Estimating volumes of colored liquids in transparent, colorless, rotationally symmetric containers

Kevin Li

(UNI: kl3285)

COMS 4735

Professor John Kender

Table of Contents

1. Introduction

The objective of this project is to develop a visual interface for estimating volumes of colored liquids in transparent, colorless, rotationally symmetrical vessels placed on horizontally flat surfaces. For instance, a user may want to determine the volume of Coca-Cola or orange juice in a symmetrical water glass. By supplying the system with a query and a set of reference images of a common vessel containing known liquid volumes, an estimate of the query's liquid volume (ml) can be generated. The system estimates the query volume by mapping a 2-dimensional image to a 3-dimensional metric without the need for depth information. This report begins by providing an overview of existing research and projects that employ various methods to address similar problems, followed by a detailed explanation of the specific approach used in this project. Subsequently, the report will then cover the domain engineering process, the image library utilized, the algorithm implementation and design, as well as the testing and results obtained.

2. Background and Related Work

Numerous methods have been implemented for dimensional analysis and object volume estimation. Despite their effectiveness, many rely on complex neural networks and classifiers that necessitate extensive training and may not work well for all scenarios, as observed through testing select open-source models. One proposed model [1] requires a single 2-dimension input image and involves a sequence of three tasks: 1) object recognition using Haar Cascades, 2) dimension calculation using a reference object and conversion factors, and 3) volume estimation with predefined 3D shape models. According to experimental results in the paper, the system is highly effective, achieving an accuracy rate of 93.69%. The introduction of a reference object with known dimensions, also known as the pixel density method, compensates for the absence of depth information in 2-dimensional images. By computing a conversion factor, scaled values can be used

to subsequently compute volumes using predefined 3-dimensional shapes. Despite the high accuracy of the system reported in the paper, the system depends on classifiers, which require pre-training and would only work with predefined regular 3-dimensional geometries. The system would likely encounter problems with more complex rotationally symmetric vessels featuring pronounced curves. Nevertheless, the use of a reference object was deemed a useful method for accounting for the lack of depth information and was thus incorporated into the final system design.

In addition to reviewing reports on dimensional analysis, projects attempting to detect liquid-liquid and liquid-air boundaries were also examined. A project [2] focusing on liquid segmentation in chemistry applications, uses computer vision-based model to recognize liquid surfaces and liquid levels in transparent containers. The system requires two 2-dimensional images as its inputs: an image of the container's edge boundaries in the query and the query itself. At a high level, the system works by scanning the vessel area in the image and generating curves corresponding to the outlines of all possible liquid surfaces in the vessel. The generated curves are each compared against the query image and rated with respect to a specific image property (i.e. intensity change and edge density). The highest scoring curve is the one that most closely resembles the actual liquid surface in the query. Although the system is capable of identifying liquid boundaries, especially in images with perspective distortion (i.e. the liquid boundary is ellipsoidal), experimental results report a high percentage of false matches. The system also requires vessel segmentation, which is a difficult process affected by the quality of input images. Images where glass vessel edges are not clearly captured can complicate estimations. While pre-trained neural networks could also be used to identify the vessel and the liquid areas, they appear to generate inaccurate results, as determined with a round of individual tests presented below:

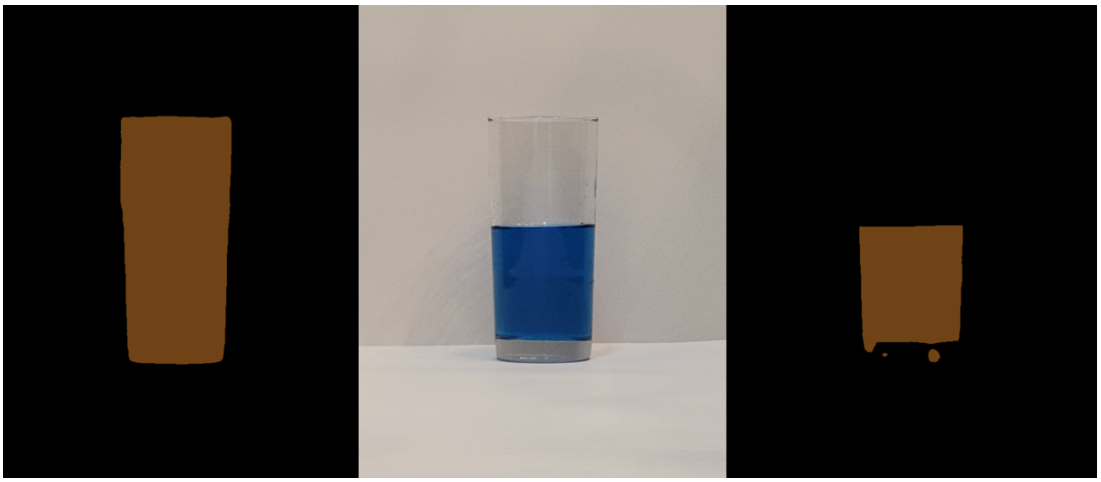


Fig. 1- Successful segmentation of container from background (left-most) but unsuccessful segmentation of liquid (right-most) using open source system

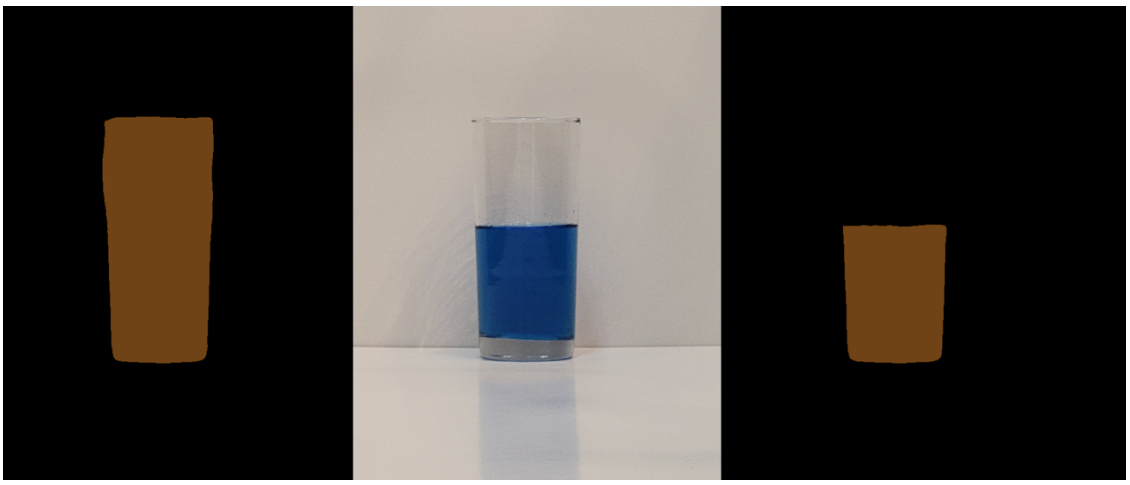


Fig. 2 - Relatively successful segmentation of container from background but unsuccesful segmentation of liquid (right-most using open-source system. Container's base is mistaken as liquid.

While the model could be useful in determining liquid boundaries that could improve this project's system accuracy, it was ultimately replaced by a more reliable segmentation method explained later in this report.

3. Domain Engineering

All images used to test the implemented system were captured under the following fixed conditions:

- White background and surface to increase contrast between the liquid and vessel
- Fixed camera, camera position, object-camera distance, and angle for each set of images for a given container
- Fixed lighting conditions (i.e. uniform light sources)
- Single fixed color liquid for each set of images

Fig. 3 - Domain engineering

A problem that was observed is the effects of parallax and perspective distortion. Images of the liquid containing vessel taken with a small object-to-camera distance would introduce greater distortion, causing the liquid boundary to be ellipsoidal rather than horizontal . This would lead to inaccurate volume estimations that would require more sophisticated liquid boundary identification. The solution is to increase the object-camera distance. By positioning the camera farther away and pointing it to the middle of the vessel, the effects of perspective distortion can be minimized [Fig. 3], and a greater range of liquid boundaries corresponding to different volumes can be made more horizontal.

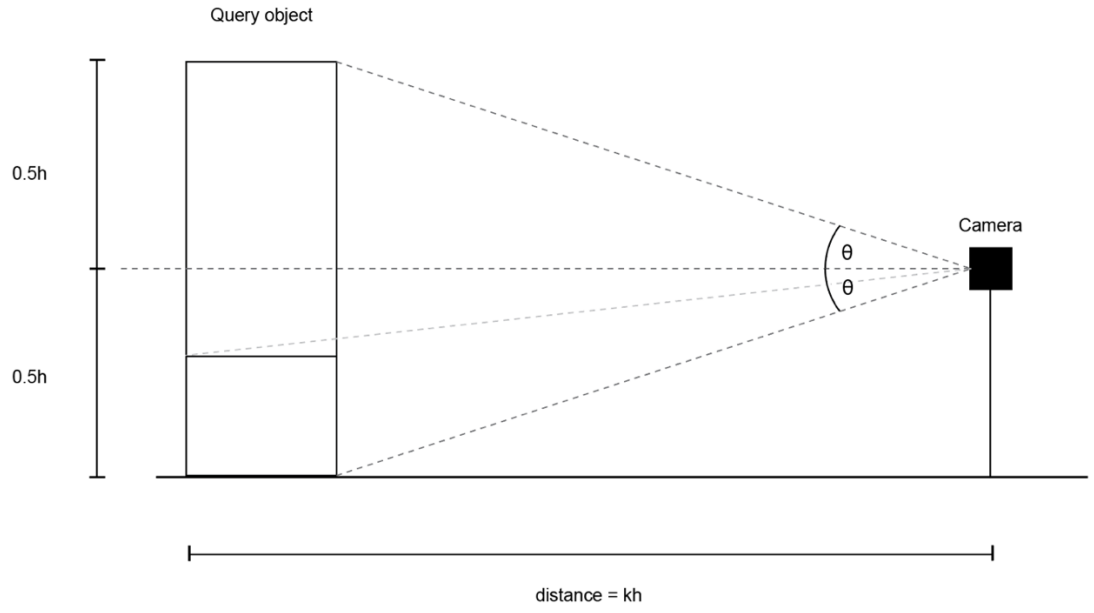


Fig. 4 - Diagram illustrating how to account for perspective distortion. h is container height and k is a constant.

$$tan\,\theta = \frac{1}{k}$$

The greater the distance, the smaller the angle θ. A camera-to-query distance of approximately 10h, which would result in an angle of 5.71 degrees (largest angle between horizontal and top and bottom vessel edges), was adopted for all final captured images used for system testing that are visible in the image library. The effects of parallax are most obvious for volumes of liquid below the vertical midpoint of the container, which are accounted for in the system design through an introduction of multiple reference images, as discussed later.

