RECOGNITION OF INCOMPLETE OBJECTS BASED ON SYNTHESIS OF VIEWS USING A GEOMETRIC BASED LOCAL-GLOBAL GRAPHS

A dissertation submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

By

MICHAEL CHRISTOPHER ROBBELOTH
B.S., Wilmington College, 2000
M.S., Bowling Green State University, 2002
M.B.A., University of Dayton, 2011

2019
Wright State University

WRIGHT STATE UNIVERSITY

GRADUATE SCHOOL

April 30, 2019

I HEREBY RECOMMEND THAT THE DISSERTATION PREPARED UNDER MY SUPERVISION BY Michael Christopher Robbeloth ENTITLED Recognition of Incomplete Objects Based on Synthesis of Views Using a Geometric Based Local-Global Graphs BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF Doctor of Philosophy.

Nikolaos G. Bourbakis, Ph.D.
Dissertation Director

Michael Raymer, Ph.D.
Director, Computer Science
and Engineering Ph.D. Program

Barry Milligan, Ph.D.
Interim Dean of the Graduate
School

Committee on
Final Examination

Nikolaos G. Bourbakis, Ph.D.

Soon M. Chung, Ph.D.

Y. Pei, Ph.D.

Arnab K. Shaw, Ph.D. (EE Dept)

ABSTRACT

Robbeloth, Michael Christopher Ph.D., Computer Science and Engineering Ph.D. program, Wright State University, 2019. Recognition of Incomplete Objects based on Synthesis of Views Using a Geometric Based Local-Global Graphs

The recognition of single objects is an old research field with many techniques and robust results. The probabilistic recognition of incomplete objects, however, remains an active field with challenging issues associated to shadows, illumination and other visual characteristics. With object incompleteness, we mean missing parts of a known object and not low-resolution images of that object. The employment of various single machine-learning methodologies for accurate classification of the incomplete objects did not provide a robust answer to the challenging problem. In this dissertation, we present a suite of high-level, model-based computer vision techniques encompassing both geometric and machine learning approaches to generate probabilistic matches of objects with varying degrees and forms of non-deformed incompleteness.

The recognition of incomplete objects requires the formulation of a database of six sided views (e.g., model) of an object from which an identification can be made. The images are preprocessed (K-means segmentation, and region growing code to generate fully defined region and segment image information) from which local and global geometric and characteristic properties are generated in a process known as the Local-Global (L-G) Graph method. The characteristic properties are then stored into a database for processing against sample images featuring various types of missing features. The sample images are then characterized in the same manner. After this, a suite of methodologies is employed to match a sample against an

exemplar image in a multithreaded manner. The approaches, which work with the multi-view model database characteristics in a parallel (e.g, multithreaded manner) determine probabilistically by application of weighted outcomes the application of various matching routines. These routines include treating segment border regions as chain codes which are then processed using various string matching algorithms, the matching by center of moments from global graph construction, the matching of chain code starting segment location, the differences in angles in the center of moments between the model and sample images to find the most similar graphs (e.g., image), and the use of Delaunay triangulations of the center of moments formed during global graph construction. The ability to find a most probable match is extensible in the future to adding additional detection methods with the appropriate weight adjustments.

To enhance the detection of incomplete objects, separate investigations have been made into rotating the exemplars in standard increments and by object extraction of segment border regions' chain codes and subsequent synthesis of objects from the multi-view database. This approach is novel and potentially extensible to compositing across multi-view segmented regions at the borders between views by either human aided input of border relations or a systematic, automated evaluation of common border objects between the views of an exemplar.

The first results are promising and trigger again the field of recognition of incomplete objects in a different way that was not studied before.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Nikolaos Bourbakis for his valuable input, encouragement, guidance and essential feedback during this long journey and with willingness to work with an older student and full-time worker in industry and academia. You have allowed me to embark on a new, profoundly rewarding journey in academia where I get to work with future computer scientists, software developers and engineers, and information technology professionals that I otherwise would not have been able to do so.

I would also like to thank the members of my committee, Drs. Soon Chung, Y. Pei, and Arnab Shaw for their valuable construction feedback and support.

I would like to thank Dr. Robert Kasper, department chair of the Computer Science Department at Mount Vernon Nazarene University (MVNU), where I now work as a full-time assistant professor, for his support and discussions. I am also appreciative to MVNU, and Drs. Robert Kasper and LeeAnn Couts (Dean of the School of Natural and Social Sciences), for the use of a workstation that allowed me to make more rapid progress on this work than I would have otherwise been able to do so using my personal equipment.

Finally, I would like to thank my wonderful wife Jacqueline, my precious daughter Octavia, and my faithful dachshund companions Mocha, Hershey, and Leonie who have supported me throughout this journey. I could not have gotten this far without their love and affection and with cheering me on when I had doubts.

I. SURVEY

INTRODUCTION

The problem of detecting objects, let along incomplete objects, is a challenging problem for a variety of reasons. For starters, even complete, fully-separated input images may have textured backgrounds, non-differentiated foreground objects, or artifacts from the capture process (e.g., noise, dust, light, etc.) that interfere with the identification process and can lead to misclassification. A misclassification, in this work, is defined as recognizing one object as another, failing to classify an object within the database of well-known objects in the system being employed, or classifying an object that should not be classified at all. Furthermore, if an object is introduced into the system with key areas missing (particularly its borders) or is inseparable due to obstruction or merging within the proximity of a larger object's extent, the problem is made all the more difficult. Sometimes, even with a whole object present, it may be altered from its original form in such a manner that its deformations make accurate recognition difficult or impossible. Accurate object recognition, particularly of incomplete or occluded objects, is a problem that is of ongoing interest to researchers in the computer vision domain.

The first part of this dissertation contains a survey that is focused on reviewing the efforts of various groups in tackling this difficult problem. There are a wide variety of techniques that have been employed and can be broken down into the following areas:

Partial Objects – a partial object, on the other hand, is one missing key areas or views. The simplest way an object is incomplete is to capture of a view of that object from only one or several perspectives. For example, we may be interested in identifying a vehicle, but if we can only capture the left side, rear view, a fractional sub-section of one side, and/or several of those perspectives, an accurate recognition becomes more difficult. A complete recognition of an object is handled if the system is able to capture left, right, top, bottom, front, and rear views of an object free of background/foreground clutter. The lack of sufficient capture devices, occlusion, or background merging contribute toward incompleteness. The works in this section consider graph, geometric, image completeness, neural networks, and computational photographic methods among other partial object recognition approaches.

Image Descriptors (fingerprinting) – the set of works in this section are focused on the use of pixels to form object signatures on both model and sample objects and use probabilistic techniques for finding a match that hopefully exceeds some lower confidence bound.

Mitigation Strategies – One way to improve the recognition of an incomplete object is to try to restore as much of the object as possible and/or limit the overall damage to the input image. After applying this preprocessing step, the techniques discussed in the other areas and/or in combination with general object recognition techniques can be applied.

Other Domain-Specific Techniques – some techniques fall outside the other areas primarily because the algorithms work only on a certain class or several classes of objects and are not generalizable across multiple classes even with extensive modifications to the

underlying systems and algorithms. However, the works in this section demonstrate how the merging of domain specific properties with general techniques can provide insights into how to improve upon the open problem of partial object recognition.

One important note to make here is that this survey will not extensively consider those methods that focus on identifying low-resolution objects or deformed objects. In the envisioned application that this survey is being prepared for, the types of objects that will be identified will not be deformed as they will be in active use – an airplane, a car, a tool, etc. In addition, the model and sample images for future experiments will be captured in such a way that they will be of sufficiently high-resolution to not need to borrow or build upon the techniques for handling low-resolution images. The noise, contrast, brightness (lighting), optics, etc., will be set up in such a way to permit ideal capturing of data. In other words, these types of imperfections are outside the scope of this survey and any future application although it might worth considering such obstacles to object identification in future works.

In the next section, some useful insights into how humans recognize partially complete objects was considered to provide a context for better understanding the level of biological inspiration to be found in the works of the subsequent sections as well as provide insight into how we might incorporate such insights into future works. After reviewing the works of various groups, a comparison of the various techniques will be conducted before finishing with a few final conclusions and thoughts. The techniques described in this dissertation are focused on **model-based recognition** where a stored set of model or exemplar images stored in a database

can be compared to a set of sample images with varying degrees and forms of non-deformed obstructions.

AN OVERVIEW ON RECOGNITION

To better understand the approaches that a computer vision based system might take in identifying incomplete objects, it is helpful to understand how the human mind deals with such shortcomings. It is possible that we might even be able to draw inspiration from various psychological and physiological sources to create a satisfactory performing integrated system of software and hardware components. [9] examined the human mind's ability to handle incomplete fragments that formed symbols that were in various states of completion and where either blocked or not blocked by a masking unit. They conducted two experiments using students from their institution's introductory psychology course to identify a set of repeated symbols. In the first experiment, [9] had the students who were assigned to one of four groups based on the fragment and mask (e.g., occlusion) type of the presented symbols identify three different sets of repeated symbols. They kept guessing until they were correct or an arbitrary one-minute period had elapsed with the amount of time it took to guess recorded. The display of symbol fragments was repeated, at various angles, and had the same area gray tone masked off. In the second experiment, [9] had a similar setup except only open fragmented symbols were used (e.g., they were not closed off), the Venus symbol (Unicode U+2640) was substituted with the numeral eight, and small and large stimuli were used. So, the masking and size of open fragmented symbols were tested. The second experiment was crafted as a byproduct of analyzing the first experiment's results. [9] noted that open fragmented symbols were easier to

identify than closed fragmented symbols and that the masking of the symbols did not have a significant influence on results in either case. They believed that closed fragmented symbols were harder to identify because they were perceived as being complete individual objects and that the subjects didn't feel a needed to try to see if the fragments formed something more complete over a larger region of the display. [9] believed it was possible that the extra lines were seen as extraneous lines in completing larger shapes. The open fragmented symbols were easier to identify, [9] argued, because the subjects observed the gaps in the fragments even with the masking element present and were able to visualize the completed symbol in their mind. The second experiment showed that small displays of repeated open fragmented symbols were harder to identify than larger ones, but only when a masking element was present; furthermore, when a masking element is present on small open symbolic fragmented displays, response times suffer. Later, [9] revisited the second experiment and allowed the subjects to make some time trial runs to get practice in identifying the symbols. In this instance, they found that it helped overall performance, but that the penalty for using a masking element was not eliminated. [9] believed this was because the subjects became overly focused on a small region in the smallest displays and could not remove the distracting masking element from their perception of the display when present.

So, on the one hand, we can remove the psychology in the computer vision application, but perhaps it shows us that we only need to consider contours of a shape when identifying even if it Is obscured or that the contours can help us to quickly eliminate many templates from

a database before applying more computationally intensive criteria to arrive at a high probabilistic match.

The experiments used by [9] were quite manual in nature. The authors in [37] help us to explore the issue of human interpretation with the aid of a computer. In their experiments, five individuals over a period of three days were used to identify a series of characters that had some varying amounts of information removed from them with square rectangles whose sizes varied. The amount of data removed from a letter ranged from 70% to 92%, the rectangle sizes ranged from 1x1 pixel to approximately 16x8 pixels, and the amount of time the letters were displayed ranged from 200 ms down to 100 ms. It should be noted that the authors in [37] used a fixed rectangle size on any one letter; but from any one subject on a given day, the size of rectangle may have been different from one instance to the next. Therefore, a ratio between the amount of data removed from the letter and the size of the rectangle used was established.

It seems obvious for the authors in [37] to say this, but they were able to show quantitatively that it was the letters with the least amount of information that were the hardest to identify. In other words, the authors in [37] concluded that those letters that used the largest sized rectangles and therefore had the highest amount of data removed, were the hardest to identify. It was shown in their study that even if letters had a high amount of data removed, but the rectangle block size was small, the subjects fared much better in correctly identifying the letters. The authors in [37] believed they observed this result because larger block sizes reduced the number of correlations between the various components that made up the letter. In other words, the distance between gaps was smaller with smaller rectangles and larger with

correspondingly larger rectangles. It would have been interesting to see what would have happened with different types of masks. They did note, though, that letters that look similar were more frequently misidentified as more of the letter was removed because they had similar characteristics in their shape with one another. Finally, the authors in [37] noted that the times the letters were displayed did not play a role because it takes the human mind 150 ms to process it.

[37] also carried this study a step further by deriving a formula to describe the information carried by a letter in an image. The formula is based on:

Whether a pixel in the image's letter is black: $\varphi_n(x,y) = \begin{cases} 1 (x,y) \in \eta \\ 0 (x,y) \notin \eta \end{cases} \eta \in A \dots z$

The probability that a pixel in the image's letter is black: $p(x,y) = {}^{1}/_{26} \sum_{p=A}^{z} \varphi_\eta(x,y)$

The information contained by one black pixel in the letter $s(x,y) = -log_2\, p(x,y)$

The formula derived by [37] is:

$$S(\eta \cap \theta) = \sum_{x=0}^{127} \sum_{y=0}^{127} s(x,y)\varphi_\eta(x,y) \cdot \varphi_0(x,y)$$

This formula, in so many works, carries all the information shared by two pixels from the Roman alphabet with the size 128 pixels squared given that the two letters contain black pixels. These letters, therefore, have their similarity defined between them with the following formula from [37]'s work:

$$r(\theta,\eta) = \frac{s(\eta \cap \theta)}{\sqrt{s(r)s(\theta)}} r \in [0,1]$$

The most similar two letters will have a score of one and the least similar two letters will have a score of zero. The order of the two letters should not matter.

With these equations, the authors in [37] calculated the similarity between O & Q to be 80%, but I and H to only be 3%. This makes sense as O & Q are very similar except for one stroke, but I *fits inside* H from the standpoint of the strokes needed to render it. However, the authors in [37] warned us that although these formulas may be based on pixels, humans recognize the features of a letter such the its angles, its strokes, its size, the thickness of the line, and so on. Perhaps this is important because these latter features can be captured in a resolution independent manner and quantified or formally defined for helping us to build rule based systems for use in a computer vision application.

The authors in [22] presented an experimental study with 18 USC undergraduates that showed it is the relative placement of dots with respect to another and not contours that help us to identify those shapes particularly when those shapes are incomplete or occluded. In this study, [22] used a 64x64 LED array board that lit up with patterns that represented the outlines of basic animals and inanimate objects using a standard coordinate addressing scheme or row and column values. There were three sessions in this experiment: contiguous dot placement, random dot placement, and evenly spaced dot placement. In the first session, the participants were asked to identify the shape as the dots were lit up in contiguous groupings of 5% of the total dots of the shape every three seconds until the participant guess the shape correctly. In

the second session, 5% of the total dots not already enabled in the shape were randomly enabled every three seconds until a correct identification occurred. Finally, in the last session, every 20th, then 19th, and so on dot in the shape was enabled every three seconds until a correct identification occurred. Of course, given how irregular some shapes can be, [22] warned us that the spread of dots may not appear evenly distributed. In addition, those dots are evenly distributed based on the addressing of the LEDs array intersections and not on the shape itself.

The results from this study by [22] were quite interesting. It showed that the participants were able to successfully identify shapes with the fewest dots when those dots were evenly distributed than when they were randomly distributed or even placed into contiguous groupings. The simplest shapes required a very small percentage of the total dots for correct identification when they were evenly distributed. Furthermore, [22] noted that the dots did not have to coincide with the inflection points of the object for correct identification in the evenly distributed case. It was also interesting to read that asymmetric shapes were not harder to identify than symmetric shapes as one could have reasoned that it is easier for the human mind to extrapolate shapes when one-half of a shape is a mirror of the other part (e.g., you only need 50% of the information relative to asymmetric shapes to generate a positive identification). [22] did make sure to demonstrate that guessing did not play a role in the results of this study. Therefore, after looking at these results, [22] concluded that many of the object recognition theories from psychology that were based on the use of contour attributes (e.g., orientation curvature and length of segments) did not hold up under these results and that it is

reasonable to conclude that it is how the dots are positioned relative to one another along the perimeter of the shape that is important in recognizing a shape.

This would suggest that any incomplete object recognition algorithmic need only consider the relative placement of points along the perimeter of the shape to correctly identify it even in the case where the object might be heavily occluded. However, I have my reservations about how well this would work for facial recognition or other areas where there many unique instances that should be recognized with a high degree of precision. Given that resolution, though, it might work in part or in whole in situations where you have a relatively fixed group of possible objects. If nothing else, it might serve as a good preprocessing step to quickly reduce the search space of identification.

The authors in [46] conducted a series of experiments that studied how the human mind handles the processing of incomplete objects when they are presented in a variety of different ways and under varying test conditions. These authors believed that how we process such information plays a role in how our recognition of objects later on even if we do not see the entire object(s) in future encounters. In other words, we fill in knowledge of a given object's hidden areas when we see it based upon our previous exposure to similar objects in the past. The authors also studied how various types of stimuli are processed by the abstract and specific object recognition processing subsystems that are present in both hemispheric regions of the brain, though the abstract subsystem is more dominant in the left-hemisphere and the specific subsystem is more dominant in the right-hemisphere. The authors in [46] noted how their work aligns with neuroimaging, neurocomputational, and neural network modeling studies.

It was interesting to read about how authors separated this notion of abstract and specific processing. The abstract processing, [46] noted, allows us to figure out what is present in all instances of category, what is absent in all instances of a category, and what may be present only some of the time. This would suggest that a rules based computer algorithm could always look for certain characteristics in a candidate or be taught to look for them. On the other hand, it suggests that the presence of a feature could narrow the search space of categories as a candidate with a certain feature would never be categorized with objects in a particular portion of a search space. The third possibility suggests the use of probabilistic matches or fuzzy logic, which requires a paradigm shift in having a computer application process. The specific subsystem is much more nuisance like in its nature and allows us to not only describe a candidate's category in a search space and universal characteristics, but precisely what it matches to in the search space. In other words, we recognize that an airplane has an engine, wings, flags, tail, etc., from an abstract level; however, our specific subsystem allows us to differentiate a commercial Boeing 777 from a F-35 fighter. The authors in [46] noted that in this subsystem patterns of such objects are retained in a distributed fashion across a collection of nodes within the brain much like the abstract characteristics are stored in a distributed fashion. This would suggest that abstract detection of incomplete objects will work on matching of characteristics, but specific incomplete object detection will have required working with what is visible in a candidate against the equivalent cropped regions of patterns in the search space.

The authors in [46] hypothesized further that abstract recognition of incomplete or occluded objects would come from the independent activation of a subset of characteristics –

what they called sub-whole activation. The specific subsystem, they believed, is pattern driven instead of feature driven – what they call whole activation. Therefore, a person trying to deal with an incomplete object will try to fill in the missing pieces based on their history and knowledge of past learning episodes. They carried out four experiments, which were labeled from their work as, "priming following brief encoding presentations, … priming following longer encoding presentations, … pre-experimental presentations of whole objects, … (and) pre-experimental presentations of whole objects not used during the experiment." In the first experiment carried out by [46], they tried to figure out if repeated exposure of the abstract and specific subsystems with incomplete objects helped with the recognition of incomplete objects using a random subsystem later. Pictures were presented separately to each hemisphere and then to both for a period. The result of this experiment was that individuals were better at recognizing the same incomplete object later than different variations of incomplete objects (e.g., two objects were named piano, but were different types of pianos), which [46] concluded meant that the specific subsystem was working better because it was less negatively affected by degrading (e.g., hiding) parts of the images to make them incomplete. In this experiment, they also focused on presenting some of the samples to both hemispheres individually (they also presented to both), but did not find any difference in performance, which was contradictory to past findings. They believed this was due to the length of sample exposure and how the abstract subsystem handles limited exposure times, which led them to their second experiment. In the second experiment, the authors in [46] lengthen the exposure times during the initial phase to three seconds per sample. They found no difference in results. In the third experiment, also to

help rectify the disparity in earlier experimental results by their colleagues, the authors in [46] presented the entire form of objects from the abstract category in a pre-experimental training session. They thought this would help the performance of the abstract subsystem. The results the authors in [46] received from this experiment were interesting in that performance improved, but only in the right hemisphere where the abstract subsystem is not dominant (recall both subsystems are present in both hemispheres, but dominance is limited to one hemisphere). They also concluded that although both subsystems should have contributed to overall improved results, they did not, which helped the researchers to understand that these subsystems act independently of one another. In the final experiment, they tried to determine whether abstract activation is most effectively done in the left hemisphere and specific activation is most effectively done in the right hemisphere by presenting additional training examples not using during the primary phase of the experiment. The authors in [46] found that the specific examples were more accurately recognized in the right (e.g., specific) hemisphere where the specific subsystem is dominant, but that both subsystems performed equally in the left hemisphere where the abstract subsystem should be dominant. Thus, pattern based recognition is more efficient (quicker), but that abstract subsystem can also perform well if called upon. Therefore, the authors concluded that the experimental results were different due to the methodology that was employed in earlier experiments with participants receiving all samples in the center of vision and with being familiar with the category names. So, the takeaway from this study for the present work is that pattern based approaches are usable without additional support, but that the use of abstract rule or construct based approaches

require an additional framework of knowledge to be successfully used – provided we are drawing from some biologically inspired sources like humans for working with incomplete objects.

PARTIAL OBJECTS

Graph Search Based

Zheng and Peng, 2013, considered working with incomplete objects to be a graph search problem whose results were statistical measures of confidence in having found the target object. For them, incomplete objects led to false positives in recognizing small objects of interest particularly given the fact that edges and regions are distorted in different ways when occluded.

So, Zheng and Peng begin their recognition process by performing binary image segmentation. Next, they try to extract a region of interest. To do this, they had to calculate a signal-to-noise (SNR) measure of each region, which is a ratio of the number of pixels making up the target region to the sum of the misclassified target and background pixels. Presumably, regions with too much noise are discarded. The authors do not elaborate of what experimental values were used here to narrow the search space. After this, Zheng and Peng try to reduce the number of possible candidates remaining by calculating what they referred to as the shape complexity of a region. This is a simple ratio of the square of the perimeter to the area of the i^{th} region. Once again, a threshold for complexity threshold cutoff is established and those fragments with too much complexity are excluded on the basis that the binary segmented form

of an object is relatively simple compared to background regions that be quite complex even after this processing. Zheng and Peng then calculate the centroids of the remaining regions before employing Canny's and Otsu's classic methods for edge and region detection.

At this point, Zheng and Peng believe the remaining regions can be mapped in various combinations to states with all those states representing a state space. They then propose that processing those combinations to be a graph search problem. Using a collection of model images that have been presumably preprocessed in a similar manner to the target images under consideration, a vector of the invariant moments is calculated for the target region (or combination of target regions) and compared to the vector of invariant moments for each model and configuration of that model. At this point, Zheng and Peng calculate a similarity index between the binary image and the model, which is the ratio of the difference of the vectors to the vector of the model expressed as a probability measure (e.g., one minus the ratio calculation). The fragment with the best calculated image is likely the region of interest containing the segmented target, which they then use the method by [61] for extracting the target from its background.

Geometry Based

The authors in [66] described a two-part algorithm for identifying obstructed objects that depend on the use of feature points (associations) and constraining geometry. The goal is to provide as many strong associations as possible between model images and sample images particularly when the objects in the sample images end up occluded by another object (e.g., increase the confidence of an object identification to some model image or set of model

images). The feature points are built using image patches and an association score is calculated based on Euclidean distances based on two items: the distance between where an image patch occurs in a model and on the sample image and the distance from one feature to the next in a sample. The geometry is then constrained on a feature and feature-pair basis using a spatially focused matrix along with angle and distance distortions between the model and sample images. The authors in [66] note that this constraining process does not hold up well to object deformations since this would increase distance and angle measurements between pairs of assignments (e.g., features close together in an object are more consistent with one another when compared between the model and sample images).

The authors in [66] carried out experimental studies by first quantizing model images with k-means clustering and sample image with forty cluster centers. The image patches from the model images are then generated as color co-occurrence matrices. Thus, the feature association match is based on the intersection of two matrices. Next, the geometric relationship is calculated as described earlier. The authors in [66] noted that other types of measurements could have been used like color or texture, but that they are more computationally expensive and do not increase the confidence of the match in a proportional manner to the amount of additional resources spent processing those measurements. The researchers used forty-pixel image patches and the support region for each feature point was limited to its ten nearest neighbors (e.g., how far does the algorithm go in constraining the geometry). In one experiment, the researchers occluded a toy dinosaur at five degree increments (72 model views total) and found their use of spectral matching with feature association to provide the highest recognition

confidence levels, at or above 85% from 20 to 70% occlusions. This was superior to spectral matching where rates went down more dramatically with increasing occlusion as it is sensitive to noise and scale invariant feature transform with feature association as the researchers believed information rich features were getting hidden. The algorithm overall has a performance of ~O(m2) with m being the number of associated features. In practice, it took a respectable, though certainly not close to real-time effort of 600 ms on average in MATLAB with a 2.4 GHz Pentium using 80 features to recognize a given object. The authors in [66] also used the Ponce Research Group object recognition database using eight object models images against 51 512x512 sample images with varying levels of occlusion from 20 to 70%. The research achieved a 95% recognition rate compared to previous state-of-the-art works, which already had recognition rates of around 90%. The researchers reported that their algorithm took around two seconds to recognize each object from this dataset using the hardware described earlier with one second spent on feature association and 900 ms spent on indexing and verification activities. They note that their method is not the quickest method compared to the others, but that the recognition rates are higher.

Instead of using image patches to determine a set of primitives for use, [16] uses corner points and their relationship among one another to build Triangular Spatial Relationship (TSRs) quadruples that are uniquely encoded in a B-tree data structure for later retrieval against a sample image. Specifically, the boundary curve for each object in the image is extracted, then the researchers use their corner detection algorithm to identify the corner points, which are then labeled as convex or concave and passed to the TSR and B-Tree encoding algorithms

respectively. The TSR portion takes three non-collinear corner points in an image, establishes midpoints and angles that subtend those midpoints, and collects the corner points, labels, midpoints, and angels into quadruples. The researchers noted that a key value in the B-tree has a list of objects associated with it where one of the encoded quadruples from an object corresponds to that key. The researchers, during the recognition phase, begin by extracting the boundary curves in the image. Next, they use their corner point detection algorithm on the extracted boundary curve before applying TSR. The TSR keys are placed into groups, which are processed one at a time. For each key in that group, the B-tree is searched for that key and if found, the object indices are extracted. Eventually, all the object indices are collected together and placed into a status (binary) matrix where the longest substring of binary ones are found for each row and then ranked to provide a probabilistic answer as to what the sample object(s) could be.

One interesting contribution from the work by [16] is to highlight the challenges in using alternative approaches. In terms of alternative geometric approaches, they noted the use of length and angles is computationally expensive since the starting location has to always be calculated and that it does not hold up well to scaling operations due to the dependence on line lengths. If k-means clustering is used to group line segments, [16] discussed how its iterative nature makes the representation and matching phases computationally expensive. As a result, the researchers noted that their colleagues tended to use dynamic programming techniques and transformation operations. In fact, [16] discussed that many other approximation techniques have been used and included, "… fuzzy relaxation procedure, generalized Hough transformation,

neuro-fuzzy reasoning, Markov model, wavelet transform, probabilistic Attributed Relation Graph (ARG) matching, stochastic Hidden Markov Model," (pg. 424). In general, [16] highlighted the fact that the time complexity of model-based systems is prohibitive, particularly as the model database grows. Consequentially, to deal with this shortcoming, the researchers noted the use of indexing methods, an artificial neural network (ANN) like a Hopfield neural network, and other related ANN approaches. The problem here, [16], discusses, is how the network should be rebuilt for insertion and deletion operations along with the expensive exponential level processing time for training the network and possible non-convergence. Of course, this might be okay for narrowly focused domains where the universe of likely objects is well-known and manageably small. Finally, [16] highlight the use of other methods like curvature scale space for handling occluded objects that utilizes the use of contours that has been mentioned in other parts of this review. In addition to potential performance bottlenecks with this method, the researchers noted that it cannot handle a scene with multiple objects well at all. Thus, the major drawback in these alternative methods for handling occlusion is their non-suitability for real-time applications.

The authors in [16] conducted two experiments with their method. In the first experiment, they built a database of twelve common tools and then occluded the tools with objects that were not in the database or were in the database. They did not have any misclassifications in their experiment and as expected, the algorithm recognizes the least occluded objects with the highest degree of confidence. If multiple objects from the database were in the sample, the more occluded objects were recognized, just at a lower degree (e.g.,

rank) of confidence. [16] did not mention how long it took to build the database or process the various sample images, though they do note that the recognition phase has a algorithmic complexity of $O(\log_r T)$ where T is the number of TSR quadruples and r is the rank of the B-tree. In the second experiment, [16] used a dataset from [11], which consisted of twenty objects. The algorithm extracted 217 corner points and generated a corresponding number of TSR quadruples of which 189 were unique.

The authors in [8] use edges in their proposed method for 2-D shape recognition particularly in the presence of occluded images. The researchers noted that their method is based on each shape in the database being unique, which they believed can be demonstrated sufficiently by complete or partial edge information of each object even if occlusion is present in the image as it needs to only find noteworthy contour for the match to be successful. Here is the basic method for the researchers' two-stage processing and matching algorithm: Given an input query image, Canny Edge Detector is used to find the edges of the shapes in the image. This is followed by the formation of a component window (CW) using morphological operations and a thinning stage. A CW is simply an array of Cartesian coordinates. Next, the most significant and measurable edge segments are extracted, which is used to build the graph for matching. [8] describes this process as starting with certain points on the contour being used as nodes of a graph with certain nodes serving as start or end points and other nods serving as bifurcation points. The algorithm then looks for connectivity between any two nodes to build the full graph using depth first search. The paths used to reach the nodes are stored as a contour branch (CBs) along with the segments between any two paths. [8] noted that each node serves as a root node

during a given iteration to fully develop the graph. The CBs are pruned by a procedure not fully explained by [8] to get what they refer to as the most significant subset of CBs, which they refer to as the Initial Feature Contour Window (IFCW). A corner detection algorithm is then used by the researchers to create the Final Feature Component Windows (FFCWs), which is the list of adjacent edges for each identified corner. Then a matching procedure is started comparing object templates in the sample against the objects in the database. The FFCWs are used as a starting point to determine which FFCW is most like the one extracted from a FFCW in the sample image. The Chamfer Matching procedure is used to make this determination. This process is repeated on the sample image at five degree intervals. The full set of Chamfer scores provides a minimum matching cost and a match is indicated if the value is less than some threshold. [8] carried out some experimental test using images from their database and found their technique had a 97 to 99 percent matching rate with few true negatives or false positives even when the objects used overlapped one another. The procedure is not able to handle differences in scale, shear or image noise, which will be a focus in a later work.

There are other ways to deconstruct an image into its constituent components. Some of these techniques are based on or are rooted in the use of a Voroni diagram. They are interesting due to how they mimic natural patterns, their inherent mathematical properties, and its use in solving computer science problems where the structure is used as a basis for more advanced work [4]. A Voroni diagram, at its core, subdivides the pixels in an image into a series of regions based on each pixel's distance to the nearest region [4]. Formally, [4] defines the Voroni

diagram as in terms of dominance of one site, p, over another site q (a site being a set of n points) where a subset of those points are as close to one site as another:

$$dom(p,q) = \{x \in R^2 | \delta(x,p) \leq \delta(x,q)\} \quad (1)$$

Delta is defined here as the Euclidean distance function. There is a perpendicular bisector that determines whether the points get assigned to p or q and serves as the separator between two regions. [4] noted that a region's coverage is therefore based on its dominance compared to the remaining regions in some larger superset of sites, S. This is formally noted as such:

$$reg(p) = \bigcap_{q \in S - \{p\}} dom(p,q)$$

An example Voroni diagram for ten sites in the plane is shown below:

The code used to generate this diagram is as follows:

```
x = gallery('uniformdata',[1 10],0);
y = gallery('uniformdata',[1 10],1);
voronoi(x,y)
nump = size(transpose(x),1)
plabels = arrayfun(@(n) {sprintf('X%d', n)}, (1:nump)');
hold on
Hpl= text(transpose(x),transpose(y), plabels, 'Fontweight', ...
'bold', 'HorizontalAlignment','center',...
'BackgroundColor', 'none');
Hold off
```

The center points were as follows:

Centers	X	Y
X1	0.950129	0.952782

X2	0.231139	0.704062
X3	0.606843	0.953877
X4	0.485982	0.598159
X5	0.891299	0.840743
X6	0.762097	0.442819
X7	0.456468	0.83682
X8	0.018504	0.518703
X9	0.821407	0.02221
X10	0.444703	0.375886

The authors in [4] noted that with any Voroni diagram there are n-1 half-planes with at most n-1 edges with any point on an edge equidistant from precisely two regions. The vertices are special in that they are equidistant from at least three regions. All the edges and vertices form a polygonal partition of the diagram and no region is empty although some are unbounded. [4] went on to note that the degenerate form of a Voroni diagram is a straight line with one unbounded edge. Furthermore, he discussed that when three edges meet, the vertex passes through at least three other regions, but will not enclose any one region. Therefore, given all these properties, [4] concluded that these properties allow the Voroni diagram to mimic the original structure in a computationally efficient manner.

So, what are the applications for Voroni diagram? [4] noted in his survey that such diagrams can be used for searching tasks where objects have attributes that can be quantified discretely using some measure of distance from one another, in cluster analysis to group related units of data together into sets, for scheduling mechanical device data retrieval operations with approximate minimal time (exact solutions are NP-complete), in translational (e.g., collision detection/avoidance), rotational, and path-planning, tasks by robots, and in triangulating an object's position.

The geometric approaches to complete or incomplete image recognition may also consider related structures as well. [4] noted that one such structure is the Delaunay triangulation. A Delaunay triangulation is an edge that connects the center points of two regions in a Voroni diagram if and only if those two regions share a common edge. The Delaunay triangulation, [4] noted, is established through the empty-circle method that considers all the triangles that can be formed by all the regions in such a way that the circumcircle of a given triangle contains only a single region. The Delaunay diagram of the previous dataset used for the Voroni diagram is shown in the next figure:

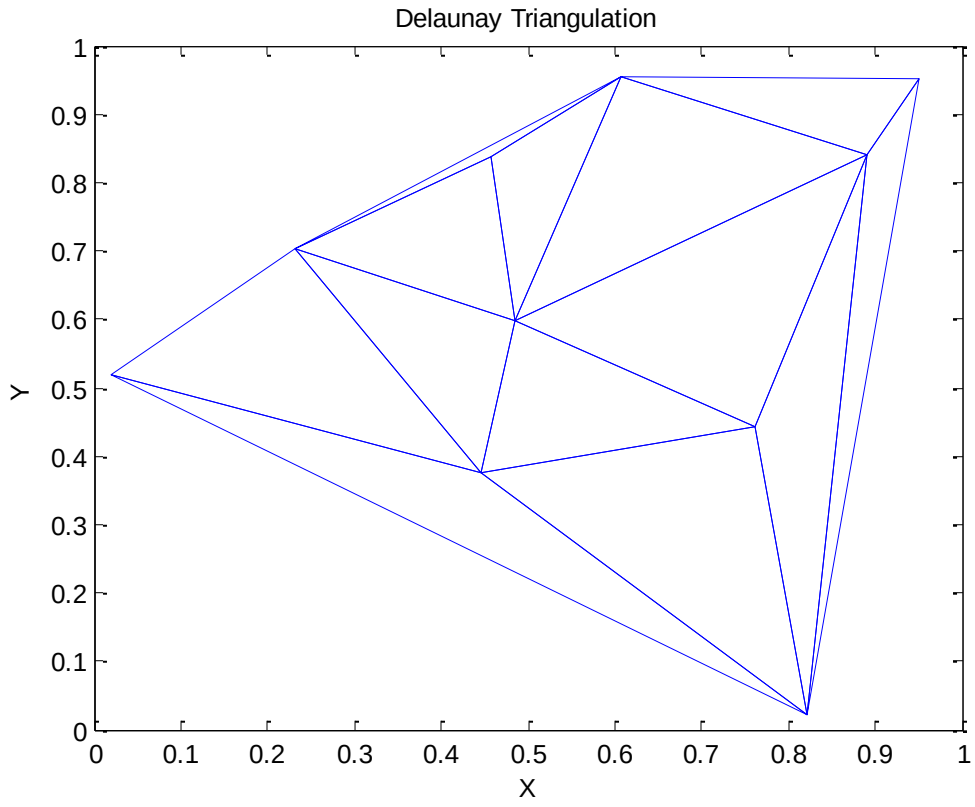


Figure 1: Delaunay Triangulation

Therefore, the authors in [4] concluded, Voroni and Delaunay structures are dual graphs of one another. The Voroni vertices exists within Delaunay triangles and the process of connecting the centers and circumcircles of the Delaunay triangulation gives us the Voroni diagram. In the next diagram, the duality of these two structures are shown:

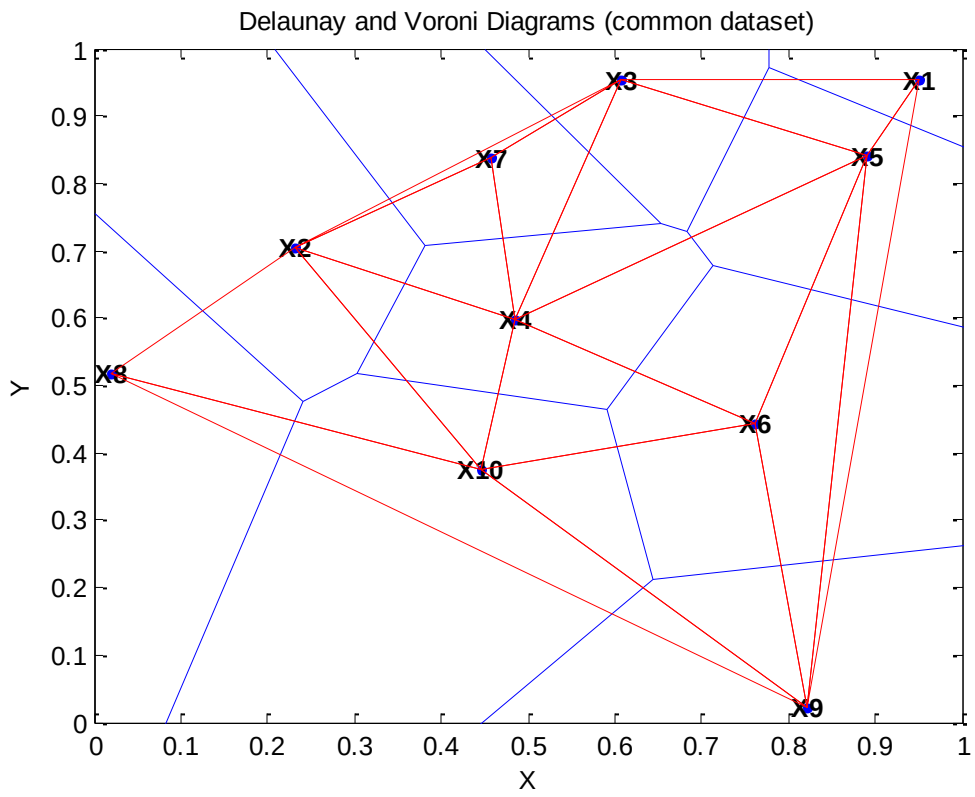


Figure 2: Delaunay and Voroni Diagrams

The remaining code needed to render the previous structure is as follows:

```
Hold on
dt = DelaunayTri(x(:),y(:));
triplot(dt, '-r');
hold off
```

The authors in [4] noted that the edges of the Delaunay diagram are orthogonal to the matching edges in the Voroni diagram, but that they do not always intersect those same edges. Finally, [4] remarked that the convex hull that can be formed from the regions in a Voroni

diagram consist of only Delaunay edges. This in turn leads to the upper bound on the number of Delaunay edges and triangles which are:

$$edges \leq 3n - 6$$

$$triangles \leq 2n - 4$$

From a computer science standpoint, [4] discussed how this structure is a supergraph of the minimum spanning tree, the Gabriel graph, and the relative neighborhood graph (RNG). The minimum spanning tree is a data structure used in the design of networks with resource minimization as a goal. For example, applications could include the minimum travel time from one point to another or the minimum amount of cable to distribute among nodes). The Gabriel graph is used in studying closeness of nearby data nodes. The RNG, which is the connected set of points whereby there does not exist any third point that is closer to two other connected points on an edge than those points are to each other.

Building on Voroni diagrams and Delaunay Triangulation, the authors in [53] tried to answer the question of when part of the object is missing from an image, due to an obstruction for example, is it still possible to identify the object without the need for human assistance. The algorithm developed by these researchers to process such incomplete objects in images is known as the Local-Global (L-G or LG) Algorithm [73-74]. At its core, this algorithm creates geometric and characteristic descriptions for each region in an image. The regions' descriptions

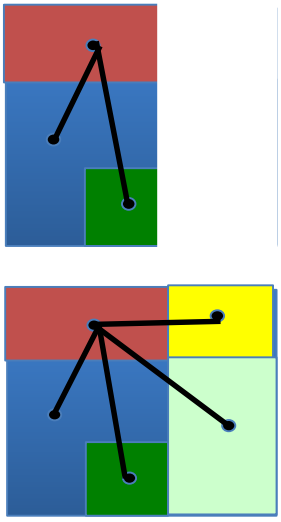


Figure 3: L-G Graph Algorithm Segmentation Concept

when pulled together provide a global description of the image. The authors in [53,73-74] go on to explain that each region or node contains a centroid, line segments that represent the region's border, the relationship among the line segments, and metadata properties for each line segment. They noted that after preprocessing the image with OpenCV K-Means to partition it for additional processing, a chain code method is applied to each region. This is followed by line segment generation using the chain code in part as input. The remaining parts of the LG Algorithm involve the derivation of the centroids, the construction of the global graph using centroids and other regional data (particularly the line segments), and the visualization of the line segment reconstruction and centroids. Figure 3 shows conceptually the final output from processing the LG Algorithm on an image.

The authors in [53,73-74] noted that a line segment, which takes a curved form, can have the following properties: start point, length, orientation, and curvature. The relationship among line segments can be any one of the following: connectivity, parallelism, relative distance, relevant magnitude, and symmetry. The following two figures shows the activity diagrams that make up the L-G processing of an image:

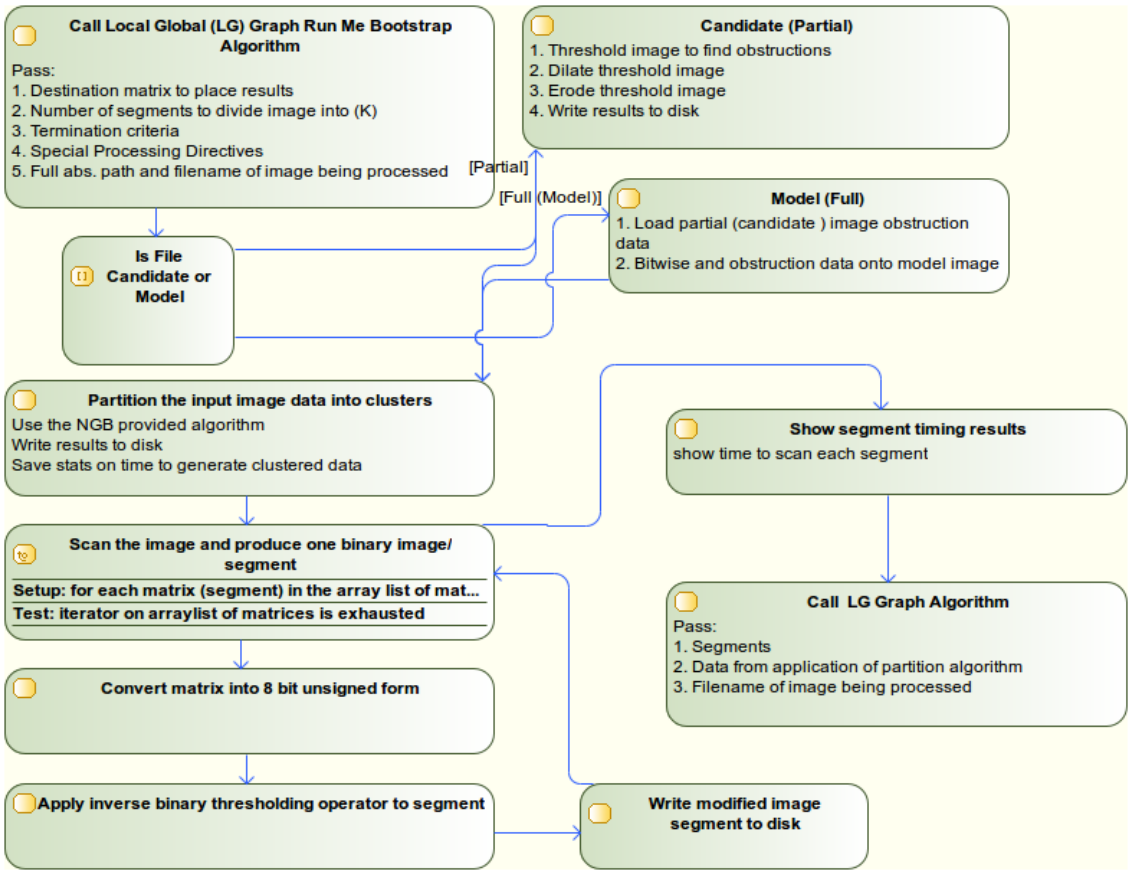


Figure 4: Early version L-G Run Me Bootstrap Activity Diagram

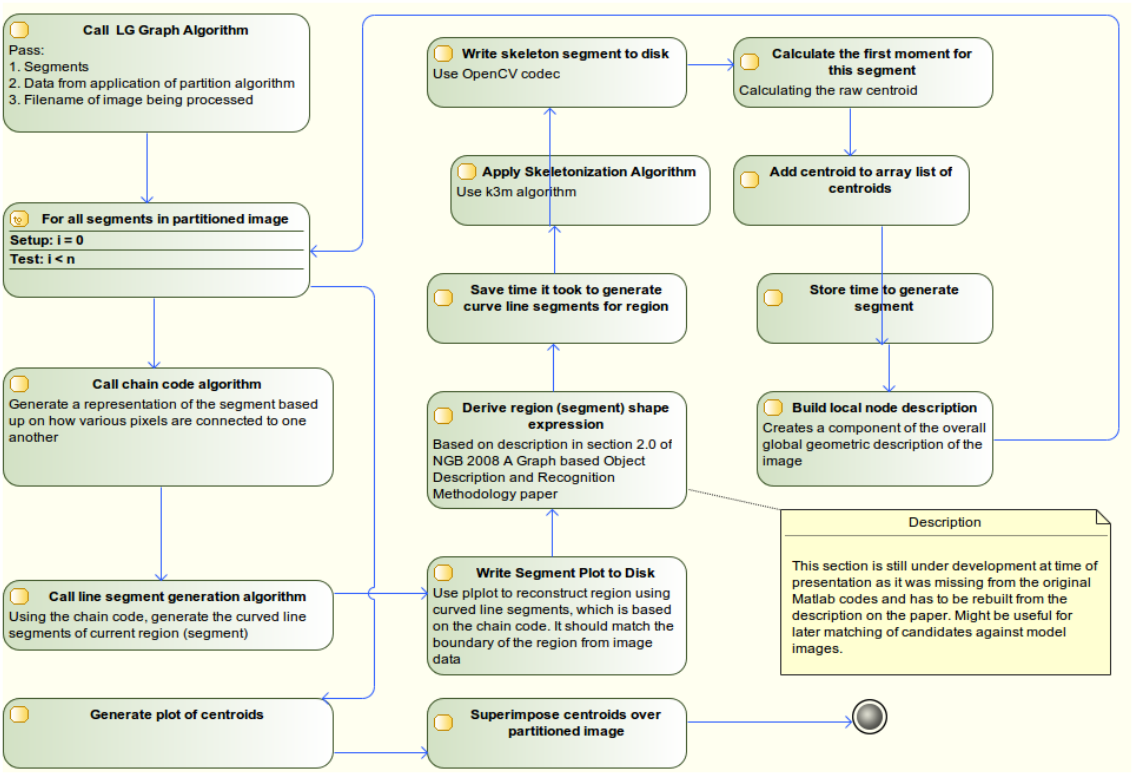


Figure 5: Early version Local Global (L-G) Graph Activity Diagram

Like many of the works reviewed in this survey, a database of templates was formed where each object (toy vehicles in this case) had associated with it a front, back, left, right, top, and bottom side views. The template objects' images were taken with a neutral tone background and were subject to the following processing steps prior to preprocessing: cropping, reduction, grayscale conversion, and moderate smoothing. The early versions of the L-G Algorithm were built using Java 6.0 programming language and processed images on a Xeon® based system with dual E5-1620 CPUs at 3.6 GHZ and 24 GB Ram (the program ran in single threaded mode though). The researchers noted that segment separation and region growing took two minutes, but that the core algorithm took 24 minutes to execute given inefficiencies in

the code to derive region meta properties. On the sample objects that were processed in the experimental phase, the authors in [53] noted that if the obstruction had any irregular edges, extraneous micro-regions were processed, which in turn affected the likelihood of a high probabilistic match. The researchers noted that the irregular edges were likely due to how the morphological operators were applied and the subsequent interaction with the model image.

In short, the authors in [53] noted several findings. First, from their experimental study, they noted that small differences in processing the obstructions and how they are draped over the model images can have a meaningful effect on the partitioning of the images and subsequent processing by the LG Algorithm. Second, when processing sample objects against the available template objects, better matches can occur if the samples are processed with the same type of obstruction to provide an apples-to-apples comparison. Third, the use of contours to help in obstruction template matching yielded poor results. Fourth, the researchers found that simple threshold, morphological, and bitwise operators worked best in identifying and transferring the obstruction. Finally, the use of floating point data provided more precise processing, but hurt confidence matching.

In the future, the authors in [53] plan to work with different partitioning algorithms, handle different types of obstructions, fix inefficiencies in the code to calculate the line segment metadata, improve the detail of line segment metadata, show the likely missing section of sample images when identifying the match, and improve the matching process to build better confidence measures.

Image Completion Based

One method that helps with handling incomplete objects in an image is to use image completion. As the name implies, the goal with image completion is to try to fill in those areas of the image foreground or background that are missing to achieve some objective. The authors in [67] noted that there are four categories of image completion techniques. The first category of techniques treats image completion as a boundary value problem to be solved using partial differential equations (PDE). The authors in [67] noted that you need information on the missing structures and undamaged textures to make this class of techniques work. This means that these techniques only work on images with minor amounts of damage. The second class of techniques, per [67] and [30] are texture synthesis based techniques that analyze the entire image to find sections that can be used to fill in damage parts of an image. The problem with this category, per [67] is that the results are not particularly realistic and fare poorly when the texture and structures of the damaged region(s) is/are different from the overall image. The third category of techniques are semi-automatic techniques where the user draws the lines or curves defining the target region along with defining structures and source regions when using texture synthesis. The trouble with these techniques are that they require human input to produce good results. Finally, there are sample based methods. Per [67] images similar to the one with damage are retrieved from some predefined database and then graph cuts are used to copy the sample regions of interest to the target picture. Although these techniques work well with complex structures, the authors along with [23] noted that results are not as good with

images having damaged sections that are non-rigid. Also, it can be hard to transfer the appropriate texture from the sample image to the damaged image.

