A Method to Automatically Learn Appearance Variability in Machine Parts During Appliance Manufacturing

---

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Engineering

---

by
Benjamin H. Shumpert
May 2019

Accepted by:
Dr. Adam Hoover, Committee Chair
Dr. Richard Brooks
Dr. Yingjie Lao

Abstract

This thesis considers the problem of learning the variability in appearance of machine parts for automated inspection during appliance manufacturing. Product quality is of great importance to appliance manufacturers. To ensure quality standards are met, manufacturers hire inspectors, whose role is to perform a manual visual inspection of each product that comes down the assembly line. If the inspector finds a defect on a given product (e.g. a missing bolt, a loose connector, or an incorrect label), that product is flagged for repair. This traditional system works well in general; however, it has two main drawbacks. First, the cost of labor to hire inspectors is significant. Second, over the course of a given day, the performance of human inspectors tends to decrease.

In recent decades, computer vision systems have been developed to augment or in some cases completely automate the inspection process. These systems work by "learning" which part appearances are acceptable and which are not through a process called training. For most inspection problems, the training process involves taking thousands (usually on the order of 10^5 or 10^6) of images of the part to be inspected and manually labeling each of those images as "acceptable" or "unacceptable." Once labeled, the dataset is run through a classification algorithm, which uses the labels to produce a reduced set of acceptable images that span the entire range of the part's acceptable appearance. Likewise, a reduced set of unacceptable images that spans the entire range of the part's unacceptable appearance is also produced. Thus, the variability in part appearance is learned via labeling and classification.

The process of manually labeling the large image datasets needed to train an inspection system in this manner is tedious and burdensome, often consuming tens to hundreds of worker-hours for a single inspection problem. Thus, this thesis proposes a generic method to automatically learn the appearance variability of any part, eliminating the need to manually label large datasets for each part.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis considers the problem of learning the variability in appearance of machine parts for automated inspection during appliance manufacturing. In the context of this thesis, machine parts refer to metal or plastic pieces used for the assembly of home appliances. Examples of machine parts include hose clamps, fasteners (such as screws, bolts, and ties), and electrical connectors. Examples of home appliances include refrigerators, ovens, stove tops, washing machines, and dryers. Appliance manufacturing is the process of commercially building home appliances in large scale (at least several hundred machines per day). This process typically involves tens to hundreds of workers assembling these units on an assembly line in a manufacturing plant. Product quality is of great importance to appliance manufacturers. Thus, many of these manufacturers implement inspection protocols to ensure their products are not defective. An inspection is the process of looking at a given part (like a hose clamp) and ensuring that it is installed properly (fully seated on the hose).

Traditionally, inspections are performed manually by humans, called inspectors. The inspector's role is to perform a manual visual inspection of each product that comes down the assembly line. If the inspector finds a defect on a given product (e.g. a missing bolt, a loose connector, or an incorrect label), that product is flagged for repair so that the issue(s) can be fixed before the product is shipped to a customer. This traditional system works well in general; however, it has two main drawbacks. First, the cost of labor to hire inspectors is significant. Second, during a given day, the performance of human inspectors tends to decrease over time due to vigilance decrement[1]

[1]In human factors psychology, vigilance decrement is the decline in human performance that occurs after several hours of repeatedly performing the same task.

[1].

In recent decades, computer vision systems have been developed to augment or in some cases completely automate the inspection process. The basic idea behind these systems is to compare an image of a part of a product coming down the assembly line, called a live image, to an image of the properly installed part, called an inspection template, in real-time. If the live image matches the template sufficiently well, the product passes the inspection; otherwise, it fails the inspection and is marked for repair. For a given inspection, the template image is determined through a process called "training." In the simplest case, the training process is straightforward as the template is simply set to an image of a properly installed part. For each product that comes down the assembly line, it either sufficiently matches this template, in which case it passes, or it does not, in which case it fails. An example of this type of inspection is a fill-level inspection for soft drink bottles (see Figure 1.1). As in the case of the fill-level inspection, the single template technique only works when the properly installed part has a unique appearance. In the case of the bottles, a properly-filled bottle looks the same every time, thus the variability in the appearance of the bottle is sufficiently modeled by a single template.

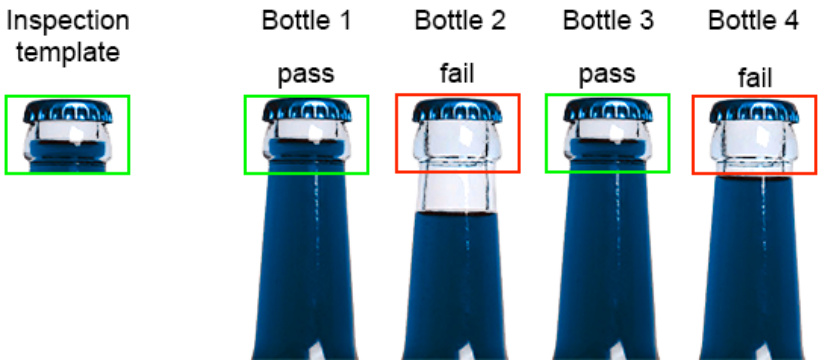


Figure 1.1: Fill-level inspection on soft drink bottles. Bottles 1 and 3 pass the inspection while bottles 2 and 4 fail. Notice how a properly-filled bottle has a unique appearance.

The appearance of a properly installed part is not always unique. For example, consider a hand-written zip code recognizer. For each live image, the system knows that the image represents one of the 10 digits 0-9. Given that there are now 10 options, a single inspection template is no longer sufficient. There must be at least one template for each digit. Furthermore, since the digits are hand-written, not all live images of the same digit will have the same appearance (see Figure 1.2). That is, the part no longer has a unique appearance. In this case, the system is trained (i.e.

the templates are determined) with a technique called supervised learning. In this approach, several examples of hand-written digits are labeled by humans as "0," "1," "2," and so on. To conduct a live inspection, then, each unknown digit can be compared against a set of templates for each class (each digit) to find the best match. In this way, the appearance variability is modeled by a set of templates for each class.

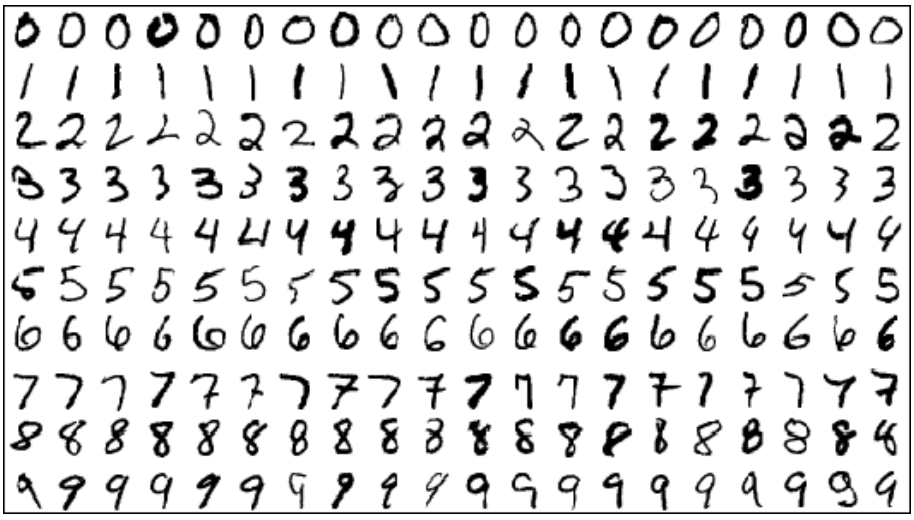


Figure 1.2: Variability in appearance of hand-written digits. A visual inspection system looking for these digits knows that there are only 10 options; however, no two hand-written figures representing the same digit look the same.

In the context of industrial manufacturing inspections, most inspection problems are even more complex than the two cases discussed above. In this context, the variability in part appearance tends to be high. Many factors can cause variability in appearance, such as rotations of the part, partial occlusions of the part, or positioning of the part. An example of this type of part is shown in Figure 1.3, which displays 35 different appearances of the same electrical connector within 35 different washing machines. In the images, the connector (white piece) is clipped into its mount (blue piece). In this case, the cause of high appearance variability is the wire (dark green line) that runs in between the connector and the camera lens. In no two images does the wire appear the exact same way.

In general, for an inspection problem with high variability in appearance, the inspection is conducted by comparing each live image to a set of acceptable templates (templates that demonstrate an acceptable appearance) and a set of unacceptable templates (templates that demonstrate an unacceptable appearance, or defect). If the live image matches one of the acceptable templates

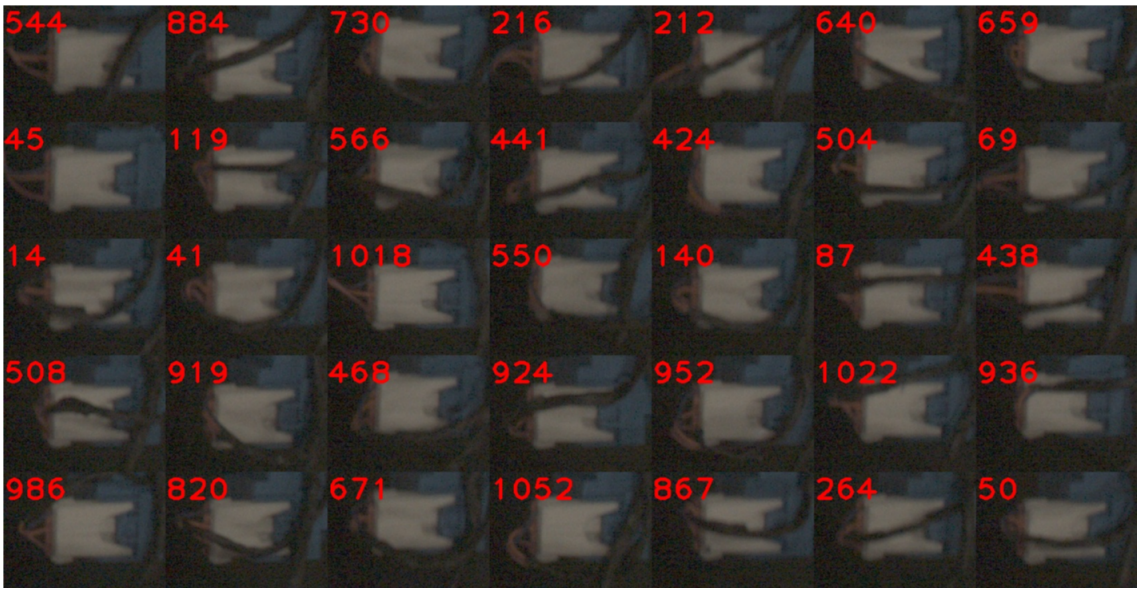


Figure 1.3: An electrical connector that demonstrates high appearance variability due to partial occlusion by a wire. Notice how, though the part has many appearances, none of these images are defects (the connector is fully connected). The numbers in each image are simply identifiers and are irrelevant in this section.

best, the product passes; if it matches one of the unacceptable templates best, it fails. In this way, the variability is modeled by two sets of templates—one for acceptable appearances and one for unacceptable appearances. The training process for this type of inspection must determine the degree of variability in part appearance. That is, the range of acceptable (and, likewise, unacceptable) appearances is unknown and must be learned. Furthermore, unlike the hand-written digit example, the number of inspection templates required to sufficiently capture a part's range of variability (the total number of classes) is also unknown, and thus must be learned as well.

Traditionally, part variability is "learned" manually by human analysis and labeling. For each inspection problem, experts on that specific part determine what the range of variability is (e.g. this part is acceptable if it is positioned at 10, 30, or 50 degrees and it is unacceptable if it is positioned at 20 or 40 degrees). Once the range is determined, each image in a large dataset, perhaps on the order of 10^5 or 10^6 images, must be labeled as one of the possible appearances. The labeled images are then run through a classifier, and the templates for each possible appearance are produced. There are three main drawbacks to this approach. First, the range of variability is left to be determined by humans, thus it is prone to error. Humans could easily miss a possible appearance. Second, the process of manually labeling the large image datasets needed to train an

inspection system in this manner is tedious and burdensome, often consuming tens to hundreds of worker-hours for a single inspection problem. Finally, this method is problem-specific, and thus must be repeated for each new inspection problem.

To overcome the shortcomings of manually learning part variability, this thesis proposes a method to learn part variability automatically. This method not only eliminates the need to label large datasets, but it also works for any inspection problem, eliminating the need for domain-specific knowledge on a part to be inspected during the training phase.

1.1 Background

1.1.1 Template Matching

A common method visual inspection systems use to compare images, such as a template image to a live image, is a process called template matching. Though there are several, the specific template matching algorithm used in this thesis is known as the normalized cross correlation (NCC). The NCC is a pixel-by-pixel comparison algorithm that determines the best match location of a template image inside a search window image, which is generally larger than the template. In addition to reporting a location, the NCC also reports an associated match score, which ranges from 0-1. A lower score indicates a worse match, whereas a higher score indicates a better match. Another byproduct of the NCC is the matched spatial filter (MSF) image, which depicts the score of the template at all possible locations across the search window. In the MSF, brighter spots correspond to locations that match better and darker spots correspond to locations that match worse. The brightest point on the MSF is the location that is returned by the NCC as the best match location.

The concept of the NCC is often best understood through an example. The following example will attempt to find the letter 'e' in an image of text. The inspection template and inspection search window for this example can be seen in Figure 1.4.

The MSF image for this match is shown in Figure 1.5. The highest scoring location across entire window is (282, 35); that is, column 282 and row 35, corresponding to the 'e' in "are" on the second line. The score is a perfect 1.0 since the template was actually cut directly from the search window in this case.

The equation for the value of the NCC at location (x, y) is given by

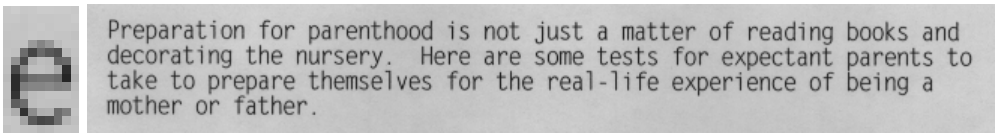


Figure 1.4: The inspection template (left, enlarged) and the inspection search window (right). The NCC looks for the temple inside the window. Clearly, several 'e's exist in the window, so the NCC will find the 'e' that most closely matches the one in the template.

Figure 1.5: The MSF image for the NCC example. Notice how bright spots occur at the locations corresponding to the center of all letters that resemble an 'e' in the search window.

$$NCC(x,y) = \frac{\sum_{s=-a}^{a}\sum_{t=-b}^{b}(T(s,t) * I(x+s,y+t))}{\sqrt{\sum_{s=-a}^{a}\sum_{t=-b}^{b}T(s,t)^2 * \sum_{s=-a}^{a}\sum_{t=-b}^{b}I(x+s,y+t)^2}}$$

where T is the template image, I is the inspection search window image, a is half the width of the template (rounded down), and b is half the height of the template (rounded down). The term in the denominator is a normalizing term that ensures all scores range from 0-1. To compute the entire NCC, this equation must be computed at each location in the search window.

1.1.2 Gradient Images

When comparing two images of machine parts, it is common to use the gradient images of the inspection template and inspection window as opposed to the originals. This is because the gradient images are resistant to changes in lighting and ignore color differences. In addition, machine parts usually have strong gradients as they are made by humans.

The gradient image is simply an image in which edge pixels (pixels that are different in intensity to their neighbors) are brighter than non-edge pixels (pixels that are similar in intensity to their neighbors). The gradient algorithm used in this thesis is known as the Sobel edge detector.

The Sobel edge detector works by finding the horizontal and vertical gradients independently, then combining the results to produce a final edge image. The horizontal edge image, G_x, is found by convolving the input image with the horizontal gradient kernel (matrix) and the vertical edge image, G_y, is found by convolving the input image with the vertical gradient kernel. The horizontal

and vertical edge images are given by

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$

where I is the input image and the * operator denotes convolution. The convolution operation is given by

$$k * I(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} k(s, t) I(x - s, y - t)$$

where k is the kernel, I is the input image, x is the row in the input image, y is the column in the input image, a is half the width of the kernel (rounded down), and b is half the height of the kernel (rounded down). This equation must be computed at each (x, y) location in the input image.

Once the horizontal and vertical gradients are found, the final gradient image is computed by taking the square root of the sum of the squares of each intermediate result; that is,

$$G = \sqrt{G_x^2 + G_y^2}$$

An example of the Sobel edge detector being applied to an image is shown in Figure 1.6.

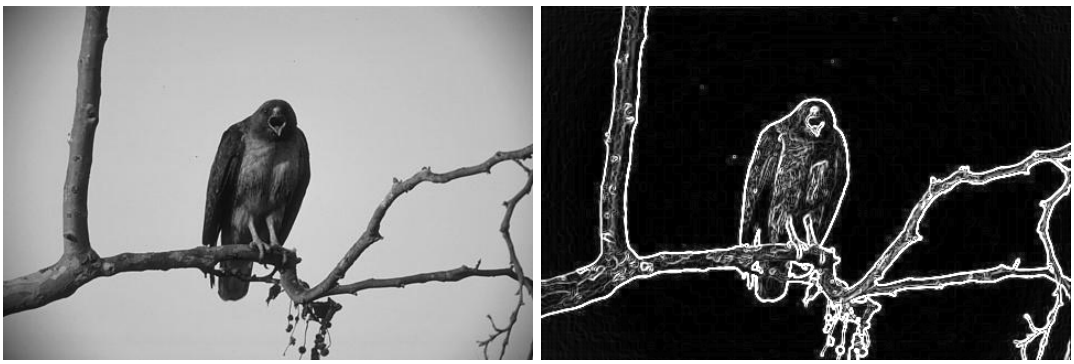


Figure 1.6: An input image of a hawk (left) and its Sobel gradient image (right). Notice how edges appear as brighter than non-edges.

