UNIVERSITÀ
DI TRENTO

SIV, IMAGE AND VIDEO

Liquid Level Recognition

256180 - Alessandro Moscatelli

257565 - Donald Gera

Contents

1 Introduction

1.1 Application domain and Attacked problem

Problem statement:
The problem we aimed to tackle is the accurate **measurement of liquid height in a transparent glass**, based only on an image.

Traditional measurement methods rely on direct physical contact, but in scenarios where automation, remote estimation, or digital analysis is required, a computer vision-based approach becomes essential. This study focuses on developing a **non-intrusive method**[1] to estimate the liquid level in a glass, which can be applicable in various domains such as:

- **Food and Beverage Industry:** Automated quality control in beverage filling lines.
- **Healthcare and Nutrition:** Monitoring fluid intake for medical applications.
- **Smart Kitchens:** Estimating liquid levels in smart cooking and IoT-enabled devices.

To address this challenge, we began by creating a dataset of proper images and developed an intuitive initial approach, which we progressively refined by making adjustments based on the issues we encountered.

1.2 Dataset

To validate and enhance our approach for liquid height measurement, we created a dataset of images capturing glasses with different liquid types. The dataset follows a structured naming convention in the format $X - Y$, where:

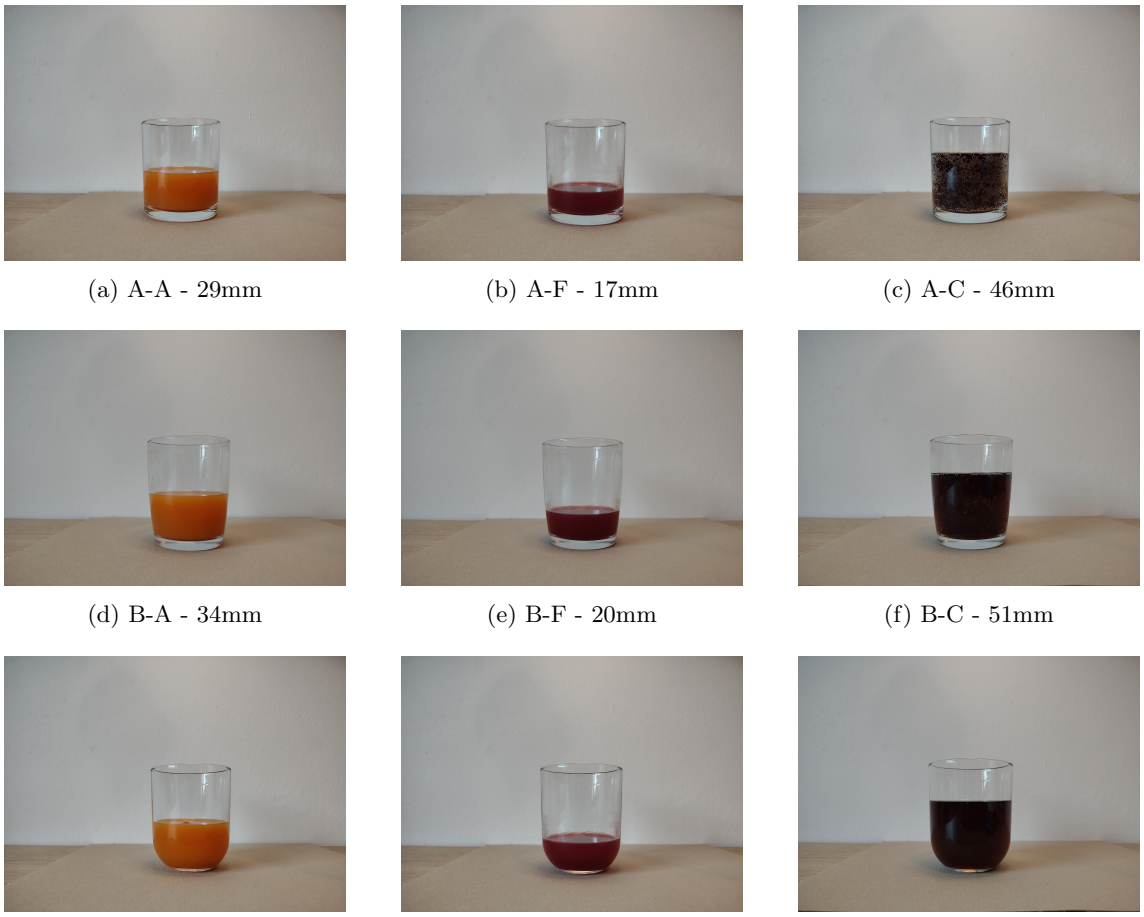
- The **first letter (X)** represents the type of glass used:
  - **A**: Straight glass
  - **B**: Slightly rounded glass
  - **C**: Rounded glass
- The **second letter (Y)** indicates the type of liquid in the glass:
  - **A**: Ace juice
  - **F**: Red fruit juice
  - **C**: Coca Cola

