Automatic Tool Holder Generation
Via Image Processing and Laser Cutting

Miriam Shamash

GSTEM

Professor Daniele Panozzo
NYU Courant Institute of Mathematical Sciences

Abstract

We propose a method to efficiently and robustly create a case to secure an object in place. This method can be used for producing environmentally-friendly shipping containers to hold objects securely during transport, or to create tool-holders to maximize organization and decrease clutter in a workspace. To achieve this goal, we propose an automated pipeline to convert a raster image of the desired layout, acquired by a commodity camera, into a set of toolpaths for a laser cutter. The image is processed by eliminating perspective distortion, thresholded and dilated to extract the outlines of the objects, and finally converted into a vector graphics file. A laser cutter can then produce cardboard sheets that, when stacked on top of one another, realize the tool holder. This algorithm produces a holder to hold any object securely, regardless of shape. We demonstrate the practical applicability of our algorithm on a set of challenging test cases, and we release our reference implementation as an open-source project to foster adoption of our technique.

Automatic Tool Holder Generator Via Image Processing and Laser Cutting

Developing shipping containers is often a step overlooked in the process of product distribution. When thinking about the online shopping process as a whole, one may picture a warehouse stacked to the ceiling with shelves full of boxes, or a package resting on the floor of a UPS delivery truck. However, a crucial step in the shipping industry is ensuring the security of objects in their respective cases. It is important for all those involved in the delivery process: customers do not want their hard-earned money to go to waste when their purchase arrives shattered, and companies strive to maintain a good reputation by ensuring positive customer experiences.

While materials like bubble wrap or packing peanuts are commonly used for this purpose, it is not environmentally friendly, cost efficient, or tidy for businesses to place these fillers in millions of packages per day. Therefore, producing a *new* method that can effectively produce secure cases for objects (without added materials) is essential to the success of a business.

My project aims to create a shipping container for an object that will protect it during delivery, without adding unnecessary and environment unfriendly materials. I aim to produce a recyclable, inexpensive cardboard sheet with cutouts for any object to fit in. These sheets of cardboard would be stacked one on top of another and glued down to firmly secure an object in place. In this project, I will specifically research how to effectively produce a holder for common, everyday tools such as a screwdriver, measuring tape, flashlight, and more. I will test different dilation values and amount of cardboard needed to make the tool holder as safe and stable as possible.

This can be achieved by creating a program that receives an image, processes the image, and returns an outline of the objects found within the image. I hypothesize that by analyzing the individual pixels found within the image, line segments outlining the contours of a particular object can be developed. These line segments can be written to a file which, in theory, can be send to a laser cutter. The laser cutter can then produce a holder for any object to fit securely.

Materials and Methods

The project is coded in Python, a scripting language widely used for its multitude of libraries and simple syntax. The code was written within Sublime Text, a common text editor, and was launched directly from terminal, a command-line system. The application was run on a 13" Macbook Pro.

Methods

Because this project requires an understanding of image processing, it is crucial to define the two types of computer graphics - raster graphics and vector graphics. Raster graphics, or bitmaps, are composed of a grid of pixels. Each pixel contains an RGB (red, green, blue) value that determines its specific color and shade. Scaling a raster image reveals the individual pixels and often results in a "blurry" looking view of the image. Raster graphics are often downloadable as JPG or PNG files. On the other hand, vector graphics are composed of a fixed set of line segments that represent shapes. They use mathematical relationships between points and the paths connecting them to describe the image. Scaling vector images preserves the shapes instead of making them appear fuzzy. Common vector formats include SVG, DXF and EPS files.

In order to cut objects using the laser cutter, the image sent to the cutter must be a vector image. However, typical images taken by a camera or a phone are raster images. Therefore, the primary goal of this project is to convert a raster graphic to a vector graphic.

Vectorization, the process of converting a raster graphic to a vector graphic, is already a highly studied field (Adobe). While it could be done manually by measuring and drawing each line segment within an image, automatic vectorization is much faster and extremely accurate. By grouping same-color pixels of a raster image into lines, curves or outlines shapes, a single object could be formed (Pepper, K. N.). In cases where the raster image contains many colors, Color Quantization must occur where the amount of colors are reduced in order to make vectorization faster.

While this method is successful, it does not suit the purpose of this project. When using the laser cutter to cut out holders, only the outline of the shape of the object is necessary. Vectorization conserves details such as color and line segments within the objects outline, unimportant details for the process of creating tool holders. As a result, I implemented my own algorithms to convert a raster image to a vector image.