1.1.3 Inspection Process

This section outlines how the inspection process is intended to work while running live on an assembly line. The inspection process occurs in two phases. The first phase determines whether or not a product is in front of the camera. If the first phase determines that a product is present, the second phase performs the actual inspection.

The first phase is called the trigger phase as it employs a process called trigger matching to determine whether or not a product is present. The idea behind this process is to look for a trigger, a pattern that is guaranteed to look the same from product to product and to never be occluded, inside a trigger search window, an area within the camera frame in which the trigger may appear. Before trigger matching can execute, an example image of the trigger, called the trigger template, must be stored. During execution, for each live frame that is grabbed from the camera, the trigger template is matched to the trigger window using the NCC. If the match score is below some threshold, the trigger is considered not found, and it is assumed that no product is present. On the other hand, if the match score is above the threshold, the trigger is considered found, and it is assumed that a product is present. The trigger matching process is illustrated in Figure 1.7.

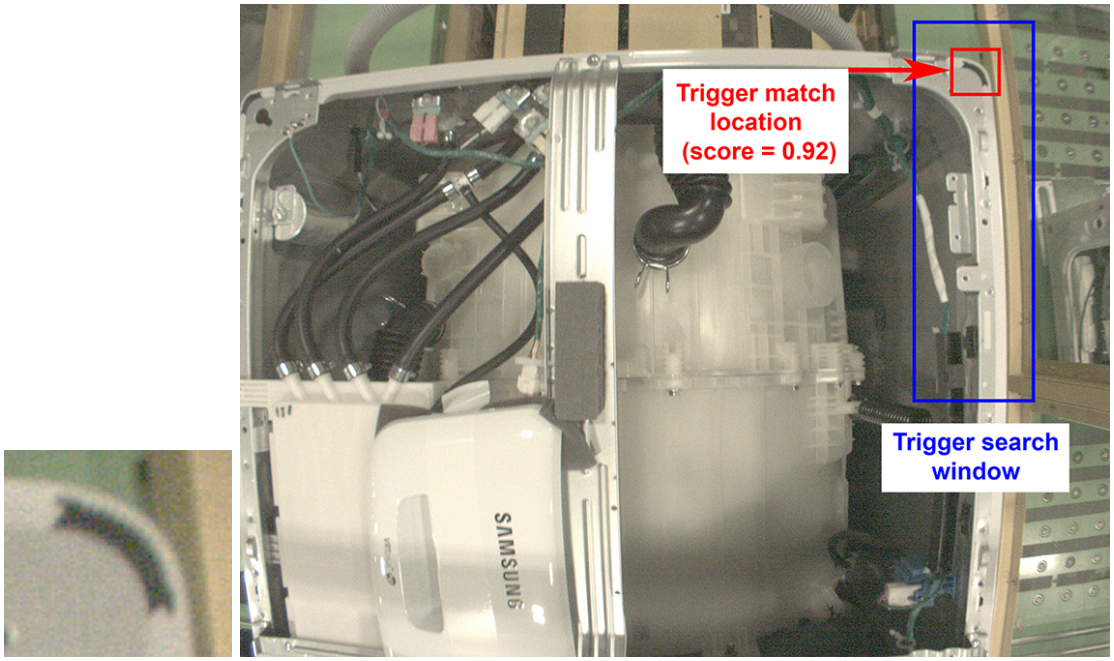


Figure 1.7: Illustration of the trigger phase on a washing machine. The trigger template is shown enlarged on the left, while the trigger matching process itself is shown on the right. In this case, the trigger threshold was set to 0.72, so this frame's score of 0.92 is high enough to assume there is a machine present; clearly, this assumption is correct in this case.

If a trigger is found during the trigger phase, the second phase, called the inspection phase, is executed. The inspection phase is similar to the trigger phase in that it looks for the part to be inspected inside an inspection window, an area in which the part is expected to appear. The inspection window is placed at a pre-determined, constant two-dimensional offset from the trigger match location. The horizontal offset is denoted as dx, and the vertical offset is denoted as dy. Before the inspection can execute, an example of the properly installed part, the inspection template, must be stored. During execution, the inspection template is matched against the inspection window using the NCC. The resulting location from this NCC operation is called the part match location, and the area to be inspected is defined as the area with top-left corner at the part match location and size equivalent to that of the inspection template. Unlike the trigger phase, the resulting score from this NCC operation is irrelevant and not used. To score the part, the NCC is taken between the gradient image of inspection template and the gradient image of the area to be inspected. The score resulting from this second NCC operation is called the match score. If the match score is above some threshold, the part is assumed to be correctly installed, and the product passes the inspection; otherwise, it fails. The inspection phase is illustrated in Figure 1.8.

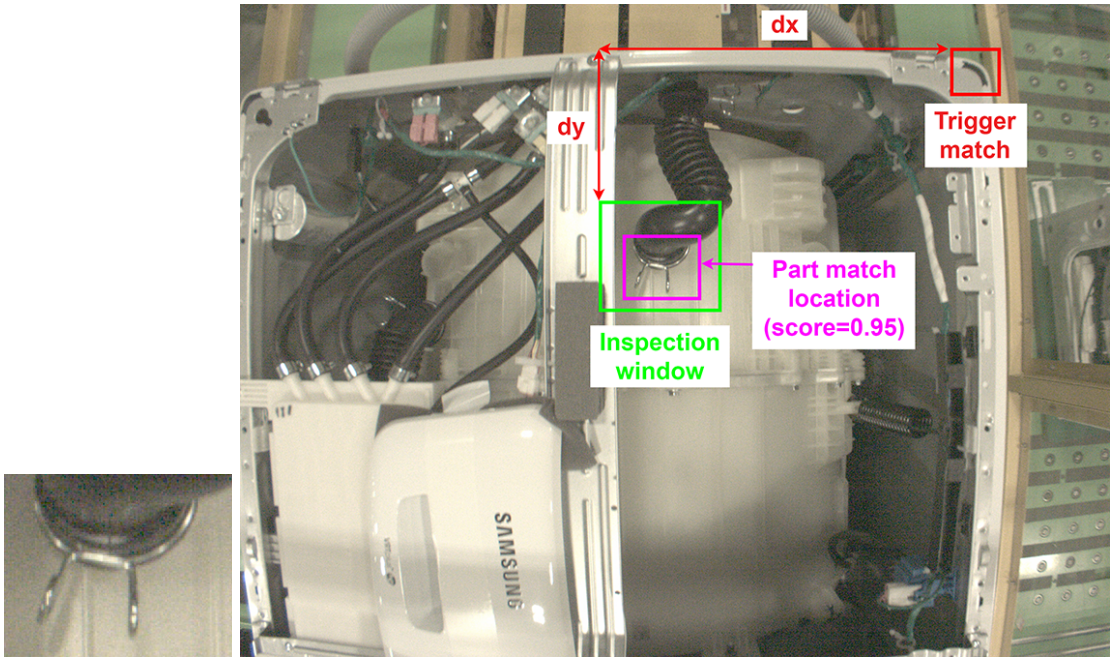


Figure 1.8: Illustration of the inspection phase on a washing machine. The inspection template is shown enlarged on the left, while the inspection itself is shown on the right. In this case, the inspection threshold was set to 0.85, so this machine's score of 0.95 is high enough to assume the part is properly installed.

During the inspection phase, the match location and match score are not computed with a single NCC. This is because each of the two NCC operations provide its own advantages in determining its respective result. For locating the part, it was determined that taking the NCC between the original window and original template is more effective than taking the NCC between the gradient image of the window and the gradient image of the template, especially when the live image looks very different from the template (see Figure 1.9). For scoring the part, it was determined that taking the NCC on the gradient images is more useful than taking the NCC on the original images. This is for two reasons. First, scores resulting from the NCC operation on the original images have a condensed range. That is, most live images match the template extremely well with scores ranging from 0.98 for the worst match to 1.0 for the best match. The NCC operation on the gradient images, on the other hand, provides a wider range of scores, typically in the range of 0.37-0.98. The wider score range allows for more discriminability between machines. Second, machine parts tend to have well-defined edges. Moreover, the most important attributes of a properly installed part are its shape (is it the right part?) and its position (is it in the right place?). Both of these attributes are apparent in the gradient image. The less important attributes include surface texture and color. Neither of these attributes are apparent in the gradient image. Therefore, scoring the gradient images highlights the attributes that are important to the inspection while subduing those that are less important.

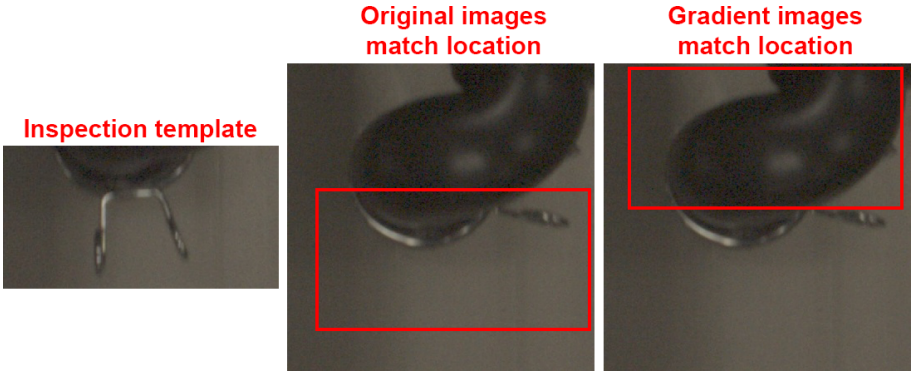


Figure 1.9: Using the original images vs. the gradient images for locating the part. The inspection template is shown on the left. The middle image is the inspection window with the match location (overlaid as a rectangle) as determined by the NCC between the original images. The image on the right is the same inspection window with the match location as determined by the NCC between the gradient images. The original, rather than the gradient, window is shown here for ease of comparison with the other window. Clearly, the NCC between the original images is more effective at locating the part.

1.1.4 Formalizing Appearance Variability

This section covers the varying degrees of range in the variability of machine part appearance. Several concepts from the opening discussion are revisited and formalized.

The basic idea behind automatic visual inspection systems is to compare an image of a part of a product on the assembly line, called a live image, to an example image of the properly installed part, called an inspection template. If the live image matches the template sufficiently well, the product is considered acceptable; otherwise, the product is considered unacceptable and is flagged for repair. This process is described in detail in Section 1.1.3.

The situation above describes a single-class inspection problem. That is, only one template is needed to conduct the inspection. This type of inspection is sufficient when there is a large difference in appearance before and after the part is installed. A large bolt is an example of this type of problem. Before installation, there is a small, dark pilot hole. After installation, there is a large, metal bolt head. The difference between these two appearances is sufficiently large to warrant a single-class inspection. See Figure 1.10 for an illustration of the concept of a single-class inspection problem.

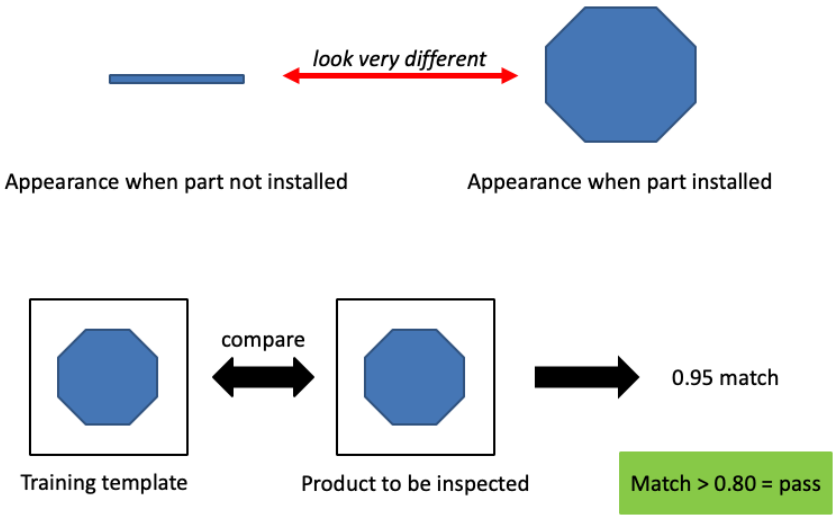


Figure 1.10: Single-class inspection problem. Notice how there is a large difference in appearance before and after the part is installed. In this case, the product passes the inspection as the live image match score (0.95) is above the pre-determined threshold (0.80).

When the difference in appearance before and after installation is small (or when the properly installed part has two appearances that are similar to each other), the inspection is classified as a dual-class inspection problem. In this case, two templates are used. One is an example of the product before the part's installation, and the other is an example of the part after proper installation. The live image is compared to both templates. If the properly installed template matches best, the product is considered acceptable. On the other hand, if the uninstalled template matches best, the product is considered unacceptable. See Figure 1.11 for an illustration of the concept of a dual-class inspection problem.

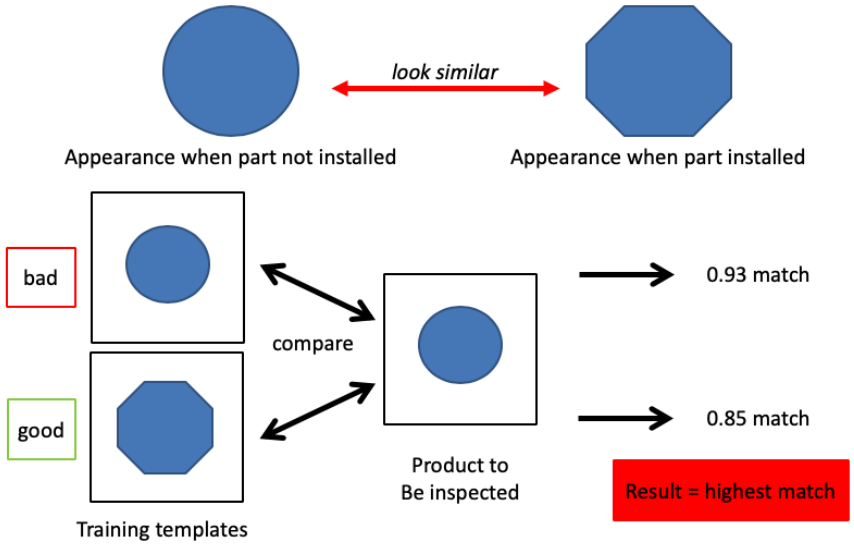


Figure 1.11: Dual-class inspection problem. Notice how there is a small difference in appearance before and after the part is installed. In this case, the product fails the inspection as the live image matches the uninstalled template better than the installed template.

In the case of a single-class or dual-class inspection problem, there is no need to label thousands of training images. The system can be trained by simply being provided with either one or two inspection templates, depending on the class. In practical application to appliance manufacturing problems, however, few inspection problems fit into one of these two classes. The appearance of most parts has some degree of variability that cannot be sufficiently captured with one or two templates. Even relatively simple parts can have a wide range of variability. Variability is caused by several factors including but not limited to: rotation of the part, partial occlusion of the part, and, especially for metal parts, optic specularities on the part. Inspection problems that exhibit variability in appearance are classified as multi-class inspection problems.

In multi-class problems, several, sometimes hundreds of templates are required to perform an inspection. Multi-class templates are divided into two sets—acceptable templates and unacceptable templates. Acceptable templates are those that capture the range of acceptable part appearances. The acceptable template set is composed of N acceptable templates. Unacceptable templates are those that capture the range of unacceptable part appearances (defects). The unacceptable template set is composed of M unacceptable templates. To perform an inspection, the inspection system compares the live image to all $N + M$ templates. If the live image is most similar to an acceptable template, the product is considered acceptable. On the other hand, if the live image is most similar to an unacceptable temple, the product is considered unacceptable. See Figure 1.12 for an illustration of the concept of a multi-class inspection problem.

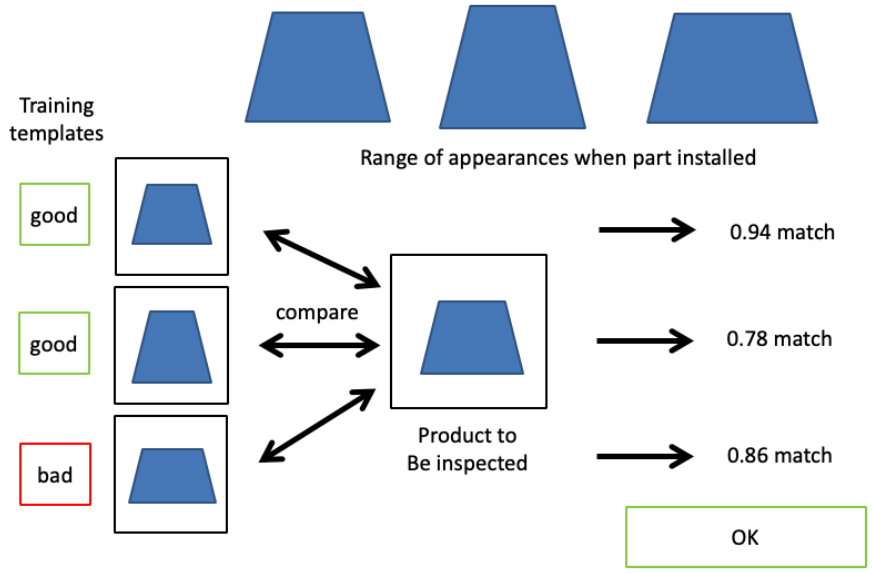


Figure 1.12: Multi-class inspection problem. Notice how this part exhibits multiple appearances. In this case, the product passes the inspection as the live image matches an acceptable template best.