This allowed us to systematically analyze how different glass shapes and liquid colors affect the measurement accuracy.

1.2.1 Example Images from the Dataset

To illustrate the dataset structure, we present example images for different combinations of glass types and liquids, with the height of the liquid on the side.

[1] A non-intrusive method refers to a technique that does not physically interact with or alter the object being measured. In the context of liquid height measurement, this means determining the height of the liquid in a glass without inserting sensors, probes, or any physical measuring tools into the liquid.



(a) A-A - 29mm (b) A-F - 17mm (c) A-C - 46mm

(d) B-A - 34mm (e) B-F - 20mm (f) B-C - 51mm

(g) C-A - 38mm (h) C-F - 24mm (i) C-C - 53mm

Figure 1: Example images from the dataset, showcasing different glass types and liquid contents.

1.2.2 Precautions for High-Quality Data Collection

To ensure the dataset is well-suited for our task, several precautions were taken during the data collection process:

- **Consistent Camera Angles:** Each image was captured as close as possible to a **90-degree perpendicular view** relative to the glass. This minimizes distortion and perspective errors, ensuring that height measurements remain accurate.
- **Uniform Lighting Conditions:** Proper lighting was used to reduce shadows and reflections, which could interfere with contour detection and edge segmentation.
- **Neutral Background Selection:** A plain background was chosen to make object detection easier.
- **Multiple Liquid Types for Robustness:** Different liquid colors and transparencies were tested to analyze how variations in **opacity and contrast** impact segmentation accuracy.
- **Consistent Distance and Framing:** The camera was kept at a **fixed distance** from the glass, ensuring a consistent pixel-to-metric ratio across images.

2 Techniques implemented

2.1 Pixels per Metric Ratio

As we anticipated, in the introduction the problem we tackled is the accurate measurement of liquid height in a transparent glass. To address this challenge, we started with the most straightforward and intuitive approach, that consisted of leveraging the known dimension (height) of the glass in order to use it as a reference object and establish a **pixels per metric (PPM) ratio** [1] that allows us to convert pixel-based measurements into real-world dimensions.

2.1.1 Pixels per Metric Ratio - Formalization

A fundamental step in our approach is computing the *pixels per metric ratio*, which acts as a scaling factor to transform pixel-based measurements into physical units (such as millimeters or inches). This calibration technique follows a straightforward principle:

1. **Reference Object Selection:** To determine the size of an object in an image, we need a known reference object with two essential properties:
   - **Known dimensions:** The height (or width) of the reference object must be predefined in a measurable unit.
   - **Unique identification:** The object should be easy to locate within the image, either by its placement (e.g., always positioned at a specific location) or by its distinctive shape or color.
2. **Computing the Ratio:** Given an image of the glass, we first detect and measure the glass's total height in pixels. Since we already know the glass's actual height in millimeters, we compute the **pixels per metric ratio** as:

$$\text{PPM} = \frac{\text{Height of Glass in Pixels}}{\text{Height of Glass in mm}} \tag{1}$$

   Once the ratio is established, we can apply it to measure other elements within the image, including the height of the liquid inside the glass.
3. **Estimating the Liquid Level:** By identifying the liquid level in the glass and measuring its height in pixels, we convert it into real-world dimensions using:

$$\text{Liquid Height (mm)} = \frac{\text{Liquid Height (pixels)}}{\text{PPM}} \tag{2}$$

This method provides a simple yet effective approach to calibrating pixel distances and estimating liquid height by using the known glass height as a reference. Serving as a foundational technique, it was further refined through improved contour detection, enhanced segmentation, noise reduction and camera calibration, all of which contributed to achieving greater accuracy in the measurement.