So, the authors in [67] decided to combine a suite of techniques along with novel modifications to produce satisfactory results. These researchers employed a multistep method using contour reconstruction and non-rigid registration followed by texture filling. The contour reconstruction involves extracting the contour of the damaged foreground structure followed by non-rigid registration where point sets are generated from the contours and aligned using a coherent point drift (CPD) algorithm. The authors in [67] then follow up the alignment step with a Bezier curve fitting procedure to eliminate gaps between the aligned contour set and the original damaged foreground structure contour set. At this point, the foreground is done and the background has to be filled with the appropriate texture(s). The authors in [67] take the foreground textures from the sample image and the non-damaged portion of the original image and apply a histogram correction needed to get the sample image to match the original image. They also use the PatchMatch algorithm from [7] to improve these results. One of the problems with these techniques used by [67] is that they are computationally expensive and require approximation refinements to get satisfactory, but tractable results. In their work, the authors in [67], used the method by [63] for searching the sample image for the regions to retrieve using global coherence constraints and then filing the damage target regions of the original image using neatest-neighbor and weighted voting scheme. The authors in [67] do deviate by using the median of the pixel color votes for each color channel. Experimentally, these authors found their work to achieve good results qualitatively and quantitatively compared to competing

methods from [63], [31], and Photoshop. The authors in [67] took their C++ and MATLAB codebases and tested using a variety of patch sizes, scale factors, and rotation factors. The quantitative measure used was peak signal-to-noise (PSNR) ratio. As mentioned earlier, performance is a constraint of their work and even on a reasonably configured PC with 2.6 GHz processor and 8 GB RAM, the processing took three to five minutes. Also, the authors of this work noted that it can be difficult to pick a good patch size as picking one that is too large will make it hard to find good redundant patches, but that one too small may not capture the complete texture. In short, the need for human intervention and computational costs are the main shortcomings of their approach although the results are impressive.

The authors in [30] uses an interactive method to work around the limitations of conventional image completion techniques. This interaction allows [30] to construct a restricted perspective mesh and then fill in the mesh using image wrapping and texture synthesis techniques on the structured and non-structured regions in such a way that multiple images, 3D depth information, and/or camera data is not needed. However, the down side of their work is that it requires interactivity by having the user identify the target region, four perspective direction points, and several fragment points for segmentation—they don't have to draw perspective lines or mark the vanishing point. However, it is not automated. [30] argue, though, that it is nearly impossible for an algorithm to find the vanishing point in an image, which is the intersection of the first and last perspective lines in the mesh generation. However, it is easy for the human visual system to identify such a system by finding and then projecting such lines to their intersection. [30] found their approach to be reasonably efficient on smaller images. The

bottleneck in their method comes when texture synthesis is needed and with respect to this operation, resolution has a noticeable impact. The final images in their work seem to have been filled in quite nicely in both horizontal and vertical directions. [30] are planning on automating their work in the future along with handling shading and structural limitations. They also want to reduce current performance issues by using GPU operations.

A very different approach to image completion was proposed by [31]. In this work, the researchers proposed a crowdsourcing solution to image completion that depended on using a popular photo sharing site, in this case Flickr, for filling in missing pieces or replacing occluded or undesirable elements. [31] believes their work is computationally efficient in part because the set of semantic differentiable scenes is much smaller than the entire search space of all images. They also found their efforts to have improved on the state of the art at the time because they were not limiting the repair of the image to the image itself via some means like texture synthesis. This also let them work around the cloning of elements in the image that in some cases would give the appearance of obvious cloning of content – e.g., it would break down the authenticity of the image to the viewer, In order to find those semantically relevant photos, [31] used a low-level gist descriptor with color information included and the missing regions excluded along with template matching using all possible translations and three scales. A gist is a construct that helps to describe the scales and orientations of the elements in a photo. They did not use higher-level textual descriptions, principal components analysis (PCA), or nearest neighbor searching. Once the appropriate match had been found, [31] used image gradient minimizing graph cut seam finding and whole image Poisson blending to integrate the fragment

into the source image in as natural a manner as possible. Finally, the user was shown the twenty best target candidates used a scoring formula described in their work. They noted that their work had successful outcomes without image artifacts, successful outcomes with minor amounts of image artifacts, and failures due to the mismatch with less popular scenes or with individual elements of popular scenes – often with high levels of image artifacts for the former case. [31] noted that the failures in the latter category occurred because there was no high-level object recognition component in their work. The authors also defined success in their study based on how well individuals identified their target images as real compared to a competing method or real images without missing components in a separate, post-processing randomized study. The outcome was influenced by the amount of time the subjects had to review the images with the results of this side study showing favorable results for their method over the competing method, but not as good as the unaltered images. In short, the authors believed that their method combined with other established techniques could generate far better image completion results although it will require a large database of images and significant amount of computing resources.

The authors in [41] envisioned the image completion problem as one that needed to be thought of as a convex optimization problem and deconstructed the RGB channels of an incomplete image into separate matrices that were then processed with their Alternating Direction Image Completion (ADIC) algorithm to recover the data missing in each channel. Their algorithm is based on a Low-rank Tensor Completion (LRTC) method of [42], but they claim to achieve convergence far more quickly by considering the input data not as a tensor (e.g., scalar

or vector), but as three matrices that they then process with their ADIC algorithm to minimize the matrices' rank. The core of the ADIC algorithm, called the Alternating Direction Method (ADM), breaks the minimization task into two smaller problems that focuses on filling in the X and Y component data in an alternating fashion until convergence is achieved. The most difficult part of the algorithm is cubic in its performance and requires an initialization step that can be filled with all zeros, random data, or a more helpful set of data values using a technique like Inverse Distance Weighting method, which uses a weighted average of known values to fill in the missing values. The authors in [41] were also able to improve the performance of the algorithm on large images by subdividing the image into smaller regions and instantiating an instance of ADIC on each region of the subdivided image. In short, the authors in [41] found their algorithm to behave well on restoring globally degraded images, with removing undesirable objects provided they were manually marked prior to processing, and on traditional image denoising tasks; but not on images where large sections were missing.

Neural Networks

The authors in [70] provided an excellent approach to identifying incomplete images. Taking inspirational cues from the human visual system, constructive perception, and electronic circuits, they developed a deep learning network called the semiconducting bilinear deep belief network (SBDBN). The SBDBN network is different from previous works, first of all, in that it tries to preserve the input data in a manner more consistent with the human visual system by using second-order planes instead of first-order vectors with respect to the input data that is supplied to the network. Second, the authors in [70] tried to create an improved neural network by using

Semiconducting Restricted Boltzmann Machines (SRBM) as a core construct of their deep learning network. This allowed them to emulate the human characteristic of focusing on what is present in an incomplete image rather than what is missing and letting the available or missing data drive initial and refined guessing during the course of processing. The authors in [70] noted in their paper that there is a "switch" or parameter or each connection between the input and hidden layer that is switched on or off during training based on whether or not a given feature is present in a given sample. This in turns effects the weights that are used in the connection dependent on the learning objective function being used. Third, the authors in [70] described their SBDBN being an improvement over previous efforts in that they employ a three stage deep learning algorithm that includes a two variable initialization of the semiconducting switches that is crucial for training and later identification of fresh incomplete images, heuristic based reconstruction of the data at each hidden layer which takes advantage of the semiconducting switches' presence or absence of data using a virtual voltage concept, and the refinement of the reconstruction via classic backpropagation through the whole model to optimize the parameters that are key to recognizing new images accurately once the training data has been processed. The authors in [70] argue that this process is analogous to the human processing of images that takes an initial guess at identifying at what he/she is looking at and then refines that assumption as additional processing takes place in the visual cortex area of the brain. They further argue that existing deep learning neural networks only utilize the refined guess and discard the initial guess, which the authors in [70] argue is crucial in human recognition tasks. The authors in [70] noted that the use of the initial guess is achieved in their work during the first stage of their

deep learning algorithm. Now in terms of experimental results, they demonstrated superior results over previous works when it came to recognizing hand written characters from the MNIST dataset with missing areas or degraded resolution and incomplete or obscured faces from the BioID and StarFace datasets. In the StarFace dataset an obscured image involves someone wearing a hat, sunglasses, etc.

Image Descriptors (e.g., fingerprinting)

One way to identify an object in an image is to form some type of unique fingerprint or signature of the image that can be compared against a database of images that were previously preprocessed and fingerprinted. The author in [44] developed a method for building a unique description of the objects in an image called Scale Invariant Feature Transform (SIFT). This approach takes an image and generates a set of feature vectors or SIFT keys from the high contrast points for each object in the image. The high contrast areas in the image are typically the edges. This algorithm is resistant to changes in rotation, scaling, and translation by using difference of Gaussian functions, scale-space pyramids, and orientation assignment. Lowe also shows in his work that it can also resist modest changes in lighting conditions and affine operations by using blurring and resampling (thresholding) operations. The large number of feature vectors generated in Lowe's work is able to handle moderate changes that might occur in an object position or its occlusion by another object. As such, it only takes a minimum of three matching feature vectors in the model and sample objects to generate a probable match. Lowe noted that the feature vector mapping is based on Euclidean distances. The matches are refined by characteristics stored in the feature vectors like location, orientation, scale, and so

forth to improve the matching process. Lowe uses a k-d tree algorithm called best-bin first search method to achieve efficient indexing and matching times along with large-bin Hough transforms to find model poses quickly. Lowe goes on to describe how the probably of a match is increased by placing all the candidate matches from the model databases into a hash table by decreasing order of size and is then processed with a verification process that uses a linear least-squares fitting method. Lowe performed a series of experimental test to verify the claims in the paper and noted that it took approximately 1.5 seconds/image to recognize the objects in a 384x512 image using a Sun Sparc 10 processor that was state of the art at the time the paper was written in the late 90s. The key generation took about 60% of the time and the indexing and matching took about 40% of the time. Lowe also spent some time noting how his approach to object recognition mirrors biological processes based on studies that were done with primates that looked at the inferior temporal cortex. He also noted that these studies showed how different types of neurons can react to more complicated shapes like faces, color, textures. All of the features seem to work, according to [44] in a parallel process that involves intense bursts of activity, but a serial binding process that mimics the verification process in his work. Lowe concluded by discussing how he would like to use multiple views in the future along with working with additional feature types like color, textures, edge groupings, and so forth.

As an alternative to constructing L-G graphs for identifying images, it is possible to use interest point detector operators as a way of building a fingerprint for an image. Typically, these operators require the use of the complete image for satisfactory operation., The authors in [38] devised an interest point detector operator that can work on incomplete images though their

definition of an incomplete image involves pixels that are missing that irregular intervals over the entire image. Their work does not seem to handle or discuss large contiguous sections of missing pixels. The authors in [38] takes those pixels where image data exists and computes an image planes using those nodes with Delaunay triangles. This allows for the construction of operator that vary in shape, size, and number of vertices over the space of the image. Since the data is incomplete, the authors in [38] use finite element method to construct these dynamically changing elements. The authors believe this method gives them he flexibility needed to apply the technique of interest point detectors operators to incomplete images. Experimentally, the used blox test images and a set of synthetic corner images from a specially constructed database with full image reconstruction where necessary to compare their techniques to conventional image detection operators. The data presented led [38] to conclude that their work performs favorably compared to standard techniques and even better (or at all) when the image data is missing a significant number of pixels throughout its entire space.

An earlier work by [68] gave the community another way to create image fingerprints. In this work, the researchers proposed a technique called *Pattern Hashing* that integrated an appearance-based approach into the framework of a geometric feature-based approach. The goals with their work was to minimize the errors from certain types of changes that might be seen in candidate images and to handle moderate amounts of occlusion and noise in candidate images. The appearance-based approach can overcome the geometric-based approach for weakness with textures that occur with faces (a focal point of their work) while also handling image noise better while the geometric-based approach can overcome the high search costs

found with matching operations in the appearance-based approach. The integration occurs with the geometric keys consisting of lines or points being replaced with image patches. The authors in [68] begin by creating a model database. For each model image, a feature point detection process is executed and from that at least three interest points are selected to form the basis. After this, what the authors in [68] refer to as appearance patches are produced for all combinations of interest points selected earlier and placed into the hash table. The function used for hashing is a combination of the angle between two vectors and the ratio of the length of the two vectors for the three interest points selected at the beginning of the process. The use of three interest points provides the invariant characteristics needed to guard against rotation, scaling and translation differences sense in candidate (sample) images. A linked list is used when a cell in the table has multiple entries. The entries in the hash table consist of a data structure with the quadrangle of the region, the portion of the image region that was cropped, and the whole image region.

The authors in [68] describe the corresponding matching procedure by discussing how they take an input sample image and apply the feature point detection procedure to generate the salient points. Next, all the triplet combinations of the salient points are generated. The authors in [68] then note that the combinations are then used for generating the patches, calculating the coordinate system, and the invariants. The invariant description is the one used for selecting the model for matching while the patches are used as the candidates. At this point the patches in the sample are compared against the model's patches using qualitative trinary representation (QTR) with the sum of absolute differences (SAD) serving to normalize the

comparison between sample and model patches. The authors in [68] note that QTR is an approach to identifying possible matches that uses the relative magnitude change of a given pixel's neighbors. Once a similarity measure exceeds a predefined threshold value, a hypothesis is generated. The hypothesis is then subjected to an interest point hypothesis space and orientation hypothesis space voting procedure to determine if there is a large number enough votes to warrant the hypothesis being supported. The authors in [68] verified their work using 850 images composed of the faces of twenty-five individuals and found acceptable results even when the image was rotated upside-down or an individual was wearing an eye patch which provided an instance of occlusion. However, the authors in [68] did not highlight more severe instances of occlusion or qualitative measure of their results.

Computational photography based

The authors in [71] provided some ideas on how to handle incomplete images. In their work, they showed how given an input image with a reflected or occluded contents can be corrected by using a series of pictures captured around that same time using a non-rotational, panorama-like capture motion. Their work was not the first to use motion parallax, but improved on the state-of-the-art techniques by using a pixel-wise flow field motion representation and an edge-flow method [71]. The work suggests that applying computational photography techniques like the one mentioned in this work as a pre-processing step could go a long way toward helping us to more accurately apply a given identification procedure. The results of [71] was quite favorable on images containing reflections and on images that were occluded, but the high-resolution images required processing on an advanced mid 2010s era

workstation with significant memory and time needed to process a single sequence of images (e.g., 20 minutes in one example using 3 GB of RAM). The authors also noted that the low-resolution version could be processed offline or through a Windows phone app in about 2 minutes using a gigabyte of RAM. The consequence is that improved identification would come at the cost of a delay, which could be a very negative factor in certain domains like security where quick analysis is sometimes needed. The speed issue aside, one other strength of this work was the use of both qualitative and quantitative methods for judging their results. The authors in [71] used the normalized cross correlation (NGC) on the results verses the background image without any reflection or occluded components.

Mitigation Strategies

One simple way to improve the identification of incomplete objects would be to use various techniques to improve the quality of the image itself or eliminate the blocked/occluded areas prior to applying some other of the other techniques that have been discussed. [1] discussed a system that combined High-Dynamic Range (HDR) and compressive imaging (CI) in a system called HDRCI to handle the capturing of objects in areas where there are high levels of contrast in the object and/or the lighting on/surrounding the object along with the capturing of details outside the human visual range (e.g., both terahertz and millimeter ranges at most opposite ends of the electromagnetic spectrum). The contributions in the work are focused on how the principal algorithm integrates these two technologies to provide images that otherwise could not be captured using conventional photographic equipment. The authors in [1] describes how HDR is a technique that provides a dynamic range greater than a conventional photograph

by taking multiple pictures of the same subject using a variety of exposure times and bracketed gains (both positive and negative). The resulting saturated image has a sort of pop or depth to it not found in the standard photograph that reveals details that might otherwise might have remained hidden. However, as artistically pleasing or functional as an HDR photograph might be, it is an artificial construct and does not show the subject as a person would have seen it. The author in [1] then describes how compressive imaging takes single pixel samples using just one detector to build the entire image and as such avoids time-consuming processing and compression steps. It works on the principals of sparsity and incoherence to build an image at sub-Nyquist rates; otherwise, as the authors in [1] noted you would need twice the number of samples in the image to reconstruct a photograph with complete accuracy. The author in [1] goes on to describe the algorithm driving the HDRCI system. It starts by setting the gain of the Analog-to-Digital Converter (ADC) and then uses compressive sampling (CS) to capture the image. After this, the algorithm by [1] then finds the pixels exceeding a predefined threshold value and they receive attenuation in such a way to keep them above the noise level, but below the saturation level. At this point, the algorithm tests if the algorithm is still below the maximum number of predefined iterations and whether additional attenuation is possible. If below the maximum number of iterations and additional attenuation is possible, the ADC gain is increased and the capture process begins again. If not, the algorithm terminates. The experimental setup by [1] demonstrated good results and eliminated the need for calibration due to the use of single-pixel detectors, but encountered issues with time and accuracy tradeoffs for the number of iterations in the algorithm and with dynamic range due to limits with pixel depth and overall

noise in the system. The author in [1] also noted the equipment in the experiment limited results. Although the technique by [1] may not be useful for this work, it does show the need to improve the inputs using a variety of techniques whenever and wherever possible.

The authors in [69] propose the use of deblurring to improve a degraded image. If it is possible to get a poor quality image that has been blurred and suffers from noise issues to look closer to what a person might have seen if observing the scene in person, then it might improve the accuracy of identifying incomplete objects. The authors in [69] noted in the background of their work that a blurred image is simply the product of a true image and a blur function with noise added to resulting output. If the blur function is known, what the authors referred to as the point spread function (PSF), then the following classical techniques can be applied: "inverse filtering, Wiener filtering, and least-squares filtering." However, if the PSF is not known, which is often, then the authors in [69] say one of the following techniques can be used: "iterative blind deconvolution, recursive filtering, blind deconvolution based on total variation, and so on." The authors in [69] warned, though, that the use of iterative or recursive techniques can lead to lengthy processing times (not to mention stack space can be easily exhausted in certain programming languages especially when they cannot handle tail recursion). Also, these algorithms use sophisticated numerical techniques that [69] note are subject to diverging and becoming unstable.

For their work, the authors in [69] drew upon the APEX method used by [10]. This method, the authors in [69] describes, depends on, "…degraded images that can be expressed as convolutions of the true image with two-dimensional Lévy "stable" probability density

functions…. A direct blind deconvolution technique based on the Fast Fourier Transform (FFT)." The successful use of this process depends on the input parameter values chosen and the number of trials conducted. The authors in [69] later noted in their work that Lévy processes are seen in many natural processes, are useful in real data analysis can occur and that they are stochastic in nature. It was [10] that made the connection between Levy processes and image processing applications due to the representation of such with Fourier transforms. This also underpins the improved technique used by the authors in [69] along with hypothesis that an image's average power spectra or Fourier amplitude meets the criteria for being a power law/heavy-tail/Pareto/Zipfain distribution. The authors in [69] improved on the APEX method by removing the iterative component. In other words, they determine the values of the input parameters to use in the APEX deblurring process without the need to run any trials. They call the use of this technique along with its hybridization with general slow evolution of continuation boundary (GSECB) Direct APEX (DAPEX). The authors in [69] mentioned that GSECB is an approach to image deblurring that that treats the deblurring process as analogous to the inverse of the heat diffusion process so that as the image is recovered, the artifacts or noise in the image is kept to a minimal.

The researchers conducted a series of experimental trails on 512x512 grayscale images where the original non-blurred images are available, and found that their technique provided qualitative and quantitative improvements. The authors in [69] used normalized mean squared error (NMSE) in their quantitative measurements and compared their work to synthetic blurred images without any corrective operations applied. The authors in [69] did not provide

measurements of standard APEX treated images or using the other techniques described in their work. Overall, the authors in [69] showed their technique provided a six-fold improvement in image quality as measured by NMSE. They also applied their technique to images where non-blurred originals were not available or did not exist and saw qualitative improvements. The researchers claim that the algorithm can be applied in a real-time manner, that is easy to implement despite the heavy mathematics behind it, and that it can process most images in a few seconds using mid 2000s low-end computing systems with an image of 720x480 taking just over 37 seconds to process. However, classic real-time algorithms and systems tend to have response times of hundreds of milliseconds or less. Of course, without a fully published algorithm it's harder to perform the asymptotic analysis needed to determine if the scale could scale well in a real-time environment. The authors in [69] also discussed in their results that it is necessary to know the signal-noise covariance ratio a priori, which is approximated by applying a mask to a flat area in the image and an area where there are structures. Furthermore, the authors in [69] required the input of sharpness parameter or overall noise in the image, which seems to be empirically derived. Finally, the researchers require the user to specify the frequency range for the deblurring function to work over.

The authors in [23] described how they improved image detection recognition by filling in the missing contours in the occluded objects. The researchers in this work used Maximum a posteriori (MAP) based Bayesian approach after applying an appropriate edge detection filter. This allow one to obtain an estimate of where the occluded part of the object might lie behind another object. However, [23] noted that since the number of ways an occluded contour could

be filled in is effectively computationally intractable that they use a database of common object shapes that were created based on human perceptions of the most natural ways to complete various partial contours. Thus, this database of complete shapes serves as our set of observations in our likelihood function. From this restricted space and application of a thin-plate-spline (TPS) model, a set of rigid and non-rigid transformations can be used and a vector space of 2D plane points can be generated. To better detect occluded objects and to eliminate false positives, [23] used a modified form of Hough transform methods called, "…step voting for occluded hypotheses," [23] (pg. 485). The independent, confidence-weighted votes come from the set of completed contours and is augmented by contour templates that come from the database discussed earlier. Naturally, the less occluded a given reconstructed contour, the higher the level of confidence in its reconstruction. [23], noted that this process of determining the best reconstructed contours is an iterative process divided between reconstructing the edges and then applying the voting procedure on the reconstructions. After this point, the researchers used what they called, "…hypothesis confirmation with occlusion relation interpretation," [23] (pg. 485). This is basically a safety check on those objects who contours have been identified as occluded to make sure the boundaries are reasonable based on a set of equations outlined in their paper. The checks are focused on looking at the endpoints.

To verify their work, [23]., conducted a series of experiments. In the first experiment, they but a database of eighty occluded images from four different classes of objects, which included giraffes, mugs, swans, and horses. The researchers followed the database building used by [28]. [23] were satisfied by the experimental results that their contributions provided an

excellent method of reliably detecting and recovering occluded objects. It is the researchers' belief in this work that prior works failed to handled occluded objects. Specifically, [23] compared their work to [52] which they considered to be the best possible work in this area prior to their contributions. [23] found the occlusion detection performance to be superior to [52]'s work by using Precision/Recall (P/R) curves for the difference classes. The researchers also found that their technique localized the objects better than the competing work. In the second and final experiment, [23] placed ellipse boards of random sizes and aspect ratios at different locations on the images coming from the ETHZ dataset. However, this generalized approach yielded only modestly successful results with the contours being reconstructed accurately only 24% of the time on average across the difference object classes. In the worst case, the authors were forced to use smoothness operators to reconstruct the contours, leading to the disappointing results.

Other Domain Specific Strategies

The authors in [57] describe how they utilize a suite of different techniques for working with images that are less than ideal to improve the automated identification of different types of rodents they are dealing with in their local environment. They digitize ink laden tracking cards from a tracking tunnel, but often find that the samples contain numerous defects that hinder the identification process. This includes footprints that are missing portions of the footprint from the ink being too light or the animal not pressing down fully on the card, areas where there is too much ink due to the animal sliding its foot causing key areas of the footprint to merge together, or the digitization process itself causing information loss. To begin to combat these

issues, the authors in [57] used Otsu's method for digitizing because they wanted a binarization method that automatically establishes the threshold values to use, but did so in an efficient and stable manner (e.g., results are consistent for the similar input data on repeated applications of the algorithm). Next, they use the geometric properties of a rodent footprint to identify the critical features, which is based on the use of centroids, major/minor axes, and vertices. These features help them to define rectangles around the toes and pads of the footprint. If these areas have white pixel holes, they are filled by a larger enclosing set of black pixels and those areas determined by automated thresholds to be noise are discarded. After this, the erosion morphological operator is applied with a dynamically sized disk mask whose size is governed by the size of adjacent regions to the one being processed. The authors in [57] did not find any benefit in applying the other morphological operators to the footprints. Then to get the best areas of the footprint, they use geometric relations and the Delaunay triangulation algorithm to get the adjacent regions and edges from which they can determine if each region is part of the central portion of the footprint or one of the smaller areas near the toes. Finally, for the remaining damages areas of the footprint, the Graham based convex-hull algorithm is applied, which was chosen for its O(n log n) worst-case asymptotic run-time and ability to apply to both polygons and a set of points.

The techniques in [57] are best suited, therefore, for use in monochromatic processing with simple polygonal shapes and a narrowly focused subject domain. The use of domain knowledge to handle incomplete information can also be seen in [12] that discussed a computer vision system for processing electrophoresis gel images of seed samples. They also use a set of

techniques to handle the samples. The process starts with eliminating areas not containing useful information, which in their case was the upper region of each sample. To do this, the sums of each row's pixel's values are added together, the sums are plotted as a curve, and the plotted curves are then smoothed using weighted averaging. The left valley is the top region and the right valley is the lower region, which is the area of interest. From the rows corresponding to the right valley, column pixel intensity values are summed to highlight the relative minima. Each minima is a possible sample center from which means and variances are calculated along with noise and false sample centers being eliminated. The authors did not elaborate the criteria by which noise and false samples are eliminated. Once the samples are found in the gel, the bands of interest, which are valleys in the plotted distributions, for each sample have to be separated. The authors use a smoothing process to aid them, which involves the use of a 3x3 convolution filter and adaptive smoothing. This eliminates noise, but also eliminates some useful data in their technique. They use domain specific heuristics at this point to restore the missing information. Finally, normalization is applied and mean pixel values are calculated for the valleys of interest from each band. A similarity measure is used to match each sample against the well-known templates where each template is a different seed type. The similarity measured used is distance measure of intensity values between sample and template. Finally, the authors suggested a majority vote method for improving results.

The fingerprint domain is an area where there are often incomplete and/or poor quality source data to work with when trying to match a sample against a database. Traditionally, fingerprints were processed using ink and paper with each digit being rolled separately from the

ink to the paper. In recent times, digital systems like the Live-Scan system, uses a scanner to process those images and may be either fixed or rolling in terms of how the fingerprints are processed ('Live Scan Fingerprinting', n.d.). The authors in [34] proposed an improved classification based approach to working with fingerprints. They noted that among the techniques for classifying fingerprints, classification techniques work well on incomplete or low-resolution images. In particular, [34] suggested using the ridge in a pressed (e.g., plain) position as a basis for classification rather than older rolling techniques. The technique starts with a classification stage that determines which pattern-based features (e.g., arch, tented arch, left/right loop, or whorl) the candidate image possesses and uses that against the database of images for which the same classification has been done previously offline. A binary scoring system is used in the matching process. Overall, this step helps to narrow the field of possible matches to the candidate image. The authors in [34] then go on to describe how the fingerprint is separated from the background by use of whole-image mean and 4x4 block level variance calculations After this, the authors describe how they obtain the maximum curvature point and use that as a focal point for later calculations using Gabor filtering. This will create a vector and is compared to the vectors generated from the database set of possible matches. Finally, the authors in [34] try to improve the accuracy of their work by considering the orientation of the fingerprint when processing it to extract individual points of interest. Specifically, they calculate the block orientation field, then the direction of each block in the orientation field using a sine trigonometric operator, and then they calculate what they refer to as the density and storage

The authors in [32] take on the difficult problem of recognizing moving occluded vehicles in traffic. These researchers focused on the convexity present in these objects and developed an algorithm that creates a dividing line to determine where two occluded vehicles are positioned with respect to one another. The work is able to detect partially and completely occluded vehicles and classify them into very broad categories. The authors start by using an object segmentation algorithm given by [13], which they found to be an efficient and accurate extraction algorithm. Next, the authors in [32] consider partially and full occlusion separately. In the partial case, detection is applied to extracted component and depends on the fact that a partially occluded vehicle does not exhibit as much concavity as does an unoccluded or fully occluded vehicle. So from this, the ratio of the object and region's convex hull is computed and subject to a Bayesian test of probabilities that the vehicle is occluded verses it is not occluded. The researchers assumed the probabilities are Gaussian, which is a typical distribution to apply in a network system like automotive traffic. To minimize misclassification of occlusion, the authors in [32] use a Bayes Minimum Risk (BMR) classifier to determine how risky a given classification is because an incorrect classification is treated much like a cache miss in a memory, it takes time to retrieve the correct piece of information and the first guess is wrong. Once the detection is complete, [32] resolves the occlusion. In the partially occluded case, the end points of the dividing line are used to remove the line itself and create two separate regions that correspond to the two vehicles. The vehicles are now separated from one another. How were the end points of the dividing line determined? [32] takes the difference between convex hull area and the object and then select the two largest portions. Next, they compute how far

each vertex of two polygons are from the complex hull and select the two points that have the most distance (one from each side horizontally). Finally, the partial occlusion can be classified as representing a large vehicle, a small vehicle, or a motorcycle using the method described by [35]. In the fully occluded case, the authors in [32] measure two components of the occluded vehicles: their widths and the ratios between the length and width of the vehicles. To get the width, pixel counts of each row of the bounding box is taken (called the horizontal histogram), each value is normalized against the ratio of the line width and the center of the line of the vehicles, and the average of the normalized values is used as an estimate of the vehicle's width. The ratio calculation discussed earlier by [32] was derived from the length of the vehicle's bounding box and the non-normalized horizontal histogram. In this instance, the vehicles are classified as large vehicles, small vehicles, motorcycles, fully occluded vertical vehicle, or fully occluded horizontal vehicle based on threshold values that are derived from a two or three class Bayes Minimum Error (BME) classifier depending on the category test.

The authors in [32] carried out an experiment using a sequence of 3,480 352x288 images captured by a low hanging CCD camera. They noted that based on the data that their work yielded effective results in terms of classification even if the processing didn't always produce clean results. The fully classification worked as intended and kept incorrect classification from occurring for long enough to allow the partial scheme to take effect as the vehicles separated from one another in traffic.

It is hard not to see the connection between detecting incomplete objects and detecting and tracking occluded objects, particularly more complicated forms such as individuals. By

focusing on the research done in computer vision on occluded subjects we can learn more about how the design, implementation, and even experimental methodology of detectors can have a significant impact on outcomes. In [60], the researchers proposed a set of double and joint person detectors that can handle far higher levels of occlusion than the state-of-the-art single person detectors in the computer vision area. [60] proposed that instead of treating occlusion as a negative characteristic to be avoided in the formulation of a person detector (e.g., they only treat the visible part of the person that can be found), they chose to use that information to their advantage by noting that multiple person occlusion have very a particular pattern to them (e.g., occlusions are evidence of multiple individuals adjacent to one another). The patterns are distinct enough to take advantage of in build a detection model. The state-of-the-art detectors, though great at handling varying imaging conditions, individuals with different poses, and overall appearance, tend to fail off in terms of their performance under anything more than the most minor levels of occlusion and that schemes intended to handle this situation require tracking and 3d approaches that require analysis over a large number of frames.

The authors in [60] started by building a double-person detector before moving on to design and implement a more general joint-person detector that is a composition of the single and double-person detectors. The researchers also demonstrated how their approach could be used in a tracking system to detect and track occluded individuals. The base of [60]'s double and joint person model is based on DPM approach of [29]. The researchers note, "Each component is a star model consisting of a root filter that defines the coarse location of two people an n deformable part filters that cover presentative parts and occlusion patters of the two people,"

(pg. 60). The difference in this work is in the body of occluded samples used for training and providing information needed for detection (e.g., the occlusion information in a given frame is not thrown away). The 1,300 synthetically generated, silhouette based training samples are both positive and negative ones with occlusion levels from zero to 85% and scales from 90 to 110% of normal and the training process itself is an attempt to learn a set of model parameters by solving a quadratic program through stochastic gradient descent and data-mining of certain negative examples that optimizes for a set of latent parameters. The improvement to the model of [29] is to use a finer grained penalty model at the component level rather than the vector level. The experimental study involved the capture of 850 double person images at varying occlusion levels. [60] found their double-person detector to provide very good results in terms of both recall and precision over the single-person detector particularly at modest to severe levels of occlusion with inferior results only at the lowest levels of occlusion.

The natural progression from the double-person detector was to build a joint-person detector that is a combination of the single and double-person detector. The key was to present these differentiated detectors as components of the base DPM mode from [29]. Specifically, [60] built this detector using three single-person and three double-person detectors. They allowed for intra-class, but not inter-class reassignment of training samples. To allow for equivalent comparisons between single and double-person hypothesis, all the components were optimized at the same time. The training data, as to be expected, was a mixture of single-person and double-person samples with occlusion level of five percent or less assigned to one of the single-person components given the single-person stronger performance at such occlusion

levels. The experimental data here came from the TUD-Pedestrians and the TUD-Crossing datasets introduced by [2]. In the TUD-Pedestrians occlusion-free dataset, the double-person detector did not do well here compared to the single-person detector, but the joint-person detector did perform better than the single-person detector because of the training set inclusion of certain examples. So, the joint-detector may not have done better with the samples, but it didn't hurt the single-person detectors in this composite detector to also have the double-person detectors included. In the mixed occlusion and occlusion-free TUD-Crossing dataset, [60] found that each type of component performed best with data most strongly suited for its type as expected, but the combination of both types of components in one detector lead to superior results. At higher precision levels, the researchers found that this detector struggled somewhat because both classes of components were activated at the same time. In addition, this detector outperformed the Hough transform based detector of [6] with less training data compared to the competing approach. Finally, this approach can be used in tracking applications, but is beyond the scope of this work.

An interesting and earlier approach to occluded human detection and segmentation was undertaken by [65]. The method presented by these two researchers depends on the use of classifiers for both detection and segmentation activities that are largely independent from one another. This means that to use this approach for different classes of objects not only means a new offline training session must be performed with a different partitioning scheme, but that the methodology for forming these classifiers may need to change depending on the subject matter. This is particularly the case for subjects that are not humans or cars where an overhead,

downward sloping camera angle is not appropriate. The authors in [65] used the classifiers in a hierarchical part based system that formed a whole-object model. The classifiers in this scheme were classified as weak and required various augmentation procedures to be added to increase accuracy. The handling of any occlusion by the authors in [65] was done only when multiple objects were detected and their silhouettes had been extracted during a segmentation process. A joint probability technique was used to determine where the multiple occluded individuals were located in the image. The technique by [65] did not consider the use of other attributes like texture or color.

As mentioned earlier, the authors in [65] used a two-stage process with both detection and segmentation activities in each stage. The first-stage, which occurs offline, is a training phase. In this phase, training data is supplied so that part and segmentation detectors can be learned to form a whole-object segmentor that is enhanced by local shape features (e.g., various body parts). The part detectors and segment classifiers are weak classifiers that according to [65] involve the use of edgelets for part detectors where a mapping occurs between a location in the image space and a probabilistic value of the detection of an object from the class of interest and binary masks serve as a two-class mapping of pixels to objects during segmentation. The whole-object segmentor with part-hierarchy, which is a refinement of their CBT algorithm in an earlier work so that it can handle occlusions and human subjects more efficiently, is then supplied during the online detection and segmentation phase against a series of input images whereby image edges are extracted from objects and simplified with a clustering algorithm as needed. The segmentation, whether done during offline or online stage, is refined with double

application of erosion and dilation operators to remove noise and complete fill in objects with gaps. The goal during the online stage is to iteratively form hypotheses on where the individuals, whom may be occluded by one another to varying degrees, are in the picture and then after performing pixel-level segmentation on the positively contributing pixels of those components handle any occlusion by forming joint probabilities. The probabilities are formed using edge and part responses using silhouettes. Finally, the system outputs which hypotheses are most likely to show individuals. The authors in [65] uses weights that during the hypothesis forming that increase for successful part detection and decrease for false detections and missed detections.

In the experimental phase, the authors in [65] trained the detection part of the algorithm on 5,000 samples with humans from different angles and 7,000 other images without humans. The researchers used a very low level of between 10-5 and 10-6 for an acceptable false alarm rate. For segmentation, 1,800 images with human samples where labeled with bounding boxes and a two-pixel width exclusion zone. From this, 1,440 samples were used for segmentation training and 360 samples were as testing subjects. This process created 800 edgelet features and 500 weak classifiers, but the authors in [65] noted no measurable over-fitting, a 94.6% successful segmentation rate, and shape detection rate of 86.8%. The trained system was then used on two datasets from [64], and [18] entitled “USC pedestrian set B” and “Zurich Mobile Pedestrian Sequences” respectively. The researchers found their approach to be superior to other state-of-the-art methods that were compared in their work for both data sets. In the USC dataset, the authors in [65] had a detection rate of 96.0% and a segmentation precision and recall rate of 81.25% and 85.61% respectively. In the first data set, the images

were 384x288 and took an average of 3.6 seconds/image to process on an Intel Xeon 3.0 GHz processor. In the second data set, the 640x480 images took about 2.5 seconds/image to process. The performance on the Zurich set was lower because of the busy backgrounds in the images.

COMPARISON OF APPROACHES

This survey will use the survey comparison technique found in [39]. The technique analyzes each work from the role of the user, the developer, and the manager. From this analysis, a ten-point value is assigned to thirteen different aspects of the work. The values are weighted according to a survey the authors in [39] gave to ten product managers, ten software developers, and ten users with industry experience. It is necessary to modify portions of this analysis to align with the subject matter in this survey as the authors in [39] dealt with scan techniques in assistive technologies for the blind. The categories are presented in table one and the weights are presented in table two.

Table 1: Comparative aspects as defined in [39]

Acronym	Description
A	Availability — The ability to obtain/implement the system based on the description of the method expressed in mathematical formula, pseudo-code, or compiled code. A higher score indicates that a satisfactory amount of information is presented in the description to replicate the system. For example, a system with a score of 10 will contain a clear description of the method and C-code that could be implemented; whereas a system with a score of 5 may only have a mathematical formula and short process description.
Co	Cost — The amount of money needed to use and/or implement the system based on the description provided. This score reflects the cost of equipment as well as implementation complexity.
FI	Further Improvements — The methodology has the potential for further enhancement. A higher score indicates that a methodology can be improved upon, whereas a system with a lower score is considered more mature and less likely to be improved upon.
MC	Model Complexity — Complexity of model used in the methodology. For example, a system utilizing a neural network or wavelet is considered more complex than one that uses a run length smoothing algorithm.
O	Originality — The methodology is based on original algorithms and/or mathematical operations; or the synergistic combination of simple methods composing a new method. A method that is referenced in the literature as original is given a higher score than one that builds on another method.
P	Prototype — The methodology has been successfully implemented at the experimental stage and produced desirable results. Scores for this aspect were also affected by the

	results presented. A paper that presented comparative results scored higher than one that presents an illustrative example.
RP	Released Product — The methodology has been implemented in a commercial setting. This aspect has a value of either 1 or 3, where the few methods/systems that have been utilized in a commercial setting are given a slight advantage over others.
Re	Reliability — The methodology produces expected results under normal operating conditions.
Ro	Robustness — The methodology produces acceptable results under extenuating circumstances. This score is based on the features of the methodology as compared to methodologies in a similar category.
Sp	Speed — Reported processing time for sample tests. Note that some authors do not report performance metrics. For these we give a score of 3 out of 10.
U	Usability — The methodology offers a user-friendly interface so that the user can work easily with it. Systems that require no user input are given a higher score than those that require input parameters or training data.
Cl	Closeness – Closeness to the technique described in [53].
M	Maturity — A measure that combines the scores of the different aspects. $Maturity = U + O + (A * P * RP) + \frac{(Re*Ro*Sp)}{Co*FI*MC}$

In table one, the maturity component is an overall scoring of the method and therefore the most important component. The authors in [39] noted that the maturity formula grouped positive and negative aspects together. Therefore, the readiness components of available (e.g., reproducible), prototyped, and in a released product and brought together as a cumulative weight as are the characteristics of a working system such as reliability, robustness, and speed. The cost, need for further improvements to be useable (particularly as it relates to closeness), and complexity are aspects that work against having a mature product. The usability and originality of a work are aspects that stand apart, but usability is an aspect that is more universally important than is originality as we will observe in table two.

Table 2: Aspect weights (avg. of perspectives divided by 10)

	A	Co	MC	O	P	RP	Re	Ro	SP	U
Manager perspective (Mp)	7.2	7.2	5.9	4.3	3.4	7	9.3	8.1	8.6	9.8
Developer perspective (Dp)	5	6.9	3.6	3.5	3.9	7.2	9.1	8.7	7.1	8.8
User perspective (Up)	9.8	9.7	1.3	1.6	2.1	9.8	8.7	9.1	8.9	9.7
Aspect weight (Mp+Dp+Up)/3	0.73	0.79	0.36	0.31	0.31	0.8	0.94	0.86	0.82	0.94

The authors in [39] established that the scoring of an aspect for a given perspective depends on an assessment of whether the criteria was not meet at all (1), it was meet somewhat (4), it was mostly meet (7), or it was completely meet (10). The background papers in the initial section and the paper by [4] in the Geometry Based section will not be scored as they provided insight to the works considered in their respective subsequent portions. The scores for each area can be seen in table three:

Table 3: Weighted aspect scores from Survey Review

	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]
A	1.46	4.38	4.38	1.46	3.65	2.92	0.73	3.65	5.84	2.92	2.92	2.19	3.65	2.92
Co	5.53	0.79	3.16	5.53	3.95	3.95	6.32	4.74	7.11	4.74	4.74	5.53	7.9	5.53
FI	5	8	6	6	8	8	5	7	7	5	4	4	5	1
MC	0.72	2.88	1.44	1.44	3.6	2.88	3.6	2.52	3.6	1.44	2.16	2.52	2.88	2.16
O	0.31	0.93	1.86	1.24	2.79	2.17	2.79	2.17	3.1	1.55	1.55	1.55	2.17	1.86
P	1.24	2.48	1.24	1.24	3.1	1.24	1.55	1.24	0.31	1.24	1.55	0.31	2.17	0.93
RP	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8
Re	6.58	6.58	6.58	6.58	5.64	6.58	4.7	7.52	6.58	5.64	4.7	4.7	6.58	3.76
Ro	3.44	3.44	8.6	6.02	3.44	6.02	5.16	6.88	6.88	5.16	4.3	4.3	6.02	1.72

Sp	2.46	5.74	8.2	2.46	1.64	1.64	0.82	3.28	2.46	6.56	2.46	2.46	0.82	2.46
U	0.94	1.88	0.94	0.94	0.94	6.58	8.46	0.94	0.94	0.94	0.94	0.94	3.76	0.94
Cl	4	4	4	7	1	1	1	1	1	4	1	1	1	1
M	37	55	59	46	61	60	60	58	65	50	45	42	62	38

	[15]	[16]	[17]	[18]	[19]	[20]	[21]
A	2.92	1.46	2.92	2.92	2.92	2.92	5.11
Co	3.16	3.16	0.79	0.79	6.583333	2.37	5.53
FI	5	6	1	2	7	5	3
MC	2.88	1.8	1.08	1.08	2.52	2.52	2.52
O	1.24	1.24	1.24	0.31	1.24	2.17	1.55
P	2.17	1.24	0.31	0.62	3.1	2.48	0.62
RP	0.8	0.8	0.8	0.8	1.6	0.8	0.8
Re	5.64	3.76	6.58	9.4	6.58	7.52	6.58
Ro	5.16	3.44	3.44	8.6	3.44	6.02	6.02
Sp	4.1	2.46	2.46	2.46	2.46	2.46	7.38
U	0.94	0.94	0.94	0.94	3.76	0.94	3.76
Cl	1	1	1	1	3	1	4
M	51	38	30	38	60.33333	54	59

Table 4: Paper Lookup Table from Survey Study

[1] Zheng and Peng, 2013
[2] Wu, Lim, & Tan, 2012
[3] Dinesh & Guru, 2010
[4] Bhattacharjee, Das, & Dutta, 2008
[5] Xiao, Li, Xie, Tan, & Mao, 2015
[6] Hao & Wu, 2012
[7] Hays & Efros 2007
[8] Li, Zhao, Xu, & Lu, 2013
[9] Zhong, Liu, Chung, & Wu, 2012
[10] Lowe, 1999
[11] Kerr, Scotney & Coleman, 2008
[12] Yamaguchi & Fukui, 2002
[13] Xue, Rubinstein, Liu, & Freeman, 2015
[14] Abolbashari, 2012
[15] Zhang and Zhang, 2007

[16] Guo, Jiang, Wang, & Gao, 2012
[17] Shin, Zheng, Russell, & Klette, 2012
[18] Hong, Xueyan, Shuxu, & Hua, 2010
[19] Heidari & Ahmadzadeh, 2013
[20] Tang, Andriluka, & Schiele
[21] Wu & Nevatia, 2008

The enumerated values for each work in the previous list apply only to the tables. Enumerated values that are discussed in the text referred to the References section enumerated values

There are several different ways to look at the qualitatively accessed data. One method is to look at the overall average and deviation of the scored methods. The data is presented in the next chart

Table 5: Weighted averages and standard deviation of scores for scoring of survey evaluated papers

	Average	Std. Dev	Possible Max
A	3.06	1.21	7.30
Co	**4.38**	**1.99**	**7.90**
FI	**5.14**	**2.05**	**10.00**
MC	2.30	0.83	3.60
O	1.67	0.72	3.10
P	1.45	0.83	3.10
RP	0.84	0.17	8.00
Re	6.13	1.28	9.40
Ro	5.12	1.75	8.60
Sp	**3.20**	**1.99**	**8.20**
U	**2.01**	**2.06**	**9.40**
Cl	2.10	1.69	10.00
M	50.87	10.13	n/a

Overall, we see some important trends here. Given the weights and maximum possible score, the cost, need for further improvement, speed, and sophistication of the user interface where the areas of greatest variation among the papers. In terms of availability, the deliverables from the various groups and researchers in this survey, for the most part, would be difficult to

implement in a system for purposes of comparative analysis or validation of results. The ability to quickly reproduce results is critical to being able to more quickly advance the state-of-the-art in a given field, more so in narrowly focused sub-domains. Given that, the results in visualized in figure 5 and show that most papers presented us with only some high level descriptions of the system, a few key equations, and/or pseudocode of select portions. However, there were some surprises from [70] and [65].

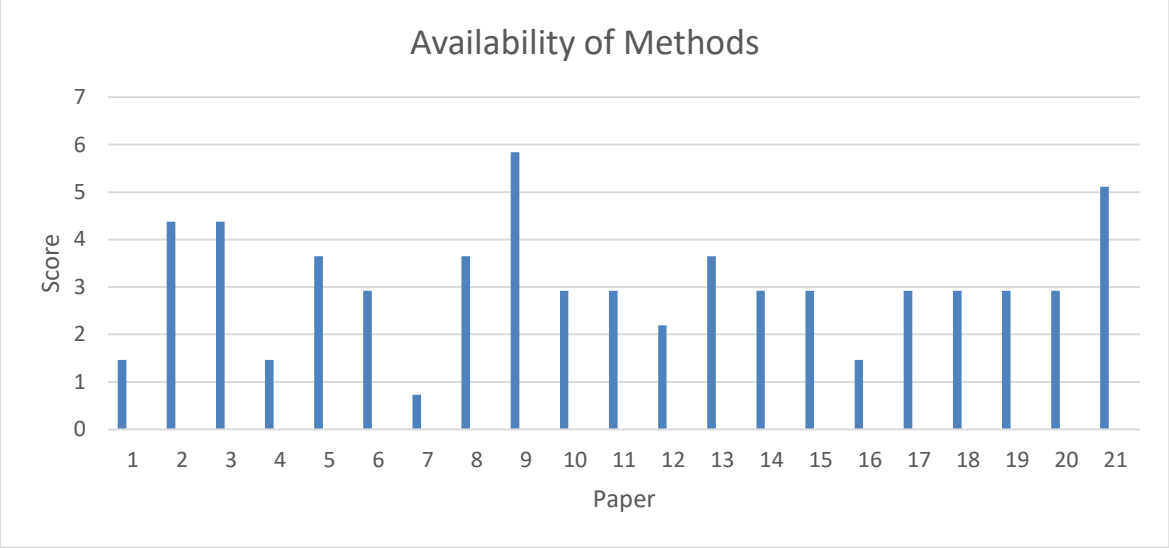


Figure 6: Availability data from Survey Review

Beyond availability, the output from the various groups of researchers seem to have been generated at moderately-high costs and seems reasonable to conclude that it would cost approximately the same to recreate a similar system in the future provided a full description of the system is available. The results for cost are visualized in figure 7. However, it might cost significantly less in the future given the performance/cost reduction curve seen in computing over the past number of decades or the economies of scale that can occur when productizing a

method. However, there were certainly outliers in some of the works. The work by [57] and [34] just needed a scanner and simple physical materials for recording the data. The software also did not seem to need particularly expensive systems to provide reasonable performance and development costs would seem to be lower given the lower originality in those papers. However, the work by [71] needed a high-end Xeon multi-core workstation with a considerable 64 GB of RAM (there were no details given on GPU utilization or type).

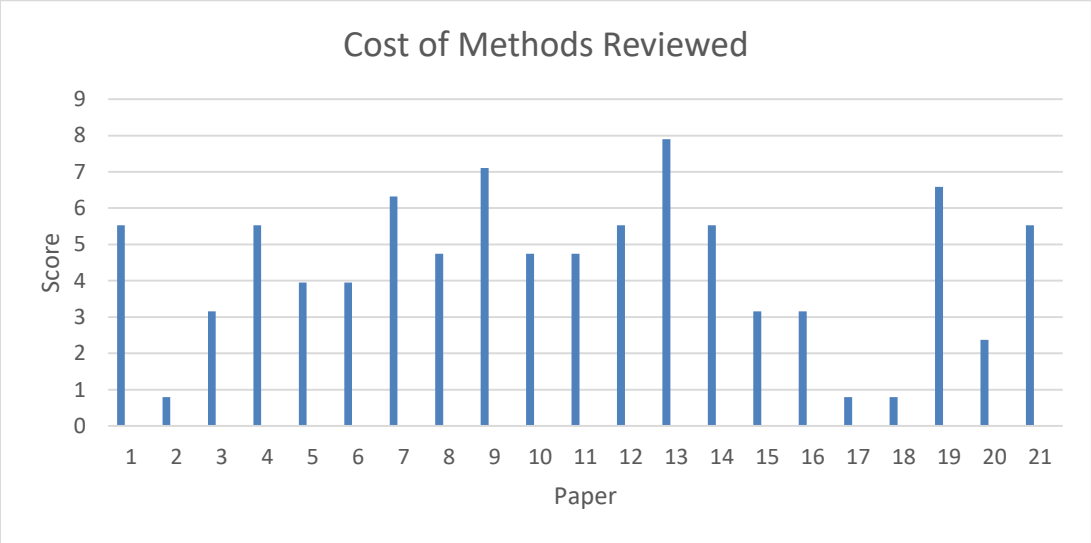


Figure 7: Cost data from Survey Review

Many of the works discussed in this survey also showed a moderate to high level of being improved upon in the future; although some more so than others. The results are shown in figure 8. The greater the amount of improvement that can be made, the less mature the method. In turn, this means the method described is less ready to be used in other researchers' efforts or in a production-ready product or tool. The works by [1] and [57] seemed to need the least amount of additional improvement. In [1], the methods described are present in other

systems that are commercially available – it is the combination of the two systems that creates a novel application. The work of [57], is a domain specific system that can be used as is to help scientists in other fields complete their studies. On the other hand, the works in the geometric based portion of the survey were only used on small datasets under a strict set of conditions and were not ready to serve as part of a larger system.

