

Counting Boxes
Aaron Holloway, Andrew Wietecha
Computer Vision (CSS 487) Autumn 2018
University of Washington Bothell
Dr. Clark Olson

Description	3
Terminology	3
Project Goals	3
Initial Goals	3
Refined Goals	3
Process	4
Image Assumptions	4
Adjustments to Input Image	4
Edge and Line Detection (A)	7
Contour Detection (B)	9
Counting from Lines (A)	11
Counting from Contours (B)	11
Explanation of Results	11
Lessons Learned	11
Deciding on an Approach to the Problem	13
Future Development/Next Steps	15
For This Project	15
For Practical Application	15
References	16

Description

This project aims to make a piece of software that counts boxes stacked on a warehouse pallet, using concepts and methods learned in Computer Vision. The overarching aim is that this software could be used to assist warehouse workers, reducing countback time and allowing workers to pick faster.

The portion of this overall goal completed for this project is identifying key points in the image, such as edges, lines, and contours, and pruning those results to only the important features.

Terminology

- **Layer** - a set of boxes of the same height that together cover approximately the top area of the pallet on which they are stacked. Layers make counting boxes on pallets fast, if the person counting know their pattern or the number of boxes per layer. For example, large boxes may be stacked 6 lengthwise, and 2 widthwise, making 8 boxes per layer. For a full pallet of 4 layers, it makes it easy to then count that there are 32 boxes on the pallet.
- **Pick/picking** - a term from warehousing, referring to taking boxes from a pallet. An employee may have a “pick order” for 3 boxes of a product, so they remove (pick) 3 boxes from a pallet.

Project Goals

Initial Goals

Our initial goal was to count boxes stacked on a warehouse pallet. We wanted to do this for a variety of common situations, including a full pallet, pallet with a partial top layer, and a nearly-empty pallet where the base of the pallet is partially visible. Edge cases would be ones such as: boxes of different sizes on the same pallet, more than one layer picked from, and crushed/damaged boxes. We intended take a field trip to a warehouse or warehouse-like environment to both gather photos, and gather more information from those who work there.


Refined Goals

Detecting boxes, the edges of a stack, or useful features of the stack proved a much harder task than anticipated. Our goal shifted to just counting boxes in single image of an ideal stack, which still proved difficult. Currently we are working to achieve a milestone towards that goal: get a clean enough set of features, such as lines, or box faces via contours, that we can extrapolate a count of all the boxes on the pallet.


Process

Initially we planned to take a trip to a warehouse-like area, such as UW Seattle’s shipping & receiving, or a Costco/Sam’s Club. However, due to having limited time to commute outside of class and wanting to spend more time coding than taking photos, we settled on a few photos from the internet, as well as a handful Aaron got permission to take at a previous place of employment. This way, we had real-world images to test how our program would work “in the field” as well as more idealized computer renders to verify the basic algorithms work as intended.

To start off, a few main sample images were selected. We eventually settled on a few:



full-72.jpg



render-multiple-stacks.jpg



Image Assumptions

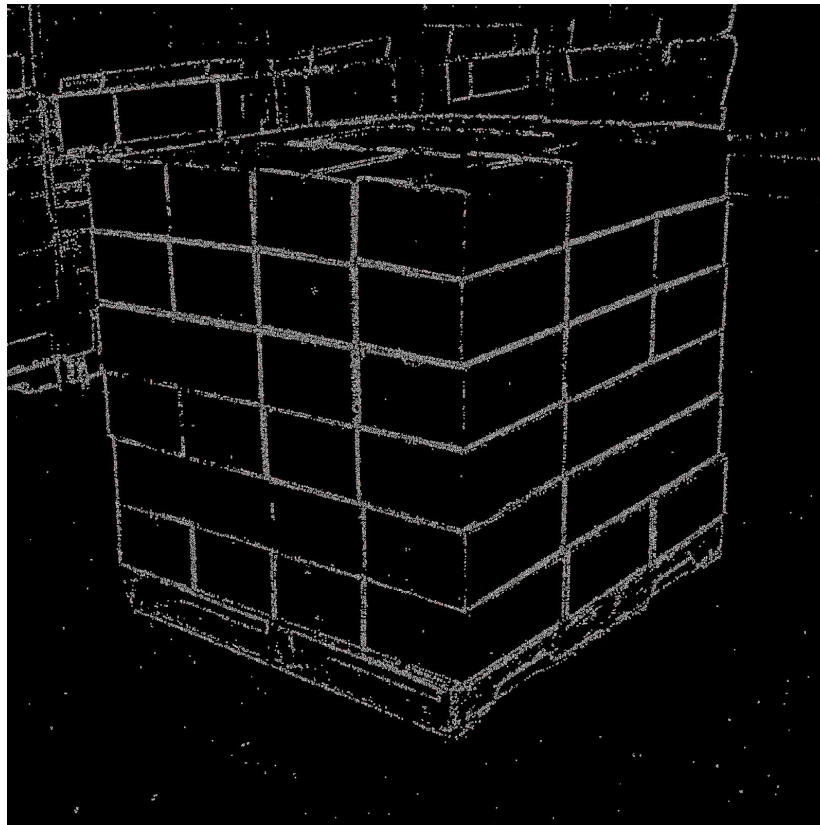
In order to develop and test any of our algorithms, we needed to decide what sorts of images would be used. What we assume/require of the input image is:

1. Angled shot such that a corner of the pallet is pointing towards the photographer
2. Top layer is picked front-to-back (items furthest away are removed last)
3. Only the topmost layer is picked from (no items are removed from lower layers until the top layer has been consumed)

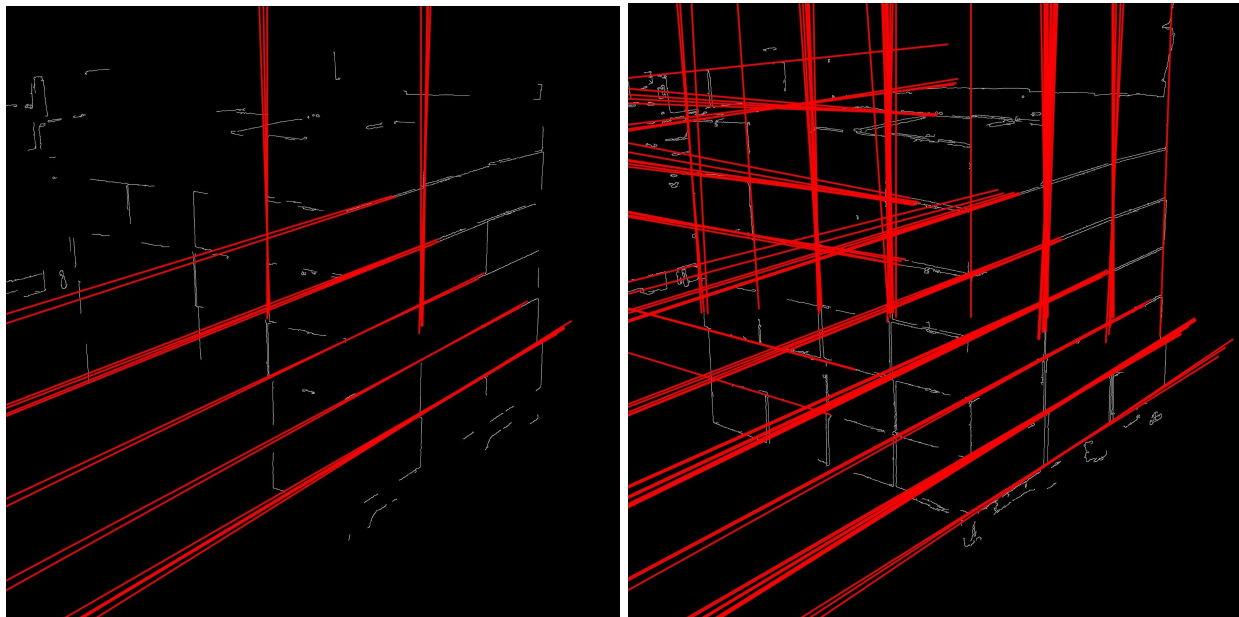
Adjustments to Input Image

Simply applying feature detection (such as Canny edge detector, Harris corner detector) to a grayscale version of the input image yielded noisy results, despite that fact that the OpenCV

implementation of Canny edge detection includes noise reduction via blurring. [1] In particular, a high resolution, but relatively poorly-lit photo of an ideal pallet stack gave a very noisy and incomplete set of edges. With some refinement of the Canny parameters, we got many of the lines, however the forward right edge of the top layer was not being detected. The transition here was mostly a bright edge, so we thought to increase the transition, and thus the likelihood of it being picked up as an edge, by increasing the saturation and contrast. This was further improved by doing a second pass, with a smaller adjustment to saturation. (However, the second pass was only necessary for full-72.jpg. For our other test images, a single pass was enough). The exact parameters for adjusting saturation and contrast will likely vary between images. We also found that blur must also be adjusted based on image resolution, with a high resolution image requiring more blurring. While these adjustments made edges more apparent, they also magnified the noise, so a blur was added for some extra smoothing. We tried Gaussian, median, and bilateral. Gaussian was best at cleaning random noise, and a median blur just afterwards swept up most of the remaining noise. MedianBlur and BilateralFilter techniques were attempted to try to preserve line strength while diminishing surface noise. In the end, Gaussian was still the most generally useful across multiple images; however, it blurred edges slightly more than BilateralFilter. The only problem that remained was the unnecessary background features.