2.1.2 Pseudocode

Algorithm 1 Liquid Height Calculation Algorithm

function CalculateLiquidHeight($image\_path,\ H^{mm}_{glass}$)
  $Image \leftarrow$ LoadImage($image\_path$)
  $GrayImage \leftarrow$ ConvertToGrayscale($Image$)
  $BlurredImage \leftarrow$ ApplyGaussianBlur($GrayImage$)
  $M_x \leftarrow$ ComputeEdges(G, direction = `'x'`)
  $M_y \leftarrow$ ComputeEdges(G, direction = `'y'`)
  $M \leftarrow$ ComputeMagnitude($M_x,\ M_y$)
  $E \leftarrow$ ThresholdImage(M)
  $C \leftarrow$ FindContours(E)
  $C* \leftarrow$ FilterContours(C)
  $H_{convex} \leftarrow$ ComputeConvexHull(C')
  $(x_G,\ y_G,\ w_G,\ h_G) \leftarrow$ BoundingBox(H_{convex})
  $H^{px}_{glass} \leftarrow h_G$
  $ROI \leftarrow$ ExtractROI($BlurredImage,\ (x_G,\ y_G,\ w_G,\ h_G)$)
  $L \leftarrow$ ThresholdImage(ROI)
  $C_{liquid} \leftarrow$ FindContours(L)
  $C^*_{liquid} \leftarrow$ SelectLargestContour(C_{liquid})
  $(x_L,\ y_L,\ w_L,\ h_L) \leftarrow$ BoundingBox(C^*_{liquid})
  $H^{px}_{liquid} \leftarrow h_L$
  $\alpha \leftarrow \frac{H^{px}_{glass}}{H^{mm}_{glass}}$
  $H^{mm}_{liquid} \leftarrow \frac{H^{px}_{liquid}}{\alpha}$
  return H^{mm}_{liquid}
end function

Description: This algorithm estimates the height of liquid in a glass from a single image by leveraging the known physical height of the glass as a calibration reference. Initially, the image is converted to grayscale and a Gaussian blur is applied to reduce noise and smooth the image. This pre-processing step simplifies the scene, making it easier to detect the edges. Next, the algorithm uses edges detection operators to compute the gradients in both the horizontal and vertical directions. By combining these gradients, a gradient magnitude image is obtained, which highlights the edges in the scene. A threshold is then applied to this image to produce a binary edge map, where the significant boundaries of the glass are clearly delineated. After detecting the edges, the algorithm locates the glass by identifying and filtering contours within the binary image. Irrelevant or small contours are discarded, and the remaining contours are merged to form a robust representation of the glass boundary. To ensure a smooth and continuous outline, a convex hull [2] is computed around these contours, and the resulting bounding box of this hull provides a measure of the glass height in pixels. With the glass successfully isolated, the next step focuses on segmenting the liquid region inside it. The region of interest corresponding to the glass is extracted, and a thresholding method is applied to differentiate the liquid from the rest of the glass. The largest detected contour in this thresholded region is assumed to represent the liquid, and its vertical extent is measured in pixels. Finally, the known physical height of the glass is used to compute a scale factor (pixels per millimeter), which allows the pixel measurement of the liquid's height to be converted into a real-world value in millimeters. In this way, the algorithm translates the pixel-based measurement into an accurate physical measurement of the liquid level.

2.1.3 Problems

Adopting this approach without refinements, however, results in a significant margin of error compared to the true measurements, leading to inaccuracies in detecting both the liquid level and the glass boundaries. There are different reasons for this:

1. Even though we tried to capture the images at a **90-degree angle**, slight deviations were inevitable, meaning the camera was not always perfectly perpendicular to the glass. Without

[2] A convex hull is the smallest convex shape that fully contains a given set of points.

this ideal condition, both the glass and the liquid it contains may appear distorted, impacting measurement accuracy.

2. The images contain **noise** in the form of background elements such as the table, shadows, and other visual artifacts. These unwanted elements are affecting the detection of the glass.