To begin, I took a picture of a set of tools with a camera and eliminated the image distortion by utilizing the OpenCV library. Next, I converted the altered image to grayscale, and then to black and white. After this, I dilated the objects within the image and creating line segments outlining these shapes. I combined the line segments to create one fluid path. Next, I eliminated jagged edges by smoothing out the path. Lastly, this path was written to an SVG file. A more detailed, technical report of the steps taken is described below.

1. Eliminating Image Distortion

    The first step in the process is to take a picture of objects on a given background with either a camera or phone. Because it is difficult to place the camera directly parallel to the plane of which the objects lay on without a tripod, the image will be distorted. This results in an angled view of the objects, which can later contribute to incorrect sizing. Correct sizing is very important for fitting the objects shape, consequently, the image distortion must be removed. This can be achieved by using OpenCV, an open source computer vision and machine learning software library (Kaehler, A., & Bradski, G. R.). The functions used are getPerspectiveTransform and warpPerspective. These functions work hand in hand to compute the perspective transform matrix and then apply it, along with the image and its size, to produce the warped image. For the image to be altered, the four coordinates of the image's corners must be inputted (See Figure A). The resulting image, as shown in Figure B, appears with the desired background of the white 11" x 17" paper and not the floor as well.

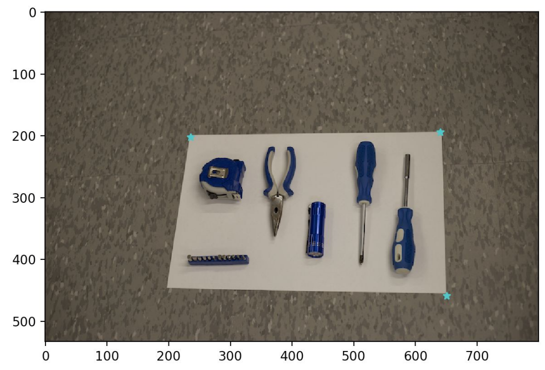

Figure A

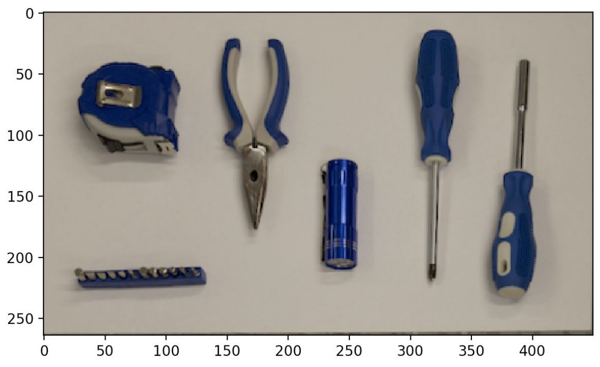

Figure B

Figure A: Original image of tools laid out on a white sheet of paper on the floor.
Figure B: Image of tools after image distortion has been eliminated

2. Converting to Grayscale

At this stage, the image taken with the camera is a raster image. In order to convert to a vector graphic, the image must first be transformed to grayscale. By averaging the RGB values of each individual pixels and assigning the RGB values with that average, an image can be converted to grayscale. All of the color values, ranging from 0 (black) to 1 (white) essentially cancel each other out and become a shade of grey, resulting in an image like Figure C.

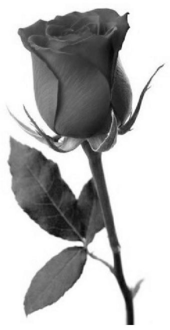

Figure C: Visual representation of a grayscale raster image

3. Converting to Black and White

Next, to convert the image to solely black and white, each pixel must be reanalyzed. If the new grey color of a pixel is darker than the user-inputted threshold, the pixel is changed to black (RGB value of 0). If the new grey color is lighter than the user-inputted threshold, the pixel is changed to white (RGB value of 1). Consequently, all the resulting pixels within the image are either black or white and result in an image like Figure D. The image can be saved as a PNG file.

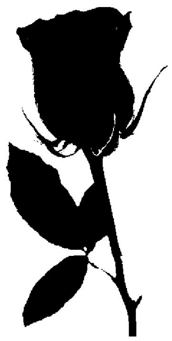

Figure D: Visual representation of a black and white raster image

Converting a raster picture to black and white is essential. Once there are only two colors, it is much easier to determine whether groups of pixels make up the same shape, since pixels making up an object would be the same color. Additionally, if an image is in greyscale, there is the potential of having shadows in the background. Thus, converting an image to black and white will differentiate the background of the image with the actual objects within the image.

