique and can be used in other computer vision problems.

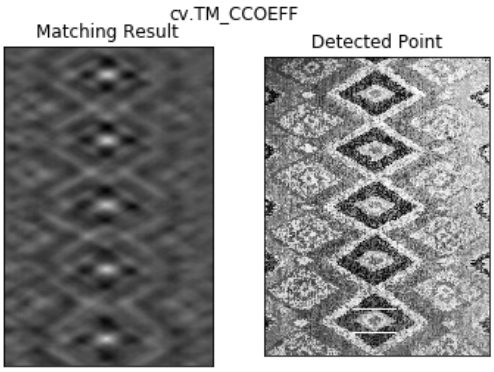
ATCO was happy with the final model we delivered to them. It satisfied their requirements and is reproducible on their computers and systems. We not only delivered a working solution, but one ATCO can use to further develop and enhance easily. They can also change the criteria variables to tune the model to fit their business needs. The model gives the next project iteration a good, well-documented foundation to build from and continue to provide a quality product to ATCO in future years.

7 Appendix

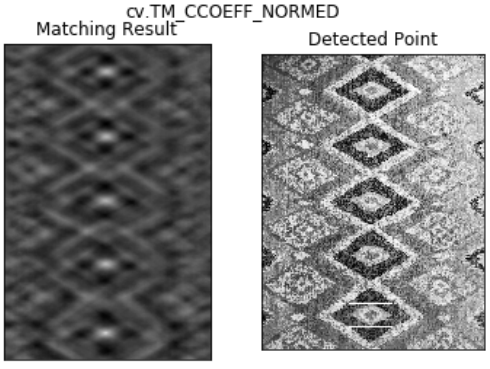
7.1 Appendix A: Template Matching Algorithms

Each of the 6 algorithms were tested on one image of a carpet with a pattern.

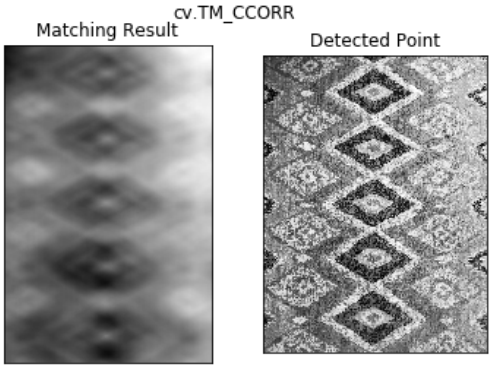
1. Template Matching Correlation Coefficient:



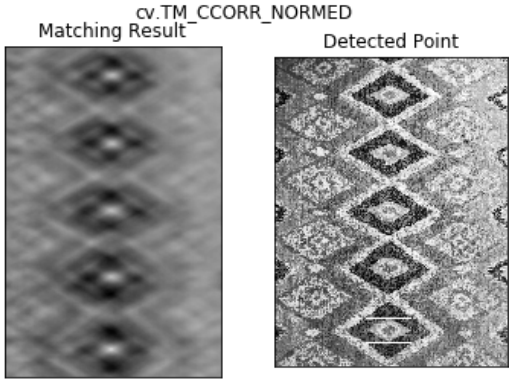
2. Template Matching Correlation Coefficient Normalized:



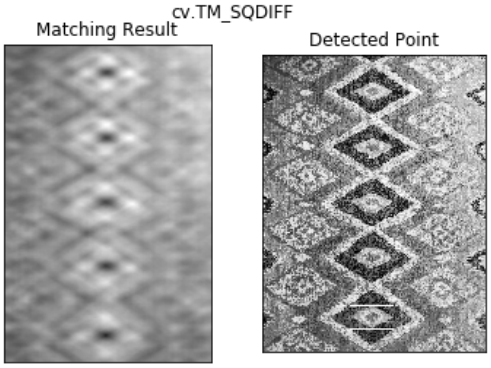
3. Template Matching Cross-Correlation:



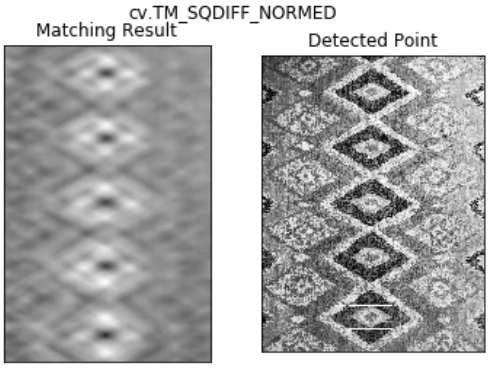
4. Template Matching Cross-Correlation Normalized:



5. Template Matching Square Difference:



6. Template Matching Square Difference Normalized:



7.2 Appendix B: Tools Employed

- Anaconda
- Python 3.6
- Libraries: os, time, cv2, numpy, imutils, easygui

7.3 Appendix C: Running the Program

In order to run the program, an Anaconda environment using Python 3.6 should be loaded with the following libraries: os, time, cv2, numpy, imutils, easygui.

Once these libraries are loaded, run the following command from within the environment:
python template_matching-v1.4.py

References

[1] OpenCV Library Documentation. *OpenCV-Python Tutorials*. `https://bit.ly/3fJqjpk`.

[2] Roberto Brunelli. *Template Matching Techniques in Computer Vision*. `https://bit.ly/2Bq9l0B`.

[3] Harrison Kinsley. *Template Matching OpenCV Python Tutorial*. `https://bit.ly/2YgKO6Z`.

[4] Kseniia Nikolskaia, Nadezhda Ezhova, Anton Sinkov, and Maksim Medvedev. *Skin Detection Technique Based on HSV Color Model and SLIC Segmentation Method*. `http://ceur-ws.org/Vol-2281/paper-13.pdf`.

[5] Adrian Rosebrock. *Object Detection and OpenCV Tutorials*. `https://www.pyimagesearch.com/`.