**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN – ĐIỆN TỬ

----------o0o----------

**TIỂU LUẬN**
**MÔN: THỊ GIÁC MÁY**

# KIỂM TRA DÁN NHÃN VÀ MỰC NƯỚC CỦA SẢN PHẨM BẰNG THỊ GIÁC MÁY

**GVHD: TS. Nguyễn Đức Thành**



| STT | HỌ VÀ TÊN | MSSV |
|---|---|---|
| 1 | Nguyễn Hoàng Ân | 1610137 |
| 2 | Hồ Hưng Duy | 1651017 |
| 3 | Lê Giang Nam | 1612102 |
| 4 | Đinh Phước Lộc | 1710178 |



**TP. HỒ CHÍ MINH, THÁNG 12 NĂM 2019**

# MỤC LỤC

# DANH SÁCH HÌNH MINH HỌA

# 1. GIỚI THIỆU

## 1.1 Tổng quan đề tài

Dây chuyền đóng chai, đóng gói sản phẩm tự động là một trong những là dây chuyền tiên tiến, hiện đại giúp các doanh nghiệp, xưởng sản xuất tiết kiệm được thời gian, chi phí và tăng năng suất công việc. Toàn bộ quá trình chiết rót đều được thực hiện theo một quy trình khép kín nghiêm ngặt nhằm đảm bảo chất lượng sản phẩm.

*Hình 1. Hình minh họa dây chuyền chiết rót và đóng chai tự động*

Một quy trình đóng chai, đóng gói tự động, sản phẩm, là tổng hợp của các tác vụ nối tiếp hoặc song song, theo một vòng lặp nhất định. Các tác vụ đó thể là: vệ sinh vật chứa, chiết rót, đóng chai, dán nhãn, kiểm tra chất lượng… Mỗi khâu có một tầm quan trọng khác nhau, nhưng đều ảnh hưởng trực tiếp đến kết quả chất lượng của sản phẩm.

Một trong những tác vụ được đánh giá là mang ý nghĩa quyết định đó là khâu kiểm tra diện ngoài của sản phẩm. Lấy ví dụ với dây chuyền sản xuất nước đóng chai. Các yêu cầu của các loại sản phẩm liên quan đến chất lỏng, luôn chú trọng đến mực chất lỏng bên trong chai, cũng như các bao bìa, dán nhãn bên ngoài.

Với mực chất lỏng trong chai, theo phương thức truyền thống, các công nhân trong nhà máy sẽ là người chịu trách nhiệm với khâu này. Phương thức được thực hiện khá đơn giản, là cho băng chuyền chạy qua một màn đèn neon, các công nhân có nhiệm vụ dùng mắt thường, để kiểm tra nước trong chai đã đạt tiêu chuẩn chưa. Việc kiểm tra bao bì sản phẩm cũng vậy, người công nhận sẽ nhận dạng được các sản phẩm đã đạt hoặc bị lỗi việc dán nhãn.

*Hình 2. Hình ảnh minh họa quy trình kiểm tra mực nước bằng mắt*

Với mục đích tối ưu nguồn nhân công, tự động hóa quy trình sản xuất, các nhà phát triển đã áp dụng những phương thức mới, áp dụng kỹ thuật mới vào các khâu sản xuất, đặc biệt là khâu phát hiện lỗi sai về mực chất lỏng trong chai và lỗi dán nhãn bao bì. Vì đặc trưng của các khâu trên cần sự quan sát, nhận dạng, và đưa ra kết luận, Xử Lý Ảnh trong công nghiệp là lĩnh vực được nghiên cứu và phát triển hàng đầu. Nhóm nghiên cứu chúng em cũng dựa vào những đặc điểm, mục tiêu của công việc, để thực hiện nghiên cứu đề tài này.


*Hình 3. Hình minh họa dây chuyền tự động hóa công đoạn kiểm tra mực nước*

## 1.2 Nhiệm vụ đề tài

- **Nội dung 1:** Thực hiện nghiên cứu và nắm vững một số giải thuật xử lý ảnh
- **Nội dung 2:** Thực hiện lấy mẫu ảnh để thực hiện chạy thử các mô hình
- **Nội dung 3:** Nhận dạng và phát hiện lỗi mực nước và lỗi dán nhãn

## 1.3 Phân chia công việc trong nhóm

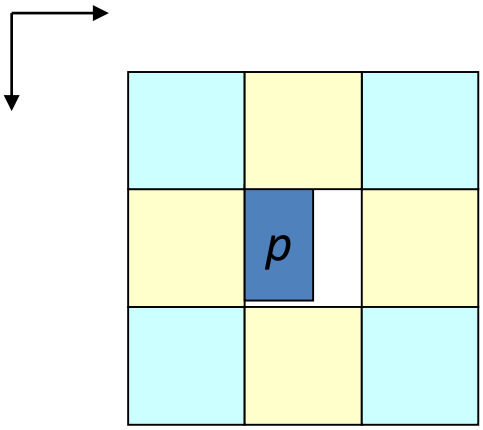
| Nhiệm vụ | Thành viên phụ trách | Ghi chú |
|---|---|---|

| | | |
|---|---|---|
| Nghiên cứu sâu các giải thuật xử lý ảnh | Hưng Duy + Phước Lộc | |
| Nghiên cứu các phương pháp Machine Learning | Giang Nam + Hàng Ân | |
| Thực hiện lấy mẫu ảnh và video | Phước Lộc | |
| Lập trình và thử nghiệm trên mẫu ảnh và video | Hưng Duy + Giang Nam | |
| Tổng hợp báo cáo | Hoàng Ân | |

## 2. LÝ THUYẾT VỀ XỬ LÝ ẢNH

- Conneted component (8)
- Phân ngưỡng otsu & Bộ lọc Gause
- Contour

### 2.1 Lân cận 8 của 1 pixel




- **Xét 1 pixel p tại (x,y):**

- Lân cận-4 của p, $N_4(p)$: các điểm có tọa độ tại (x+1,y), (x-1,y), (x,y+1), (x,y-1).
- Lân cận chéo-4 của p, $N_D(p)$: (x+1,y+1), (x+1,y-1), (x-1,y+1), (x-1,y-1).
- Lân cận-8 của p, $N_8(p)$: $N_4(p)$ và $N_D(p)$.

