

Fall 2017

## Thinking Outside The Box: Computing 3D Volume in 2D

Alexandra D. Morris  
Bard College, am5246@bard.edu

Follow this and additional works at: [https://digitalcommons.bard.edu/senproj\\_f2017](https://digitalcommons.bard.edu/senproj_f2017)



Part of the [Graphics and Human Computer Interfaces Commons](#)



This work is licensed under a [Creative Commons Attribution-NonCommercial-No Derivative Works 4.0 License](#).

---

### Recommended Citation

Morris, Alexandra D., "Thinking Outside The Box: Computing 3D Volume in 2D" (2017). *Senior Projects Fall 2017*. 28.

[https://digitalcommons.bard.edu/senproj\\_f2017/28](https://digitalcommons.bard.edu/senproj_f2017/28)

This Open Access work is protected by copyright and/or related rights. It has been provided to you by Bard College's Stevenson Library with permission from the rights-holder(s). You are free to use this work in any way that is permitted by the copyright and related rights. For other uses you need to obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/or on the work itself. For more information, please contact [digitalcommons@bard.edu](mailto:digitalcommons@bard.edu).

# Thinking Outside The Box: Computing 3D Volume in 2D

A Senior Project submitted to  
The Division of Science, Mathematics, and Computing  
of  
Bard College

by  
Alexandra Morris

Annandale-on-Hudson, New York  
December, 2017



# Abstract

This project explores how to compute 3D volume of cardboard boxes in 2D without a calibrated camera. Computer vision techniques to obtain 3D volume typically require camera calibration, the standard method for mapping 3D points to 2D. We created our own solution that doesn't rely on camera calibration and obtains the areas of each box with unknown dimensions with the help of chessboard pattern placed on each box side. The solution is a proportion that given the box area in pixels, chessboard pattern in pixels, and the chessboard pattern in inches, determines the box area in inches. We tested this method on 20 boxes, 5 pictures of each side for one box. The results showed positive feedback compared with the defined areas/ volumes and compared with the results of our Homographies. Ultimately we determined that our solution has the potential, with improved photos, test methods, etc. to accurately find an unknown box's volume given only the provided 2D data.





# Contents

<b>Abstract</b>	<b>iii</b>
<b>Dedication</b>	<b>vii</b>
<b>Acknowledgments</b>	<b>ix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
<b>2 Background</b>	<b>3</b>
2.1 Pin-Hole Cameras . . . . .	3
2.1.1 Homogeneous Coordinates . . . . .	3
2.1.2 Transformations . . . . .	4
2.1.3 Camera Calibration . . . . .	6
2.1.4 Homographies . . . . .	7
2.2 Resources . . . . .	7
2.2.1 OpenCV . . . . .	7
<b>3 Methods and Execution</b>	<b>9</b>
3.0.1 Proposed Algorithm for finding area . . . . .	10
3.0.2 Homographies . . . . .	15
<b>4 Results</b>	<b>17</b>
4.0.1 Algorithmic Results . . . . .	17
4.0.2 Homography Results . . . . .	28
<b>5 Discussion</b>	<b>33</b>
5.1 Challenges/ Limitations . . . . .	33
5.2 Conclusions . . . . .	36
<b>6 Future Directions</b>	<b>39</b>

<b>7 Appendix</b>	<b>41</b>
<b>Bibliography</b>	<b>61</b>

# Dedication

I would like to dedicate my research to my parents who have relentlessly cultivated my curiosity and guided me to be disciplined in order to achieve my goals. In addition I dedicate my project to my sister Avery, who is the best sister that anyone could ask for. In addition, I thank my Grandma, Aunt Frannie and Uncle Richard, for their unrelenting generosity and encouragement. Without them, this project might not have happened.



# Acknowledgments

I thank and acknowledge Keith for guiding me along the way and initiating and cultivating my fascination with computer vision. I am indebted to his support, in addition to the support from the other computer science faculty, Sven and Becky who similarly have been profound inspirations in my life.



# 1

## Introduction

### 1.1 Motivation

You are taping up boxes of winter gear to put away in storage. In anticipation of having more than the average amount of boxes, you call the storage facility and request a large unit. Now that the boxes are all packed up, you load them into the car, and drive to the storage unit. After unloading and placing the boxes in the unit, you realize you've made a terrible mistake; you've rented out a unit twice the size of all your boxes. Precision in the process of moving, whether it involves keeping track of your belongings in each box or knowing the total amount of goods you are storing, is a field that lags behind in fulfilling modern technology's potential. At the end of each school year, Bard students face this dilemma with regards to moving. Bard College requires that each student move his/her belongings out of his/her dorm for the summer. Each year, students become over sized storage unit victims and overpay for something that should be so simple. Through this study, we investigate how to obtain a box's volume (with unknown dimensions) without using camera calibration, which is the standard method for mapping 3D points to 2D.[2] Not only is obtaining box size data important for individuals planning on moving or renting a storage unit, but imagine the negative financial repercussions of a UPS employee who isn't sure how many boxes he/she can fit in his truck? Envision a company that manages a large amount of inventory overpaying for an over sized space thus setting them back



financially? In addition, according to the United States Census Bureau, young adults have the highest migration rate compared to any other generation and account for 43% of movers[7]. Due to this statistic, the Millennial generation, which relies heavily on smart phone assistance would benefit greatly from a program to detect box sizes in order to improve their moving experiences. Gaining 3D data (without relying on camera calibration) for any box size is not only a fascinating computer vision problem, but if well executed, there could be an economic demand to solve this problem for the Millennial generation. In this project, we ask: Can we accurately obtain a box's volume of unknown dimensions using an uncalibrated camera? We created our own solution that doesn't rely on camera calibration and obtains the areas of each box of unknown dimensions with the help of chessboard pattern placed on each box side. The solution is a proportion that given the box area in pixels, chessboard pattern in pixels, and the chessboard pattern in inches, determines the box area in inches. We tested this method on 20 boxes, 5 pictures of each side for one box and obtained positive results that were similar to our defined data.

# 2

## Background

### 2.1 Pin-Hole Cameras

#### 2.1.1 Homogeneous Coordinates

Homogeneous coordinates are essentially cartesian coordinates  $[x, y]$  with the addition of another parameter:  $w$ , or the scalar.

$$\tilde{v} = \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Homogeneous coordinates are widespread within computer vision as they enable transformations such as translation, rotation, scaling and perspective projection to be represented as a matrix by which the homogeneous coordinate vector is multiplied. Homogeneous coordinates can be used for both 2D and 3D points making them incredibly versatile[1].

$$\tilde{v} = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

In addition, it is very simple to convert homogeneous coordinates to non-homogeneous coordinates. All you need to do is divide all parameters by the scalar,  $w$ [1].

$$\tilde{v} = \begin{bmatrix} x \\ y \\ w \end{bmatrix} \rightarrow \begin{bmatrix} x/w \\ y/w \end{bmatrix} = v$$

If you have a non-homogeneous point and want it to be homogeneous, all you need to do is add the additional scale parameter, and normalize it making  $w = 1$ . [1]

$$v = \begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \tilde{v}$$

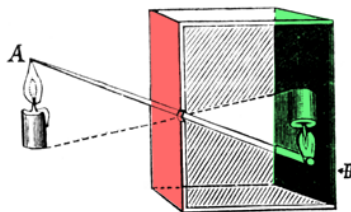
### 2.1.2 Transformations

Homogeneous coordinates are essential to completing nearly all computer vision geometric transformations that account for perspective. But first, it is necessary to understand the pin-hole camera model.

Much of computer vision assumes the pin-hole camera model (also known as camera obscura). Solem writes about the pin-hole camera in **Programming Computer Vision with Python**:

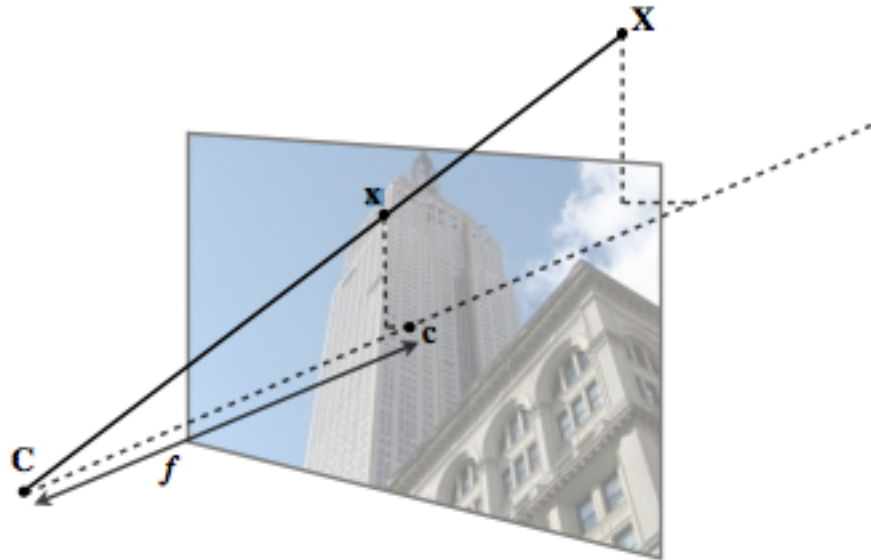
“The name comes from the type of camera, like a camera obscura, that collects light through a small hole to the inside of a dark box or room. In the pin-hole camera model, light passes through a single point, the camera center, C before it is projected in front of the camera center. The image plane in an actual camera would be upside down behind the camera center, but the model is the same[2].”