To illustrate the concept of a multi-class problem, consider the electrical connector shown in Figure 1.3. The wire leading to this particular connector tends to run in between the inspection camera and the connector itself. In this case, the inspection should not fail simply because there is a wire running across the view of the connector; rather, there should be a template capturing that appearance and classifying it as acceptable. In fact, there should be an acceptable template for when the wire runs across the upper part of the connector, the middle part of the connector, the bottom part of the connector as well as when the wire crosses at 10 degrees, 20 degrees, 30 degrees,

etc. Likewise, there should be unacceptable templates for when the connector is loose on one side, loose on the other side, loose on both sides, not present, etc.

The large number of templates required by multi-class inspection problems is often generated by a classification algorithm, such as the k-nearest neighbors (k-NN) algorithm, which classifies each image based on its assigned label. Through the classification process, the original dataset on the order of hundreds of thousands or millions of images is reduced to tens or hundreds of images, called classes, each labeled as either acceptable or unacceptable. Together, all class images span the entire range of the part's appearance that was captured in the training data. After classification, each class image can be used as an inspection template by the visual inspection system to conduct live product inspections. The lableing and classification process is considered to be a supervised learning method. That is, the machine training process is aided by human interaction.

1.2 Related Work

An early example of an automatic visual inspection system was developed by Perkins in 1983 [2]. In 1986, Tsatsoulis and Fu developed a similar system that improved on Perkins' system by inspecting complex assemblies step by step rather than all at once [3]. This project also produced the Simple Assembly Inspection Language (SAIL), which is a primitive C-based programming language used to program inspection algorithms.

Since the creation of these early inspection systems, much progress has been made in this domain. Overall progress is well-documented by literature surveys [8, 9]. In addition, several specific proposals have been made to improve the base technology that supports inspection systems. Aksoy et al. proposed a method that eliminates the need for inspection templates and the NCC computation [7]. This method works best for parts with well-defined edges that intersect at right angles. Killing et al. proposed a fuzzy neural network as opposed to the classic simple threshold-based system [6]. The team found that the fuzzy neural network outperformed the simple threshold system when a live image was not similar to any of the training images. Choi et al. proposed a method to automatically determine an optimal inspection template for a given inspection problem [4]. Fouda and Ragab proposed a fast-NCC algorithm that computes the NCC score more efficiently than the traditional algorithm by utilizing a pyramid data structure [5].

Many industries rely on visual inspection systems. The semiconductor industry relies on

automatic inspection technology to verify the proper construction of the small, complex assemblies common to that industry [16]. The semiconductor industry also leverages systems that can automatically detect the quality of solder joints [13]. The automotive industry relies on inspection systems to verify the proper assembly of fasteners [14] and weld joints [17]. Though not manufacturing based, the railroad industry uses automatic inspection technology to detect cracks in the rail surface [10] and missing fasteners [11, 12].

The most relevant work to this thesis is a patent by Cognex Corporation [15]. In this patent, the authors describe a semi-supervised method for determining the variability in part appearance.

Again, one of the main advantages of the method explored in this thesis is that it is generic, and thus can be applied to any part. All of the works here are tailored to a specific part (or family of parts) and would have to be reprogrammed to work on new parts.

1.3 Novelty

As opposed to the supervised method discussed above in Section 1.1.4, this thesis proposes the use of an unsupervised learning approach, which does not require human intervention, to learn the variability in machine part appearance. The method takes the unlabeled training image dataset as input and produces a set of inspection templates, each template representing a cluster of similar-looking images in the original dataset as determined by the NCC score. In addition, this method (henceforth, the clustering algorithm) is generic and can be applied to any part.

Since no labels are required by this algorithm, the produced template set is not automatically separated into an acceptable set and an unacceptable set. Thus, once the templates are produced, each must be labeled by a human as acceptable or unacceptable in order to be usable by an inspection system. Though the labeling task is not completely eliminated in this sense, the number of images that must be labeled is reduced by several orders of magnitude (tens or hundreds as opposed to hundreds of thousands or millions).

Specifically, the goal of this thesis is to evaluate the following four conjectures:

1. Given a large set of unlabeled images of a machine part, is it possible to automatically cluster them into groups that represent the range of variability of appearance?

2. How does the number of clusters vary depending on the total variability of appearance?

3. Can the clustering algorithm separate defects into one or more clusters that are distinct from the clusters modeling the range of variability of appearance?

4. Can the range of variability of appearance be characterized as Gaussian? Where do defects lie within the distribution?

Chapter 2

Methods

2.1 SEHA Plant

The data used for this thesis was gathered from the Samsung Electronics Home Appliance America (SEHA) plant in Newberry, SC (Figure 2.1). This 450,000 square foot plant, which employs over 700 people, began production in January 2018 [18]. The plant currently produces front-load and top-load washing machines. The plant's output is approximately 1,500 washing machines per day.

Figure 2.1: Aerial view of the SEHA plant in Newberry, SC [18]. This is the data collection site for all data used in this thesis.

The SEHA plant has two main assembly lines, one for top-load washing machines and another for front-load washing machines. All of the data collected for this thesis was collected on the front-load line (see Figure 2.2). In reference to Figure 2.2, washing machines move down the line from right to left. On the far right side of this view, the tub is placed into the frame. As the machine proceeds down the line, workers install components that further connect the tub to the frame. For example, many hoses are installed that lead from components on the frame to the tub. In the center of the view in the figure, inspectors perform an 11-point inspection that verifies the proper assembly of these connecting components. As the machine moves into the left portion of the view, final inspections and tests are performed before the machine comes off the line and is ready to ship.

Figure 2.2: Overhead panoramic view of front-load washing machine assembly line at SEHA plant.

Before the data collection process began, the assembly line was studied and a handful of candidate collection (camera) locations were identified. These candidate locations were selected based on the number of inspectable parts visible from the position and the quality of the view of those parts. The quality of the view of a given part was determined primarily by the amount of occlusion and the view angle. Clearly, less occlusion was preferred to more. The preferred view angle was not the same for every part. For example, the ideal view angle for an electrical connector would be a head-on view, whereas the ideal view angle for a bolt would be a side view. These two views most facilitate the visual inspection of each respective part. It would be difficult (if not impossible), for example, to inspect a bolt from a head-on view. Each candidate location was fixed with a camera and recorded for 30 minutes. Each video was analyzed and and ranked based on the number of components that reasonably met the aforementioned occlusion and view criteria. The

highest ranking location was chosen as the final data collection location for this thesis (henceforth, the collection site).

The collection site, pictured in Figure 2.3, immediately follows a late-stage manual inspection station. The washing machine is nearly completely built at this point; in fact, final inspections begin only a couple of stations ahead. The collection camera was mounted to a metal crossbar that was installed to the tops of the safety walls (yellow bars) on either side of the assembly line. This positioning allowed the camera to capture the washing machines at a straight-on angle with the washing machine passing through the middle of the camera frame (as opposed to the angled view provided by mounting the camera on top of one of the safety walls, for example); see Figure 2.3. The camera used for data collection is the Basler acA2500-14gc [19], which is an industrial area scan camera built for computer vision applications. This camera records a 2590x1942 pixel frame at a maximum of 14 frames per second.

Figure 2.3: Camera installation at the collection site (left) and an example camera frame captured from this position (right). When the assembly line is in motion, the washing machines enter the frame from the bottom, move upwards, and exit the frame at the top.

2.2 Inspection Problems

The manual visual inspection performed immediately before the collection site is an 11-point inspection. From these 11 inspection problems, three were chosen for analysis in this thesis; that is, images of these problems will be run through the clustering algorithm, which is described in detail in Section 2.5. In terms of the goals of this thesis, the resulting clusters from these three problems

will aid in the evaluation of the first conjecture presented in Section 1.3. These three problems are highlighted in the camera frame shown in Figure 2.4.

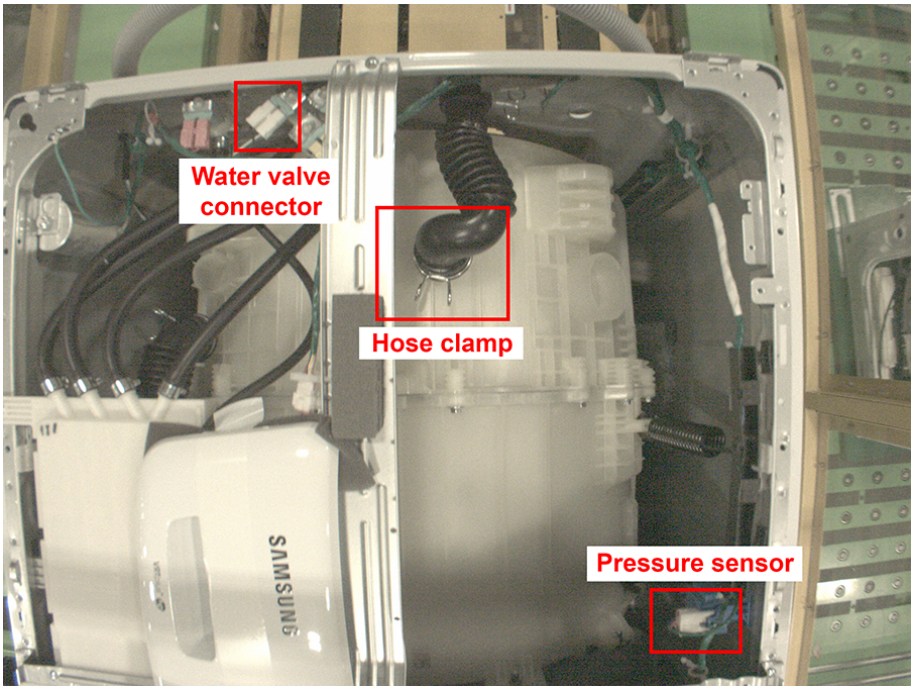


Figure 2.4: The three inspection problems chosen for analysis in this thesis.

2.2.1 Water Valve Connector

The first problem is an electrical connector that powers a water valve (Figure 2.5). To be properly installed, the connector must be fully inserted so that it is flush against the mount (green piece). This problem was chosen due to its low appearance variability. The goal of analyzing this part is to determine whether the proposed method can detect a defect, thus aiding in the evaluation of the third conjecture. Given this problem's low appearance variability, the method should be able to filter out any defects in the dataset and assign them to their own clusters.

2.2.2 Hose Clamp

The second problem is a hose clamp (see Figure 2.6). To be properly installed, the hose clamp but be secure and fully seated on the hose. This problem was chosen due to its obvious natural appearance variability. That is, this problem exhibits several acceptable appearances due to

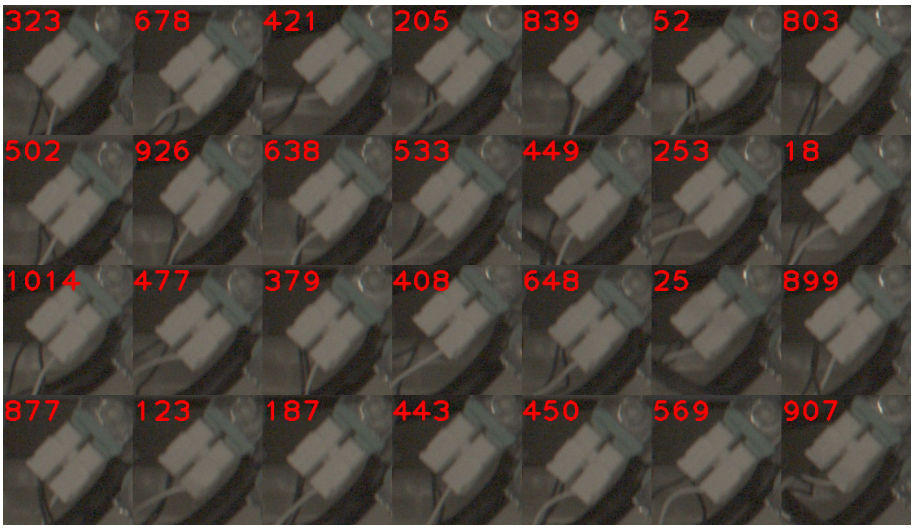


Figure 2.5: Collage of water valve connectors from 28 washing machines. Notice how the connector has low appearance variability. The numbers in each image are simply identifiers and are irrelevant in this section.

the angle of the clamp, which is unrelated to whether the product is defective or not. The goal of analyzing this problem is to determine how many clusters are needed to model the natural variability in this part.

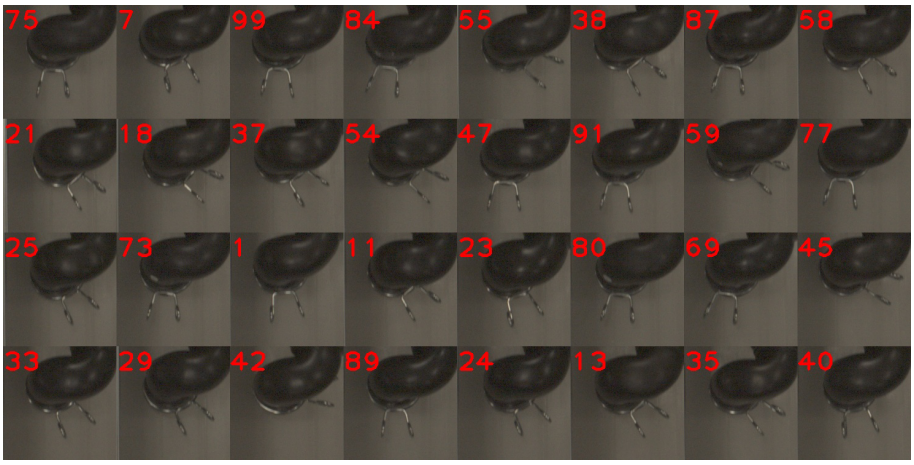


Figure 2.6: Collage of hose clamps from 32 washing machines. Notice how the angle of the clamp varies from machine to machine. The numbers in each image are simply identifiers and are irrelevant in this section.

2.2.3 Pressure Sensor

The third problem is a pressure sensor (see Figure 2.7), which is housed inside a white connector. To be properly installed, both prongs of the connector must be fully snapped into the mount (blue piece). This problem was chosen due to its high natural appearance variability. In this case, the variability is caused by the wire passing in front of the connector (dark green line). Similar to the hose clamp, this wire does not make the part defective, it simply adds to the number of possible appearances of the part. Compared to the hose clamp, the pressure sensor has much higher appearance variability[1]. In the case of the clamp, the clamp is free to rotate about a single axis. In the case of the pressure sensor, since the wire is unconstrained, it is able to rotate about any axis, bend at any point, and cross in front of the connector at any angle. The goal of analyzing this part is to determine whether the proposed method is capable of modeling such high appearance variability.

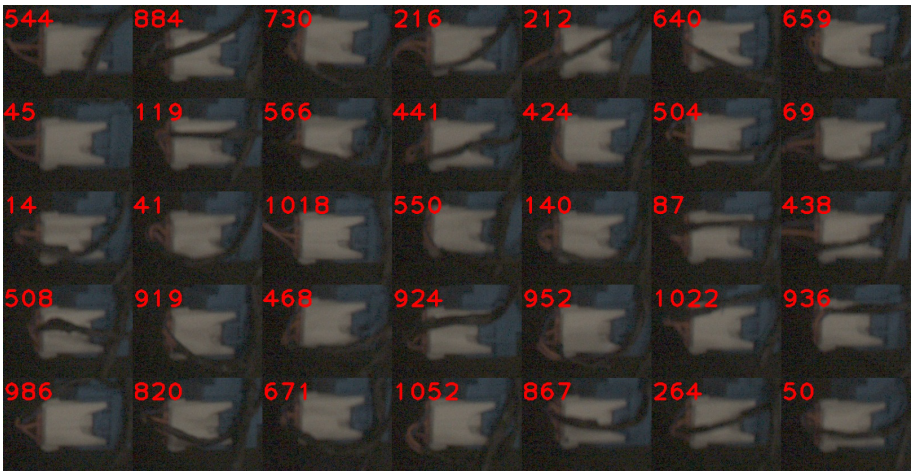


Figure 2.7: Collage of pressure sensors from 35 washing machines. Notice how the sensor has high appearance variability due to the wire passing in front of it. The numbers in each image are simply identifiers and are irrelevant in this section.

[1]This statement is quantified in Section 2.6.

2.3 Data Collection

2.3.1 Method

A data collection program[2], `S`, was written to manage the camera and save data (see Appendix A). Rather than recording and saving continuous video, it was decided that still images of each washing machine would be captured and stored. In order to programmatically determine when a machine was in front of the camera, and thus when a picture should be taken, trigger matching (see Section 1.1.3) was used.

For the data collection program, the chosen trigger was a corner of the washing machine frame (see Figure 1.7). The trigger threshold used in this program was 0.72. For each frame grabbed from the camera, the trigger template was matched to the trigger search window to determine whether or not a machine was present. If the trigger was found in a given frame, that frame was saved. It should be noted that, in this case, the trigger search window was sufficiently long enough and the camera's frame rate was sufficiently fast enough to capture multiple frames per washing machine. In fact, about 10 frames were collected per machine with each subsequent frame showing the washing machine slightly higher in the frame.

2.3.2 Collection Summary

Using the data collection program described in the previous section, data was collected at the collection site for about three months. A summary of the collected data is shown in Table 2.1.

Dataset	No. of Machines	Start Date	End Date	Notes
Pilot	3,437	12/6/2018	12/10/2018	First collection test.
Low Gain	19,634	1/11/2019	1/29/2019	Largest dataset. After collection, it was determined that the camera gain was too low (the images were dark).
High Gain	3,378	2/11/2019	2/12/2019	Adjusted gain and exposure time to increase image brightness.
New Model	4,520	2/12/2019	2/21/2019	New washing machine model. Production slow (about 1 machine/10 minutes).

Table 2.1: Data collection summary.

[2]The author acknowledges and thanks Surya Sharma as the lead software developer for this data collection program. The source code and a user manual are available at `http://cecas.clemson.edu/~ahoover/samsung/`.

2.3.3 Inspection Problem Cropping

Once images of washing machines were captured, the inspection windows of the the three specific inspection problems of interest had to be cropped from these frames. To accomplish this task, a cropping program, `cropIW`, was developed (see Appendix A).