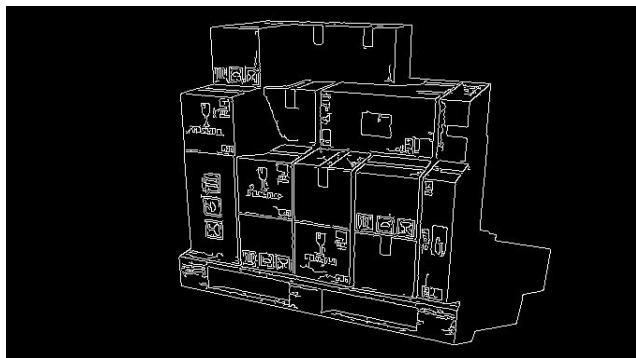


Early Canny image above - notice the significant amount of noise, and “fuzzy” lines detected.

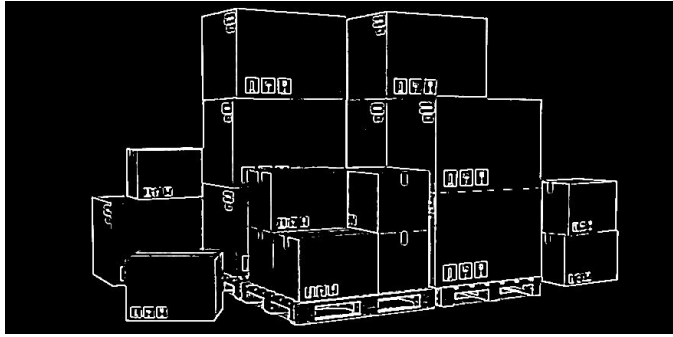


Above is an early Hough line detection - before and after adjusting the saturation, contrast, and adding extra blurring. Notice how the top forward right edge, and the bottom forward left edge are still undetected. These are two of the lines we had difficulty getting, and are the reason we chose to adjust the saturation and contrast.

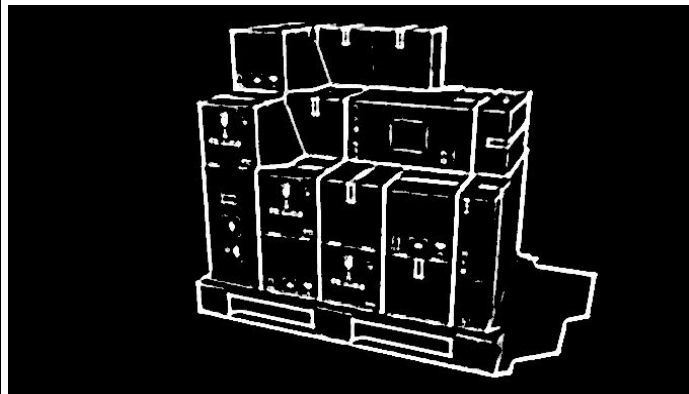
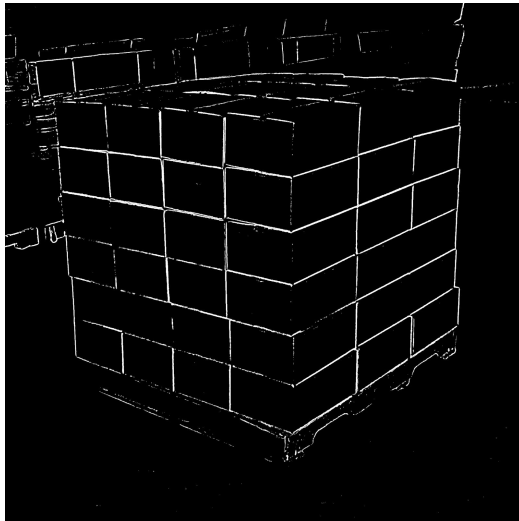
Early Canny:



Further Noise Removal:



Laplacian:



We also tried line detection with a Laplacian edge detector, which gave stronger, cleaner edges compared to Canny detector, though it also shows more of the background features as well. Due to the stronger edges, we ultimately chose to go with the Laplacian method.

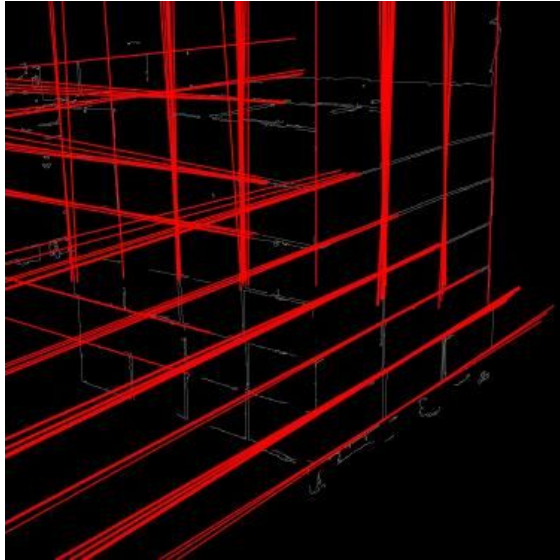
Edge and Line Detection (A)

Initially we implemented only edge detection, as it's something we were familiar with from class. It occurred to us later, though, that we didn't just need edges, such as the outline of the whole stack, but rather lines on the stack interior. The important lines were not just the outside edges, but the lines formed from crevices between boxes.