A camera obscura is in its basic form a box with a hole on one side. Light passing through that hole forms an inverted image of the object on the opposite side of the box [10].



[10]






In the following camera obscura model, we define variables enabling a variety of calculations.



[2]

In the pin-hole camera model,  $C$  represents the camera center and  $f$  is the focal length or distance from the camera center to the image plane. The image point,  $c = [c_x, c_y]$  is where the optical axis intersects the image plane[2].  $X$  is the 3D point, and  $x$  is the 2D image point.

We can use the pin-hole camera model to perform transformations allowing us to manipulate images. With the additional help of homogeneous coordinates, we can perform translation, rotation, scaling, perspective projections, and calibrate cameras.

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \mathbf{I} &   & \mathbf{t} \end{bmatrix}_{2 \times 3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix} \mathbf{R} &   & \mathbf{t} \end{bmatrix}_{2 \times 3}$	3	lengths	
similarity	$\begin{bmatrix} s\mathbf{R} &   & \mathbf{t} \end{bmatrix}_{2 \times 3}$	4	angles	
affine	$\begin{bmatrix} \mathbf{A} \end{bmatrix}_{2 \times 3}$	6	parallelism	
projective	$\begin{bmatrix} \tilde{\mathbf{H}} \end{bmatrix}_{3 \times 3}$	8	straight lines	

[1] The calibration matrix is composed of translation and rotation transformations.

### 2.1.3 Camera Calibration

We can model the pin-hole camera by a matrix that maps 3D points to 2D[2]. We first find the camera matrix, which includes many of the same variables from the Pin-Hole camera model.

The intrinsics matrix ( $\mathbf{K}$ ) includes the focal length and the image point ( $\mathbf{c}$ ).

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Next the camera matrix ( $\mathbf{P}$ ) is obtained when the intrinsics matrix is multiplied by the extrinsics matrix  $[\mathbf{R} \mid \mathbf{t}]$ .

$$\mathbf{P} = \mathbf{K}[\mathbf{R} \mid \mathbf{t}]$$

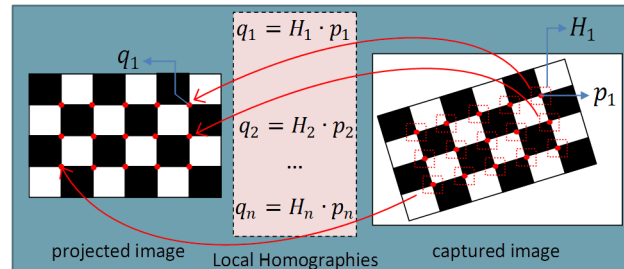
Finally, the camera matrix is multiplied by the 3D point ( $\mathbf{X}$ ) to determine the 2D point ( $\mathbf{x}$ ).

$$\mathbf{x} = \mathbf{P}\mathbf{X}$$

Camera calibration is the process of finding  $\mathbf{P}$  (the camera matrix) from 3D/2D corresponding coordinates, though for this study, it is unnecessary and we work only with 2D planes.

### 2.1.4 Homographies

Homographies are essentially projective transformations as indicated above. See figure 2.1.2 They are used to map points from one projective space to another[2]. The concept of mapping one projective space to another can be seen in this image.



[11]

We used the OpenCV library's function `getPerspectiveTransform()` in order to obtain our homography data.

## 2.2 Resources

### 2.2.1 OpenCV

We used the OpenCV library as the basis of our project. The OpenCV Library, or Open Computer Vision Library is an open source computer vision and machine learning library. It contains more than 2500 optimized algorithms ranging from face detection, to camera calibration, to augmented reality. This Library can be used in Python, C++, and Java[5]. Due to the simple interface, we decided to use Processing, a more user-friendly, graphically minded Java platform. We began using the data structures `PVectors` and `Mat`, and the OpenCV functions `findChessboardCorners()` and `getPerspectiveTransform()`.

`PVectors` are essentially 2 or 3 dimension vectors that take both magnitude and direction as inputs. The `PVector` class has convenient functions that allow obtaining magnitude, direction, multiplying or adding vectors simple. We incorporated `PVectors` in order to acquire both the box and chessboard areas, enabling us to accurately calculate the area of a box side in pixels.

Though we used `findChessboardCorners()` for a majority of the study, we ended up not using it as it presented too many bugs to enhance our program.

`findChessboardCorners()` takes a grid chessboard image and determines the 2D pixel locations of the corners.

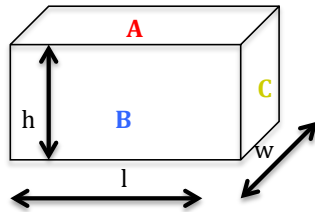




# 3

## Methods and Execution

### 3.0.1 Proposed Algorithm for finding area



**1. Load box side image.**

The box sides are categorized as the following:

$$\textit{Plane A} = l * w$$

$$\textit{Plane B} = l * h$$

$$\textit{Plane C} = w * h$$

Each image is identified as: BoxIdentifier\_PlaneType\_PictureNumber.jpg

**2. Enter the number of chessboard squares in pop-up window.**

The chessboard number only includes squares that have a clear boundary edge. The user clicks on the 4 corners (clockwise, starting from top left) that make up the perimeter of the chessboard region.



**3. The user clicks on the 4 corners of the box plane region, clockwise, starting from top left.**



**4. Compute the real area (in inches) of the chessboard region.**

$$chessRealArea = .828 * box\_num$$

.828 is the area of one single chessboard square.

**5. The areas (in pixels) of the box side and chessboard region are calculated**

**6. The real area (in inches) is found using the following proportion**

$$\frac{Chessboard\ region\ area\ (PX)}{Box\ Plane\ Region\ (PX)} = \frac{Chessboard\ region\ area\ (IN)}{Box\ Plane\ Region\ (IN)}$$

$$Box\ Plane\ Region\ (IN) = \frac{Box\ Plane\ Region\ (PX) * Chessboard\ region\ area\ (IN)}{Box\ Plane\ Region\ (PX)}$$

The user defines the chessboard region to have 24 squares as sectioned off in the above image (for the majority of the images). The corners of the box side region are shown in blue. Our area finding algorithm is repeated for each of the three planes.

To find the pixel areas, we needed to ensure that our algorithm was adaptable for any quadrilateral, as the pixel areas are not specifically defined quadrilaterals as they would be in real world areas. Given that we had 4 PVectors (each being one of the clicked corners), we found two parallel angles within the quadrilateral. Then, we extracted the lengths of all the sides, and calculated the following area:

$$area = .5 * (a * b) * \sin(ang\_AB) + .5 * (c * d) * \sin(ang\_CD)$$

$a, b, c, d$  are the distances between corners

$ang_{AB}$  = the angle in between distance a and b

$ang_{CD}$  = the angle in between distance c and d

While each side's area is being found, the length, width, and height of the box (in inches) are not able to be obtained as our ratio only relates area, not side measurements. Thus, we needed a way to determine the volume of a box with only the three given areas. Using substitution, we found a succinct formula for the box's volume in inches.

Given that-

$$Plane A = l * w$$

$$Plane B = h * l$$

$$Plane C = h * w$$

from the formula,

$$V = l * w * h$$

the volume of a box is

$$Volume = Plane B * \sqrt{\frac{Plane C * Plane A}{Plane B}}$$

By substitution:

$$V = l * w * h$$

$$V = B * w$$

and,

$$A = l * w$$

$$l = \frac{A}{w}$$

Then,

$$B = h * l$$

$$B = h * \frac{A}{w}$$

$$\frac{w}{A} * B = h * \frac{A}{w} * \frac{w}{A}$$

$$\frac{w}{A} * B = h$$

Similarly,

$$C = h * w$$

$$h = \frac{C}{w}$$

Next,

$$\frac{w}{A} * B = h$$

$$\frac{w}{A} * B = \frac{C}{w}$$

$$\frac{A}{w} * \frac{w}{A} * B = \frac{C}{w} * \frac{A}{w}$$

$$B = \frac{C * A}{w^2}$$

$$\frac{1}{C * A} * B = \frac{C * A}{w^2} * \frac{1}{C * A}$$

$$\frac{B}{C * A} = \frac{1}{w^2}$$

$$B * w^2 = C * A$$

$$\frac{B * w^2}{B} = \frac{C * A}{B}$$

$$\sqrt{w^2} = \sqrt{\frac{C * A}{B}}$$

$$w = \sqrt{\frac{C * A}{B}}$$

Finally,

$$V = l * w * h$$

$$V = B * w$$

$$Volume = Plane B * \sqrt{\frac{Plane C * Plane A}{Plane B}}$$

### 3.0.2 Homographies

The homography method used to verify our algorithm has two components. First, we find the homography that maps the chessboard pixel corners to the chessboard inch corners. Since the chessboard squares each have a pre-measured side length, we could easily obtain the chessboard coordinates in inches.