For each frame captured from the given dataset on the given date, the program starts by running a trigger match on the frame. The match returns the location of the trigger in the frame. This location may or may not be the same for each frame as the washing machine does not always appear in the same exact place. The inspection window is placed at a constant horizontal and vertical offset from the trigger match location. This inspection window is the portion of the frame that is cropped and saved. It is assumed that that component of interest is somewhere within this window. The cropping process is identical to the first part of the inspection phase of the inspection process (see Section 1.1.3). Figure 2.8 demonstrates the cropping process.

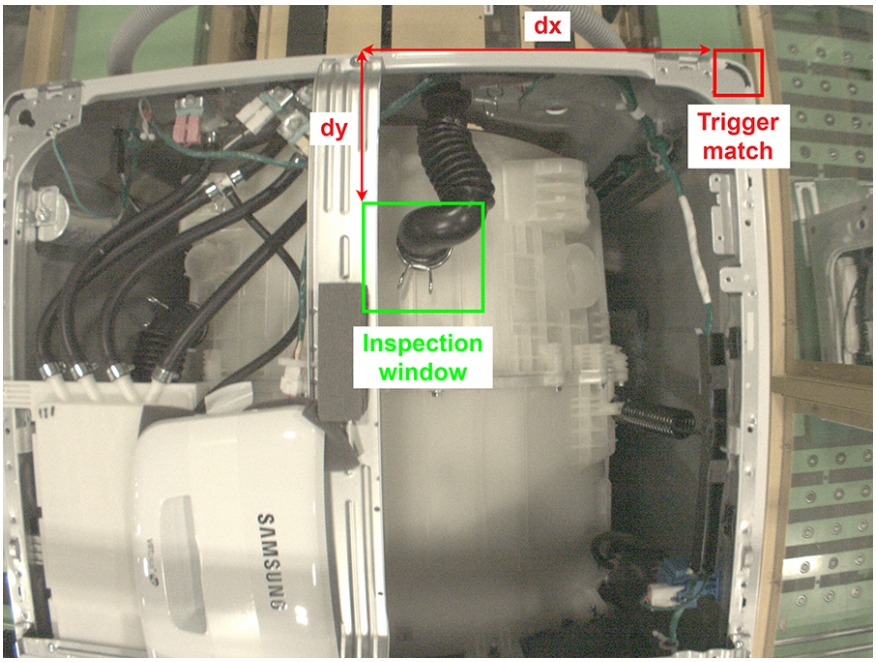


Figure 2.8: An example of the cropping procedure on the hose clamp. The inspection window is placed at a constant horizontal offset (dx) and constant vertical offset (dy) from the trigger match location. The inspection window is cut and saved.

To test the clustering algorithm, the three inspection problems of interest were cropped from the frames captured on December 6^{th}, which is part of the Pilot dataset, for a total of 1,052 windows per problem.

2.4 Theory of Modeling Variability

This section describes the theory behind appearance variability and the assumptions made therein. First, it is assumed that the distribution of the acceptable appearances of a given problem (as determined by the NCC score of each window against an "average" appearance of the problem) is roughly Gaussian. That is, most acceptable parts look similar to each other. This is represented by the high point in the Gaussian curve. Second, it is assumed that the vast majority of parts are not defects, thus the average appearance should be an acceptable appearance. Third, it is assumed that unacceptable appearances (defects) appear as small blips far below the mean of the Gaussian as defects should not match the average appearance well at all. These assumptions are displayed graphically in Figure 2.9.

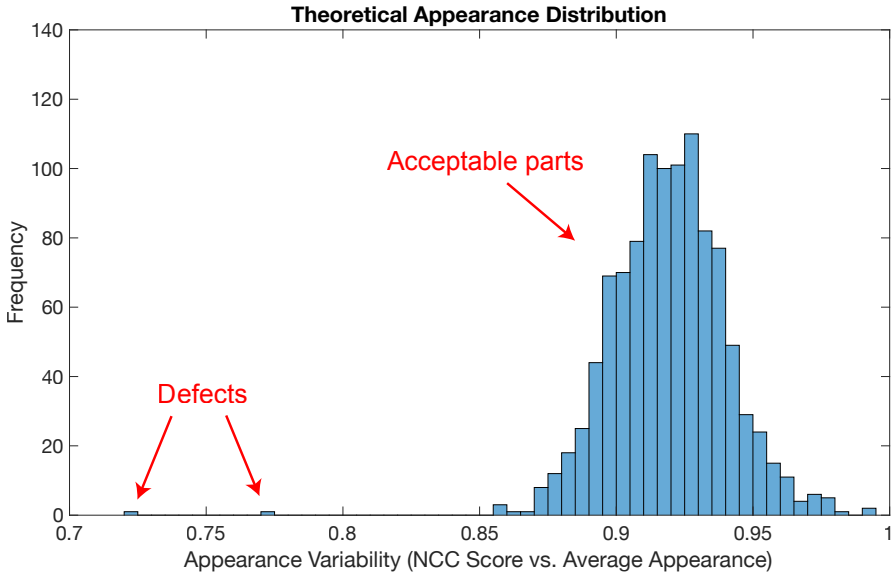


Figure 2.9: Theoretical model of appearance variability. In this model, defects are less numerous than acceptable appearances and score low against the average appearance.

Under these circumstances, defects like those highlighted in the figure should be easy to detect. The more difficult problem is if any defects lie within the score range of acceptable parts, which corresponds to roughly 0.55-0.80 in the figure. In this difficult case, the defects tend to blend in with the acceptable appearances, and are thus much more difficult to filter out using the NCC score against the average appearance. The clustering algorithm will test this theory to see whether defects tend to lie far from or well within the acceptable appearance distribution.

2.5 Clustering Algorithm

2.5.1 Overview

The goal of the clustering algorithm is to separate the set of input inspection windows into multiple clusters based on appearance. That is, after execution, each cluster should contain windows that look like each other. The basic idea behind the algorithm is to start by assuming each inspection window belongs to a single default cluster. Each cluster, including the default, has a special window called the seed, which represents the appearance of the cluster as a whole and is used to compare the appearance of other windows against the appearance of the cluster. Once the default cluster is created, the algorithm finds that cluster's worst match, which is the window within the cluster that has the least similar appearance to the seed. Next, a new cluster is created and seeded with the worst match in the default cluster (see Figures 2.10-2.13).

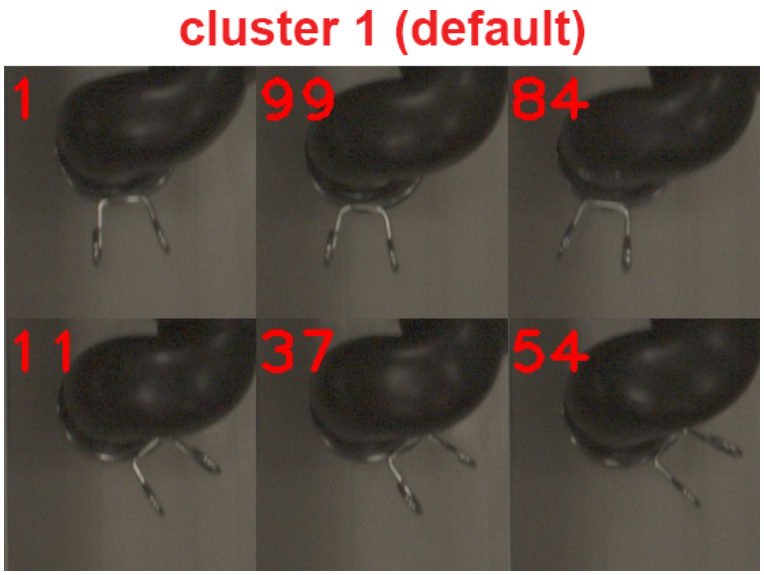


Figure 2.10: First, the default cluster (cluster 1) is populated with all available inspection windows. In this case, there are six total windows.

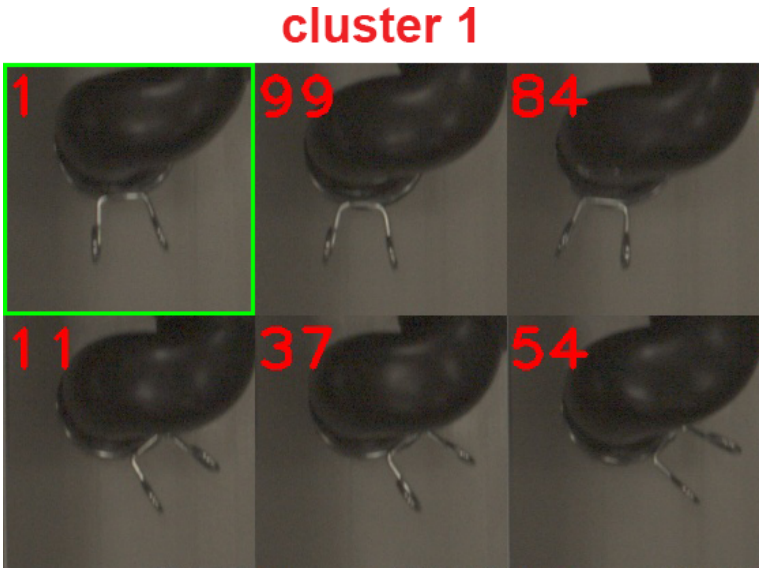


Figure 2.11: Second, the seed of the default cluster is assigned. The seed is outlined in green. In this case, the seed is set to window 1.

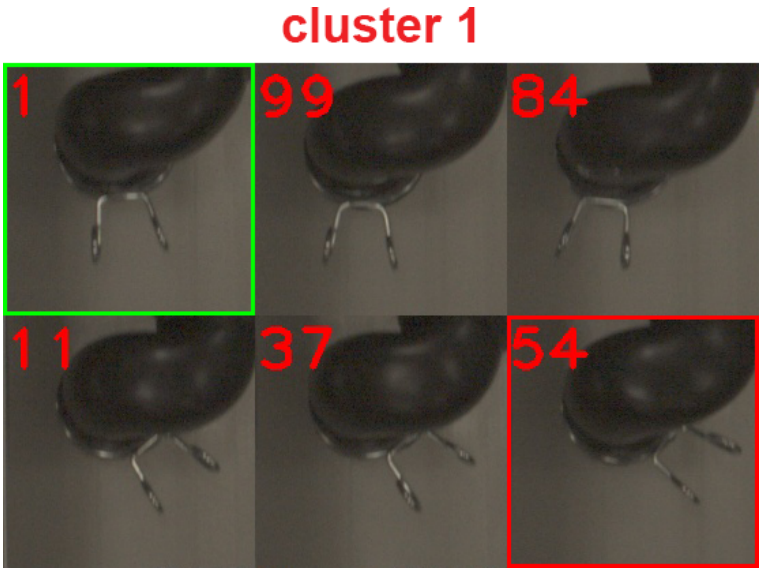


Figure 2.12: Third, the worst match is found. The worst match is outlined in red. In this case, the worst match is window 54.

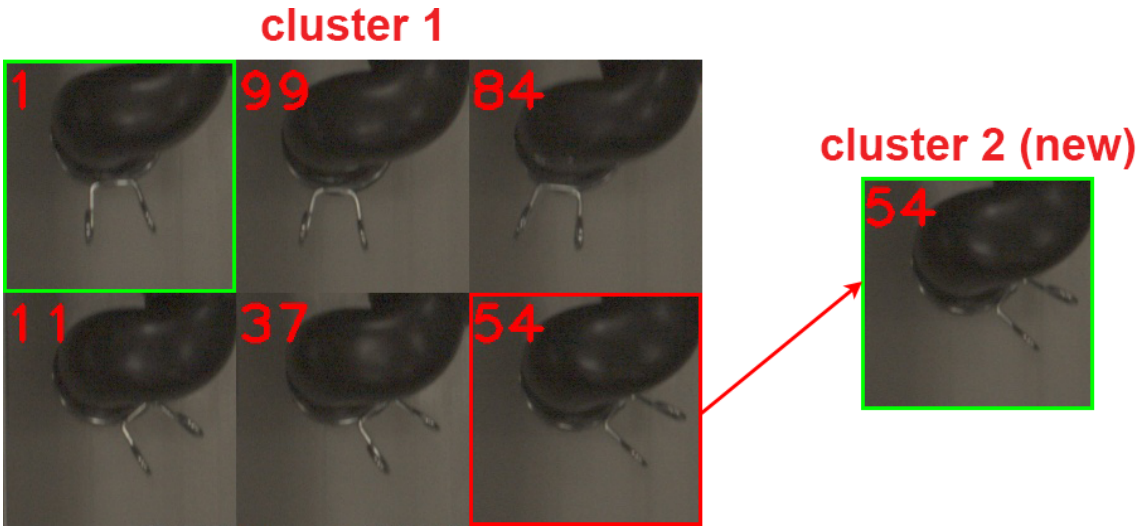


Figure 2.13: Fourth, a new cluster (cluster 2) is seeded with the worst match from the default cluster.

After seeding the new cluster, the algorithm then compares each non-seed window to the seeds of the default cluster and the new cluster. Each window is assigned to the cluster whose seed it matches best (see Figure 2.14).

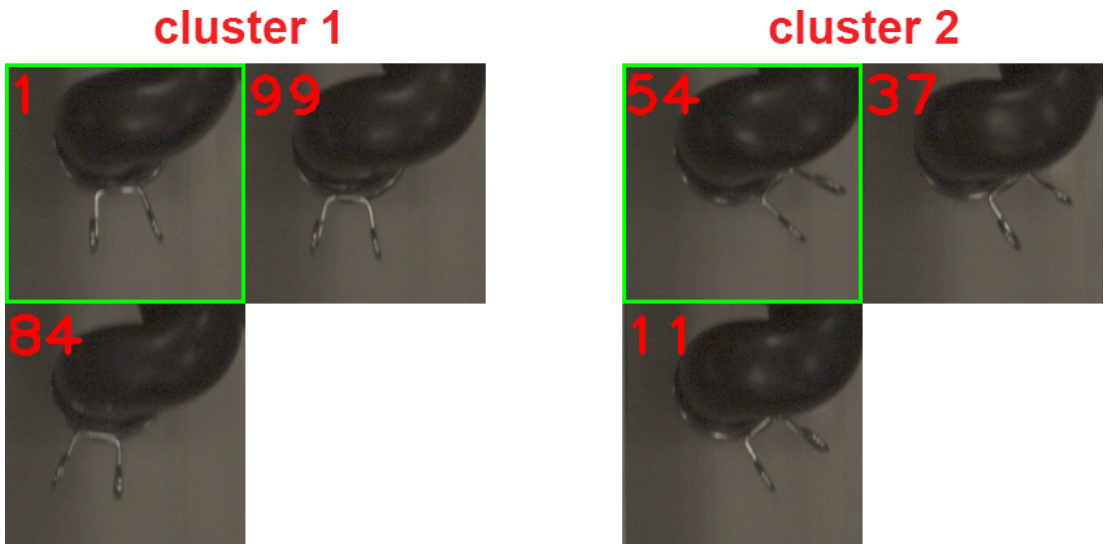


Figure 2.14: After seeding the new cluster, the algorithm reassigns all non-seed windows. In this case, both clusters end up with three members. Notice how similar-looking windows are members of the same cluster.

After reassigning the windows, a variability score is computed for each cluster. A higher variability score indicates higher appearance variability among the members of that cluster. The worst match in the cluster with the higher score is used to seed a new cluster (see Figure 2.15), the windows are reassigned, and the process repeats. This is the general idea behind the clustering algorithm; the algorithm is explained in detail in the section below.

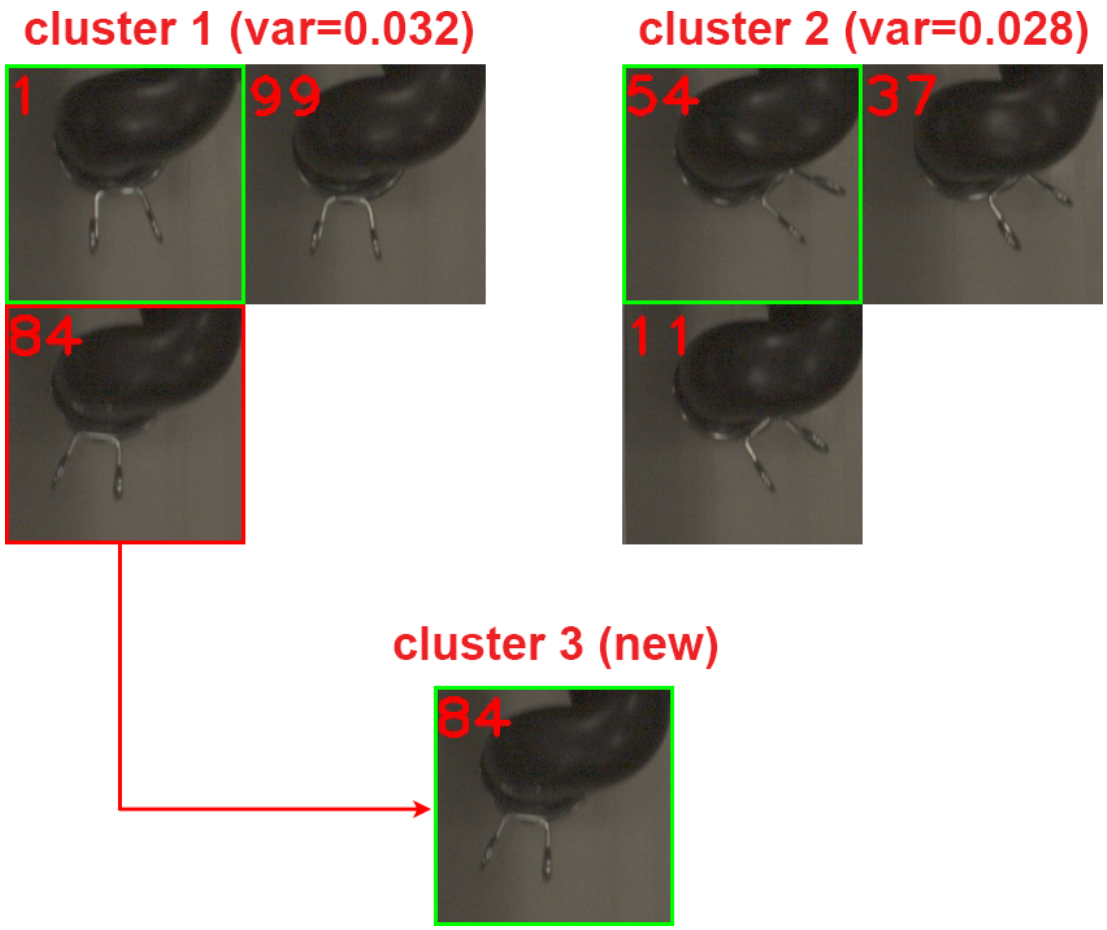


Figure 2.15: In subsequent iterations, the algorithm computes an appearance variability score (var) for each cluster. A new cluster is seeded with the worst match in the cluster with the highest variability score. In this case, cluster 1 has the highest variability, so a new cluster (cluster 3) is seeded with the worst match in cluster 1, window 84. After seeding, the windows will be reassigned and the process will repeat.