- **Kết nối giữa 2 pixel:**

2 pixel p và q được gọi là kết nối nếu chúng là lân cận tương ứng và mức xám của chúng thỏa tiêu chuẩn tương đồng V.

- Kết nối-4: $q \in N_4(p)$.
- Kết nối-8: $q \in N_8(p)$.
- Kết nối hỗn hợp-m: thỏa 1 trong 2 điều kiện sau
  - $q \in N_4(p)$.
  - $q \in N_D(p)$ và $N_4(p) \cap N_4(q) = \emptyset$.

- **Đo khoảng cách:**

Cho các pixels p, q và z có tọa độ (x,y), (s,t), (u,v) tương ứng, hàm khoảng cách D có các đặc tính sau:

a. $D(p, q) \geq 0$ [$D(p, q) = 0$, if $p = q$]

b. $D(p, q) = D(q, p)$

c. $D(p, z) \leq D(p, q) + D(q, z)$

**Euclidean distance**:

$$D_e(p, q) = [(x - s)^2 + (y - t)^2]^{1/2}$$

**City block distance**:

$$D_4(p, q) = |x - s| + |y - t|$$

**Chess board distance**:

$$D_8(p, q) = \max(|x - s|, |y - t|)$$

## 2.2 Phân ngưỡng Otsu

- **Khái niệm phân ngưỡng:**

Phân ngưỡng ảnh là tách hình ảnh của một một vùng ảnh ra khỏi ảnh nền.

- **Lược đồ xám (Histogram):**

Là một đồ thị dạng thanh biểu diễn tần suất xuất hiện các mức xám của ảnh. Trong đó trục hoành biểu diễn giá trị mức xám của ảnh có giá trị từ 0 đến 255, trục tung biểu diễn tần suất xuất hiện mức xám của ảnh.

Công thức tổng quát:

$$p(r_k) = \frac{n_k}{MN} \qquad (1.1)$$

Trong đó:

$p(r_k)$ là tần suất xuất hiện mức xám

$n_k$ là giá trị điểm ảnh tại vị trí k

M,N : kích cỡ ma trận ảnh.

- **Thuật toán Otsu:**

- Lặp với tất cả giá trị ngưỡng.
- Ứng với mỗi giá trị ngưỡng, đo phương sai kết hợp cùng lớp ($\sigma_W$) dựa trên phương sai của vùng nền ($\sigma_b$) và vùng cảnh ($\sigma_f$) với trọng số xác suất ($W_b$, $W_f$) tương ứng.
- Giá trị ngưỡng tương ứng với phương sai cùng lớp nhỏ nhất.

$$\text{Within Class Variance} \quad \sigma_W^2 = W_b\,\sigma_b^2 + W_f\,\sigma_f^2$$

- Có thể cải tiến nhanh hơn dùng đo phương sai kết hợp giữa lớp ($\sigma_B$) lớn nhất.

$$\begin{aligned}\text{Between Class Variance} \quad \sigma_B^2 &= \sigma^2 - \sigma_W^2 \\ &= W_b(\mu_b - \mu)^2 + W_f(\mu_f - \mu)^2 \\ &= W_b\,W_f\,(\mu_b - \mu_f)^2\end{aligned}$$

- Trình tự thực hiện:

1. Sử dụng lược đồ Histogram biểu diễn tần suất xuất hiện mức xám
2. Chọn một ngưỡng $T_k = k$, $(0<k<L-1)$ để phân ảnh đầu vào thành 2 lớp $C_1$ (tập hợp các điểm ảnh có giá trị $\leq k$) và $C_2$ (tập hợp các điểm ảnh có giá

trị lớn hơn k). Tỉ lệ lớp $C_1$ với số lượng điểm ảnh k với tổng số lượng điểm ảnh được ký hiệu $P_1(k)$, tương tự $C_2$ ký hiệu là $P_2(k)$.

$$P_1(k) = \sum_{i=0}^{k} p_i \quad (1.2)$$

$$P_2(k) = \sum_{i=k+1}^{L-1} p_i = 1 - P_1(k) \quad (1.3)$$

Sau đó tính giá trị trung bình $m_1$ của lớp $C_1$

$$m_1(k) = \sum_{i=0}^{k} iP\left(\frac{i}{C_1}\right) = \frac{1}{P_1(k)} \sum_{i=0}^{k} iP_i \quad (1.4)$$

Tương tự, ta tính $m_2$ của $C_2$

$$m_2(k) = \sum_{i=k+1}^{L-1} iP\left(\frac{i}{C_2}\right) = \frac{1}{P_2(k)} \sum_{i=k+1}^{L-1} iP_i \quad (1.5)$$

3. Theo Otsu, ta sẽ tính ngưỡng k* mà giá trị tại đó sự chênh lệch giữa hai đoạn (màu nền và màu ký tự) đạt giá trị cực đại, ký hiệu $\sigma_B^2(k^*)$, được tính:

$$\sigma_B^2(k^*) = \underset{0 \le k \le L-1}{Max} \sigma_B^2(k) \quad (1.6)$$

Trong đó $\sigma_B$ là phương sai hai lớp $C_1$ và C2. Ta có:

$$\sigma^2_B = P_1(m_1 - m_G)^2 + P_2(m_2 - m_G)^2$$
$$= P_1 P_2 (m_1 - m_2)^2$$
$$= \frac{(m_G P - m)^2}{P_1(1 - P_1)}$$

Từ công thức trên ta suy ra:

$$\sigma^2_B(k) = \frac{\left[m_G P_1(k) - m(k)\right]^2}{P_1(k)\left[1 - P_1(k)\right]} \quad (1.7)$$

Trong đó:

- $m_G$ là giá trị trung bình của ảnh.

$$m_G = \sum_{i=0}^{L-1} ip_i \text{ hoặc } m_G = P_1 m_1 + P_2 m_2 .$$

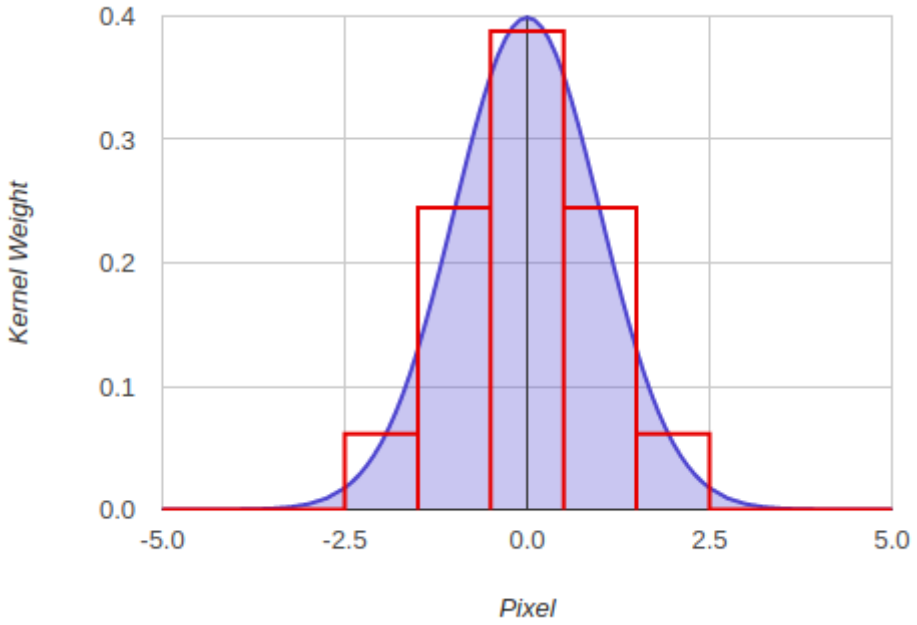
- $m_k$ là giá trị trung bình đến ngưỡng k. Với $m_k = \sum_{i=0}^{k} ip_i$ .

Nếu có nhiều giá trị $\sigma^2_B$ lớn nhất bằng nhau, ta sẽ chọn k có giá trị lớn nhất làm ngưỡng k*, sau đó ta thực hiện nhị phân biển số theo theo ngưỡng.

## 2.3 Bộ lọc Gaussian:

- Bộ lọc Gauss được cho là bộ lọc hữu ích nhất, được thực hiện bằng cách nhân chập ảnh đầu vào với một ma trận lọc Gauss sau đó cộng chúng lại để tạo thành ảnh đầu ra.

- Ý tưởng chung là giá trị mỗi điểm ảnh sẽ phụ thuộc nhiều vào các điểm ảnh ở gần hơn là các điểm ảnh ở xa. Trọng số của sự phụ thuộc được lấy theo hàm Gauss (cũng được sử dụng trong quy luật phân phối chuẩn).
- Dưới đây là biểu diễn ma trận lọc Gauss:




Hình 4. Ma trận lọc Gauss

Giả sử ảnh là một chiều. Điểm ảnh ở trung tâm sẽ có trọng số lớn nhất. Các điểm ảnh ở càng xa trung tâm sẽ có trọng số giảm dần khi khoảng cách từ chúng tới điểm trung tâm tăng lên. Như vậy điểm càng gần trung tâm sẽ càng đóng góp nhiều hơn vào giá trị điểm trung tâm.
**Chú ý:** Trên thực tế, việc lọc ảnh dựa trên hàm Gauss 2 chiều (ngang và dọc). Phân phối chuẩn 2 chiều có thể biểu diễn dưới dạng:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Hoặc:

$$G\_0(x,y) = Ae^{\frac{-(x-\mu\_x)^2}{2\sigma^2\_x} + \frac{-(y-\mu\_y)^2}{2\sigma_y^2}}$$

Trong đó $\mu$ là trung bình (đỉnh), $\sigma^2$ là phương sai của các biến số $x$ và $y$.

**Tham số $\mu$ quyết định tác dụng của bộ lọc Gauss lên ảnh. Độ lớn của ma trận lọc (kernel) cần được lựa chọn cho đủ rộng.**

## 2.4 Contour

- Contour là "tập các điểm-liên-tục tạo thành một đường cong (curve) (boundary), và không có khoảng hở trong đường cong đó, đặc điểm chung trong một contour là các các điểm có cùng /gần xấu xỉ một giá trị màu, hoặc cùng mật độ. Contour là một công

cụ hữu ích được dùng để phân tích hình dạng đối tượng, phát hiện đối tượng và nhận dạng đối tượng".

- Để tìm contour chính xác, chúng ta cần phải nhị phân hóa bức ảnh ( không phải ảnh grayscale). Các kỹ thuật nhị phân hóa ảnh ở xử lý ảnh cơ bản có thể liệt kê đến là đặt ngưỡng, hoặc candy edge detection.
- Trong opencv, việc tìm một contour là việc tìm một đối tượng có màu trắng trên nền đen.