The built in OpenCV homography function called `getPerspectiveTransform()` accepts the 4 outer chessboard pixel coordinates as matrices. This function takes two matrices the source and the destination. It returns the 3x3 homography matrix. Still using homogeneous coordinates, we need to relate the resulting homography ( $H$ ) to the box area pixel corners. In order to do this, we take our 3x4 matrix of all the box area pixel corners and use matrix multiplication to merge it with our  $H$ . Finally we convert the resulting 3x4 matrix to non-homogeneous coordinates and obtain the real area of each side in inches.

The homography algorithm to find the real areas (in inches) works as follows:

1. We input the total number of chessboard squares, and pre-define the chessboard square length thus the chessboard area is determined.
2. The user inputs:
  - the chessboard area pixel coordinates by clicking on the corners.
  - the box area pixel coordinates by also selecting the corners.
3. The OpenCV function `getPerspectiveTransform()` is called and returns the homography matrix.
4. The homography matrix is multiplied by the box area pixel coordinates to obtain the real area in inches.

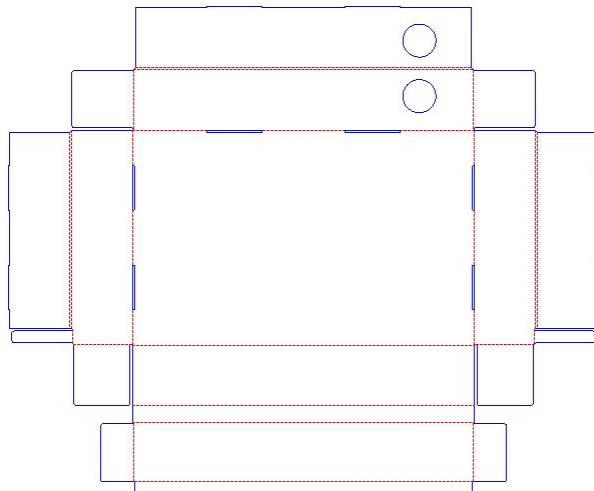


# 4

## Results

### *4.0.1 Algorithmic Results*

For the majority of our tests, we used International Paper boxes, a box company that Amazon commonly uses to ship their products. International Paper boxes were a convenient option as each box had a labeled identifier (e.g 1A7 or 1AD) and the dimensions of these boxes were easy to obtain online [9]. Soon after evaluating the pre-measured International Paper box dimensions, we determined that the length, width, and height of the boxes are measured when the box is flat packed. Thus for the other boxes which we measured by hand, we made the necessary adjustments to have the measurements consistent with International Paper’s measurements.





A Flat-packed box.

[12]



**Box Size Range: 160 inches<sup>3</sup>**

**to**

**6,672.75 inches<sup>3</sup>**

[13]

The equipment used to obtain our results included an iPhone camera, 20 brown cardboard boxes (International Paper Brand, Target, Lowes, etc.), a printed copy of OpenCV chessboard pattern, packing tape, and a gray backdrop (which provided enough contrast to the brown boxes) [4] [9].

For our volume algorithm tests, we had the following assumptions:

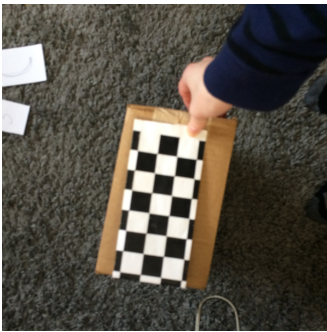
1. There are chessboard squares in a grid on each box surface.
2. The box surface is flat and parallel to the chessboard square sheet of paper.
3. All box side corners are visible.

These are the following types of images in our dataset:

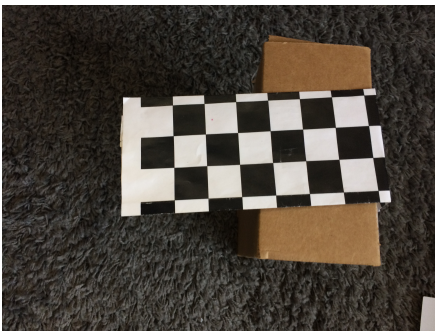
1. The image plane is parallel to the 3D points and there are 24 chessboard squares counted.



2. The image plane is parallel to 3D point and less chessboard squares are on the box sides.



3. The smaller box surface and chessboard square pattern extend beyond the side's area.



## 4. 24 square image with distortion



## 5. The image contains less than 24 squares and the image is distorted.



The boxes we tested ranged in volume between  $160in^3$  to  $6672.75in^3$ . We tested our algorithm on 20 boxes, 5 images per box side, thus 15 images per box. The results and data extracted from these tests are listed below.

Table 4.0.1. Volume Comparisons

Box	Avg Volume	Defined Volume	Volume Avg Error
A3 or BY1	383.07	367.50	4.24%
Shar	153.15	224.86	31.89%
int paper 1AD BNA	280.70	448.88	37.47%
LLBean	330.77	550.00	39.86%
Target model 439 sm	1385.50	1653.75	16.22%
N3 or B41	617.94	1023.75	39.64%
CVS box	2383.09	3655.39	34.81%
Lowes Box	1824.99	2304.00	20.79%
Int Paper 1A5 B45	596.10	705.38	15.49%
int paper 0A0	94.61	135.28	30.06%
Int paper A4	550.63	714.00	22.88%
Int paper 1A7	702.78	841.00	16.44%
int paper BP0	122.75	160.00	23.28%
int paper 2AA	1356.58	1632.00	16.88%
int paper 2A0	1873.55	2240.00	16.36%
int paper BP1	226.97	283.22	19.86%
int paper 2A5	785.14	935.00	16.03%
int paper 2A8	3822.86	4446.00	14.02%
int paper 3A1	6652.82	6672.75	0.30%
int paper B0	1820.16	2057.00	11.51%

Table 4.0.2. Area Average Compared With True Areas

Box	length	width	height	Defined Area (A)	Defined Area (B)	Defined Area (C)
A3 or BY1	10	7	5.25	70.00	52.5	36.75
Shar	10.25	6.75	3.25	69.19	33.3125	21.9375
int paper 1AD BNA	13.5	9.5	3.5	128.25	47.25	33.25
LLBean	13.75	8	5	110.00	68.75	40
Target model 439 sm	17.5	13.5	7	236.25	122.5	94.5
N3 or B41	16.25	12	5.25	195.00	85.3125	63
CVS box	21.65	13.4	12.6	290.11	272.79	168.84
Lowes Box	16	12	12	192.00	192	144
Int Paper 1A5 B45	13.5	11	4.75	148.50	64.125	52.25
int paper 0A0	9.25	6.5	2.25	60.13	20.8125	14.625
Int paper A4	12	8.5	7	102.00	84	59.5
Int paper 1A7	14.5	8	7.25	116.00	105.125	58
int paper BP0	10	8	2	80.00	20	16
int paper 2AA	24	16	4.25	384.00	102	68
int paper 2A0	20	16	7	320.00	140	112
int paper BP1	13.25	9.5	2.25	125.88	29.8125	21.375
int paper 2A5	20	11	4.25	220.00	85	46.75
int paper 2A8	26	19	9	494.00	234	171
int paper 3A1	31	20.5	10.5	635.50	325.5	215.25
int paper B0	17	11	11	187.00	187	121