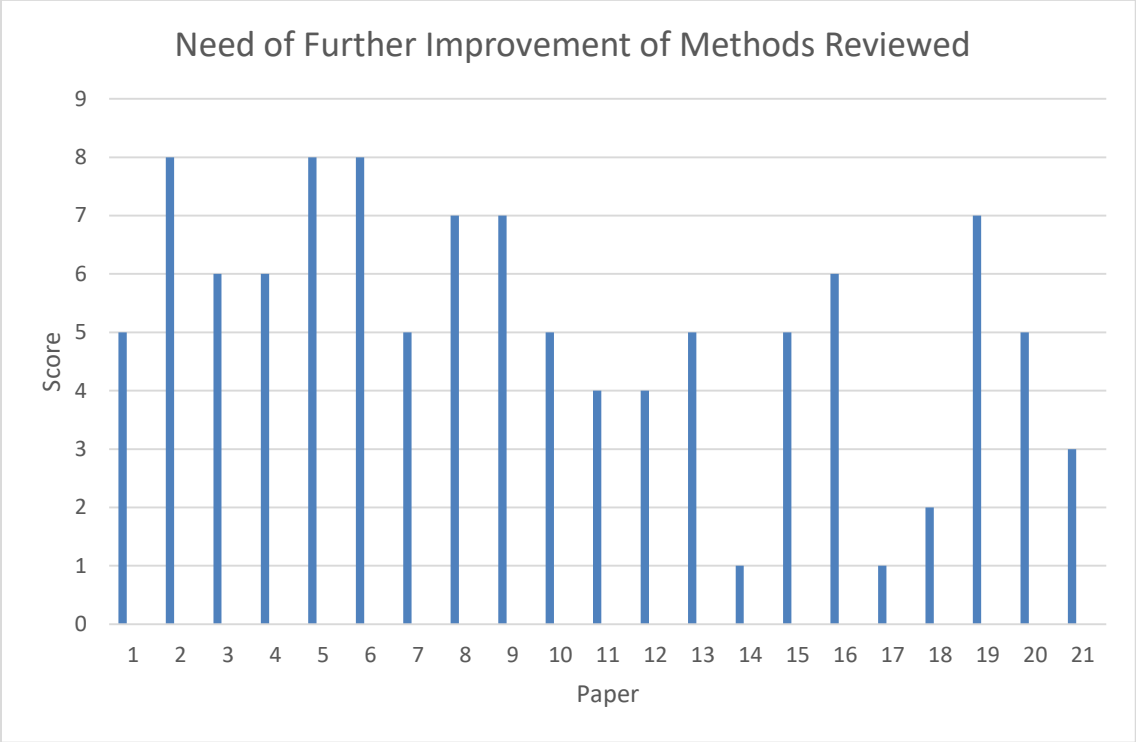


Figure 8: Further Improvement Comparison from Survey Review

It might have been surprising to see such low model complexity numbers, but there was a low weight attached to this category. Given the top possible score was a 3.6, a 2.30 value with 0.83 deviation shows moderate levels of complexity in the methods from various groups with a wide range of variability in overall complexity. The results for this category are shown in figure

9. The image completion and neural network methods tended to have higher levels of complexity than did the geometry based methods. However, a complex method may not always provide for a mature, capable system from the viewpoint of the various stakeholder groups identified in this paper. The more complex the model, the harder it is to understand, set-up, verify, maintain, and utilize in other applications.

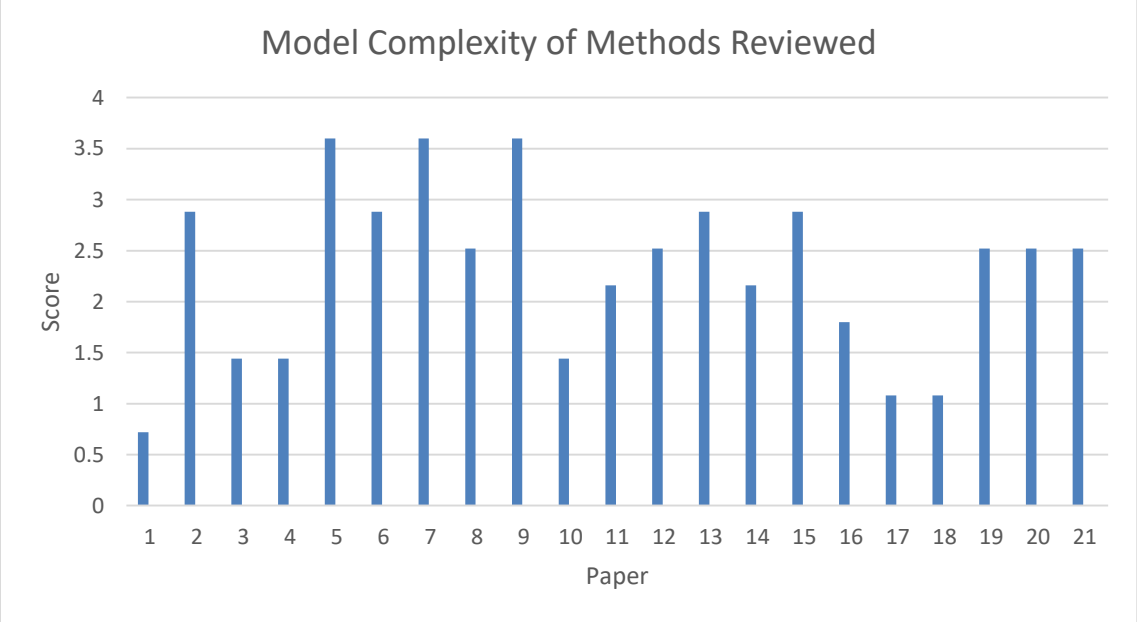


Figure 9: Model Complexity Comparison

Similarity, originality and prototyping success also had low weights as those factors are not as critical for long-term success if the method proposed is solid and a clear path to successful commercial implementation can be seen. In other words, the academic community may reward such aspects of basic and applied research, but the developers, managers, and users in the commercial world are not as strongly influenced by such early work. However, successfully prototyping can be seen as way to provide a clear proof-of-concept for a method,

particularly if the method is highly original. There was not enough detail in the paper to draw conclusions as to why some groups had better prototyping success than others. The results for prototyping scores are shown in figure 10 whereas the results for originality are shown in figure 10.

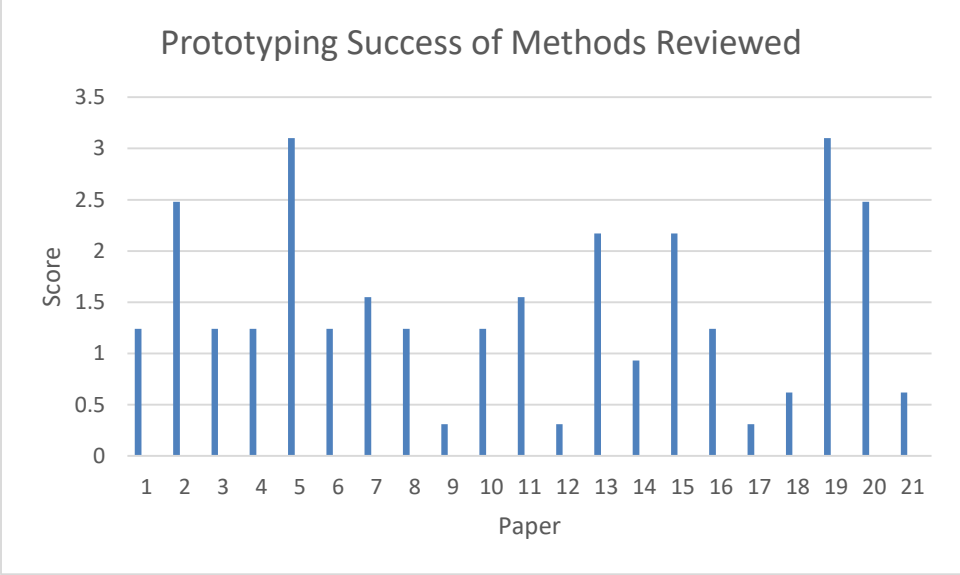


Figure 10: Prototyping Comparison

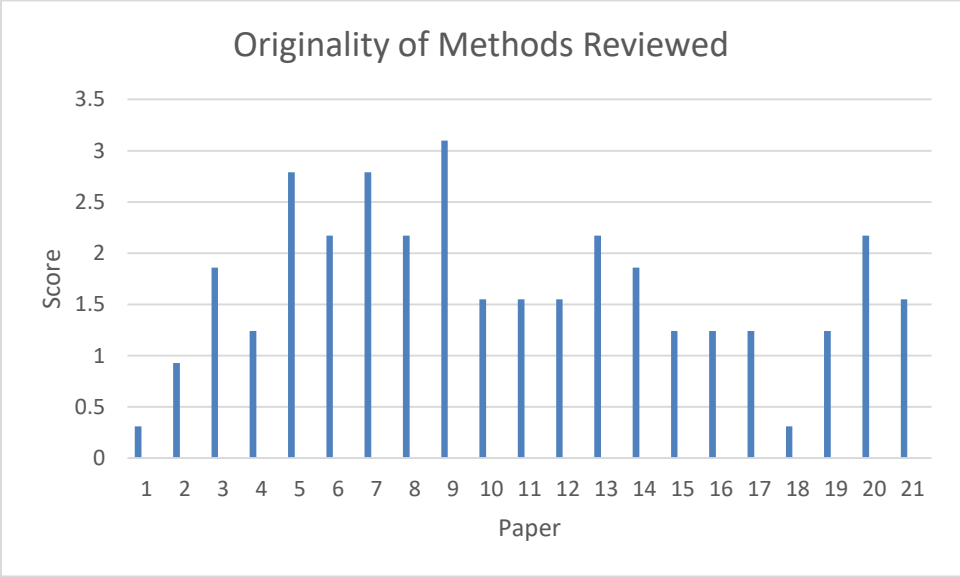


Figure 11: Originality Comparison from Survey Review

Similarly, the released product category had uniform low scores as all the submitted work came from researchers who were in the earliest phases of their researchers and/or did not show any evidence of moving their researchers beyond the proof-of-concept phase. There is no need to visualize this data as it provides no additional insights into the results presented here. Beyond these important considerations, reliability and robustness are two uniformly important characteristics among all three stakeholder groups. The results for these categories can be seen in the figure 12 below. For the most part, most researchers met the condition of having a reliable solution for all stakeholder groups, but their work had the remote possibility or actual occasional glitch of showing problematic results on a given execution run under expected considerations (or they did not rule out their approach going awry from time to time under

expected conditions). If the technique was weighted toward employing a stochastic approach, like [60]'s method, then this is understandable. Occasionally, there was a pleasant surprise with [34]'s group demonstrating superior reliability, but focusing on a narrow domain that will typical improve results as the problem search space is being restricted. This is not meant to diminish their work, but to acknowledge there are often tradeoffs when working on general solutions verses domain restricted solutions. The reliability and robustness of solutions to problems in a narrow domain will generally be better than solutions that are generic in nature because you can use domain knowledge in crafting the solution. The lower level of reliability in [1]'s work is evident of this tradeoff with his general approach to the problem. Now, with respect to robustness, the results were much lower across the board. This is expected given the early nature of the work presented by most of the groups reviewed in this survey. It's difficult in most cases to handle corner cases, problematic data sets, or all permutations of a problem statement when trying to establish through proof-of-concept a particular approach in solving a problem. [16] and [34], though, proposed solutions that had a high degree of robustness. In the case of [16], it was the generic geometric approach and efficient data structures that allowed for overall solid performance under a variety of different conditions. In [34]'s case, the solution was narrowly tailored to the fingerprint domain and employed several methods to improve the quality of data input and clean-up any possible artifacts in the data input to improve overall results.

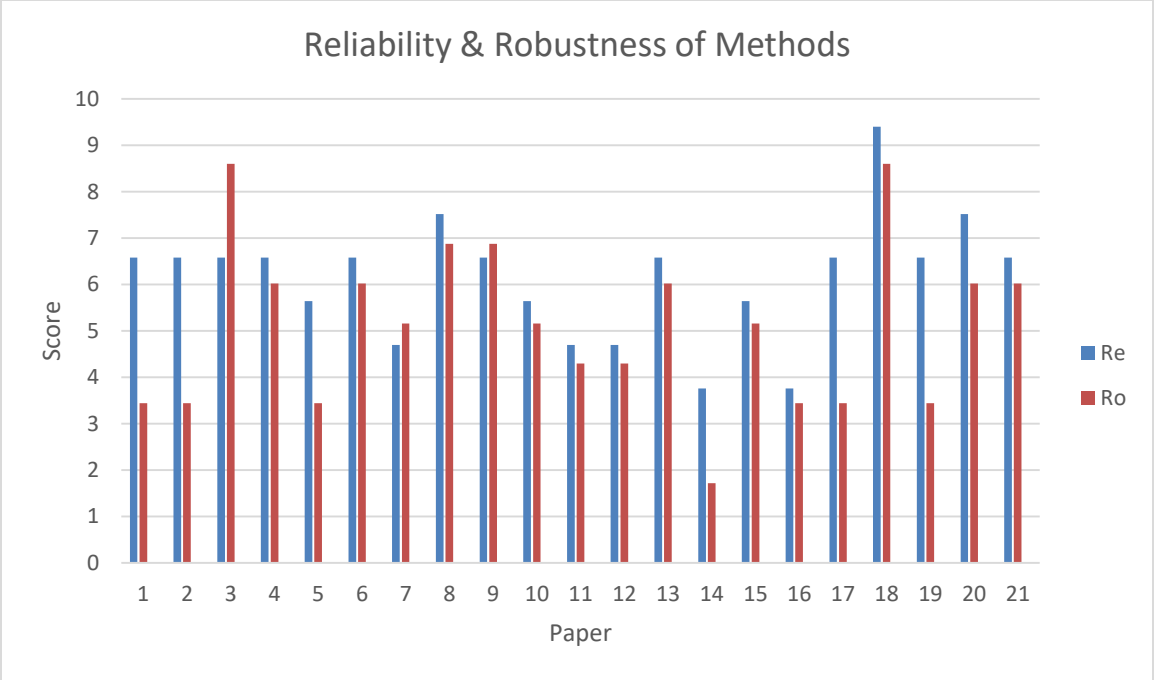


Figure 12: Reliability & Robustness Comparison from Survey Review

In an end-use scenario, speed is an important practical consideration. Given the basic and early applied research in the reviewed works, it's understandable that most methods might not have the best possible performance profile. In many cases, there was simply no performance data given as can be observed in figure 13 below showing the trending of the data around the weighted score of 2.46 to indicate no data was available. However, in the works by [69] and [44], we saw processing times in the range of seconds, which would work wonderfully in an embedded context. Provided the average and worse case algorithm performance of these algorithms are logarithmic to a low degree of polynomial, the performance should hold up under the weight of increasing dataset sizes. In the methods presented by [67] and [71],

performance was on the order of minutes using a fairly robust workstation for the timeframe in which the research was done. Given the scoring of these works in the *FI* category, it appears that there are moderate to considerable levels of improvements that would be made with additional research efforts. In many cases, the researchers did not focus on gathering performance metrics due to time constraints or a desire/need to focus on other elements of the research.

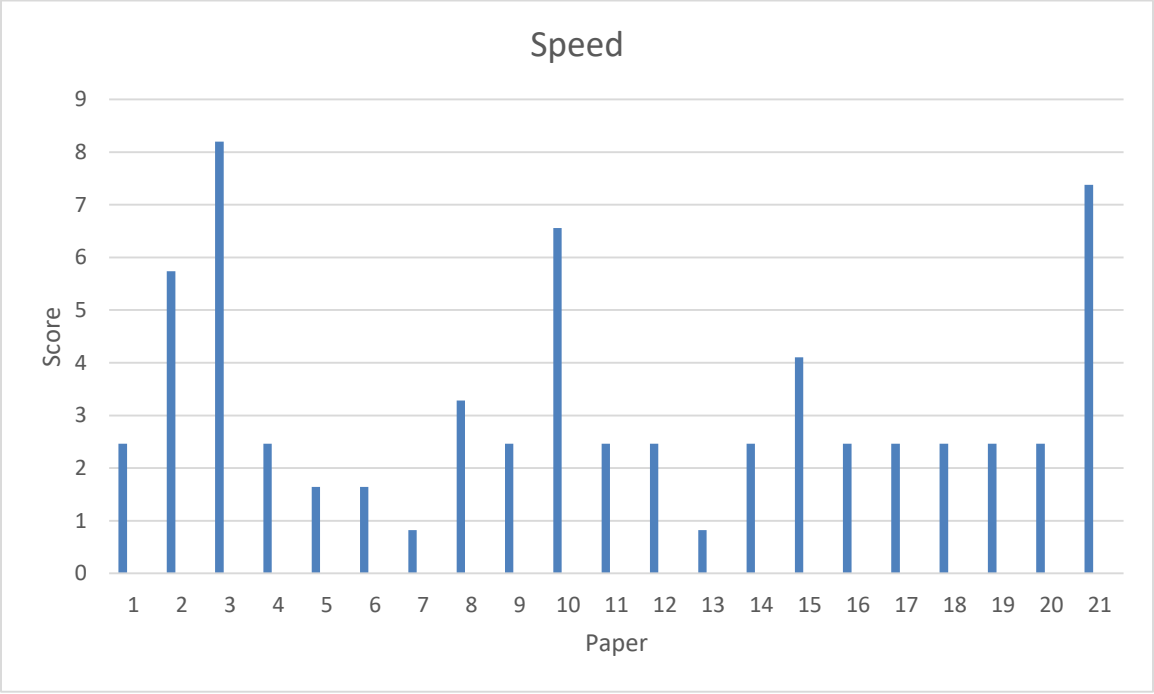


Figure 13: Speed Comparison from Survey Review

SUMMARY AND CONCLUSIONS

This survey has offered an overview on the various methodologies that dealt with the recognition of incomplete objects. It is reasonable to conclude that there is a great deal of work left to be done in this area. In general, solutions had poor reproducibility (e.g.., availability). The reviewed most likely would have moderate to high costs of implementation in a production environment. Only some of the solutions, which were moderately complex overall, would have a fair degree of malleability to changes. It was disappointing to see that a few approaches had claimed good prototype level runs in a testing environment, few were being used outside of academia. Finally, although most solutions are not close to the current work, the work by [8] is closest to the current work. It should be noted that maturity scores are neither good or bad. The more mature solutions give us a baseline to compare against. The more mature technologies are ready for transfer to the commercial sector. The less mature solutions provide an opportunity to investigate, compare and contrast to the current work, and improve upon for use in the current work.

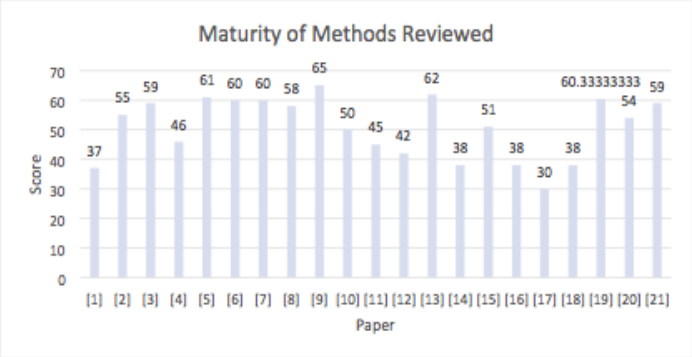


Figure 14: Maturity of Methods Reviewed for Survey Review

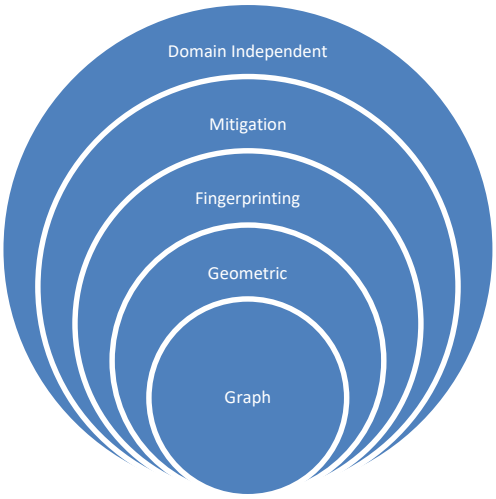


Figure 15: Layered Categorization of Current Method

The remaining chapters will look at a novel graph-based geometric approaches that are steeped in the use of computer vision and image processing and analysis techniques for handling incomplete objects. The computer vision techniques can help us to extract and categorize information from images for later analysis and the image processing components can help us to transform the image a priori to the extraction of important features useful in the identification of obstructed images. The nodes of these graphs will contain fingerprints of the components of the processed images. This approach serves as a search space reduction process that in conjunction with other approaches used in the preprocessing phase to mitigate inaccurate recognition of incomplete objects help to provide a domain independent approach that is different from other proposed techniques.

II. THE METHODOLOGY

In the first part of this chapter we briefly describe the overall methodology with emphasis on the novel parts of it. In the second part, we are dealing with details and the set of procedures needed to extract a set of candidate images for subsequent analysis and comparison against a set of exemplar model images with the goal of accurately identifying each sample image with a high degree of precision even in the presence of visible obstruction(s). A low probability match indicates an image without a close exemplar image and depending on the domain may be a suitable image for inclusion in the model set provided a clean set of images can be obtained at a future point in time.

A BRIEF OVERVIEW OF THE L-G GRAPH BASED METHODOLOGY

The overall view of this methodology for the recognition of incomplete objects based on their complete representation of their six views is graphically described in the following picture. In particular, We consider that the known objects in the database have only six views {left (l), right (r), top (t), bottom (b), front (f), and back (c)}. Each view is also associated with its own L-G graph model and Delaunay triangulation model. Each of these two model has its own features. For instance, the L-G Graph holds the representation of the local graph of its segment of an object-view, which consists of: the nodes, the connection points, the centroids of each local region, the size and the shape of the components that compose a region, and the global graph which holds the detailed structure of the regions connected through the centroids. On the other hand, the Delaunay model offers a lighter description of the global graph by connecting centroids of certain regions in neighborhoods.

THE L-G GRAPH

At this point we briefly describe the L-G graph method, since it will play the most important role of this research work. The following figures, 16 and 17, show two simple examples to better understanding of the L-G graph method used here.

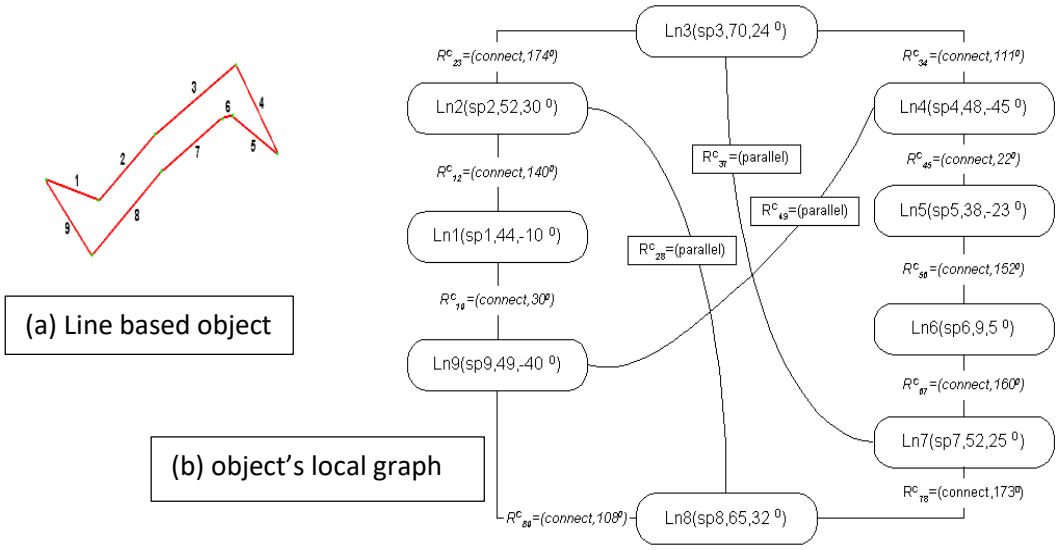


Figure 16: This is an example of local graph. Left image is the object or a single region. The number besides every line is the index. Its local graph representation is shown on right. For simplicity, only some fields of local graph are shown

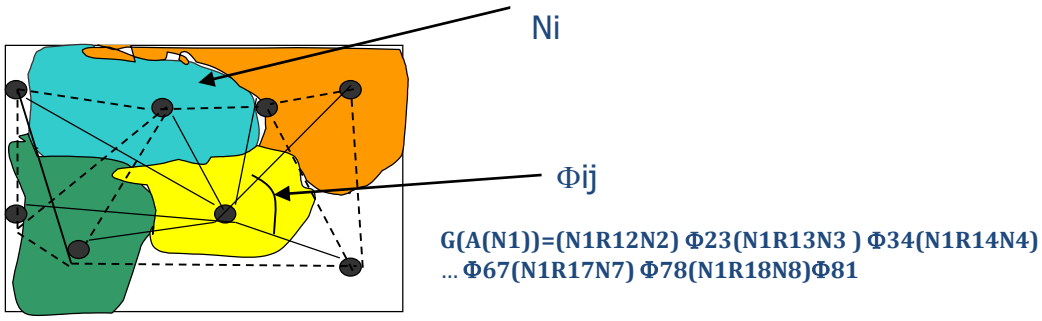


Figure 17: The Global graph is a structure that carries inter-region relations. Each node of this graph represents a region, i.e. it contains the respective local graph.

THE L-G GRAPH APPROACH FOR REPRESENTING INCOMPLETE OBJECTS

Now at this point we will graphically present the L-G graph methodology for the representation and recognition of the incomplete objects. In particular, each known object is expressed in six views in the database as mentioned above. Each view is expressed in an L-G graph. Then the unknown "image" x is also expressed into an L-G graph form (e.g., LG(x)). At this point, the unknown L-G graph is compared to each of the six L-G graph views of all the known objects in the database. The following figure shows graphically the steps of comparison.

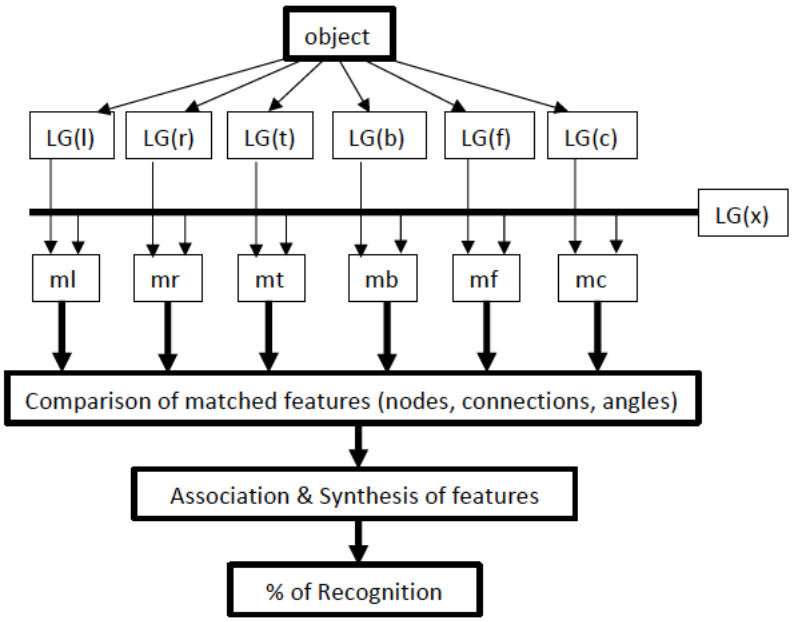


Figure 18: The graphical representation of the comparison recognition methodology

The matched L-G graph components (nodes, distances, angles) are unified into a new L-G graph model that represents the % of recognition of the incomplete view of the unknown object. Below we present an example of the methodology with a car.

THE STEPS OF THE OVERALL METHODOLOGY

In a refinement to the original methodology, we only consider the matching of centroids from the global graph along with its Delaunay derivation. The seven major steps in this process are as follows:

1. Take pictures of the six views of the object
2. Develop the L-G graph for each of them
3. Extract the L-G graph of the unknown object that is incomplete
4. Compare each L-G graph view against the unknown object
5. Remove any redundancies
6. Associate the matched parts of the L-G graph
7. Determine the percentage of recognition

At this place we present a detailed example in order to describe the methodology and show the way that works. For this purpose we present the six view of the object can be seen in the following contact sheet:

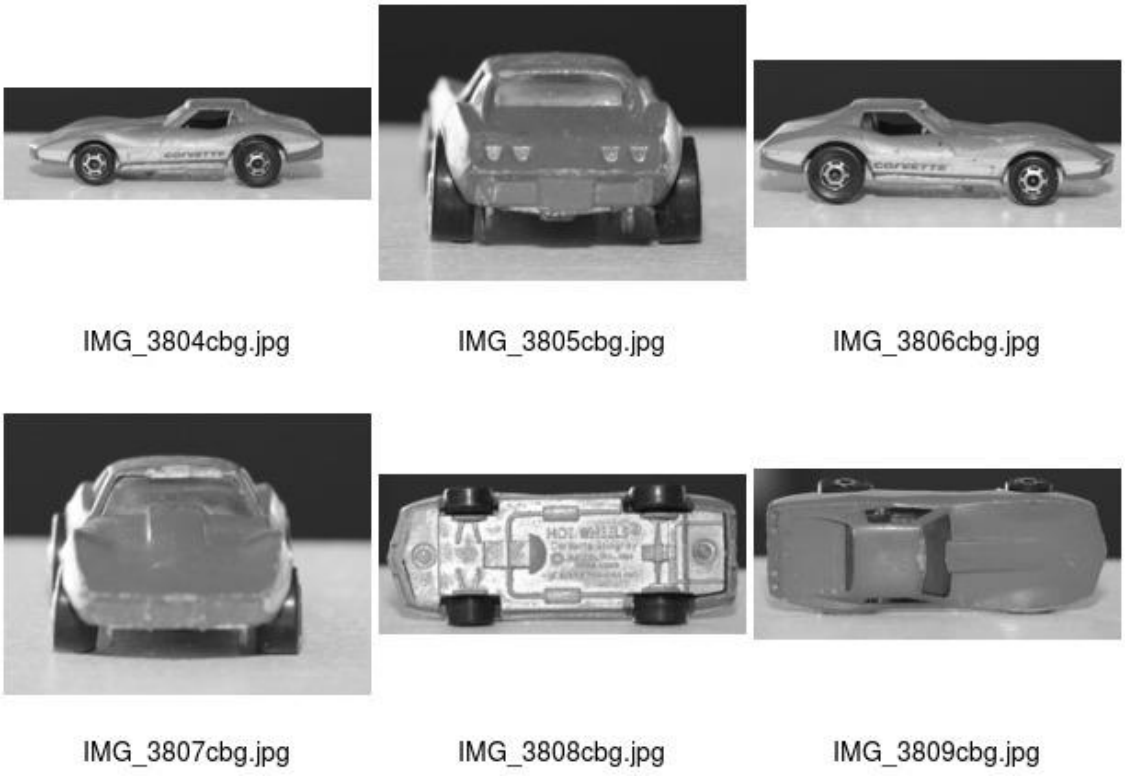


Figure 19: Multi-sided View of the Corvette Model Images

The preprocessing de-noising and sharpening operations are shown in the following contact sheet figures:

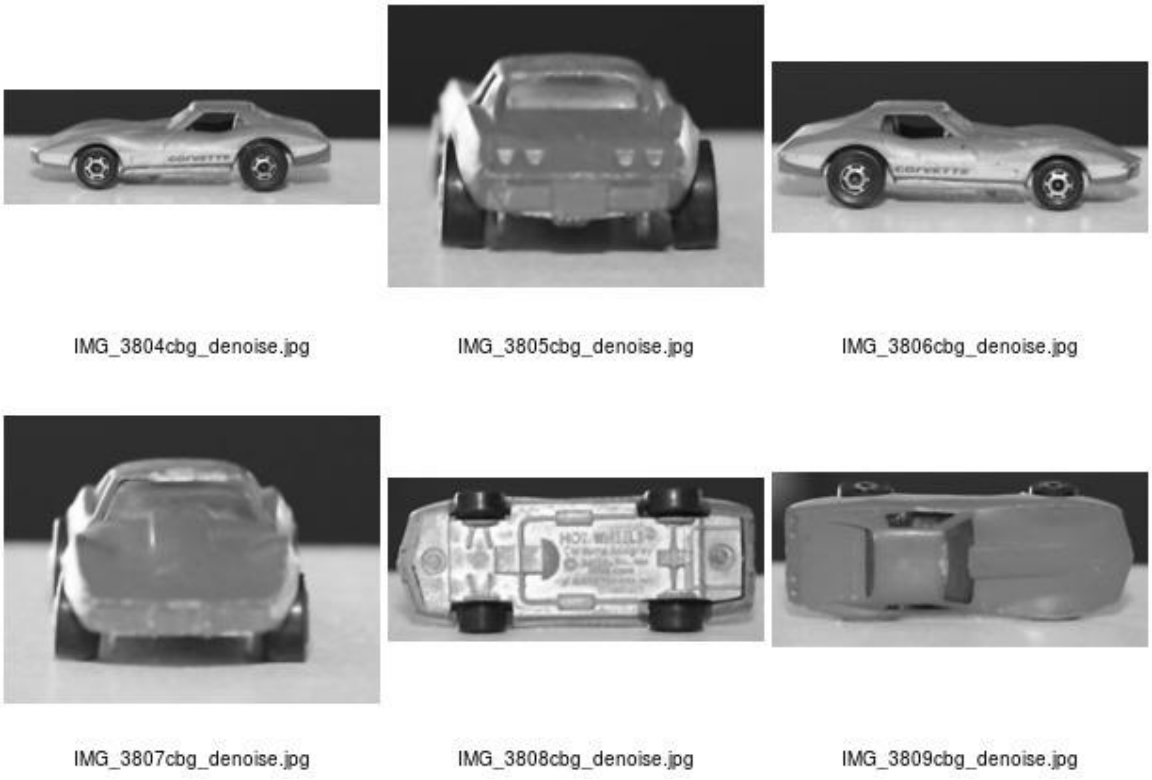


Figure 20: De-noising preprocessing Operator Applied to Corvette Model Images

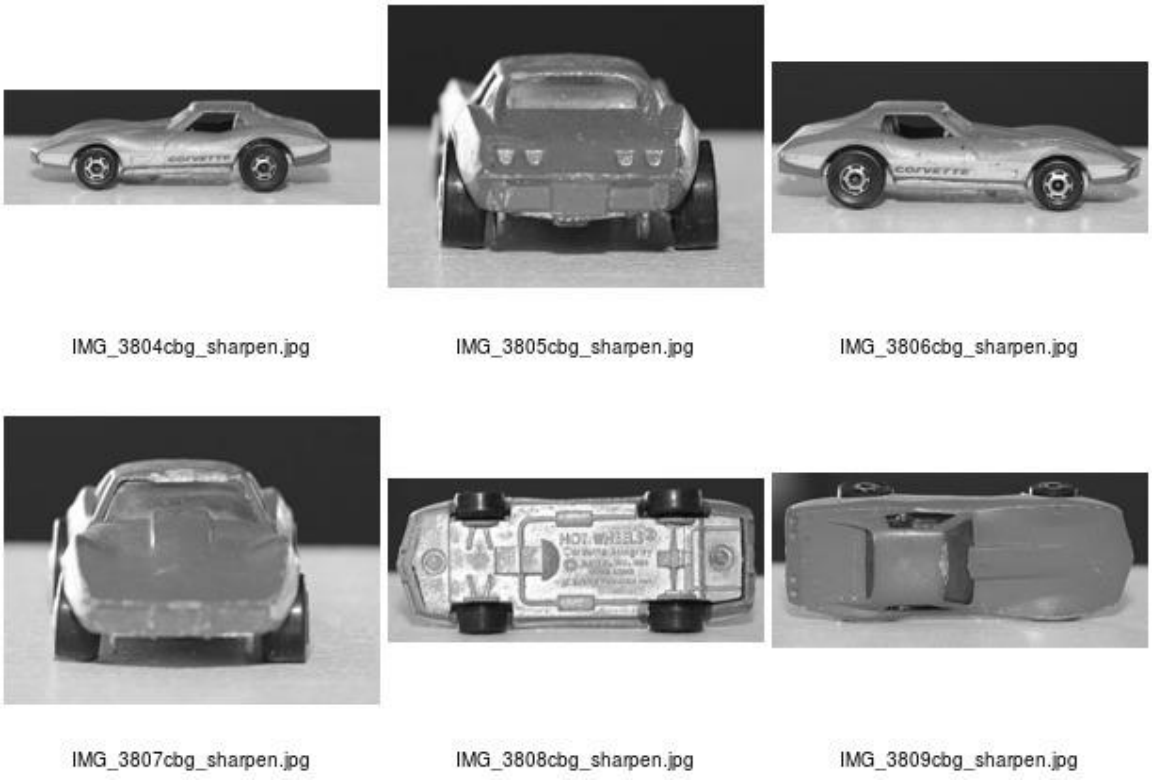


Figure 21: Sharpening Preprocessing Operator Applied to Corvette Model Images

After preprocessing, k-means segmentation of the model images took place specifying four clusters and at most sixteen iterations or pixel level convergence of centers, which generated the following result:

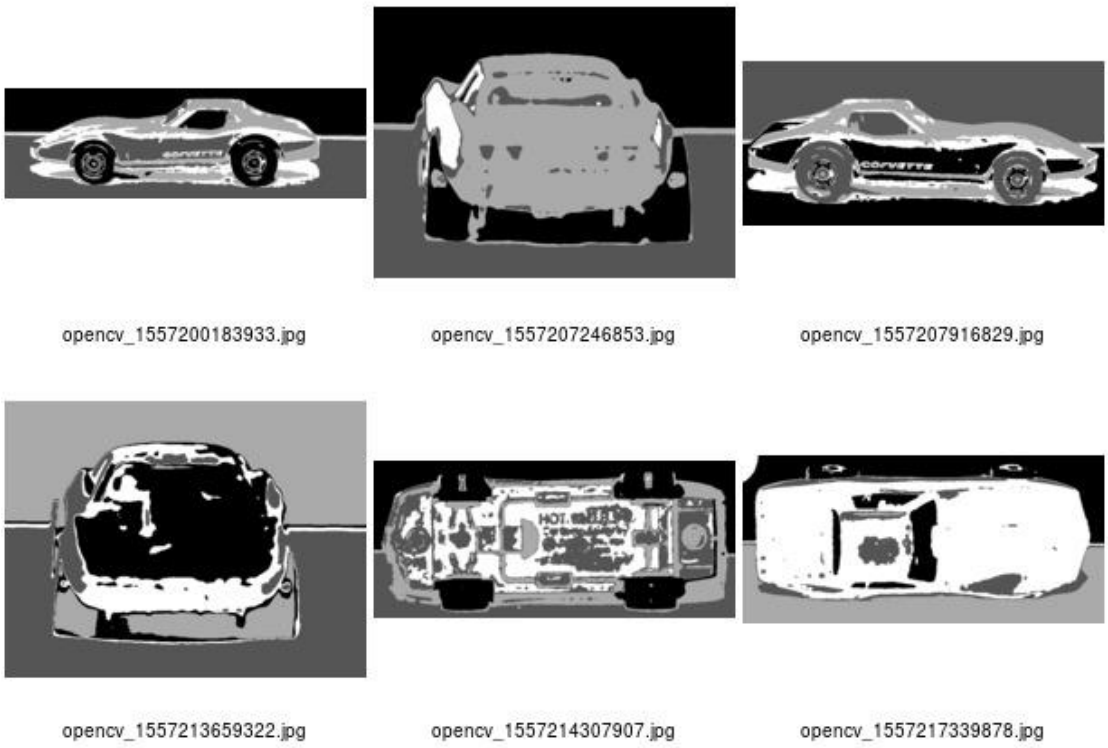


Figure 22: Segmentation of Corvette Model Images

After processing the model images with the L-G graph algorithm and deriving the Delaunay triangulation from the global portion of the L-G graph, the following results can be observed in the next set of figures.

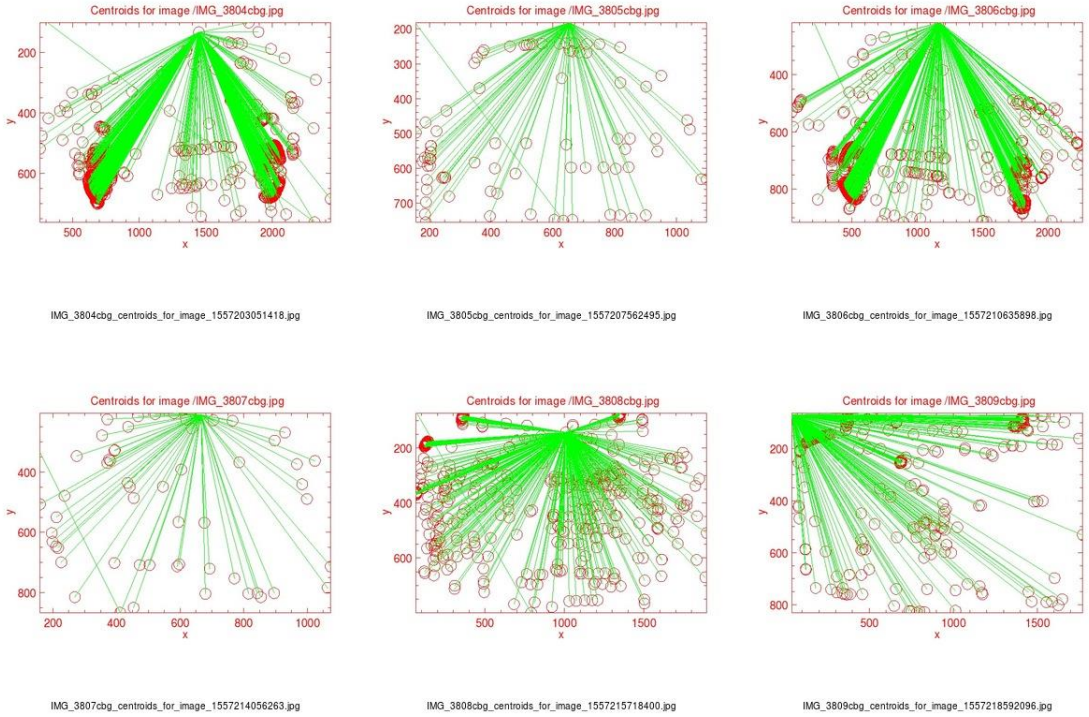


Figure 23: Global Graphs for Corvette Model from Mutli-view (Six Sides)

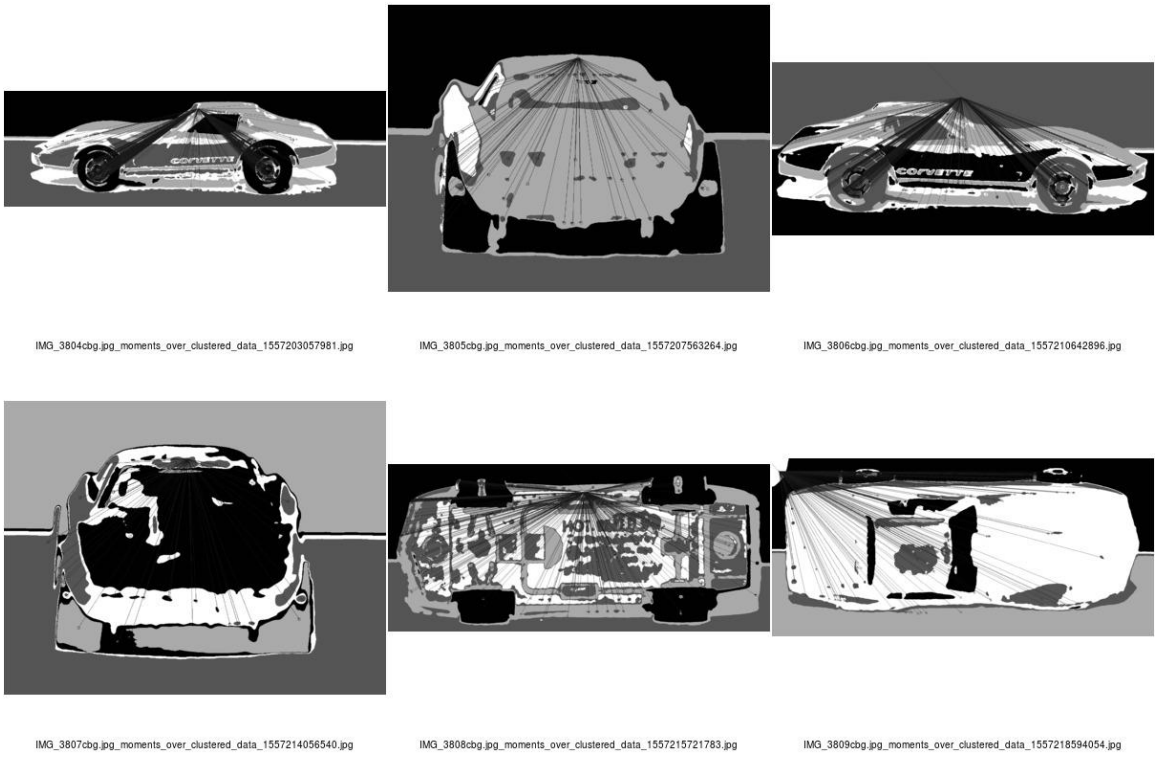


Figure 24: Global Graphs Overlaid onto Segmented Model Images of Corvette from Multi-View (Six Sides)

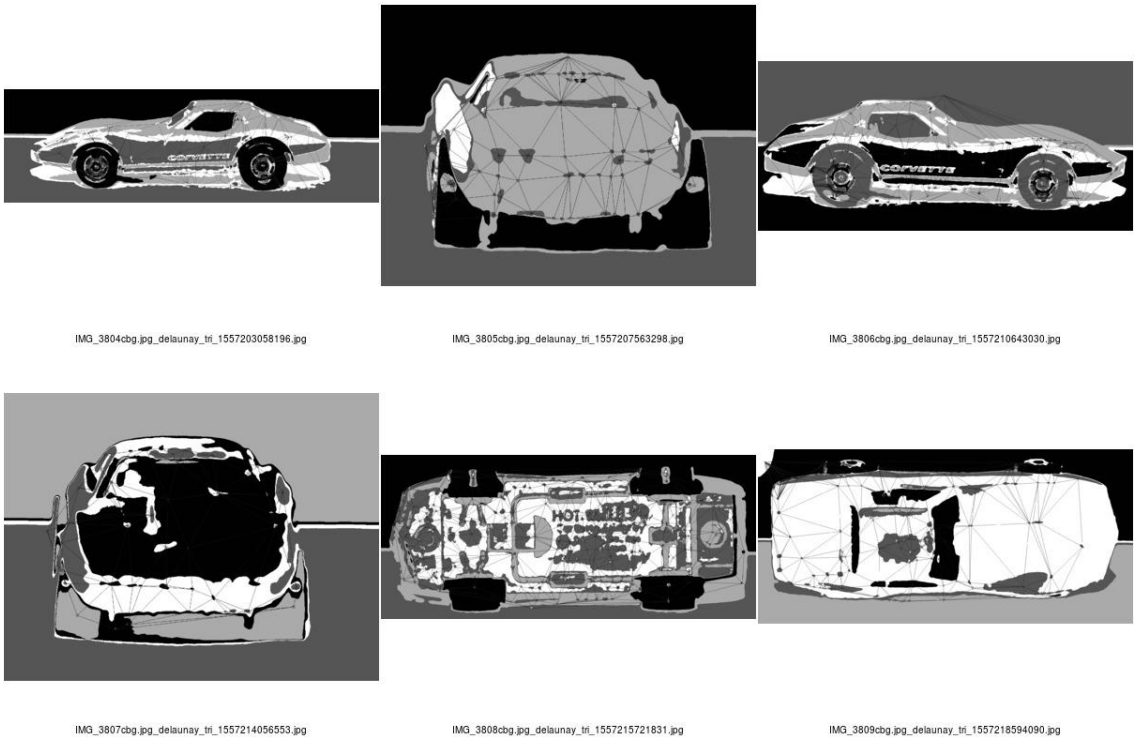


Figure 25: Delaunay Graphs Overlaid on Segmented Corvette Multi-View Images (Six Sided)

Next, we took an off-angle shot of the corvette as already shown in the Synthesis of Views under Incomplete Recognition. Following this, to simulate having an incomplete object, we divided the off-angle shot of the corvette into five slices as shown in the next contact sheet using the ImageMagick convert command line utility convert utility with the following command line: `convert -crop 20%100% IMG_3816_corvette_offangle_to_slice.jpg IMG_3816_corvettte_offangle_partial.jpg`. This generates five images with a "-<image number>" value appended to the filename as shown in the following contact sheet:

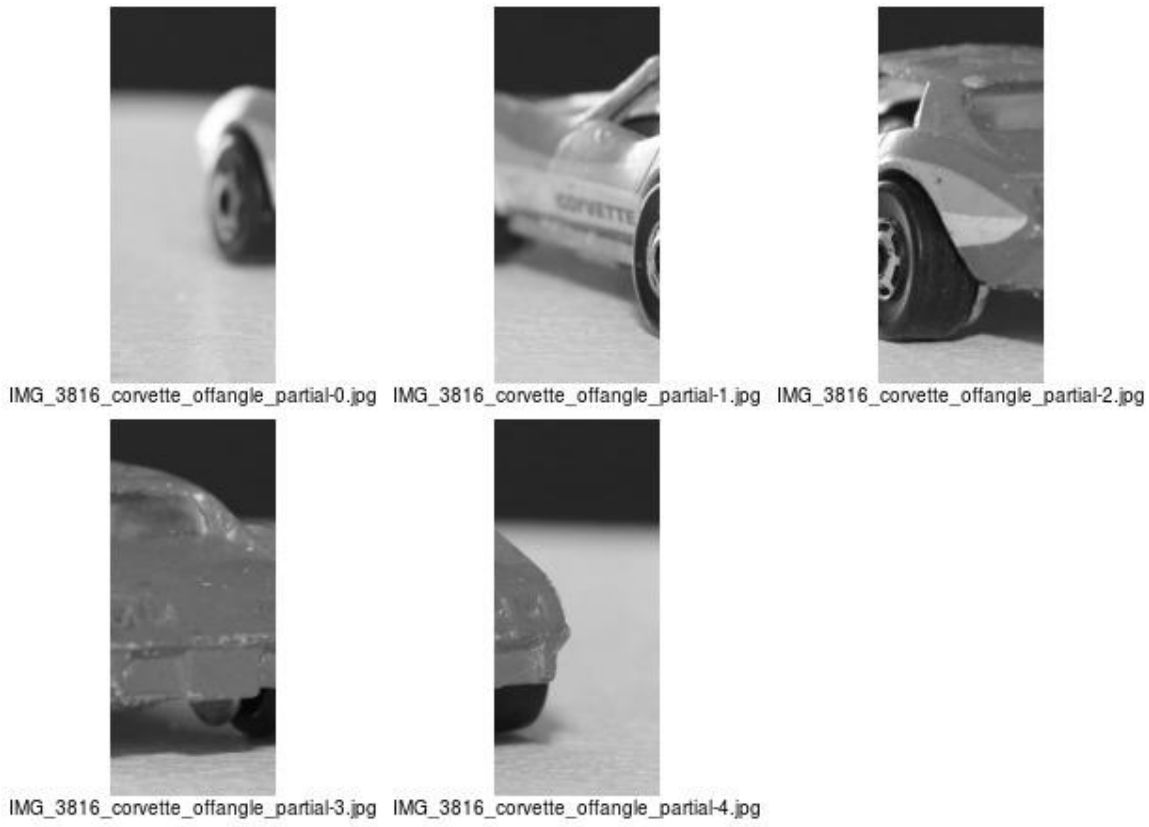


Figure 26: Unknown Incomplete Object Contact Sheet

Once the slices are generated it is necessary to adjust the position and canvas size of the slices to match the original model images. This is acceptable given a conceived system will take both model and incomplete images under similar conditions and we need to have a fixed coordinate system to work from when performing image identification. The moments and Delaunay triangulation for the middle and most interesting slice is shown in the next few figures:

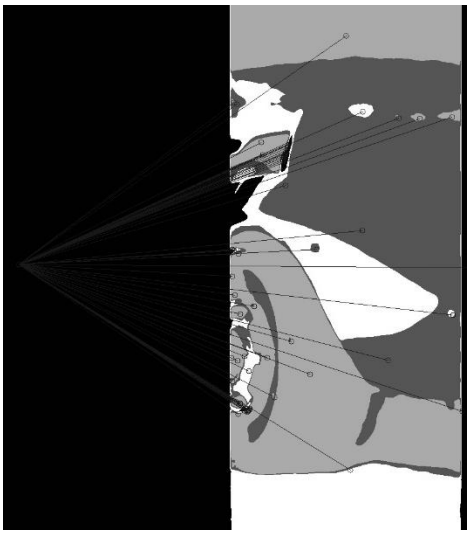


Figure 27: Moments over Middle Slice of Incomplete Off-Angle Corvette

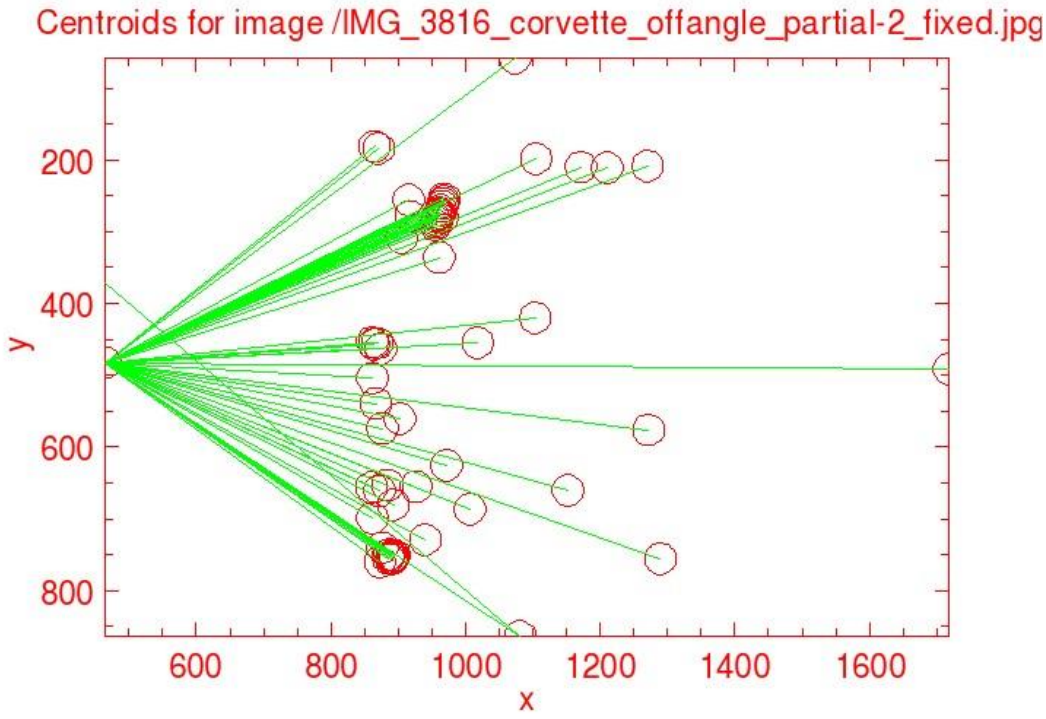


Figure 28: PLplot Rendering of Moments over Middle Slice of Incomplete Off-Angle Corvette

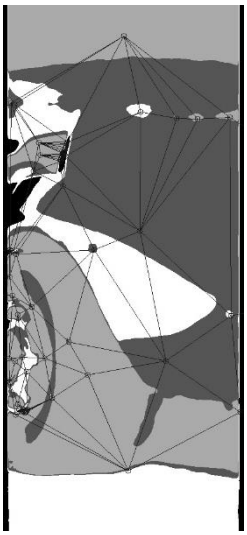


Figure 29: Delaunay Triangulation of Middle Slice of Incomplete Off-Angle Corvette (Cropping may have removed some parts)

In the left most slice, containing the front left wheel and part of the front body piece, the following centroid matches were obtained (+/- error of 10% for all matches using centroids from the global graph):

Table 6: Centroid Matches from Incomplete Offset Corvette Leftmost Slice

Model Image	# Matching Moments	Match Prob.	Match Prob. Per.
data:IMG_3804cbg.jpg	1	0.166667	16.66667
Accumulated Values	**1**	**0.166667**	**16.66667**

Given that this slice has such little data with the front left wheel and hood, it's not unexpected that only the left model view provided any matching segments.