We started with a Canny edge detector and no image pre-processing. This resulted in a lot of noise, as explained earlier. A lot of time was spent refining the blur, contrast, and saturation settings in order to get a usable image.

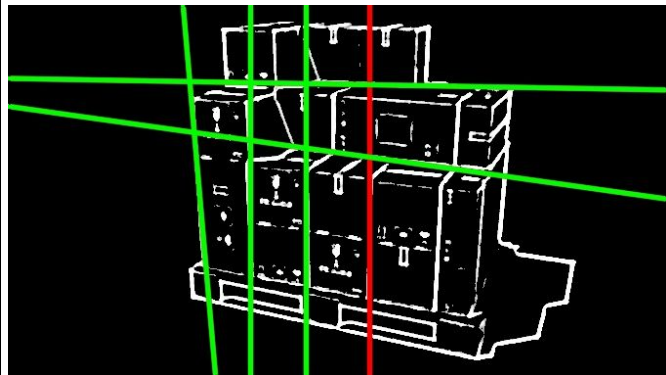
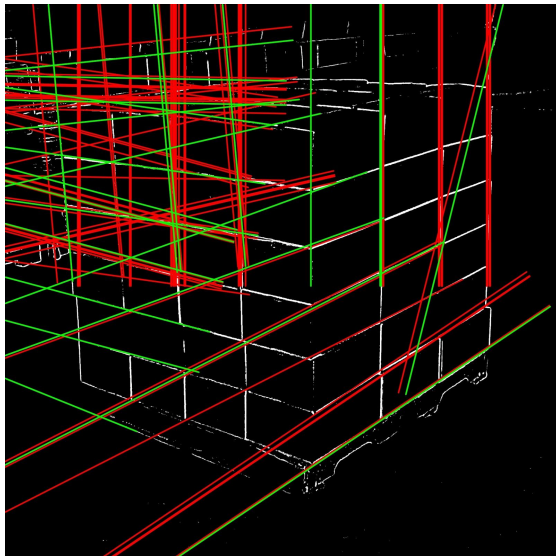
We were eventually able to accomplish fairly accurate edge detection, and in an effort to refine our lines we tried both the Laplacian and Sobel operators. Sobel was quickly removed due to redundancy, but Laplacian proved more effective than Canny in some cases at creating well defined perimeter lines. However, Laplacian also had gray value noise that we culled to black under a threshold.

Later Hough line detection from Canny edge detection, after further refinement



There's only so much refinement that can be done using only the parameters for the `HoughLines()` method, so a method was developed to prune the lines. The algorithm iterates through a vector containing all the found line segments, and compares them to all the remaining lines. If the x and y difference is under a certain threshold, the line segment is removed from the list. As you can see below, this is effective at reducing some of the lines, however there are still issues with it: 1) some “junk” lines remain from the background, 2) it removes some lines we wanted to keep, and 3) the pruning does not guarantee that the *best* lines remain, only that lines that are “too similar” are removed.

LaPlacian edge detection, Hough Line detection, and pruning (original lines, green - lines remaining after pruning):



The lines were still not entirely reliable, however. As Andrew worked to refine the Hough lines, Aaron looked at another option that also built on the work completed thus far: contour detection. The hope was to find the box faces on the left and right sides, and along a vertical stack.

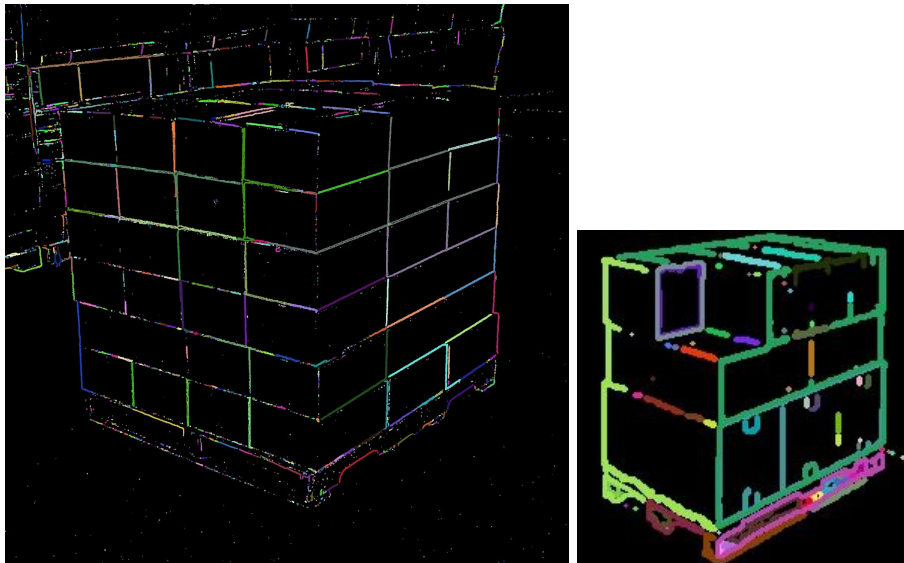
Contour Detection (B)