Table 4.0.3. Algorithmic Results

	(A) Min Error	(B) Min error	(C) Min Error	(A) Max Error	(B) Max Error	(C) Max Error
Box						
A3 or BY1	8.05%	0.40%	5.53%	13.29%	5.45%	34.87%
Shar	0.50%	9.09%	6.38%	4.05%	15.17%	18.97%
int paper 1AD BNA	0.89%	10.12%	1.14%	16.21%	14.74%	9.25%
LLBean	0.73%	7.29%	2.70%	8.44%	16.20%	12.65%
Target model 439 sm	0.18%	17.98%	9.13%	10.82%	36.59%	23.82%
N3 or B41	0.77%	0.28%	0.84%	4.47%	13.49%	43.42%
CVS box	0.40%	1.36%	3.04%	7.12%	9.17%	9.93%
Lowes Box	6.09%	6.66%	10.56%	10.70%	10.70%	21.04%
Int Paper 1A5 B45	9.81%	16.32%	8.61%	16.16%	23.16%	16.83%
int paper 0A0	0.67%	0.91%	3.06%	13.23%	7.41%	14.75%
Int paper A4	11.28%	1.13%	6.80%	13.80%	11.88%	12.98%
Int paper 1A7	18.18%	6.98%	10.18%	23.43%	10.00%	17.30%
int paper BP0	10.00%	1.10%	0.51%	19.68%	4.68%	20.29%
int paper 2AA	13.97%	7.51%	13.20%	15.96%	12.93%	21.32%
int paper 2A0	15.68%	8.97%	13.50%	17.04%	14.20%	18.89%
int paper BP1	13.09%	4.33%	2.54%	18.72%	12.20%	20.64%
int paper 2A5	11.21%	11.23%	10.68%	14.35%	20.39%	21.69%
int paper 2A8	13.05%	12.96%	15.75%	15.66%	19.58%	22.09%
int paper 3A1	13.74%	50.64%	18.68%	17.77%	55.36%	23.15%
int paper B0	11.02%	18.78%	20.50%	19.35%	21.36%	24.77%

Table 4.0.4. Algorithmic Results

Box	(A) Avg Error	(B) Avg Error	(C) Avg Error
A3 or BY1	4.42%	2.36%	16.43%
Shar	0.83%	10.70%	9.29%
int paper 1AD BNA	5.09%	12.77%	1.40%
LLBean	3.82%	12.13%	8.14%
Target model 439 sm	1.81%	24.95%	18.44%
N3 or B41	2.41%	4.51%	16.08%
CVS box	2.73%	0.57%	5.67%
Lowes Box	8.25%	7.91%	15.30%
Int Paper 1A5 B45	13.58%	19.87%	12.61%
int paper 0A0	5.18%	1.72%	8.86%
Int paper A4	12.54%	4.27%	8.81%
Int paper 1A7	21.31%	8.57%	13.81%
int paper BP0	13.92%	0.16%	10.73%
int paper 2AA	14.86%	10.84%	16.51%
int paper 2A0	16.43%	11.29%	15.89%
int paper BP1	15.76%	8.32%	9.95%
int paper 2A5	12.63%	15.09%	16.78%
int paper 2A8	14.48%	16.39%	19.12%
int paper 3A1	15.51%	52.24%	21.34%
int paper B0	14.58%	19.80%	22.46%

Table 4.0.5. Algorithmic Results

Box	Min A	Max A	Min B	Max B	Min C	Max C
A3 or BY1	63.49	79.30	49.64	53.81	38.78	49.57
Shar	66.38	71.52	36.34	38.36	17.78	23.40
int paper 1AD BNA	107.46	130.52	40.29	42.47	32.35	36.33
LLBean	100.72	109.20	57.61	63.74	34.94	38.92
Target model 439 sm	223.35	261.82	144.53	167.32	103.13	117.01
N3 or B41	186.28	196.50	74.32	96.82	35.64	65.29
CVS box	269.46	304.77	256.04	297.82	152.08	163.71
Lowes Box	203.69	212.54	204.79	212.55	159.21	174.29
Int Paper 1A5 B45	163.07	172.50	74.59	78.97	56.75	61.04
int paper 0A0	52.17	60.53	20.46	22.36	14.18	16.78
Int paper A4	113.51	116.07	82.99	93.98	63.55	67.22
Int paper 1A7	137.09	143.18	112.46	115.64	63.91	68.03
int paper BP0	88.00	95.74	19.34	20.94	15.92	19.25
int paper 2AA	437.66	445.27	109.66	115.18	76.97	82.50
int paper 2A0	370.17	374.53	152.56	159.88	127.12	133.15
int paper BP1	142.35	149.43	31.10	33.45	21.92	25.79
int paper 2A5	244.66	251.56	94.54	102.34	51.74	56.89
int paper 2A8	558.48	571.37	264.33	279.83	197.94	208.77
int paper 3A1	722.83	748.46	490.33	505.69	255.46	265.08
int paper B0	207.60	223.18	222.13	226.94	145.81	150.97

By manually entering the total number of chessboard squares, we guarantee that in each image, the chessboard squares are arranged in a quadrilateral shape. After we obtained the areas for each side of every box, we used the following calculations to help evaluate our algorithm's accuracy:

1.

$$\text{Error For Box Plane} = \frac{|\text{Defined Area} - \text{Test Result Area}|}{\text{Defined Area}}$$

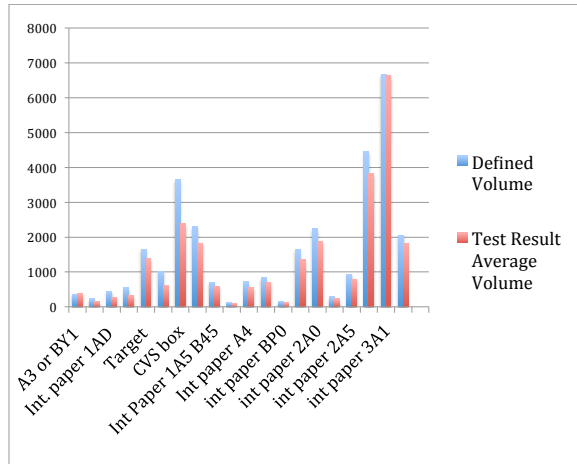
2.

$$\text{Volume} = \text{Plane B avg} * \sqrt{\frac{\text{Plane C Avg} * \text{Plane A Avg}}{\text{Plane B Avg}}}$$

3.

$$\text{Volume Average Error} = \frac{|\text{Defined Volume} - \text{Test Result Volume}|}{\text{Defined Volume}}$$





Our results can also be viewed in the below column chart for easy comprehension. Our algorithm worked for the distorted and undistorted images. This result along with successful tests gave our area obtaining ratio validity.



Box A3 Distorted Image:  
64.36 square inches



Box A3 Undistorted Image:  
63.73 square inches



Box A3 Distorted Image:  
49.64 square inches



Box A3 Undistorted Image:  
49.84 square inches



10.25 x 6.75 x 3.25 Distorted Image:  
130.52 square inches



10.25 x 6.75 x 3.25 Undistorted Image:  
129.39 square inches



Target Box Distorted Image:  
261.82 inches



Target Box Undistorted Image:  
236.67 square inches



Box A4 Distorted Image:  
76.57 square inches



Box A4 Undistorted Image:  
74.59 square inches

#### 4.0.2 Homography Results

As discussed previously, we decided to use homographies to use as evidence that our proposed algorithm produces similar results. The homography results were very successful. Though we didn't test the homography program on all of our box images, the results are pretty much spot on with the pre-defined box areas.

Table 4.0.6. Plane A: True Values vs. Our Results vs. Homography

Box	Our Test Result Avg (Plane A)	Plane A Defined Area	Plane A Area: Test 1	Plane A Area: Test 2
A3 or BY1	66.91	70.00	53.78	64.86
Target model 439 sm	240.53	236.25	201.73	192.10

Table 4.0.7. Plane B: True Values vs. Our Results vs. Homography

Box	Our Test Results (Plane B)	Plane B Defined Area	Plane B Area: Test 1	Plane B Area: Test 2
A3 or BY1	51.26	52.5	39.69	41.37
Target model 439 sm	153.07	122.50	120.77	123.28

Table 4.0.8. Plane C: True Values vs. Our Results vs. Homography

Box	Our Test Results Avg (Plane C)	Plane C Defined Area	Plane C Area: Test 1	Plane C Area: Test 2
A3 or BY1	42.79	36.75	33.49	35.33
Target model 439 sm	111.92	94.50	94.54	93.48

In this diagram, the areas for the first box, plane A are not as accurate due to the fact that the areas in the images are not exactly flat as we were not consistent in taping down the box flaps. Though this doesn't necessarily justify our algorithm as being a method to find the volume of any box, the similar feedback between the homography and our proposed algorithm results is evidence that our method has the potential to be valid.



# 5

## Discussion

### 5.1 Challenges/ Limitations

Over the course of this project, we faced a fair number of challenges and limitations that prevented certain aspects of the project from becoming reality.