2.5.2 Formal Description

Before explaining the clustering algorithm in detail, the concept of a cluster must be formalized. A cluster is defined by the following attributes: its id, its seed, and its members. The id is an integer that uniquely defines the identity of the cluster; no two clusters have the same id. The seed is a special member of the cluster that represents the appearance of the entire cluster. Finally, the members are the windows other than the seed that belong to the cluster. Each member has an associated match location and score (discussed below).

Furthermore, when comparing appearances, this algorithm does not compare entire windows to each other. Rather, the seed of each cluster has an associated inspection template that captures the portion of the window that matters (similar to the inspection phase of the inspection process described in Section 1.1.3). There are two main reasons for this strategy. First, for most parts, it is not likely that the entire window contributes to whether the part is properly installed or not. For example, when inspecting a connector, it is unnecessary to look at the entire connector and the surrounding area. The only area that actually needs attention is the area where the end of the connector meets its mount. The use of an inspection template allows the unimportant areas to be ignored and not contribute to the overall NCC score. Second, the use of an inspection template allows the area of interest to appear at any location within the window. A part may not always be located at the exact same offset from the trigger, thus the position of the area of interest may shift slightly from window to window. The use of an inspection template prevents these slight changes in location from affecting the NCC score.

The clustering algorithm begins by loading the inspection windows of interest from file. Each window is smoothed with a 7x7 Gaussian kernel to reduce noise. After smoothing, the gradient images of each window are computed. Next, the default cluster, `DC`, which is assigned id 1, is created. By convention, the seed of this cluster is set to the window with id 1. For `DC`, the inspection template of the seed, `ST`, is manually cropped and saved before the algorithm executes. Once the seed is set, the gradient image of `ST` is computed. Next, the cluster is "built." The building process occurs as follows: For each remaining inspection window, `IW`, the steps below are completed.

1. Match `ST` to `IW`. The location of this match is the location associated with `IW` in this cluster.
2. Crop an inspection template, `IT`, from `IW` whose top-left corner is the location of the seed match and whose size is the same as `ST`.

3. Compute the gradient image of `IT`.

4. Take the NCC of the gradient image of `ST` and the gradient image of `IT`. The score of this match is the score associated with `IW` in this cluster.

The location of the part is determined by the original images while the score of the part is determined by the gradient images for the same reasons laid out in Section 1.1.3. Once the default cluster is built, it is added to the list of clusters (it is the only cluster in the list thus far).

After the default cluster is built, the algorithm enters an iterative stage that creates one new cluster per iteration. During each iteration, the following steps are taken:

1. Compute the appearance variability score of each cluster, which is the standard deviation of the member's scores (not including the score of the seed with itself).

2. Find the cluster with the highest appearance variability, `HC`, and seed a new cluster, `NC`, with the lowest scoring member in `HC`. `ST` in `NC` is set to the `IT` that was cropped for this `IW` during the last (re)building stage.

3. Add `NC` to the cluster list.

4. Rebuild all clusters.

The rebuilding process is similar to the process of building the default cluster. During rebuilding, each cluster starts with only one member—its seed. For each inspection window that is not a seed, that window is matched against the `ST` of each cluster using the same location and scoring schemes as the initial building phase. The window is then added as a member to the cluster whose `ST` it matches best.

The iterative stage terminates when either the cluster with the maximum appearance variability falls below the maximum cluster variability threshold, `MCVT`, or the maximum number of clusters is reached. The program developed to run this algorithm, `cluster2`, takes these two numbers as command line arguments (see Appendix A). Once the iterative stage is complete, the program writes the details of each cluster to file so they can be analyzed.

2.5.3 Psuedocode

Psuedocode for the entire algorithm is given below.

```
// create default cluster
cluster_t DC
DC.id = 1
DC.seed = 1
DC.members = {1}

// build default cluster
for each inspection window IW:
    loc = match(DC.ST, IW)
    IT = crop(IW, {loc.x,loc.y,DC.ST.width,DC.ST.height})
    IT_edges = sobel(IT)
    score = match(DC.ST_edges, IT_edges)
    DC.addMember(IW)

// add default cluster to cluster list
list clusters
clusters.add(DC)

// cluster loop
loop:
    // check for maximum number of clusters
    if maximum number of clusters reached:
        break

    // find cluster with highest appearance variability
    max = 0
    for each cluster C:
        if standardDeviation(C.scores) > max:
            max = C.scores
            HC = C

    // check for acceptable maximum cluster variability
    if max < MCVT:
        break

    // find lowest scoring member in HC
    min = 1.0
    for each member M in HC:
        if M.score < min:
            min = M.score
            minMember = M

    // create new cluster
    cluster_t NC
    NC.id = clusters.getLast().id + 1
    NC.seed = minMember
    clusters.add(NC)

    // clear all clusters
    for each cluster C:
        C.removeAllMembers()
```

```
    // rebuild all clusters
    for each inspection window IW:
        maxScore = 0.0
        for each cluster C:
            loc = match(C.ST, IW)
            IT = crop(IW, {loc.x,loc.y,C.ST.width,C.ST.height})
            IT_edges = sobel(IT)
            score = match(C.ST_edges,IT_edges)
            if score > maxScore:
                maxScore = score
                MC = C
        MC.addMember(IW)
end loop

// write results to file
for each cluster C:
    C.write()
```

2.5.4 Manual Selection of Default Cluster Seed Templates

As discussed in Section 2.5, the seed template for the default cluster must be cropped manually before the algorithm executes. The details of these templates for each inspection problem of interest are shown in Figure 2.16. Each template was chosen on the basis of being the smallest area of its respective inspection window that could be used to differentiate itself from other windows.

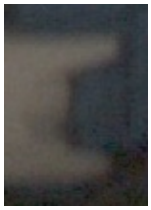
2.5.5 Manual Tuning of `MCVT`

As mentioned in Section 2.5, the clustering algorithm terminates when either the maximum number of clusters is reached or the variability score of the cluster with the highest appearance variability falls below the maximum cluster variability threshold, `MCVT`. The ideal value of `MCVT` may change from problem to problem. This is for two reasons. First, not every problem has the same amount of natural appearance variability. In general, problems with less appearance variability require a lower (more strict) value of `MCVT`. Second, the size of the inspection template for a given problem is proportional to the value of `MCVT`. In general, larger templates do not require as strict of a value of `MCVT` as smaller ones. This is because the scores computed with larger templates, as opposed to smaller ones, tend to vary more since it is less likely that all pixels of the part to be inspected will line up directly with those in the seed template.

In order to determine the ideal value of `MCVT` for any given problem, hundreds of inspection

(a) Water valve connector template (size=80x80).

(b) Hose clamp template (size=332x172).



(c) Pressure sensor template (size=75x105).

Figure 2.16: Manually cropped default cluster seed templates for the three inspection problems of interest.

problems would have to analyzed and a function would have to be fit to the inherent variability and template size variables. The fitting of this function is out of the scope of this thesis. In order to determine the values of `MCVT` to use for each of the three problems of interest in this thesis, a manual tuning process was performed.

The manual, brute force tuning process was completed as follows: For each inspection problem, the clustering algorithm was run without the `MCVT` stopping condition. That is, the algorithm was terminated based on the number of clusters created rather than the variability score of the cluster with the highest appearance variability. The algorithm was run repeatedly with each run allowing one more cluster than the previous. The initial run allowed only a single cluster. After each run, the resulting clusters were visually inspected. The repeated runs were stopped once, by visual inspection, each cluster contained no within-cluster visual dissimilarities. That is, the mem-

bers of each cluster looked sufficiently like the other members of that cluster[3]. The number of runs required meet this condition (thus, the number of clusters required for that problem) was noted. After determining the number of required clusters, the clustering algorithm results were analyzed to determine which value of `MCVT` would have produced the same number of clusters. The results of the manual tuning process for each problem are shown in Table 2.2.

Problem	No. of Clusters	`MCVT`
Water valve connector	8	0.0150
Hose clamp	22	0.0550
Pressure sensor	37	0.0232

Table 2.2: Manual tuning of `MCVT` for each problem of interest.

The values of `MCVT` presented in the table are the values of `MCVT` used by the clustering algorithm to produce the final clustering results for each problem. These results are discussed in the next chapter.

2.6 Quantifying Appearance Variability

In Section 2.2, it is mentioned that the goal of analyzing each respective part is to see how many clusters are generated by the clustering algorithm given the specific attributes of each part. This is idea is somewhat related to the second conjecture presented in Section 1.3; however, in order to directly evaluate this conjecture, the appearance variability of each problem must be quantified.

The variability of a given problem was quantified by computing the average intensity difference per pixel across all templates, μ. This value was computed by first generating a difference accumulation image (DAI) for each problem. The DAI keeps track of the sum of the differences between subsequent templates. For example, to compute the DAI for a given problem, first, template 1 is subtracted from template 0, and the results are added into the DAI. Next, template 2 is subtracted from template 1, and the results are added into the DAI. This process continues for all templates. Once the DAI is computed, the normalized DAI (NDAI) is computed by dividing each number in the DAI by the total number of templates. The average value in the NDAI is equivalent to μ, which is the quantification of the variability of the given problem. The resulting μ values for

[3]The author acknowledges that this process is subjective; however, it is necessary to ground truth the value of `MCVT` used for the three inspection problems of interest.

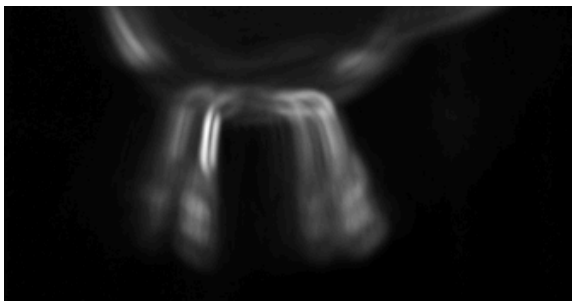
the three problems of interest are given in Table 2.3.

Problem	μ	Rank
Water valve connector	4.96	Low
Hose clamp	5.73	Medium
Pressure sensor	8.54	High

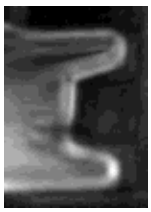
Table 2.3: Quantification of appearance variability for each problem of interest. Each problem is assigned a relative rank.

Visualizations of the NDAI of each problem of interest are shown in Figure 2.17. Qualitative analysis of these images agree with the quantitative measures computed above. The variability of each problem is proportional to the percentage of pixels in the NDAI visualization that are bright (near white). By inspection, it is clear that bright pixels take up the largest proportion of the pressure sensor's NDAI, the smallest proportion of the water valve connector's NDAI, and the hose clamp's NDAI lies somewhere in the middle.

(a) Water valve connector.



(b) Hose clamp.



(c) Pressure sensor.

Figure 2.17: Visualization of the NDAIs of the three inspection problems of interest.

2.7 Cluster Analysis

In order to facilitate the analysis of the clusters produced by the clustering algorithm, a visualization program, `clusterViewer`, was developed (see Appendix A). This program displays the members of each cluster and lists the attributes of each cluster (see Figure 2.18).

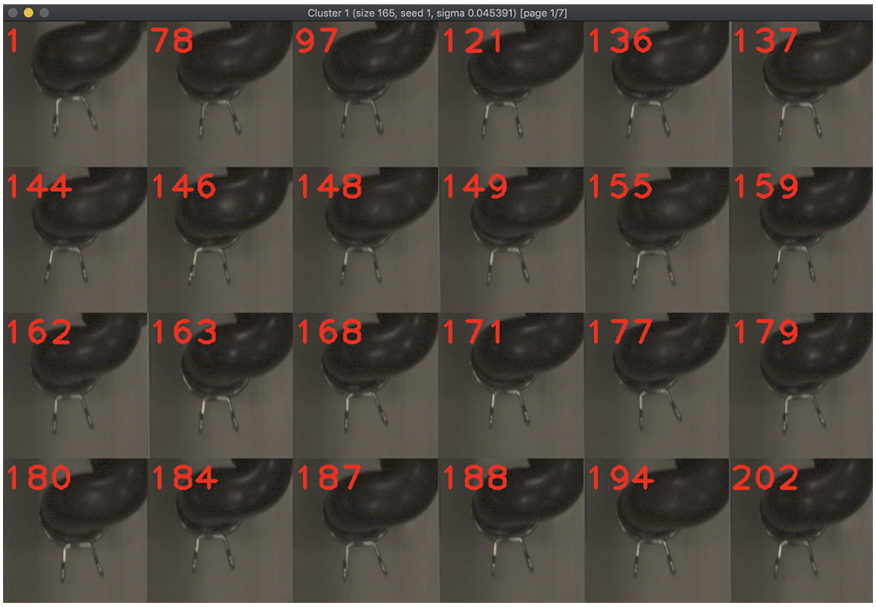


Figure 2.18: Screenshot of `clusterViewer`. The current view is showing the first 24 windows in cluster 1 (page 1 of 7). As seen in the title bar, the program is reporting the cluster's attributes: its seed is 1, its size is 165, and its variability score (sigma) is 0.045.

In addition to the features described above, this program also allows the user to view the windows in several configurations. First, by pressing 'E,' the gradient images of the smoothed windows are displayed in place of the original windows (see Figure 2.19). This is because the clustering program smooths each inspection window with a 7x7 Gaussian kernel before clustering in order to minimize noise. The original windows are not shown as smoothed because they would be harder to analyze with the human eye, whereas the gradient images are shown as smoothed so that, given scoring is based on gradient images, the images being scored by the clustering algorithm can be viewed directly. Second, by pressing 'L,' the seed match location and score are overlaid on top of each window (see Figure 2.20). When this feature is activated, the area outlined in red represents the inspection template of that window. The template location can differ from window to window, as seen by the different locations of the red rectangles in the figure. The gradient image and score/location features can be toggled independently of each other.

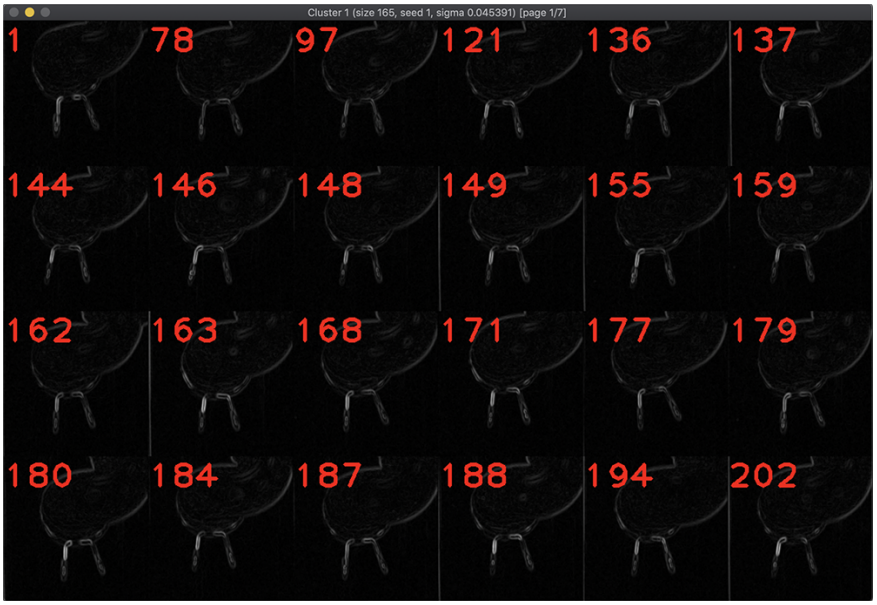


Figure 2.19: Screenshot of `clusterViewer`. Edge mode is toggled on.

Figure 2.20: Screenshot of `clusterViewer`. Location and score mode is toggled on. The rectangle on each window corresponds to the match location of the seed template; that is, each rectangle outlines the inspection template associated with that window.

Chapter 3

Results

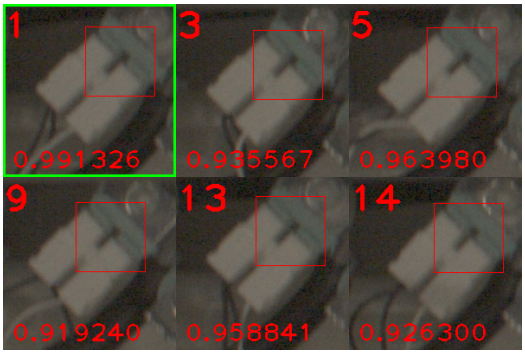
3.1 Clustering Algorithm Output

This section presents the resulting clusters produced by the clustering algorithm for each of the three inspection problems of interest. In this section, the entire inspection window of each cluster member is shown. As discussed in Section 2.5, the clustering algorithm only accounts for each member's inspection template, the sub-area of the inspection window that is important. Therefore, for each window shown, its template is outlined in red. In other words, when comparing different members for similarities and differences, only the portion of the windows within the red rectangles are relevant. In addition, although only a sample of the 1,052 clustered windows for each problem are shown, the author visually verified that—across all 1,052 windows for each problem—no cluster contained significant within-cluster visual dissimilarities.