4. Dilating the Image

In order for the object to lay comfortably, but securely, within the holder, the holder must be slightly bigger than the object itself. As a result, the black and white image of the object must be dilated. The OpenCV library is once again utilized, using the dilation function. For a given pixel within the image, if the surrounding kernel, or matrix, of the given size is black, then the pixel also becomes black (Cressie, N., & Serra, J). In this case, a 3x3 kernel is used. This process results in larger objects within the image.

5. Creating Line Segments

Afterwards, I wrote an algorithm that would create line segments based off of the dilated, black and white image. Each pixel within the image is looped through and each of the four surrounding pixels (top, bottom, left and right) are analyzed. If the pixel is not the same color as it's neighboring pixel, a line segment is drawn to separate the two, effectively creating a list of edges found within the shape once completed. This method produces a list of line segments in the format of (x1,y1), (x2,y2).

6. Creating One Path

Considering efficiency and time, it is not useful to send a list of individual line segments to the laser cutter. Instead, these line segments are joined together, allowing the laser cutter to cut through one consistent path (or paths if multiple objects are present in the image). Each of edges are looped through and the distance between the adjacent edges is computed by taking the square root of the sum of the length of each vector squared. If the distance between the two vectors is very small, in this case 1e-8, then the vectors are combined. The boolean array “visited” is created in order to ensure only the edges that were unvisited would be looked at. If the edge was visited, the loop will break. All these edges are then added together to create one path.

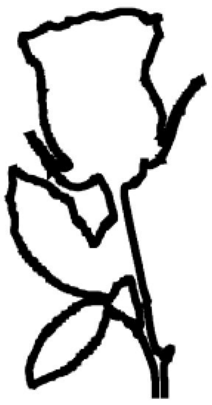
Figure E: Visual representation of an outline of an image

7. Smoothing Out Edges

In addition to creating one path, smoothing out the jagged points of the path will eliminate the total time that the laser cutter will take. Thus, another algorithm is implemented. A loop runs through each path and, excluding the first and last points, takes the average of the x and y values of the two adjacent points and replaces that line segment with the average, therefore getting rid of any rough lines and smoothing the lines out.

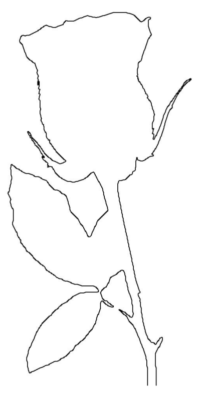
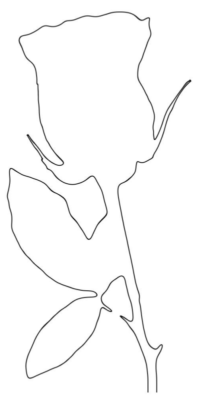

Figure F Figure G

Figure F: Outline before smoothing
Figure G: Outline after smoothing

8. Creating an SVG File

Next, an svg file is created using this new list of paths. Scalable Vector Graphics (**SVG**) is an XML-based vector image format for two-dimensional graphics. It is the equivalent of a PNG or JPG file for raster graphics. Before drawing all of the paths into the file, small paths are cleaned out to make the outline more cohesive and the printing more efficient. Additionally, conversions between pixels and inches must be made in order to ensure that the generated outline will actually fit the shape of the tools. This can because the width and height of the image, or the size of the white sheet of paper, is known. Lastly, the laser cutter requires lines to be 0.001 inches thick and the stroke of the lines to be completely black (RGB value of 0). These parameters are ensured within the code.

Procedure

In order to make my program easy to use for the everyday person, I created a user interface using Matplotlib, a Python 2D plotting library (See Figure H). The user interface walks the user through the process of creating the outline phase by phase and allows the user to adjust certain values by adjusting sliders and buttons. Said values include the number of times the image is dilated and smoothed, and the threshold value for converting to black and white. Additionally, the user interface allows the image distortion step to be interactive, allowing the user to click on the four corners of the canvas instead of manually inputting the coordinates. To advance to each step, the user can simply click on the “next” button. When the vector graphic of the outline is created, done in a bold red font appears.