From the Delaunay triangulation for this slice and all other slices, the probability of a match comes not from the matching number of model and sample segments, but from the number of vertices from the total possible vertices with duplicates within a group removed

given that a vertex may form the corner of many different triangles within a Delaunay graph. Note that due to synthesis of regions, the contribution counts and probability matches may be greater than expected. The results of have an error rate of +/- 3%. The results obtained for the first slice can be seen in the following table:

Table 7: Delaunay Matches from Incomplete Offset Corvette Leftmost Slice

Model	Contribution Count	Prob. Match	Prob. Match Percentage
data:IMG_3808cbg.jpg	3	0.008174	0.817439
Accumulated Values	**3**	**0.008174**	**0.817439**

When processing the second from the leftmost slice, the following results were obtained for matching by centroid location:

Table 8: Centroid Matches from Incomplete Offset Corvette Second Leftmost Slice

Model Image	# Matching Moments	Match Prob.	Match Prob. Per.
data:IMG_3804cbg.jpg	2	0.046512	4.651163
data:IMG_3808cbg.jpg	1	0.023256	2.325581
data:IMG_3807cbg.jpg	2	0.046512	4.651163
data:IMG_3809cbg.jpg	4	0.093023	9.302326
data:IMG_3806cbg.jpg	2	0.046512	4.651163
Accumulated Values	**11**	**0.255815**	**25.5815**

In this experimental run, the left, right, and top sides contributed two segments out of 43 each, the top contributed four segments, and the bottom contributed one segment. The remaining 32 were not associated with any of the sides. The Delaunay graph matching generated the following results:

Table 9: Delaunay Matches from Incomplete Offset Corvette Second Leftmost Slice

Model	Contribution Count	Prob. Match	Prob. Match Percentage
data:IMG_3807cbg.jpg	7	0.114754098	11.47540984
data:IMG_3808cbg.jpg	81	0.220708447	22.07084469
data:IMG_3809cbg.jpg	168	0.685714286	68.57142857
data:IMG_3804cbg.jpg	206	0.479069767	47.90697674
data:IMG_3805cbg.jpg	35	0.421686747	42.1686747
data:IMG_3806cbg.jpg	62	0.158567775	15.85677749
Accumulated Totals	**559**	**2.08050112**	**208.050112**

It can be observed that the strongest contributing matches come from the left and top portions of the vehicles with weaker matches from the remaining sides.

The middle slice, which includes the driver rear side, part of the top of the vehicle, and part of the rear of the vehicle, generated the following centroid matches:

Table 10: Centroid Matches from Incomplete Offset Corvette Middle Slice

Model Image	# Matching Moments	Match Prob.	Match Prob. Per.
data:IMG_3804cbg.jpg	2	0.033333	3.333333
data:IMG_3806cbg.jpg	3	0.05	5
data:IMG_3809cbg.jpg	5	0.083333	8.333333
data:IMG_3807cbg.jpg	3	0.05	5
data:IMG_3808cbg.jpg	3	0.05	5
Accumulated Values	**16**	**0.266667**	**26.66667**

The strongest number of matching segments, from the total of 60 possible. came from the top of the vehicle, which is easily seen in the offset image slice – so this is not unexpected. The Delaunay graph generated the following set of matches:

Table 11: Delaunay Matches from Incomplete Offset Corvette Middle Slice

Model	Contribution Count	Prob. Match	Prob. Match Percentage
data:IMG_3807cbg.jpg	6	0.098360656	9.836065574
data:IMG_3808cbg.jpg	74	0.201634877	20.16348774
data:IMG_3809cbg.jpg	26	0.106122449	10.6122449
data:IMG_3804cbg.jpg	44	0.102325581	10.23255814
data:IMG_3805cbg.jpg	75	0.903614458	90.36144578
data:IMG_3806cbg.jpg	172	0.439897698	43.98976982
Accumulated Values	**397**	**0.948341262**	**185.195572**

The second to right most slice, with the middle portion of the left side rear, top, and inside of the right wheel from the perspective of the left side generated the following set of centroid matches:

Table 12: Centroid Matches from Incomplete Offset Corvette Second Right-Most Slice

Model Image	# Matching Moments	Match Prob.	Match Prob. Per.
data:IMG_3804cbg.jpg	8	0.571429	57.14286
data:IMG_3806cbg.jpg	1	0.071429	7.142857
Accumulated Values	**9**	**0.642857**	**64.28571**

Here the strongest number of matches, from fourteen segments, came from the model's left side view, which is not surprising. When considering the Delaunay triangulation for this slice, the following results were obtained:

Table 13: Delaunay Matches from Incomplete Offset Corvette Second Rightmost Slice

Model	Contribution Count	Prob. Match	Prob. Match Percentage
data:IMG_3808cbg.jpg	22	0.059945504	5.994550409
data:IMG_3809cbg.jpg	10	0.040816327	4.081632653
data:IMG_3804cbg.jpg	9	0.020930233	2.093023256
data:IMG_3806cbg.jpg	5	0.012787724	1.278772379
Accumulated Values	**46**	**0.134479787**	**13.4479787**

Interestingly, the strongest contributions here came from the top and bottom portions of the model's multi-view images most likely due to more of the top of the vehicle being in the slice and from the bottom due to its simpler structure and coincidental correspondence. There was also some left and right (likely due to mirror effects) of model image contributions.

When processing the right most segment, the following results were obtained for matching by centroid location:

Table 14: Centroid Matches from Incomplete Offset Corvette Right Most Slice:

Model Image	# Matching Moments	Match Prob.	Match Prob. Per.
data:IMG_3804cbg.jpg	1	0.166666667	16.66666667
data:IMG_3805cbg.jpg	1	0.166666667	16.66666667
data:IMG_3806cbg.jpg	3	0.5	50
Accumulated Values	5	0.833333	83.33333

This was in an unknown image with six segments. The matching found that the left and rear model views contributed 16% of the total match and that the right side contributed 50% of the total match. The remaining views did not contribute to the overall recognition of the object and one segment was not matched to any of the model views. The contributions of the model images from the Delaunay derived global graph is as follows:

Table 15: Delaunay Matches from Incomplete Offset Corvette Rightmost Slice

Model	**Contribution Count**	**Prob. Match**	**Prob. Match Percentage**
data:IMG_3808cbg.jpg	4	0.010899183	1.089918256
data:IMG_3809cbg.jpg	6	0.024489796	2.448979592
data:IMG_3804cbg.jpg	26	0.060465116	6.046511628
Accumulated Values	**36**	**0.095854095**	**9.585409476**

Given that this slice only contains a small portion of the vehicle, the overall matching counts were lower with the left side providing the greatest contribution with the top and bottom sides providing fewer contributing vertices.

Overall, the contributions are summarized in the following table:

Table 16: Summary of Contributions from Model Images Toward Matching Incomplete Off-Angle Corvette

Image Filename	Perspective	C-S0	C-S1	C-S2	C-S3	C-S4	D-S0	D-S1	D-S2	D-S3	D-S4
data:IMG_3804cbg.jpg	Left	16.67%	4.00%	3.33%	57.00%	16.67%	0.00%	47.91%	10.23%	2.09%	6.05%
data:IMG_3805cbg.jpg	Rear	0.00%	0.00%	0.00%	0.00%	16.67%	0.00%	42.17%	90.36%	0.00%	0.00%
data:IMG_3806cbg.jpg	Right	0.00%	4.00%	5.00%	7%	50.00%	0.00%	15.86%	43.99%	1.28%	0.00%
data:IMG_3807cbg.jpg	Front	0.00%	2.00%	5.00%	0.00%	0.00%	0.00%	11.48%	9.84%	0.00%	0.00%
data:IMG_3808cbg.jpg	Bottom	0.00%	2.00%	5.00%	0.00%	0.00%	3.00%	22.07%	20.16%	5.99%	1.09%
data:IMG_3809cbg.jpg	Top	0.00%	6.00%	8.33%	0.00%	0.00%	0.00%	68.57%	10.61%	4.08%	2.45%
Totals		**16.67%**	**18.00%**	**26.67%**	**64.14%**	**83.33%**	**3.00%**	**208.05%**	**185.20%**	**13.45%**	**9.59%**

	Error Rate
Centroid (C)	10%
Delaunay* (D)	3%

*Contributions > 100% due to Synthesis of Regions and Correspondence of Coordinates across Multi-views (duplicates within model view removed)

The contribution counts of nodes and vertices from the global graph and Delaunay derived graph are summarized in the following table:

Table 17: Summarized Contribution Counts of Centroid and Delaunay Matching of Incomplete Off-Angle Corvette

Image Filename	Perspective	C-S0	C-S1	C-S2	C-S3	C-S4	D-S0	D-S1	D-S2	D-S3	D-S4
data:IMG_3804cbg.jpg	Left	1	2	2	8	1	3	206	44	9	26
data:IMG_3805cbg.jpg	Rear	0	0	0	0	1	0	35	75	0	0
data:IMG_3806cbg.jpg	Right	0	2	3	1	3	0	62	172	5	0
data:IMG_3807cbg.jpg	Front	0	1	3	0	0	0	7	6	0	0
data:IMG_3808cbg.jpg	Bottom	0	1	3	0	0	0	81	74	22	4
data:IMG_3809cbg.jpg	Top	0	1	5	0	0	0	168	26	10	6
Totals		**1**	**7**	**16**	**9**	**5**	**3**	**559**	**397**	**46**	**36**
							Duplicates are removed on a per view basis, but there may coincidental correspodance from Synthesis or reflective views				

	Error Rate
Centroid (C)	10%
Delaunay* (D)	3%

THE DETAILED DESCRIPTION OF THE STEPS IMPLEMENTATION

In this work, a command-line driven research code, named Obstruction, was developed using the Oracle Java programming language and used a set of MATLAB derived codes developed by Dr. Bourbakis and previous research students as an initial basis for segmentation and processing of the image. The syntax of the code is compatible with Java 8.0 or later. The design documents used as a basis for the design of the system was developed using ArgoUML 0.34 [14], PlantUML version 8050 or later [54], and Draw.io version 10.6.3 or later [20]. The active development on ArgoUML ended a few years ago and PlantUML and Draw.io proved to be a more robust environment for developing the necessary analysis and design documents for this work. However, ArgoUML was utilized during some of the earlier phases of this work and some documentation from that portion of the effort will be found throughout this document. The Obstruction code has dependencies on a number of third party Java API libraries that include: OpenCV for use of common Computer Vision operations, the Apache Poor Obfuscation Implementation (POI) framework for generating Office compatible spreadsheets in Object Linking and Embedded (OLE) Compound Document Format [47], PLplot for visualizing the intermediate products and final results, Apache PDFBox for generating Portable Document Formatted (PDF) versions of some of the intermediate and final results [40], and HyperSQL for storing the database of model images (e.g., the images' constituent parts) [24]. There is an extended discussion on HyperSQL in chapter three. The implementation was done inside the Eclipse Integrated Development Environment (IDE) Oxygen 3a (4.7.3) or later [19]. The code has

been and will continued to be maintained in a public source code repository on GitHub under the mrobbeloth-wright-state account at https://github.com/mrobbeloth-wright-state/obstruction . The repository is also linked to another GitHub user account: mrobbeloth. The source code base has been open sourced for use by the larger research community to advance the state of the art in incomplete image detection. The source code has been released under an Apache License 2.0, which allows for private or commercial use, modifications, general distributions, and does not prevent some from using it to assist in generating patents (not trademarks though). The only real restrictions include not providing any warranty or liability in its use and requiring proper attribution.

OPENCV API NOTES

OpenCV is a computer vision library that is open source with bindings for several programming languages, including Java, and works across a variety of platforms from desktop operating systems like Microsoft Windows, Linux, and Apple OS X to mobile systems like Apple iOS and Android [48]. It can take advantage of multiple processors and hardware accelerators in certain situations. It was chosen for use to remove the need for using a closed-source and proprietary solution like MATLAB while achieving a higher level of performance than possible with a MATLAB clone like Octave. Although there are several packages available for use within Java, the core (e.g., Core), image procedure (e.g., *imgproc*), and image codec (e.g., *imgcodec*) packages are probably used the most. The Core package holds the clear majority of the OpenCV model classes along with many common MATLAB equivalent supported algorithms. For example, the Core package has calls to perform common math operations on matrices, to

generate random data, to shuffle data, etc. The *imgcodec* package is primarily used for writing intermediate generated and final graphical deliverables to secondary storage.

PLPLOT API NOTES

PLplot is an API plotting package commonly found in the scientific fields for use in generating visualizations of data [27, 36]. The other major package that most researchers would be familiar with is GNUPlot. It has a set of library routines that be used with appropriate bindings on a variety of different programming languages, including Java, in a batch oriented or interactive manner. Although PLplot is capable of generating a variety of plots, this work only uses the PLplot API to generate x-y plots. The code works by creating a `PLStream` object, calling a variety of methods to set parsing options, general operation and output options, background and foreground plotting colors, etc. The stream is initialized with the `init` method, the plot extents are set with the `env` method, the labels are added with the `lab` method and then various drawing operations are then called, which include: `line`, `poin` (e.g., point), etc. In some cases, special Unicode characters are specified as a parameter to the overloaded drawing operation methods. The stream is then closed to write the file to disk. The “Sample Run – Model Image” section contains examples of PLplot generated files. In particular, the reconstruction of the segmented regions is produced by conversion of the chain codes into plots.

APACHE PDFBOX API NOTES

Although not utilized at the time of the writing, previous versions of the obstruction code made use of Apache PDFBox API for helping to convert together various output images

into Adobe PDF format with the goal of generating documentation. There were several methods added to the `ProjectUtilties` class: `writePDFtoDisk`, `writePDFtoDISKDblSrc`, and `mergePDFs`. The mergePDFs method takes advantage of the `PDFMergerUtility` class to merge PDF documents together and has straightforward methods that include `setDestinationFileName`, `addSource` and `mergeDocuments` to complete the merging process. The writePDFtoDISK and writePDFtoDISKDblSrc methods include the `PDFDocument` and `PDDocumentInformation` class to generate the PDF document and embedded metadata along with calls to `setAuthor`, `setProducer`, `setKeywords`, `addPage`, and various drawing methods. The API requires the use of an `PDPageContentStream` to hold and format the content for an individual page.

APACHE POI API NOTES

The Apache POI API has been an incredibly useful API for use in this work. Although it can be used to generate any type of Microsoft productivity document, it is primarily used in this work to generate Excel spreadsheet to capture various sets of data that are not necessarily meant for long-term storage into the database. Specifically, the data on finding a most probable, final match is stored into a classic Excel spreadsheet file. To generate a spreadsheet, one begins by creating an workbook using the `XSSFWorkbook` object. Next, the user adds worksheets using various `XSSFSheet` objects. Then with each worksheet object, the user adds rows and cells using `XSSFRow`, `XSSFCell`, `BorderSyle`, and `CellType` objects along with the `createRow`, `createCell`, `setCellStyle`, and `setCellValue` methods among others. The spreadsheet data is written to disk using the workbook's `write` method and finally the file

is closed using the workbook's `close` method in accordance with standard usage expectations. If various worksheets have to be referenced at different points in the code, the `getSheet` method of the `XSSFSheet` class can be used. There is a legacy method, `generatePresentation` in the `ProjectUtilites` class of the obstruction codebase, that was used for documentation purposes, in previous versions of the code. It started by creating a master slide using the `XSLFSLideMaster` class to which the user would add a layout object using the `XSLFSlideLayout` class and then individual slides using the `XSLFSlide` class. On each `XSLFSlide` class object, `XSLFTextShape` and `XSLFPictureShap` objects would be added and various methods called to add text and pictures (the images generated during a particular study).

ANALYSIS & DESIGN

In this work, a researcher looking to identify an obstructed image would likely need to perform a set of operations with respect to building a reference database of model images and then processing individual images or a suite of images against the database of whole images. To capture these requirements, the Unified Modeling Language (UML) use case diagram is employed. After consideration, the use cases have been identified for this work in the following figure:

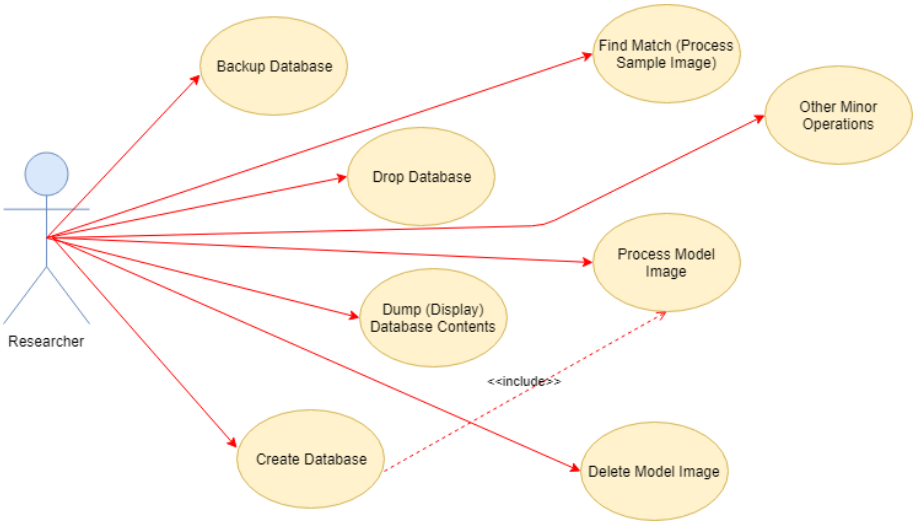


Figure 30: Use Case Code Requirements

The researcher should spend most of his time finding a match once a set of model images have been processed a priori by the system. There are some ancillary database functions that the researcher may carry out. The most important one of these is to back up the database of model images to an archive file for safe keeping or to study the use of a different set of model images on the same code to test the efficiency of the research code in different contexts, to test the code with different types of obstructions, some combination of the previous two, etc. The researcher may also wish to display an abbreviated and human readable form of the database contents to ensure that the model images are being processed as expected. Finally, to a lesser extent, the researcher may wish to start the modeling process over by clearing out (e.g., dumping) the database, verifying the setup of the research code by performing a version check to display system, image, and visualization properties, or verifying operation of the code by running a test mode. The test code, though, is only present for legacy reasons, and was developed prior to the implementation of the database capabilities in the current work. Some

actions, such as *process model image*, may implicitly trigger the activation of other use cases by the research code. In the case just mentioned, the code will trigger the *create model* database use case.

When having the software system process an image, the following process illustrated in figure 31 is used. This is a top-level basic flow of the software system. The details of many of the components in this diagram will be discussed in greater detail in their respective section. The preprocessing steps, highlighted in figure 32, are now discussed. The researcher supplies a list of images for the system to process. The researcher also has to indicate if model or sample images are being processed. There is no ability to mix operational modes in this implementation. It should be noted that modifying the set of processed model images may invalidate earlier results with images that provide a stronger probabilistic match or a weaker match for images that are a close, but poorer match due to distracting features in those new model images not previously considered by any of the techniques in this work. In either operating mode, the system will start with a segmentation process, followed by a binary conversion process. The images are loaded in gray-scale mode and preprocessed prior to being sent to the segmentation routine. The preprocessing begins with a custom sharpening (e.g., focusing) of the foreground area with a Gaussian Blur. The extra noise present in the capture of the image along with the rough textures that might be present in many model images aren't needed for the matching process. This should aid any concavities or convexities generated by shadow. In particular, though, we are interested in the most significant components of the model images and a Gaussian Blur, serving

as a low-pass filter, allows those high-frequency (e.g., component edges at points of significant grayscale shifts) to come through the filter.

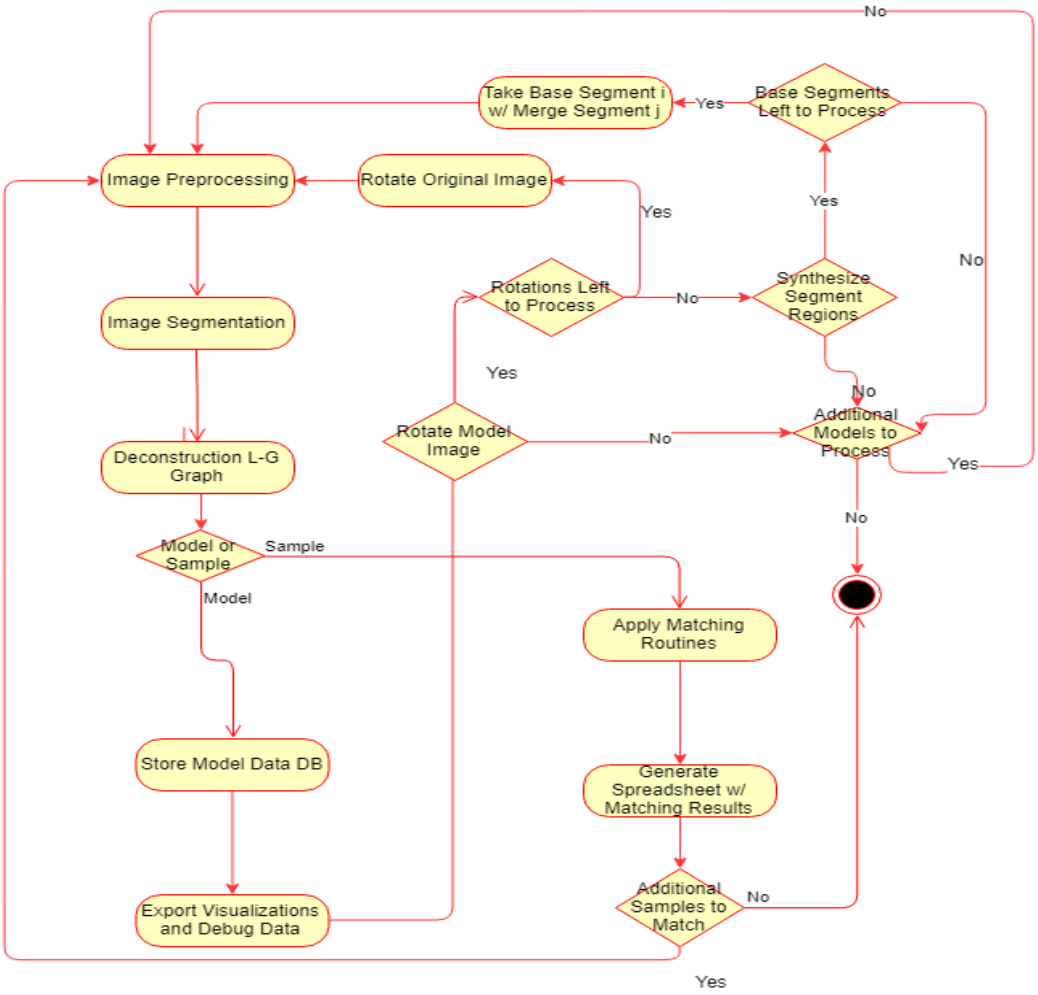


Figure 31: Basic Obstruction Program Flow Activity Diagram

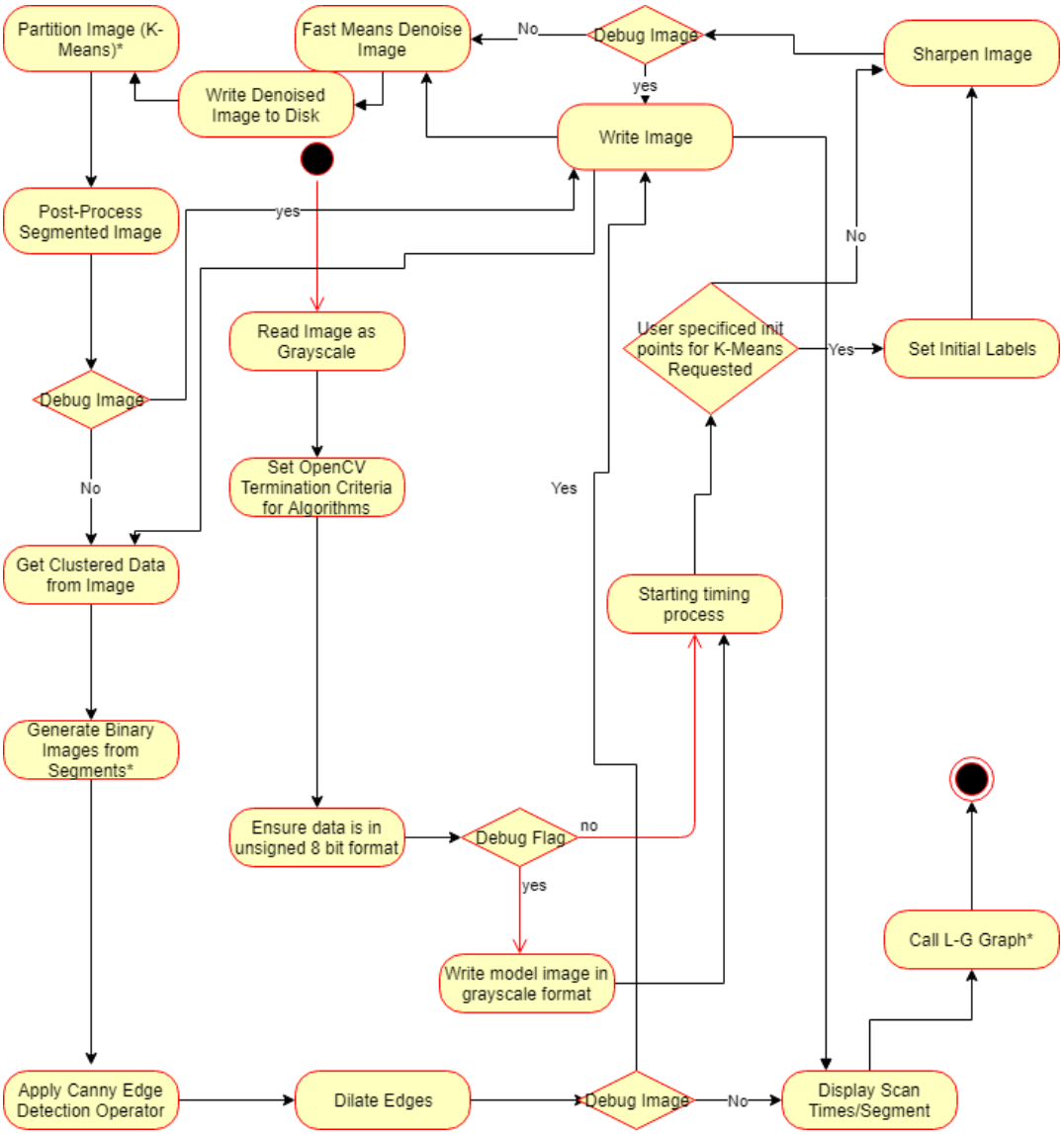


Figure 32: Preprocessing Operations in Obstruction Code

After this, a de-noising operator is applied. Specifically, the OpenCV *fastNlMeansDenoising* operator is applied which attempts to remove noise by taking the average pixel values of each pixel and is based on the algorithm by [72]. The parameters used when calling this method request a seven-pixel template window and a twenty-one-pixel search window along with an aggressive specifier for the amount of noise to remove (e.g., h=85). [21]

noted that window sizes should be odd and that modest removal of artifacts can be achieved with h=3. Once pre-processing is complete, segmentation begins. The segmentation routine is a K-Means algorithm supplied by the OpenCV API or a custom implementation by Bourbakis translated from the original MATLAB codes used in previous research efforts not related to incomplete object recognition. For the OpenCV or NGB version, it requires the number of clusters to be present at the end and the termination criteria. The OpenCV version also requires how the centers are to be initialized (NGB version uses random centers).

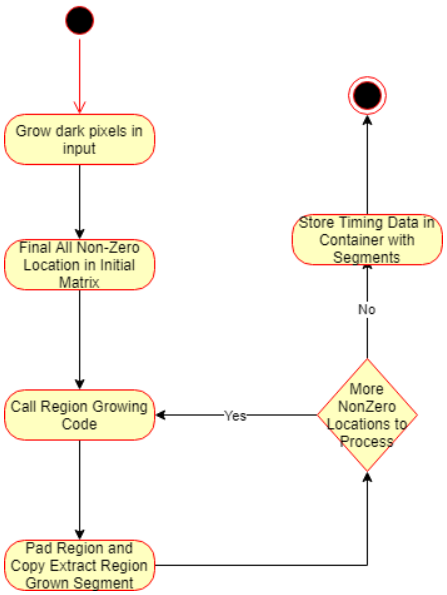


Figure 33: Generate Binary Segments from Clusters (Partitions) using ScanSegments()

Following the segmentation process, which requires the post-processing of clusters into partitions and the extraction of binary image segments using the region growing algorithm from within `ScanSegments()`, the resulting model or sample images have a Canny operator and morphological dilate operator applied on a per segment basis to isolate and enhance the edges of the object being processed. The Canny uses a multi-part algorithm that begins by reducing

noise using a 5x5 Gaussian filter followed by application of a Sobel kernel horizontally and vertically to find an intensity gradient in the image [48,50]. This allows the algorithm to find the edge gradient direction of each pixel in each segment image as follows:

$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2}$$

$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

Figure 34: Edge Gradient and Angle Formulas Used by OpenCV Canny Operator [48,50]

After this, the Canny algorithm attempts to determine which pixels are part of the edge of each segment by checking to see if the pixel is part of a local maximum in the direction of the gradient and if so passes it on for hysteresis threshold as shown in the following figure:

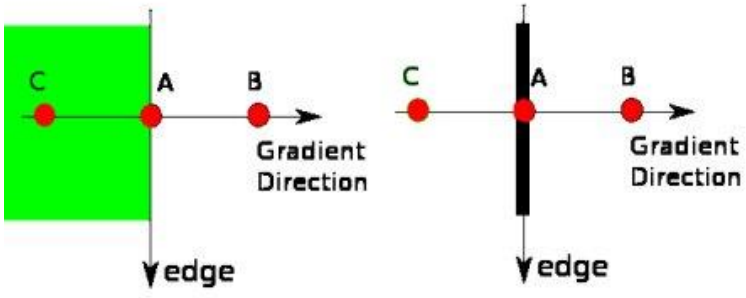


Figure 35: Edge Direction in OpenCV Canny Operator [48,50]

Finally, the minimum threshold and maximum threshold values, which were passed as parameters to the operator method are applied. Normally, those pixels whose intensity are above the maximum are part of the edge, those between the minimal and maximum might be

part of the edge if they are connected to other possible edge pixels, and those below the minimum are discarded [48]. In this work, zero is passed for both the minimum and maximum to include all possible pixels retrieved from the edge detection portion given that each segment is now binary in form. The binary conversion process then takes the gray scale segments of the image and converts them into black and white segments based on application of a threshold operator and morphological operator. After this, the scan segment processing times are displayed and optionally saved to the debug log file and each binary segment is processed by the L-G graph code whereby a complete geometric description of the segment is generated. Referring back to the basic obstruction activity flow diagram, after processing by the L-G graph, if the segment belongs to a model image, the pertinent information from the L-G graph algorithm processing needed for future comparison is stored in the model database. Then final visualizations are exported and debug data is exported to spreadsheet files and text logs as appropriate. If the user requests 45 degree rotations to be performed on the model image they are done at this time on the original image source. After all rotations are processed, synthesis of segments on a per-view basis is performed if requested by the user. The reader is referred to the section, Synthesis of Views, in the Incomplete Recognition chapter for additional details. If the segment belongs to a sample image, a set of matching routines is applied on a per-segment basis using the multi-view model image database. Once all the segments have been processed, a spreadsheet with matching results is generated along with timing information on the overall

process being stored in the debug log.

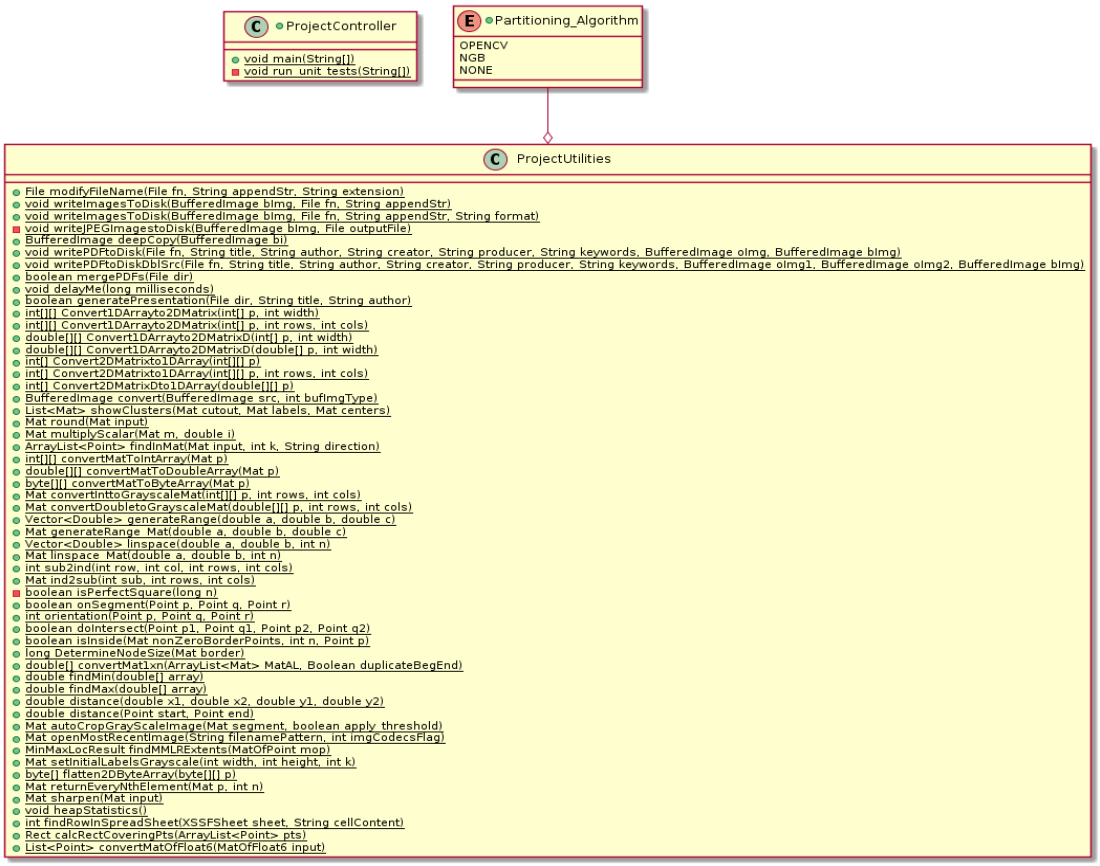


Figure 36: Project Controller and Associated Classes

The basic building block classes of the obstruction code are simple in their overall organization even if the operational details of how individual methods are implemented are more complicated. The top-level class diagram, a sort of pseudo-UML class diagram, is shown in the next figure:

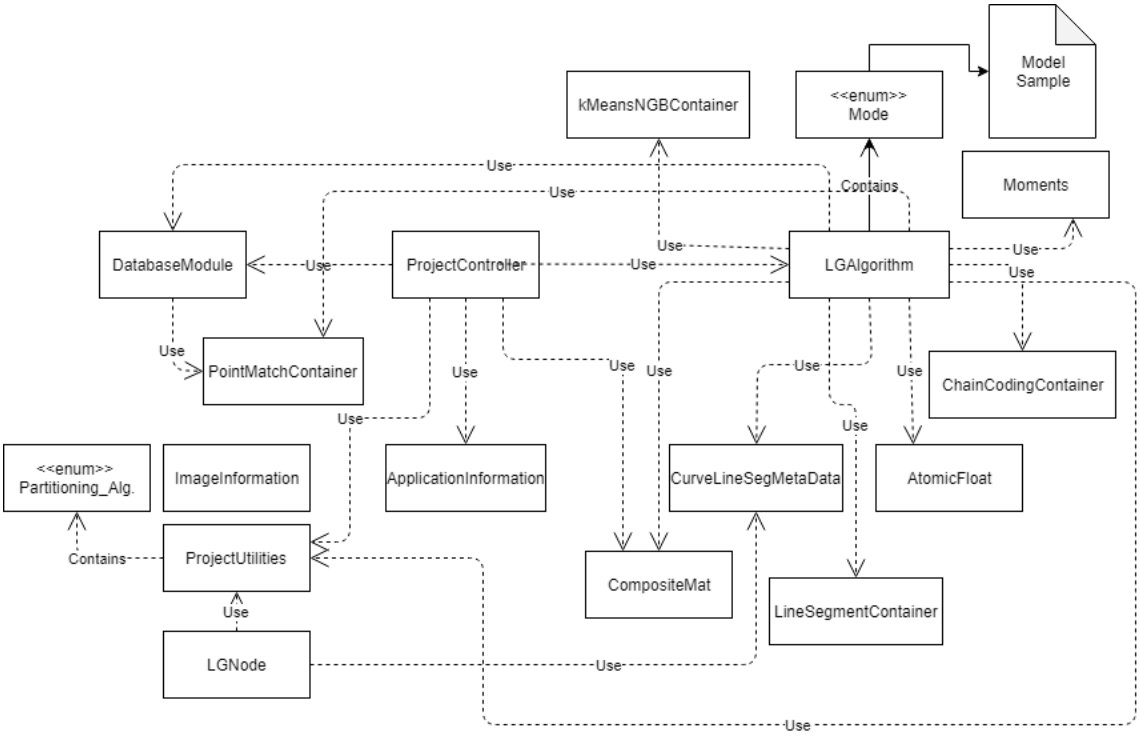


Figure 37: Top-Level Obstruction Class Diagram

This diagram helps to show the single main controller class, `ProjectController`, works with the various in-memory model classes, the model database (e.g., `DatabaseModule`), the primary model classes (e.g., `LGNode` and `LGAlgorithm`), the secondary model classes (for example, `ChainCodingContainer`), the major utility support class (e.g., `ProjectUtilities`), and several minor support classes (for example, `ApplicationInformation` and `ImageInformation`).

`ProjectController` is responsible for starting the session. It is called from both the main method and contains a legacy testing method. The main method simply hands off to static

methods in other classes depending on the mode of operation selected by the researcher specified via command line parameter. The usage statement is of the form:

```
java –jar obstruction.jar {debugFile=/path/to/debug/file}
plplot.libdir=/path/to/plplot/native/library/code --command
{paramas}
```

However, this work does not normally use the jar generated form of the code. Instead, the code is directly called from the class files that are generated using [19]'s Debug or Run Configuration feature. This is a handy method for running Java code by using a GUI to specify various configurations, which include the specification of the main class, program arguments, virtual machine arguments, the working directory, preferred Java Runtime Edition (if multiple versions are installed), class path modifications, etc. It is easier to create these configurations verses using a scripting build file, but these configurations are not easy to port from one research station to another. Therefore, notes are usually kept on the key parts of the different configurations and the configurations are recreated on different systems. A sample configuration is shown in the next several diagrams:

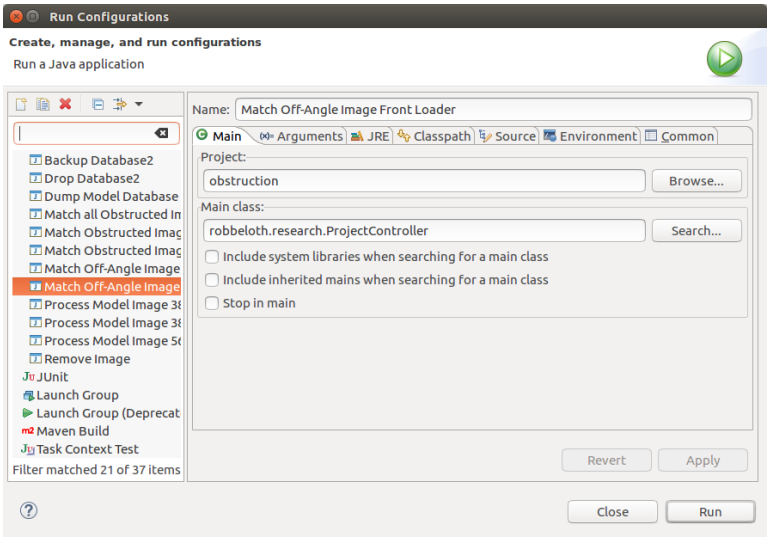


Figure 38: Sample Run Configuration, Eclipse, Main Tab

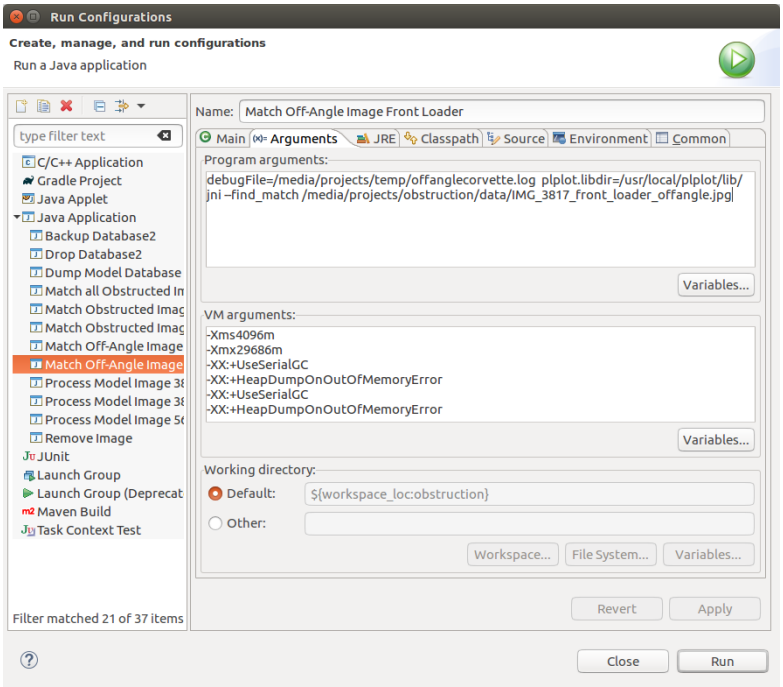


Figure 39: Sample Run Configuration Eclipse, Arguments Tab

The *ProjectUtilities* class is a utility classes consisting of a hodgepodge of primarily publicly accessible static methods for processing images, converting objects from one form to another, writing output data to disk in a variety of formats, performing operations of matrices, checking data structures for certain characteristics, performing input/output operations, etc. This utility class also houses an enumeration for selecting the type of partitioning algorithm.

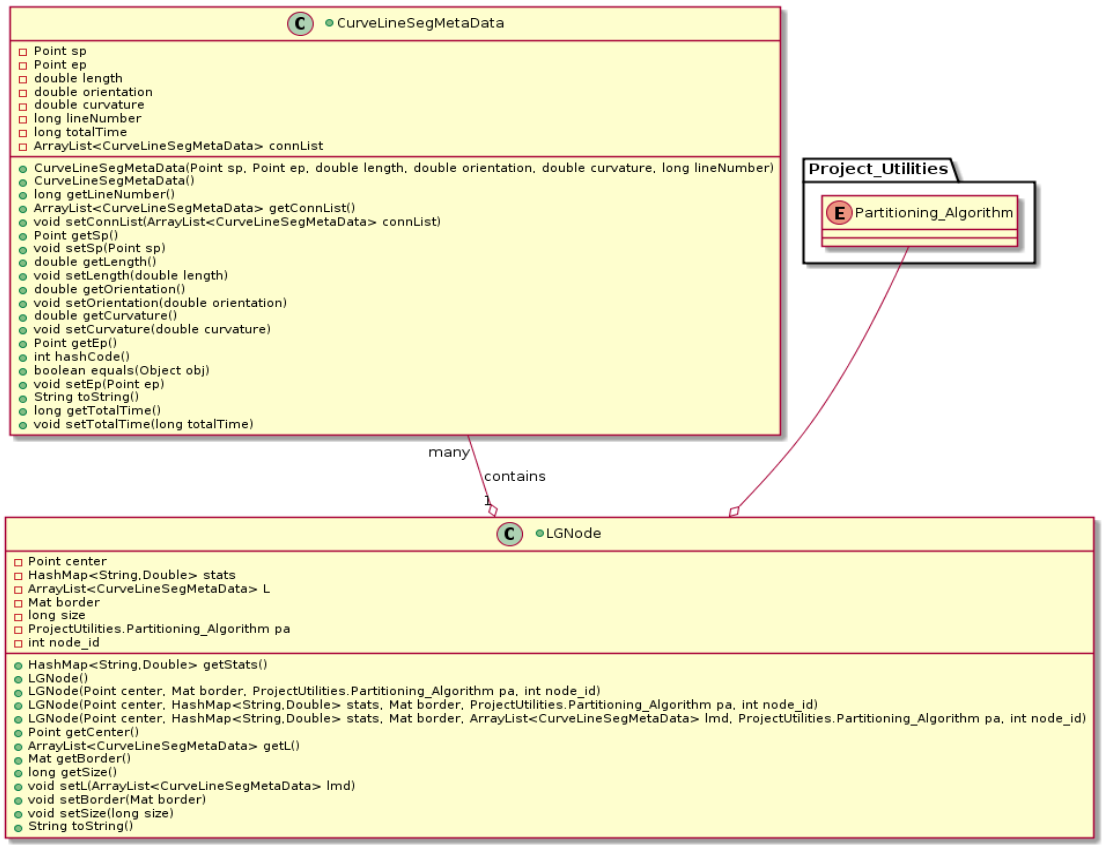


Figure 40: LGNode and Supporting Classes

The core model class is the *LGNode* class. The objects of this class are the nodes of images processed by the LG Algorithm. A node in this algorithm consists of a center point, a set of border points, the number of pixels belonging to the region, a set of curved line segment

metadata objects, a node identifier, the partitioning algorithm used, and the partitioned and scanned region grown data. There a number of publicly accessible getter and setter methods along with several constructors and an overridden `toString`() method to provide a human-readable state descriptor of an instance of the LGNode class for debugging or record keeping purposes. In general, any class described here with an overridden `toString`() implementation is fulfilling a similar purpose. The set of curved line segment metadata objects, come from the `CurveLineSegMetaData` class. An object from this class has a starting point (sp), an ending point (ep), length in pixels, orientation in degrees, a curvature value, its sequence number (e.g., line number), and the time to calculate it took the LG Algorithm to calculate it in nanoseconds (ns). There are several constructors, getter, and setter convenience methods to work with the members' instances of this class.