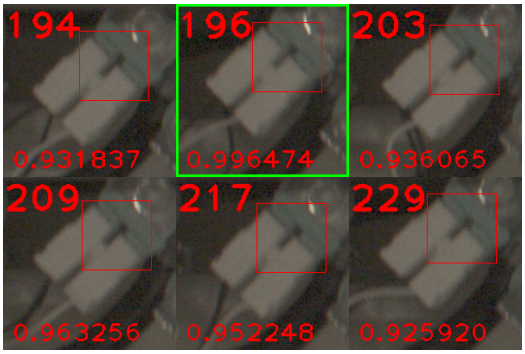
3.1.1 Water Valve Connector

The clustering algorithm produced eight clusters for this problem. For illustration, Figure 3.1 shows the eight clusters. For each cluster with more than six members, the seed plus five other random members are displayed. For clusters with six members or fewer, the entire cluster is shown.

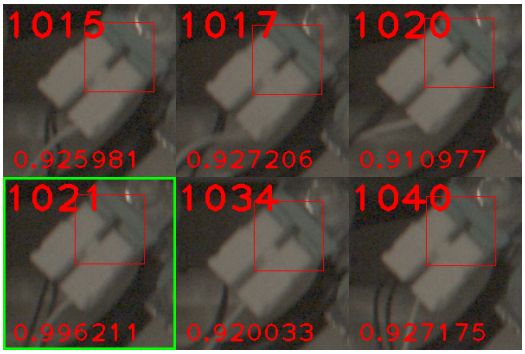
For this problem, the clustering algorithm was able to generate a defect cluster; that is, a cluster whose seed is a defect. It is clear that cluster 5 is a defect cluster since, in window 988 (the seed), the connector is not completely flush against the mount.



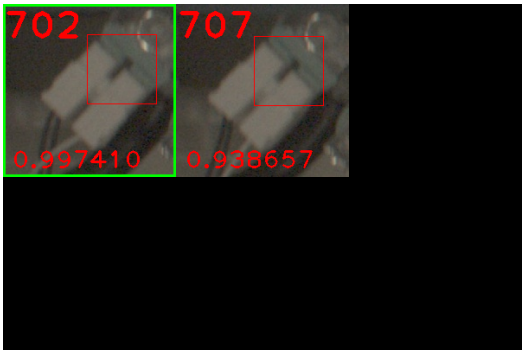
(a) Cluster 1 (size=469, var=0.014)



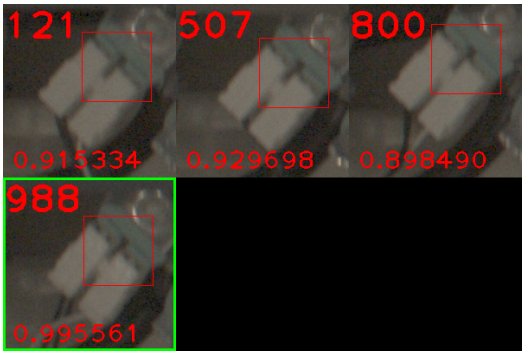
(b) Cluster 2 (size=55, var=0.014)



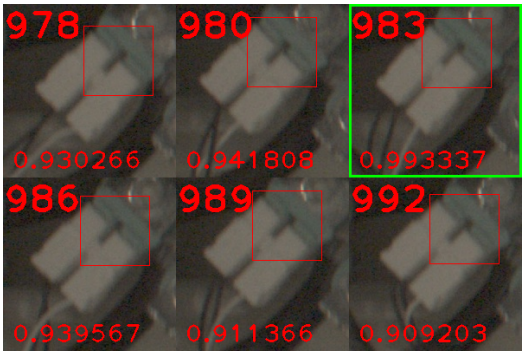
(c) Cluster 3 (size=180, var=0.013)



(d) Cluster 4 (size=2, var=0.000)

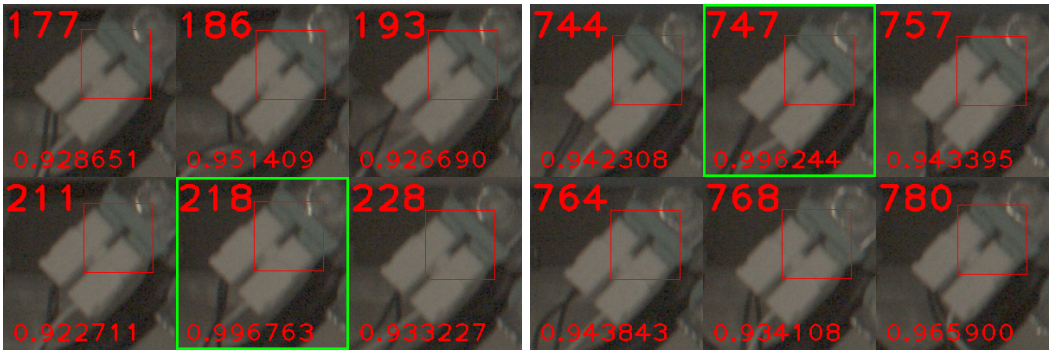


(e) Cluster 5 (size=4, var=0.013)



(f) Cluster 6 (size=130, var=0.014)

Figure 3.1: The eight water valve connector clusters produced by the clustering algorithm (continued on next page). Each seed is outlined in green. For each window, the template is outlined in red and the score of that template against the seed template is shown in the lower portion of the window.

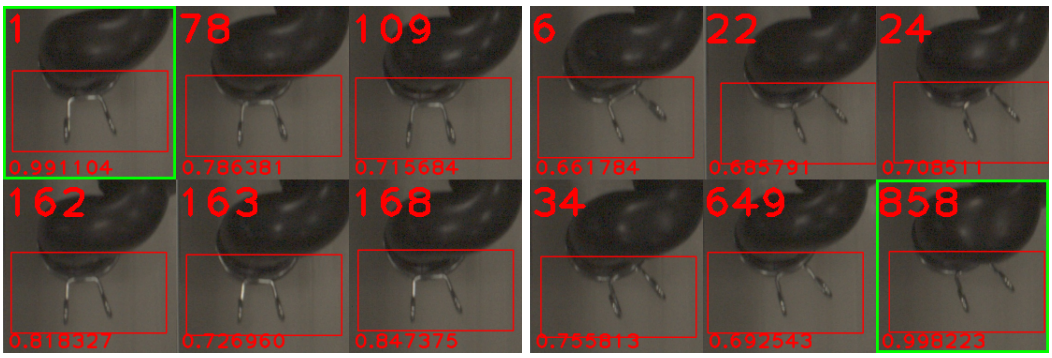


(g) Cluster 7 (size=64, var=0.012) (h) Cluster 8 (size=148, var=0.014)

Figure 3.1: The eight water valve connector clusters produced by the clustering algorithm (continued from previous page). Each seed is outlined in green. For each window, the template is outlined in red and the score of that template against the seed template is shown in the lower portion of the window.

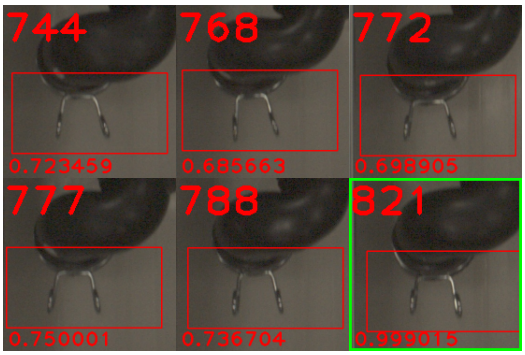
3.1.2 Hose Clamp

The clustering algorithm produced 22 clusters for this problem. For illustration, Figure 3.2 shows eight of the 22 clusters. For each cluster shown with more than six members, the seed plus five other random members are displayed. For clusters with six members or fewer, the entire cluster is shown. For details on all 22 clusters produced for this problem, see Appendix B.

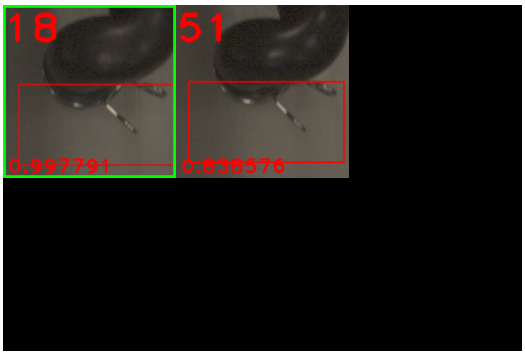


(a) Cluster 1 (size=120, var=0.051) (b) Cluster 4 (size=7, var=0.030)

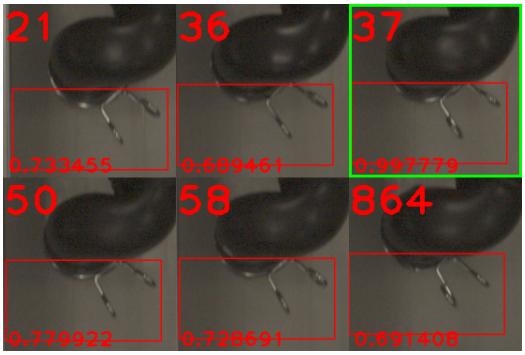
Figure 3.2: Eight sample hose clamp clusters produced by the clustering algorithm (continued on next page). Each seed is outlined in green. For each window, the template is outlined in red and the score of that template against the seed template is shown in the lower portion of the window.



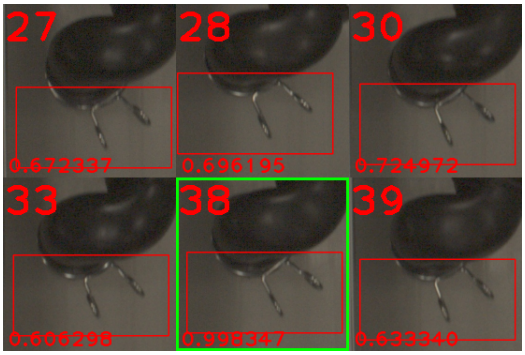
(c) Cluster 5 (size=60, var=0.043)



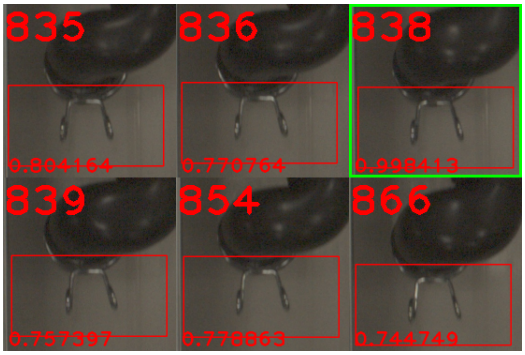
(d) Cluster 6 (size=2, var=0.000)



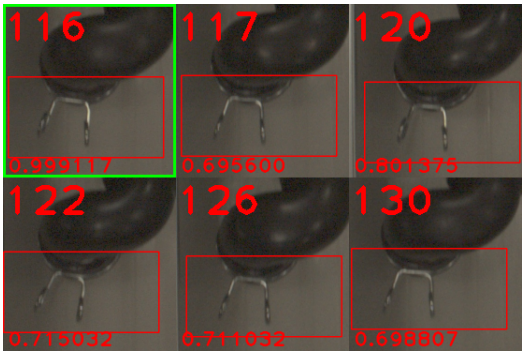
(e) Cluster 9 (size=12, var=0.044)



(f) Cluster 11 (size=17, var=0.049)



(g) Cluster 15 (size=115, var=0.039)

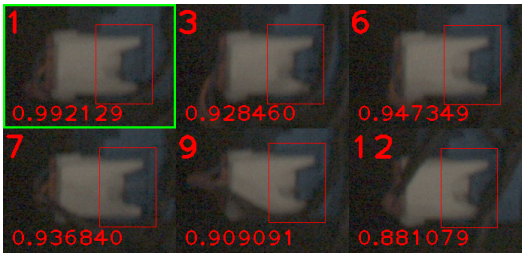


(h) Cluster 17 (size=25, var=0.042)

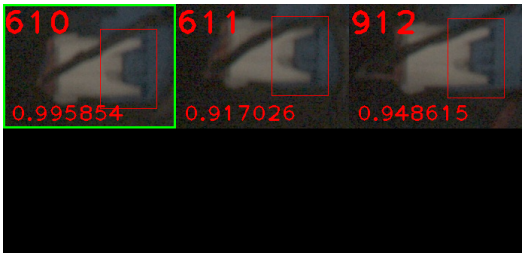
Figure 3.2: Eight sample hose clamp clusters produced by the clustering algorithm (continued from previous page). Each seed is outlined in green. For each window, the template is outlined in red and the score of that template against the seed template is shown in the lower portion of the window.

3.1.3 Pressure Sensor

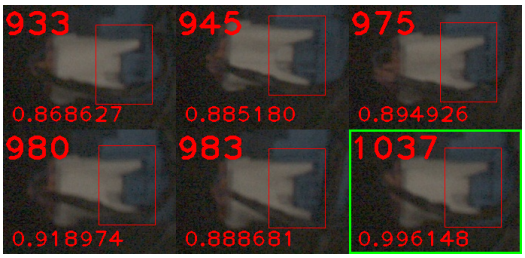
The clustering algorithm produced 37 clusters for this problem. For illustration, Figure 3.3 shows 10 of the 37 clusters. For each cluster shown with more than six members, the seed plus five other random members are displayed. For clusters with six members or fewer, the entire cluster is shown. For details on all 37 clusters produced for this problem, see Appendix B.



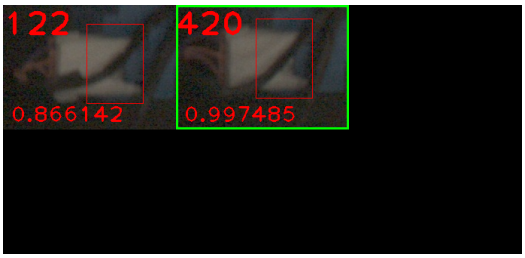
(a) Cluster 1 (size=363, var=0.023)



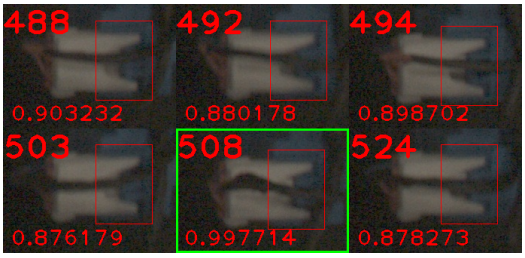
(b) Cluster 4 (size=3, var=0.016)



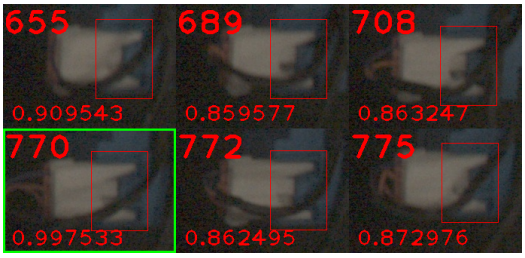
(c) Cluster 5 (size=36, var=0.015)



(d) Cluster 8 (size=2, var=0.000)



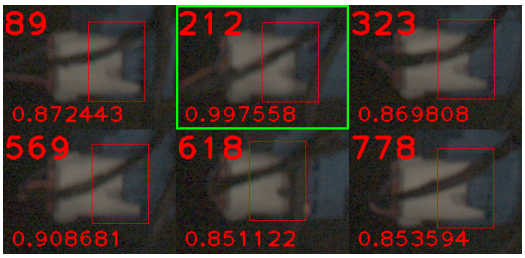
(e) Cluster 13 (size=32, var=0.021)



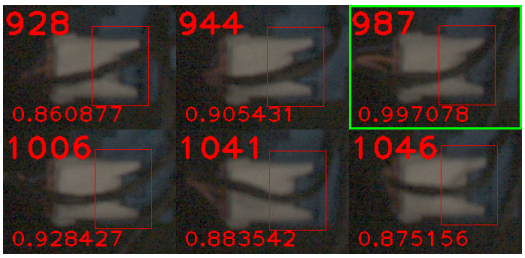
(f) Cluster 19 (size=33, var=0.022)

Figure 3.3: Ten sample pressure sensor clusters produced by the clustering algorithm (continued on next page). Each seed is outlined in green. For each window, the template is outlined in red and the score of that template against the seed template is shown in the lower portion of the window.

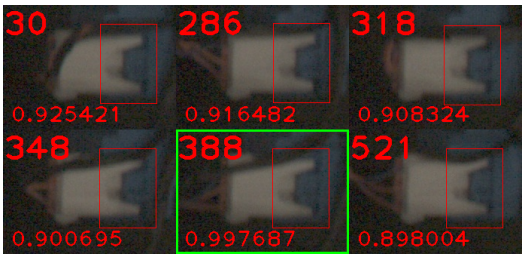
(g) Cluster 22 (size=13, var=0.015)



(h) Cluster 27 (size=7, var=0.020)



(i) Cluster 28 (size=24, var=0.023)



(j) Cluster 31 (size=15, var=0.018)

Figure 3.3: Ten sample pressure sensor clusters produced by the clustering algorithm (continued from previous page). Each seed is outlined in green. For each window, the template is outlined in red and the score of that template against the seed template is shown in the lower portion of the window.