The first significant hurdle we encountered was the OpenCV `findChessboardCorners()` function. Though initially we aimed to use this function to automatically determine the number of small chessboard squares in each image, the function had too many problems in actually detecting the corners, that using it would make the results faulty. This function returned three types of erroneous feedback:

- a. Detected no corners

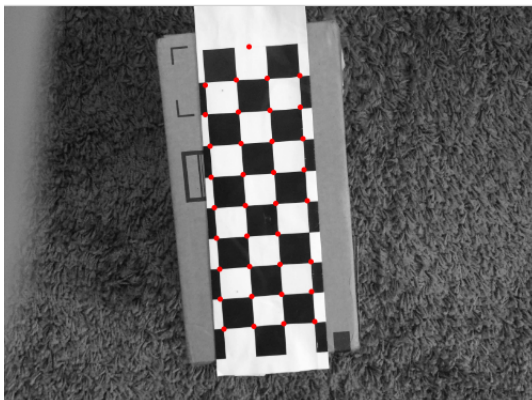




b. Detected too few corners



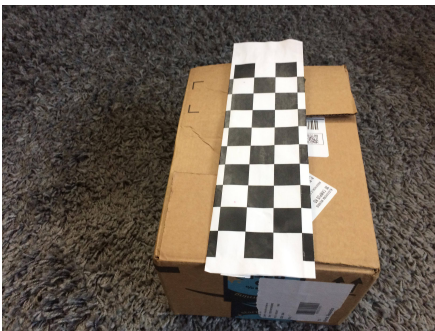
c. Detected too many corners and/ or in the wrong locations



Due to the temperamental nature of the `findChessboardCorners()` function, we resorted in having the user click on the outer four chessboard corners as well as the outer box area corners. This solution was not ideal but due to the time constriction, was necessary.

Relying on the user to click the outer box and chessboard region corners showcased other limitations in our program. Not only does having the user click on the corners increase the likelihood of inaccuracies, but we learned that there are likely discrepancies between the initial box measurements (determined when the boxes are flat-packed in 2D) and when they are measured after being constructed in 3D. When a box is assembled, often the length, width, and height are amplified because of how a box is folded. Since the user clicks on the corners while the box is in 3D, this might have caused discrepancies in the outcome.

In addition, the data set of images could have been more precise. In many of the photos, the box flaps were not properly taped down so the chessboard pattern did not lay flat on the box side. This inconsistency made it so in many images the box sides weren't actually flat surfaces, altering the accuracy of the outcome.



Box flaps are not properly taped

Occasionally not all the box corners were displayed in the images. If we had access to a larger backdrop, it would have been easier to incorporate all the box corners into every frame. Also,

with a larger backdrop, we would have been able to vary our test results more and use larger boxes, extending the distance between image points and the 3D points.



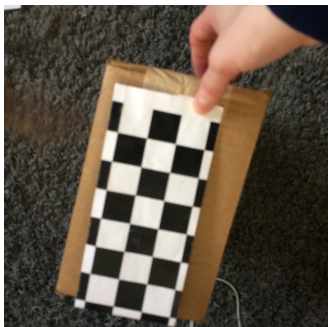
Not all box corners are visible

In evaluating this project as a whole, it could have been more productive to control the angles and amounts of image distortion to obtain more consistent feedback.

Also, we could have ensured more accuracy if we used all International Paper boxes that were pre-measured or only used boxes that were hand measured [9]. This would have guaranteed more consistency in our data.

## 5.2 Conclusions

In conclusion, our results were more successful when images included more chessboard squares. This is an example of an unsuccessful image, that contains too few chessboard squares resulting in a less accurate volume calculation.



In addition, the fact that the homography results were similar to our algorithm's results suggests that our algorithm has the potential to be a valid way to determine a box's volume without the use of camera calibration.



# 6

## Future Directions

In terms of future pursuits, we can greatly improve this project in the following ways.

Incorporating the use of Harris corners could work better than the `findChessboardCorners()` function to automate the process of finding the number of squares. Also, we could fully implement line detection to automatically detect the box areas' boundaries. It would be beneficial to explore the optimal amount of chessboard squares for a box surface. From a few experiments, we have found that the more chessboard squares on a box side, the more accurate the area calculation is of that surface.



In this example, the box side has a chessboard duct tape pattern on it. There are 246 small

squares on this surface. The area of one small square is .0977 inches. When the area is calculated, the result is 134.14 square inches. Thus the error is only 9.7% which is less than our average error for this side which was 14%. Thus, this method of including more chessboard squares per surface shows preliminary potential. In addition, if we were to address the problem of millennials needing to know the total volume of all their boxes quickly, a more practical solution may be a mobile application platform for a future project.

# 7

## Appendix

Finding Volume Algorithm Code

```
import Jama.*;
import gab.opencv.*;
import static javax.swing.JOptionPane.*;
import java.io.PrintWriter;
import java.io.BufferedReader;

float chessPixelArea;
ArrayList<PVector> cornerPoints;
OpenCV opencv;
int np1 = 0;
PVector p1[] = new PVector[8];
int wid = 4;
int leng = 9;
int npMAX = 8;
```



```
int num = 1;
int box_num;
PImage pix;

void setup() {
  background(255);
  smooth();

  pix = loadImage("box_6/side_A/6_A_2.jpg");

  pix.resize(500, 0);
  size(pix.width, pix.height);

  opencv = new OpenCV(this, pix);
  opencv.gray();

  cornerPoints = opencv.findChessboardCorners(leng, wid);
}

void mousePressed() {
  if (mouseX < width && mouseY < height) {
    p1[np1] = new PVector(mouseX, mouseY, 1);

    np1++;
  }
}
```

```
float calcPixelAreaREAL(PVector v1, PVector v2, PVector p, PVector p2) {

    PVector v1_copy = new PVector(v1.x, v1.y);
    PVector v2_copy = new PVector(v2.x, v2.y);

    float a = v2.dist(p);
    float b = v1.dist(p);
    float c = p2.dist(v1);
    float d = p2.dist(v2); /

    v1.sub(p);
    v2.sub(p);
    float p_ang = PVector.angleBetween(v1, v2);

    v1_copy.sub(p2);
    v2_copy.sub(p2);

    float p2_ang = PVector.angleBetween(v1_copy, v2_copy);

    float areaa = .5*(a*b)*sin(p_ang) + .5*(c*d)*sin(p2_ang);
    return areaa;
}

void draw() {

    image(opencv.getOutput(), 0, 0);
```

```
noStroke();

float chessRealArea = .828*box_num;

for (PVector p : cornerPoints) {
    fill(255, 0, 0);
    ellipse(p.x, p.y, 5, 5);
}

if (np1 >= 8 && npMAX >=8) {

    loadPixels();

    updatePixels();
}

for (int i = 0; i < np1; i++) {

    fill(0, 0, 255);
    ellipse( p1[i].x, p1[i].y, 20, 20);

    if (i==7) {

        PVector pv1 = new PVector (p1[0].x, p1[0].y);
        PVector v1 = new PVector (p1[1].x, p1[1].y);
        PVector pv2 = new PVector (p1[2].x, p1[2].y);
        PVector v2 = new PVector (p1[3].x, p1[3].y);

        PVector pp1 = new PVector(p1[4].x, p1[4].y);
```

```

PVector vv1 = new PVector(p1[5].x, p1[5].y);
PVector pp2 = new PVector(p1[6].x, p1[6].y);
PVector vv2 = new PVector(p1[7].x, p1[7].y);

chessPixelArea = calcPixelAreaREAL(v1, v2, pv1, pv2);
float boxPixelArea = calcPixelAreaREAL(vv1, vv2, pp1, pp2);

println("area of box Pixel is " + boxPixelArea);
println("area of chessPixel is " + chessPixelArea);

println("box number is " + box_num);
println("area of box plane is " + (boxPixelArea*chessRealArea)/chessPixelArea + " inch

try {
    File f = dataFile("results_7.txt");
    PrintWriter out = new PrintWriter(new BufferedWriter(new FileWriter(f, true)));
    out.println((boxPixelArea*chessRealArea)/chessPixelArea);
    out.flush();
    out.close();
}
catch (IOException e) {
    println(e);
}

output.println("Test 5: Area of box plane is " + ((boxPixelArea*chessRealArea)/chessPixelArea));
output.flush();

to the file

    output.close();