4. Image Library

Approximately 100 images were captured for the entire project, but only 26 of them [Fig. 5, 6, 7] were used in the final system testing. This is due to the changes made to domain engineering to improve system accuracy (i.e. changes to light source and object-camera distance, realization of asymmetrical nature of containers, etc.). Images for a total of 3 different types of colorless transparent rotationally symmetric containers were used for the final system test. As outlined above, images for each container type, with varying volumes of liquid, were captured under fixed conditions. Additional apparatus used during image capturing include:

- Food coloring (to create dark colored liquids)
- A measuring cylinder (to measure liquid volume to pour into the container for both the references and queries)

All images are elevation views of the containers and were captured under conditions outlined above. All images were labeled with their volume (ml). A selection of images from the library are visible below:

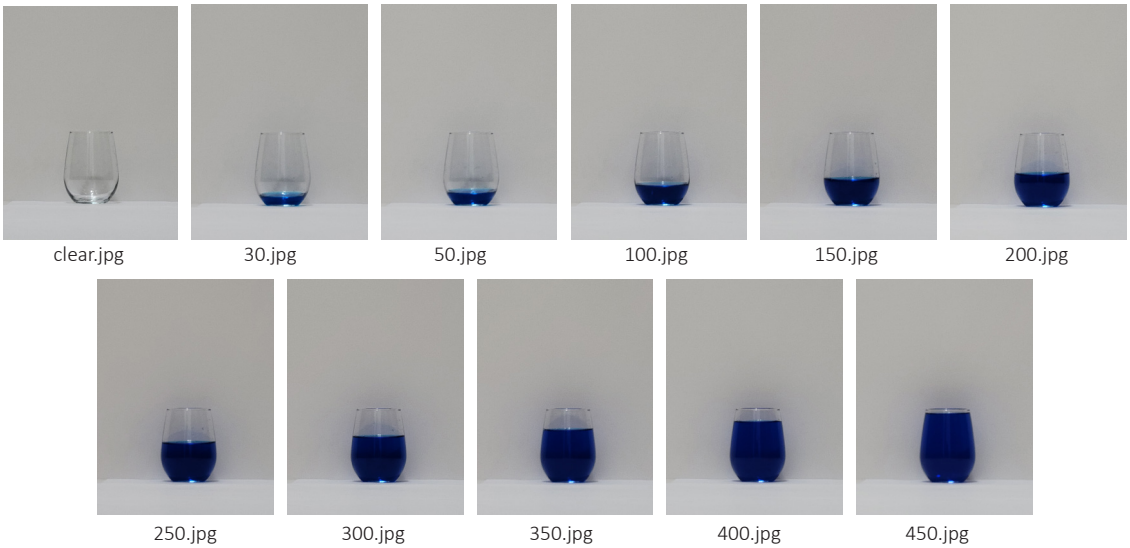


Fig. 5 – 'Juice' glass container with varying volumes of liquid (Used in final system test)

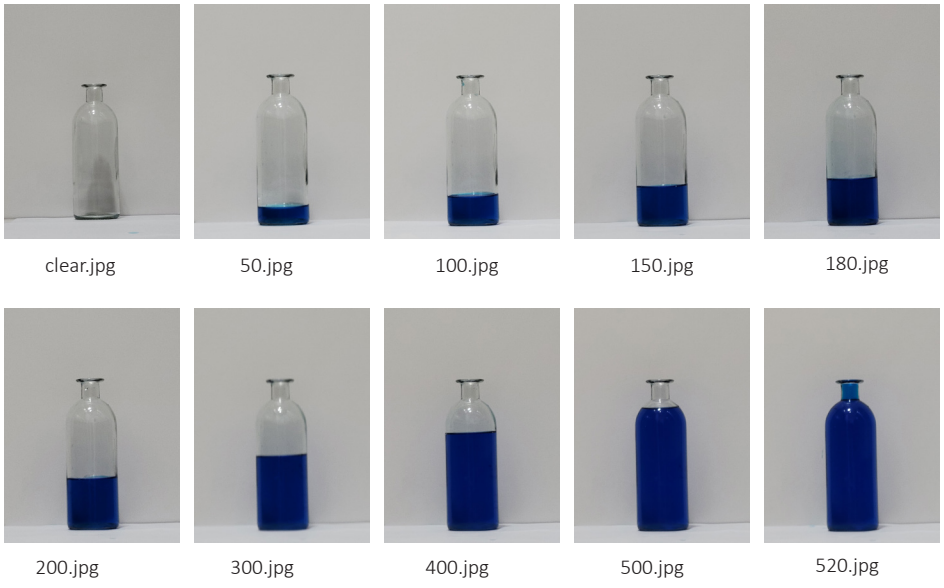


Fig. 6 - 'Neck' glass container with varying volumes of liquid (Used in final system test)

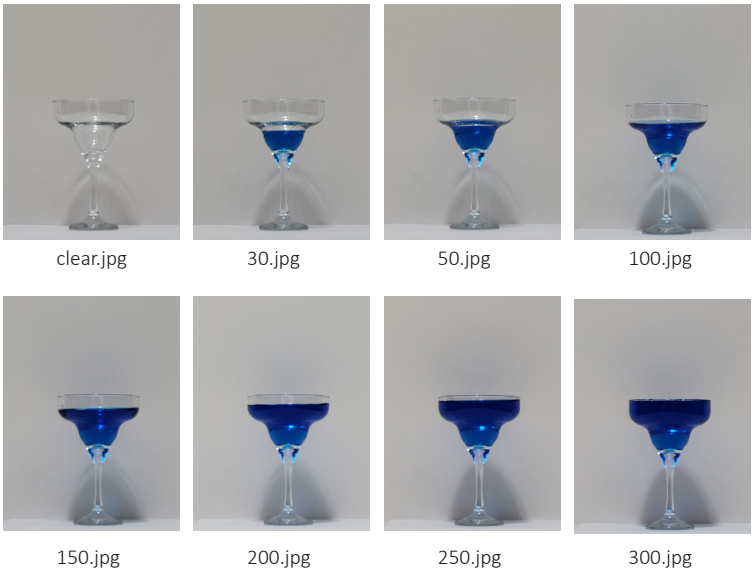


Fig. 7 – 'Wine' glass container with varying volumes of liquid (Used in final system test)

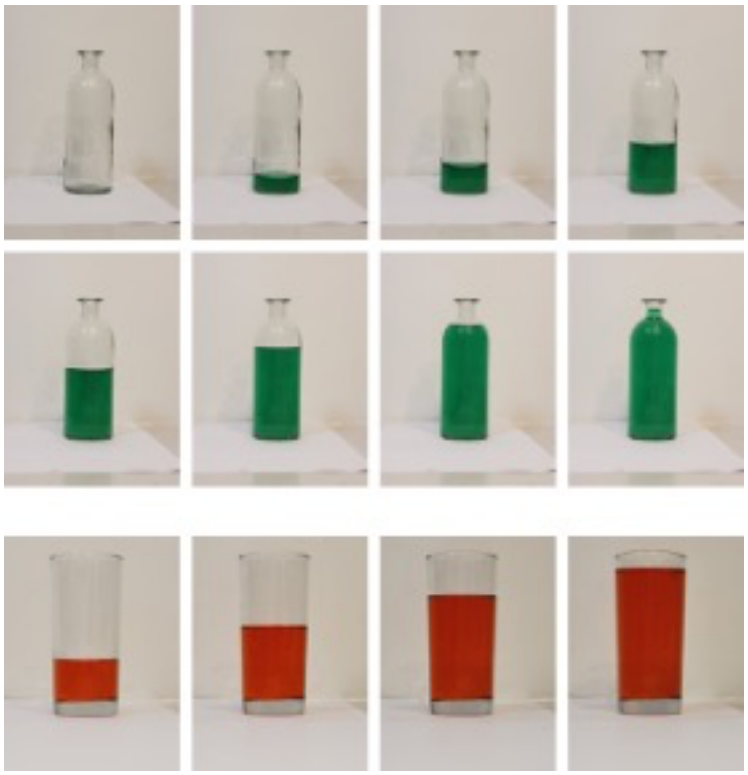


Fig. 8 - Forfeited images due to improper domain engineering
(i.e. small camera-object distance with large perspective distortion)

5. Algorithm Design and Implementation

5.1. Specifications

Main system specification (system.py)

Language: Python 3.9

Libraries: OpenCV, Numpy

Front-end application specification (system.py, app.py, index.html, results.html)

Languages: HTML, CSS, JavaScript, Python 3.9

Libraries: Flask, OpenCV

5.2. System Breakdown

The final system (system.py for general system and app.py for system optimized for the front-end interface) estimates the liquid volume in a query using reference images containing known liquid volumes.

Inputs:

- Reference images with known liquid volumes (ml)
- Query image
- Query ground truth volume (ml) *(Not used in decision-making; only used for error computation)*

Outputs:

- Estimation of liquid volume (ml) in the query (and error calculations)