## 2.5 Opening

- Opening là một trong những phép toán điển hình của các phép toán hình thái học. Phép Opening được thực hiện bằng phép co (Erosion) trước sau đó mới thực hiện phép giãn nở (Dilation).
- Công thức:

$$A \circ B = (A \ominus B) \oplus B$$

- Ứng dụng: Phép toán mở (Opening) được ứng dụng trong việc loại bỏ các phần lồi lõm và làm cho đường bao các đối tượng trong ảnh trở nên mượt mà hơn.
- Ví dụ: sử dụng phép Opening trong python

```
opening = cv.morphologyEx(img, cv.MORPH_OPEN, kernel)
```

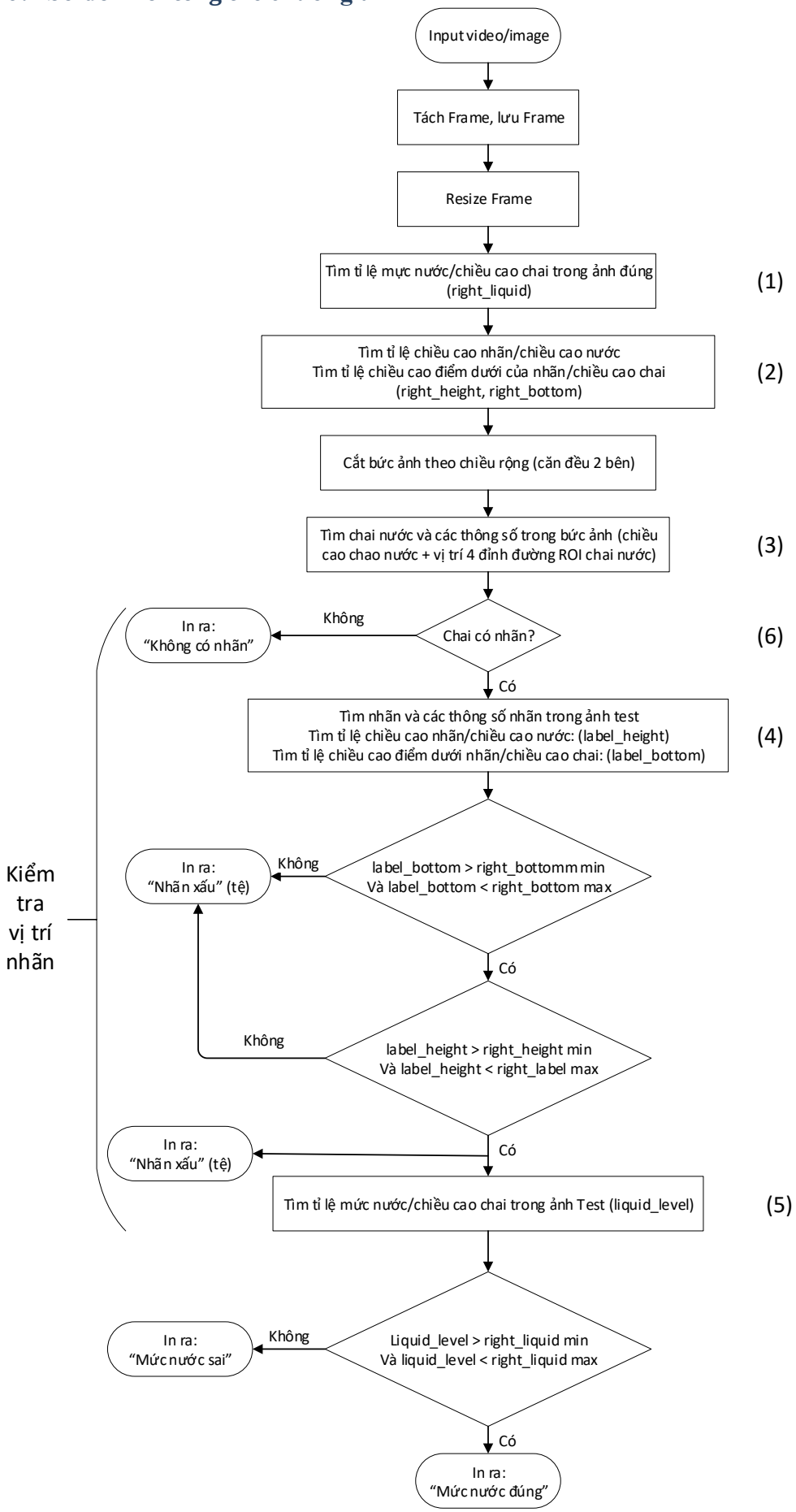
Và kết quả như hình bên dưới:

*Hình 5. Kết quả trước và sau khi sử dụng phép toán Opening*

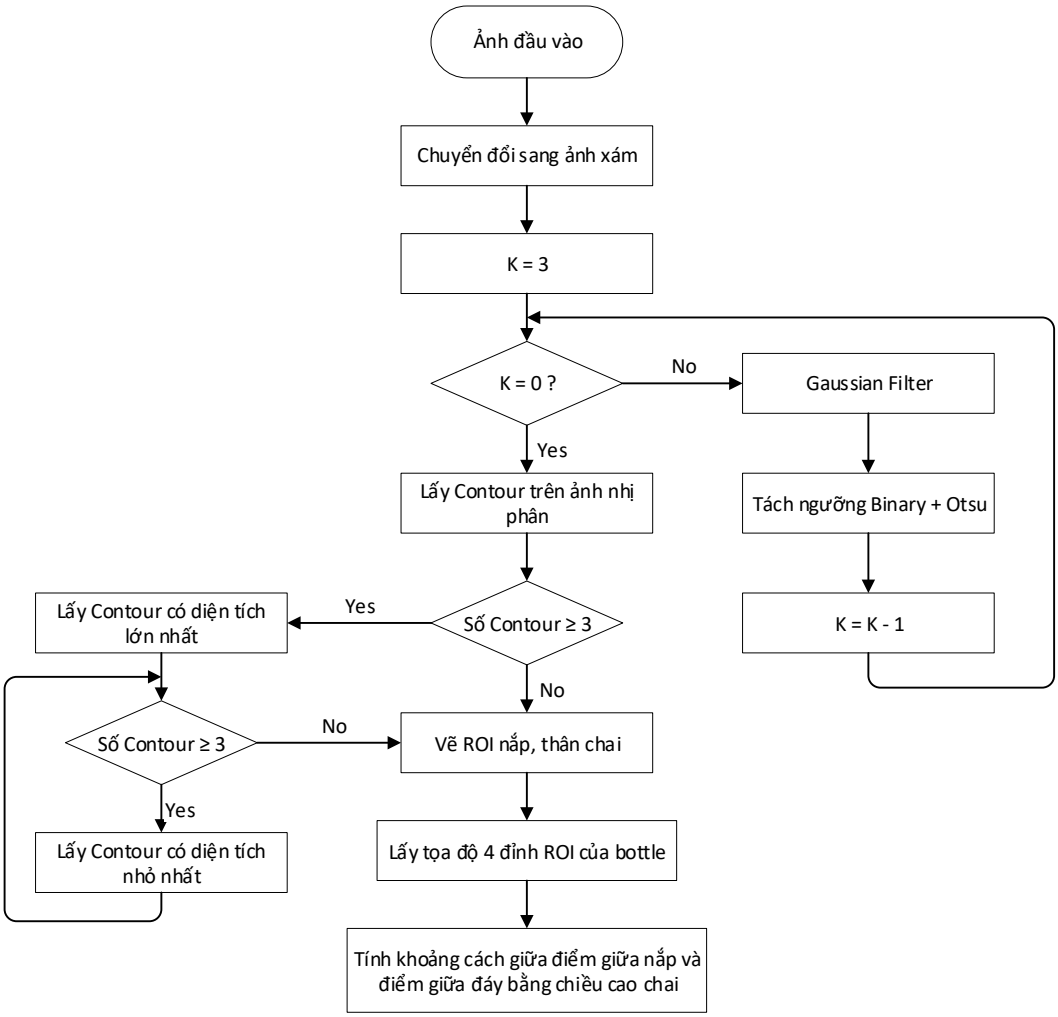
# 3. QUÁ TRÌNH THỰC HIỆN

## 3.1 Sơ đồ khối

## a. Sơ đồ khối tổng thể chương trình

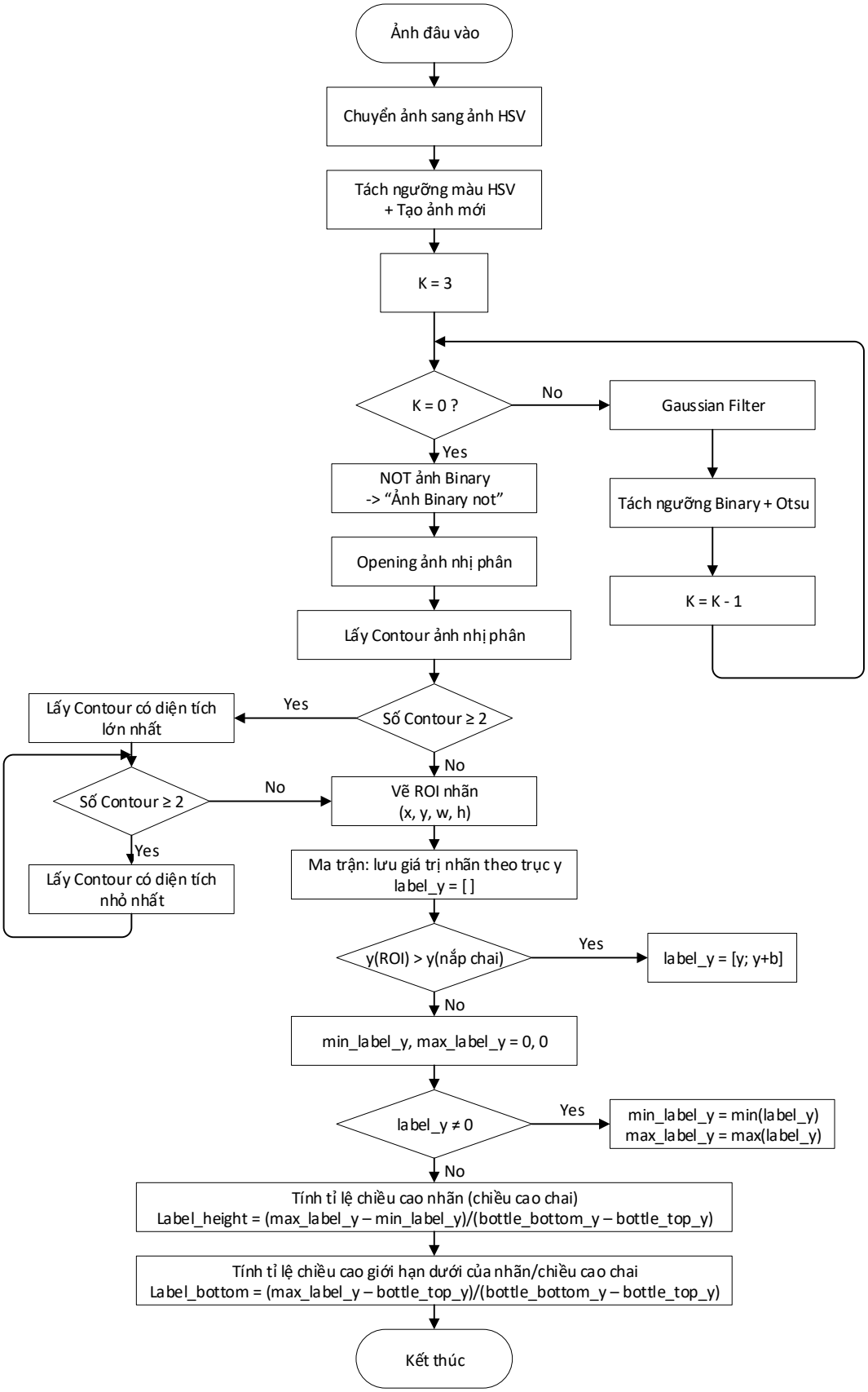


**b. Sơ đồ tìm tỉ lệ:** $\frac{\textbf{Mức nước}}{\textbf{Chiều