3. No **intrinsic** or **extrinsic camera calibration** was performed. Without accounting for these parameters, the images are susceptible to radial and tangential lens distortion, which can alter the perceived dimensions of objects.

Solutions:

- **Camera Calibration:** Measurement accuracy can be significantly improved by performing a proper camera calibration, which involves computing both the **intrinsic** and **extrinsic** parameters of the camera. This step helps correct distortions caused by lens properties and perspective misalignment, ensuring that the captured images provide a more accurate representation of real-world dimensions.

- **Noise Reduction and Background Filtering:** To mitigate unwanted noise, such as table surfaces, shadows, and reflections, which can interfere with accurate object segmentation, we applied **background subtraction techniques** to minimize artifacts and enhance object visibility.

  Additionally, to improve edge detection and contour extraction, we employed several image processing techniques, which are briefly outlined below:

  - **Otsu's Thresholding**: An automatic thresholding method that separates the foreground from the background by minimizing the intra-class variance between the two regions.
  - **Adaptive Thresholding**: Instead of using a fixed threshold, we computed threshold values dynamically based on percentile selection. This approach helps retain only significant edges, improving contour extraction by filtering out unnecessary details.
  - **Morphological Operations**: These techniques refine object boundaries by removing small noise artifacts and filling gaps in detected contours. Specifically, two operations are defined:
    1. **Opening**: A combination of erosion followed by dilation, effectively eliminating small noise while preserving object structure.
    2. **Closing**: A combination of dilation followed by erosion, useful for bridging small gaps in detected edges and improving contour completeness.

2.2 Camera Calibration

The process of determining a camera's parameters is known as **camera calibration**. This involves estimating all the necessary **intrinsic and extrinsic parameters** required to establish an accurate relationship between a **3D point in the real world** and its corresponding **2D projection (pixel)** in an image captured by the calibrated camera. [2]
Camera calibration typically involves recovering two main types of parameters:

- **Intrinsic Parameters:** These describe the internal characteristics of the camera and lens system, including:
  - **Focal length** (f_x, f_y)
  - **Optical center** (c_x, c_y)
  - **Radial and tangential distortion coefficients** of the lens

- **Extrinsic Parameters:** These define the camera's position and orientation relative to a fixed world coordinate system, consisting of:
  - **Rotation matrix R**
  - **Translation vector t**

2.3 Geometry of Image Formation

When a 3D point in the real world is projected onto a 2D image plane, it must first be transformed from the **world coordinate system** to the **camera coordinate system** using the **extrinsic parameters** $(\mathbf{R}, \mathbf{t})$. This transformation is represented as:

$$\mathbf{P}_c = \mathbf{R}\mathbf{P}_w + \mathbf{t} \tag{3}$$

where:

- $\mathbf{P}_w = (X_w, Y_w, Z_w)$ represents the point in world coordinates.
- $\mathbf{P}_c = (X_c, Y_c, Z_c)$ represents the transformed point in camera coordinates.
- $\mathbf{R}$ is the 3×3 rotation matrix.
- $\mathbf{t}$ is the 3×1 translation vector.

Once the point is expressed in camera coordinates, it is projected onto the image plane using the intrinsic camera parameters, completing the mapping from 3D space to 2D image pixels. The details of this projection process are elaborated in the following subsections.

2.3.1 Projection of a 3D Point onto the Image Plane

Using the **intrinsic parameters** of the camera, we project a 3D point onto the image plane. The equations that relate a 3D point (X_w, Y_w, Z_w) in world coordinates to its corresponding projection (u, v) in the image coordinates are given by:

$$\begin{bmatrix} u' \\ v' \\ z' \end{bmatrix} = \mathbf{P} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{4}$$

where the final image coordinates (u, v) are computed as:

$$u = \frac{u'}{w'} \tag{5}$$

$$v = \frac{v'}{w'} \tag{6}$$

2.3.2 Camera Projection Matrix

The matrix $\mathbf{P}$ is a 3×4 **projection matrix** consisting of two main components:

- **Intrinsic Matrix** ($\mathbf{K}$): Contains the camera's internal parameters.
- **Extrinsic Matrix** ($[\mathbf{R} \mid \mathbf{t}]$): Represents the camera's position and orientation, combining the 3×3 rotation matrix $\mathbf{R}$ and the 3×1 translation vector $\mathbf{t}$.