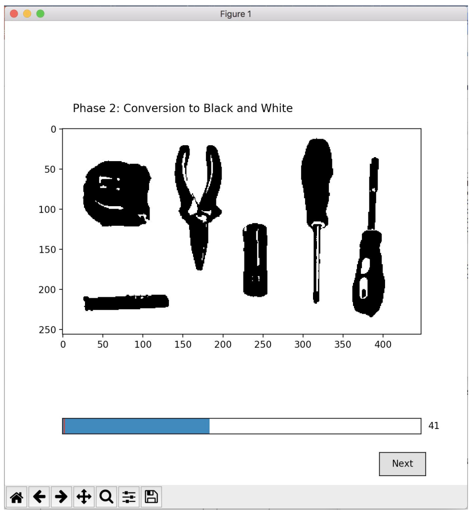

Figure H: Phase two of the user interface

Once the vector image was created, it could then be sent to the laser cutter. I used an Epilog Mini 18, which is desktop sized and features an 18” x 12” work area.

Results and Discussion

To maximize the security of the holder I created, I wanted to test which specific parameters would create the most successful tool case. I decided to test different dilation values because it would have the greatest impact on the fit of the tools within their respective cutouts. If the cutouts were too big, it wouldn't matter what the other parameters are - the tools would not be secure. As a result, the black and white threshold, clean up and smoothing numbers were kept the same in order to accurately test the dilation. The black and white threshold determined at which point the pixels would turn to black or white. The clean up number determined the length at which small paths would be eliminated from the final outline. Lastly, the smoothing number determined how many times the outline would be run through the smoothing algorithm. The black and white threshold was kept at 45, the cleanup number at 20, and the smoothing number at 50. I tested dilation values of 0, 10, 20, 30, 40, and 50 (See Figure I, J).

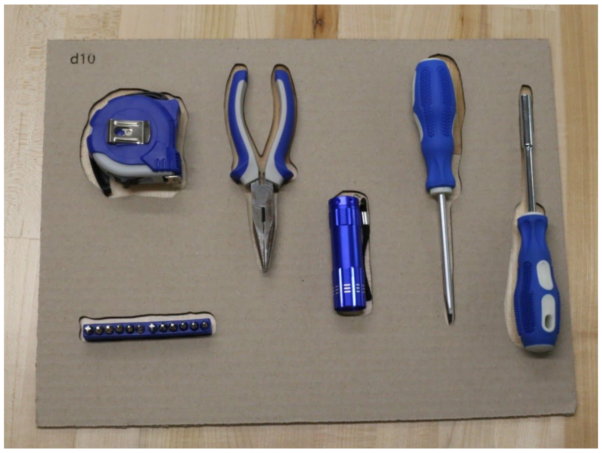

Figure I

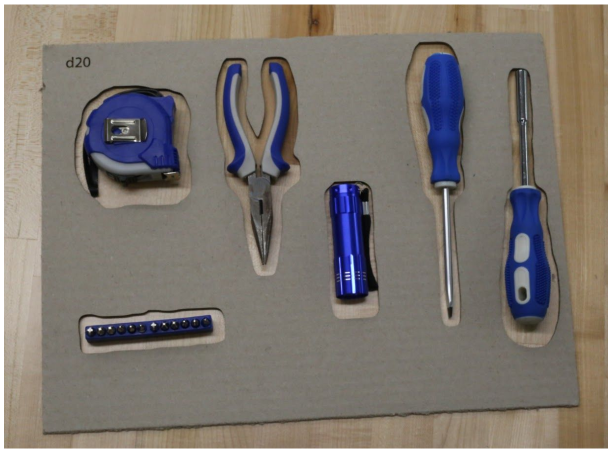

Figure J

Figure I: Dilation value of 10 (d10)
Figure J: Dilation value of 20 (d20)

After printing out each holder with the given dilation values, I concluded that a dilation value of 10 minimized enough movement to secure the tools in place yet allowed enough extra room to easily take the tools out of their cutouts. I then cut out six more holders with a dilation value of 10 and glued them together to produce the final prototype (See Figure K). Each holder took approximately one minute twenty three seconds to cut.