cao chai trong ảnh đúng}}$ **(1)**

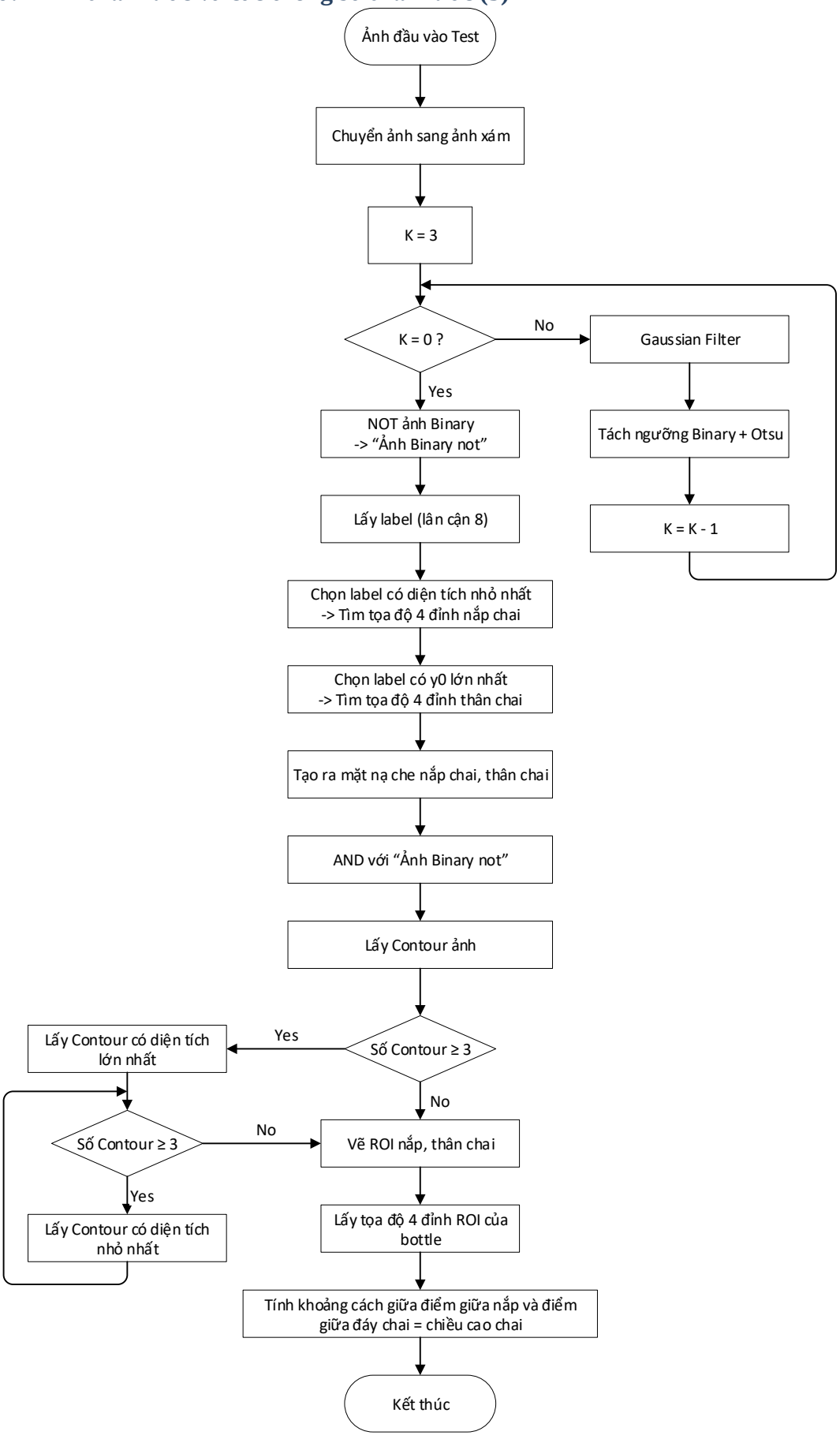


## c. Sơ đồ tìm các tỉ lệ (2) và (4):

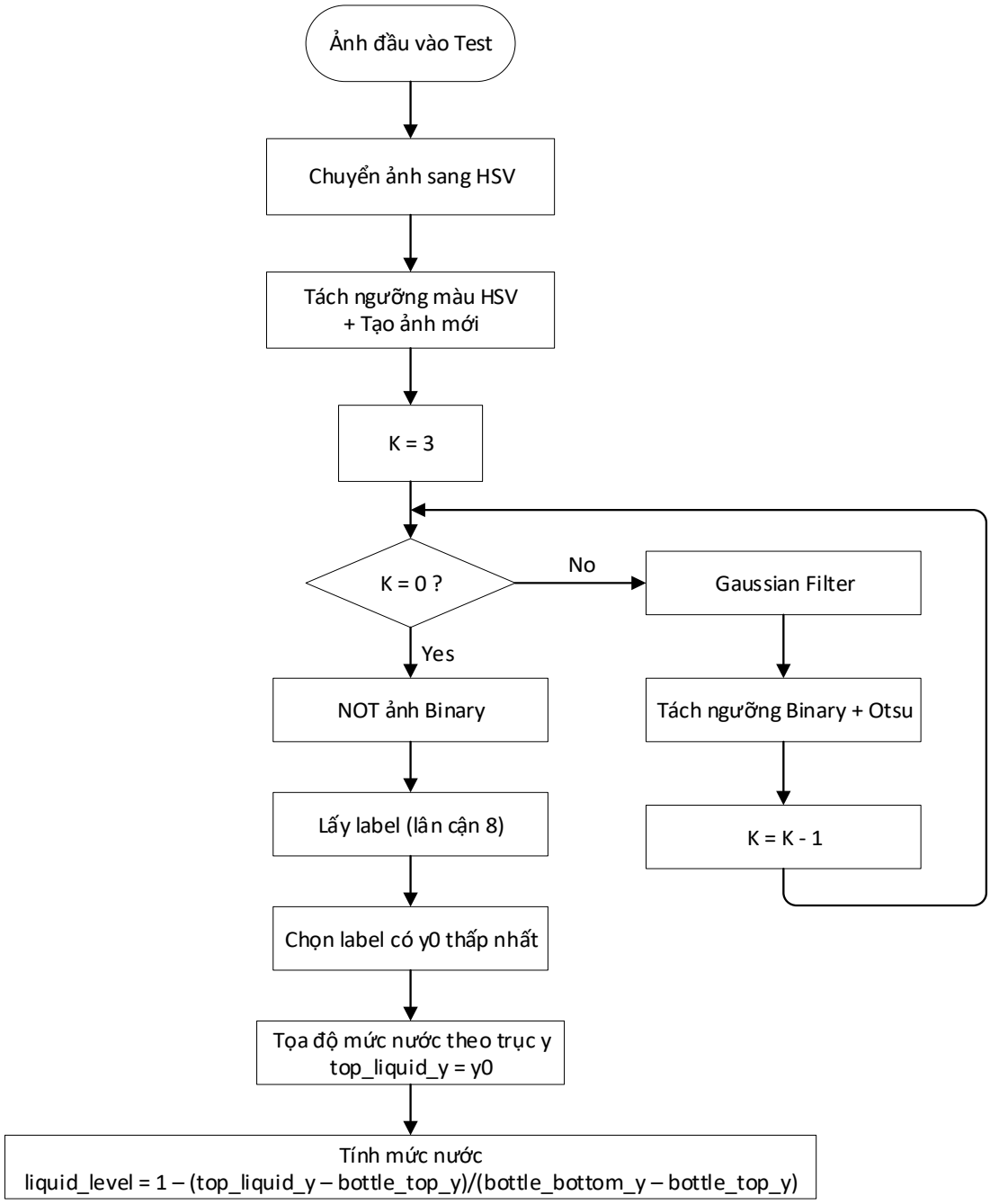
$$\frac{Chiều\ cao\ nhãn}{Chiều\ cao\ nước}\ ;\quad \frac{Chiều\ cao\ điểm\ dưới\ của\ nhãn}{Chiều\ cao\ chai}$$