Thus, the projection matrix is given by:

$$\mathbf{P} = \underbrace{\mathbf{K}}_{\text{Intrinsic Matrix}} \times \underbrace{[\mathbf{R} \mid \mathbf{t}]}_{\text{Extrinsic Matrix}} \tag{7}$$

2.3.3 Intrinsic Camera Matrix

The intrinsic matrix $\mathbf{K}$ is an upper triangular matrix, defined as:

$$\mathbf{K} = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{8}$$

where:

- f_x, f_y are the focal lengths in the x and y directions (often assumed to be equal).
- c_x, c_y are the coordinates of the optical center (principal point) in the image plane.
- γ represents the **skew** between the x and y axes (usually 0).

The intrinsic matrix **K** defines the transformation from **camera coordinates** to **image pixel coordinates**.

2.4 Camera Calibration Step by Step using OpenCV

2.4.1 Step 1: Defining Real-World Coordinates with a Checkerboard Pattern

Our world coordinates are defined using a **checkerboard pattern** that is attached to a wall or a fixed surface. The 3D points are the corners of the squares in the checkerboard.
Any corner of the checkerboard can be chosen as the **origin** of the world coordinate system. The $\mathrm{X_w}$ and $\mathrm{Y_w}$ axes are aligned along the plane of the checkerboard, while the $\mathrm{Z_w}$ axis is perpendicular to it. Since all points on the checkerboard lie in a single plane, we can assume:

$$Z_w = 0 \tag{9}$$

This assumption simplifies calibration by treating all points as belonging to the XY-plane.

Calibration Process Using the Checkerboard:
During calibration, we estimate the camera parameters using a set of known 3D points (X_w, Y_w, Z_w) and their corresponding 2D pixel locations (u, v) in the image. To obtain these 3D points, we photograph a checkerboard pattern with known dimensions at multiple orientations. And as already said, since the checkerboard is a planar surface, we can arbitrarily choose $\mathrm{Z_w} = 0$ for every point. Given that the squares are equally spaced, the ($\mathrm{X_w}$, $\mathrm{Y_w}$) coordinates of each 3D point can be defined by selecting a single reference point at (0,0) and computing the remaining points relative to it.

Why is the Checkerboard Pattern Used for Calibration?
Checkerboard patterns are widely used in camera calibration due to their **distinct and easily detectable corners**. The corners of the squares serve as high-contrast reference points that are easy to locate, as they exhibit **sharp gradients** in both horizontal and vertical directions. Additionally, these corners are **stable** because they lie at the intersections of checkerboard lines, ensuring precise detection. These properties make the checkerboard a **robust and reliable** tool for camera calibration.

2.4.2 Step 2: Capturing Multiple Images of the Checkerboard from Different Viewpoints

To accurately calibrate the camera, we capture multiple images of the checkerboard pattern from different viewpoints. There are two common methods to introduce perspective variations for calibration:

- **Moving the Camera:** The checkerboard is kept **fixed**, while the camera is moved around, capturing images from different angles, distances, and orientations.
- **Moving the Checkerboard:** The camera remains **stationary**, while the checkerboard is moved to different orientations. This provides similar mathematical constraints for calibration.

Both approaches provide similar calibration results, as they introduce variations in the relative positioning of the camera and the reference pattern. In our setup, we specifically adopted the second approach, where the **camera remained stationary** while the **checkerboard was moved** to different orientations. This allowed us to maintain a fixed camera position, reducing additional sources of distortion and ensuring consistent viewpoint calibration across images.

Figure 2: Examples of checkerboard images captured from different viewpoints.

2.4.3 Step 3: Finding 2D Coordinates of the Checkerboard

Having multiple images of the checkerboard and knowing the 3D world coordinates of its corners, the final step before calibration is to find the 2D pixel locations of these corners in each image.

Detecting Checkerboard Corners:
OpenCV provides a built-in function called `findChessboardCorners`, which detects a checkerboard pattern in an image and returns the coordinates of its corners.

```
cv2.findChessboardCorners(image, patternSize, flags)
```

The function returns:

- **retval:** A boolean value (true or false) indicating whether the checkerboard pattern was successfully detected.
- **corners:** An array of 2D points representing the detected checkerboard corners in the image.