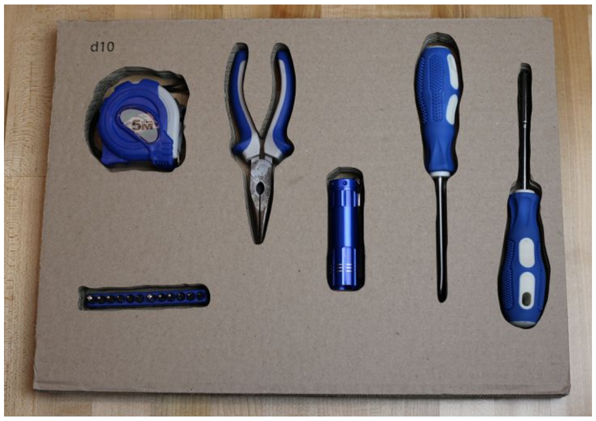

Figure K: Final result

Like any study, my project had limitations and given more time could be easily improved. The main limitations were image size, image type, and image material. Improvements can be made to the user interface.

Because my algorithm loops through each individual pixel within a given image, it would be difficult to process an image containing over a couple thousand pixels. Small images, such as 300 x 500 pixels, are ideal. Therefore, images too large must be manually resized before processed. In the future, I would automatically resize the image within my user interface.

Furthermore, images must be a PNG format to be analysed by my code. I would like to broaden the scope and either process other raster image formats or provide an option within the user interface to convert an image of a different format to a PNG. Additionally, given more time, I would add a text box to the user interface to allow input on the name of the SVG file created. The same could be done to select a photo to create the holder in the beginning of the process.

Lastly, I would like to test producing holders out of other materials, such as foam, with a laser cutter. Although the cardboard holders effectively hold the objects in place, cardboard would not prevent objects made of sensitive materials, such as glass, from breaking. Businesses that produce glass objects would still have to use package peanuts and bubble wrap. Therefore, in future studies, I would like to test what material would maximize security for delicate materials.

While this project aims to create shipping packages on a commercial scale, it can also assist a variety of professions to organize their workspace. Because the algorithms implemented to produce the holder sustains a variety of shapes and sizes, it can be applied to hold not only tools but other common, everyday objects as well.

In conclusion, my project successfully developed a prototype tool holder that is fast and simple to produce, recyclable, and easy to store away. By processing an image's pixels and forming an outline made up of line segments, a raster image can be converted to a vector image. This can be achieved by taking a photo, eliminating image distortion, converting the image to black and white, dilating the image, and finally, creating an outline. This vector image can then be sent to a laser cutter to create a holder out of cardboard that will securely hold a set of tools.

Businesses can use this holder to firmly secure their products and prevent breakage during delivery. In an age where the effort to decrease the human ecological footprint on the environment is ever growing, the cardboard holders provide an excellent alternative to package peanuts and bubble wrap.

Acknowledgements

I would like to thank the following individuals for assisting me throughout the course of developing this project.

Professor Daniele Panozzo was incredibly kind and welcoming and I am incredibly lucky to have had him as my mentor and have worked with him for the past five weeks. He was always excited to see the work that I've accomplished and that always made me even more excited to work on the project. I would also like to thank the post-doc who helped me throughout the project, Teseo Schneider. Not only is Teseo an expert debugger and fantastic teacher, he is also an awesome and funny person whom I had the greatest pleasure working with. I could not have completed this project without Teseo. Lastly, thank you to Francisca Gil Ureta for assisting me with the report and for helping us understand how to use the laser cutter. It was very rewarding to see and talk to a woman working in computer science and accomplishing great things.

Thank you to Catherine Tissot and Monica Parham, the ultimate power duo, for instantly resolving each and every minor concern that surfaced during my time at GSTEM. Additionally, I would like to recognize my tutor, Priyanka Bhadoriya, for encouraging and ensuring me that I could do it from the beginning of the project until the end. Her moral support was instrumental to my success. GSTEM has been an unforgettable experience that I am incredibly thankful to have participated in.

Finally, I would also like to thank my twin sister, Sarah Shamash, for proofreading and correcting my grammar.

References

Adobe (1992), Adobe Streamline User Guide (version 3 for Windows ed.), Mountain View, CA: Adobe Systems.

Cressie, N., & Serra, J. (1988). *Image analysis and mathematical morphology. Theoretical advances*. New York: Academic Press.

Kaehler, A., & Bradski, G. R. (2017). *Learning OpenCV 3: Computer vision in C with the OpenCV library*. Sebastopol, CA: OReilly.

Pepper, K. N. (2005), "Cartooning with CorelDRAW", in Corel, CorelDRAW Handbook: Insights from the Experts, CorelDRAW X3 Graphics Suite, Corel Corporation, pp. 64–77.