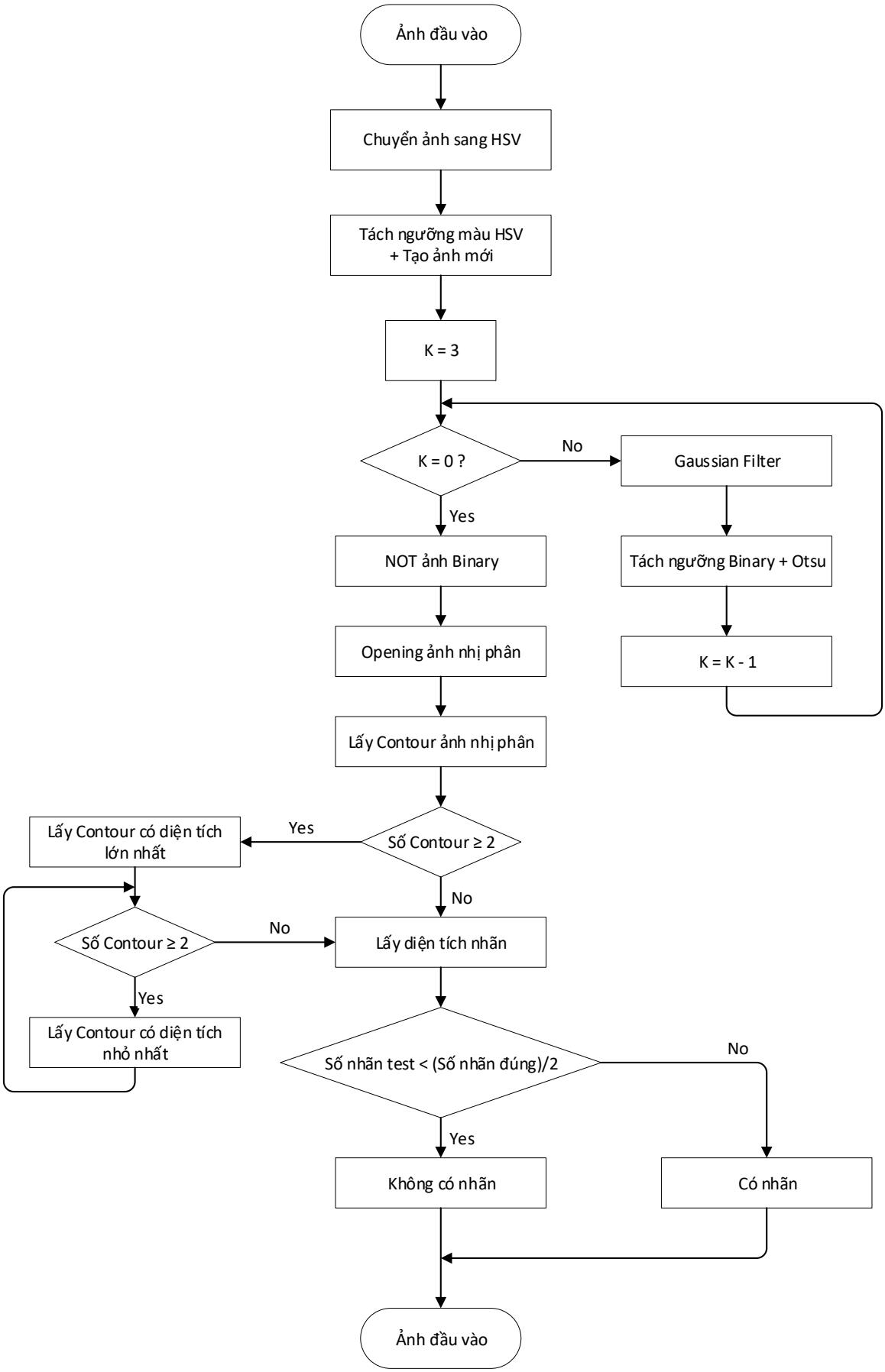
## d. Tìm chai nước và các thông số chai nước (3)



**e. Sơ đồ khối tỉ lệ $\frac{Mức\ nước}{Chiều\ cao\ trong\ ảnh\ Test}$ (5)**



## f. Sơ đồ kiểm tra nhãn có hay không (6)



## 3.2 Các mẫu ảnh và các trường hợp test

Quy trình thực hiện của nhóm áp dụng theo sơ đồ được thể hiện ở bên trên. Nhóm chọn chai nước giải khác Coca-Cola để thực hiện đề tài. Sau đây là mẫu đúng của sản phẩm:


*Hình 6. Mẫu chai đạt yêu cầu về nhãn và mực nước*

Các trường hợp lỗi, không đạt yêu cầu:

Nhãn bị dán sai vị trí. Đặc điểm của bức hình là:

- Có thể thấy nhãn đã bị dán sai vị trí so theo chiều dọc.
- Không thể nhận biết được là mực nước có đúng hay không.