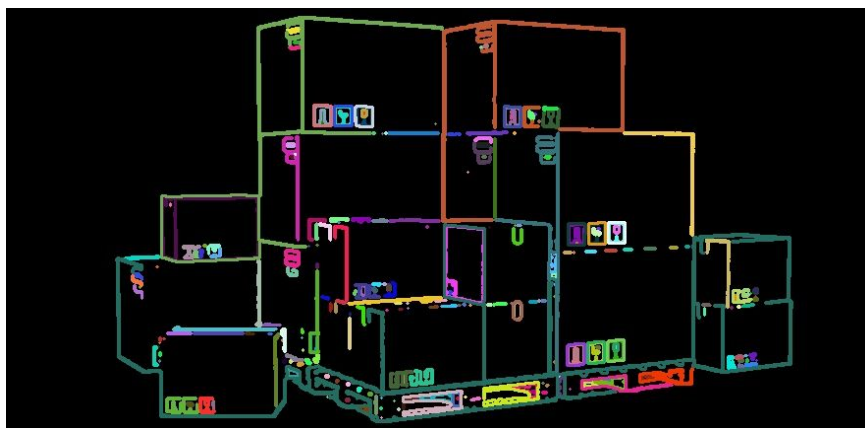
Contour detection was one method attempted to identify items on the pallet; this technique was somewhat successful, depending on the clarity of the preprocessed image (Contrast/Saturation, Blurring, Edge Detection). Background imagery in photos was very challenging to remove entirely, which resulted in a lot of small artifact contours to be removed. Even for any given box on the pallet, edges were not perfectly smooth. Inherently, the face of a box was set at a perspective angle from the camera and did not result in a clean square to be easily detected geometrically.

Initially, contours were fragmented along a single image. Applying Dilation to the Laplacian image before contour detection was able to create contours that wrapped an entire box face. However, the results were still quite imperfect. Some boxes had poorly lit lines that caused two or more boxes to identify as a single contour or bleed into the background.

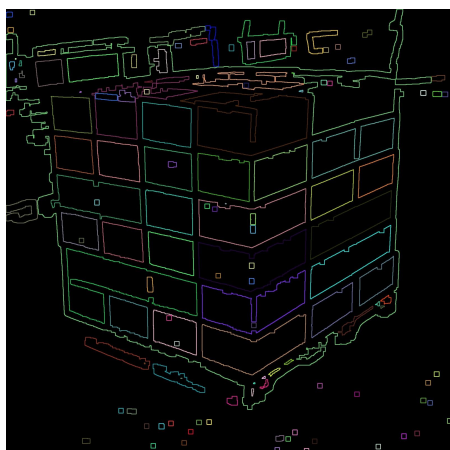
Late into testing, a script was written to cull contours under a certain arclength. It was originally intended to identify the most common arclength and eliminate the rest, but sadly the most common arclength ended up being the smaller artifact objects and not the box faces.

Initial Canny Results:



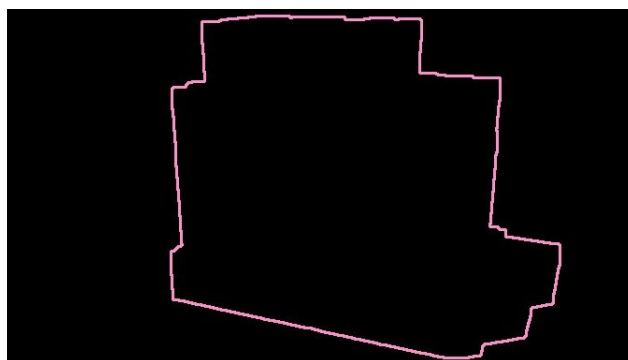
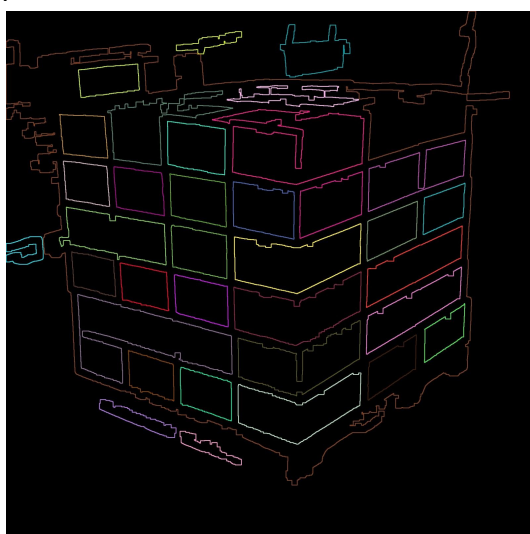


Post-Dilation on Laplacian:



Final (Post-Dilation, Low Arclength Culling):

As seen below, values that work well for a high resolution photo really cause problems for smaller images. We needed to find a way to normalize our images or else pass in variable parameters based on some factor of size.