```

```
        exit();
    }
}

..

import gab.opencv.*;
import org.opencv.imgproc.Imgproc;
import org.opencv.core.MatOfPoint2f;
import org.opencv.core.Point;
import org.opencv.core.Size;

import org.opencv.core.Mat;
import org.opencv.core.CvType;

OpenCV opencv;
PImage src;
float cardWidth = 3*.828;
float cardHeight = 8*.828;
int np1 = 0;
ArrayList<PVector> points = new ArrayList(4);

void setup() {
    src = loadImage("5_C_5.jpg");
```

```
src.resize(500, 500);
size(src.width, src.height);
opencv = new OpenCV(this, src);
}

void mousePressed() {
    if (mouseX < width && mouseY < height) {
        points.add(new PVector(mouseX, mouseY));
        println(mouseX, mouseY);
        np1++;
    }
}

Mat getPerspectiveTransformation(ArrayList<PVector> inputPoints, float w, float h) {
    Point[] canonicalPoints = new Point[4];
    canonicalPoints[0] = new Point(w, 0);
    canonicalPoints[1] = new Point(0, 0);
    canonicalPoints[2] = new Point(0, h);
    canonicalPoints[3] = new Point(w, h);

    MatOfPoint2f canonicalMarker = new MatOfPoint2f();
    canonicalMarker.fromArray(canonicalPoints);

    Point[] pointsArr = new Point[4];
    for (int i = 0; i < 4; i++) {
```

```
    pointsArr[i] = new Point(points.get(i).x, points.get(i).y);
}

MatOfPoint2f marker = new MatOfPoint2f();
marker.fromArray(pointsArr);

return Imgproc.getPerspectiveTransform(marker, canonicalMarker);
}

void draw() {
    image(src, 0, 0);

    for (int i = 0; i < np1; i++) {

        fill(255, 0, 0);
        ellipse( points.get(i).x, points.get(i).y, 20, 20);

        if (i==3) {

            Mat transform = getPerspectiveTransformation(points, cardWidth, cardHeight);

            println(transform.dump());
            noLoop();
            exit();
        }
    }
}
```

```
    }  
  }  
}
```

```
    import gab.opencv.*;  
import org.opencv.imgproc.Imgproc;  
import org.opencv.core.MatOfPoint3f;  
import org.opencv.core.Point;  
import org.opencv.core.Size;  
import org.opencv.core.Point3;  
  
import org.opencv.core.Mat;  
import org.opencv.core.CvType;  
import org.opencv.core.Core;
```

```
OpenCV opencv;  
PImage src;  
float cardWidth = 3*.828;  
float cardHeight = 8*.828;  
int np1 = 0;  
PVector pv1;  
PVector v1;  
PVector pv2;  
PVector v2;
```



```
void setup() {  
    src = loadImage("5_C_5.jpg");  
    src.resize(500, 500);  
    size(src.width, src.height);  
    size(300, 300);  
    opencv = new OpenCV(this, src);  
  
    Point3[] canonicalPoints = new Point3[4]; //for transformed card. will be inch chessboard out  
    canonicalPoints[0] = new Point3(274, 151, 1); //counter clockwise!!!!!! start from top right  
    canonicalPoints[1] = new Point3(110, 164, 1);  
    canonicalPoints[2] = new Point3(139, 369, 1);  
    canonicalPoints[3] = new Point3(312, 346, 1);  
  
    Mat box_inch_pts = new Mat(3, 4, CvType.CV_32FC1);  
  
    Mat temp_H_inv = new Mat(3, 3, CvType.CV_32FC1);  
    Mat temp = new Mat();  
  
    temp_H_inv.put(0, 0, new double[] { //COPY HERE!!!!!!!!!!!!  
  
        0.05092402444659763, -0.005819888508182581, -7.062434704679569,  
        0.005086898655335161, 0.06994485651085894, -15.09282831037952,  
        0.0005803706726008797, 5.488099837932256e-05, 1  
    }  
}
```

```

);
//row,col
// println(temp_H_inv.dump());

box_inch_pts.put(0, 0, new double[] {
    canonicalPoints[0].x, canonicalPoints[1].x, canonicalPoints[2].x, canonicalPoints[3].x,
    canonicalPoints[0].y, canonicalPoints[1].y, canonicalPoints[2].y, canonicalPoints[3].y,
    canonicalPoints[0].z, canonicalPoints[1].z, canonicalPoints[2].z, canonicalPoints[3].z
});

//temp_H_inv.mul(box_inch_pts);
Core.gemm(temp_H_inv, box_inch_pts, 1, new Mat(), 0, temp);
// opencv.convertPointsFromHomogeneous(temp);
//make homogeneous !!!!!

//System.out.println("box_inch_pts =\n" + temp.dump());
// println((double)temp.get(0, 0)/(double)temp.get(2, 0));

Mat final_box_inch_pts = new Mat(3, 4, CvType.CV_32FC1);
/*
//x's in nonhomogeneous
println(temp.get(0, 0)[0]/temp.get(2, 0)[0]); //couner clockwise
println(temp.get(0, 1)[0]/temp.get(2, 1)[0]); //
println(temp.get(0, 2)[0]/temp.get(2, 2)[0]);
println(temp.get(0, 3)[0]/temp.get(2, 3)[0]);

```

```

println();

//y's in nonhomogeneous
println(temp.get(1, 0)[0]/temp.get(2, 0)[0]); //counterclockwise
println(temp.get(1, 1)[0]/temp.get(2, 1)[0]); //
println(temp.get(1, 2)[0]/temp.get(2, 2)[0]);
println(temp.get(1, 3)[0]/temp.get(2, 3)[0]);

*/

pv1 = new PVector ((float)(temp.get(0, 1)[0]/temp.get(2, 1)[0]), (float)(temp.get(1, 1)[0]/temp.get(2, 1)[0]));
v1 = new PVector (((float)temp.get(0, 0)[0]/(float)temp.get(2, 0)[0]), (float)(temp.get(1, 0)[0]/temp.get(2, 0)[0]));
pv2 = new PVector((float)(temp.get(0, 3)[0]/temp.get(2, 3)[0]), (float)(temp.get(1, 3)[0]/temp.get(2, 3)[0]));
v2 = new PVector ((float)(temp.get(0, 2)[0]/temp.get(2, 2)[0]), (float)(temp.get(1, 2)[0]/temp.get(2, 2)[0]));
/*
final_box_inch_pts.put(0, 0, new double[] {
    (float)temp.get(0, 0)/(float)temp.get(2, 0), temp.get(0, 1)/temp.get(2, 1), temp.get(0, 2)/temp.get(2, 2),
    temp.get(1, 0)/temp.get(2, 0), temp.get(1, 1)/temp.get(2, 1), temp.get(1, 2)/temp.get(2, 2),
    temp.get(2, 0)/temp.get(2, 0), temp.get(2, 1)/temp.get(2, 1), temp.get(2, 2)/temp.get(2, 2)
}); */

// println("final_box_inch_pts =\n" + final_box_inch_pts.dump());
}

```

```

float calcPixelAreaREAL(PVector v1, PVector v2, PVector p, PVector p2) {

    PVector v1_copy = new PVector(v1.x, v1.y);
    PVector v2_copy = new PVector(v2.x, v2.y);

    float a = v2.dist(p); //v2.y - p.y;
    float b = v1.dist(p); //v1.x - p.x;
    float c = p2.dist(v1); //p2.y-v1.y;
    float d = p2.dist(v2); ///p2.x - v2.x;

    v1.sub(p);
    v2.sub(p);
    float p_ang = PVector.angleBetween(v1, v2);
    //-----

    v1_copy.sub(p2);
    v2_copy.sub(p2);

    float p2_ang = PVector.angleBetween(v1_copy, v2_copy);
    //-----

    float areaa = .5*(a*b)*sin(p_ang) + .5*(c*d)*sin(p2_ang);
    //println("area is " + areaa);
    return areaa;

    //area should be 2437.5
}

```

```
void draw() {  
    float box_real_area = calcPixelAreaREAL(v1, v2, pv1, pv2);  
    println("box real area is " + box_real_area);  
    exit();  
}
```

Table 7.0.1. Algorithmic Results

Box	length	width	height	Defined Area (A)
A3 or BY1	10	7	5.25	70
Shar	10.25	6.75	3.25	69.19
int paper 1AD BNA	13.5	9.5	3.5	128.25
LLBean	13.75	8	5	110
Target model 439 sm	17.5	13.5	7	236.25
N3 or B41	16.25	12	5.25	195
CVS box	21.65	13.4	12.6	290.11
Lowes Box	16	12	12	192
Int Paper 1A5 B45	13.5	11	4.75	148.5
int paper 0A0	9.25	6.5	2.25	60.13
Int paper A4	12	8.5	7	102
Int paper 1A7	14.5	8	7.25	116
int paper BP0	10	8	2	80
int paper 2AA	24	16	4.25	384
int paper 2A0	20	16	7	320
int paper BP1	13.25	9.5	2.25	125.88
int paper 2A5	20	11	4.25	220
int paper 2A8	26	19	9	494
int paper 3A1	31	20.5	10.5	635.5
int paper B0	17	11	11	187

Table 7.0.2. Algorithmic Results

Box	Defined Area (B)	Defined Area (C)	test 1(A)	test 2 (A)	test 3 (A)	test 4 (A)	test 5 (A)
A3 or BY1	52.50	36.75	79.30	64.36	63.65	63.73	63.49
Shar	33.31	21.94	66.99	71.52	66.38	69.53	68.65
int paper 1AD BNA	47.25	33.25	114.11	107.46	129.39	130.52	127.10
LLBean	68.75	40.00	100.72	106.58	106.30	106.21	109.20
Target model 439 sm	122.50	94.50	236.67	223.35	247.95	232.86	261.82
N3 or B41	85.31	63.00	186.28	191.83	196.50	187.43	189.50
CVS box	272.79	168.84	304.77	288.95	274.52	269.46	273.30
Lowes Box	192.00	144.00	206.66	203.69	210.26	206.09	212.54
Int Papr 1A5 B45	64.13	52.25	167.55	168.69	171.49	172.50	163.07
int paper 0A0	20.81	14.63	52.17	53.74	58.90	60.53	59.70
Int paper A4	84.00	59.50	116.07	114.49	113.51	115.65	114.23
Int paper 1A7	105.13	58.00	142.20	143.18	141.02	140.13	137.09
int paper BP0	20.00	16.00	90.58	91.85	89.49	88.00	95.74
int paper 2AA	102.00	68.00	439.17	441.32	437.66	441.88	445.27
int paper 2A0	140.00	112.00	370.94	370.17	374.53	374.02	373.25
int paper BP1	29.81	21.38	148.78	145.03	142.35	149.43	142.98
int paper 2A5	85.00	46.75	245.68	245.55	244.66	251.56	251.45
int paper 2A8	234.00	171.00	565.59	564.16	558.48	568.03	571.37
int paper 3A1	325.50	215.25	742.18	748.46	725.30	722.83	731.71
int paper B0	187.00	121.00	215.34	213.22	211.93	207.60	223.18