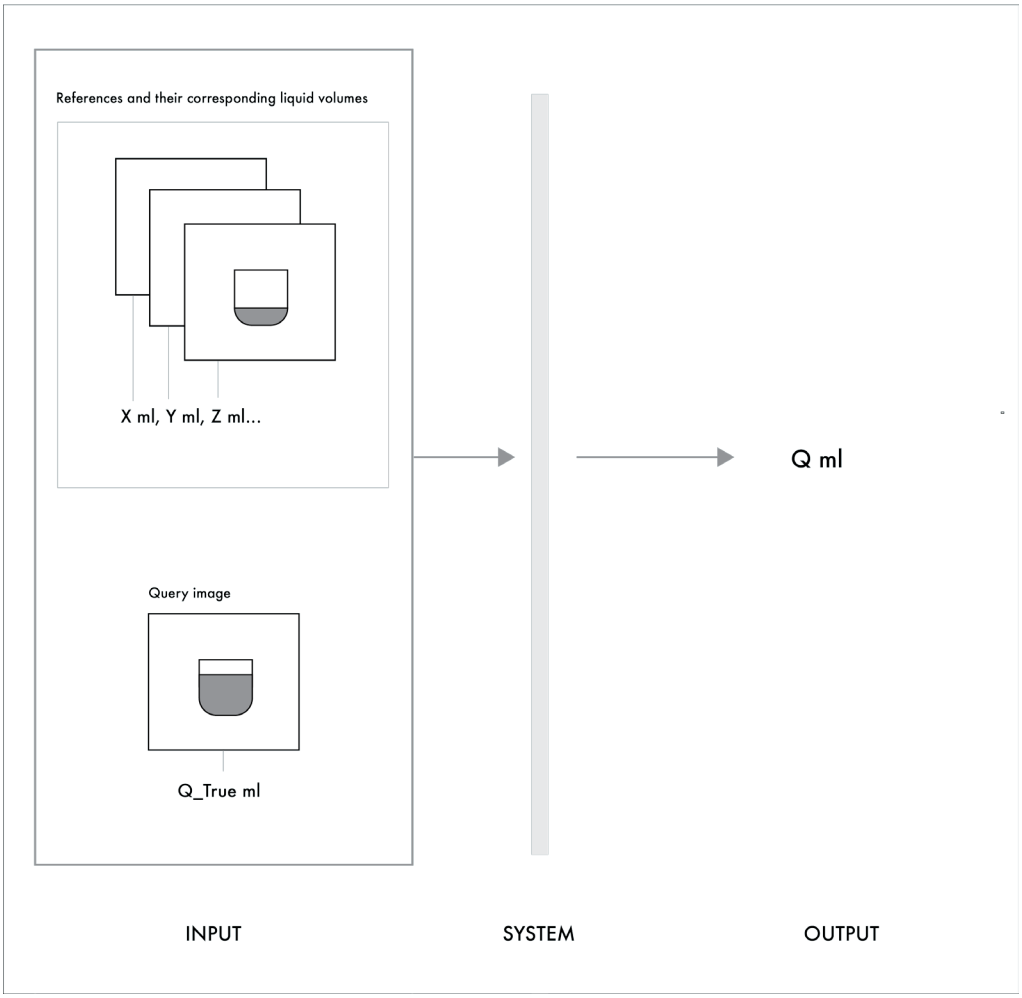


Fig. 9 - Diagram illustrating system usage.

Note: Query and reference images must be taken under same conditions for the same vessel.
Reference and queries must contain a dark colored liquid.
Q_True is not used in decision-making but in system assessment.

At a high-level, the system algorithm can be broken down into the following steps:

1. Reference processing:
    1. Identification of colors in images using k-means clustering in every reference's down-sampled image

    2. Segmentation of liquid from container and background in every original reference image using identified liquid color
    3. Volume estimation of liquid in original reference image using integral calculus (disc method) based on identified liquid areas
    4. Computation of cubic-pixel-to-ml conversion factors
2. Query processing:
    1. Identification of colors in images using k-means clustering in every reference's down-sampled image
    2. Segmentation of liquid from container and background in every original reference image using identified liquid color
    3. Volume estimation of liquid in original reference image using integral calculus (disc method) based on identified liquid areas
    4. Cubic pixel volume comparison with references to determine best conversion factor. Two methods, linear interpolation and closest reference, are used to determine conversion factors.
    5. Mapping of cubic pixel volume to ml volume using determined conversion factors
3. Output result: system estimation liquid volume and errors

5.3. Design Explanation

One of the trickiest parts of the project was how to accurately identify the liquid in the image. While liquid surface and container edge detection was initially considered as a possible method for this project, rounds of testing proved that edge detection methods are ineffective, as explained in the literature review section. Although many edge detection methods, including

Canny and Sobel methods, may be effective for opaque and colored materials, they fail to successfully identify edges of glass containers and return disconnected contours. Factors including glare and light refection on glass containers also disrupt edge detection. As such, edge detection is not used in the final system design.

Instead, k-means clustering [3] was identified to be a plausible and highly effective way of segmenting the colored liquid from the container and background. Based on the assumption that the vessel is colorless and transparent, and the liquid is colored, k-means clustering iterates through all pixels, considers all existing colors in the image, and subsequently creates two clusters or two color regions. The LAB color space was used for the clustering algorithm, as it was determined that using an HSV color space would lead to inconsistent results, especially since lighting conditions can greatly affect the algorithm [Fig. 5]. The LAB color space is also better for color-based segmentation methods than its RGB counterpart. Using k-means clustering, a mask of the liquid can be subsequently generated and used to estimate the liquid volume. Additional area segmentation techniques are also present in the system to remove small disconnected areas not representing the liquid (likely due to light reflections).

While k-means clustering is an effective means of identifying a separate color region in the image, it can be computationally expensive, especially if the image size is large, as a greater number of pixels would have to be considered. To improve efficiency, a downsampled version of the references and queries are used to identify the liquid color using k-means clustering. This significantly reduces runtime. Although using a down sampled version might affect accuracy, as the identified color regions in the down sampled version might differ from the original, similar results generated by applying k-means clustering on both the down sampled and original images suggest that running k-means clustering on a down sampled image does not sacrifice accuracy [Fig.

15, 22]. Without using a down sampled version of the image, the entire system could as long as 1 minute to run (depending on the number of reference images).

Using integral calculus, and more specifically, volumes of solids of revolution, the 2-dimensional liquid mask can be mapped to a 3-dimensional volume. The system uses the disc integration method to compute volume: each row of non-white pixels is treated as a cylindrical disc and its volume is computed using the standard formula, $V = \pi r^2 h$, where h is the pixel height (i.e. 1 pixel). The volumes of individual rows are subsequently summed to obtain an overall volume. The benefit of using disc integration is that as the algorithm attempts to find the left and rightmost pixels of each row to determine the radius; any glare or light reflection on the middle container surface would not affect the calculation. A problem with this method, however, is that it requires careful domain engineering to eliminate internal reflections along the edges which can be mistaken as liquid, as in [Fig. 11]. In this case, the system identified liquid volume would be much greater than the true volume as it uses the left-most and right-most pixels of a row containing non-white pixels.

Fig. 10 - Example of failed k-means clustering based segmentation using HSV color space. Original image (left) and binary intermediate (right).

Fig. 11 - Example of case where internal reflection leads to problematic segmentation
Original image (left) and binary intermediate (right).

During early rounds of system testing, it was also noticed that the closer a reference's volume is to the query, the more accurate the results, and vice versa. This is likely due to the influence of perspective distortion, which is difficult to eliminate. Increasing the distance between the camera and containers may further minimize the distortion. The two graphs below [Fig. 12, 13] illustrate the deviation between the ground truth and estimation as the reference image's volume increases or decreases from the query's volume, indicating the potential need to use multiple references. By introducing more references, the system can choose one that is closest to the query to estimate liquid volume more accurately as the effects of perspective distortion would be minimized.

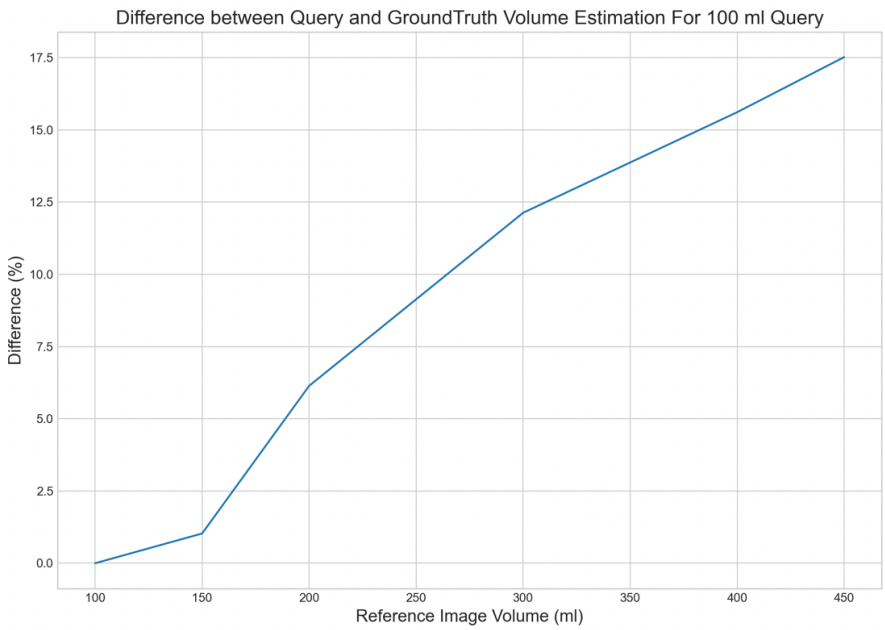


Fig. 12 – Percentage difference in liquid estimation for a 100 ml query with different references

Fig. 13 – Percentage difference in liquid estimation for a 450 m query with different references

Another test was performed on the same vessel but testing for queries with a fixed 250ml reference [Fig. 14]. Unsurprisingly, there are differences between the estimated and ground truth values on either side of the reference's volume. The differences, however, appear to be more substantial for queries with lower volumes. This, again, can likely be attributed to the ellipsoidal liquid boundary observable when less of the container is filled. This test and the two tests above reinforced the understanding that using references with volumes closer to the queries would improve estimations as the images would be affected by a more similar degree of perspective distortion.

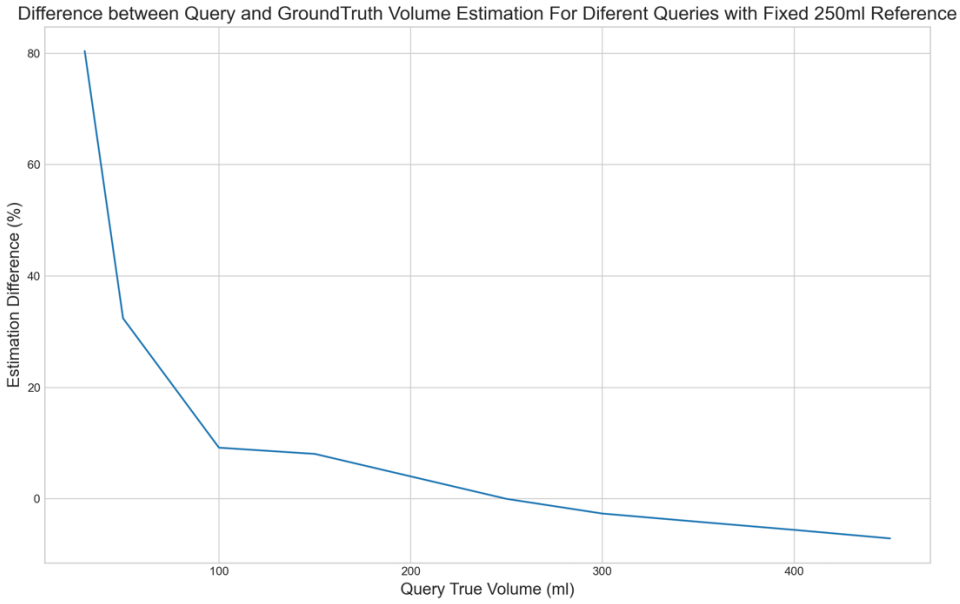


Fig. 14 – Error (%) of estimated volumes for different queries with fixed 250 ml reference

To account for perspective distortion, the system allows multiple reference inputs. The introduction of multiple references allows the system to choose between different conversion factors to optimize results for a given query. The conversion rate to be adopted for a query depends on the differences in cubic pixel volumes between the query and the references. The conversion

factor associated with the reference leading to the smallest difference between the query and reference's computed cubic volumes is used for the final computation.