Counting from Lines (A)

One of the ideas we had was to find the lines between boxes, and look for intersections or corners along 3 key axes: along the left stack face, along the right stack face, and along the closest vertical corner. We would then count these intersections and use that information to calculate a simple length x width x height volume in boxes. Since we weren't able to isolate only the lines we needed, we did not get to this phase of implementation.

Counting from Contours (B)

Despite having an easy count of total contours, it was less discernible which were boxes and which belonged to other objects in the scene. A problem that, once solved, would still not have been enough. A count would need to be done on both sides of the visible pallet to multiply a vertical count with both halves of the horizontal count (angled differently), but defining which contours belonged to which side wasn't straightforward.

We were unable to come up with a solution to how to easily identify contours along a line, nor did we figure out a good way to bound the search space to the pallet and its contents.

Explanation of Results

Our results fell short of our initial goal, as finding important features proved a greater challenge than expected. We were able to successfully produce fairly clean edge detection for images at a variety of resolutions. We were able to produce lines that corresponded to some of the major axis along the pallet, but frequently had too many or too few lines. We were also able to create contours for those images that similarly identified box faces. Sadly, identifying lines and contours in a way that allowed us to easily and accurately count each of 3 dimensions was not something we were able to finish.

Lessons Learned

- Image Distinctions
 - Considerations like resolution and lighting have serious and compounding impact on every other step of the process. We could have saved time by working with a single image, but building a system that can accommodate a large variety is no small feat.
- Pre-Processing Image
 - Playing with saturation/contrast helped to increase the difference between edges and non-edges, improving edge and line detection results
- Noise Elimination

- Pre-processing also magnified noise, so further noise reduction was necessary
- Different blurring techniques can be used, together or separately, to address certain types of noise.
- Fine tuning these parameters, number of passes, and sequence of operations takes a lot of time and tinkering.
- Corner Detection
 - This didn't quite work how we were expecting it to, and we decided it wasn't valuable for our application. We may have still been able to combine this information with line/contour data to narrow down our space somehow.
 - PreCornerDetection may have been useful. Further understanding of a feature map is needed.
- Edge Detection
 - Pre-processing the image greatly improved results, though still fell short of ideal.
 - Canny was great for hard edges, but had a lot of erratic motion in the lines. Not so good for following curves, as we learned in class.
 - Laplacian was great for tracing our shapes, but had a lot of errant gray data that we had to spend extra time trying to remove.
 - These results in particular had everything to do with our noise elimination sequence.
- Line Detection
 - Pruning may be necessary, though a simple implementation only reduces, it doesn't necessarily keep the lines we want. A further-refined line detection algorithm could possibly improve pruning results.
 - Too many lines, not enough lines, lines deviating in unexpected ways. Getting exactly the 3 lines we wanted out of the picture was not nearly as easy as we imagined it would be.
 - Even getting lines between boxes, a simple human task, is not so simple for computers.
- Contour Detection
 - Contours gave us nice representative shapes near our objects, but were initially disjointed and hard to isolate from artifact contours and melded shapes.
 - We probably could have combined contour and line data if we had been able to isolate them further in order to identify boxes along our axis.
- Working with Data
 - Large vectors of lines, or vectors of vectors of points, are not very intuitive to iterate through for very specific information. In the absence of a well known solution, even something as simple as finding unique values (or angles) among the set is a task that can take hours to implement.

Part of the challenge, and frustration, stemmed from the fact that the task of estimating the number of boxes is very simple for us as humans. Boxes are very easy for us to identify, and to extrapolate unseen sides/edges from. Computer Vision, at least the methods we were introduced to, can only see in 2D. The computer (without a machine learning module) has no

experience, and without a depth sensor, no sense of depth. What is easy for us, we needed to analyze and break down into even simpler tasks so a computer can attempt to mimic what we see and how we process it. This is the very essence of computer vision. Having come through this project, though we did not meet our initial goal, we have a much greater understanding and appreciate for this complex area of computer science.

Deciding on an Approach to the Problem

About half of our time was spent deciding how to approach the problem, testing implementations of an experimental approach, and refining parameters to see if a better result could be obtained.