Figure 41: LGAlgorithm Control Class (rotated to landscape orientation to improve visibility)

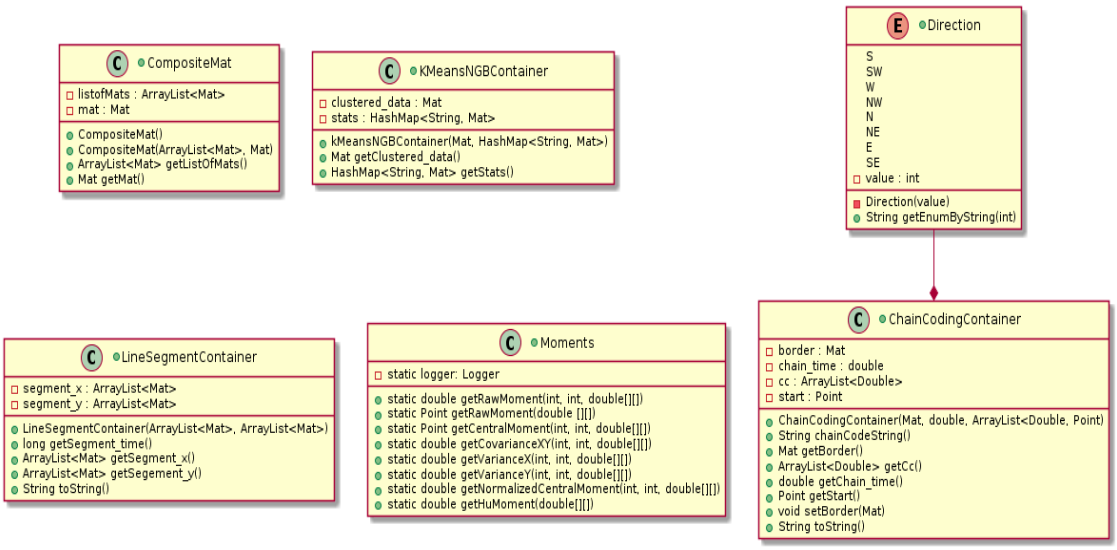


Figure 42: Secondary Model Classes Class Diagram

The LG Algorithm class is a control class where the processing and matching of images are carried out. There are a few string descriptor objects to help record some statistical data about the entire process. Most the methods in this class are private static methods. The details for these processes will be discussed in other areas of this work. This class does hold an enumeration for use in conditional statements where the next step to take is based on whether the image being processed is a model image or a sample image. There are a number of ancillary classes for transporting data structures around the research code in a more organized manner. If the Bourbakis implemented K-Means algorithm is used for partitioning, the clustered data and some statistics generated during that process are stored in an object of the `KMeansNGBContainer` class. The `CompositeMat` objects are used to hold the individual binary converted segments for later processing and analysis. The `ChainCodingContainer` class objects holds the chain code representation of the border of each binary segment in the

`cc` field. These objects also hold a record of the amount of time it took to generate its chain code and the object contour pixel set in the border `Matrix` object. In addition, there is a `Point` object holding the starting x and y coordinates of the segment (e.g., the chain code started at coordinates indicated in the start object). Finally, there is an enumeration that provides a human readable description of the chain code in terms of Cardinal directions. The `LineSegmentContainer` objects are a collection of all the (x,y) coordinate lists that make up a region's border from the image. They are generated from the chain code itself and not by means of reprocessing the pixels that make up a border region. That said, it should approximate on a 1-to-1 basis the coordinates of the border pixels. For all the secondary model classes up to this point, the methods classes are getter and setter in nature and augmented by an overridden *toString* method to provide a more human readable description of an instance of the object. The Moments' class objects are a little different, though, and play an important role in the matching process. Recall, moments provide a mathematical or physical description of the shape or distribution of a set of data points. In a 2D continuous function, this is defined by the order (p+q) using (Hu, 1962):

Or given the discrete case of working with a grayscale image with pixel intensities I(x,y):

$$M_{Pq} = \sum_{x} . \sum_{y} (x - \bar{x})^P (y - \bar{y})^q f(x, y)$$

$$\bar{x} = \frac{M_{10}}{M_{00}}, \bar{y} = \frac{M_{01}}{M_{00}}$$

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (x - \bar{x})^p (y - \bar{y})^q f \cdot (x, y) \, \mathrm{d}x \, \mathrm{d}y$$

In this work, given the statistical nature of the image processing being done, we are interested in the moments providing the centers of mass for each region. Going back to our earlier activity diagram on building the local node objects, we can obtain the central moments of the regions in the image by calling the *getRawMoment()* method and supplying the appropriate parameter to get the central moment or the convenience method *getCentralMoment()* can be accessed to bypass some post-call steps. Although not used at this time, there are additional convenience methods for obtaining higher-order raw moments (e.g., variance, skewness, etc.).

Although a full discussion of the database used in this work is deferred until chapter three, the Java code for working with this database is presented here. It is a critical class in this work for the long-term storage of model data. The methods in the `DatabaseModule` class are designed to create, modify, scrub, and retrieve data. It initially looks like a utility class, like `ProjectUtilities`, but actually employs a `Singleton` pattern with the connection to the database made in the `main` method of the `ProjectController` class. The class diagram is shown in the following figure:

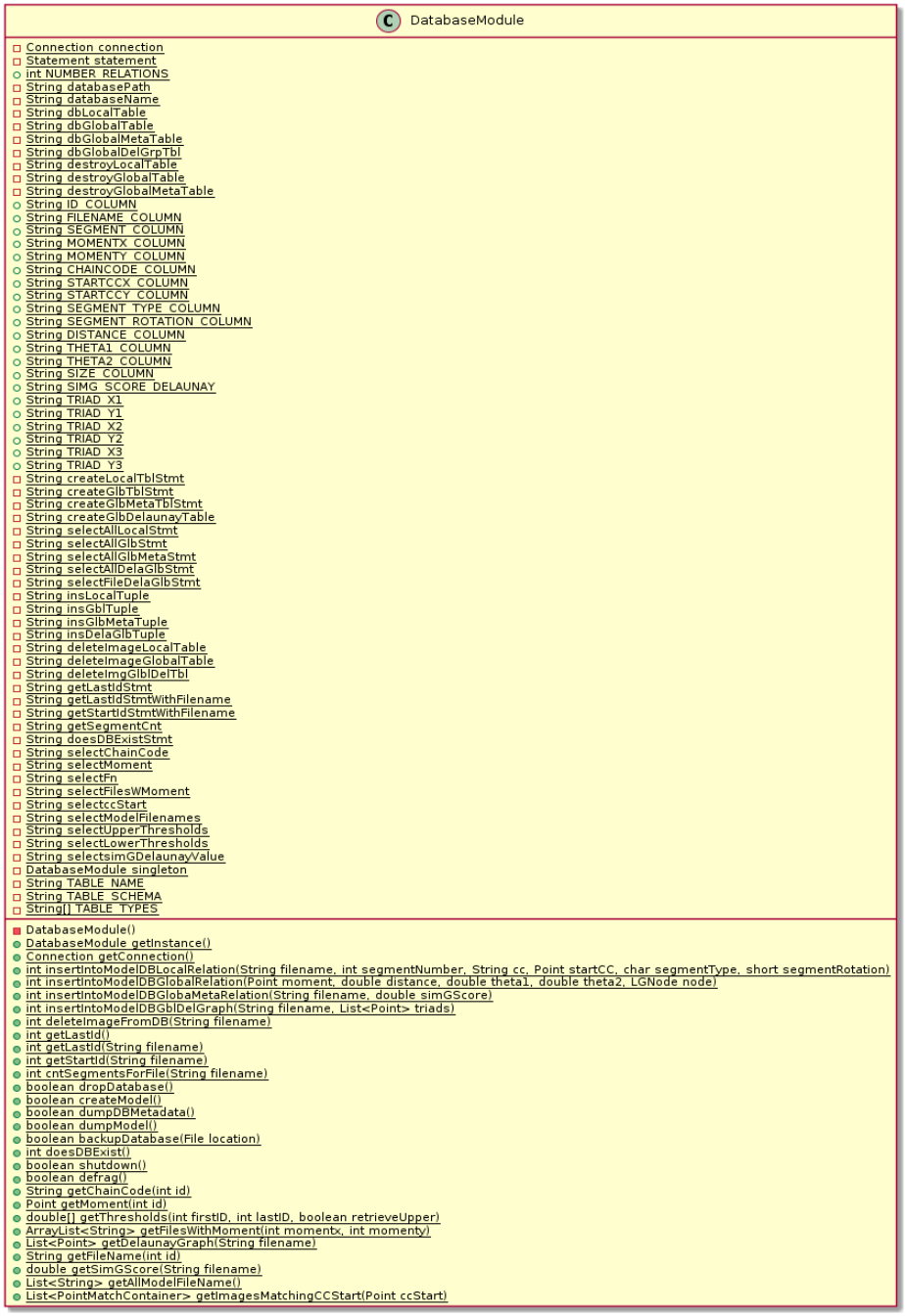


Figure 43: DatabaseModule Class Diagram

Note the use of the `getInstance` method to ensure that the `Singleton` pattern is enforced by returning only the same DatabaseModule object each time. A Java `Connection` class object aids in establishing this connection. The first time the object will be instantiated, but the same object reference will be returned on any subsequent calls to the method. Many of the methods in this class trace back to the use case diagram discussed earlier. The ability of the researcher to back-up the database is performed by the `backupDatabase` method, to display the contents of the database with `dumpModel`, to drop (e.g., remove or delete) the database with `dropDatabase`, and to create the database with `createModel`. The addition of model data on a per segment basis is accomplished with a set of `insertIntoModelDBx` methods. The sample image matching routines in the `LGAlgorithm` Class make use of a variety of select (e.g., query) methods from the DatabaseModule such as `getChainCode`, `getMoment`, etc., to get the data in the form of Java `ResultSet` and `ResultSetMetaData` objects that serve as an abstraction to working with a Sequel (SQL) cursor. Most of the SQL code is embedded in `String` based private instance variables that also relay on other String instance variables storing the name of the database, the names of the tables, and various column names in those tables. Given the need to fill in SQL parameters using the data passed in the various method calls, Java `PreparedStatement` objects are used to prepare the SQL statements for execution with the HyperSQL engine. The few SQL statements that do not involve the use of parameterized data use the Java SQL `Statement` object and its `executeQuery` method. The reader is referred to chapter three for a more in-depth treatment of the database processing.

Finally, there are a handful of minor support classes in the application. The first one is meant to provide diagnostic information—`ApplicationInformation`. This purpose of this class is to describe the target system's ability to read and write various image formats and supplements the description of system properties, the system environment settings, and visualization properties. The methods of this class and other supplemental API method calls are called with a runtime configuration using the command flag `–version`. The contents of the file are too long to include in this work, but the reader is redirected to the `docs` subdirectory of the research code on the GitHub obstruction software repository where the file `ver_ex_RobbelothResearch_Workstation.log` contains an example. Next, to support atomic operations on floating point values the `Number` derived `AtomicFloat` [45] class is provided and is used in the matching routines utilizing a `ForkJoinPool` set of worker thread objects. Last, there is the `ImageInformation` class to report on basic information about an image such as `BufferedImage` type, pixel size, the number of color components, transfer type, transparency type, color model, and characteristics of grayscale images were applicable. The class diagram for the minor support classes is shown in the following diagram:

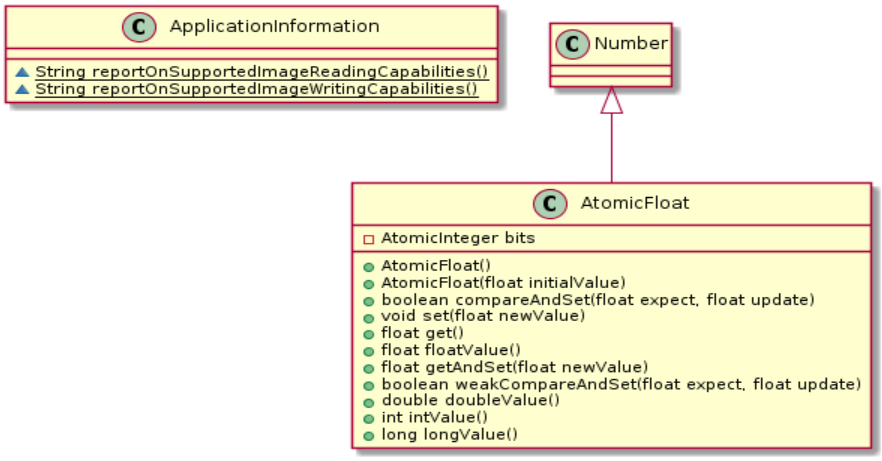


Figure 44: Minor Support Classes

From an implementation standpoint, the codebase currently sits with the following approximately code sizes at the time of publication:

Table 18: Approximate Codebase Size by Class

Class	Lines of Code (LOC)
ApplicationInformation	87
AtomicFloat	51
ChainCodingContainer	150
CompositeMat	102
CurveLineSegMetaData	192
DatabaseModule	1755
ImageInformation	162
kMeansNGBContainer	31
LGAlgorithm (with extensions and matching methods for processing models and samples)	5683
LGNode	158
LineSegmentContainer	92
Moments	178
PointMatchContainer	54
ProjectController	656
ProjectUtilities	2263
*** OVERALL ***	**11614**

From the time that the codebase was added to a code repository on GitHub starting in the July 2015 following a presentation on [53] at NAECON 2015, over 400 commits have been made with several test branches and addition commits made.

SEGMENTATION

Segmentation is a well-known process in computer vision whereby a system attempts to group an image composed of a set of pixels into larger units known as super-pixels or partitions. The goal with segmentation is to simplify the structure of the image for later analysis. The grouping of the image via segmentation into larger units permits the system to identify boundaries or objects present within the image [59]. The boundaries of objects constituting an image are at its core straight lines, due to nature of a raster image, but with tight enough directional changes and resolution may appear as a curved line. An image containing true curves is only possible with vector graphics as the curves are encoded as mathematical descriptions to allow the reconstruction of an image at any given resolution on an appropriate display. Of course, the display itself is subject to aliasing effects of the construction process even if the representation beneath is a nice mathematically formulated curve.

In any event, segmentation serves to group pixels together by means of a metric so that each one is labeled with an identifier that denotes the nature of its grouping. In this work, the distance of a pixel from some set of anchor center points is used as the metric to organize and regroup the pixels as the centers themselves are updated. The center points are used later in the construction of the L-G Graph.

In this work, the K-means algorithm is used to segment the image after some initial preprocessing to turn the image into a grayscale formatted image using OpenCV Matrix object based 8-bit unsigned floating-point encoded values (e.g., `CvType.CV_8U`) and application of a blurring operator from the OpenCV API to remove obvious image artifacts. If you recall, grayscale (e.g., gray level) images have pixels encoded between 0 and 255 (e.g., unsigned eight bit entries). Later, when the algorithm retrieves the edges in the segment, we only need to store binary image (or bitmap) values where each pixel holds a zero or one. For now, the data structures choices might not be the most optimal OpenCV Matrix format to use, particularly if characteristics like texture, intensity, brightness/contrast shifts, etc., are desired characteristics to be track of on a per segment basis, but it appears to achieve good enough results for those characteristics that this work focuses on to achieve results. In fact, given the use of resource constrained hardware, such concessions are necessary. Therefore, it appears that down sampling to eight bit integers or bit level entries is fine since we are most concerned with the edges of the segments.

Aside from the data structure used to hold the image data or one of its derivatives, an interesting question that hasn't been quiet resolved to this researcher's satisfaction is what are the optimal set of preprocessing operators (e.g., algorithms) to apply to the image to achieve the best accuracy in matching a candidate image to a model image? Earlier, a set of operations were described that seem to provide acceptable results, but it seems reasonable to conclude that better results could be achieved with additional applied experimentation. The preprocessing application is quite critical to overall success and it should be applied consistency

to all model and sample images to maximize reliability. As for the K-Means algorithm itself, it is classified an iterative partitioning algorithm that groups related pixels into *K* clusters [5]. There are a host of alternative approaches from simple threshold application to edge detection to partial differential methods and more. This work makes use of threshold operators after the initial segmentation to simplify the L-G algorithm processing of regions. The generation of chain codes and the reverse edge construction in the modified L-G algorithm is an edge *detection* or reconstruction algorithm. Edge detection helps the research code in capturing important geometric details of an image for use in advanced analysis by looking for intensity differences. In particular, given a grayscale Image of the form I(x,y), we can define the intensity changing function using the squared gradient magnitude, defined as follows [26]:

$$\|\nabla_I\|^2 = \left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2$$

When this function generates a significantly large enough value to meet some threshold, an edge has been detected. This work does not focus on capturing changes due to differences in depth or surface colors and textures. Also, according to [26], the edge detection methods aren't always reliable in that it will find edges where none exist and miss other important edges. However, the application of some operators like the Canny edge operator can provide better results than simply using a standard local gradient magnitude edge operator. As such. there are two segmentation-based implementations available for use in this work that will provide a foundation for application of the edge detection operators. The first implementation is the one provided in the OpenCV API. The second implementation comes from a converted

MATLAB code used in earlier works on L-G graph construction by Bourbakis. Both algorithms operate in a similar manner:

1. Choose K cluster centers by fixed assignment or random placement
2. Process the pixels in each row and column. Calculate the distance from that pixel to each center and assign it to the cluster with the smallest distance from it to that cluster's center
3. Recalculate the centers of each cluster
4. Repeat steps two and three until an iteration limit has been exceeded, no change is occurring in any center, or the change in the centers is on average less than some ε

The OpenCV algorithm provides a greater variety of initial placement parameters. This work has used the PP_CENTERS based on the work of [3]. The authors noted that the placement of cluster centers is done in such a way to, "… minimize the average squared distance between points in the same cluster." [3] went on to note that their placement strategy with a traditional K-Means algorithm yields an algorithm with θ(log k) run-time performance and acceptable levels of accuracy.

The alternative approach was to use the Bourbakis K-Means algorithm translated from the original MATLAB source code. Verification of identical performance of the two code bases was performed by fixed placement of initial centers instead of allowing the algorithm to randomly place those centers by the OpenCV static uniform random call (*Core.randu*) in the OpenCV API Core library. OpenCV matrix objects perform the bookkeeping as necessary.

Statistical data on average intensity, average row value, and average column is maintained and returned to the caller. It is not utilized in other parts of the research code, now, though (e.g., It was meant to be as faithful a translation from the original source as possible and is maintained for legacy purposes). The algorithm provides the same color (or grayscale tone in this case) for pixels belonging to the same area as is consistent with the operation of the K-Means algorithm. The standard distance formula utilizing common static method calls from Java's Math API is used for calculating the distance of each pixel to each center.

Overall, the OpenCV API is heavily utilized in much of the converted code as it is part of a mature image processing and computer vision library that continues to receive support from open-source community contributors and it provided a faithful set of approximate substitutes to Matlab operators and algorithms for use in implementing and extending the L-G algorithm.

LOCAL-GLOBAL (L-G) GRAPH ALGORITHM

As mentioned earlier in this work during the survey, the L-G algorithm creates geometric and characteristic descriptions for each region in an image. The regions' descriptions when pulled together provide a global description of the image. Each region can be considered to be a segmented object of the image. The algorithm in this work requires the list of image segments generated during the segmentation process, the filename of the image being worked on, the partitioning algorithm used, whether a model or sample image is being processed, whether or not to generate the Delaunay triangulation graph from the global set of nodes, whether or not to generate rotations and synthesized regions (model processing only), and whether to generate debug data among other inputs. Once connected to an initialized database (model images only with a calculation of the next identifier number to use), the global portion of the graph is initialized along with several ancillary data structures to track chain codes, graph/segment timing information, centroids, etc. Also, the output directory is created if not present from previous executions.

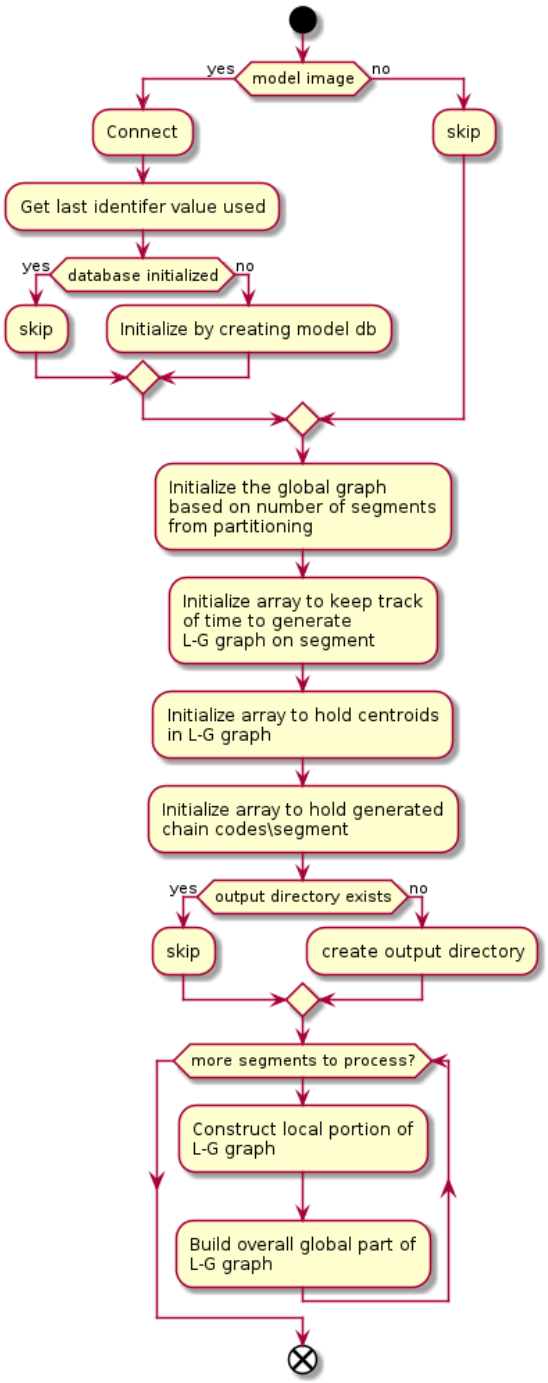


Figure 45: L-G Activity Diagram Initialization and High-Level Processing

Next, the core of the algorithm is executed. There are two main parts that form the core of the L-G algorithm: first, the local portion of the L-G graph containing the geometric description of the line segments for each image segment is formed; second, the global portion of the L-G graph is constructed which focuses on the establishment of the centroid regions and the connection of the starting segment centroid to the other centroids in the remaining segments of the image. Once completed, the algorithm will have generated an overall geometric description of the image under consideration. Figure 45 shows us the initialization that take place during the execution of the L-G algorithm and the high-level activities for each segment.

In figure 46, a more detailed view is shown of the location construction

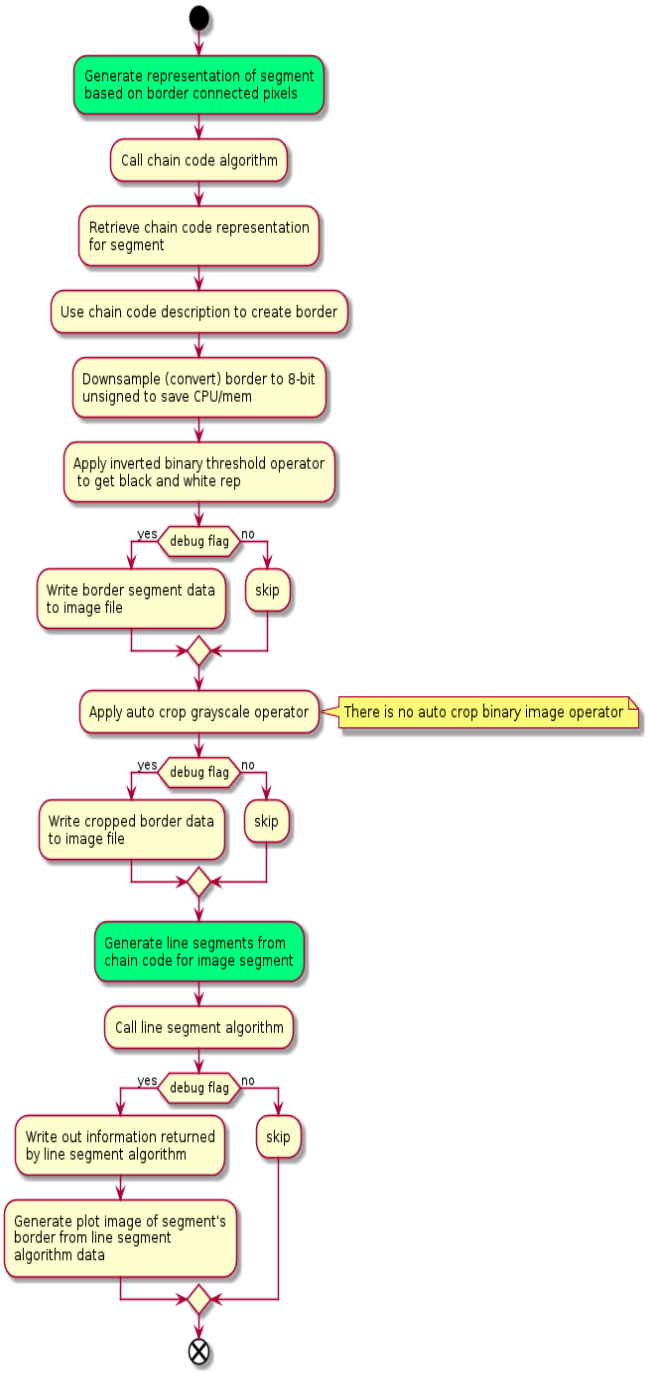


Figure 46: (1st part) Local Graph Generation L-G Algorithm (occurs for each segment of picture)

phase of the L-G graph. The process starts by generating an internal representation of the image segment. The approach chosen in this work is to use chain codes. Thus, the chain code algorithm is called (method *chaincoding1*), whose behavior will be discussed shortly. After the algorithm returns, the chain code representation for the segment is retrieved. A chain code representation uses a compass (e.g., cardinal) metaphor. Given a starting point, all points relative to that one is represented by use of a system of cardinal directions, namely: N, NE, E, SE, S, SW, W, and NW. The cardinal directions can be replaced with simple integer values to be more compact and an enumeration used to covert between the two systems for human readability if so desired. This is demonstrated in

figures 47 and 48. A small bit field array could make future operations even more efficient, but this optimization has not been carried out yet. In any case, the chain code algorithm generates a string description of the border region of the segment, which can be used to create or recreate part of the image. The border image Is then down sampled from a floating-point representation to a simple 8-bit integer representation to save CPU cycles and memory storage in future operations. Following this conversion, an inverted binary threshold operator is applied to generate a simple black and white outline of the segment's border that is easy to process by analysis and visualization algorithms. If the debug flag has been set, a visual representation of the inverted border region segment data will be written to disk. It is important to recognize that is a representation based on the chain code and not the original image. The next major step is to auto-crop the image to make sure the segment size has been minimized. After this, the L-G algorithm can begin to generate line segments (method

NW (7) or{-1, 1}	N (0) or {0,1}	NE (1) or {1, 1}
W (6) or {-1 ,0}	X	E (2) or {1, 0}
SW (5) or {-1, -1}	S (4) or {0, -1}	SE (3) or {1, -1}

Figure 47: Human Readable Cardinal Directions of a Pixel's Possible Neighbors with Computer Enumerable Translations

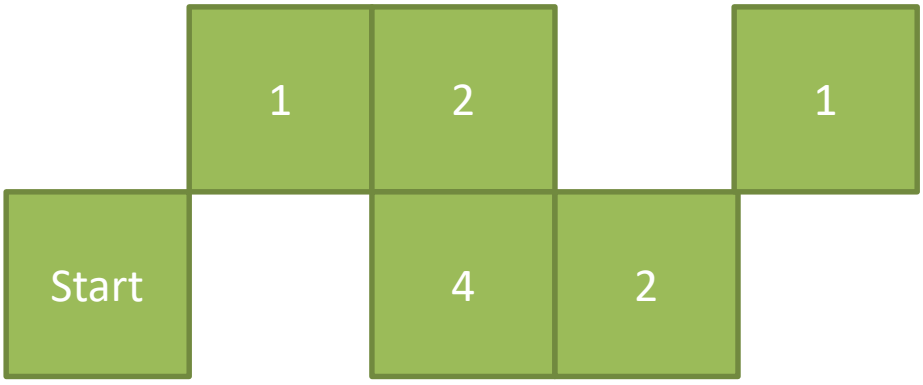


Figure 48: Generating the Chain code by Walking the Line Border Pixels

line_segment) from the chain code that can be labeled and analyzed to determine various properties applicable to each line component that will aid future detection tasks. In this method, black pixels are generated from the starting point location by using each chain code entry and the neighbor array of x and y axis single position shifts to *walk* the border of the segment. There is a sensitivity parameter that can be supplied to create coarser or finer lines depending on the researcher's needed accuracy. The greater the accuracy, the greater the CPU/memory demands on the system. For now, the highest level of sensitivity with single pixel resolution is requested. Once the set of line components for the image segment under consideration have been created, information about those components and a visualization of those components will be written out to disk if the debug flag is set. The visualization is

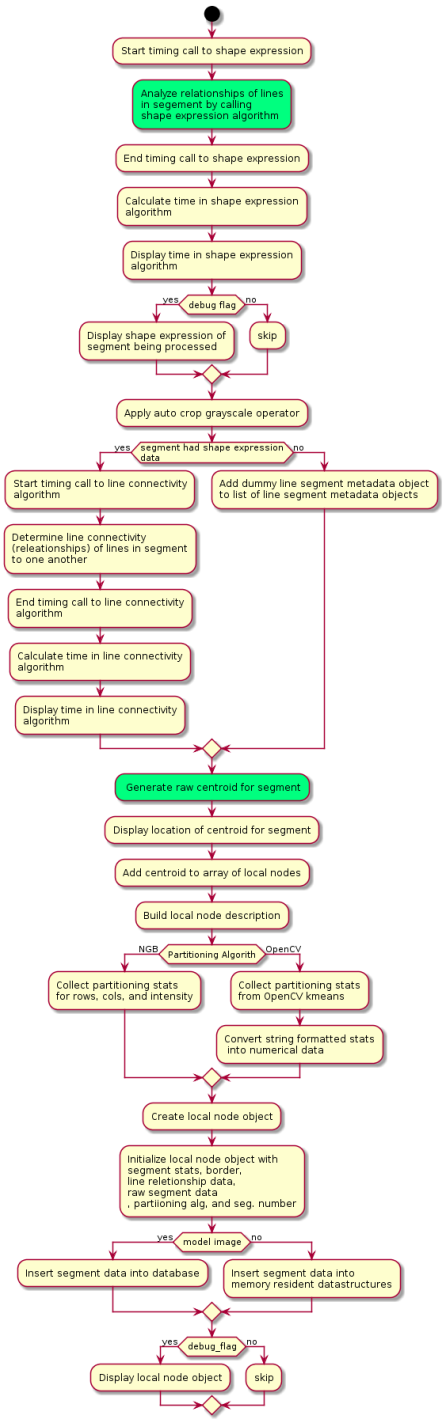


Figure 49: Final steps in creating local node in Local-Global graph algorithm

performed by the PLplot library and requires some data conversions from formats that OpenCV understands (e.g., the MATLAB compatible formats) to standard Java array lists data structures and finally double arrays by means of a self-coded, linear iterative static conversion method. The visualization is written to disk in the jpeg format with a ten-pixel border and text title, “Segment plot for segment i,” where the parameter ‘i’ is the number of the segment being processed. The entire visualization is rendered using the PLplot line command.

Once this is done, some time is spent understanding the characteristics of the segment by processing it with a shape expression algorithm. This algorithm will form the basis of a future object extraction and representation process used in a more comprehensive matching algorithm over earlier attempts used by this work in initial research. The algorithm generates the length, orientation, and curvature of each line in the segment. The results from this algorithm are largely dependent on the

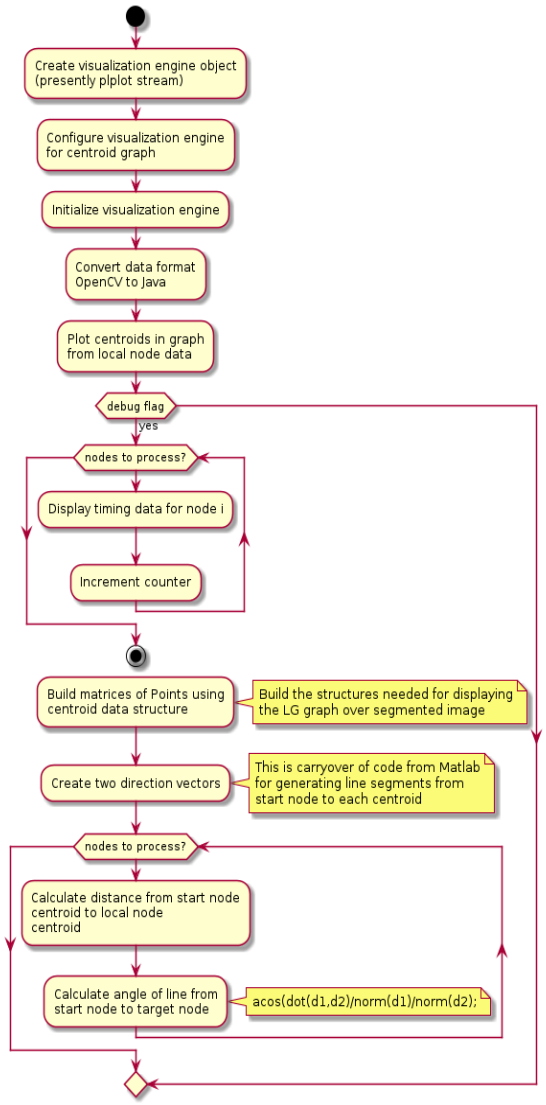


Figure 50: Start generating global graphs

sensitivity used by the line segmentation algorithm. After this, a line connectivity algorithm is executed to determine any properties that the lines in a segment may have in relation to one another. Next, the centroid for the segment is calculated. If the segment being calculated is part of a model image, it is added into the obstruction database; otherwise, it is stored into an array of centroids for the image being processed. Finally, if the debug mode flag is set, a human readable display of the local node's objects will be made.

At this point the local node of the graph for the given segment has been generated. Now, the focus shifts to showing the final construction of the global graph. This is seen in figure 51. This starts with recording in a graph the construction of the centroids. After initializing and setting up the visualization engine, the data that is in OpenCV format is converted into standard Java data structures for use in graphing. After this, some extra diagnostic data for code optimization purposes is made

available if the debug flag is set, and then the process of generating the next visualization commences. So, the first graph that was generated was showing just where the centroids are located in each node of the graph. However, it is also useful to see where those nodes are in relation to the segmented image. Therefore, a series of calculations is carried to generate line segments from each local node's center to the start node's center. To draw these line segments property, a distance vector is generated to know how long each line should be in the graph. Next, the angle at which to draw that line is calculated by use of the arccosine function with the norm of the two distance vectors.

Once the calculations are done, all the points making up those lines must be built as shown in earlier figures. So, for each node, the coordinates and clustered data is passed to a construct lines routine. Next, the moments that were calculated earlier for each local node are written out to disk. The final critical calculation to be used in addition to the location of the moments is the angle of differences between the start node and each local node. A routine is called to calculate that data and then an iterative loop is used to record the data on a per node basis in the master spreadsheet. Finally, the spreadsheet resources are cleaned before the core of the algorithm completes and control is passed to the matching phase for those images being processed that are sample images.

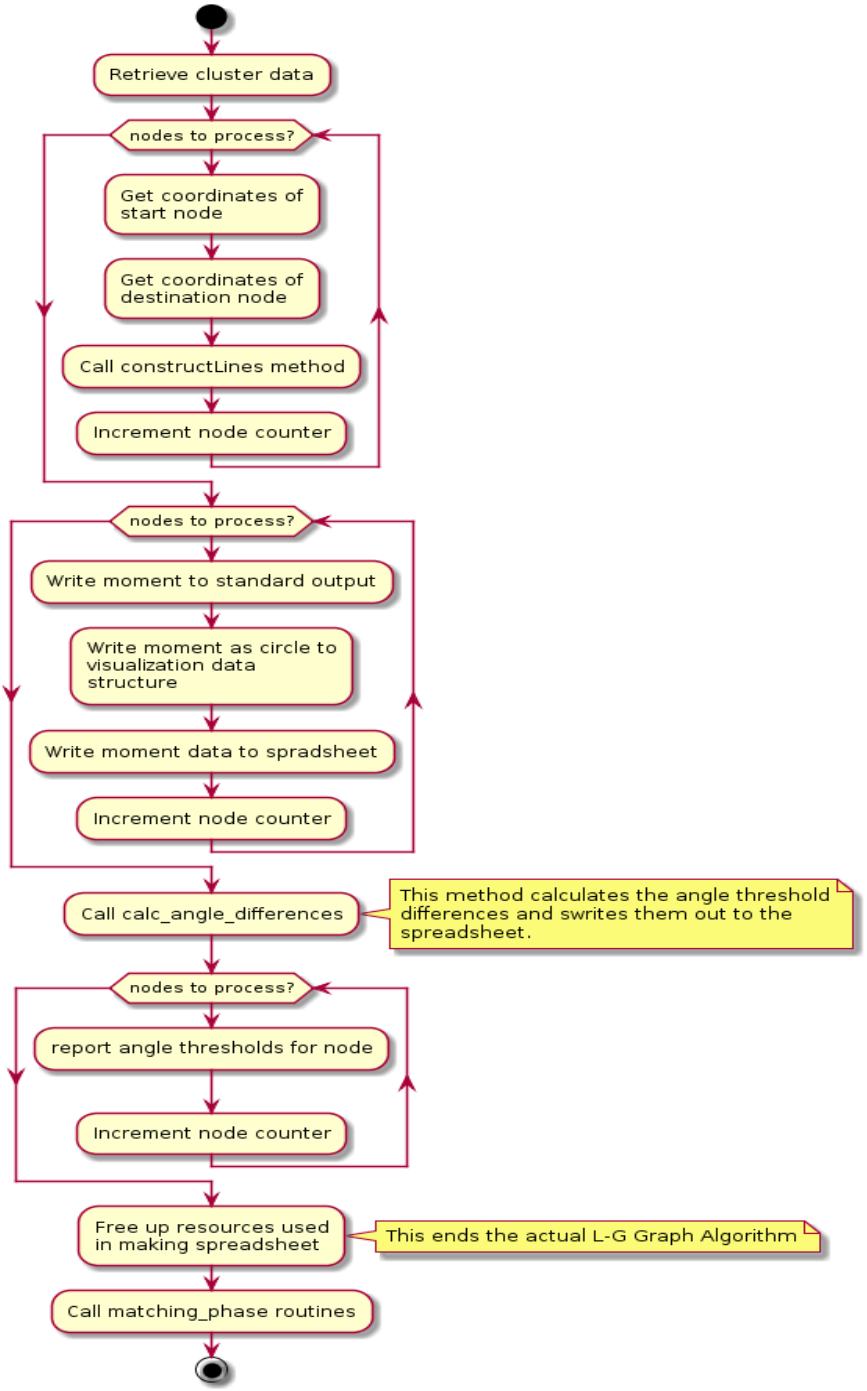


Figure 51: Generate final data from L-G Algorithm

SAMPLE RUN – MODEL IMAGE

At this point, it is helpful to show a sample run of what occurs when the software processes a sample image for inclusion into the model database. The process for any sample images starts the same, but does not have its data stored into the database and has its in-memory data used in the matching routines to find a probable match, where the model image does not. If a model image is processed without a debug flag set, most of the image files are not written to disk and only a small subset of the diagnostic data is written to standard output. However, a variety of intermediate images is written to disk and additional debug data is logged to disk if the debug flag is set. In any event, after reading the sample image into memory, it is written back out to disk to verify it was read with the proper parameters (not shown here). For example, in the current implementation, an image is read as a grayscale image and the data down sampled to 8-bit unsigned integer values. A sample run for a model image starts with an image being cropped, reduced, grayscale converted (if not already done), sharpened, and smoothed. The sharpening operator is a combination of a Gaussian blur and weighted addition operator from OpenCV's image processing and core libraries. The de-noising operator is OpenCV's non-local means de-noising operator *fastNIMeansDenoising*. This is different from the typical Gaussian operator that

Figure 53: Model image with preprocessing operations applied

Figure 53: OpenCV processed image of dump truck passenger side

tallies a pixel's adjoining pixels' values to which it applies an averaging or median value. The *fastNIMeansDenoising* operator finds a series of similar windows similar to the pixel's neighborhood and then applies an averaging operator. The goal with the latter two steps is to remove as many artifacts as possible while strengthening the most prominent borders that might interfere with pattern matching. Next, the partitioning algorithm is applied. Depending on the configuration of the parameters and the algorithm chosen, highly variables results will occur. Therefore, it is important when building the database of model images (including distractors) that all images be processed with the same set of parameters and algorithm. In the future, the database should be expanded to capture this metadata so that any candidate images are processed under similar conditions to minimize the risk of false positives or missed matches. The parameters used include the number of clusters, the number of iterations that the partitioning algorithm runs to refine the results, and the number of initial configurations to try for placements of the cluster centers. There is also a speed/accuracy tradeoff in the configuration of these parameters with higher values requiring greater processor and memory resources on the target machine. In figure 54, we observe the output of the OpenCV algorithm for the entire image specified with only two clusters,

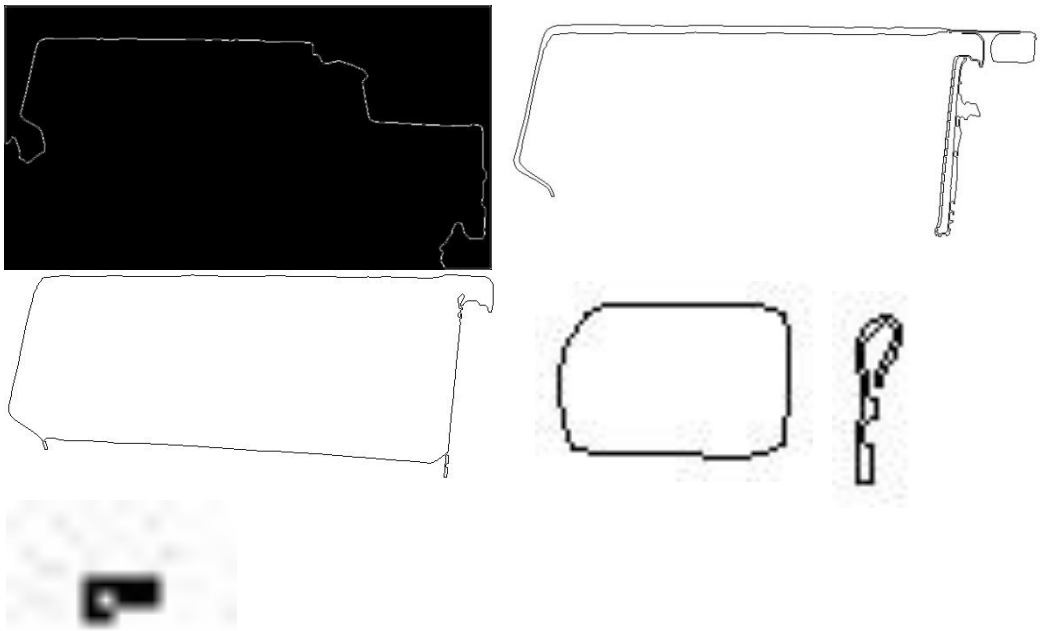


Figure 54: Selected uncropped and cropped segments from partitioning (low parameter values)

terminating criteria of either sixteen iterations or an epsilon change in centers of less one pixel; [3] center initialization, which is the current best practice to use in a partitioning algorithm, and use of the OpenCV partitioning algorithm. There are two sets of the images produced with figure 54 showing some of the uncropped images with the first two segments and cropped segments in the last two since it makes it easier to see the contents of the segment. Indeed, for some of the sparse segments that get grouped into clusters, the application of the cropping operator makes it far easier to study data poor segments. In either set of images, the first segment gives you the border around the truck, the second the interior of the truck's continuous portion, the third gap between the truck bed and the cabin, and the fourth the undercarriage portion between the rear and front tires. Later, when the L-G algorithm is building the chain code representation of the segments, the chain code

representation can be used to rebuild the segments to verify the accuracy of the algorithm as seen in the next set of pictures:

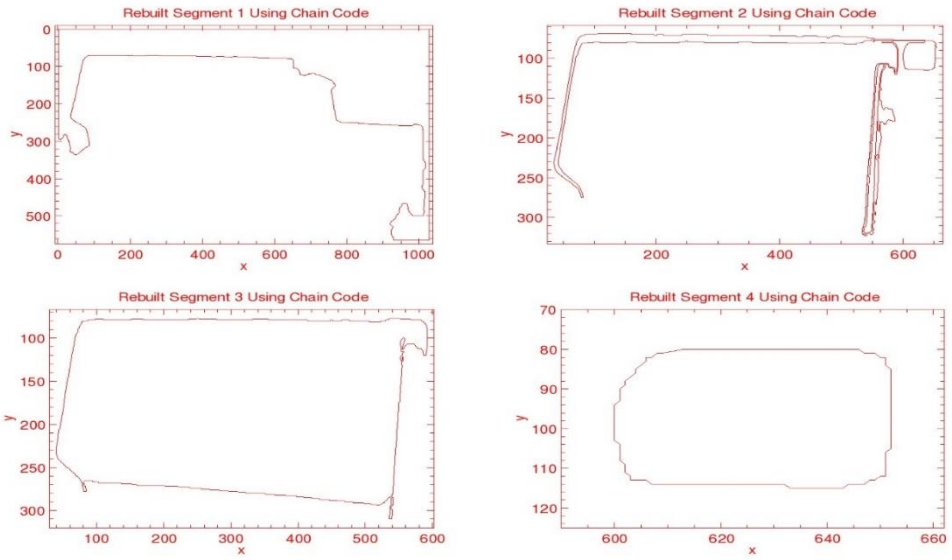


Figure 55: Reconstructed borders of select regions

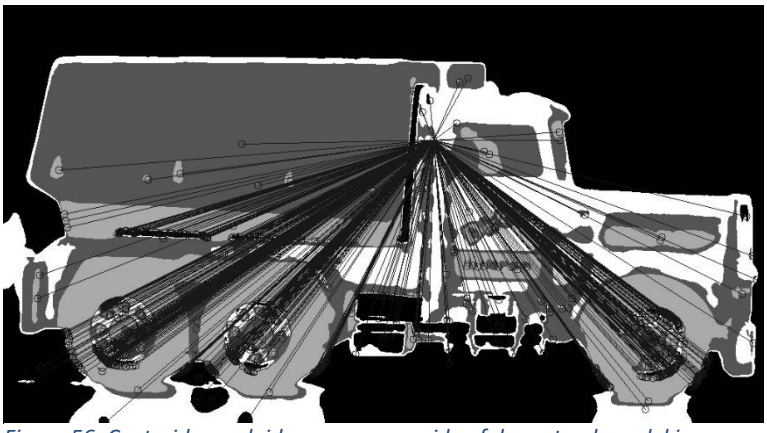


Figure 56: Centroids overlaid on passenger side of dump truck model image

Keep in mind that the reconstructed border segments in figure 54 are inverted and reflected from the partitioned images due to a difference in coordinate systems (a future improvement could be to normalize these images to reflect the base image coordinate system). There is an earlier process that also reconstructs the borders using OpenCV, though it seems to have a bug when handling

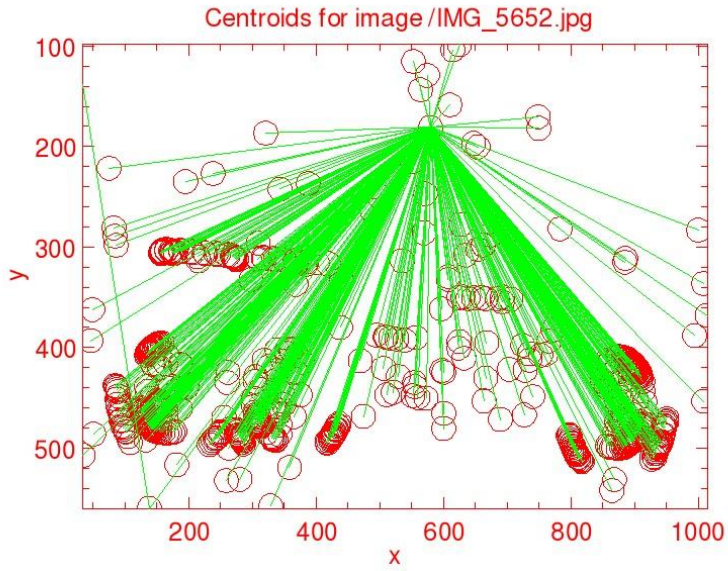


Figure 57: Centroids for passenger side of dump truck model image

large segments whose borders reach the extent of the image (not shown here). Eventually all the local nodes are processed and the following visualization is generated during the global construction showing the connecting of the starting node to every other centroid in the image (see figure 57). We can take this visualization and reconstruct it over the original partitioned image as shown in figure 56. The visual data is also captured in an Excel spreadsheet that is generated courtesy of an Apache POI API that is discussed elsewhere in this document. At the time of this writing, the document generated two tabs of data with the first tab containing the moments and the second containing the angle differences for all the segments in the picture (figures 59a-59b).