Moreover, the introduction of linear interpolation for the conversion factor further fine-tunes the results. For queries with pixel cubic volumes between those of two references, a conversion factor is computed using linear interpolation. For queries with volumes that lie below or above a single reference's volume, the closest reference approach is adopted, where the conversion factor used for the query is simply the one associated with the reference with the closest cubic pixel volume. A simulation [Fig. 15, 22] testing versions with and without linear interpolation (for queries with above and below references) shows that its inclusion improves system accuracy. Linear interpolation was thus incorporated into the final system.

The final code for the system and its front-end application can be found in the Appendices.

5.4. Front-end Application

The front-end utilizes the implemented system in system.py. Relevant code files include app.py, system.py, index.html, and results.html. As inputs, the index.html webpage receives a query image, query ground truth volume (ml), reference images, and their volumes. The estimation, calculated errors, and binary intermediates are returned to results.html.

6. Results

Results generated from a simulation for three different containers can be found below in [Fig. 15]. For queries of each container type, 3 references with low, medium (approximately half-filled), and high liquid volumes were used. A total of 17 queries were performed across the 3 different container types. Some results between the interpolation and closest reference methods

are equal, but this is because the system with interpolation uses closest reference when the query's pixel volume does not lie in between that of two references. Comparisons were made in cases where different results were obtained.

Results

Total proportion of cases with a difference in estimation: 11/17

(%) Linear interpolation outperformed closest reference = (10 / 11) * 100 ≈ 90.9%

Average relative error (%) of either method:

Interpolation = 8.89 %

Closest Reference = 11.57 %

Average absolute difference (ml) of either method:

Interpolation = 8.05

Closest Reference = 13.41

Max absolute difference (ml):

Interpolation = 27.83

Closest Reference = 36.52

Container Type	References (ml)	Ground Truth Volume (ml)	Method	Estimated Volume (ml)	Difference = Estimation - Ground Truth (ml)	Aboslute Difference (ml)	Relative Error (%)
juice	50, 200, 350	30	Interpolation	40.56204742	10.56204742	10.56204742	35.21
			Closest reference	40.56204742	10.56204742	10.56204742	35.21
		100	Interpolation	115.9228658	15.92286583	15.92286583	15.92
			Closest reference	106.2500423	6.250042282	6.250042282	6.25
		150	Interpolation	147.9856717	-2.014328336	2.014328336	1.34
			Closest reference	130.3898243	-19.61017571	19.61017571	13.07
		250	Interpolation	249.7915661	-0.2084338934	0.2084338934	0.08
			Closest reference	244.4986666	-5.501333412	5.501333412	2.2
		300	Interpolation	297.8866937	-2.113306272	2.113306272	0.7
			Closest reference	285.7266328	-14.2733672	14.2733672	4.76
		400	Interpolation	395.3419167	-4.658083313	4.658083313	1.16
			Closest reference	395.3419167	-4.658083313	4.658083313	1.16
		450	Interpolation	439.2444504	-10.75554956	10.75554956	2.39
			Closest reference	439.2444504	-10.75554956	10.75554956	2.39
neck	100, 300, 500	50	Interpolation	60.72005059	10.72005059	10.72005059	21.44
			Closest reference	60.72005059	10.72005059	10.72005059	21.44
		150	Interpolation	136.9739471	-13.02605289	13.02605289	8.68
			Closest reference	130.6678943	-19.33210574	19.33210574	12.89
		180	Interpolation	170.5725667	-9.427433251	9.427433251	5.24
			Closest reference	156.1854048	-23.814	23.814	13.23
		200	Interpolation	186.8769103	-13.12308968	13.12308968	6.56
			Closest reference	167.8440832	-32.15591676	32.15591676	16.08
		400	Interpolation	401.5321386	1.532138572	1.532138572	0.38
			Closest reference	396.3803393	-3.619660732	3.619660732	0.9
		520	Interpolation	519.1626716	-0.8373284453	0.8373284453	0.16
			Closest reference	519.1626716	-0.8373284453	0.8373284453	0.16
wine	50, 150, 300	30	Interpolation	23.89150118	-6.108498823	6.108498823	20.36
			Closest reference	23.89150118	-6.108498823	6.108498823	20.36
		100	Interpolation	72.17085329	-27.82914671	27.82914671	27.83
			Closest reference	63.48465971	-36.51534029	36.51534029	36.52
		200	Interpolation	195.0016244	-4.998375617	4.998375617	2.5
			Closest reference	190.8519458	-9.148054189	9.148054189	4.57
		250	Interpolation	246.9598818	-3.040118237	3.040118237	1.22
			Closest reference	235.908287	-14.09171295	14.09171295	5.64
Statistical summary							
Average relative error (%)	Interpolation: 8.89%						
	Closest Reference: 11.57%						
Average absolute difference (ml)	Interpolation: 8.05		Max absolute difference = 27.82914671				
	Closest Reference: 13.41		Max absolute difference = 36.5153402927579				

Fig. 15 - Simulation results testing linear interpolation and closest reference

7. Discussion and Conclusion

Between linear interpolation and closest reference, results [Fig. 15] suggest that linear interpolation significantly improves volume estimations and outperforms the latter. This result justifies the use of linear interpolation in the final system design. The average absolute difference (ml) for linear interpolation, and thus the final system, is 8.05 ml. The average relative error for the final system with linear interpolation is rather high (8.89%) but the metric may not be suitable for assessing system accuracy. Instead, an assessment of the total percentage of estimations lying within a range of $+/- \ k \ ml$ may be better, as relative errors for queries with small volumes can be large (i.e., for 10 ml, a 3 ml difference is equivalent to a 30% relative difference).

Results analysis

Total range of linear interpolation estimates lying in the +/- k range of the ground truth:

Ground truth +/- 10 ml = 10/17 = 58.8 %

Ground truth +/- 15 ml = 15/17 = 88.8%

Ground truth +/- 20 ml = 16/17 = 94.1%

Ground truth +/- 30 ml = 17/17 = 100%

From the results above, it is evident that the system successfully estimates liquid volumes for 15/17 queries for the 3 different containers used in the project to a range of +/- 15 ml. This coincides with the requirements outlined in the evaluation metric section of the project proposal. Even though using relative difference as a metric of success is poor, the initial proposal considers an acceptable system performance as achieving volume estimation errors of less than 20% for at least

80% of the test cases, and 15/17 queries (88.8%) performed in the test have relative errors less than 20%. Thus, the system successfully fulfills the proposal's requirements.

Regardless of the system's relative success, it struggles to segment liquids from containers in certain cases, especially when the container has thick edges and internal reflection [Fig. 16]. However, because the references and queries are captured under the same conditions, the additional areas are negated during the conversion factor process as both the references and queries would contain the same false positive pixels (i.e. systematic error). The final estimation is thus not affected. This problem can be solved by improving domain engineering by minimizing internal reflection and further developing the existing liquid segmentation method.

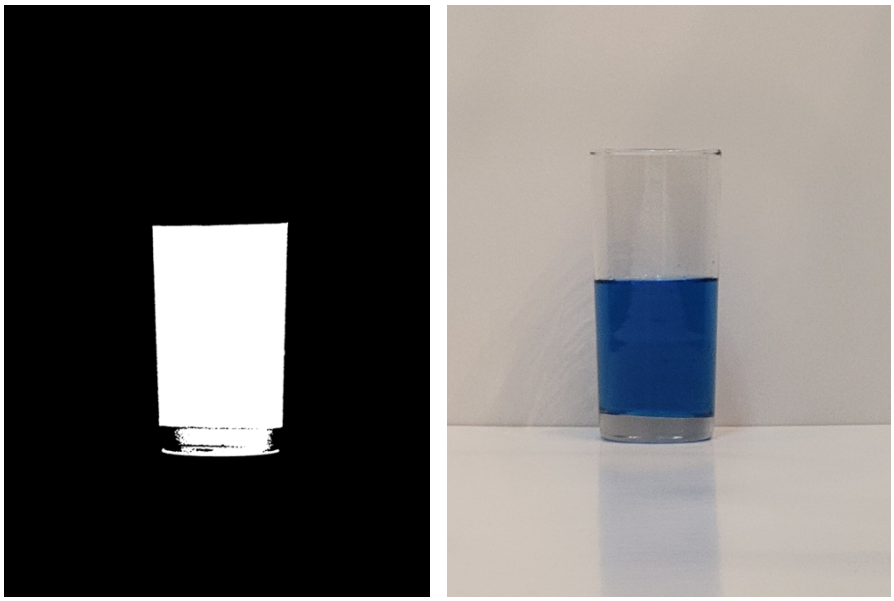


Fig. 16 – Example of improper liquid segmentation using implemented system.

Upon closer analysis of the binary intermediates of the image library [Fig. 17, 18], the importance of domain engineering is further emphasized. For certain images, the system incorrectly identifies dark internal edges as liquid [Fig. 17]. Since the false-positive regions have

relatively small widths, they do not appear to drastically affect the estimation. [Fig. 18] is another example of where internal reflections can negatively affect liquid segmentation. While the system appears to work relatively well with more complex containers, including the wine glass with greater curvature and internal reflection, additional tests on a greater range of rotationally symmetric glass containers are necessary to fully assess how effective the system is for a wide range of containers.