We brainstormed several possible approaches, and experimented with ideas from all of them:

- Find the general cube that contains all of the stacked boxes. This could be done by finding the pallet edges, perhaps using some feature matching to identify the side of the pallet (the holes where a forklift can pick up the pallet).
This would eliminate any background “junk” features that would be present in real life, such as warehouse shelves and other pallets.
 - Then detect lines, corners, or other useful features within this bound.
- Find important, and likely strong (easy to detect) edges such as the left and right edges of the pallet, and a vertical edge of the corner nearest the camera. With just these 3 lines, we could find intersections or corners, and count them along each of those lines. We could then perform a single $L \times W \times H$ calculation.
 - Further lines would be needed for a partial top layer
- Use size information to calculate volume, since in our application, we’d know the general sizes of pallets - a rare piece of information not found from the images themselves.

At first we considered calculating the measurements of boxes along the pallet against the length of the pallet itself (which is often a standard measurement). Over the course of our design meetings, we realized that our goal was much simpler and we didn’t need to measure anything, only count boxes. Thus, it wouldn’t matter if the pallet or boxes were measured in inches, centimeters, miles, fish, or bananas. A box was one unit, and that was all the measurement information we needed. Instead, we decided to focus on isolating the pallet in the image and counting rows/columns of boxes by looking at corners or edges.

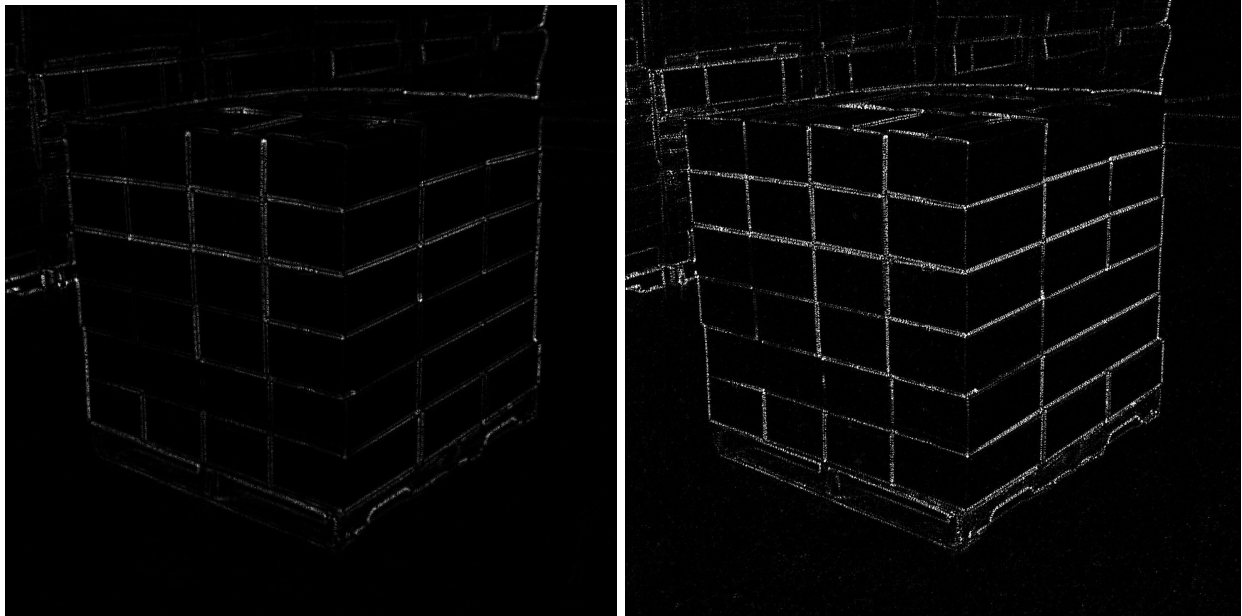
In our initial design, we wanted to isolate the pallet with a bounding box. We thought we could measure corners and extend a line from one corner to another to create our space. We also knew lines across our three dimensions would be critical, but underestimated how difficult it would be to produce them cleanly.

Similarly, we didn’t consider the need for per-photo adjustments to size, operation parameters, differences in noise intensity, and background details causing conflict. We did plan to address some of these with a region of interest, but we didn’t consider how difficult finding an ROI would be while these issues were in place.