3.2 Evaluation of Conjectures

3.2.1 Conjecture 1: Viability of the Algorithm

It is clear from the results shown in the previous section that, given an inspection problem, the clustering algorithm is capable of grouping the problem's set of inspection windows into clusters that sufficiently capture the range of appearance variability. As the appearance variability of the problem increased[1], the resulting clusters had a higher degree of within-cluster visual dissimilarity. For example, the members of the pressure sensor clusters look slightly less like each other than those of the hose clamp clusters. Likewise, the members of the hose clamp clusters look slightly less like each other than those of the water valve connector clusters. This result is due to the fact that problems with higher variability are more difficult to cluster. Even so, the degree of the observed within-cluster visual dissimilarity for all three of these problems—even the pressure sensor—is still acceptable.

[1]As discussed in Section 2.3, the relative variability levels of the water valve connector, hose clamp, and pressure sensor are low, medium, and high, respectively.

3.2.2 Conjecture 2: Degree of Variability vs. Number of Clusters

The evaluation of this conjecture is straightforward when comparing the relative degrees of variability of the three inspection problems against the number of clusters produced for each problem. This comparison is shown in Table 3.1.

Problem	Variability Rank	No. of Clusters
Water valve connector	Low	8
Hose clamp	Medium	22
Pressure sensor	High	37

Table 3.1: Degree of variability and number of clusters produced.

Clearly, the higher the appearance variability of a given problem, the more clusters that are required to sufficiently capture the range of appearance. In other words, problems with relatively low variability reach sufficiently low levels of variability with a fewer number of clusters than those with relatively high variability. This effect can be observed when the average μ value[2] across all clusters is plotted against the number of clusters. This curve is plotted for all three inspection problems of interest in Figure 3.4.

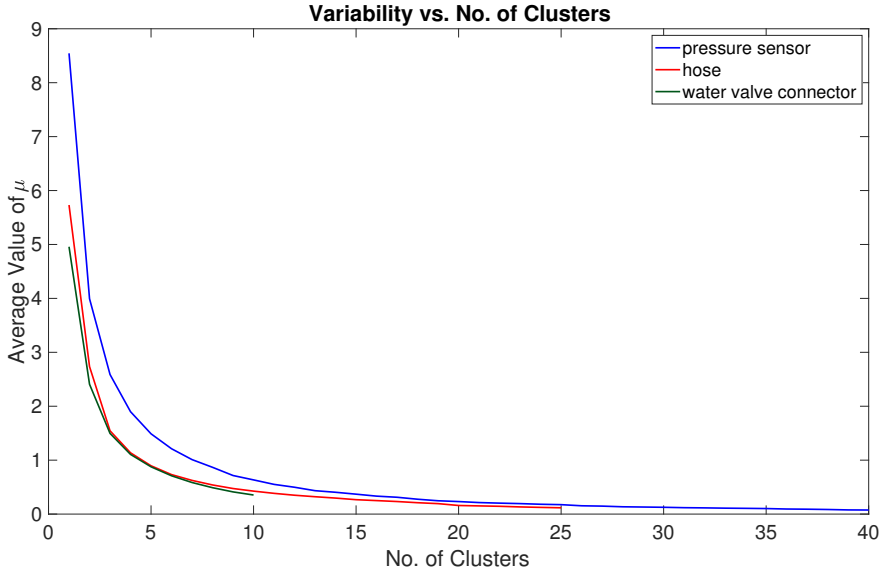


Figure 3.4: Average cluster variability vs. number of clusters. Problems with lower initial (single cluster) variability reach sufficiently low levels of variability in fewer clusters than those with higher initial variability.

[2]As discussed in Section 2.3, this value, which is the average value of a cluster's NDAI, is a quantification of the appearance variability of that cluster.

3.2.3 Conjecture 3: Ability to Detect Defects

Given the results for the water valve connector problem, it is clear that the clustering algorithm is able to detect defects. This fact is apparent when directly comparing the seed windows of clusters 1 and 2 (non-defect clusters) against the seed window of cluster 5 (a defect cluster); see Figure 3.5. The fact that the clustering algorithm was able to detect a defect is particularly interesting given the images collected for this thesis were taken immediately after a human-performed inspection that checked for the proper installation of this part (among 10 others).

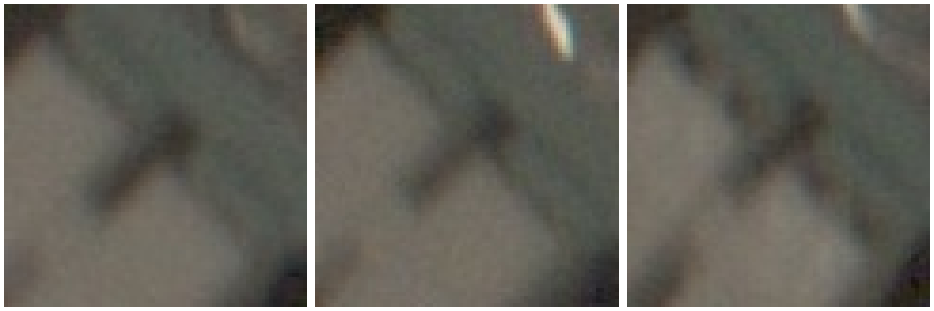


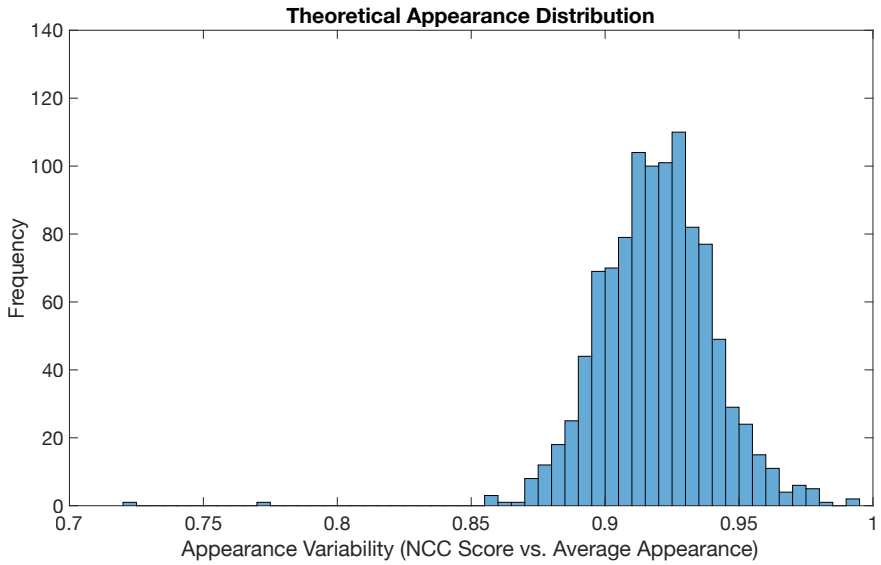
Figure 3.5: Seed template of non-defect clusters 1 and 2 (left and middle, respectively) vs. seed template of defect cluster 5 (right). It is clear that the part shown on the right is not properly installed since the connector is not flush against the mount.

3.2.4 Conjecture 4: Accuracy of Appearance Distribution Theory

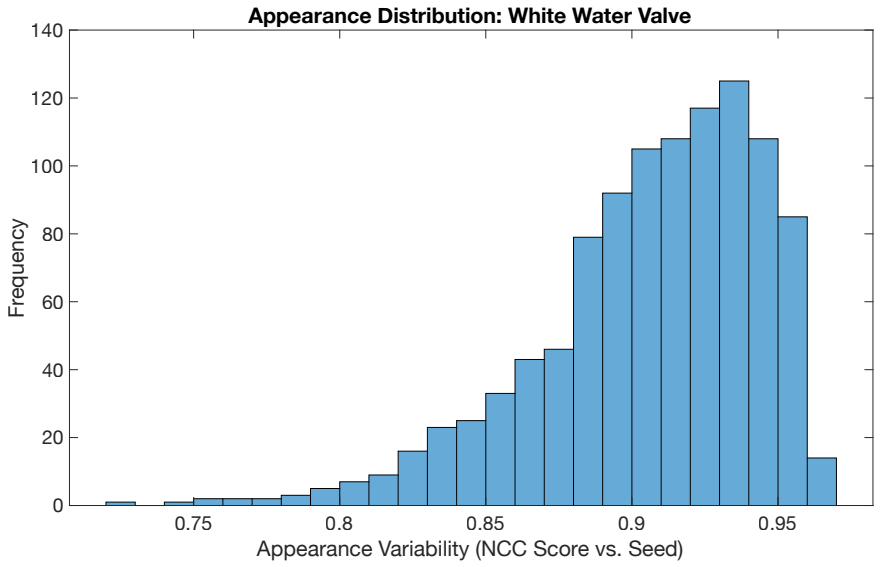
In order to evaluate this conjecture, the theoretical appearance distribution (discussed in Section 2.4) is compared to the actual appearance distribution of the water valve connector problem (see Figure 3.6). The actual distribution of the water valve connector problem is computed by comparing the seed template of the default cluster (template 1) against all other templates.

Upon analysis, it is clear that these two distributions are not similar. Specifically, the actual distribution has positive skew and contains no obvious outliers. This suggests that, in reality, there is not one single acceptable appearance that dominates the distribution. Rather, there exist multiple acceptable appearances that score similarly—but not quite as high as—the most common acceptable appearance, which in this case scores around 0.93 against the default cluster's seed. In order to gain more insight into this effect, it is useful to analyze the distribution of cluster sizes for all three problems (see Figure 3.7). It is clear from this distribution that the vast majority of clusters are relatively small. Given that only one of these small clusters is a defect (specifically,

cluster 5 from the water valve connector problem), this distribution indicates that there exist several infrequent examples of acceptable appearances that do not look similar to each other in any given problem. Thus, the appearance distribution theory does not predict the shape of actual appearance distributions accurately.



(a) Theoretical appearance distribution (originally presented in Section 2.4).



(b) Actual appearance distribution of the water valve connector problem.

Figure 3.6: Theoretical vs. actual appearance distribution. The actual distribution has positive skew and contains no obvious outliers.

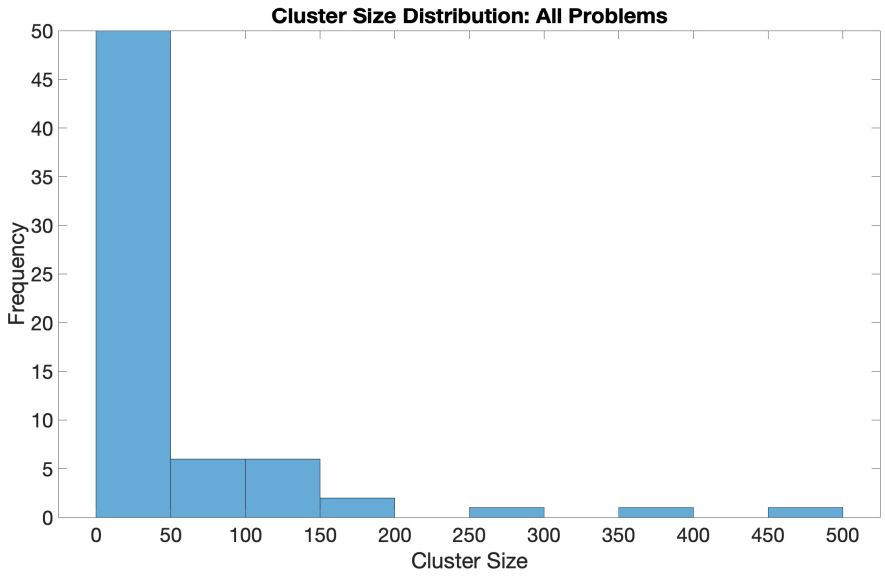


Figure 3.7: Distribution of cluster sizes for the clusters of all three inspection problems (67 clusters total). The vast majority of clusters are relatively small, indicating that there exists much more than a single acceptable appearance for any given problem.

In addition to shape, the distribution theory also predicts that defects are low-scoring outliers against the average appearance. Given that the actual distribution of the water valve connector problem contains no outliers, this prediction does not hold. In fact, the defects (the four members of the defect cluster) lie much closer to the mean than anticipated (see Figure 3.8). The practical significance of this finding is that defects, at least for this problem, are particularly difficult to spot because they look very similar to properly installed parts. This could explain why four out of the 1,052 (about 0.4%) were missed by the operator of the manual inspection station immediately preceding the collection site.

Given the appearance distribution theory did not accurately predict the general shape of the appearance distribution nor the location of defects, it should be revised to account for multiple acceptable appearances (most of which occur infrequently) and defects that lie within those acceptable appearances.

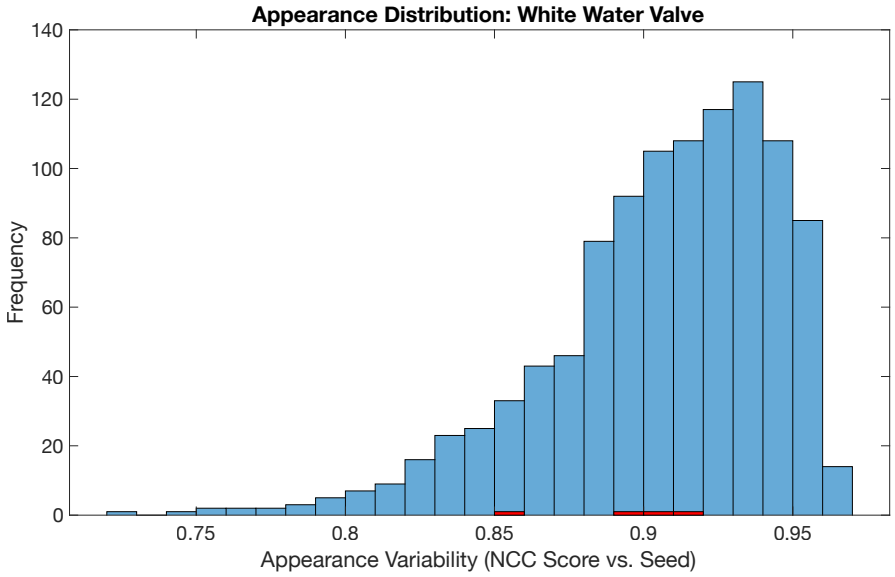


Figure 3.8: Actual appearance distribution of the water valve connector problem with defect scores highlighted in red. The defects tend to "blend in" with the acceptable appearances.

Chapter 4

Conclusion

4.1 General Discussion

Overall, the clustering algorithm proposed by this thesis performed reasonably well. For three different inspection problems of varying appearance variability, the algorithm was able to separate the 1,052 examples of each problem into viable clusters.

Problems with a higher degree of appearance variability tended to require more clusters in order to sufficiently capture the problem's full range of appearance. This is evident in the fact that the clustering algorithm produced 8, 22, and 37 clusters for the water valve connector (low variability), hose clamp (medium variability), and pressure sensor problems (high variability), respectively. It is expected that this pattern would continue for future inspection problems.

In addition to producing viable clusters, the clustering algorithm was also able to identify a defect by producing a defect cluster (namely, cluster 5 for the water valve connector problem). Given the collection site immediately follows a manual inspection station, the fact that a defect was found at all is surprising initially. However, upon analysis of the appearance distribution of defects for the water valve connector problem (see Figure 3.8), it is clear that defects are difficult to spot because they look very similar to acceptable appearances. This explains why about 0.4% of the water valve connector samples were defects. This finding also has implications on the difficultly of the clustering problem in general (in terms of being able to conduct reliable inspections with the results). Since defects lie within the range of acceptable appearances, it will take a larger number of clusters to filter those defects out of the non-defect clusters. In order to perform reliable inspections,

as many defect clusters as possible should be produced to ensure no unacceptable appearances are overlooked.

The appearance distribution theory developed in Section 2.4 did not hold for two reasons. First, the theory inaccurately predicted the shape of the appearance distribution. This is because the theory predicted a single acceptable appearance, resulting in a Gaussian distribution. In reality, there are many acceptable appearances, most of which are infrequent, thus the actual shape of the distribution is a Gaussian with significant positive skew. Second, the theory inaccurately predicted the location of defects within the appearance distribution. The theory predicted that defects would score the lowest; however, in reality, the scores of defects tended to blend in with those of acceptable appearances.

4.2 Limitations

One limitation of this study is the number of example images used for each inspection problem. As mentioned in Chapter 1, many problems require at least an order of magnitude greater than 1,000 test images to sufficiently capture the problem's range of appearance. Therefore, the 1,052 samples used in this study may not be sufficient to capture the entire range of appearance exhibited by the three problems of interest. This does not mean that the clustering algorithm itself is insufficient; rather, this means that, in order to perform a reliable inspection with the clusters produced by this algorithm, a larger dataset is advised. As presented in Section 2.3.2, a dataset of approximately 19,000 images was collected from the SEHA plant. This dataset was not processed by the clustering algorithm due to time constraints. The data organization, cleaning, and cropping processes require substantial amounts of time to complete for a dataset of such size.

Another inherent limitation of the data used in this study is the low illumination caused by improper collection camera settings. For all images analyzed in this thesis, the camera settings were not ideal, resulting in relatively dark images. These settings were adjusted for later datasets. Dark images tend to have gradient images with less prominent edges than bright ones. Therefore, brighter images would result in more accurate template comparisons, ultimately resulting in better clustering behavior.

Finally, it is not advisable to run an actual inspection system with the results produced in this thesis for two reasons. First, as mentioned above, the dataset size is probably not large enough

to sufficiently capture the range of appearances of these parts. This is especially true for the hose clamp and pressure sensor problems, the two problems with higher appearance variability. Second, the angles at which the parts were captured may not be ideal for inspections. This is especially true for the hose clamp problem. The top-down view of the hose clamp used here would not be an ideal angle at which to judge whether or not the clamp is fully seated, and, by extension, whether or not it is properly installed. The top-down view is appropriate, however, for observing the different positions of the clamp. Thus, given the purpose of this thesis is to group images based on differences in appearance, the top-down view is ideal for testing the clustering algorithm, but it is not necessarily ideal for performing reliable inspections.