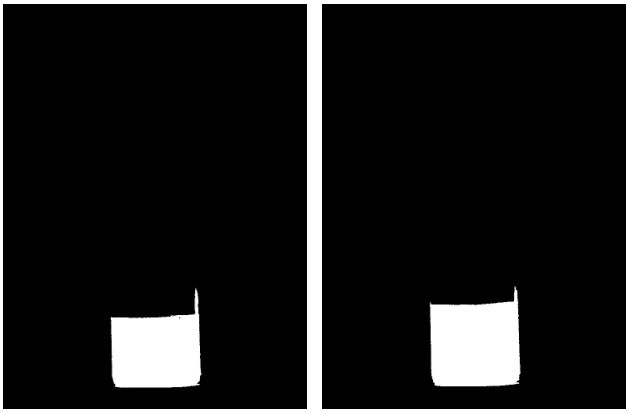
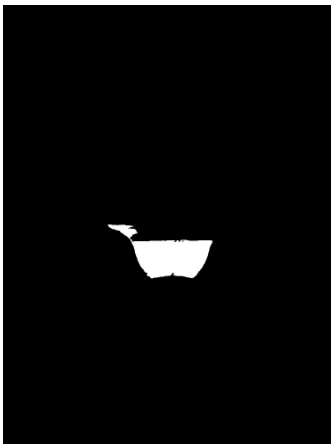


Fig. 17 – Binary intermediates of 'neck' container with liquid. System incorrectly recognizes internal dark edges as liquid.



30.jpg

Fig. 18 – Binary intermediate of 'wine' container with liquid. System incorrectly recognizes a part of the container as liquid.

Other potential sources of error include:

- Incorrect measurement of ground truth liquid volumes during the image capturing process
- Perspective distortion and non-horizontal liquid boundaries

Ways to improve system accuracy include:

- Minimize effects of perspective distortion through increasing camera-object distance during image capture
- Reducing internal reflection, glare, etc. that affect liquid segmentation processes
- Increasing the number of reference images set apart at equal intervals
- Introducing of non-linear interpolation and extrapolation techniques for conversion factor estimations

Overall, although the system works well with dark colored liquids (particularly blue) and simple, transparent, colorless rotationally symmetrical containers, it struggles to accurately estimate low volumes of liquid, especially when perspective distortion and non-horizontal liquid boundaries are apparent. The system also faces challenges in correctly estimating more complicated vessels, such as wine glasses. High-quality images, good domain engineering, and a large number of reference image inputs appear to be crucial in ensuring a high system success rate.

9. References

[1] A. S. Parihar, M. Gupta, V. Sikka, and G. Kaur, "Dimensional analysis of objects in a 2D image," in 2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT), 2017, pp. 1–7, doi: 10.1109/ICCCNT.2017.8203937.

[2] S. Eppel and T. Kachman, "Computer vision-based recognition of liquid surfaces and phase boundaries in transparent vessels, with emphasis on chemistry applications," 2014.

[3] L. Gur Arie, "Image color Segmentation by K-means clustering algorithm," Medium, Feb. 14, 2023. [Online]. Available: https://towardsdatascience.com/image-color-segmentation-by-k-means-clustering-algorithm-5792e563f26e.

[4] J. Chopin, H. Laga, and S. J. Miklavcic, "A new method for accurate, high-throughput volume estimation from three 2D projective images," Int. J. Food Prop., vol. 20, no. 10, pp. 2344–2357, 2017, doi: 10.1080/10942912.2016.1236814.

[5] M. Cobo et al., "Artificial intelligence to estimate wine volume from single-view images," Heliyon, vol. 8, no. 9, Art. no. e10557, 2022, doi: 10.1016/j.heliyon.2022.e10557.

[6] A. Rosebrock, "Measuring size of objects in an image with OpenCV," PyImageSearch, Mar. 28, 2016. [Online]. Available: https://pyimagesearch.com/2016/03/28/measuring-size-of-objects-in-an-image-with-opencv/.

[7] Y. Yue, W. Jia, and M. Sun, "Measurement of Food Volume Based on Single 2-D Image without Conventional Camera Calibration," in Conf. Proc. Annu. Int. Conf. IEEE Eng. Med. Biol. Soc., 2012, pp. 2166–2169, doi: 10.1109/EMBC.2012.6346390.

10. Appendices

10.1. Binary intermediates of image library

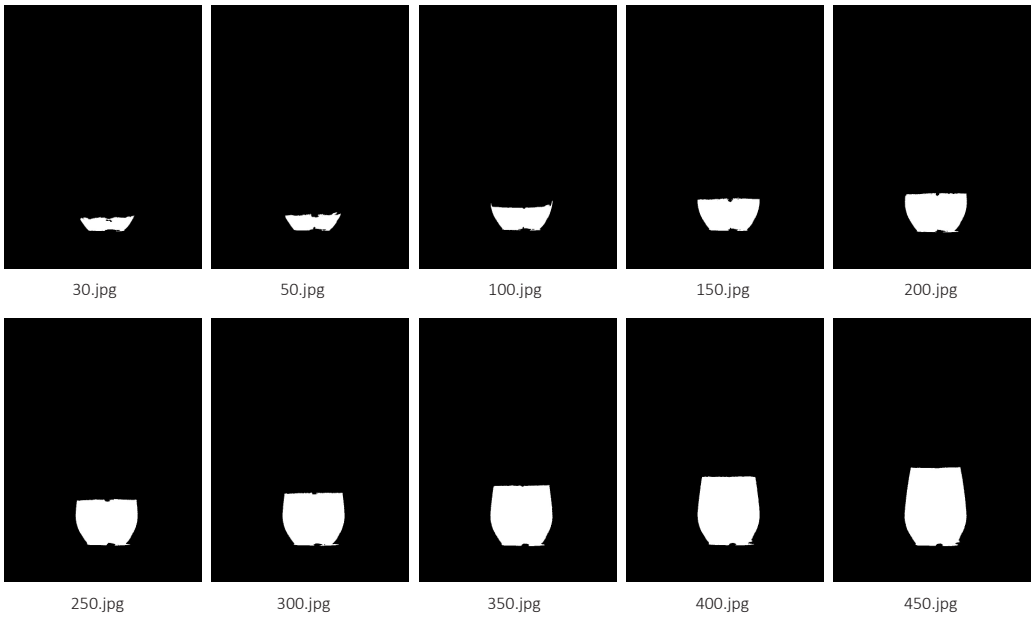


Fig. 19 – 'Juice' binary intermediates generated from system using Fig.5

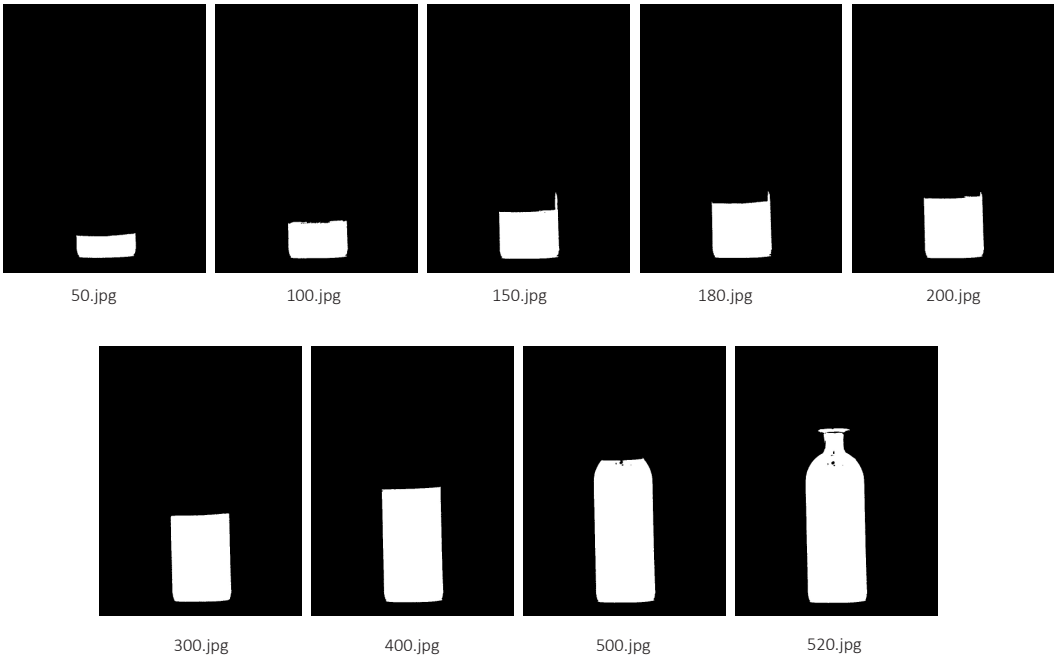
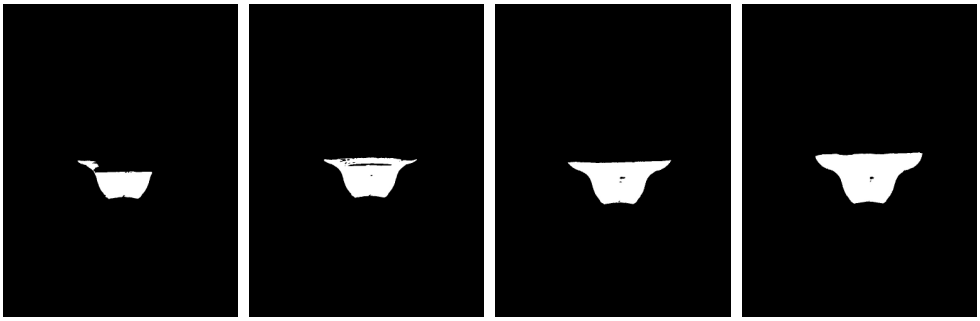