Refining Checkerboard Corners
Achieving precise calibration requires accurately detecting the **location of checkerboard corners** at a **sub-pixel level**. To improve corner localization, OpenCV provides the `cornerSubPix` function, which refines the detected corner positions by searching for the most accurate location within a small neighborhood.

```
cv2.cornerSubPix(image, corners, winSize, zeroZone, criteria)
```

These refined detected corners are then used as input for the camera calibration step.

Figure 3: Examples of detected corners in the previously captured checkerboard images.

2.4.4 Step 4: Camera Calibration

The final step of calibration is to pass the 3D world coordinates of the pattern points and their 2D image locations into OpenCV's `calibrateCamera` function. This function estimates the intrinsic and extrinsic parameters of the camera, correcting for distortions. The implementation follows the method proposed in [3], which relies on multiple views of a known calibration pattern.

```
cv2.calibrateCamera( objectPoints, imagePoints, imageSize, None, None )
```

The `calibrateCamera` function returns:

- **retval:** The re-projection error, which measures the accuracy of the calibration.
- **cameraMatrix:** The intrinsic matrix that describes the internal properties of the camera.
- **distCoeffs:** The distortion coefficients for correcting radial and tangential distortions.
- **rvecs and tvecs:** The rotation and translation vectors, which define the pose of the camera relative to the checkerboard in each captured image.

By following this methodology, we ensured high-precision camera calibration, effectively reducing measurement errors and improving the accuracy of liquid detection.

3 Conclusions

In this short report, we explored a computer vision-based approach for estimating the liquid height in a glass. The study investigated two main methodologies: a direct measurement approach based on pixels per metric ratio and a more advanced camera calibration-based method to correct for distortions and improve measurement accuracy. To achieve this, we first curated a structured dataset of images, capturing different glasses filled with various liquids while ensuring controlled conditions such as viewpoint consistency and minimal distortions. The initial approach relied on a simple method that utilized the known glass height dimensions to estimate the liquid height based on the pixel-to-metric ratio. Since this method was prone to errors caused by noise, natural camera perspective and distortions, we applied pre-processing techniques, such as noise reduction and camera calibration as to achieve more precise measurements.

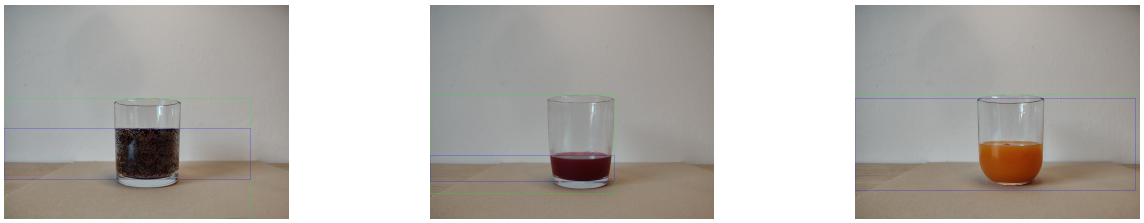


Figure 4: Initial detection with bounding boxes identifying the glass and liquid

Figure 5: Final improved detection with bounding boxes identifying the glass and liquid

The above comparison highlights the improvements obtained through pre-processing techniques. Initially, the detection suffered from incorrect contour identification due to reflections, background noise and shadows interfering. After implementing adaptive thresholding, morphological operations, and perspective correction, the system achieved a reliable and robust liquid height estimations. A final precision analysis confirmed that the liquid height estimation reached an accuracy within a **$\pm$3mm range**, with the only exception of B-C, which has a slightly larger error. The minor detection discrepancies are due to problems such as uneven lighting, shadows, reflections, and glass transparency.

References

[1] A. Rosebrock, “Measuring size of objects in an image with opencv,” 2016, https://pyimagesearch.com/2016/03/28/measuring-size-of-objects-in-an-image-with-opencv/.

[2] OpenCV Documentation, “Camera Calibration with OpenCV,” 2024, https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html.

[3] Z. Zhang, “A flexible new technique for camera calibration,” Microsoft Research, Redmond, WA, USA, Tech. Rep. MSR-TR-98-71, 1998, https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr98-71.pdf.