4.3 Future Work

The most obvious next step for this work is to run the clustering algorithm on a dataset an order of magnitude larger than the size 1,052 one analyzed in this thesis. It would also be recommended to ensure the collected images have proper lighting to enhance the results of the algorithm.

Besides better data, there are also several improvements that could be made to the core algorithm to increase performance. First, the seed of the default cluster should be picked more intelligently than simply assigning it to the first window. Ideally, the seed of the default cluster would be set to the template that best represents the average appearance of all windows in the dataset. This window could be found by cross-comparing every window to every other window and determining which window scores highly (above some threshold) against the most other windows.

Second, more inspection problems should be analyzed so that a function can be fit in order to automatically determine the value of `MCVT`. The algorithm becomes much more powerful if this value can be chosen automatically so that the number of clusters does not have to be determined brute force like it was in this thesis. As mentioned in Section 2.5.5, the value of `MCVT` depends on the natural variability of the problem and the chosen template size. After brute forcing hundreds of inspection problems, a function could be fit to the variability and template size variables to reliably predict the value of `MCVT` for new problems.

Third, the algorithm should compare templates more intelligently. As presented in this work, when comparing two windows, the algorithm compares the entire template of one window to

the entire template of the other. This is an issue for problems whose ideal inspection area does not take up the entire template. For some inspection problems, the solution to this issue is to simply reduce the size of the template. For others, the solution is not so simple. For example, consider the hose clamp problem. For any given template, the clamp itself only takes up a small proportion of the whole template; however, the template size cannot reduced or else other possible appearances of the clamp would be clipped. In other words, the large template size is need to fully capture any possible appearance that the clamp could exhibit. In order to solve this problem, the algorithm should be able to learn which parts of the template are important (that is, those that should contribute to the NCC score) and those that are not. This could be accomplished by implementing a dynamic mask that only takes the important parts of each template into account.

Appendices

Appendix A Software Details

This appendix details the four pieces of software that were developed to support this thesis. All software is written in C++ and utilizes the OpenCV library. An overview of the programs is given in Table A.1.

Program	Description
`S`	Collects data at the SEHA plant.
`cropIW`	Crops inspection windows for a specific inspection problem from the frames captured by `S`.
`cluster2`	Clusters the inspection windows generated by `cropIW`.
`clusterViewer`	Visualizes the clusters produced by `cluster2`.

Table A.1: Software overview.

A.1 File Structure

In order to fully understand the the inputs and outputs of the programs described in this appendix [1], it is necessary to understand the file structure used by these programs. The file structure is organized as follows:

```
~/                                  // base
    data/
        pilot/                      // frames collected from pilot dataset
            dec5/
            dec6/
            ...                     // other dates
        ...                         // other datasets
        IWs/                        // inspection windows
            pilot/
                hose/
                    IW__1.jpg
                    IW__2.jpg
                    ...             // other inspection windows
                    count.txt
                    filelist.txt
                ...                 // other parts
            ...                     // other datasets
    programs/
        cropIW/
            configurations/
                pilot/
```

[1] The information in this section applies to all programs except `S`, which was developed on a separate operating system.

```
            hose/
                config.ini
                tt.jpg
            ...             // other parts
        ...                 // other datasets
    Makefile
    config.h                // class that handles configurations
    config.cpp
    cropIW.cpp              // main file for crop program
cluster2/
    ITs/                    // inspection templates
        pilot/
            hose/
                IT_1.jpg    // manually cut template
                ...         // automatically generated templates
            ...             // other parts
        ...                 // other datasets
    Makefile
    cluster.h               // class that defines a cluster
    cluster.cpp
    cluster2.cpp            // main file for cluster2 program
    clusterViewer.cpp       // main file for clusterViewer program
```

As shown above, each source code directory contains a `Makefile` that will compile the program in that directory. In the `cluster2/` directory, the `Makefile` must be manually changed to either compile `cluster2` or `clusterViewer`. These programs share a directory since they operate on similar data.

A.2 S

This program is a GUI-based Windows application that was developed in Visual Studio on Windows 10. The purpose of this program is twofold. First, the program collects data. Second, the program displays the details of the inspection process as it runs live. This is accomplished via a GUI that displays the live camera view and overlays rectangles over the trigger window, trigger match location, inspection window, and part match location when a washing machine passes the camera. The GUI-related operations of this program were required by a separate project, and are not directly relevant to this thesis.

A.2.1 Inputs

As this program is a desktop application, it does not take any command line arguments. This program does rely, however, on a configuration file that details the size and locations of the

trigger and inspection windows, the paths of the trigger and inspection templates, and other minor settings. As most of the options in this file deal with the GUI-related operations, and thus are not relevant to this thesis, the format of this file will not be discussed in detail.

A.2.2 Outputs

All frames captured in a single day are stored in the same directory. For example, all frames collected on January 26, 2019 are stored in `1-26-19/`. Each frame is named according to the date, time, and machine number. For example, a frame captured at 11:27:08.0256 on January 26, 2019 from the 5^{th} machine to pass the camera that day would be named `2019-01-26__11_27_08_0256__5.jpg`. Different frames captured from the same machine are differentiated by the time stamp, which is precise to 1/10 of a millisecond.

A.3 `cropIW`

This program is a Unix-based executable developed in Visual Studio Code on macOS Mojave. The purpose of this program is to crop inspection windows from the frames captured by `S`. An example usage of this program is given by

```
./cropIW pilot dec6 hose
```

which crops the hose clamp from all frames captured on December 6^{th}, which is part of the pilot dataset.

A.3.1 Inputs

The three command line arguments taken by this program are the dataset, the date, and the part. As discussed in section 2.3.2, the collected frames are organized by dataset, which is a set of frames grouped by common properties.

Several additional details needed by this program are are stored in a configuration file, "config.ini," which is located at `~/programs/crop/configurations/{DATASET}/{PART}/config.ini`. There is no reason other than cleanliness (keeping the number of command line arguments to a minimum) that these inputs are read from file. The contents and format of the configuration file are defined in Table A.2.

Line No.	Symbol	Description
1	TW_X	Trigger window x-coordinate (top left corner).
2	TW_Y	Trigger window y-coordinate (top left corner).
3	TW_W	Trigger window width.
4	TW_H	Trigger window height.
5	IW_DX	X-offset from trigger match location (top left corner) to inspection window x-coordinate (left edge).
6	IW_DY	Y-offset from trigger match location (top left corner) to inspection window y-coordinate (top edge).
7	IW_W	Inspection window width.
8	IW_H	Inspection window height.
9	FRAME	Frame number to crop (1-indexed). Multiple frames of each machine may be captured. Only one frame (specifically, the one with the most ideal view of the part) should be cropped.

Table A.2: config.ini contents and format.

An example configuration file is given below.

```
1640
350
400
1500
-375
225
200
200
5
```

This file indicates the following: The trigger window is located at $(1640, 350)$ (top-left corner) and has size 400x1500. The top-left corner of the inspection window is located 375 pixels to the left of and 225 pixels down from the trigger match location. The size of the inspection window is 200x200. The inspection window is cropped from the 5^{th} frame captured of this washing machine.

The final input needed by this program is the trigger template to be used for the trigger match, "tt.jpg," which is located at ~`/programs/crop/configurations/{DATASET}/{PART}/tt.jpg`.

A.3.2 Outputs

Each cropped inspection window is stored in a directory unique to its dataset-part pair. For example, the hose clamp inspection windows cropped from the pilot dataset are stored in ~`/data/IWs/pilot/hose/`. Within the dataset-part directory, each window is assigned an id, which

is unique to each window. The id assigned to the first window is 1, and subsequent windows are assigned $2, 3, ..., n$, where n is the total number of windows in the directory.

In addition to the inspection windows, this program generates two extra files in the dataset-part directory, "count.txt" and "filelist.txt." The former contains a single integer indicating the total count of windows in the directory. The program reads this number during setup to determine the next window id to assign in the case the program is run for multiple dates in the same dataset. The latter generates a complete list of all windows in the directory. This file is needed by `cluster2` to read in the windows for a given dataset-part pair.

A.4 `cluster2`

This program is a Unix-based executable developed in Visual Studio Code on macOS Mojave. The "2" in the name simply denotes that this program is the successor of an earlier, unsuccessful attempt at implementing the clustering algorithm (not discussed in this thesis). As the main driver of the work in this thesis, the purpose of this program is to separate a set of inspection windows into clusters. An example usage of this program is given by

```
./cluster2 pilot hose 0.025 32
```

which executes the clustering algorithm on the inspection windows of the hose clamp cropped from the pilot dataset.

A.4.1 Inputs

The four command line arguments taken by this program are the dataset, the part, the maximum allowable variability, and the maximum allowable number of clusters.

This program reads in the available inspection windows for the given dataset-part pair by referencing the list in the "filelist.txt" document generated by `cropIW`.

A.4.2 Outputs

Immediately before termination, this program outputs the information about each resulting cluster to a cluster file, "Nclusters.txt," where N is the number of clusters generated. The format of this file is given below.

```
DATASET PART                                    // dataset and part
1 SIZE SEED VARIABILITY                         // first cluster id and stats
MEMBER_ID MATCH_LOC_X MATCH_LOC_Y SCORE         // member list and details
...
2 SIZE SEED VARIABILITY                         // second cluster
MEMBER_ID MATCH_LOC_X MATCH_LOC_Y SCORE
...
...                                             // more clusters
```

An example (abbreviated) cluster file is given below.

```
pilot hose
1 34 1 0.023    // first cluster
1 123 237 0.99
32 125 241 0.93
...
2 45 17 0.034   // second cluster
17 124 232 0.99
51 123 239 0.91
...
...             // more clusters (omitted)
```

This file indicates the following: Cluster 1 has size 34, seed 1, a variability score of 0.023, and members $1, 32, ...$ Cluster 2 has size 45, seed 17, a variability score of 0.034, and members $17, 51, ...$ The match results for each member of each cluster. For example, the seed template of cluster 1 matches window 32 at location $(125, 241)$ with a score of 0.93.

This cluster file is needed by `clusterViewer` to visualize clusters.

A.5 `clusterViewer`

This program is a Unix-based executable developed in Visual Studio Code on macOS Mojave. The purpose of this program is to visualize the clusters generated by `cluster2`. An example usage of this program is given by

```
./clusterViewer 20clusters.txt 4 6 0.5
```

which displays the clusters stored in "20clusters.txt," which is a cluster file generated by `cluster2`, in 4 rows and 6 columns.

A.5.1 Inputs

The four command line arguments taken by this program are the cluster file path, the number of rows to display per page, the number of columns to display per page, and the scale factor.

The scale factor is applied to the windows before being displayed. For example, a scale factor of 0.5 indicates that the original windows will be scaled to 50% of their original size before being displayed. This is essentially a zoom option.

This program is interactive. It continuously grabs input from the user to decide which commands to execute. The full list of commands is given in Table A.3.

Key	Command	Description
ESC	Quit	Terminates the program.
SPACE	Forward	Go to the next page of this cluster. If currently on the last page, go to the first page of the next cluster.
B	Back	Go to the previous page of this cluster. If currently on the first page, go to the last page of the previous cluster.
N	Next	Go to the first page of the next cluster.
P	Previous	Go to the first page of the previous cluster.
G	Go to	Go to the first page of the indicated cluster. For example, to go to cluster 12, type 'G,' '1,' '2,' then 'G' again. The command executes once the second 'G' is pressed.
L	Loc/Score	Toggle the seed match location and score overlay.
E	Edges	Toggle gradient images.
S	Save	Save the current page to file.

Table A.3: Listing of `clusterViewer` commands.

A.5.2 Outputs

The only output generated by this program is the visual cluster display.

Appendix B Cluster Details

This appendix lists the full details of each cluster produced by the the clustering algorithm for the three inspection problems of interest.

B.1 White Water Valve

The results for this problem are shown in Table B.1.

ID	Size	Seed	Variability Score
1	469	1	0.014
2	55	196	0.014
3	180	1021	0.013
4	2	702	0.000
5	4	988	0.013
6	130	983	0.014
7	64	218	0.012
8	148	747	0.014

Table B.1: Water valve connector cluster details.

B.2 Hose Clamp

The results for this problem are shown in Table B.2.

B.3 Pressure Sensor

The results for this problem are shown in Table B.3.

ID	Size	Seed	Variability Score
1	120	1	0.051
2	9	23	0.046
3	169	530	0.051
4	7	858	0.030
5	60	821	0.043
6	2	18	0.000
7	1	42	0.000
8	68	960	0.044
9	12	37	0.044
10	3	55	0.027
11	17	38	0.049
12	5	1046	0.040
13	5	856	0.048
14	8	976	0.037
15	115	838	0.039
16	4	32	0.049
17	25	116	0.042
18	288	653	0.054
19	11	178	0.022
20	118	412	0.041
21	2	54	0.000
22	3	44	0.014

Table B.2: Hose clamp cluster details.

ID	Size	Seed	Variability Score
1	363	1	0.023
2	12	907	0.013
3	13	746	0.020
4	3	610	0.016
5	36	1037	0.015
6	2	969	0.000
7	19	927	0.018
8	2	420	0.000
9	61	23	0.018
10	2	988	0.000
11	16	956	0.018
12	6	463	0.012
13	32	508	0.021
14	47	864	0.021
15	2	924	0.000
16	115	510	0.021
17	9	830	0.016
18	19	819	0.020
19	33	770	0.022
20	10	487	0.017
21	10	903	0.019
22	13	103	0.015
23	5	883	0.018
24	13	981	0.011
25	1	724	0.000
26	59	844	0.023
27	7	212	0.020
28	24	987	0.023
29	1	857	0.000
30	2	861	0.000
31	15	388	0.018
32	2	938	0.000
33	6	887	0.017
34	12	902	0.019
35	13	482	0.012
36	24	926	0.021
37	43	870	0.017

Table B.3: Pressure sensor cluster details.

Bibliography

[1] C. D. Wickens, J. G. Hollands, S. Banbury, and R. Parasuraman. *Engineering Psychology and Human Performance*. New York, NY: Routledge, 2016, pp. 26.

[2] W. A. Perkins. "INSPECTOR: A Computer Vision System that Learns to Inspect Parts." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-5, no. 6, pp. 584-592, 1983.

[3] C. Tsatsoulis and K.-S. Fu. "Using machine vision for the inspection of industrial assemblies." *NDT International*, vol. 19, no. 4, pp. 263-270, 1986.

[4] H. Choi, R. M. Gupta, and S. Suh. "Quality Measurement Of Template Models And Automatic Template Model Selection." *International Conference on Control, Automation and Systems*, 2012, pp. 1044-1048.

[5] Y. Fouda and K. Ragab. "An Efficient Implementation of Normalized Cross-Correlation Image Matching based on Pyramid." *International Joint Conference on Awareness Science and Technology & Ubi-Media Computing*, 2013, pp. 98-102.

[6] J. Killing, B. W. Surgenor, and C. K. Mechefske. "A machine vision system for the detection of missing fasteners on steel stampings." *The International Journal of Advanced Manufacturing Technology*, vol. 41, no. 7-8, pp. 808-819, Jun. 2008.

[7] M. S. Aksoy, O. Torkul, and I. H. Cedimoglu. "An industrial visual inspection system that uses inductive learning." *Journal of Intelligent Manufacturing*, vol. 15, no. 4, pp. 569-574, 2004.

[8] E. N. Malamas, E. G. M. Petrakis, M. Zervakis, L. Petit, and J.-D. Legat. A survey on industrial vision systems, applications and tools." *Image and Vision Computing*, vol. 21, no. 2, pp. 171-188, 2003.

[9] B. Zitov and J. Flusser. "Image registration methods: a survey." *Image and Vision Computing*, vol. 21, no. 11, pp. 977-1000, 2003.

[10] L. Chen, Y. Liang, and K. Wang. "Inspection of rail surface defect based on machine vision system." *International Conference on Information Science and Engineering*, 2010, pp. 3793-3796.

[11] E. Resendiz, J. Hart, and N. Ahuja. "Automated Visual Inspection of Railroad Tracks." *Intelligent Transportation Systems, IEEE Transactions on*, vol. 14, no. 2, pp. 751-760, 2013.

[12] C. Aytekin, Y. Rezaeitabar, S. Dogru, and I. Ulusoy. "Railway Fastener Inspection by Real-Time Machine Vision." *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 45, no. 7, pp. 1101-1107, 2015.

[13] H. Wu, X. Zhang, H. Xie, Y. Kuang, and G. Ouyang. "Classification of Solder Joint Using Feature Selection Based on Bayes and Support Vector Machine." *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 3, no. 3, pp. 516-522, 2013.

[14] L. Rusli and A. Luscher. "Fastener identification and assembly verification via machine vision." *Assembly Automation*, vol. 38, no. 1, pp. 1-9, 2018.

[15] S. Barker and D. J. Michael. "Semi-Supervised Method for Training Multiple Pattern Recognition and Registration Tool Models." May 2017.

[16] Y. Arai, Y. Tachimura, and M. Akiba. "Inspection System, Inspection Method, and Method for Manufacturing Semiconductor Device." Oct. 2007.

[17] J. Zhou, I. Lee, B. Thomas, R. Menassa, A. Farrant, and A. Sansome. "Applying spatial augmented reality to facilitate in-situ support for automotive spot welding inspection." *International Conference on Virtual Reality Continuum and Its Applications in Industry*, 2011, pp. 195-200.

[18] T. Anderson. "Samsung adds second production line at Newberry County plant." *Upstate Business Review*, Mar. 21, 2018.

[19] Basler AG. Basler ace acA2500-14gc. Available from `https://www.baslerweb.com/en/products/cameras/area-scan-cameras/ace/aca2500-14gc/`.