Fig. 20 – 'Neck' binary intermediates generated from system using Fig. 6



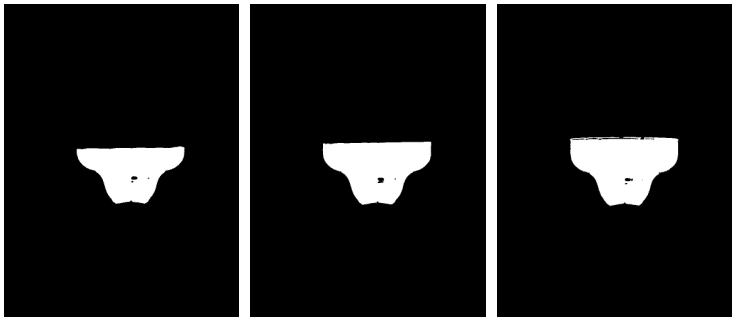


Fig. 21 – 'Wine' binary intermediates generated from system using Fig. 7

10.2. Additional figures

Container Type	References (ml)	Ground Truth Volume (ml)	Method	Estimated Volume (ml)	Difference = Estimation - Ground Truth (ml)	Absolute Difference (ml)	Relative Error (%)
Juice	50, 200, 350	30	Interpolation	41.33772125	11.33772125	11.33772125	37.79%
			Closest Reference	41.33664327	11.33664327	11.33664327	37.79%
	100		Interpolation	112.8186486	12.8186486	12.8186486	12.82%
			Closest Reference	105.0472777	5.0472777	5.0472777	5.05%
	150		Interpolation	150.4273135	0.4273135334	0.4273135334	0.28%
			Closest Reference	134.5259494	-15.47405058	15.47405058	10.32%
	250		Interpolation	251.0058138	1.005813823	1.005813823	0.40%
			Closest Reference	246.1311124	-3.868887572	3.868887572	1.55%
	300		Interpolation	299.0624055	-0.937594214	0.937594214	0.31%
			Closest Reference	287.9843984	-12.01506155	12.01506155	4.01%
	400		Interpolation	395.3256776	-4.674322381	4.674322381	1.17%
			Closest Reference	395.3257288	-4.674271225	4.674271225	1.17%
	450		Interpolation	438.4827022	-11.51729784	11.51729784	2.56%
			Closest Reference	438.4827022	-11.51729784	11.51729784	2.56%
Neck	100, 300, 500	50	Interpolation	59.38973315	9.389733154	9.389733154	18.78%
			Closest Reference	59.38973315	9.389733154	9.389733154	18.78%
	150		Interpolation	134.6923642	-15.30763578	15.30763578	10.21%
			Closest Reference	128.4287807	-21.57121926	21.57121926	14.38%
	180		Interpolation	166.8679089	-13.13209106	13.13209106	7.30%
			Closest Reference	152.5295833	-27.47041671	27.47041671	15.26%
	200		Interpolation	185.2106101	-14.78938991	14.78938991	7.39%
			Closest Reference	165.3976395	-34.60236053	34.60236053	17.30%
	400		Interpolation	401.7427037	1.742703701	1.742703701	0.44%
			Closest Reference	397.5461693	-2.453830652	2.453830652	0.61%
	520		Interpolation	515.0182418	-4.981758225	4.981758225	0.96%
			Closet Reference	515.0182418	-4.981758225	4.981758225	0.96%
Wine	50, 150, 300	30	Interpolation	24.0456831	-5.954336902	5.954336902	19.85%
			Closest Reference	23.99039976	-6.009600241	6.009600241	20.03%
	100		Interpolation	72.36006898	27.63993102	27.63993102	27.64%
			Closest Reference	63.95622244	36.04377756	36.04377756	36.04%
	200		Interpolation	198.0913514	-1.908648585	1.908648585	0.95%
			Closest Reference	191.6948596	-8.305140411	8.305140411	4.15%
	250		Interpolation	253.8601594	3.860159448	3.860159448	1.54%
			Closest Reference	236.7957269	-13.20427308	13.20427308	5.28%
Statistical summary							
Average relative error (%)	Interpolation: 8.85%						
	Closest Reference: 11.48%						
Average absolute difference (ml)	Interpolation: 8.319123394	Max absolute difference = 27.63993102					
	Closest Reference: 13.40974115	Max absolute difference = 36.04377756					

Fig. 22 - Simulation results using k-means clustering on full size images (less efficient system design). Results are close to those of final system that applies k-means clustering on down sampled query images.

10.3 Code

Four code files can be found on the subsequent pages:

- system.py
- app.py
- index.html
- results.html

system.py

```
import cv2 as cv
import numpy as np
import os

#-------Display image method----------
def show(img):
    '''Show image using cv'''
    cv.imshow("Img",img)
    cv.waitKey(0)
    cv.destroyAllWindows()
    return

#-------Helper methods for main--------
def remove_small_areas(image):
    '''Return largest area for a given binary image.'''
    num_labels, labels, stats, centroids = cv.connectedComponentsWithStats(image)

    largest_label = 1+np.argmax(stats[1:,cv.CC_STAT_AREA])
    img_largest = np.zeros_like(image)
    img_largest[labels==largest_label] = 255

    return img_largest

def liquid_segmentation(img):
    '''Use k-means clustering to segment colored liquid from surroundings,
       assuming that the color of the liquid is dark and that the
       vessel is clear'''

    lab_image = cv.cvtColor(img, cv.COLOR_BGR2LAB)
    pixels = lab_image.reshape((-1,3)) # Reshape to 2D array of pixel
    pixels = np.float32(pixels) # Convert pixel values to floats

    # Criteria for kmeans clustering algorithm
    # (Criteria, iterations, epsilon convergence accuracy)

    criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 10, 0.1)

    # Run k-means clustering algorithm
    # Form 2 clusters -> assuming that the container is transparent
    # and the background is white

    downsampled_img = downsample_image(lab_image)
    downsampled_pixels = downsampled_img.reshape((-1,3)) # Reshape to 2D array of pixel
    downsampled_pixels = np.float32(downsampled_pixels) # Convert pixel values to floats

    _,labels,centers =
cv.kmeans(downsampled_pixels,2,None,criteria,10,cv.KMEANS_PP_CENTERS)

    centers = np.uint8(centers)
```

```
    segmented_image = centers[labels.flatten()]
    segmented_image = segmented_image.reshape(downsampled_img.shape)

    # Get the colors - there should be a total of 2
    unique_colors = np.unique(segmented_image.reshape(-1, segmented_image.shape[2]),
axis=0)

    # Find liquid color, assuming that the color with the higher value (darker) is the
liquid
    liquid_color = unique_colors[np.argmax(np.mean(unique_colors, axis=1))]

    binary_image = np.zeros(segmented_image.shape[:2], dtype=np.uint8)
    binary_image[(segmented_image == liquid_color).all(axis=2)] = 255 # determine pixels
with the liquid color

    liquid_image = remove_small_areas(cv.bitwise_not(binary_image))

    return liquid_image

def cfactor(pixelV,realV):
    '''Returns conversion factor between pixel and real liquid volumes'''
    return pixelV/realV

def downsample_image(img, max_dimension=1000):
    '''
    Downsample image. Used in k-means sampling based image segmentation.
    '''
    max_original_dimension = max(img.shape[:2])
    scale_factor = max_dimension / max_original_dimension
    scale_factor = min(1.0, scale_factor)  # Ensure scale_factor is not greater than 1

    return cv.resize(img, (0, 0), fx=scale_factor, fy=scale_factor,
interpolation=cv.INTER_AREA)

#-------Volume estimation functions-------
def vol_calc_disc(img):
    '''Determine volume of liquid using a binary silloheutte
    of the liquid (elevation view).

    img should be binary.
    The disk method is used to determine
    the overall volume. Each disk width is equivalent 1 pixel. Each row
    of pixels is thus treated as a cylindrical disk. '''

    total_volume = 0

    for y in range(img.shape[0]-1,-1,-1): # Iterate through each row of the image
        row = img[y,:]
        ones_indices = np.where(row==255)[0] # Find the pixels that are white - i.e.
represent liquid

        if ones_indices.size > 0: # If there are white pixels in a given row
            # Determine the leftmost and rightmost pixel indices and determine the
radius
            # Then use pi*r**2*h to determine the volume of each disk
```

```
            height = 1  # Since one pixel height rows
            leftmost = ones_indices[0]
            rightmost = ones_indices[-1]
            radius_pixels = (rightmost-leftmost)/2
            radius = radius_pixels
            disk_volume = np.pi * (radius**2)*height
            total_volume += disk_volume
    return total_volume

def find_closest_references(ref_pvol_cfs, query_pvol):
    '''
    Find the conversion factors for reference images with liquid
    volumes below and above the liquid's cubic pixel volume
    ref_pvol_cfs = List of ref. cubic volumes and cfs
    query_pvol = Query pixel volume
    '''

    ref_volumes_pixel = [volume_pixel for volume_pixel, cf in ref_pvol_cfs]
    ref_volumes_pixel.sort()

    below = None
    above = None

    if query_pvol <= ref_volumes_pixel[0]:
        above = ref_pvol_cfs[0]
    elif query_pvol >= ref_volumes_pixel[-1]:
        below = ref_pvol_cfs[-1]
    else:
        for idx, volume_pixel in enumerate(ref_volumes_pixel):
            if query_pvol >= volume_pixel:
                below_idx = idx
                below = ref_pvol_cfs[below_idx]
            if query_pvol <= volume_pixel:
                above_idx = idx
                above = ref_pvol_cfs[above_idx]
                break

    return below, above

def ref_factors(ref_imgs,ref_volumes):
    '''
    Compute conversion factors for a list of ref images and their
    corresponding ground truth volumes
    '''

    conversion_factors = []

    for ref_img, ref_volume in zip(ref_imgs,ref_volumes):
        ref_volume_pixel = vol_calc_disc(ref_img)
        cf = cfactor(ref_volume_pixel,ref_volume)
        conversion_factors.append((ref_volume_pixel,cf))

    return conversion_factors
```

```
def interpolate_cf(query_pvol,above,below):
    '''
    Interpolate the cf for the query pixel volume based
    on the conversion factors for reference bolumes
    below and above.
    '''
    if below is None:
        # If there is no reference below the query volume, use the closts reference's cf
        return above[1]
    else:
        weight = (query_pvol-below[0])/(above[0]-below[0])
        interpolated_cf = (1-weight)*below[1] + weight*(above[1])
    return interpolated_cf

#-----------------------------Main method-----------------------------
#---------------------------------------------------------------------
def main(ref_img_paths,ref_volumes,q_fpath,q_volume,interpolation=True):
    '''Function takes in the paths to reference and query images
        to compute the volume of the liquid being held within in the query image.
        Reference image volumes and needed for computation of conversion factors.
        Query volume is alsoo used to assess system accuracy.

        ref_img_paths = list of filepaths to references iamges
        ref_volumes = list of volumes corresponding to each image in ref_imgs
        q_fpath = filepath to query image as a string
        q_volume = volume of query as a numeric value
        interpolation = whether interpolation should be used or not
        '''

    #-----------------------1. Reference image processing--------------------
    #------------------------------------------------------------------------

    # Read in all images and filter them
    ref_imgs = [cv.bilateralFilter(cv.imread(os.path.abspath(path)),9,75,75)for path in
ref_img_paths]
    ref_liquid_imgs = [liquid_segmentation(img) for img in ref_imgs] # Segment out
liquid
    conversion_factors = ref_factors(ref_liquid_imgs,ref_volumes) # Compute cubic pixel
volumes and conversion factors
    conversion_factors = sorted(conversion_factors,key=lambda x:x[0]) # Sort the list by
cubic-pixel volume

    #-----------------------2. Query image processing-----------------------
    #-----------------------------------------------------------------------
    q_img = cv.imread(os.path.abspath(q_fpath))
    q_filtered_img = cv.bilateralFilter(q_img,9,75,75)
    q_liquid_img = liquid_segmentation(q_filtered_img)
    q_vol = vol_calc_disc(q_liquid_img)
    vol_cf_below, vol_cf_above = find_closest_references(conversion_factors,q_vol)

    if interpolation and vol_cf_below is not None and vol_cf_above is not None:
        f = interpolate_cf(q_vol, vol_cf_above, vol_cf_below)
    else:
        if vol_cf_below is None:
```

```
            f = vol_cf_above[1]
        else:
            f = vol_cf_below[1]

    q_vol_estimate_pixel = vol_calc_disc(q_liquid_img)
    estimation = q_vol_estimate_pixel/f

    #----------------------3.Display results---------------------------------
    #------------------------------------------------------------------------
    relative_diff = -(1-(estimation/q_volume))
    abs_diff = abs(estimation-q_volume)
    print('Query true volume: ',q_volume)
    print('Estimated volume: ',estimation)
    print('Relative Difference (%): ', relative_diff)
    print('Absolute Difference (ml):', abs_diff)
    return estimation,relative_diff, abs_diff

#----------------------------Main for app--------------------------
#------------------------------------------------------------------
def app_main(ref_img_paths,ref_volumes,q_fpath,q_volume,interpolation=True):
    ''' Application version of main()
        Function takes in the paths to reference and query images
        to compute the volume of the liquid being held within in the query image.
        Reference image volumes and needed for computation of conversion factors.
        Query volume is alsoo used to assess system accuracy.

        ref_img_paths = list of filepaths to references iamges
        ref_volumes = list of volumes corresponding to each image in ref_imgs
        q_fpath = filepath to query image as a string
        q_volume = volume of query as a numeric value
        interpolation = whether interpolation should be used or not
        '''

    #----------------------1. Reference image processing--------------------
    #-----------------------------------------------------------------------

    # Read in all images and filter them
    ref_imgs = [cv.bilateralFilter(cv.imread(os.path.abspath(path)),9,75,75)for path in
ref_img_paths]
    ref_liquid_imgs = [liquid_segmentation(img) for img in ref_imgs] # Segment out
liquid

    conversion_factors = ref_factors(ref_liquid_imgs,ref_volumes) # Compute cubic pixel
volumes and conversion factors
    conversion_factors = sorted(conversion_factors,key=lambda x:x[0])# Sort the list by
cubic-pixel volume

    #----------------------2. Query image processing-----------------------
    #-----------------------------------------------------------------------
    q_img = cv.imread(os.path.abspath(q_fpath))
    q_filtered_img = cv.bilateralFilter(q_img,9,75,75)
    q_liquid_img = liquid_segmentation(q_filtered_img)
    q_vol = vol_calc_disc(q_liquid_img)
    vol_cf_below, vol_cf_above = find_closest_references(conversion_factors,q_vol)
```

```
    if interpolation and vol_cf_below is not None and vol_cf_above is not None:
        f = interpolate_cf(q_vol, vol_cf_above, vol_cf_below)
    else:
        if vol_cf_below is None:
            f = vol_cf_above[1]
        else:
            f = vol_cf_below[1]

    q_vol_estimate_pixel = vol_calc_disc(q_liquid_img)
    estimation = q_vol_estimate_pixel/f

    #----------------------3.Display results---------------------------------
    #------------------------------------------------------------------------
    relative_diff = -(1-(estimation/q_volume))
    abs_diff = abs(estimation-q_volume)

    return ref_liquid_imgs,q_liquid_img,estimation,relative_diff,abs_diff


#-------------------------Simulation method-------------------------
#-------------------------------------------------------------------
def reference_processing(ref_img_paths,ref_volumes):
    '''
    Processing reference images and computing conversion factors
    '''
    ref_imgs = [cv.imread(os.path.abspath(path)) for path in ref_img_paths]
    ref_liquid_imgs = [liquid_segmentation(img) for img in ref_imgs] # Segment out
liquid
    conversion_factors = ref_factors(ref_liquid_imgs,ref_volumes) # Compute cubic pixel
volumes and conversion factors
    return conversion_factors

def query_processing(q_fpath,q_volume,conversion_factors,interpolation=True):
    '''
    Processing query
    '''
    q_img = cv.imread(os.path.abspath(q_fpath))
    q_filtered_img = cv.bilateralFilter(q_img,9,75,75)
    q_liquid_img = liquid_segmentation(q_filtered_img)
    q_vol = vol_calc_disc(q_liquid_img)

    vol_cf_below, vol_cf_above = find_closest_references(conversion_factors,q_vol)

    if interpolation and vol_cf_below is not None and vol_cf_above is not None:
        f = interpolate_cf(q_vol, vol_cf_above, vol_cf_below)
    else:
        if vol_cf_below is None:
            f = vol_cf_above[1]
        else:
            f = vol_cf_below[1]

    q_vol = vol_calc_disc(q_liquid_img)
    estimation = q_vol/f
```

```
    return estimation

def run_simulation(containers):
    '''
    Function to run simulation. Results outputted to
    simulation_results.txt in the same directory.
    Function takes in a dictionary as its input:
    For each k:v pair
        k = container type
        v = Tuple(ref_img_paths,ref_volumes,q_img_paths,q_volumes)
    '''
    with open("simulation_results.txt", "w") as outfile:
        for container_type, (ref_img_paths, ref_volumes, query_img_paths, query_volumes)
in containers.items():

            references = reference_processing(ref_img_paths,ref_volumes)
            # Compute reference conversion factors once for each container type for
efficiency

            for q_fpath, q_volume in zip(query_img_paths, query_volumes):
                outfile.write(f"Container type: {container_type}\n")
                outfile.write(f"Query: {q_fpath}\n")
                outfile.write(f"Ground truth volume: {q_volume}\n")

                for interpolation in [True, False]: # Run iteration on both systems:
interpolation and closest method
                    method = 'Interpolation' if interpolation else 'Closest reference'
                    outfile.write(f"{method} method\n")
                    estimation = query_processing(q_fpath, q_volume,
references,interpolation)
                    error = abs(estimation - q_volume) / q_volume

                    outfile.write(f"Estimated volume: {estimation}\n")
                    outfile.write(f"Difference: {estimation-q_volume}\n")
                    outfile.write(f"Relative error: {error:.2%}\n")
                outfile.write("\n")

            outfile.write("-----------------------------------------------\n")

if __name__ == '__main__':

    # Image paths for simulation
    containers = {
        "juice":(['./final_images/juice/50.jpg',
                  './final_images/juice/200.jpg',
                  './final_images/juice/350.jpg'
                 ],
                 [50,200,350],
                 ['./final_images/juice/30.jpg',
                  './final_images/juice/100.jpg',
                  './final_images/juice/150.jpg',
                  './final_images/juice/250.jpg',
                  './final_images/juice/300.jpg',
                  './final_images/juice/400.jpg',
```

```
                './final_images/juice/450.jpg'
                ],
                [30,100,150,250,300,400,450]
                ),
    "neck":(['./final_images/neck/100.jpg',
             './final_images/neck/300.jpg',
             './final_images/neck/500.jpg',
             ],
             [100,300,500],
             ['./final_images/neck/50.jpg',
              './final_images/neck/150.jpg',
              './final_images/neck/180.jpg',
              './final_images/neck/200.jpg',
              './final_images/neck/400.jpg',
              './final_images/neck/520.jpg'
             ],
             [50,150,180,200,400,520]
             ),
    "wine":(['./final_images/wine/50.jpg',
             './final_images/wine/150.jpg',
             './final_images/wine/300.jpg'
             ],
             [50,150,300],
             ['./final_images/wine/30.jpg',
              './final_images/wine/100.jpg',
              './final_images/wine/200.jpg',
              './final_images/wine/250.jpg'
             ],
             [30,100,200,250]
            )
}

run_simulation(containers) # Running simulation
```