Moment	X	Y	Distance (1
0	580.6345	170.4199	0
1	467.3142	305.5315	176.3423
2	553.4208	210.0813	48.10001
3	594.2125	156.9083	19.15533
4	656.5	154	77.62212
5	728.0353	169.9	147.4017
6	595.8202	184.236	20.53021
7	645.75	236.0357	92.44167
8	684.5	237	123.3732
9	690.5	237	128.4653
10	697	237	134.0666
11	702	237	138.4287
12	718.0952	237.8095	153.0909
13	617.8551	251.6812	89.37986
14	655.6852	252.7778	111.4245
15	635.75	251.25	97.83261
16	693.3333	251.8333	139.0294
17	713.4211	253.8947	156.8449
18	977	269.5	408.5615
19	831.5	273	271.028
20	759.8111	293.6556	217.4656
21	213.6298	309.8615	392.602

Figure 59: Excel spreadsheet data from passenger side of dump truck model image (moments) (a)

Node	Θ1	Θ2	Size/Area
0	-82.522	-91.707	573903
1	-91.707	-84.3282	401810
2	-84.3282	-81.628	2766
3	-81.628	-77.0363	222
4	-77.0363	-71.5649	0
5	-71.5649	-81.1868	160
6	-81.1868	-76.3704	82
7	-76.3704	-73.2293	26
8	-73.2293	-72.7545	0
9	-72.7545	-72.2428	0
10	-72.2428	-71.8511	0
11	-71.8511	-70.5805	0
12	-70.5805	-78.4017	17
13	-78.4017	-75.2045	65
14	-75.2045	-76.8988	47
15	-76.8988	-72.1558	2
16	-72.1558	-70.5018	9
17	-70.5018	-51.9898	15
18	-51.9898	-61.0348	0
19	-61.0348	-65.5831	0
20	-65.5831	-114.732	82
21	-114.732	-80.2352	844

Figure 59: Excel spreadsheet data from passenger side of dump truck model image (angle differences) (b)

Given that this is a model image, the most important components of data needed for future comparisons against sample images is kept in a three tables of the obstruction database. In the first, shown in following figure, the reader will note the segment identifier, source model file name, the dimensions of the extracted segment, the beginning portion of the chain code for the border region of the segment, and the length of the chain code.

```
0,data:IMG_5652.jpg,1,(1,1,0,0,0,0,0,0,0,0,)CC Length=7627 start(2,4) rotation=0 type S
1,data:IMG_5652.jpg,2,(1,2,2,2,2,2,2,2,2,2,)CC Length=8803 start(70,102) rotation=0 type S
2,data:IMG_5652.jpg,3,(1,2,1,2,1,2,2,2,2,2,)CC Length=3075 start(78,537) rotation=0 type S
3,data:IMG_5652.jpg,4,(1,2,1,2,0,1,0,1,1,0,)CC Length=307 start(81,609) rotation=0 type S
4,data:IMG_5652.jpg,5,(1,2,1,1,1,0,1,0,0,0,)CC Length=237 start(98,555) rotation=0 type S
5,data:IMG_5652.jpg,6,(1,2,0,0,6,5,5,7,7,0,)CC Length=43 start(102,614) rotation=0 type S
6,data:IMG_5652.jpg,7,(1,2,0,1,0,1,0,0,0,0,)CC Length=891 start(109,560) rotation=0 type S
7,data:IMG_5652.jpg,8,(0,1,0,0,0,0,0,0,0,0,)CC Length=67 start(122,573) rotation=0 type S
8,data:IMG_5652.jpg,9,(0,0,0,0,1,0,0,0,0,0,)CC Length=71 start(136,563) rotation=0 type S
9,data:IMG_5652.jpg,10,(0,0,0,0,7,0,7,0,0,0,)CC Length=453 start(137,739) rotation=0 type S
10,data:IMG_5652.jpg,11,(1,0,0,0,0,0,1,0,0,0,)CC Length=8559 start(155,562) rotation=0 type S
```

Figure 60: First few rows from local table of obstruction database containing dump truck passenger side

In the next figure, global model data consists of centroids forming a global graph, the distance from the start node to each segment node, the lower and upper angle threshold measurements from the start node to the segment, and the size in pixels of the segment.

0, moment(579,181), distance=0.0, theta1_angle=60.30807747675438, theta2_angle=79.76102595578334, segment_area=4 pixels
1, moment(582,291), distance=110.53673861803598, theta1_angle=79.76102595578334, theta2_angle=167.43206397926963, segment_area=22375 pixels
2, moment(321,187), distance=258.28101524568524, theta1_angle=167.43206397926963, theta2_angle=-27.59255401816102, segment_area=23397 pixels
3, moment(624,98), distance=93.67947569580227, theta1_angle=-27.59255401816102, theta2_angle=-92.98618225725434, segment_area=0 pixels
4, moment(552,115), distance=71.10325412342182, theta1_angle=-92.98618225725434, theta2_angle=-27.186350395021908, segment_area=960 pixels
5, moment(616,103), distance=85.3328070262154, theta1_angle=-27.186350395021908, theta2_angle=91.94239442391157, segment_area=252 pixels
6, moment(551,215), distance=44.362943197802686, theta1_angle=91.94239442391157, theta2_angle=-18.759383517974673, segment_area=1 pixels
7, moment(574,128), distance=52.25669970766421, theta1_angle=-18.759383517974673, theta2_angle=35.582742891626744, segment_area=360 pixels
8, moment(564,143), distance=40.94069531968483, theta1_angle=35.582742891626744, theta2_angle=13.170203965507124, segment_area=336 pixels
9, moment(749,181), distance=170.21619686552907, theta1_angle=13.170203965507124, theta2_angle=133.03836115151793, segment_area=21933 pixels
10, moment(364,338), distance=266.49859638964676, theta1_angle=133.03836115151793, theta2_angle=10.143140270187354, segment_area=3951 pixels10,data:IMG_5652.jpg,11,(1,0,0,0,0,0,1,0,0,0,)CC Length=8559 start(155,562) rotation=0 type S

Figure 61: First few rows from global table of obstruction database containing dump truck passenger side

In the third table, the global model metadata consists of graph similarity index scores for each model image and its rotated segments as derived from the Delaunay graph for the centroids of the segments. The chain codes and the graph similarity index scores are primary components used in the identification of partially obstructed images.

```
Model data/IMG_5652.jpg has sim_g score 0.2698234014596443
Model data/IMG_5652_r135.jpg has sim_g score 0.5439556369824528
Model data/IMG_5652_r180.jpg has sim_g score 0.9184428315581111
Model data/IMG_5652_r225.jpg has sim_g score 0.19792206301429402
Model data/IMG_5652_r270.jpg has sim_g score 0.49525101763907736
Model data/IMG_5652_r315.jpg has sim_g score 0.4865866430355273
Model data/IMG_5652_r45.jpg has sim_g score 0.0
Model data/IMG_5652_r90.jpg has sim_g score 0.15904050283669358
```

Figure 62: First few rows from global meta table of the obstruction database containing dump truck Delaunay graph similarity scores for the driver side of the dump truck and rotations of that image

The last table was created in more recent updates to the research code based on the generation of a Delaunay graph using several different OpenCV libraries. The code to do this is as follows:

```
Rect r = ProjectUtilities.calcRectCoveringPts(centroid_array);
Subdiv2D subdiv = new Subdiv2D(r);
int ptCnt = 0;
for (Point p : centroid_array ) {
        if (!Double.isNaN(p.x) && !Double.isNaN(p.y)) {
                subdiv.insert(p);
                ptCnt++;
        }
}
MatOfFloat6 triangleList = new MatOfFloat6();
subdiv.getTriangleList(triangleList);
List<Point> convertedTriangleList =
ProjectUtilities.convertMatOfFloat6(triangleList);
```

Figure 63: Code to generate Delaunay Triangulation

There is no direct method to generate a Delaunay graph using the OpenCV library. Instead, the code starts by generating a rectangle covering the centroids of the segments in the image. Next, a data structure useful in working with 2D planar subdivisions is allocated. After this, the centroid points are added into this data structure object before calling its `getTriangleList` method, which useful the supplied information to generate the Delaunay triangulation in the `triangleList` data structure. Specifically, `triangleList` is a `MatofFloat6` data structure contains sets of triad coordinates corresponding to the vertices

of the triangles making up the Delaunay triangulation. An example of Delaunay triangulation superimposed over the segmented image of the dump truck is shown in the following figure:

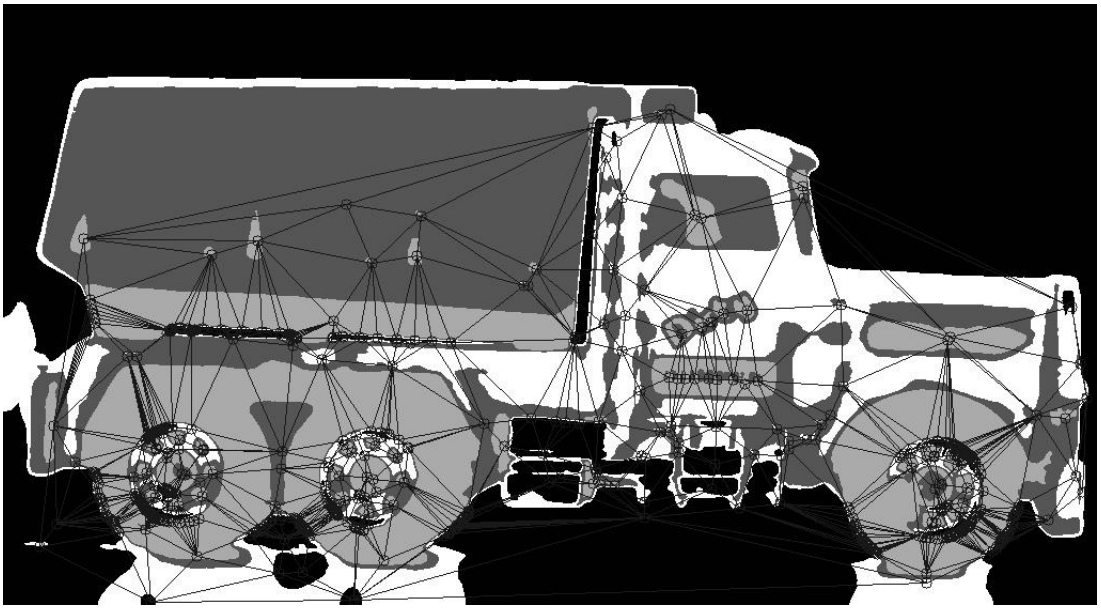


Figure 64: Delaunay Triangulation superimposed over Segmented Image

After the Delaunay graph is generated, it is passed to the routine needed to calculate angle differences for the triangles in the graph. Finally, the information is passed to a routine to calculate its graph similarity score.

OBJECT EXTRACTION

In this work, it is important to understand that what we take to be an object with the naked eye is not necessary how the computer represents an object in a scene. It is possible for us to build algorithms and their corresponding data structures in such way that the code interacts with an object in a scene that is a part of a larger object, a component in an object, an amalgamation of components in an object, an amalgamation of objects in a scene, and so forth. What is important to realize is that we want to extract objects in such a manner that they make future detection against a set of known objects as accurate and speedy as possible given the tradeoffs between those characteristics.

The process of object extraction, whether with the model image or a sample image, begins with the clustering of the image into a set of regions through the use of a standard K-means data grouping algorithm into four sets with no more than sixteen iterations of the algorithm or a change in the center of the clusters of no more than one pixel. There is a tradeoff here between the number of sets, terminal criteria, input image sizes (normally 1024x562, but could vary) and resource constraints for both building the model database and for matching using the suite of detection methods. The greater the number of sets, the greater the potential of improved accuracy, but the computational costs increase significantly to achieve these improved rates of accuracy. After this, the clusters are scanned and segments are generated through a region growing and padding process. The border regions making up these segments are then used as basic unit of input into the generation of the chain codes, derived lines and properties of the chain codes, various other metadata measures (such as the angles of the

centroids between subsequent pairs of segments), etc. This helps to generate the global graph structure that provides additional aids in the matching of obstructed images. The database is used to maintain this robust collection of data (see section three for information on how the model exemplars are stored). At the time of this work, the global graph structure is not just the location of the centroids of the segmented regions, which helps to form a graph structure that can be used for additional analysis, but is part of a LGNode object that contains information processing times, the curved line segment metadata information, the pixels making up the border (in OpenCV Matrix format), the number of pixels that constitute the border, an identifier, and the partitioning algorithm used to build that node among other units of information. This work is not presently making extensive use of the curved line segment metadata information, but it is certainly an area of future work that might be promising to draw upon for future use in the machine learning algorithms as additional attributes used in the Delaunay graph matching method or in a separate pass that uses these attributes along with chain codes or other datasets.

OBJECT REPRESENTATION

In this work, the fundamental form of object representation is the LGNode. The following instance variables are shown in the following code segment:

```
public class LGNode {
	/* location of the node, correspondent to region's centroid
	 * or the center of gravity */
	private Point center;

	// Region stats
	private HashMap<String, Double> stats;
```

```
        /* Contains average of all rows, cols, and intensity for region
           Note there are no stats for regions created by region growing
           process where small regions are inserted within the original
           clusters */
        public HashMap<String, Double> getStats() {
              return stats;
        }

        // local graph associated with this node (region)
        private ArrayList<CurveLineSegMetaData> L;

        // object contour pixel set
        private Mat border;

        // number of pixels belonging to this region
        private long size;

        // Partitioning algorithm used
        private ProjectUtilities.Partitioning_Algorithm pa =
                    ProjectUtilities.Partitioning_Algorithm.NONE;

        // Node index for LG Algorithm processing
        private int node_id = -1;
```

…

Note that every node object corresponds to one part of the image as segmented by the K-Means algorithm in the execution of the L-G graph algorithm. The collection of LGNode objects forms a global graph that is stored within a traditional Java ArrayList data structure. Inside of each node is a Java Point object representing the X,Y coordinates of the centroid, the set of curved line segment objects making up the segmented region, the object contour pixel set in OpenCV matrix form, the size of the region in pixels, an identifier corresponding to the order in which it was processed by the L-G Algorithm (and which for model images will match the value used for storage in the database), etc. In fact, the OpenCV Matrix objects, which are essentially

sophisticated floating-point or integer based array data structures, hold the majority of the image data in various intermediate forms in this work.

The set of curved line segment objects making up the segmented region has the following in-code representation of class instance variables:

```
/**
 * Based on SH = U{Ln_j, R^c_j,j+1, Ln_j+1}
 * with S = Ln_n1 . R^c_12, Ln_n2, ...
 * Lj = {sp, l, d, cu}
 * sp=starting point
 * l = length
 * d = orientation
 * cu = curvature
 *
 * @author mrobbeloth
 *
 */
public class CurveLineSegMetaData {
	private Point sp;		// starting point of a curved line segment
	private Point ep;		// ending point of a curved line segment
	private double length;	// length of curved line segment (pixels)
	private double orientation;	// in degrees
	private double curvature;	//
	private long lineNumber;	// sequence number
	private long totalTime;  // total time to calc curved line segment in ns
	private ArrayList<CurveLineSegMetaData> connList;

…
```

The definition stored in the instance variables of this class come from [53] using the following formulations:

$SH = \cup \{Ln_j \cdot R^C_{j,j+1} \cdot Ln_{j+1}\}$; two or more consecutive lines and the relationship between them

Where S = $Ln_{n1} \cdot R^C_{1,2} \cdot Ln_{n2} \cdot R^C_{2,3} \cdot Ln_{n3} \cdot R^C_{3,4} \cdot Ln_{n4} \cdots Ln_{n-2} \cdot R^C_{1,2} \cdot Ln_{n-2n-1} \cdot R^C_{n-1,n} \cdot Ln_n\}$

SH represents the of a set of curved line segments that are related to one another by connectivity, parallelism, relative distance, relevant magnitude, and/or symmetry. A curved line segment possesses the starting point, length, curvature, and orientation properties at a minimum. The data stored in the objects in this class will serve an important role in future extensions to this work.

DEPLOYMENT

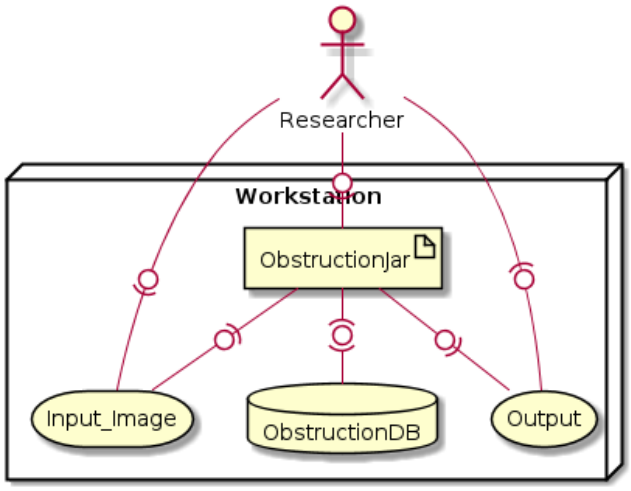


Figure 65: Obstruction Deployment Diagram

The obstruction code, in its current form, has a simple deployment format as shown in figure 65. The researcher interacts with a workstation, supplying it with the necessary digital images for input as part of modeling (e.g., exemplar database building) and sampling activities. The primary executable is a Java Archive (JAR) file containing the necessary Java encoded classes and resources for loading and execution by the Java runtime system on the workstation. The Obstruction code then reads from and writes to the HyperSQL database (DB) to hold the key characteristics and analyzed data from a set six-sided model images. The code will generate various outputs depending on whether a model or sample image is being processed. The outputs for processing model images can be more verbose when a debug flag is set in the code.

III. DATA REPRESENTATION WITH MULTI-VIEW MODEL DATABASE

All the model images in this work are represented by six views: top, bottom, forward, rear, left, and right sides. Attempts were made to capture model images under controlled conditions where it was possible to prevent spurious reflections that might cause irregular boundaries or other sharp intensity changes due to modifications of areas of light and dark, thereby creating phantom segments. Interestingly enough, the six-sided set of views provides us with enough information theoretically to construct an approximate 3D representation of the objects for identification. For the current work, having a set of six-sided model images allows for the extraction of multiple datasets that allow for determination of multiple properties that are encoded in a database appropriate manner. These views are not necessary orthogonal, though they are approximately orthogonal in nature subject to conditions under which the model photographs were taken. It is the image contents and their properties that are of interest here for this work (e.g., the object's spatial structure). A subset of the images in this work found is found in following several figures.

Figure 66: Subset of Cropped and Color Balanced Source Images

Figure 67: Subset of Grayscale Source Images

Figure 68: Later revision of model and sample images (one distractor image)

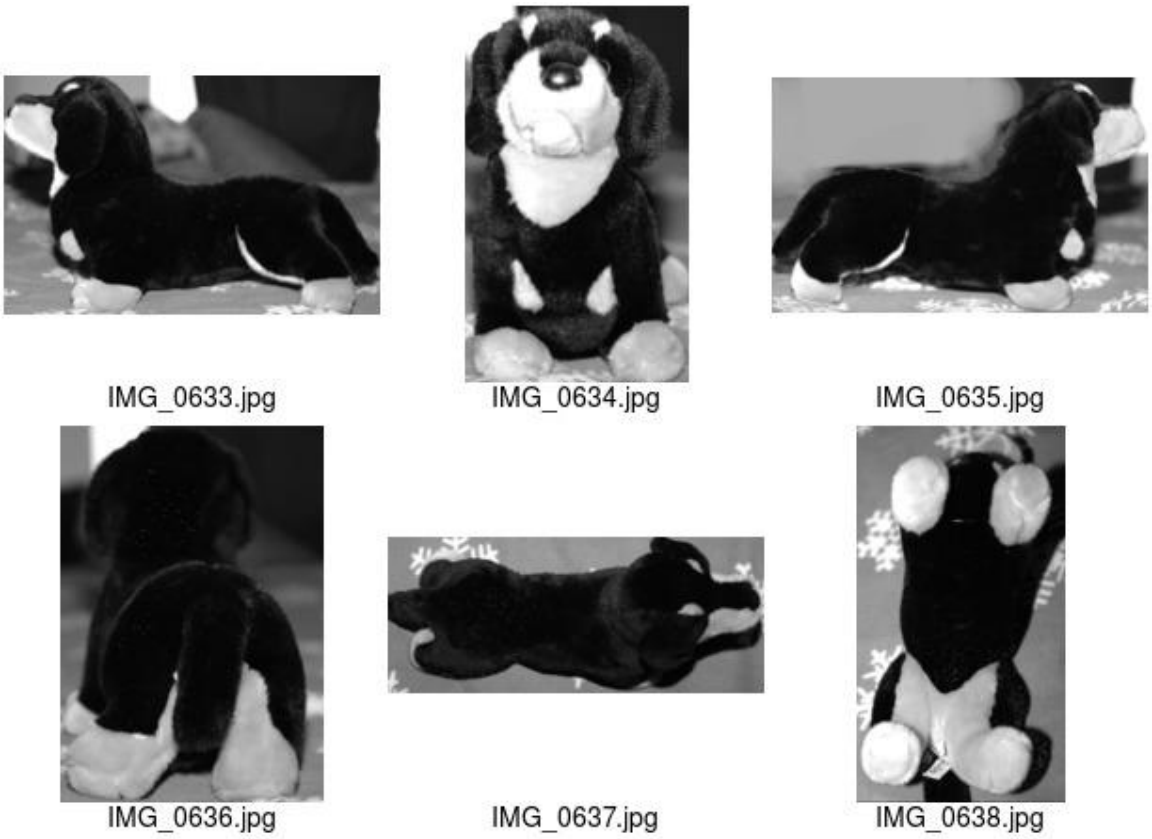


Figure 69: Close-up of Distractor Image Views in Model Database

Most of the model images in this work are toy vehicles with the occasional distractor image such as the stuffed toy dachshund. The images were shot in both the cameras proprietary RAW image format as well as its natively converted Joint Photographic Experts Group (JPEG) format. The original model images were capture using f/5 aperture priority mode (where the f/stop or aperture value is fixed) with 50 mm focal length on a Manfrotto tripod mount, ISO 400 speed, and a variable exposure time of $1/6^{th}$ to $1/50^{th}$ of a second. The RAW images from the first camera had a resolution of 3888x2592 with file size of 9 to 10 MB prior to any processing. The later model images were taken in aperture priority mode at f/stop value f/4 at ISO 100 speed

and variable exposure time of 1/13th to 1/20th of a second. The later images had an original resolution of 4000x6000 with the RAW images being 28-30 MB in size and the JPEG images being approximately 4-7 MB in size. All JPEG images were originally processed from sRGB color profile. These *lower* f/stop values force the lens to focus on object itself and leave the background blurry, which will help in speeding up image processing tasks along with improving overall accuracy by removing unnecessary details. The images were cropped to varying sizes from 662x611 to 1809x993 and the RGB color curve profiles were balanced. Finally, the images were converted to grayscale reducing image sizes to 500 KB to 1 MB. When deciding on an appropriate capture mode, it was decided that it was more important to have uniform levels of sharpness in the photos for feature extraction than it was to have an optimal lighting profile, which lead to the choice of using aperture priority mode over exposure priority or fully manual modes. In order to ensure that the images were tac sharp when captured at the slower speeds, the camera was tripod mounted and a flash was used. The original sets of images were taken using a Canon EOS Digital Rebel Xs digital camera. Later image sets were captured using a Canon Rebel 77D. Some of the images in the one proof sheet with artificial defects in them represent some of the sample images that are used, but were not used in building the database of images. The sample images are processed against a model database once it has been built. In all cases, the images are down sampled from full color images to grayscale ones at a reduced resolution. Finally, for completeness, the contact (e.g., proof) sheets was generated using the Linux montage utility using variations of the command similar to this one: montage -verbose -label

'%f' -font Helvetica -pointsize 14 -background '#ffffff' -fill black -define jpeg:size=200x200 -geometry 200x200+2+2 -auto-orient IMG_*.jpg <destination filename>.

The images themselves are not stored in the database, but their processed representations are stored in the database in four key tables: OBSTRUCTION_LOCAL, OBSTRUCTION_GLOBAL, and OBSTRUCTION_GLOBAL_META, and OBSTRUCTION_DEL_GRPH_TABLE. The overall organization and schemas for the database can be seen in the next several figures and were generated by the open source SchemaSpy Java program, which is also included in the docs subdirectory of the obstruction codebase on GitHub. The source images are kept in their original and preprocessed formats for reprocessing as the database format and algorithms making up this work are improved. The processed data is kept in a traditional Structure Query Language (SQL) that is managed by the HyperSQL 2.4.1 or later engine due to its solid support of SQL constructs and tight integration with the Java programming language using a single Java Archive (JAR) library file. It does not require the configuration of server instances, virtual machine or container instances in the cloud The downside to using this particular relation database management software (RDMS) is that it does not scale well for intensive computational use on workstations.

It is important to note that we do not store any of the matrices holding actual image data. It was discovered in earlier versions of the code for this work that is too resource intensive to maintain Binary Large Object (BLOB) fields and/or process those BLOB fields efficiently. Instead, the lines making up the border region of the segments are maintained as Character Large Object (CLOB) fields. In the OBSTRUCTION_LOCAL relation, information on each segment's local

properties are kept. This includes an auto-incrementing identifier along with the file to which the segment belongs, its border line chain code representation, and the coordinates at which each set of border pixels begin for the segment. Furthermore, if the segment comes from an image that has been rotated internally by the OpenCV API, the amount of rotation is noted (it will be in increments of 45 degrees, matching the various cardinal directions). Finally, the type of segment is noted, which can be one of the following: standard (S), synthesized (Y – built from a base segment and a merging segment in what is called a merged segment in an outward fashion from the beginning segment of the image), or rotated (R). In the OBSTRUCTION_GLOBAL relation, global properties for each segment are kept, including the data needed to build the overall graph structure of centroids for the segments. This includes an auto-incrementing identifier that is cross-referenced with the one in the local relation as a foreign key (it is also a primary index in the global relation), the first moment or centroid of the segment (previously in the local relational and moved into this during the first major database code rewrite), the distance in pixels from the start node to the centroid of each segment, and the angle of measurement from the start node to each segment's centroid. In the OBSTRUCTION_GLOBAL_META table, the graph similarity score from the Delaunay construction of the global graph is kept for each model image. In the final relation, OBSTRUCTION_DEL_GRPH_TABLE, the data needed to construct the Delaunay triangulation of the global graph is stored. Specifically, the (X,Y) coordinates of each triangle and associated model filename is kept in each record of the relation.

A few other minor notes on database limitations and operation. One limitation is that synthesized rotated segments are not considered. The chain code, being represented by a character blob field (CLOB), will be stored in a separate lobs file by the HyperSQL database. The lobs extension file is used to hold character and binary blob fields. At the time of this work, compression is not being used on the lobs file to allow maximum access speed to its contents, allow this is still a slow process. Some may wonder why a VARCHAR or LONGVARCHAR was not used. Although the average size of CLOB field is 4.2 KB, well within the use of a VARCHAR; the maximum size that has occasionally been observed is 33 KB. This could be obtained simply with the use of: `select avg|max(length(CHAINCODE)) as avg|max_length from obstruction`. Once a character field is more than a few kilobytes in size, it is recommended to use CLOBs [58]. The mixture of very short chain codes and quiet long chain codes in other cases makes it hard to find a good datatype for use with the storing of the chain codes. Earlier experiments trying to store and use OpenCV matrices in a BLOB field resulted in acceptably low performance on lower-end hardware machines. Therefore, even for a small set of images, storage requirements are on the order of gigabytes. However, the files do compress quite nicely (over 99%), allowing for the easy transfer of the database between multiple machines where I can have experimental studies running on some of the systems, and do new development work or review on other machines.

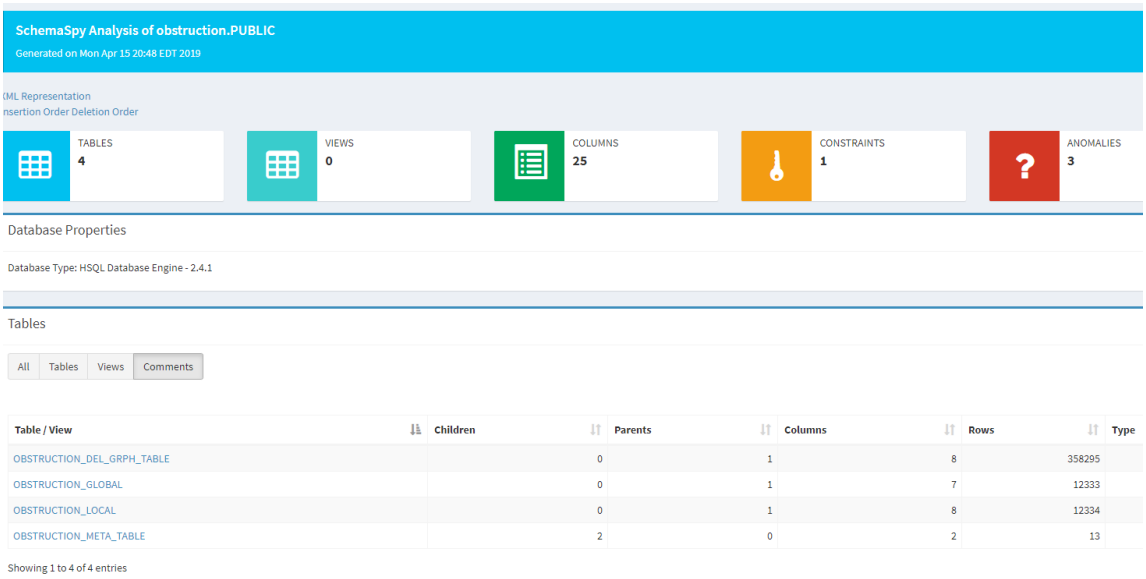


Figure 70: Relations (tables) in obstruction code

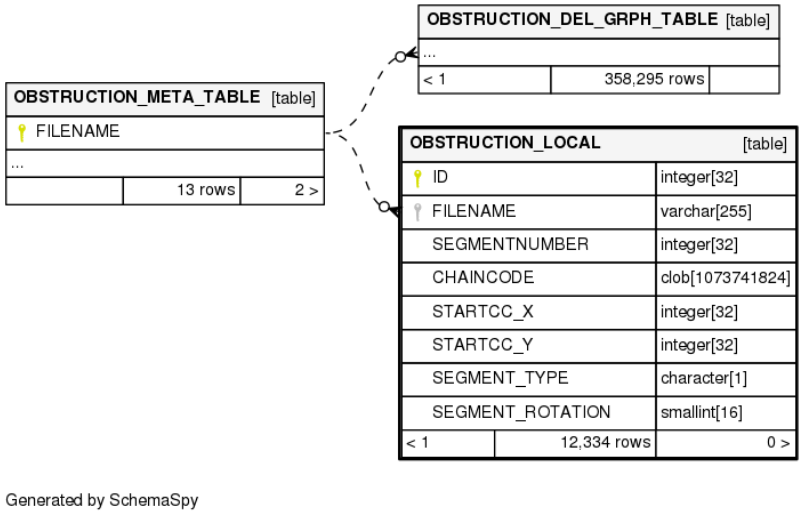


Figure 71: Schema View of Local Relation of the Obstruction Database

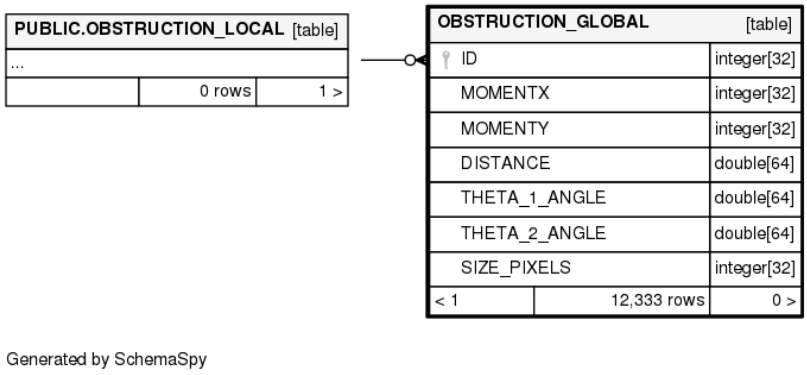


Figure 72: Schema View of Global Relation of the Obstruction Database

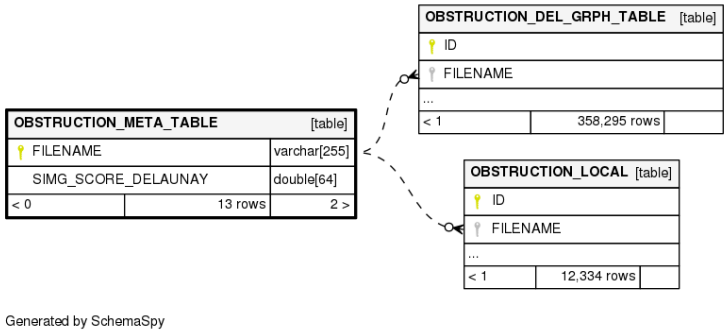


Figure 73: Schema View of Global Meta Relation of the Obstruction Database

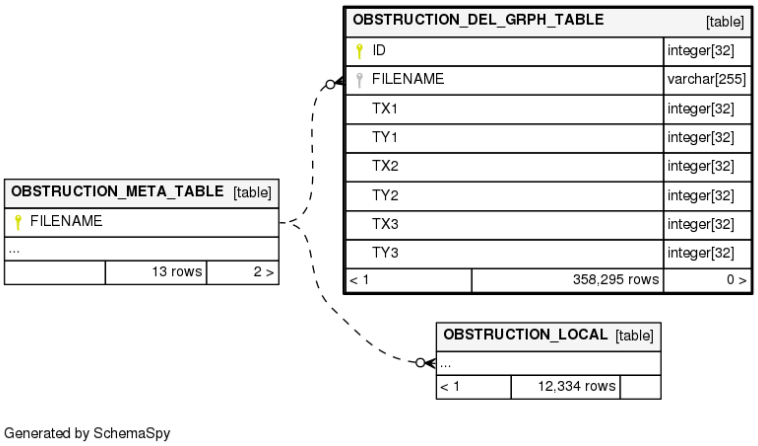


Figure 74: Schema View of Delaunay Graph Relation of the Obstruction Database

IV. INCOMPLETE RECOGNITION

CHAIN CODE MATCHING

One of the techniques used in this work for the probabilistic matching of candidate images against the body of six sided model images is to use string similarity measures on the chain codes describing the border regions of the nodes of the images. The nodes are those generated from the application of the local-global graph process. There are a variety of measures that can be used and this work used a Java based string similarity library developed by [15]. The measures include:

- Levenshtein
- Normalized Levenshtein
- Damerau-Levenshtein
- Optimal String Alignment (OSA)

- Jaro-Winkler
- Longest Common Subsequence (LCS)
- Metric Longest Common Subsequence (MLCS)
- NGram Distance
- QGram (Ukkonen) Distance
- Cosine Similarity
- Jaccard Index
- Sorensen-Dice coefficient

The algorithm used to call the various string similarity algorithms for the purpose of finding a match using chain codes follows a similar process to the one used for Q-Gram matching in the following activity diagram:

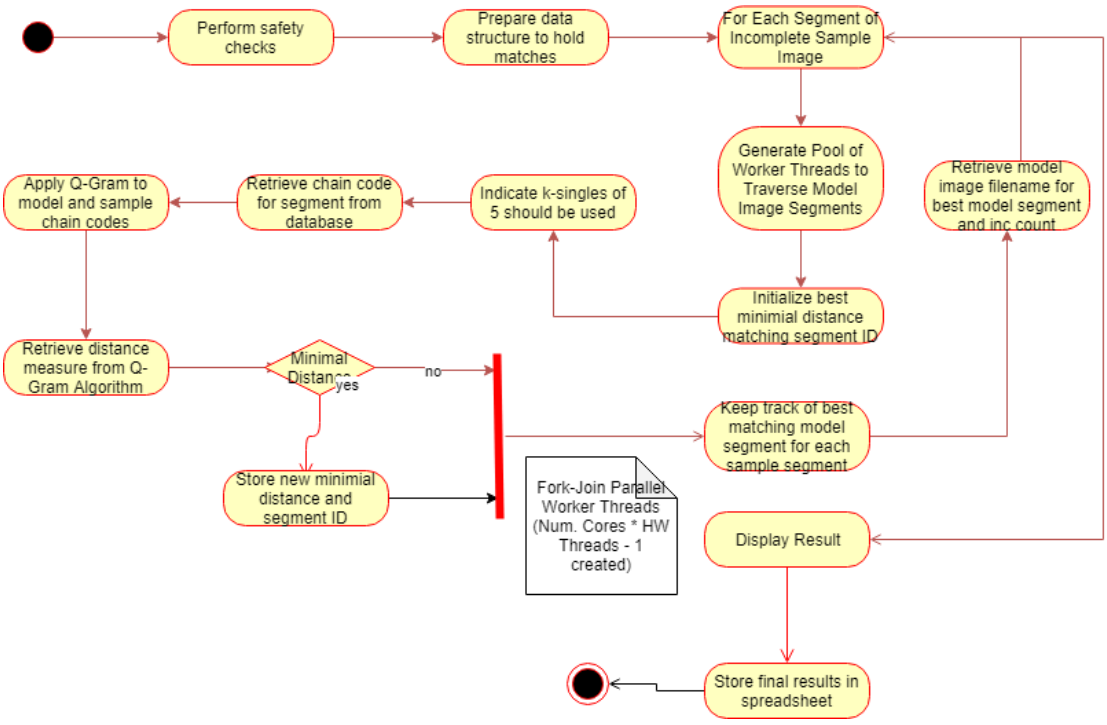


Figure 75: Match to Model by Q-Gram Distance Activity Diagram (Chain Code Matching Example)

There are some minor differences for the setup of parameters in the other chain code related matching routines, but the workflow is basically the same. It is interesting to note that one of performance improvements used for later versions of this work was to create a pool of worker threads and have them work through the all the model segment data stored in the database. The λ expression used to generate the pool was `IntStream.rangeClosed(0, lastEntryID).parallel().forEach(i) -> { … });`.

Initially, a series of experimental runs were performed to evaluate performance and accuracy of the different string similarity algorithms against several sample images. In the first

test, model image IMG_5652 was run against the set of preprocessed model images in the database, and the following results were obtained:

Algorithm	Best Match	Probability of Match
Levenshtein	IMG_5652	100%
Normalized Levenshtein	IMG_5652	100%
Damerau-Levenshtein	IMG_5652	100%
OSA	IMG_5652	100%
Jaro-Winkler	IMG_5652	100%
LCS	IMG_5652	100%
MLCS	IMG_5652	100%
NGram	IMG_5652	100%
QGram	IMG_5652	100%
Cosine Similarity	IMG_5652	100%

In the next experiment, a large portion of the same image was obstructed by a simulated black object with no transparency. The obstruction covered approximately 1/3 to 1/2 of the image's right side (e.g., the forward cabin portion of the passenger side perspective of the dump trunk).

Algorithm	Best Match	Probability of Match

Levenshtein	IMG_5652	70.09%
Normalized Levenshtein	IMG_0634	82.75%
Damerau-Levenshtein	IMG_5652	70.09%
OSA	IMG_5652	70.09%
Jaro-Winkler	IMG_0638_r45	41.03%
LCS	IMG_5662_r315	15.81%
MLCS	IMG_5652	71.79%
NGram	IMG_5652	Thread did not finish
QGram	IMG_5652	73.50%
Cosine Similarity	IMG_5661_r270	56.65%

In the next image, the obstruction was switched to back 1/3 to 1/2 of the image.

Algorithm	Best Match	Probability of Match
Levenshtein	IMG_5652	62.44%
Normalized Levenshtein	IMG_5652	62.44%
Damerau-Levenshtein	IMG_5652	62.44%
OSA	IMG_5652	62.44%
Jaro-Winkler	IMG_5655_r255	35.21%
LCS	IMG_5662_r315	14.09%
MLCS	IMG_5652	63.38%

NGram	IMG_5652	
QGram	IMG_5652	62.26%
Cosine Similarity	IMG_5661_r270	54.55%

After this image was tested, the background was removed and only the foreground toy truck was left. The area surrounding the toy truck consist of all black pixels.

Algorithm	Best Match	Probability of Match
Levenshtein	IMG_5652	67.56%
Normalized Levenshtein	IMG_5652	67.56%
Damerau-Levenshtein	IMG_5652	67.56%
OSA	IMG_5652	67.56%
Jaro-Winkler	IMG_0638_r45	43.16%
LCS	IMG_5662_r315	16.09%
MLCS	IMG_5652	68.10%
NGram	IMG_5657	91%
QGram	IMG_5652	69.44%
Cosine Similarity	IMG_5672_r45	37.14%

The fourth partial image is the passenger side of the toy dump truck with five vertical black bars placed at even lengths across the image. The results of running this partial sample image against the database are as follows:

Algorithm	Best Match	Probability of Match
Levenshtein	IMG_5652	57.40%
Normalized Levenshtein	IMG_5652	57.40%
Damerau-Levenshtein	IMG_5652	57.40%
OSA	IMG_5652	57.40%
Jaro-Winkler	IMG_0638_r45	40.26%
LCS	IMG_5662_r315	14.55%
MLCS	IMG_5652	57.14%
NGram	IMG_5652	Thread crashed
QGram	IMG_5652	59.21%
Cosine Similarity	IMG_5652	71.85%

The fifth partial image is the passenger side of the toy dump truck with three horizontal black bars placed at even lengths across the image. The results of running this partial sample image against the database are as follows:

Algorithm	Best Match	Probability of Match
Levenshtein	IMG_5652	66.14%

Normalized Levenshtein	IMG_5652	66.14%
Damerau-Levenshtein	IMG_5652	66.14%
OSA	IMG_5652	40.21%
Jaro-Winkler	IMG_0638_r45	39.95%
LCS	IMG_5662_r315	13.23%
MLCS	IMG_5652	39.69%
NGram	IMG_5652	Thread crashed
QGram	IMG_5652	39.69%
Cosine Similarity	IMG_5672_r45	41.56%

The next obstructed image is the passenger side dump truck, which is rotated 45% counter-clockwise, but has no obstruction regions. The matching procedure definitely shows degradation for the most part.

Algorithm	Best Match	Probability of Match
Levenshtein	IMG_5652	40.21 %
Normalized Levenshtein	IMG_5652	40.21 %
Damerau-Levenshtein	IMG_5652	40.21%
OSA	IMG_5652	40.21%
Jaro-Winkler	IMG_0638_r45	43.81%
LCS	IMG_0634_r45	12.37%

MLCS	IMG_5652	45.53%
NGram	IMG_5652	100%
QGram	IMG_5652	39.69%
Cosine Similarity	IMG_5661_r270	47.94%

In the seventh attempt, the side car image was moderately pixelated to simulate low quality or degraded image capture say during low-light or adverse weather events. This is shown in the next figure.

Figure 76: Pixelated Partial Image of Toy Dump Truck

The results of the various string code matching algorithms on the chain codes of the various segments are shown in the following chart:

Algorithm	Best Match	Probability of Match
Levenshtein	IMG_5652	40.21%

Normalized Levenshtein	IMG_5652	40.21%
Damerau-Levenshtein	IMG_5652	40.21%
OSA	IMG_5652	40.21%
Jaro-Winkler	IMG_5652	43.81%
LCS	IMG_0634_r45	12.37%
MLCS	IMG_5652	39.69%
NGram	IMG_5652	Thread crashed
QGram	IMG_5652	39.69%
Cosine Similarity	IMG_5672_r45	39.47%

In the eighth attempt, the side car image was severely pixelated to simulate very low quality or highly degraded image capture say during low-light or adverse weather events. This is shown in the next figure.

Figure 77: Severe Pixelated Toy Dump Truck Side

Algorithm	Best Match	Probability of Match
Levenshtein	IMG_5652	12.50%
Normalized Levenshtein	IMG_5652	Thread crashed
Damerau-Levenshtein	IMG_5652	12.50%
OSA	IMG_5652	12.50%
Jaro-Winkler	IMG_0638_r180	12.50%
LCS	IMG_2420	12.50%
MLCS	IMG_5652	12.50%
NGram	IMG_5652	Thread crashed
QGram	IMG_0634_r315	12.50%
Cosine Similarity	IMG_5672_r45	47.05%

Finally, a wavy distortion pattern was applied to our sample image. Although this image and the previous two are not technically obstructed, it is important to understand the limitations of this work when using images that fall somewhat outside the scope of the current work. This is shown in the next figure.

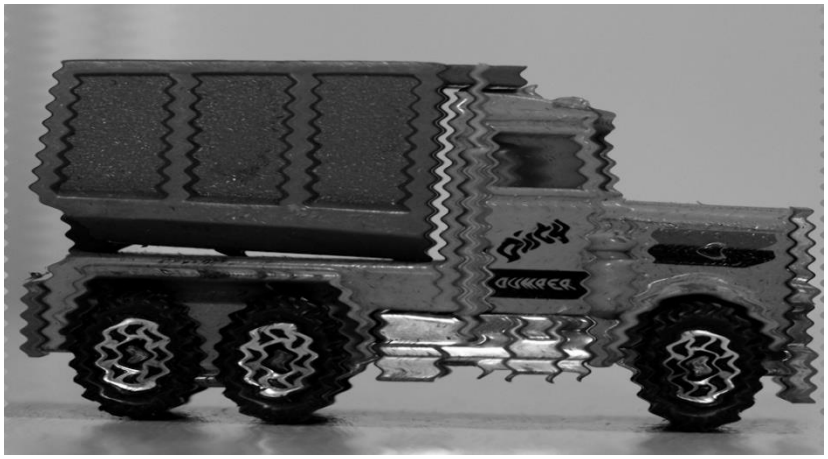


Figure 78: Wavy Distortion Pattern of Toy Dump Truck Side

The results of the various string code matching algorithms on the chain codes of the various segments are shown in the following chart:

Algorithm	Best Match	Probability of Match
Levenshtein	IMG_5652	56.52%
Normalized Levenshtein	IMG_5652	Thread Crashed
Damerau-Levenshtein	IMG_5652	63.75%
OSA	IMG_5652	63.75%
Jaro-Winkler	IMG_0638_r45	48.75%

LCS	IMG_0634_r45	15.54%
MLCS	IMG_5652	64.82%
NGram	IMG_5652	Thread Crashed
QGram	IMG_5652	64.10%
Cosine Similarity	IMG_5661_r270	42.13%

In this last attempt, although the distortion looks extreme, the segments are able to be processed in roughly the same fashion as the unmodified model image and the algorithms perform reasonably well.

In earlier attempts, out of memory errors were encountered with certain threads. However, this was corrected by ensuring a minimal heap size of 2 GB and a maximum heap size of 16 GB. The larger heap size is only really needed when constructing the database, matching runs tend to use several GB of memory. N-Gram performance was highly unpredictable due to need to configure shingle size (five was used in earlier experiments), to lead to the conclusions that the results from this matching algorithm are not particularly useful.

On the basis of these experiments, the choice was made to move forward with using Q-Gram string similarity algorithm as one of the metrics to be used in matching. Although it may not score as high as some of the other methods, it does have a linear runtime of O(m+n) where m and n are the model and candidate strings being compared. Some of the other methods had higher order polynomial runtimes on the order of O(m*n). The Q-Gram algorithm works on the

basis of finding the best substring matches of length q between two strings [62]. The greater the number of q-grams found in both the chain code of the model and the sample, the greater the likelihood that the sample image belongs to a particular model image. The downside of using this algorithm is that it has a space requirement of O (Σ^q+m+n), which results in a larger memory footprint. This is something that could limit its useful in resource constrained embedded devices. Overall, though, it is a generally good algorithm to use for matching purposes given the tradeoff between performance and accuracy. The use of secondary matching criteria, such as matching exact or similar centroids, can help improve overall accuracy and achieve acceptable results. A five-gram value was chosen to look for extended matching of subsets of chain codes between the sample and model database of images. This will correspond to matching five-pixel border sequences in the sample and model image. Although an optimized gram value for the Q-Gram algorithm was not experimentally determined, it is reasonable to hypothesize that smaller values are more likely to be subject to aliasing effects resulting from the capture or preprocessing phases and larger values are computationally more expensive to work on even given the runtimes of this algorithm. It might serve as an interesting extension to the current work to explore such an optimization as part of a larger follow-up work product. What are the statistical distributions to be found in chain codes that could guide Q-Gram value size for improved matches? Do they follow basic polygonal shapes? Do the chain codes show us a language for identifying objects that an algorithm like this can help us to capture? These are interesting questions to consider in the future. In the meantime, the codebase has been updated to support specifying which string matching algorithms to use during a matching

session. This will facilitate future experimentation with this core aspect of incomplete recognition

SYNTHESIS OF VIEWS

One of the ways to achieve a higher recognition rate is to synthesize several segments together into ever larger components until the full image is reconstructed. The original concept for the extension of the L-G Graph is shown in the following figures:

Figure 79: Synthesis Extension to L-G Graph Algorithm Concept

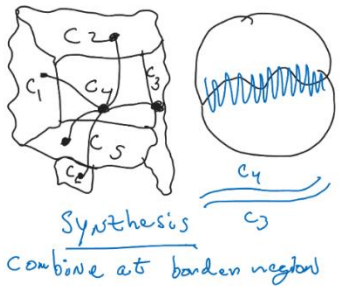
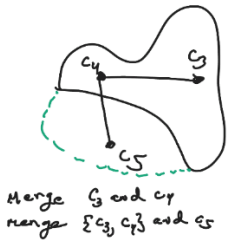


Figure 80: Synthesis Extension to L-G Algorithm Concept -- Merging Successive Regions



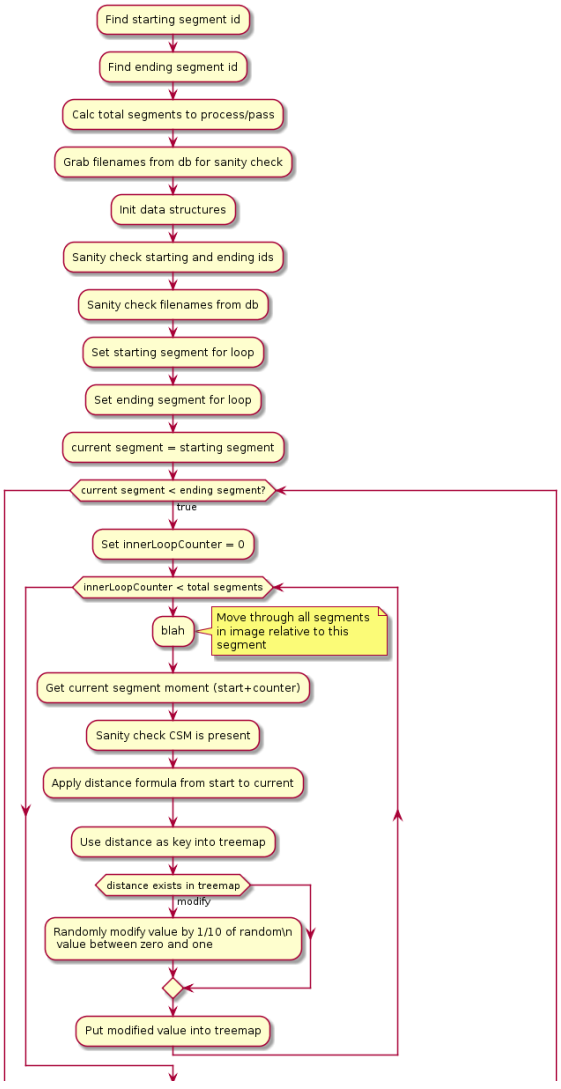


Figure 81: First part Synthesis algorithm

To implement the concept in this work, the synthesis of the model images occurs only after it has been fully deconstructed and the appropriate segment data has been stored into the obstruction database. In the original implementation, synthesis consisted of three phases: sorting, review, and merging. The first phase begins by determining which segments to work with along with a series of validation checks. Next, the distance from a starting segment's moment to every other segment's moment is measured and sorted into a Java TreeMap data structure. In Java, a TreeMap is a red-black tree. The keys used are distances. If there is a collision after calculating a particular distance between nodes, a slight random distance is added to the segment measure causing the collision. The adjustment is at most one-thousands of one-unit distance. The sorted segment distance is then displayed (optionally) if in debug mode. Finally, the segments are progressively merged in sorted

distanced order. The segments are merged using a 50% mix from both the base and merging segments with the OpenCV weighted add operator (e.g., the alpha and beta parameters are set to 0.5; gamma is not set). As a cleanup step, the combined image is processed using the OpenCV threshold operator to *refresh* the pixels in the synthesized segment image. The merged segments are then reprocessed by the L-G algorithm and the resulting base datasets are stored into the obstruction database. Presently, due to processing and memory limitations, only several of the segments are designated as start segments and processed against all the other segments in the image.