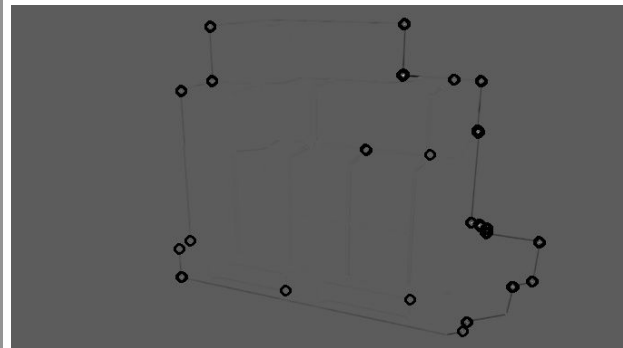
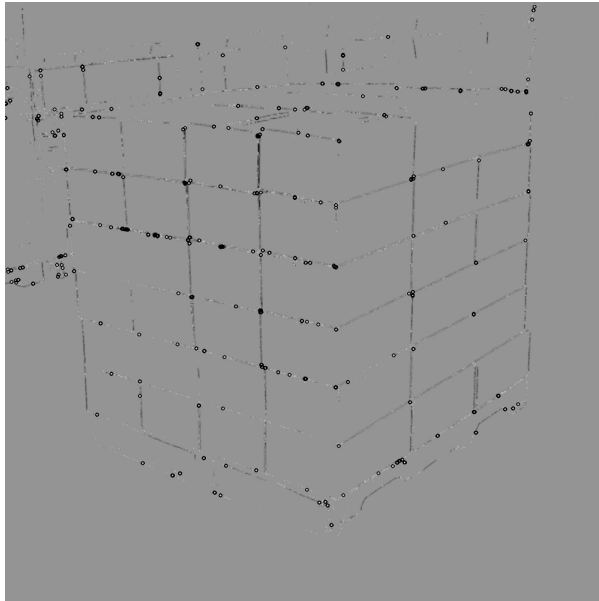
Yet, when it came to counting boxes along an axis, edge detection was too jagged and uneven to identify box divisions. We decided to turn to lines and/or box faces for counting.

One of our first attempts at finding useful information from the image was a `preCornerDetection()` method found in OpenCV's documentation while looking for feature detection methods. This gives a feature map, but we were unsure how to use this with other methods we know of for feature detection. A quick search didn't turn up how a feature map could be used in feature detection, so the idea was scrapped.

`preCornerDetection` results:



Another early effort was made to identify the points where boxes met by using corner detection. The concept was to count corners along each of three axis. We found quickly that it was susceptible to noise, but also frequently found corners in mass and along edges. We deemed it unusable for a simple counting technique, and moved on to lines and contours.



Future Development/Next Steps

For This Project

- Bounding Area Around Pallet and Contents to help eliminate background data as false positives in a counting process.
- Segmenting contours into lines along one of three (most common in bounding space?) axis.
- Identify key lines that correspond to the 3 axis we are trying to count across.
- Count line intersections.
- A last minute realization was that we may be able to count across the axis on the Laplacian by identifying the number of times value changes from black to white along a single (key) line. This would eliminate Hough and Contour Detection from the process and might have simplified things for us. However, this still requires the ability to identify which lines in the image are the most appropriate 3 to count along and may have other unforeseen challenges.

For Practical Application

The ultimate goal, in which we achieved some measure of progress towards, is use in a warehouse environment. The idea is to have a pared-down AR headset, such as a Hololens

with minimal features. The user would stand to the side of a pallet, with enough of an angle so that 2 sides are clearly visible, and press a button on the side of the headset to take a photo and start the counting process. The headset would then display the count for the current pallet, and could also send this information to the warehouse's database. The headset could also provide the picker with information, such as how many items to pick from this location.

The headset would likely only require:

- 3-6 color AR display: red, green, yellow, and second set of colorblind-safe colors
- Limited audio - negative and affirmative beeps
- Depth camera

It would NOT need spatial audio, millions of colors, or even to constantly be on.

Other features could be developed using CV, to further assist picking and even inbound/outbound auditing, including: reading UPC, date, and lot code, and checking this against the list of UPC, date, and lot code for the corresponding order. Verification of the order, or any discrepancies, could be reported automatically. The all auditor would need to do is walk around the pallet, taking photos, or perhaps video, from every side. This could greatly reduce the time spent auditing orders, allowing more orders to be audited and prepared for shipping in a single shift.

Expanding to further features beyond the scope of computer vision, the headset could also track the location of its wearer and forklifts operating within the warehouse, giving the wearer a visual and/or audio alert when there is a forklift approaching. Furthermore, the headsets could be used to display pick order information, eliminating handhelds and wrist devices, leaving both hands free to pick up boxes and operator warehouse equipment such as pallet jacks.

The headsets would be a limited-feature version of their current 2018 commercial models, and may save on battery by only using the camera when the button is pressed, and not needing to “watch” for motion controls or hand gestures (unless a gesture is to be used to initiate the count)

If this were to be used with an augmented reality (AR) headset, such as the HoloLens and HoloLens 2, or even a device equipped with a Kinect, we would be able to make use of the depth camera such hardware is equipped with, as suggested on a Stack Overflow question [2]. This could make detecting corners and broad surfaces, such as the sides of a stack, or even crevices between boxes, much easier. [2]

References

[1] Canny Edge Detection. OpenCV Documentation 3.1.0
https://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html

[2] Cube detection using C++ and openCV. Stack Overflow.
<https://stackoverflow.com/questions/31555867/cube-detection-using-c-and-opencv>

Used as a starting point:

Opencv detect cubes (corners). Stack Overflow.

<https://stackoverflow.com/questions/31772804/opencv-detect-cubes-corners>