app.py

```
from flask import Flask, render_template, request, abort, send_from_directory
import os
from system import app_main
import shutil
import cv2 as cv

app = Flask(__name__)

#------------------Application configuration-----------------------
#------------------------------------------------------------------
app.config['STATIC'] = 'static/'
app.config['UPLOAD_FOLDER'] = os.path.join(app.config['STATIC'], 'upload/')
app.config['REF_FOLDER'] = os.path.join(app.config['UPLOAD_FOLDER'], 'references/')
app.config['QUERY_FOLDER'] = os.path.join(app.config['UPLOAD_FOLDER'], 'query/')

app.config['BINARY'] = os.path.join(app.config['STATIC'],'binary/')
app.config['REF_BIN'] = os.path.join(app.config['BINARY'],'references_binary/')
app.config['QUERY_BIN'] = os.path.join(app.config['BINARY'],'queries_binary/')

#------------------Application helper methods----------------------
#------------------------------------------------------------------
def delete_files_in_directory(directory):
    '''Delete all preexisting files in a given directory'''
    for filename in os.listdir(directory):
        file_path = os.path.join(directory, filename)
        try:
            if os.path.isfile(file_path) or os.path.islink(file_path):
                os.unlink(file_path)
            elif os.path.isdir(file_path):
                shutil.rmtree(file_path)
        except Exception as e:
            print(f'Failed to delete {file_path}. Reason: {e}')

def get_image_paths(folder):
    '''Gets paths of all images in folder'''
    image_extensions = ('.jpg', '.jpeg', '.png')
    filenames = os.listdir(folder)
    image_paths = [os.path.join(folder, filename) for filename in filenames if
filename.lower().endswith(image_extensions)]
    return image_paths

def is_valid_image(file):
    '''Check if uploaded file is valid'''
    allowed_mimetypes = {'image/jpeg', 'image/png', 'image/jpg'}
    return file.mimetype in allowed_mimetypes

#------------------Main application methods------------------------
#------------------------------------------------------------------

@app.route('/')
```

```
def index():
    # Render template
    return render_template('index.html')

@app.route('/binary/<path:path>')
def serve_binary_images(path):
    return send_from_directory('static/binary', path)

@app.route('/submit', methods=['POST'])
def submit():
    '''
    Submission handler:
    1. Intializaiton - removal of files from preexisting system run
    2. File and input retrieval from front-end
    3. main() to estimate volume in query
    4. Results processing
    5. Rendering results.html page
    '''
    if request.method == 'POST':

        # 1. Initialization - create directories if they do not exist

        if not os.path.exists(app.config['UPLOAD_FOLDER']):
            os.makedirs(app.config['UPLOAD_FOLDER'])

        if not os.path.exists(app.config['REF_FOLDER']):
            os.makedirs(app.config['REF_FOLDER'])

        if not os.path.exists(app.config['QUERY_FOLDER']):
            os.makedirs(app.config['QUERY_FOLDER'])

        if not os.path.exists(app.config['BINARY']):
            os.makedirs(app.config['BINARY'])

        if not os.path.exists(app.config['REF_BIN']):
            os.makedirs(app.config['REF_BIN'])

        if not os.path.exists(app.config['QUERY_BIN']):
            os.makedirs(app.config['QUERY_BIN'])

        # Deleting files from previous run
        delete_files_in_directory(app.config['REF_FOLDER'])
        delete_files_in_directory(app.config['QUERY_FOLDER'])
        delete_files_in_directory(app.config['REF_BIN'])
        delete_files_in_directory(app.config['QUERY_BIN'])

        # 2. Retrieve files from front-end submission and store

        ref_images = request.files.getlist('ref_image[]')  # Use getlist() to handle
multiple files
        ref_volumes = request.form.getlist('ref_volume[]')  # Use getlist() to handle
multiple volumes

        query_image = request.files['query_image']
```

```
        query_volume = request.form['query_volume']

        # Convert all numerical inputs for ref_images and check if numerical inputs are
valid
        for i in range(len(ref_volumes)):
            try:
                ref_volumes[i] = float(ref_volumes[i])
                query_volume = float(query_volume)
            except ValueError:
                abort(400, description="Invalid reference/query volume (ml) value. It
should be a float or an integer.")

        # Checking if reference file uploads are valid. If so, store as img, volume
pairs
        image_volume_pairs = []
        for ref_image, ref_volume in zip(ref_images, ref_volumes):

            if not is_valid_image(ref_image):
                abort(400, description="Invalid reference image format.")

            with open(os.path.join(app.config['REF_FOLDER'], ref_image.filename), 'wb')
as f:
                shutil.copyfileobj(ref_image.stream, f)

            image_path = os.path.join(app.config['REF_FOLDER'], ref_image.filename)
            image_volume_pairs.append((image_path, ref_volume))

        if not is_valid_image(query_image):
            abort(400, description="Invalid query image format.")

        with open(os.path.join(app.config['QUERY_FOLDER'], query_image.filename), 'wb')
as f:
            shutil.copyfileobj(query_image.stream, f)

        query_image_path = get_image_paths(app.config['QUERY_FOLDER'])[0]

        ref_images_paths = [path for path,volume in image_volume_pairs]
        ref_volumes = [volume for path ,volume in image_volume_pairs]

        # 3. Calling main system
        ref_liquid_images_list, q_liquid_image, estimation, relative_diff, abs_diff =
app_main(ref_images_paths,ref_volumes,query_image_path,query_volume)

        # 4. Results processing

        # Storing reference images
        for i, (b_img, volume) in enumerate(zip(ref_liquid_images_list, ref_volumes)):
            output_path =
os.path.join(app.config['REF_BIN'],f'ref_liquid_image_{i}.jpg')
            cv.imwrite(output_path,b_img)

        # Storing query image
        q_output_path = os.path.join(app.config['QUERY_BIN'],f'q_liquid_image.jpg')
        cv.imwrite(q_output_path,q_liquid_image)
```

```
        # Get image paths
        ref_bimages_paths = [os.path.join(app.config['REF_BIN'],
f'ref_liquid_image_{i}.jpg') for i in range(len(ref_volumes))]
        query_bimage_path = get_image_paths(app.config['QUERY_BIN'])[0]

        # Convert absolute paths to relative paths
        ref_bimages_paths = [os.path.relpath(path, app.config['STATIC']) for path in
ref_bimages_paths]
        query_bimage_path = os.path.relpath(query_bimage_path, app.config['STATIC'])

        # 5. Rendering results.html
        return render_template('results.html', ref_liquid_images=ref_bimages_paths,
q_liquid_image=query_bimage_path,
                               estimation=estimation, relative_diff=relative_diff,
abs_diff=abs_diff,
                               ref_volumes = ref_volumes,query_volume=query_volume)

if __name__ == '__main__':
    app.run(debug=True)
```

templates/index.html

```
<!--index.html-->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Liquid Volume Estimation</title>
    <style>
        body {
            margin: 0;
            padding: 20px;
            font-family: Arial, sans-serif;
            font-size: 16px;
            color: #333;
        }
        h1, h2, p {
            margin-top: 0;
        }
        h1 {
            font-size: 36px;
            font-weight: bold;
            color: #0072c6;
        }
        h2 {
            font-size: 24px;
            font-weight: bold;
            color: #333;
        }
        p {
            font-size: 16px;
            line-height: 1.5;
        }
        label {
            font-weight: bold;
            margin-right: 10px;
        }
        input[type="file"] {
            margin-bottom: 10px;
        }
        input[type="text"] {
            margin-bottom: 20px;
            padding: 5px;
            border-radius: 5px;
            border: 1px solid #ccc;
        }
        button {
            background-color: #0072c6;
            color: #fff;
            border: none;
            padding: 10px;
            border-radius: 5px;
```

```
            cursor: pointer;
        }
        button:hover {
            background-color: #005a9e;
        }
        input[type="submit"] {
            background-color: #0072c6;
            color: #fff;
            border: none;
            padding: 10px;
            border-radius: 5px;
            cursor: pointer;
        }
        input[type="submit"]:hover {
            background-color: #005a9e;
        }
        .container {
            display: flex;
            flex-wrap: wrap;
            justify-content: space-between;
            align-items: center;
            margin-top: 30px;
        }
        .input-container {
            width: 45%;
            margin-bottom: 20px;
        }
    </style>
    <script>
        function addReferenceInput() {
            const refContainer = document.getElementById('ref_container');
            const newRefInput = document.createElement('div');
            newRefInput.classList.add('input-container');
            newRefInput.innerHTML = `
                <label for="ref_image">Reference Image:</label>
                <input type="file" name="ref_image[]" required><br>
                <label for="ref_volume">Reference Volume:</label>
                <input type="text" name="ref_volume[]" required><br>
            `;
            refContainer.appendChild(newRefInput);
        }
    </script>
</head>
<body>
    <h1>Liquid Volume Estimation</h1>
    <p>
        Estimation of volumes of colored liquids in rotationally symmetric
        transparent colorless containers.
        <br><br>
        Upload at least 1 reference image with its known volume.
        <br>
        Upload the query image with its known volume. <br><br>
        <i>NOTE: The query volume is not used in decision-making. It is only used
        for error computation.</i>
```

```
        <br><br>
        Supported file types: .jpg,.jpeg,.png
        <br><br>
    </p>
    <form action="{{ url_for('submit') }}" method="POST" enctype="multipart/form-data">
    <table>
        <tr>
            <div id="ref_container">
                <td>
                    <label for="ref_image">Reference Image:</label>
                    <input type="file" name="ref_image[]" required><br>
                    <label for="ref_volume">Reference Volume (ml):</label>
                    <input type="text" name="ref_volume[]" required><br>
                <div>
                    <button type="button" onclick="addReferenceInput()">Add another
reference</button><br>
                    <p></p>
                </div>
                </td>
            </div>

            </div>

            <td>
                <label for="query_image">
                    <label for="query_image">Query Image:</label>
                    <input type="file" name="query_image" required><br>
                    <label for="query_volume">Query Volume (ml):</label>
                    <input type="text" name="query_volume" required><br>
                    <input type="submit" value="Submit">
                </label>

            </td>
        </tr>
    </table>
    </form>
</body>
</html>

<!-- There appears to be a disjunct between teh received image inputs and the volumes -
need to sort internall to prevent errors -->
<!-- Likely due to the use of inserting new references that mess up the linking -->
<!-- Consider downsampling the image to increase efficienty - currently runs too slowly-
->
```

templates/results.html

```
<!--results.html-->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Results</title>
    <style>
        body {
            margin: 0;
            padding: 20px;
            font-family: Arial, sans-serif;
            font-size: 16px;
            color: #767676;
        }
        h1, h2, p {
            margin-top: 0;
        }
        h1 {
            font-size: 36px;
            font-weight: bold;
            color: #005a9e;
        }
        h2 {
            font-size: 24px;
            font-weight: bold;
            color: #005a9e;
        }
        p {
            font-size: 16px;
            line-height: 1.5;
        }
        .results {
            display: flex;
            flex-wrap: wrap;
            justify-content: space-between;
            align-items: center;
            margin-top: 30px;
        }
        .result {
            width: 30%;
            margin-bottom: 30px;
        }
        .result img {
            max-width: 100%;
            height: auto;
        }
        .back-link {
            display: block;
            margin-top: 30px;
            font-weight: bold;
```

```
            color: #0072c6;
        }
        table {
            border-collapse: collapse;
            width: 100%;
            margin-top: 30px;
        }
        th, td {
            text-align: left;
            padding: 8px;
            border-bottom: 1px solid #ddd;
        }
        th {
            background-color: #f2f2f2;
        }
    </style>
</head>
<body>
    <h1>Results:</h1>
    <div class="results">
        <div class="result">
            <h2>Estimation (ml)</h2>
            <p>{{ estimation }}</p>
        </div>
        <div class="result">
            <h2>Relative Difference (%)</h2>
            <p>{{ relative_diff }}</p>
        </div>
        <div class="result">
            <h2>Absolute Difference (ml)</h2>
            <p>{{ abs_diff }}</p>
        </div>
        <div class="result">
            <h2>Query Image (Binary Intermediate)</h2>
            <table>
                <tr>
                    <th>Image</th>
                    <th>Volume (ml)</th>
                </tr>
                <tr>
                    <td><img src="{{ q_liquid_image }}" alt="Query Liquid Image"></td>
                    <td>{{query_volume}}</td>
                </tr>
            </table>
        </div>
        <div class="result">
            <h2>Reference Images (Binary Intermediates)</h2>
            <table>
                <tr>
                    <th>Image</th>
                    <th>Volume (ml)</th>
                </tr>
```

```
                {% for i in range(ref_liquid_images|length) %}
                    <tr>
                        <td><img src="{{ ref_liquid_images[i] }}" alt="Reference Liquid
Image"></td>
                        <td>{{ref_volumes[i]}}</td>
                    </tr>
                {% endfor %}
            </table>
        </div>
    </div>
    <a href="{{ url_for('index') }}" class="back-link">Back to Home</a>
</body>
</html>
```