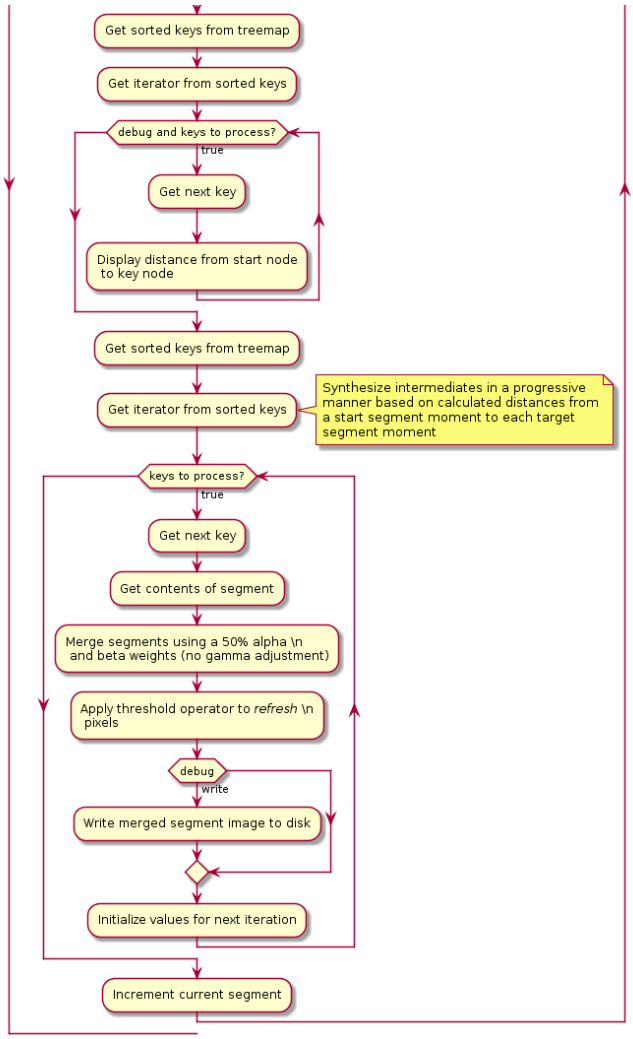


Figure 82: Second part Synthesis Algorithm

It has been found that the algorithm did not perform very well in creating merged segments from adjacent sections of the starting segment. In other words, given a starting segment, it was believed that the shortest distance segments would be found in the adjacent (e.g., bordering) segments of the starting segment. Subsequently, larger segments could then be formed or merged by connecting the neighbor segments of the starting segment's neighbor segments, and so on. However, due to the manner in which the scan segments were initially processed and grown into regions following the k-means segmentation, segments in different areas of the image had centers with very close distance measurements to the start segment. When sorting all the segments' moments by distance to the starting segment's first moment, segments from different areas of the image would be merged instead of adjacent segments.

In the second version of the synthesis algorithm, segments were merged sequentially in the same order as they were originally processed by the L-G graph algorithm. The segments were processed and grown using a row major ordering from the top-left to the bottom-right corner of the image that looked for occurrences of non-zero labels (e.g., non-white or foreground pixels indicating the presence of an object's exterior border). In other words, we take a given segment and then combine it with one of the other segments in the image and then maintain the same base image and merge it with another processed segment and so on. When processing a subsequent segment this work does not consider the previous one for the merging. Although this would be more desirable in forming more sophisticated composite objects, the computational and memory costs have proven to be too expensive given the provided hardware resources. This approach should in theory help with partial images where obstructions are not contained to one particular portion of the image. The activity diagram showing this revision of the synthesis algorithm is shown in the next diagram

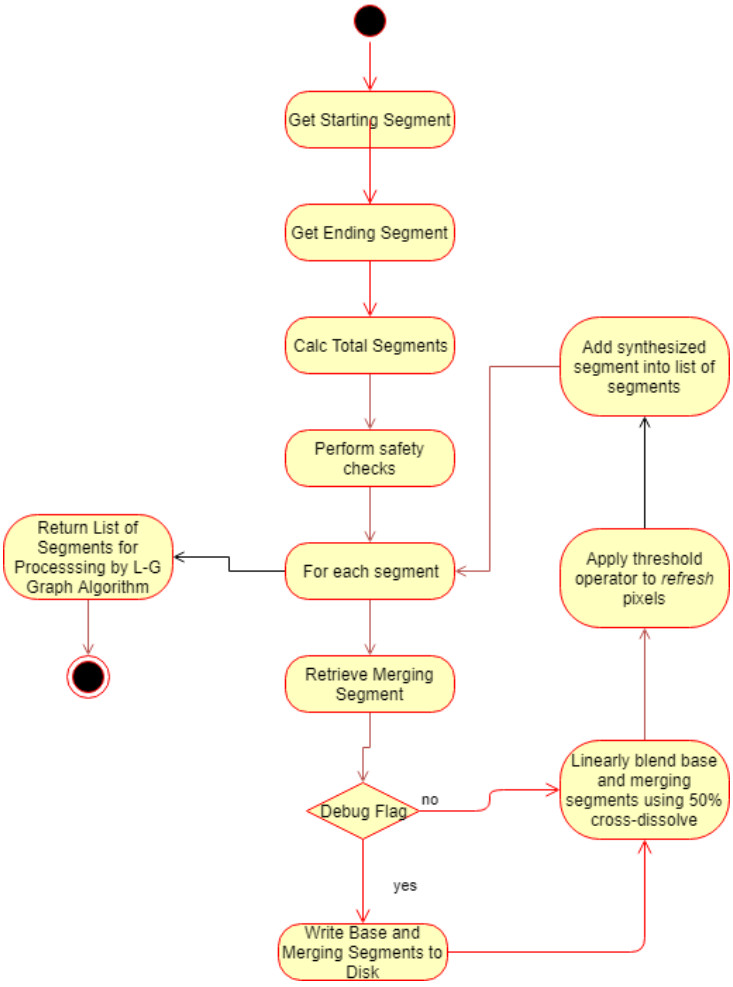


Figure 83:Activity Diagram for 2nd Version of Synthesis (Synthesis_sequential method in codebase)

The second version of the synthesis algorithm appears in the method `Synthesize_sequential` in the online codebase. The blending operation is controlled by the equation: $g(x) = (1 - \alpha)f_0(x) + af_1(x)$ [49]. The α operator is used to adjust the level of cross-dissolve between the two images. The use of a 50% cross-dissolve through by application of a threshold operator is sufficient to merge both segments. As an example of this approach consider the following three images from the dump truck example:

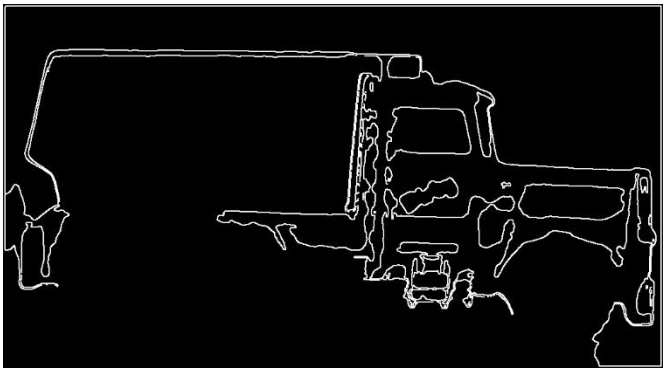


Figure 84: Base Segment 2 Toy Dump Trunk Passenger Side

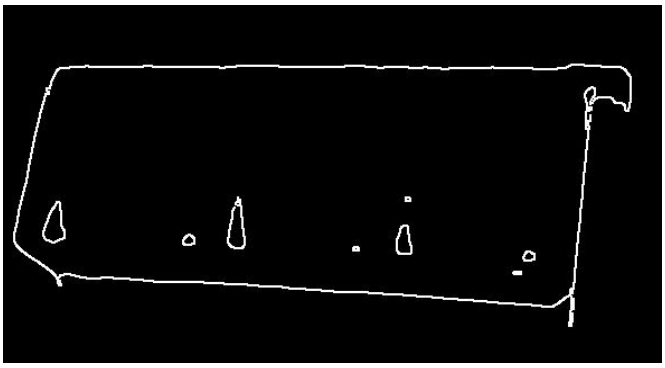


Figure 85: Merging Segment 2 Toy Dump Trunk Passenger Side

It is indeed possible that recognition rates have improved as a result of having these macro objects, but a longer-term study will be needed to determine the overall impact. It might also be necessary to better separate foreground and background artifacts to prevent unnecessarily large base segments. The increase in storage requirements for processing segments of this manner is n^2 where n is the size of the number of segments generated by the original k-means algorithm.

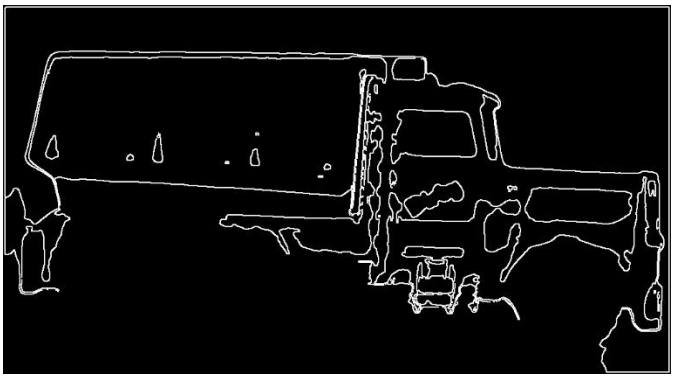


Figure 86: Merged Segment 2 Toy Dump Trunk Passenger Side

In this example, we have a large initial segment of vehicle and connected background artifacts that is being merged with another region coming from the cargo area of the vehicle that has some paint chips that were interpreted as separate borders despite smoothing and other pre-processing efforts. If, however, we have a partial sample image being processed from the passenger side where the lower half of the image is obstructed along with the wheel well at the front, this image might help us to provide a better match. Indeed, there are a variety of interesting macro object combinations that could help us to detect a variety of different obstructions to varying degrees of confidence. The downside is that the initial model processing setup is computationally intensive and protracted in duration. However, despite the issues with performing synthesis of views, this is certainly a worthy avenue of future study.

In a variation of the synthesis of views, this work wanted to consider the impact of capturing objects at an off-angle axis from one of the standard six-sided views. For this part, two new model images were processed: a toy corvette car and a front loader industrial toy vehicle.

These model images, represented by images IMG_3804 to IMG_3815 can be seen in chapter three under the revised set of model images. The off-angle sample images can be seen in the following figure:

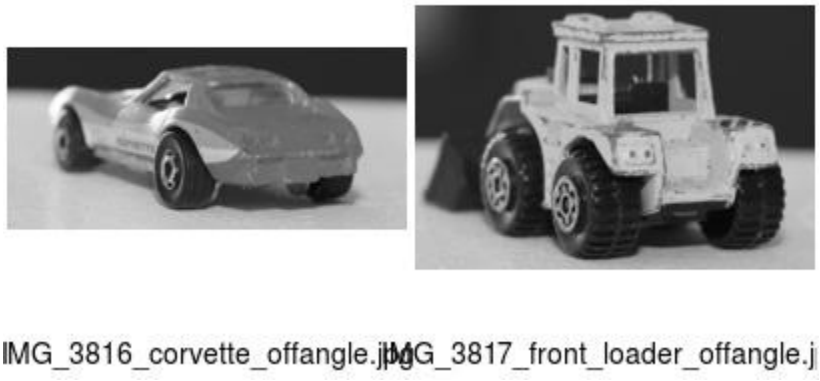


Figure 87: Off-angle Model Contact Sheet

For the corvette, in the next few figures you will observe the de-noised image of the sample (it is not obstructed for this initial test), the centroid values as visualized by PLplot, overlaid onto the segmented images by OpenCV, and the Delaunay triangulation:

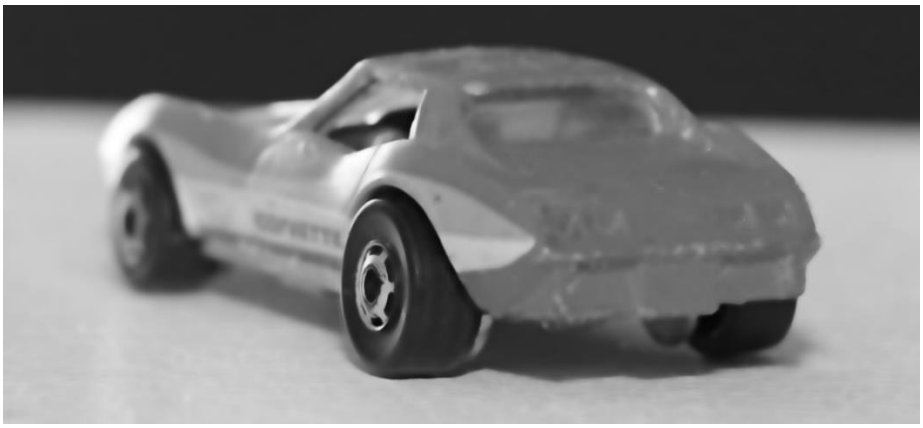


Figure 88: Corvette Off-Angle De-noised

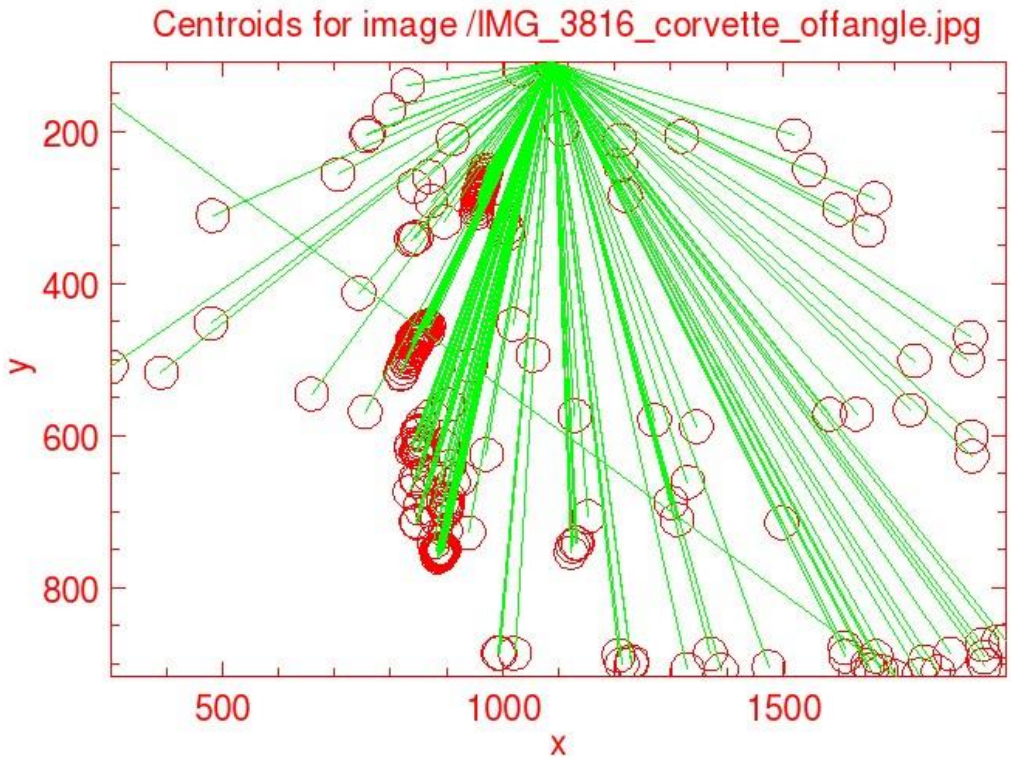


Figure 89: Corvette Off-Angle Centroid Visualization

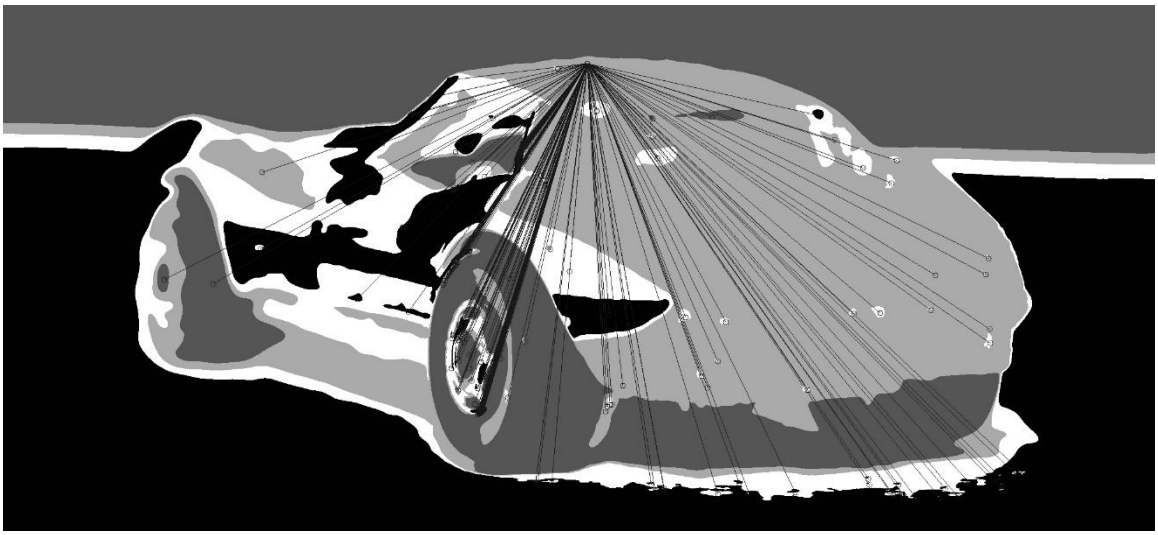


Figure 90: Corvette Off-Angle Moments Over Clustered Data

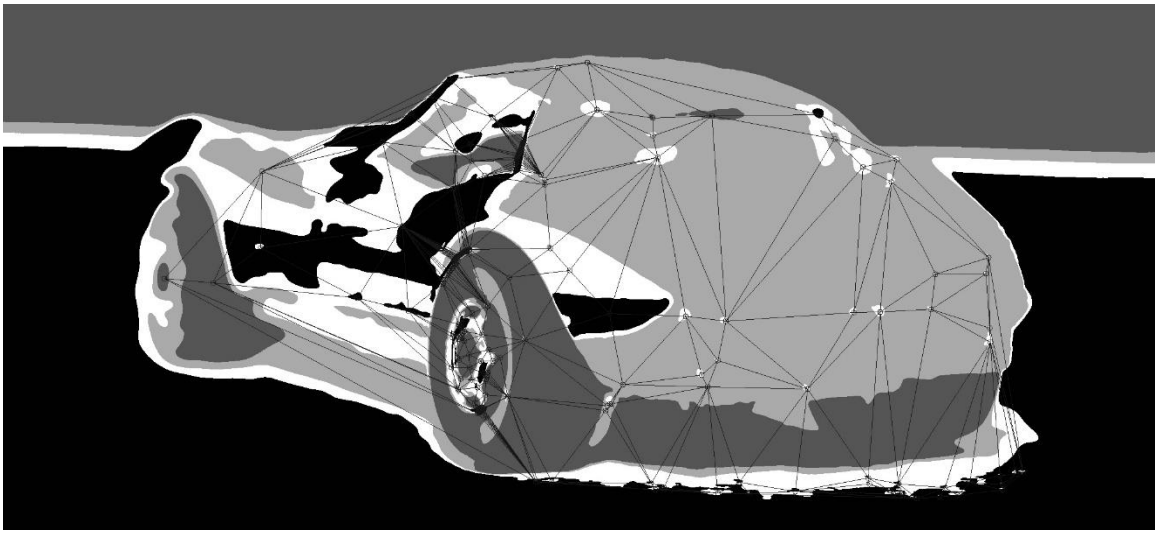


Figure 91: Corvette Off-Angle Delaunay Triangulation

The matching procedures returned disappointing results as shown in the following table:

Table 19: Partial Matching Results Off-Angle Corvette -- 1st run

	$\alpha(S_i)$	$\beta(C_i)$	$\gamma(CS_i)$	$\varepsilon(LCS_i)$	$\zeta(SIM_G)$	$\eta(WDG)$	M_j
data:IMG_3804cbg.jpg	**0.092105**	0	**0.004386**	**0.092105**	**0.952801**	**0.02739726**	**0.270876**
data:IMG_3805cbg.jpg	0.039474	**0**	0.004386	0.030702	0.781051	**0.02968037**	0.207763
data:IMG_3806cbg.jpg	0.074561	0	0.004386	0.061404	0.845377	**0.06849315**	0.235248
data:IMG_3807cbg.jpg	**0**	0	**0.004386**	**0.008772**	**0.98598**	**0.0022831**	**0.248688**
data:IMG_3808cbg.jpg	0.039474	0	0.004386	0.057018	0.561559	**0.06392694**	0.158153
data:IMG_3809cbg.jpg	**0.048246**	**0**	**0.004386**	**0.039474**	**0.98598**	**0.08675799**	**0.262065**
data:IMG_3810cbg.jpg	0.039474	0	0.004386	0.074561	0.65665	**0.1780822**	0.185434
data:IMG_3811cbg.jpg	0.039474	0	0.004386	0.035088	0.897062	**0.0141553**	0.237643
data:IMG_3812cbg.jpg	**0.504386**	**0**	**0.004386**	**0.425439**	**0.996827**	**0.0847488**	**0.410391**
data:IMG_3813cbg.jpg	0.02193	0	0.004386	0.035088	0.654812	**0.0141553**	0.174449
data:IMG_3814cbg.jpg	0.057018	0	0.004386	0.04386	0.595369	**0.2305936**	0.166605
data:IMG_3815cbg.jpg	0.017544	0	0.004386	0.061404	0.515268	**0.07534247**	0.144168
data:IMG_5652.jpg	0.026316	0	0.004386	0.035088	0.98598	**0.0130137**	0.240037

It was hoped that the strongest match would be with IMG_3805, which is the rear of the corvette. Instead, the strongest match came from the side view of the front loader, which is odd, followed by the driver side of the corvette, then the top of the corvette, and the front of the corvette. Although, far from ideal, most of the relatively stronger matches did come from the corvette model images.

As a confirmation to the first, the database was entirely rebuilt without rotations and the study ran again with the following results:

	$\alpha(S_i)$	$\beta(C_i)$	$\gamma(CS_i)$	$\varepsilon(LCS_i)$	$\zeta(SIM_G)$	$\eta(WDG)$	M_j
data:IMG_0633.jpg			0.004386		0.580809	4.57E-05	0.145641
data:IMG_0634.jpg			0.004386	0.008772	0.940321	0	0.237273
data:IMG_0635.jpg	0.004386		0.004386		0.98598	0.004566	0.247591
data:IMG_0636.jpg			0.004386		0.946881	0	0.237159
data:IMG_0637.jpg			0.004386		0.507089	0	0.127211
data:IMG_0638.jpg	0.02193		0.004386	0.013158	0.601632	0	0.156768
data:IMG_2420.JPG			0.004386		0.667652	0.027397	0.167352

data:IMG_2426.JPG		0.004386		0.98598	0	0.246934
data:IMG_2427.JPG		0.004386		0.670724	0	0.16812
data:IMG_2428.JPG		0.004386		0.633504	0	0.158815
data:IMG_2432.JPG	0.004386	0.004386		0.98598	0	0.247591
data:IMG_2439.JPG		0.004386		0.712535	0	0.178572
data:IMG_2440.JPG		0.004386		0.676507	0	0.169565
data:IMG_3804cbg.jpg	0.083333	0.004386	0.127193	0.620158	0.031963	0.193417
data:IMG_3805cbg.jpg	0.017544	0.004386	0.008772	0.804754	0.018265	0.206013
data:IMG_3806cbg.jpg	0.017544	0.004386	0.013158	0.98598	0.045662	0.252197
data:IMG_3807cbg.jpg		0.004386	0.008772	0.98598	0	0.248688
data:IMG_3808cbg.jpg	**0.333333**	**0.004386**	**0.131579**	**0.98598**	**0.02968**	**0.323249**
data:IMG_3809cbg.jpg	0.026316	0.004386	0.02193	0.691962	0.075342	0.181762
data:IMG_3810cbg.jpg	0.02193	0.004386	0.048246	0.609229	0.114155	0.165684
data:IMG_3811cbg.jpg	0.02193	0.004386	0.013158	0.98598	0.091324	0.252855
data:IMG_3812cbg.jpg	0.048246	0.004386	0.057018	0.986057	0.116438	0.265593
data:IMG_3813cbg.jpg	0.004386	0.004386	0.013158	0.98598	0.006849	0.250223
data:IMG_3814cbg.jpg	0.030702	0.004386	0.035088	0.513613	0.164384	0.140465
data:IMG_3815cbg.jpg	0.074561	0.004386	0.144737	0.98598	0.06621	0.287065
data:IMG_5652.jpg	0.004386	0.004386	0.017544	0.98598	0.022831	0.2511
data:IMG_5653.jpg	0.004386	0.004386	0.004386	0.957344	0	0.24131
data:IMG_5654.jpg	0.004386	0.004386	0.004386	0.718852	0.038813	0.181687
data:IMG_5655.jpg	0.008772	0.004386		0.98598	0	0.248249
data:IMG_5656.jpg		0.004386		0.98598	0.002283	0.246934
data:IMG_5657.jpg	0.008772	0.004386	0.02193	0.563438	0.022831	0.147
data:IMG_5658.jpg	0.004386	0.004386	0.008772	0.98598	0	0.249346
data:IMG_5660.jpg	0.004386	0.004386	0.008772	0.98598	0	0.249346
data:IMG_5661.jpg	0.039474	0.004386	0.039474	0.524543	0	0.14539
data:IMG_5662.jpg		0.004386	0.008772	0.699852	0	0.177156
data:IMG_5663.jpg		0.004386	0.004386	0.703229	0.004566	0.177123
data:IMG_5664.jpg		0.004386	0.008772	0.596828	0.013699	0.1514
data:IMG_5665.jpg		0.004386		0.866413	0	0.217042
data:IMG_5666.jpg	0.004386	0.008772	0.004386	0.617143	0.004566	0.156698
data:IMG_5667.jpg	0.008772	0.004386	0.004386	0.98598	0	0.249127
data:IMG_5668.jpg	0.004386	0.004386	0.004386	0.816174	0	0.206017
data:IMG_5669.jpg	0.013158	0.004386	0.008772	0.98598	0.006849	0.250662
data:IMG_5670.jpg	0.017544	0.004386	0.008772	0.575172	0.004566	0.148617
data:IMG_5671.jpg		0.004386	0.004386	0.668258	0	0.16838
data:IMG_5672.jpg	0.100877	0.004386	0.105263	0.83411	0.013699	0.24515

data:IMG_5673.jpg	0.004386	0.004386	0.004386	0.721079	0	0.182244
data:IMG_5674.jpg	0.017544	0.004386	0.008772	0.516238	0.025114	0.133884
data:IMG_5675.jpg		0.004386		0.659042	0	0.165199
data:IMG_5676.jpg	0.004386	0.004386	0.008772	0.536271	0.006849	0.136919
data:IMG_5677.jpg	0.004386	0.004386		0.98598	0.011416	0.247591
data:IMG_5678.jpg	0.013158	0.004386	0.004386	0.810424	0.002283	0.205896
data:IMG_5679.jpg	0.013158	0.008772	0.026316	0.98598	0.009132	0.254609
data:IMG_5680.jpg		0.004386	0.017544	0.607689	0.002283	0.15587
data:IMG_5681.jpg	0.004386	0.004386	0.004386	0.906352	0	0.228562
data:IMG_5682.jpg		0.004386	0.013158	0.531442	0.011416	0.243538

Figure 92: Matching Results Off Angle Corvette -- 2nd run

In this case, the underside of the corvette was the closest match at 32% followed by the top side of the front loader at 28%. It appears that some of the matching routines had difficulties with the off-center capture and did not generate any useful data for matching. It was curious to see the Weka library using J48 evaluator match the off-angle capture to one of the front loader images with confidence matches to the corvette in the 10% range.

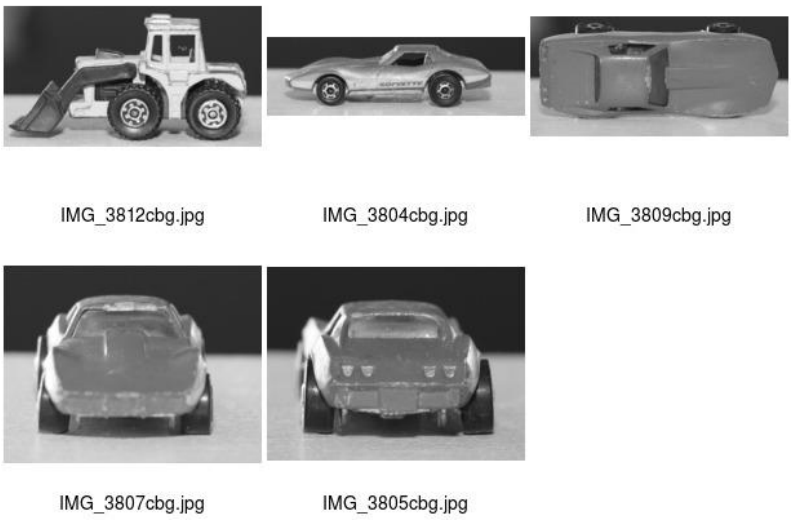


Figure 93: Initial Matches for Off-Angle Corvette Sample

The results with the front loader are presented in the next few figures and tables in the same order as was used with the corvette:

Figure 94: Front Loader Off-Angle De-noised

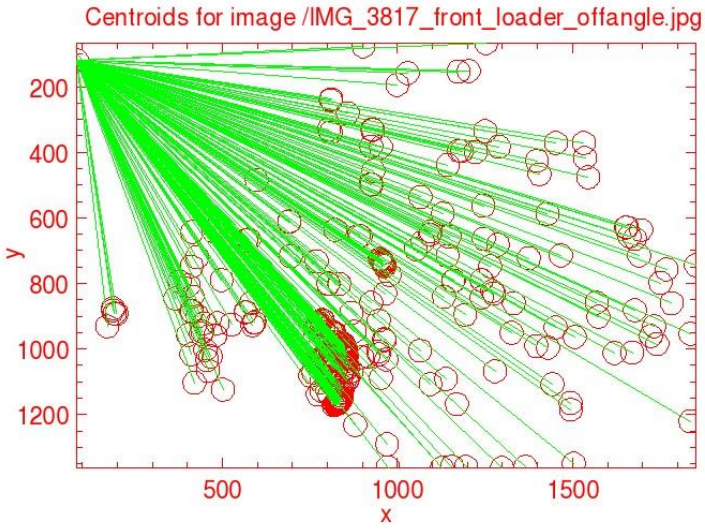


Figure 95: Front Loader Off-Angle Centroid Visualization

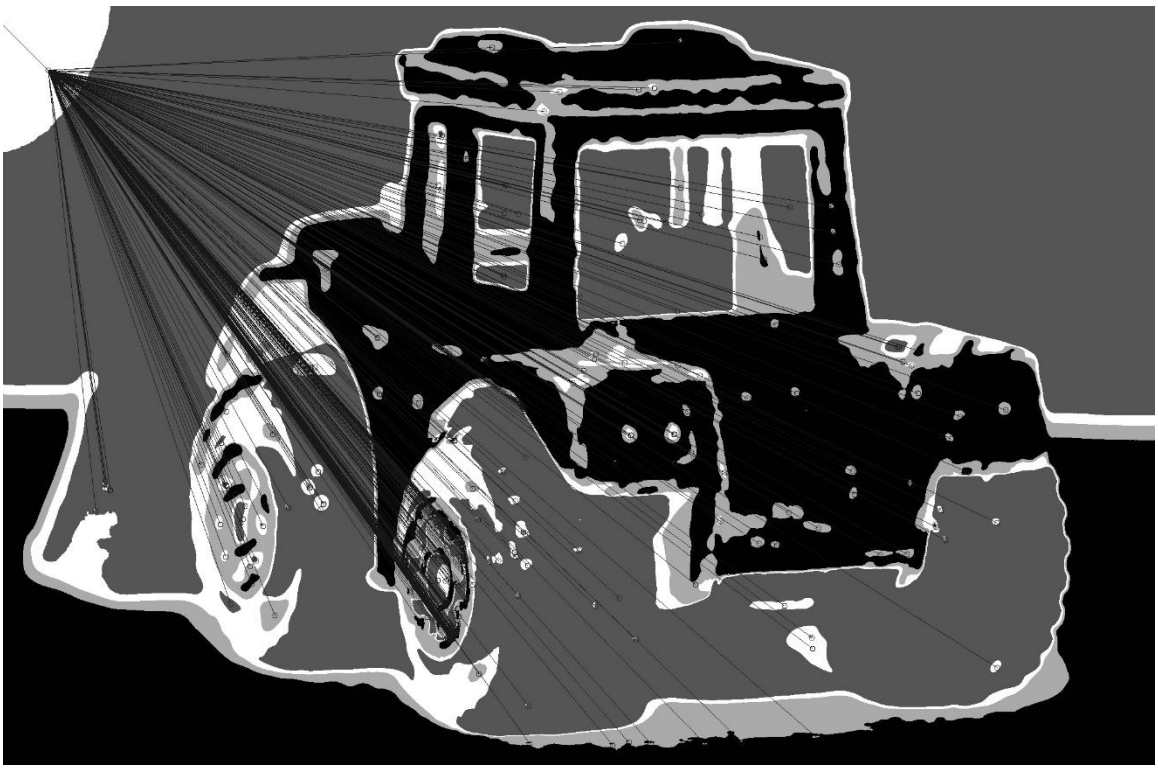


Figure 96: Front Loader Off-Angle Moments Over Clustered Data

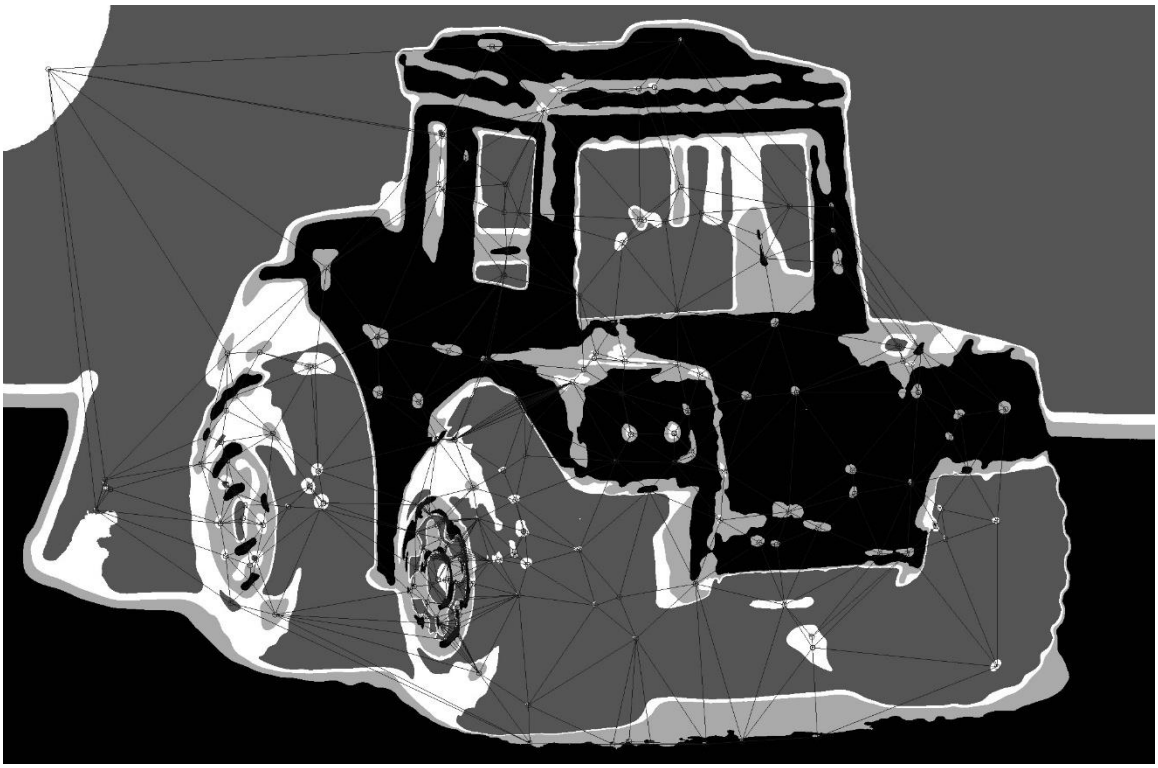


Figure 97: Front Loader Off-Angle Delaunay Triangulation

Table 20: Partial Matching Results Off Angle Front Loader

	$\alpha(S_i)$	$\beta(C_i)$	$\gamma(CS_i)$	$\varepsilon(LCS_i)$	$\zeta(SIM_G)$	$\eta(WDG)$	M_j
data:IMG_3804cbg.jpg	**0.102102**	**0.024024**	**0.003003**	**0.117117**	**0.89314**	**0.01702786**	**0.263525**
data:IMG_3805cbg.jpg	0.018018	0	0.003003	0.024024	0.927466	0.00773994	0.239674
data:IMG_3806cbg.jpg	**0.054054**	**0.012012**	**0.015015**	**0.084084**	**0.997424**	**0.05263158**	**0.276383**
data:IMG_3807cbg.jpg	0.027027	0	0.003003	0.015015	0.834639	0.0123839	0.216017
data:IMG_3808cbg.jpg	0.114114	0.03003	0.033033	0.054054	0.665675	0.04334365	0.199151
data:IMG_3809cbg.jpg	0.024024	0.018018	0.021021	0.039039	0.834639	0.01083591	0.223074
data:IMG_3810cbg.jpg	0.045045	0.006006	0.009009	0.057057	0.782949	0.1331269	0.215107
data:IMG_3811cbg.jpg	0.051051	0	0.003003	0.069069	0.948795	0.2879257	0.258971
data:IMG_3812cbg.jpg	**0.354354**	**0**	**0.003003**	**0.303303**	**0.850805**	**0.0758514**	**0.326815**
data:IMG_3813cbg.jpg	0.048048	0	0.003003	0.063063	0.78073	0.1424149	0.215303
data:IMG_3814cbg.jpg	0.102102	0.018018	0.021021	0.075075	0.707901	0.0758514	0.210309
data:IMG_3815cbg.jpg	0.024024	0.012012	0.015015	0.051051	0.607191	0.1393189	0.167714
data:IMG_5652.jpg	0.036036	0.018018	0.021021	0.048048	0.834639	0.00154799	0.223195

The results here were slightly more favorable most likely due to the sharp angles of the segmented regions resulting from the less sleek design of this utilitarian vehicle. The top overall match was the left side of the front loader, which is shown clearly in the off-angle image capture. However, several of the other top matches were in the corvette collection of model images followed by other views of the front loader.

Although the initial results were somewhat less than desired, they do give use a baseline from which to improve future result. So, in the figure 99, we see a sketch of how synthesis currently works and how it might work in a third version where we consider the border segments between the left and top or the right and top, or the left and rear, etc. That is, we merge the segments along the following borders:

Table 21: Synthesis Version 3 Proposed Merging of Segments

Right	Top
Right	Bottom
Right	Front
Right	Rear
Left	Top
Left	Bottom
Left	Front
Left	Rear

To simplify implementation, we may just merge the starting segment from one side with each of the segments from another images or each segment from one image with the segment on another image. An important question here is how far into each image do we merge segments? Does it make sense to have such a limit such as 1/5 of the side that intersects with another image or do we just merge all object regions across these spaces as shown in the following figure:

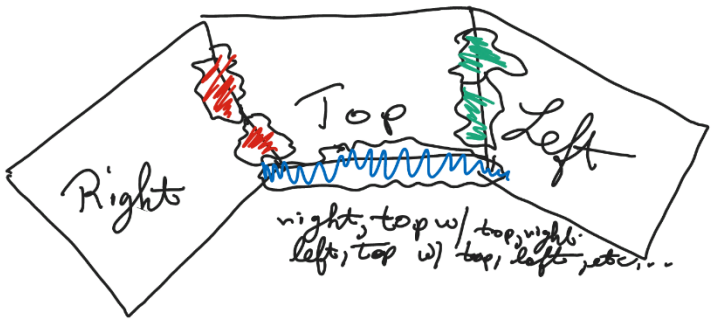


Figure 98: Determining How Far to Synthesize Across Views

Once again, the increased preprocessing computation of the model images is the price to be paid for potentially increased recognition of off-angle sample captures by an input device. The implementation to do this will be non-trivial and will require quite a considerable investment over just trying to generate synthesized macro segments from within a single view of the multi-view model.

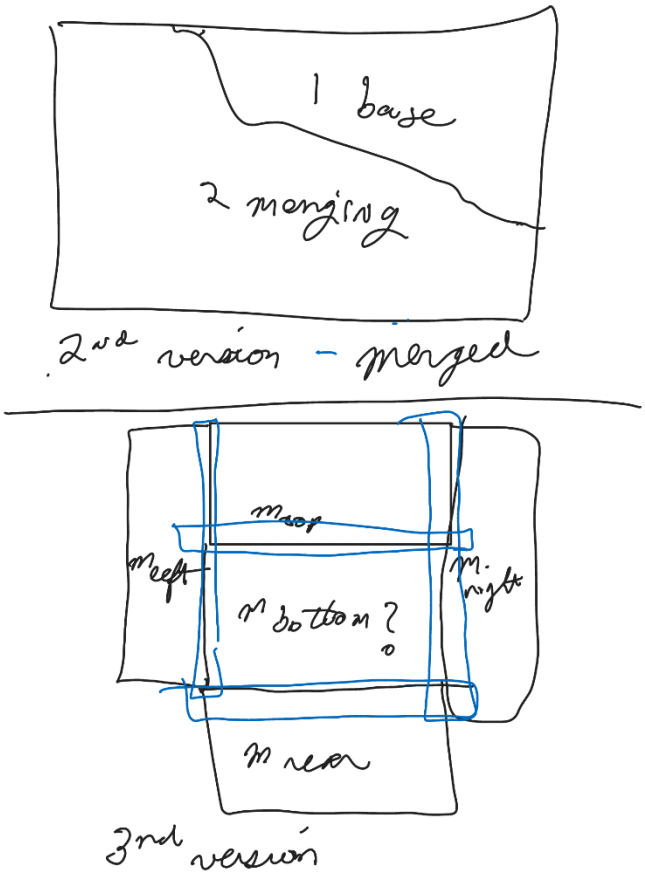


Figure 99: Simple Sketch of Moving forward with Synthesis

From an implementation standpoint, it will be necessary to have an input file specifying the images belonging to a particular model. A comma separated values (CSV) file is sufficient although creating an Extensible Markup File (XML) with schema would allow for a more robust handling of input. The CSV file would could have the following make-up:

Table 22: Proposed CSV Format for Synthesis Processing

ImageLeft, ImageRight, ImageFront, ImageRear, ImageTop, ImageBottom
IMG_3804cbg.jpg, IMG_3806cbg.jpg, IMG_3807cbg.jpg, IMG_3805cbg.jpg, IMG_3809cbg.jpg, IMG_3808cbg.jpg
IMG_3812cbg.jpg, IMG_3810cbg.jpg, IMG_3813cbg.jpg, IMG_3811cbg.jpg, IMG_3815cbg.jpg, IMG_3814cbg.jpg
…

Table 23: Proposed XML Format for Synthesis Processing

```
<?xml version="1.0"?>
<Model>
        <Views ImageLeft="IMG_3804cbg.jpg" ImageRight=" IMG_3806cbg.jpg"
        ImageFront=" IMG_3807cbg.jpg" ImageRear=" IMG_3805cbg.jpg" ImageTop="
        IMG_3809cbg.jpg" ImageBottom=" IMG_3808cbg.jpg"></Views>
        <Views ImageLeft="IMG_3812cbg.jpg" ImageRight=" IMG_3810cbg.jpg"
        ImageFront=" IMG_3813cbg.jpg" ImageRear=" IMG_3811cbg.jpg" ImageTop="
        IMG_3815cbg.jpg" ImageBottom=" IMG_3814cbg.jpg"></Views>
</Model>
```

TERTIARY METHODS

In this work, several ancillary methods were also used to enhance the quality of a match. One of those techniques was to use the matching of moments. It is a straightforward, but low fidelity technique that matches an image based on which model image has the highest number of moments in conjunction with the sample image. Obviously, this matching technique requires both the sample and input images to have captured under similar conditions for any segments to match. Therefore, it is given a modest weighting in the overall determination of which model image a sample image most likely matches. In addition, we could consider matching using the various properties of the line segments forming the border of each segment

to fall under these ancillary methods. This included counting the number of segments where there was correspondence on the starting point of the chain code and looking for the longest common substring string.

GEOMETRIC DATA AND MACHINE LEARNING

In one of the final approaches to establish a suite of techniques for matching incomplete images in a stochastically meaningful manner, consideration of whether using various geometric properties as the data source in a machine learning environment could improve recognition. For this category of match, the Weka collection of machine learning algorithms were used. It's a framework that permits easy, programmatic use of various Machine Learning methods with a moderate amount of setup effort, uses open-source machine learning algorithms that anyone can audit and modify, and permits the addition of new machine learning algorithms [17]. Given its pure Java implementation, the Weka library integrates particularly well with the source code body used in this work. It is capable of operating on large datasets like those derived from application of the L-G Graph Algorithm on images. For an initial test, this work reused the Delaunay graphs originally used in the graph similarity matching method described elsewhere in this document. The Delaunay graph was built using the centroids that made up the global graph output from the L-G Graph Algorithm. The centroids were fed into a routine to calculate a rectangle covering those points, then supplied to a constructor of an object whose instantiating class can perform planar subdivision on an array of two-dimensional points (SubDiv2D). This class of OpenCV is ready-made to work with Delaunay triangulations or Voroni diagrams. The set

of triangles can then be retrieved for connecting in displaying an image or for use in other applications. In this work, the set of triangles were supplied to the Weka framework for analysis.

It was natural to use a supervised approach in trying to perform the match given the exemplar set of labeled six-sided model images. The dataset attributes, therefore, consisted of the vertices of each Delaunay triangle (referred to as triads in the code) and the filename of the image itself as the final attribute. The triads are considered to be numeric data and the filename is considered to be a nominal attribute (e.g., it is an attribute that represents some fixed set of values, which in this case comes from the set of possible model filenames). Each triangle naturally covered a set of segments (e.g., components) from the original image which was generated using a K-Means algorithm. K-Means is a clustering algorithm and unsupervised learning method [17]. After forming the set of attributes to use in the training and matching phases, the training is then performed by working through the set of model images in the database, including all rotations. Each image filename is pulled from the database and use to query for its corresponding Delaunay triangulation graph that was calculated during the database building phase. An Instance object is formed from each triad and its corresponding filename (serving as the label) and then added to Instances object. Both the Instance and Instances objects are part of the Weka library for holding training and test data. The sample (e.g, obstructed image) is then processed in a similar manner except its Instance objects are added to a sample Instances object and not the training Instances object. Subsequently, a classifier has to be initialized and supplied with the training dataset to build the appropriate classifier. For this work, it was decided to use the J48 classifier to build the set of matching classes. It is a tree-

based classification algorithm derived from the ID3 algorithm and is an open-source implementation of C4.5 (Kaur & Chhabra, 2014). J48 allows for missing values, although this work doesn't use that feature of the algorithm, along with decision tree pruning and decision rule sourcing (Kaur & Chhabra, 2014). In general, a tree-based classification algorithm uses observations about something of interest coming from data making up the branches of the tree to reach conclusions about the values in the leaves (e.g., terminal nodes) of the tree. Furthermore, given J48 should only be used on to reach conclusions on a discrete set of values (e.g., a partial image is a match to one of the model database images), it is a classification tree based algorithm. In classification based systems, the leaves are the classes and the branches represent conjunction of values that led to a one class label over another being chosen. It would be interesting to see other tree-based, decision orientated classifiers would work on the supervised datasets.

Once the model has been built using the chosen classifier, an Evaluator object has to be created to evaluate the model against the partial images. The Evaluator also expects one or more AbstractOutput objects to be supplied to store the results. Given AbstractOutput is a generic type, something concrete like an InMemory (e.g., natural representation) object or PlainText (e.g., plain text) object has to be supplied. It is also possible to provide other Outcome objects such as CSV (Comma Separated Values), HTML (Hypertext Markup Language), XML (Extensible Markup Language), or Null. The InMemory format was used for processing results to put into the spreadsheet data and the plain text format was used for storing diagnostic data in a text file. The code for this matching method is too long to post in this document (the reader

should consult the codebase), but an activity diagram showing the steps is shown in the following figure:

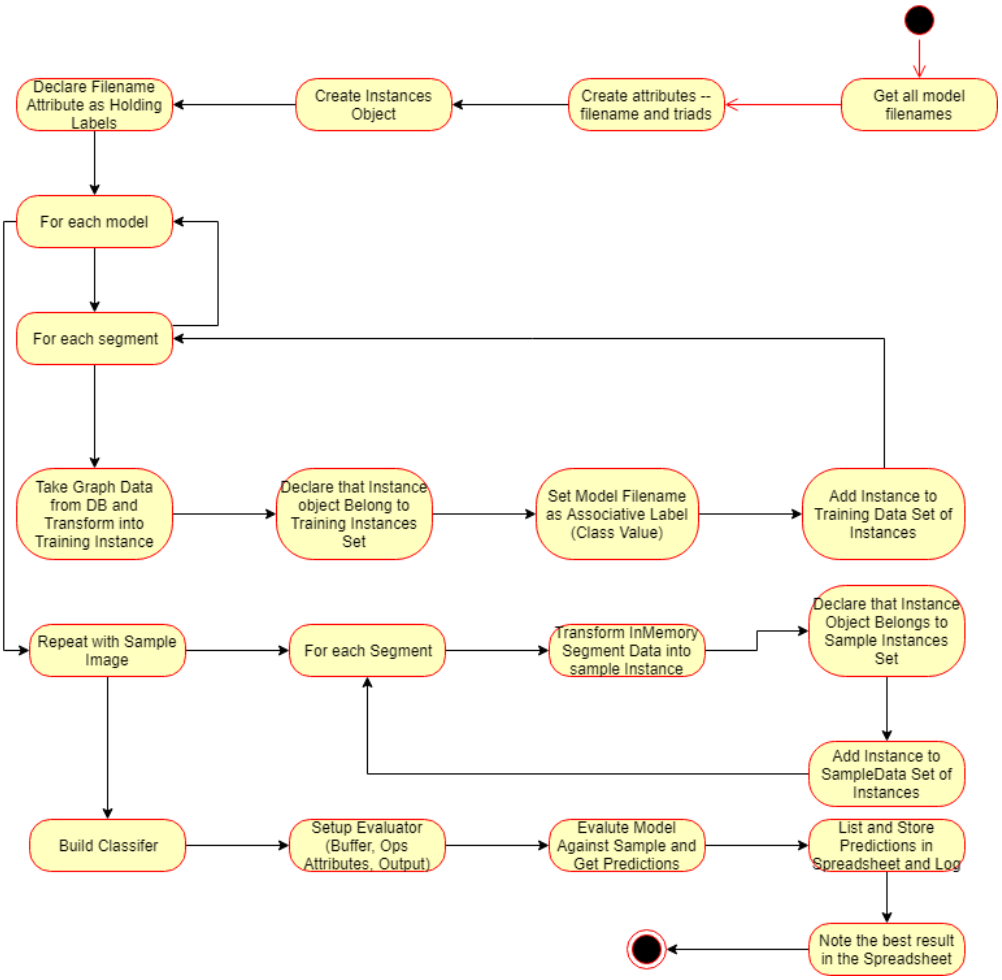


Figure 100: Match to Model by Delaunay Graph Activity Diagram (Using Weka ML Library)

Using the partial sample images from the chain code matching section, the following results were obtained:

Table 24: Matching using the J48 classifier

PARTIAL	BEST PREDICTED IMAGE	# Instances Matching	Probability of a Match (Against total # of prediction objects)	Missed Match # Instances Matching (IMG_5652)
PARTIAL2	IMG_5652	189	49.60%	
PARTIAL3	IMG_5652	351	49.71%	
PARTIAL4	IMG_5652	280	38.62%	
PARTIAL5	IMG_5652	410	53.45%	
PARTIAL6	IMG_5661_r135	25	6.67%	0 (though IMG_5652_r135 had four matching instances)
PARTIAL7	IMG_5652	23	16.31%	
PARTIAL8	IMG_5677	2	7.99%	1
PARTIAL9	IMG_5652	427		

The results show that matching works the best when slices or portions are missing from the sample image (e.g., part(s) of the image are obstructed or removed). PARTIAL6 is a rotation and this breaks down, although it does try to match using one of the rotated images and achieves a weak match for the wrong image. The correct rotated model image had a very weak match relative to IMG_5661, which is the rear of a classic black sedan toy car. PARTIAL7 and PARTIAL8 are blurred partial images and either weakly match the correct image or weakly match another image. Would it help to have blurred instances of all model images at a number of different levels? It's hard to stay without additional investigation. PARTIAL9 is a wavy transform or the original image and generates a strong match as the segmentation most likely deviates little from how the unmodified model image was processed. The K statistic, provided for

nominal class identification sessions by the Evaluator object, was 1.0, which corresponds to complete agreement with the labels [55]. In other words, we can have full confidence that this machine learning algorithm is doing much better than random change at making an identification given the sample data and existing set of model images even if some PARTIAL are misidentified or don't match at a particularly high level. If the K statistic had been less than or equal to zero, then we would be doing no better than random chance [55]. Given this is a nominal, rather than numerical matching process, there is no correlation coefficient, error rate, or root relative squared error to be returned from the Evaluator object.