Table 7.0.3. Algorithmic Results

Box	test 1(B)	test 2 (B)	test 3 (B)	test 4 (B)	test 5 (B)
A3 or BY1	50.29	49.64	49.84	52.71	53.81
Shar	36.34	36.35	38.36	36.55	36.78
int paper 1AD BNA	40.66	40.45	42.47	40.29	42.21
LLBean	62.11	58.39	63.74	57.61	60.22
Target model 439 sm	144.53	151.60	167.32	153.33	148.57
N3 or B41	85.07	75.23	96.82	74.32	75.90
CVS box	256.04	264.88	276.51	260.95	297.82
Lowes Box	206.95	212.55	206.06	204.79	205.55
Int Paper 1A5 B45	78.10	78.97	76.57	76.10	74.59
int paper 0A0	20.46	21.00	20.46	21.58	22.36
Int paper A4	87.01	84.95	82.99	89.00	93.98
Int paper 1A7	115.64	114.72	113.98	113.87	112.46
int paper BP0	19.78	19.49	20.61	19.34	20.94
int paper 2AA	112.80	113.98	115.18	109.66	113.65
int paper 2A0	155.24	157.30	154.08	152.56	159.88
int paper BP1	32.16	32.60	32.16	31.10	33.45
int paper 2A5	98.47	94.54	96.65	97.12	102.34
int paper 2A8	271.04	268.11	264.33	278.41	279.83
int paper 3A1	490.63	494.42	496.68	505.69	490.33
int paper B0	222.13	224.38	224.36	226.94	222.31

Table 7.0.4. Algorithmic Results

Box	test 1(C)	test 2 (C)	test 3 (C)	test 4 (C)	test 5 (C)
A3 or BY1	39.20	43.96	42.43	49.57	38.78
Shar	23.40	19.04	20.54	17.78	18.74
int paper 1AD BNA	32.55	33.63	36.33	33.72	32.35
LLBean	37.48	36.07	34.94	38.92	36.32
Target model 439 sm	103.13	113.92	117.01	115.50	110.06
N3 or B41	63.71	65.29	63.53	35.64	36.18
CVS box	160.66	163.71	160.35	152.08	159.56
Lowes Box	168.25	161.57	159.21	166.84	174.29
Int Paper 1A5 B45	58.32	59.75	61.04	56.75	58.33
int paper 0A0	16.43	14.18	15.45	16.76	16.78
Int paper A4	64.93	67.22	64.42	63.55	63.58
Int paper 1A7	63.91	66.11	67.08	68.03	64.93
int paper BP0	18.49	18.21	19.25	15.92	16.72
int paper 2AA	77.77	79.06	79.83	82.50	76.97
int paper 2A0	129.41	127.12	127.84	133.15	131.49
int paper BP1	22.64	23.35	21.92	23.81	25.79
int paper 2A5	51.91	51.74	55.73	56.71	56.89
int paper 2A8	208.77	197.94	203.72	201.28	206.76
int paper 3A1	256.71	255.46	265.08	264.22	264.44
int paper B0	147.43	148.27	150.97	145.81	148.38

Table 7.0.5. Algorithmic Results

	(A) Error Test 1	(A) Error Test 2	(A) Error Test 3	(A) Error Test 4	(A) Error Test 5
Box					
A3 or BY1	13.29%	8.05%	9.07%	8.96%	9.29%
Shar	3.18%	3.37%	4.05%	0.50%	0.77%
int paper 1AD BNA	11.02%	16.21%	0.89%	1.77%	0.89%
LLBean	8.44%	3.11%	3.36%	3.45%	0.73%
Target model 439 sm	0.18%	5.46%	4.95%	1.43%	10.82%
N3 or B41	4.47%	1.63%	0.77%	3.88%	2.82%
CVS box	5.05%	0.40%	5.37%	7.12%	5.80%
Lowes Box	7.63%	6.09%	9.51%	7.34%	10.70%
Int Paper 1A5 B45	12.83%	13.60%	15.48%	16.16%	9.81%
int paper 0A0	13.23%	10.62%	2.04%	0.67%	0.70%
Int paper A4	13.80%	12.24%	11.28%	13.38%	11.99%
Int paper 1A7	22.58%	23.43%	21.57%	20.80%	18.18%
int paper BP0	13.23%	14.81%	11.86%	10.00%	19.68%
int paper 2AA	14.37%	14.93%	13.97%	15.07%	15.96%
int paper 2A0	15.92%	15.68%	17.04%	16.88%	16.64%
int paper BP1	18.19%	15.22%	13.09%	18.72%	13.59%
int paper 2A5	11.67%	11.61%	11.21%	14.35%	14.29%
int paper 2A8	14.49%	14.20%	13.05%	14.98%	15.66%
int paper 3A1	16.79%	17.77%	14.13%	13.74%	15.14%
int paper B0	15.15%	14.02%	13.33%	11.02%	19.35%

Table 7.0.6. Algorithmic Results

	(B) Error Test 1	(B) Error Test 2	(B) Error Test 3	(B) Error Test 4	(B) Error Test 5
Box					
A3 or BY1	4.21%	5.45%	5.06%	0.40%	2.50%
Shar	9.09%	9.12%	15.17%	9.71%	10.40%
int paper 1AD BNA	13.95%	14.38%	10.12%	14.74%	10.67%
LLBean	9.66%	15.07%	7.29%	16.20%	12.41%
Target model 439 sm	17.98%	23.75%	36.59%	25.17%	21.28%
N3 or B41	0.28%	11.82%	13.49%	12.88%	11.03%
CVS box	6.14%	2.90%	1.36%	4.34%	9.17%
Lowes Box	7.79%	10.70%	7.33%	6.66%	7.06%
Int Paper 1A5 B45	21.79%	23.16%	19.41%	18.67%	16.32%
int paper 0A0	1.70%	0.91%	1.70%	3.70%	7.41%
Int paper A4	3.58%	1.13%	1.21%	5.95%	11.88%
Int paper 1A7	10.00%	9.13%	8.43%	8.32%	6.98%
int paper BP0	1.10%	2.54%	3.07%	3.31%	4.68%
int paper 2AA	10.59%	11.75%	12.93%	7.51%	11.42%
int paper 2A0	10.88%	12.36%	10.06%	8.97%	14.20%
int paper BP1	7.87%	9.34%	7.87%	4.33%	12.20%
int paper 2A5	15.84%	11.23%	13.70%	14.26%	20.39%
int paper 2A8	15.83%	14.58%	12.96%	18.98%	19.58%
int paper 3A1	50.73%	51.90%	52.59%	55.36%	50.64%
int paper B0	18.78%	19.99%	19.98%	21.36%	18.88%

Table 7.0.7. Algorithmic Results

	(C) Error Test 1	(C) Error Test 2	(C) Error Test 3	(C) Error Test 4	(C) Error Test 5
Box					
A3 or BY1	6.68%	19.61%	15.45%	34.87%	5.53%
Shar	6.67%	13.21%	6.38%	18.97%	14.56%
int paper 1AD BNA	2.12%	1.14%	9.25%	1.42%	2.70%
LLBean	6.30%	9.82%	12.65%	2.70%	9.21%
Target model 439 sm	9.13%	20.56%	23.82%	22.22%	16.46%
N3 or B41	1.13%	3.63%	0.84%	43.42%	42.58%
CVS box	4.84%	3.04%	5.03%	9.93%	5.50%
Lowes Box	16.84%	12.20%	10.56%	15.86%	21.04%
Int Paper 1A5 B45	11.62%	14.36%	16.83%	8.61%	11.64%
int paper 0A0	12.37%	3.06%	5.67%	14.58%	14.75%
Int paper A4	9.12%	12.98%	8.27%	6.80%	6.86%
Int paper 1A7	10.18%	13.99%	15.66%	17.30%	11.94%
int paper BP0	15.54%	13.84%	20.29%	0.51%	4.49%
int paper 2AA	14.37%	16.27%	17.39%	21.32%	13.20%
int paper 2A0	15.55%	13.50%	14.14%	18.89%	17.40%
int paper BP1	5.92%	9.24%	2.54%	11.41%	20.64%
int paper 2A5	11.04%	10.68%	19.20%	21.30%	21.69%
int paper 2A8	22.09%	15.75%	19.13%	17.71%	20.91%
int paper 3A1	19.26%	18.68%	23.15%	22.75%	22.85%
int paper B0	21.85%	22.54%	24.77%	20.50%	22.62%