If we change the evaluator to the LMT classifier, which uses linear regression based decision tree models, we obtain the following results:

Table 25: Matching using the LMT classifier

PARTIAL	BEST PREDICTED IMAGE	# Instances Matching	Probability of a Match (Against total # of prediction objects)	Missed Match # Instances Matching (IMG_5652)
PARTIAL2	IMG_5652_r315	182	47.77%	53
PARTIAL3	IMG_5652_r315	177	25.07%	138
PARTIAL4	IMG_5652_r315	203	27.25%	122
PARTIAL5	IMG_5652_r315	188	24.87%	153
PARTIAL6	IMG_2420_r315	62	16.58%	40
PARTIAL7	IMG_5652	25	17.99%	
PARTIAL8	IMG_2427_r315	5	20.00%	1
PARTIAL9	IMG_5652_r315	221	21.03%	181

It's not surprising that the results were not very good given the object identification tasks do not fit into the mold of a bipartite decision, but rather is a multimodal classification problem. Having said that, it does come remarkably close to identifying the correct image with

most of the partial sample images when presented. One other downside to the use of the classifier is the multi-hour time needed to process the model images prior to first matching.

FINDING A MOST PROBABLE, FINAL MATCH

In this work, a set of techniques have been employed to find a match. For each input sample image, the following calculation is made against each model image:

$$M_j = \alpha(S_i) + \beta(C_i) + \gamma(CS_i) + \varepsilon(LCS_i) + \zeta(SIM_G) + \eta(WDG)$$

Where M_j is the probabilistic measure between zero and one that some sample input image i matches a model image j given the weights alpha, beta, gamma, epsilon, zeta, and eta which are experimentally set to summed to one. Breaking this down by individual measure: S_i represents the use of chain-code matching by QGram from the application of the Local-Global graph algorithm, C_i represents the use of centroid matching, CS_i represents the correspondence of chain code starting locations on the Cartesian plane, LCS_i represents a match based on longest common subsequences of chain codes (e.g., on a total count of matching segments), and SIM_G represents a match based on the smallest delta between a model image and the sample image being processed using global graph similarity measures. WDG is the use of the global graph from the L-G graph algorithm transformed into a Delaunay graph whose triangles are then fed as a set of inputs into the Weka ML library for use with one of the supervised learning algorithms. In terms of weights: Alpha (α) is set to a 0.25 weight with beta (β) set to 0.15, gamma (γ) set to 0.05, epsilon (ε) set to 0.10 weight, zeta(ζ) set to 0.20 weight, and eta (η) set to 0.25. The matching model image, therefore, is the one where:

$$Max(M_1, M_2, \ldots, M_j)$$

In one experimental run using the PARTIAL2 sample image of a toy dump truck where the left half portion is completely obstruction, the following results were obtained:

Table 26: Finding a Most Probably, Final Match Results (Black Text is for IMG_5652 Comparison, Red Text for Best Match)

Sample	Best Overall Match	$\alpha(S_i)$	$\beta(C_i)$	$\gamma(CS_i)$	$\varepsilon(LCS_i)$	$\zeta(SIM_G)$	$\eta(WDG)$	M_j
PARTIAL2	IMG_5652	10.34%	42.36%	36.94%	10.83%	65.75%	49.61%	25.97%
PARTIAL3	IMG_5652	10.19%	46.28%	36.91%	9.09%	99.16%	49.71%	34.14%
PARTIAL4	IMG_5674_r315	6.366% (40.05%)	41.11% (DNF)	39.52% (DNF)	6.366% (34.78%)	69.69% (80.27%)	38.62% (0.275%)	25.65% (33.02%)
PARTIAL5	IMG_5652	10.83%	55.66%	52.64%	10.58%	91.52%	53.45%	34.67%
PARTIAL6	IMG_5654	1.55% (3.608%)	0.00% (0.00%)	1.03% (0.05%)	2.577% (2.577%)	92.34% (98.92%)	2.67% (80.00%)	23.94% (25.83%)
PARTIAL7	IMG_5652	13.16%	2.63%	2.63%	14.47%	97.71%	16.31%	29.69%
PARTIAL8	IMG_5652	0.00%	0.00%	5.88%	5.88%	97.78%	4.00%	26.21%
PARTIAL9	IMG_5674_r315	3.07% (50.09%)	6.69% (0.03%)	6.15% (0.00%)	3.44% (46.29%)	91.36% (50.89%)	39.21% (27.55%)	24.94% (29.51%)

Within a given category, model image IMG_5652 may not have been the best match. However, given the weights used and results from individual matching routines, model IMG_5652 was the best match in a number of cases. In general, the chain-code matching methods performed moderately well, but take significant resources in memory and time to complete. The machine learning processing of the Delaunay graphs take far fewer resources, complete much more quickly than the chain code matching methods to yield similar results. The use of the SIM_G matching method is relatively quick, requires moderate levels of resources, and yields reasonable good levels of confidence for all model images. It took almost fifty hours to match all partial images against the set of model images or just slightly more than six hours per

image on a Xeon E5-2560 based workstation running six hyper-thread cores at 2.0 GHz each (twelve hardware threads total).

V. FINDINGS AND CONCLUSIONS

PERFORMANCE METRICS

The model building phase is largely a serial process, at least at the application level of the research code. VisualVM shows the typical CPU usage:

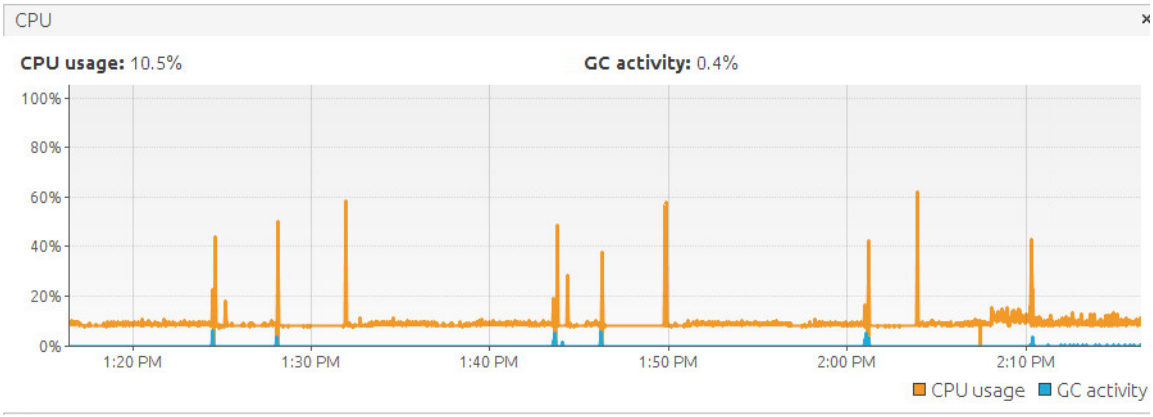


Figure 101: Typical CPU usage during model image building phase

Most of the threads are started either by the database engine, other third party APIs, or the core Java classes. The processes taking the most time include:

- Generate binary image segments from clustered binary image
- Building the chain code for a segment
- Getting the raw centroid value

- Converting OpenCV Matrices to double arrays
- Building a new Local-Global Node object
- Generating PLplot images

This is illustrated in the following figure captured from VisualVM profiling dashboard:

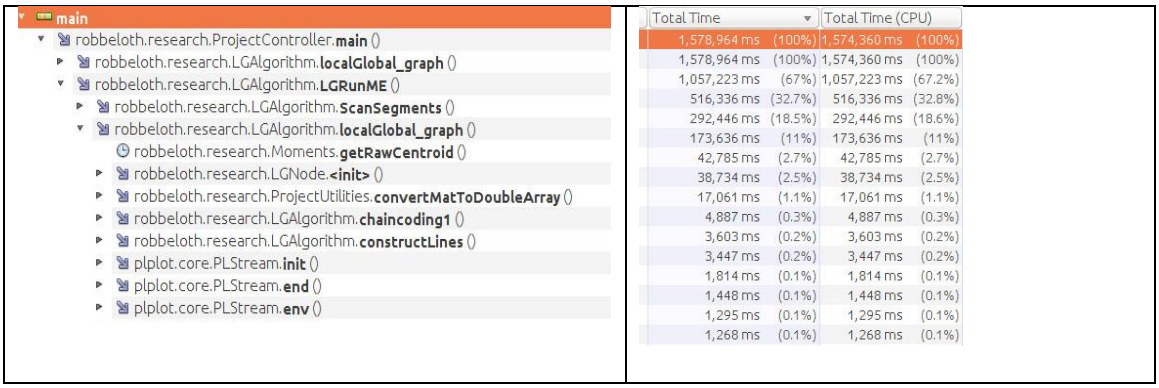


Figure 102: Long running activities in model image building phases

Profiling of code during the matching phase was done to identify slow-performing code for possible performance improvements. Profiling of the Java code was done using Oracle's VisualVM tool [56]. Snapshots of resource utilization were captured during the segmentation and matching phases which occur both when matching obstructed images and building the full six-sided model image database. During the segmentation phase, the following CPU intensive areas were identified in the following figure:

Name	Self Time	Self Time (CPU)	Total Time	Total Time (CPU)	Invocation
robbeloth.research.ProjectUtilities.doIntersect (org.opencv.core	0.070 ms (37.6%)	0.071 ms (38%)	0.129 ms (21.8%)	0.129 ms (21.9%)	18720448
robbeloth.research.ProjectUtilities.orientation (org.opencv.core	0.044 ms (23.7%)	0.044 ms (23.5%)	0.044 ms (7.4%)	0.044 ms (7.5%)	75701800
robbeloth.research.ProjectUtilities.isInside (org.opencv.core.Ma		0.033 ms (17.6%)	0.188 ms (31.8%)	0.187 ms (31.8%)	25112
org.opencv.core.Mat.get (int, int)	0.024 ms (12.9%)	0.024 ms (12.8%)	0.024 ms (4.1%)	0.024 ms (4.1%)	38435891
robbeloth.research.ProjectUtilities.onSegment (org.opencv.core	0.013 ms (7%)	0.013 ms (7%)	0.013 ms (2.2%)	0.013 ms (2.2%)	23501691
robbeloth.research.LGAlgorithm.determine_line_connectivity (ja	0.001 ms (0.5%)	0.001 ms (0.5%)	0.002 ms (0.3%)	0.002 ms (0.3%)	1
org.opencv.core.Point.equals (Object)	0.001 ms (0.5%)	0.001 ms (0.5%)	0.001 ms (0.2%)	0.001 ms (0.2%)	2088024

Figure 103: CPU profiling of segmentation phase hotspots

Based on total CPU time and invocations, those methods determining various line properties and region size (in pixels) were the worst performing methods. All the data in this part of the code is in memory, so it is not I/O bound. A possible refinement for this code is to either improve the overall efficiency of the algorithms here and/or find ways to parallelize the code, which has not been done. In the matching phase, there is a more complicated analysis due to the presence of multithread code in both traditional and resource pool form. Overall, though, the following CPU intensive areas were identified in the following figure:

Name	Self Time	Self Time (CPU)	Total Time	Total Time (CPU)	Invocation
robbeloth.research.LGAlgorithm.lambda$1 (String, java.util.concurrent.a	64.0 ms (49%)	0.184 ms (0.7%)	104 ms (16%)	25.7 ms (7.6%)	17827646
robbeloth.research.DatabaseModule.getChainCode (int)	0.223 ms (0.2%)	0.136 ms (0.5%)	23.7 ms (3.6%)	18.6 ms (5.5%)	22846306
org.hsqldb.Session.execute (org.hsqldb.result.Result)	0.192 ms (0.1%)	0.149 ms (0.5%)	22.3 ms (3.4%)	18.0 ms (5.3%)	68668237
org.hsqldb.StatementDMQL.execute (org.hsqldb.Session)	0.078 ms (0.1%)	0.060 ms (0.2%)	13.7 ms (2.1%)	11.6 ms (3.5%)	45821346
org.hsqldb.StatementQuery.getResult (org.hsqldb.Session)	0.074 ms (0.1%)	0.059 ms (0.2%)	13.4 ms (2.1%)	11.4 ms (3.4%)	45821346
org.hsqldb.QuerySpecification.getResult (org.hsqldb.Session, int)	0.068 ms (0.1%)	0.057 ms (0.2%)	13.3 ms (2%)	11.3 ms (3.4%)	45821346
org.hsqldb.QuerySpecification.getSingleResult (org.hsqldb.Session, int)	0.121 ms (0.1%)	0.100 ms (0.4%)	13.1 ms (2%)	11.1 ms (3.3%)	45821346
info.debatty.java.stringsimilarity.LongestCommonSubsequence.distance	1.15 ms (0.9%)	0.010 ms (0%)	23.4 ms (3.6%)	10.9 ms (3.2%)	17827642
info.debatty.java.stringsimilarity.LongestCommonSubsequence.length (S	22.2 ms (17%)	10.9 ms (39.5%)	22.2 ms (3.4%)	10.9 ms (3.2%)	15231665
org.hsqldb.QuerySpecification.buildResult (org.hsqldb.Session, int[])	0.456 ms (0.3%)	0.364 ms (1.3%)	12.7 ms (1.9%)	10.8 ms (3.2%)	45821346
org.hsqldb.jdbc.JDBCResultSet.getString (String)	0.038 ms (0%)	0.031 ms (0.1%)	13.3 ms (2%)	10.4 ms (3.1%)	22846780
org.hsqldb.jdbc.JDBCResultSet.getString (int)	0.067 ms (0.1%)	0.058 ms (0.2%)	12.8 ms (2%)	10.0 ms (3%)	22847433
org.hsqldb.types.ClobDataID.getSubString (org.hsqldb.SessionInterface,	0.030 ms (0%)	0.019 ms (0.1%)	12.2 ms (1.9%)	9.62 ms (2.9%)	22846306
org.hsqldb.types.ClobDataID.getChars (org.hsqldb.SessionInterface, I	**0.052 ms (0%)**	**0.043 ms (0.2%)**	**12.1 ms (1.9%)**	**9.59 ms (2.8%)**	**22846306**

Figure 104: CPU Profiling Hotspot During Matching Phase (LCS is one)

The increased demand on CPU and memory resources can be clearly seen in the following overview diagram at about the fifteen minute mark:

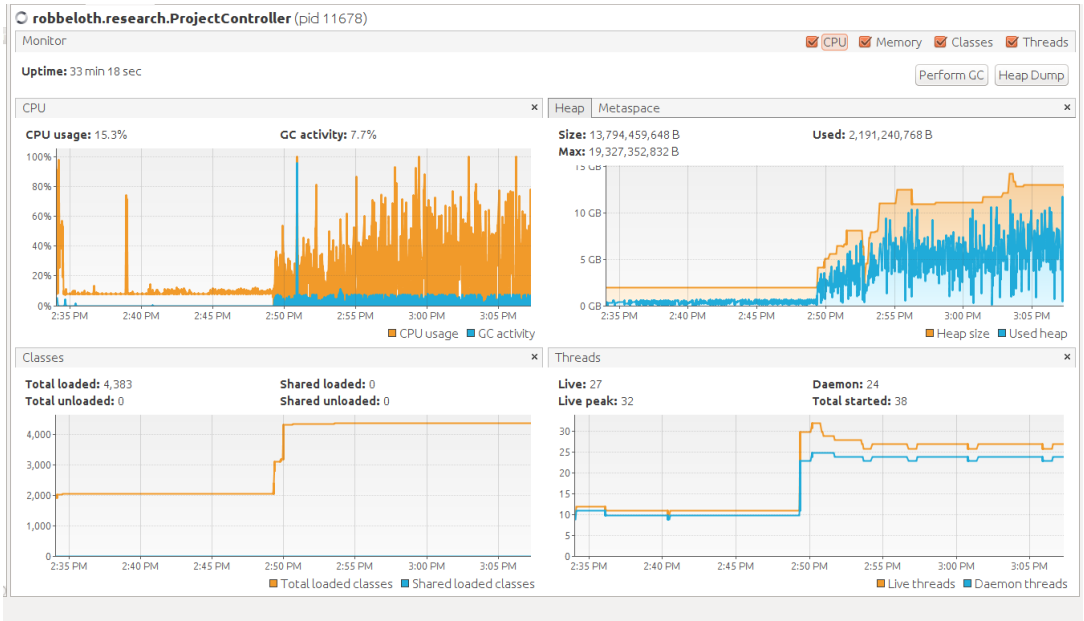


Figure 105: Transition from Processing Sample to Matching Against Model Images

The CPU spent the most time going out to traditional spinning hard disk drives to retrieve chain codes from the database. I have found little to no improvement in the caching of character blob fields (e.g., the chain codes) from the database into memory and the caching of additional rows from the obstruction table. I did not find it helpful to move the database to faster I/O storage such as a Solid-State Hard Disk Drive (SSD) or M.2 enclosure on the Serial ATA (SATA) or Peripheral Component Interconnect (PCI) buses. Therefore, it appears that most of the code is CPU rather than I/O bound.

It was discovered in some experimental runs that that parallelization of all database lookup code within each chain code based matching routine saved approximately 11%, a

relatively modest improvement. The parallelization was done using the Java Collections' `IntStream` parallelization.

Beyond this, a great deal of difficulty was encountered during experimental studies in preventing Java heap space exhaustion at various times, particularly when having OpenCV build Delaunay derived global graph due to the highly complex set of triangles generated by the processed and the resulting memory requirements. However, I have resolved the worse of those issues. In general, the research code is running with a specification of an initial 2GB heap space, a maximum allowed heap space of 29GB, and use of the serial memory management engine. Most of the time, during Model processing to build the image database, memory usage would vary between 500 MB and 1.5 GB and not seen expansion of the heap space above 2 GB. I experimented with most of the Java runtime memory management engines before simply settling on the standard Serial engine, which allows for the return of heap space to the operating system. In part, the extensive use of native memory allocations by OpenCV and long-term references by various sections of code to Plain Old Java Objects (POJOs) does not allow the Java memory management engine to quickly recover heap space. In addition, the use of the Serial engine is not as CPU efficient as some of the parallel and concurrent memory management engines, though not all those permit return of heap space to the operating systems and lowering of overall memory used by the software. At the time of this work, attempts continue to find a better set of Java runtime parameters to use with an emphasis on optimal memory management settings.

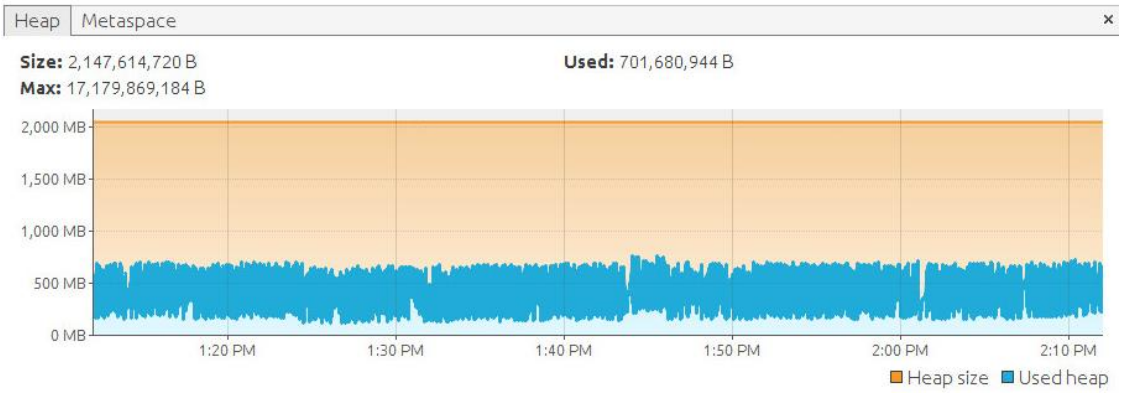


Figure 106: Memory Spikes During Full Database Build of Model Image Processing due to Delaunay Construction

In fact, it was not an uncommon occurrence as the model database kept increasing in size to encounter Java `OutofMemory` runtime errors from the heap space being exhausted. It was often necessary to increase the max heap space parameter to deal with the extended periods of memory being held by native code allocations that were difficult for the virtual memory manager to collect as shown in the following figure:

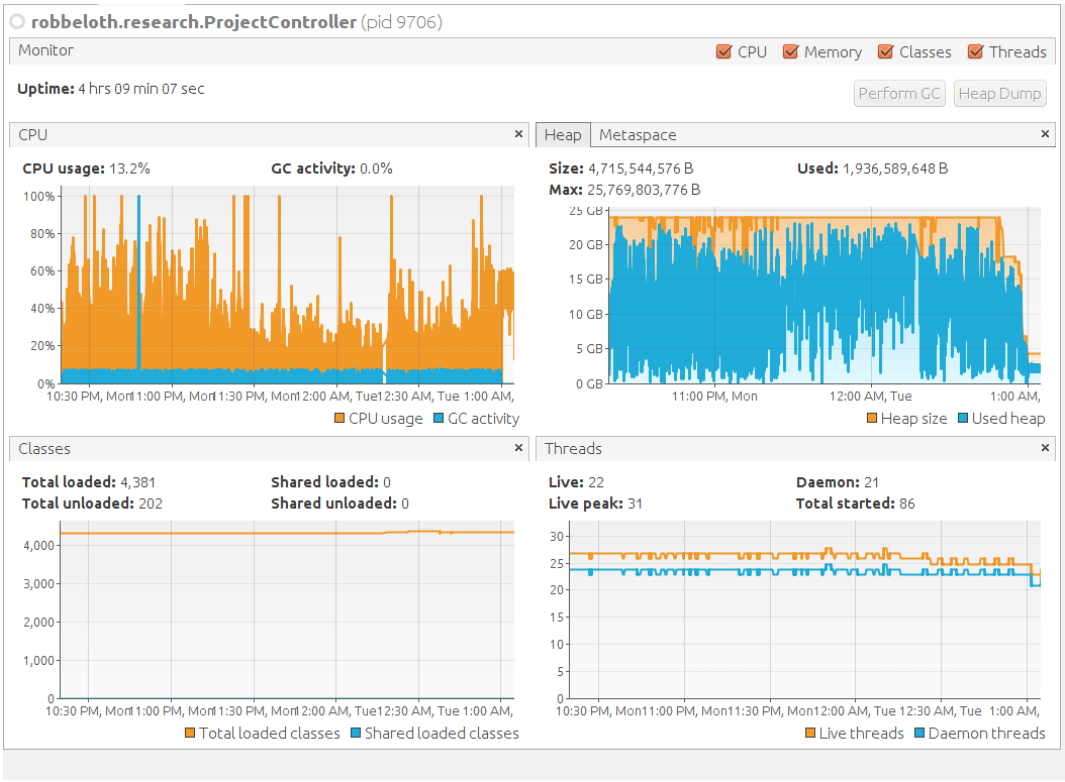


Figure 107: Heap Space Exhaustion During Experimental Run at 1:00 am Mark

When a matching routine returns successfully and the experimental run is close to complete, we see the following changes in memory, but not CPU utilization. It would be helpful to spend more time studying the heap dumps in greater detail and correspond those times with thread traces to see where the most troublesome allocations are occurring.

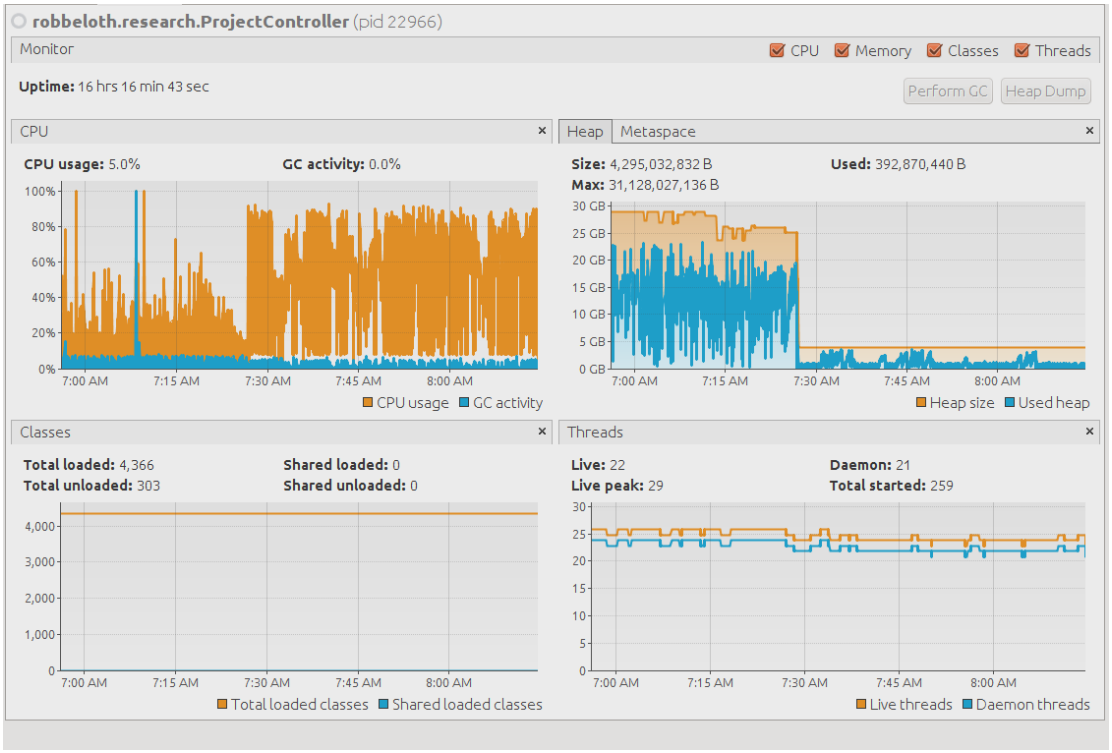


Figure 108: Completion of a Matching Routine and Release and Reclamation of Memory Resources by GC

One of the heap dumps that was obtained at the conclusion of this work showed that only a small part of the total memory allocations was tied up in Java native memory structures. It would be interesting to carry this line of investigation further to see where the additoinal memory allocations are tied up. I suspect they are a part of the native memory allocations, but time constrains prevented a more indepth investigation.

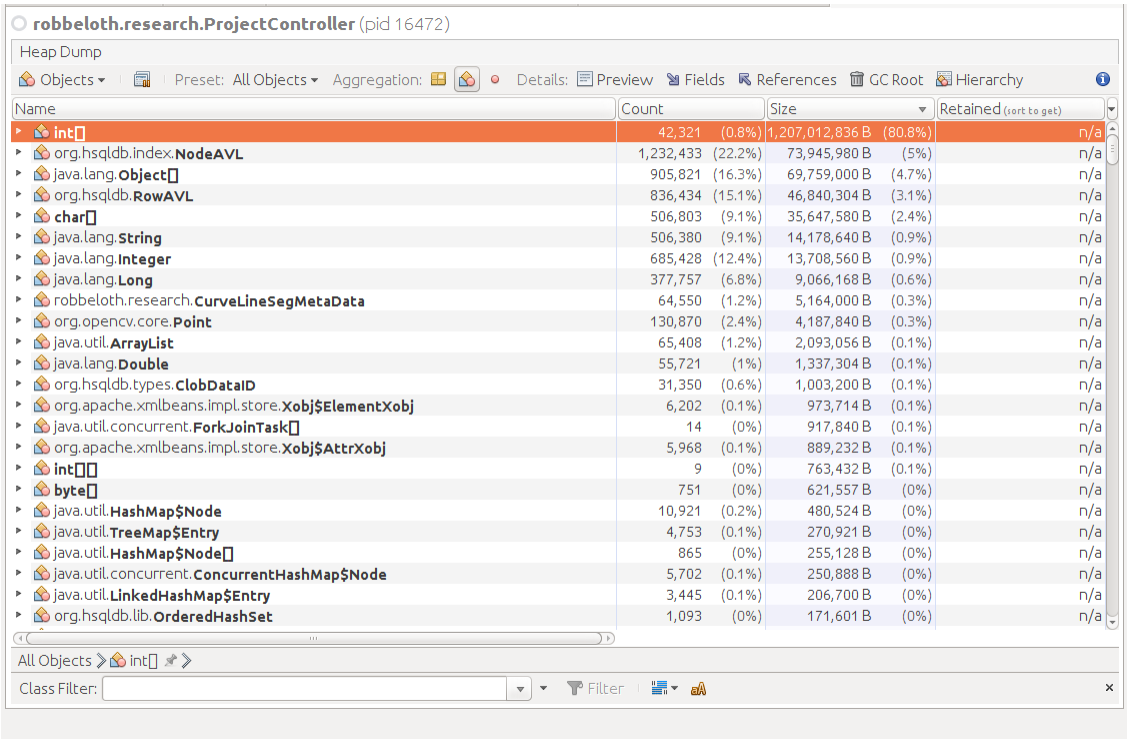


Figure 109: Heap Dump During Matching Phase

It is not surprising given the image processing work being done that a good deal of the allocations are tied up in integer array allocations. A large number of allocations are also tied up in chain codes and where need the use of `ArrayList`to handle primitive data in object form. The HyperSQL database calls also utilize a fairly good chunk of memory as well.

OVERALL CONCLUSIONS

Although the results for finding matches with individual images and obstructions are documentated elsewhere in this work, it can be concluded that the suite of methods for finding a match utilizing geometric characteristics is a promising area of research that deserves further investigation even if the matches are weakly or moderately correlated in this work. In particular,

one of the first modifications to be made would be to replace the use of LCS in the suite of matching methodologies with Levenshetin or other related method. Originally, it was chosen, despite its low scoring in chain coding because it was hypothesize that some images might have long unobstructed border regions that might work well with this algorithm. This turned out to not be the case. Overall, though, the chain code utilziation in the L-G graph algorithm of segment border region seems to provide the best level of performance when choosing the correct string matching algorithm or set of algoirthms. Furthermore, the use of the Delaunay triangulation in the Weka machine learning library seems promising with future work that considers the input of different attributes beyond the Delaunay triangulation, tries to work with different evaluators, and/or tries to vary operational parameters such as the weights for nodes for the ML algorithms being used. Incomplete object recongition, especially when dealing with potentially off-angle targets, is going to take considerable effort, novel algorithm development, and significant computing resources before more meaningful, quantative improvements in recongition will be achieved. However, this work demostrates how to move forward with such efforts and provides a framework by which some efforts can be undertaken in an environment that does not have depencies on commerical codebases or propertietary, closed-off algorithms. The code in this work is fully open-source with a permissive license for allowing new research efforts by this author or others into the forseeable future.

CONTRIBUTIONS OF THIS WORK

The major contributions of this work include the following items:

1. A novel methodology for recognition of objects based on their incomplete representation in a form of Local-Global Graphs and Delaunay triangulation.
2. Established an open-source Apache licensed framework for future investigations into incomplete object recognition using all open-source, non-commercial codebases.
3. Demonstrated a framework of methods for detection of incomplete objects using a variety of geometric and hybrid geometric/machine learning methods via the Weka library.
4. Demonstrated the use of chain codes as a string matching problem amenable for use with string matching algorithms.
5. Adopted the L-G Graph Algorithm for use with incomplete object recognition.
6. Demonstrated how to break down multi-view model images using K-Means segmentation and L-G Graph algorithm into an efficient set of attributes for storage and retrieval into and from a database.
7. Performed and published an extensive survey in the area of incomplete object recognition.
8. Highlighted the issues that make it difficult to achieve efficient, robust, and reliable incomplete object recognition

VI. FUTURE WORK

One of the areas of future research will be to improve the parallelization of the code base to determine if it is possible to make the code more production worthy than it is in its current form for use in future commercial or other research applications with myself and/or in collaboration with others. In particular, the only parts of the codebase that are parallelized right now are the matching routines that extract candidate model segments from the database. The processing of model images to build the initial database is still sequentially driven and would require moderate levels of refactoring of the code to implement. It may prove helpful to separate the model processing and image matching codebases into two separate projects to improve overall maintainability for future work. Ultimately, it would be interesting to see others take the progress made in this work would improve it to a point that it can be used in post-processing of reconnaissance or security images in the military or law enforcement sectors. Developing a robust vision system, particularly one that works with obstructed systems, has turned out to be a particularly difficult task and the resulting system is more brittle than what would be acceptable in a production system at present.

A second area of optimization work to coincide with the first would be to experiment with model and sample image run-time and accuracy results from down-sampling the OpenCV data structure sizes and input data/intermediate data sizes to reduce CPU load and memory

footprint. In addition, matching parameters will need to be fined tune to improve the overall accuracy and performance tradeoffs.

A third area of optimization work would be to take advantage of the GPU capabilities built into the OpenCV framework with an appropriate graphics video card. To go along with this effort, work will be done to secure grants to allow either the purchase of more robust workstations or servers with GPU capabilities or cloud computing resources from reputable vendors such as Amazon, Google, or Microsoft. In addition, it would be interesting to explore the use of various cloud services for hosting the database, separate server instances for the processing of new model images and the processing of obstructed sample images into various server components, or even being able to use serverless, event-driven computation instances such as Amazon Lambda, Google Cloud Functions, or Microsoft Azure Functions in parallel for processing an obstructed image against each of the matching techniques discussed in chapter four. The difficulty here would be needing to see all the data for analysis to an instances, which might not be a significant barrier if the instance was down at a fine-grained enough computational level or it would need to tie into an online database instance of the multi-view model database.

Fourth, it would be interesting to see if recognition rates could be improved by expanding the scope of synthesis processing, utilizing the various borderline and segment properties in additional ways. It is suspected that recognition rates would improve somewhat, but at the cost of extensive additional preprocessing of model images. However, without making an attempt in this area, it is unknown what that level of improvement could be yielded from such as effort.

Alternatively, expanded integration of using various geometric properties in new matching or extended matching routines that utilize the Weka Machine Learning framework beyond what has already been done in this current work may show some additional improvements.

Fifth, in the defense proposal it was noted that work on low-resolution objects could prove to be an interesting extension. Low-resolution objects take less time to acquire and process, but suffer from a loss of detail that is difficult to overcome without considering novel methods to work with information remaining in the image. One possibility is to consider taking the sample images at a low resolution, but with a variety of exposure points and combine to a high dynamic range (HDR) image to see if critical region details can be extracted.

Sixth, I would like to take greater advantage of the shape expression properties that have been captured by the L-G Algorithm and use them in additional matching routines and/or as additional attributes for the Weka ML algorithms and see how the matching probabilities change.

Seventh, I would like to utilize or repurpose a testing framework like Junit or utilize some other scripting framework to test sets of different preprocessing operators to find a set lead to higher recognition rates. It would be necessary to have the database and set of detection methods fixed to run this suite of tests. It might be interesting to vary the parameters to those operators to measure the influence on detection rates. It would take considerable time and resources to run such a narrow study.

Finally, it would be nice to expand the nature and form of obstructed sample images.

REFERENCES

[1] Abolbashari, M. (2012). High dynamic range compressive imaging: A programmable imaging system. Optical Engineering, 51(7), 071407. doi:10.1117/1.oe.51.7.071407

[2] Andriluka, M., Roth, S., & Schiele, B. (2008). People-tracking-by-detection and people-detection-by-tracking. 2008 IEEE Conf. on Computer Vision and Pattern Recognition. doi:10.1109/cvpr.2008.458758

[3] Arthur, D., & Vassilvitskii, S. (2007). K-Means++: The Advantages of Careful Seeding. Conference: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, 2007, . doi:10.1145/1283383.128349

[4] Aurenhammer, F. (1991). Voronoi diagrams---a survey of a fundamental geometric data structure. ACM Computing Surveys, 23(3), 345–405. doi:10.1145/116873.116880

[5] Barghout, L., & Sheynin, J. (2013). Real-world scene perception and perceptual organization: Lessons from computer vision. Journal of Vision, 13(9), 709–709. doi:10.1167/13.9.709

[6] Barinova, O., Lempitsky, V., & Kohli, P. (2012). On detection of multiple object instances using Hough transform. Computer Vision and Pattern Recognition Conference (CVPR)

[7] Barnes, C., Shechtman, E., Finkelstein, A., & Goldman, D. B. (2009). PatchMatch. ACM Transactions on Graphics, 28(3), 1. http://doi.org/10.1145/1531326.1531330

[8] Bhattacharjee, S., Das, S., & Dutta, A. (2008). Occluded shape (2-D) recognition using edge based features. TENCON 2008 - 2008 IEEE Region 10 Conference. doi:10.1109/tencon.2008.4766572

[9] Brown, J. M., & Koch, C. (1993). Influences of closure, occlusion, and size on the perception of fragmented pictures. Perception & Psychophysics, 53(4), 436–442. http://doi.org/10.3758/BF03206787

[10] Carasso, A., Bright, D., & Vlada, A. (2002). APEX method and real-time blind deconvolution of scanning electron microscope imagery. Optical Engineering, 41(10), 2499. doi:10.1117/1.1499970

[11] CHAN, C.-J., & CHEN, S.-Y. (2002). RECOGNIZING PARTIALLY OCCLUDED OBJECTS USING MARKOV MODEL. International Journal of Pattern Recognition and Artificial Intelligence, 16(02), 161–191. doi:10.1142/s0218001402001642

[12] Chung, J. C. H., Litt, M., & Leininger, G. (1990). A computer vision system for automated corn seed purity analysis. Proceedings of the Third International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems - IEA/AIE '90. http://doi.org/10.1145/98784.98822

[13] Chiu, C.-C., Ku, M.-Y., & Liang, L.-W. (2010). A robust object segmentation system using a probability-based background extraction algorithm. IEEE Transactions on Circuits and Systems for Video Technology, 20(4), 518–528. doi:10.1109/tcsvt.2009.2035843

[14] CollabNet, Inc. (2001). Argouml.Tigris.Org. Retrieved November 24, 2016, from ArgoUML, http://argouml.tigris.org/

[15] Debatty, T. (2014). java-string-similarity. Retrieved from https://github.com/tdebatty/java-string-similarity

[16]DINESH, R., & GURU, D. S. (2010). CONCEPT OF TRIANGULAR SPATIAL RELATIONSHIP AND B-TREE FOR PARTIALLY OCCLUDED OBJECT RECOGNITION: AN EFFICIENT AND ROBUST APPROACH. International Journal of Image and Graphics, 10(03), 423–448. doi:10.1142/s0219467810003846

[17]Eibe Frank, Mark A. Hall, and Ian H. Witten (2016). The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", Morgan Kaufmann, Fourth Edition, 2016.

[18]Ess, A., Leibe, B., & Van Gool, L. (2007). Depth and appearance for mobile scene analysis. 2007 IEEE 11th International Conference on Computer Vision. doi:10.1109/iccv.2007.4409092

[19]Foundation, E. (2016, November 24). The eclipse foundation open source community website. Retrieved November 24, 2016, from https://eclipse.org/

[20]Alder, G., & Benson, D. (2019). Draw.io. Northamptonshire, UK: JGraph Ltd. Retrieved from https://www.draw.io/

[21]García, G. B. (2015). Learning Image Processing with OpenCV. Community experience distilled.

[22]Greene, E. (2007). Recognition of objects displayed with incomplete sets of discrete boundary dots 1,2. Perceptual and Motor Skills, 104(3c), 1043–1059. http://doi.org/10.2466/pms.104.4.1043-1059

[23]Guo, G., Jiang, T., Wang, Y., & Gao, W. (2012). Recovering missing contours for occluded object detection. IEEE Signal Processing Letters, 19(8), 463–466. doi:10.1109/lsp.2012.2203592

[24]HSQL Development Group. (2001). HSQLDB. Retrieved November 24, 2016, from HSQLDB - 100% Java Database, http://hsqldb.org/

[25]Hu, M.-K. (1962). Visual pattern recognition by moment invariants. IEEE Transactions on Information Theory, 8(2), 179–187. doi:10.1109/tit.1962.1057692

[26]Huttenlocher, D. (2004). Computer Vision. In A. Tucker (Ed.), Computer Science Handbook (Second, pp. 43-1 to 43-23).

[27]Irwin, A., Ross, A., Furnish, G., & Babock, H. (2019). PLplot. Retrieved from http://plplot.sourceforge.net/

[28]Ferrari, V., Tuytelaars, T., & Gool, L. V. (2006). Object detection by contour segment network. In A. Leonardis, H. Bischof, & A. Pinz (Eds.), Lecture Notes in Computer Science (European Conference Computer Vision 2006 ed.) (pp. 14–28). doi:10.1007/11744078_2

[29]Felzenszwalb, P. F., Girshick, R. B., McAllester, D., & Ramanan, D. (2010). Object detection with Discriminatively trained part-based models. IEEE Transactions on Pattern Analysis and Machine Intelligence, 32(9), 1627–1645. doi:10.1109/tpami.2009.167

[30]Hao, C., & Wu, E. (2012). Interactive image completion with perspective constraint. Proceedings of the 11th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry - VRCAI '12. http://doi.org/10.1145/2407516.2407557

[31]Hays, J., & Efros, A. A. (2007). Scene completion using millions of photographs. ACM Transactions on Graphics, 26(3), 4. http://doi.org/10.1145/1276377.1276382

[32]Heidari, V., & Ahmadzadeh, M. R. (2013). A method for vehicle classification and resolving vehicle occlusion in traffic images. 2013 First Iranian Conference on Pattern Recognition and Image Analysis (PRIA). doi:10.1109/pria.2013.6528435

[33]Heidari, L., & Peng, F. (2013). Object recognition with incomplete features based on evidence accumulation. MIPPR 2013: Automatic Target Recognition and Navigation. http://doi.org/10.1117/12.2031497

[34]Hong, J., Xueyan, L., Shuxu, G., & Hua, F. (2010). Classification of the Incomplete Fingerprint Image. 2010 International Conference on Computer, Mechatronics, Control and Electronic Engineering. http://doi.org/10.1109/cmce.2010.5610327

[35]Hsieh, J.-W., Yu, S.-H., Chen, Y.-S., & Hu, W.-F. (2006). Automatic traffic surveillance system for vehicle tracking and classification. IEEE Transactions on Intelligent Transportation Systems, 7(2), 175–187. doi:10.1109/tits.2006.874722

[36]Irwin, A., Ross, A., Furnish, G., & Babcock, H. (2008). PLplot home page - main. Retrieved November 24, 2016, from PLplot, http://PLplot.sourceforge.net/

[37]Jiang, Y., & Wang, S. (2006). The Human Visual Recognition Ability for Incomplete Letters. First International Conference on Innovative Computing, Information and Control - Volume I (ICICIC'06). http://doi.org/10.1109/icicic.2006.364

[38]Kerr, D., Scotney, B., & Coleman, S. (2008). Interest point detection on incomplete images. 2008 15th IEEE International Conference on Image Processing. http://doi.org/10.1109/icip.2008.4711880

[39]Keefer, R., & Bourbakis, N. (2015). A survey on document image processing methods useful for Assistive technology for the blind. International Journal of Image and Graphics, 15(01), 1550005. doi:10.1142/s0219467815500059

[40]Lehmkühler, A. (2009). A java PDF library. Retrieved November 24, 2016, from https://pdfbox.apache.org/

[41]Li, W., Zhao, L., Xu, D., & Lu, D. (2013). Efficient Image Completion Method Based On Alternating Direction Theory. In D. Suter, 2013 20th IEEE International Conference on Image Processing (ICIP) (pp. 700–703). Melbourne, Australia: IEEE.

[42]Liu, J., L., Przemyslaw, M., Peter, W., & Jieping, Y. (n.d.). Tensor completion for estimating missing values in visual data (pp. 2114 – 2121). IEEE. http://doi.org/10.1109/ICCV.2009.5459463

[43]Live Scan Fingerprinting. (n.d.). Retrieved 3 October 2015, from http://www.fingerprinting.com/live-scan-fingerprinting.php

[44]Lowe, D. G. (1999). Object recognition from local scale-invariant features. Proceedings of the Seventh IEEE International Conference on Computer Vision. doi:10.1109/iccv.1999.790410

[45]LundBlad, A. (2018). AtomicInteger. Retrieved from https://stackoverflow.com/users/276052/aioobe

[46]Marsolek, C. J., & Burgund, E. D. (2002). Visual Recognition and Priming of Incomplete Objects: The Influence of Stimulus and Task Demands. Rethinking Implicit Memory, 139–156. http://doi.org/10.1093/acprof:oso/9780192632326.003.0007

[47]Oliver, A. C., Stampoultzis, G., Sengupta, A., Klute, R., & Fisher, D. (2002). Apache POI - the java API for Microsoft documents. Retrieved November 24, 2016, from https://poi.apache.org/

[48]OpenCV Developers Team. (2016). About OpenCV. Retrieved November 24, 2016, from OpenCV, http://opencv.org/about.html

[49]OpenCV Development Team. (2019). Adding (blending) two images using OpenCV. Retrieved from https://docs.opencv.org/2.4/doc/tutorials/core/adding_images/adding_images.html

[50]OpenCV Development Team. (2019). Canny Edge Detection. Retrieved from https://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html

[51]Osisanwo, F. ., Akinsola, J. E. ., Awodele, O., Hinmikaiye, J. ., Olakanmi, O., & Akinjobi, J. (2017). Supervised Machine Learning Algorithms: Classification and Comparison. International Journal of Computer Trends and Technology, 48(3), 128–138. https://doi.org/10.14445/22312803/IJCTT-V48P126

[52]Riemenschneider, H., Donoser, M., & Bischof, H. (2010). Using partial edge contour matches for efficient object category localization (11th European Conference on Computer Vision ed.). doi:10.1007/978-3-642-15555-0_3

[53]Robbeloth, M., & Bourbakis, N. (2015). An LG Graph Monitoring Scheme for Representing Incomplete Objects. National Aerospace & Electronics Conference & Ohio Innovation Summit (NAECON-OIS)

[54]Roques, A. (2009, April 17). Open-source tool that uses simple textual descriptions to draw UML diagrams. Retrieved November 24, 2016, from PlantUML, http://plantuml.com/

[55]Schahczenski, C. (2018). Weka Error Measurements. Retrieved from https://katie.mtech.edu/classes/csci347/Resources/Weka_error_measurements.pdf

[56]Sedlacek, J., & Hurka, T. (2017). VisualVM. Oracle. Retrieved from https://visualvm.github.io/

[57]Shin, B.-S., Zheng, Y., Russell, J., & Klette, R. (2012, November 26). Improved segmentation for footprint recognition of small mammals. http://doi.org/10.1145/2425836.2425890

[58]Simpson, B., & Toussi, F. (2018). HyperSQL Database Engine 2.4.1. Retrieved from http://hsqldb.org/doc/2.0/guide/guide.pdf

[59]Shapiro, L. G., & Stockman, G. C. (2001). Computer vision. United States: Prentice Hall.

[60]Tang, S., Andriluka, M., & Schiele, B. (2013). Detection and tracking of occluded people. International Journal of Computer Vision, 110(1), 58–69. doi:10.1007/s11263-013-0664-6

[61]Tianxu, Z., Nong, S., Guoyou, W., & Xiaowen, L. (1996). An effective method for identifying small objects on a complicated background. Artificial Intelligence in Engineering, 10(4), 343–349. http://doi.org/10.1016/0954-1810(96)00014-3

[62]Ukkonen, E. (1992). Approximate String Matching with q-grams and Maximal Matches. Theor. Comput. Sci., 92(1), 191–211. http://doi.org/https://doi.org/10.1016/0304-3975(92)90143-4

[63]Wexler, Y., Shechtman, E., & Irani, M. (2007). Space-time completion of video. IEEE Transactions on Pattern Analysis and Machine Intelligence, 29(3), 463–476. http://doi.org/10.1109/tpami.2007.60

[64]Wu, B., & Nevatia, R. (2005). Detection of multiple, partially occluded humans in a single image by Bayesian combination of edglet part detectors. ICCV

[65]Wu, B., & Nevatia, R. (2008). Detection and segmentation of multiple, partially occluded objects by grouping, merging, assigning part detection responses. International Journal of Computer Vision, 82(2), 185–204. doi:10.1007/s11263-008-0194-9

[66]Wu, J. Y., Lim, K. B., & Tan, M. H. Y. (2012). Spectral technique to recognise occluded objects. IET Image Processing, 6(2), 160. doi:10.1049/iet-ipr.2010.0505

[67]Xiao, M., Li, G., Xie, L., Tan, Y., & Mao, Y. (2015). Contour-guided image completion using a sample image. Journal of Electronic Imaging, 24(2), 023029. http://doi.org/10.1117/1.jei.24.2.023029

[68]Yamaguchi, O. & Fukui. (2002) K. Pattern hashing - object recognition based on a distributed local appearance model Proceedings. International Conference on Image Processing.

[69]Zhang, J., & Zhang, Q. (2007). Noniterative blind image restoration based on estimation of a significant class of point spread functions. Optical Engineering, 46(7), 077005. doi:10.1117/1.2757195

[70]Zhong, S., Liu, Y., Chung, F., & Wu, G. (2012). Semiconducting bilinear deep learning for incomplete image recognition. Proceedings of the 2nd ACM International Conference on Multimedia Retrieval - ICMR '12. http://doi.org/10.1145/2324796.2324836

[71]Xue, T., Rubinstein, M., Liu, C., & Freeman, W. T. (2015, July 27). A computational approach for obstruction-free photography. http://doi.org/10.1145/2766940

[72]Buades, A., Coll, B., & Morel, J.-M. (2011). Non-Local Means Denoising. Image Processing On Line, 1. https://doi.org/10.5201/ipol.2011.bcm_nlm

[73]Bourbakis, N.(1988), The Local Global Graph Model for Object Representation and Recognition, Technical Report 4, Jan. 1988, GMU, VA, USA

[74] Bourbakis,N. (2002), Emulating human visual perception for measuring differences in images using an SPN graph approach, IEEE Trans on Systems, Man and Cybernetics, vol. 32, no. 2, pp. 191-201, 2002

[75]M. Robbeloth and N. Bourbakis, Progressive recognition of incomplete objects, Int. Journal on AI tools, tbs.