Table 7.0.8. Algorithmic Results

Box	(A) Avg	(B) Avg	(C) Avg
A3 or BY1	66.91	51.26	42.79
Shar	68.62	36.88	19.90
int paper 1AD BNA	121.72	41.22	33.72
LLBean	105.80	60.41	36.75
Target model 439 sm	240.53	153.07	111.92
N3 or B41	190.31	81.47	52.87
CVS box	282.20	271.24	159.27
Lowes Box	207.85	207.18	166.03
Int Paper 1A5 B45	168.66	76.87	58.84
int paper 0A0	57.01	21.17	15.92
Int paper A4	114.79	87.58	64.74
Int paper 1A7	140.72	114.13	66.01
int paper BP0	91.13	20.03	17.72
int paper 2AA	441.06	113.06	79.23
int paper 2A0	372.58	155.81	129.80
int paper BP1	145.72	32.29	23.50
int paper 2A5	247.78	97.82	54.60
int paper 2A8	565.52	272.34	203.69
int paper 3A1	734.10	495.55	261.18
int paper B0	214.26	224.02	148.17

Table 7.0.9. Algorithmic Results

Box	(A) Min Error	(B) Min error	(C) Min Error	(A) Max Error	(B) Max Error	(C) Max Error
A3 or BY1	8.05%	0.40%	5.53%	13.29%	5.45%	34.87%
Shar	0.50%	9.09%	6.38%	4.05%	15.17%	18.97%
int paper 1AD BNA	0.89%	10.12%	1.14%	16.21%	14.74%	9.25%
LLBean	0.73%	7.29%	2.70%	8.44%	16.20%	12.65%
Target model 439 sm	0.18%	17.98%	9.13%	10.82%	36.59%	23.82%
N3 or B41	0.77%	0.28%	0.84%	4.47%	13.49%	43.42%
CVS box	0.40%	1.36%	3.04%	7.12%	9.17%	9.93%
Lowes Box	6.09%	6.66%	10.56%	10.70%	10.70%	21.04%
Int Paper 1A5 B45	9.81%	16.32%	8.61%	16.16%	23.16%	16.83%
int paper 0A0	0.67%	0.91%	3.06%	13.23%	7.41%	14.75%
Int paper A4	11.28%	1.13%	6.80%	13.80%	11.88%	12.98%
Int paper 1A7	18.18%	6.98%	10.18%	23.43%	10.00%	17.30%
int paper BP0	10.00%	1.10%	0.51%	19.68%	4.68%	20.29%
int paper 2AA	13.97%	7.51%	13.20%	15.96%	12.93%	21.32%
int paper 2A0	15.68%	8.97%	13.50%	17.04%	14.20%	18.89%
int paper BP1	13.09%	4.33%	2.54%	18.72%	12.20%	20.64%
int paper 2A5	11.21%	11.23%	10.68%	14.35%	20.39%	21.69%
int paper 2A8	13.05%	12.96%	15.75%	15.66%	19.58%	22.09%
int paper 3A1	13.74%	50.64%	18.68%	17.77%	55.36%	23.15%
int paper B0	11.02%	18.78%	20.50%	19.35%	21.36%	24.77%

Table 7.0.10. Algorithmic Results

Box	(A) Avg Error	(B) Avg Error	(C) Avg Error
A3 or BY1	4.42%	2.36%	16.43%
Shar	0.83%	10.70%	9.29%
int paper 1AD BNA	5.09%	12.77%	1.40%
LLBean	3.82%	12.13%	8.14%
Target model 439 sm	1.81%	24.95%	18.44%
N3 or B41	2.41%	4.51%	16.08%
CVS box	2.73%	0.57%	5.67%
Lowes Box	8.25%	7.91%	15.30%
Int Paper 1A5 B45	13.58%	19.87%	12.61%
int paper 0A0	5.18%	1.72%	8.86%
Int paper A4	12.54%	4.27%	8.81%
Int paper 1A7	21.31%	8.57%	13.81%
int paper BP0	13.92%	0.16%	10.73%
int paper 2AA	14.86%	10.84%	16.51%
int paper 2A0	16.43%	11.29%	15.89%
int paper BP1	15.76%	8.32%	9.95%
int paper 2A5	12.63%	15.09%	16.78%
int paper 2A8	14.48%	16.39%	19.12%
int paper 3A1	15.51%	52.24%	21.34%
int paper B0	14.58%	19.80%	22.46%

Table 7.0.11. Algorithmic Results

Box	Min A	Max A	Min B	Max B	Min C	Max C
A3 or BY1	63.49	79.30	49.64	53.81	38.78	49.57
Shar	66.38	71.52	36.34	38.36	17.78	23.40
int paper 1AD BNA	107.46	130.52	40.29	42.47	32.35	36.33
LLBean	100.72	109.20	57.61	63.74	34.94	38.92
Target model 439 sm	223.35	261.82	144.53	167.32	103.13	117.01
N3 or B41	186.28	196.50	74.32	96.82	35.64	65.29
CVS box	269.46	304.77	256.04	297.82	152.08	163.71
Lowes Box	203.69	212.54	204.79	212.55	159.21	174.29
Int Paper 1A5 B45	163.07	172.50	74.59	78.97	56.75	61.04
int paper 0A0	52.17	60.53	20.46	22.36	14.18	16.78
Int paper A4	113.51	116.07	82.99	93.98	63.55	67.22
Int paper 1A7	137.09	143.18	112.46	115.64	63.91	68.03
int paper BP0	88.00	95.74	19.34	20.94	15.92	19.25
int paper 2AA	437.66	445.27	109.66	115.18	76.97	82.50
int paper 2A0	370.17	374.53	152.56	159.88	127.12	133.15
int paper BP1	142.35	149.43	31.10	33.45	21.92	25.79
int paper 2A5	244.66	251.56	94.54	102.34	51.74	56.89
int paper 2A8	558.48	571.37	264.33	279.83	197.94	208.77
int paper 3A1	722.83	748.46	490.33	505.69	255.46	265.08
int paper B0	207.60	223.18	222.13	226.94	145.81	150.97

Table 7.0.12. Algorithmic Results

Box	Avg Volume	Defined Volume	Volume Avg Error
A3 or BY1	383.07	367.50	4.24%
Shar	153.15	224.86	31.89%
int paper 1AD BNA	280.70	448.88	37.47%
LLBean	330.77	550.00	39.86%
Target model 439 sm	1385.50	1653.75	16.22%
N3 or B41	617.94	1023.75	39.64%
CVS box	2383.09	3655.39	34.81%
Lowes Box	1824.99	2304.00	20.79%
Int Paper 1A5 B45	596.10	705.38	15.49%
int paper 0A0	94.61	135.28	30.06%
Int paper A4	550.63	714.00	22.88%
Int paper 1A7	702.78	841.00	16.44%
int paper BP0	122.75	160.00	23.28%
int paper 2AA	1356.58	1632.00	16.88%
int paper 2A0	1873.55	2240.00	16.36%
int paper BP1	226.97	283.22	19.86%
int paper 2A5	785.14	935.00	16.03%
int paper 2A8	3822.86	4446.00	14.02%
int paper 3A1	6652.82	6672.75	0.30%
int paper B0	1820.16	2057.00	11.51%

# Bibliography

- [1] Richard Szeliski, *Computer Vision: Algorithms and Applications*, 1st, Springer-Verlag New York, Inc., New York, NY, USA, 2010.
- [2] Jan Erik Solem, *Programming Computer Vision with Python*, O'Reilly, 2012.
- [3] Neale Godfrey, *The Young And The Restless: Millennials On The Move*, Forbes (2016).
- [4] Atduskgreg, *atduskgreg/opencv-processing* (2017).
- [5] *Camera calibration With OpenCV OpenCV 2.4.13.4 documentation*, OpenCV (2017).
- [6] Adrian Rosebrock, *Find distance from camera to object using Python and OpenCV*, Py Image Search (July 12, 2016).
- [7] Megan J. Benetsky, Charlynn A. Burd, and Melanie A. Rapino, *Young Adult Migration: 20072009 to 20102012*, Census.gov (March 2015).
- [8] *How to Find the Area of a Quadrilateral*, wikiHow (May 22, 2017).
- [9] Kevin NA, *A Catalog of Amazon.com Box Sizes*, Incomptech.
- [10] *3D Viewing: the Pinhole Camera Model*, Scratch Pixel.
- [11] Gabriel Taubin and Daniel Moreno, *Simple, Accurate, and Robust Projector-Camera Calibration*, Brown University School of Engineering (2012).