Nhãn bị dán lệch vị trí và mực nước thấp hơn mực nước chuẩn.

Trường hợp không có nhãn và mực nước thấp hơn mực nước chuẩn.

Trường hợp dán nhãn đúng, nhưng mực nước thấp hơn mực nước chuẩn.

Trường hợp dán nhãn đúng, nhưng mực nước thấp hơn mực nước chuẩn.

## 3.3 Thực hiện lập trình

Nhóm chọn ngôn ngữ lập trình là Python, với thư viện OpenCV.

```
#-----------------------------------------------------------------------------------------------------
# This software is using for checking the liquid volume and the label position of bottle. This background
# of this project is white and the object is Coca Cola bottle.
#-----------------------------------------------------------------------------------------------------
import cv2
from PIL import Image
import matplotlib.pyplot as plt
```

```
import numpy as np
import os, sys
import configparser
import argparse
from pylab import *
import imutils
from imutils import perspective
from scipy.spatial import distance as dist
from imutils import contours
from datetime import datetime
#-------------------------------------------------------------------------------------------------
# This is configuration for ROI
config = configparser.ConfigParser()
config.read("system.ini")
IMAGES_MASK = config.get("path", "images_mask")
SOURCE_IMAGES_PATH = config.get("path", "source_dir")
RESULT_IMAGES_PATH = config.get("path", "result_dir")
CUT_BORDER = config.getfloat("geometry", "cut_border")
PRE_ROI_WIDTH = config.getfloat("geometry", "pre_roi_width")
ROI_WIDTH = config.getfloat("geometry", "roi_width")
# -------------------------------------------------------------------------------------------------
# Threshold to detect label ( relative)
LABEL_H_MIN = 0
LABEL_H_MAX = 250
LABEL_S_MIN = 142
LABEL_S_MAX = 255
LABEL_V_MIN = 106
LABEL_V_MAX = 255
#-------------------------------------------------------------------------------------------------
# threshold for liquid level
LIQUID_LEVEL_H_MIN = 0
LIQUID_LEVEL_H_MAX = 255
LIQUID_LEVEL_S_MIN = 0
LIQUID_LEVEL_S_MAX = 111
LIQUID_LEVEL_V_MIN = 0
LIQUID_LEVEL_V_MAX = 111
```

```
#-----------------------------------------------------------------------------------------------------
BANG_HEIGHT = 0.2
#-----------------------------------------------------------------------------------------------------
def input_image(path):
        image = cv2.imread(path,1)
        image = cv2.resize(image, None, fx=0.3, fy=0.3, interpolation=cv2.INTER_CUBIC)
        image_e = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        return image
#-----------------------------------------------------------------------------------------------------
def midpoint(ptA, ptB):
        return ((ptA[0] + ptB[0]) * 0.5, (ptA[1] + ptB[1]) * 0.5)
#-----------------------------------------------------------------------------------------------------
def his_enhance(image):
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        image = cv2.equalizeHist(image)
        image = cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)
        return image
#-----------------------------------------------------------------------------------------------------
def get_roi(image, left_border, right_border):
    image_h, image_w, _ = image.shape
    left_part = image[:, 0:left_border]
    right_part = image[:, right_border:image_w]
    return np.column_stack((left_part, right_part))
#-----------------------------------------------------------------------------------------------------
def find_background(image, roi):
    image_h, image_w, _ = image.shape
    hsv_img = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
    roi_hist = cv2.calcHist([hsv_roi], [0, 1], None, [180, 256], [0, 180, 0, 256])
    mask = cv2.calcBackProject([hsv_img], [0, 1], roi_hist, [0, 180, 0, 256], 1)
    ksize = int(0.005 * image_h)
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
    mask = cv2.filter2D(mask, -1, kernel)
    _, mask = cv2.threshold(mask, 30, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
    return mask
```

```
#-----------------------------------------------------------------------------------------------------
def get_pre_borders(mask):
        components = cv2.connectedComponentsWithStats(mask, connectivity=8, ltype=cv2.CV_32S)
        _, labelmap, stats, centers = components
        st = stats[:, 3]
        largest = np.argmax(st)
        st[largest] = 0
        second = np.argmax(st)
        left = stats[second, 0]
        width = stats[second, 2]
        right = left + width
        roi_width = int(width * ROI_WIDTH)
        return left, right, roi_width
# -----------------------------------------------------------------------------------------------------
def ConnectedComponents(bin1):
        components = cv2.connectedComponentsWithStats(bin1, connectivity=8, ltype=cv2.CV_32S)
        _, labelmap, stats, centers = components
        st = stats[:, 4]
        st_y = stats[:, 1]
        S_largests = [np.argmax(st)]
        S1000 = np.where(st > 1000)
        st_arg = S1000[0]
        number_label = len(st_arg)
        mask_bin = np.zeros(bin1.shape, dtype=np.uint8)
        if (len(st_arg) > 2):
                # Bang /lid area
                S_smallest_arg = np.argmin(st)
                left_bang = stats[S_smallest_arg,0]
                top_bang = stats[S_smallest_arg,1]
                right_bang = left_bang + stats[S_smallest_arg,2]
                bottom_bang = top_bang + stats[S_smallest_arg,3]
                # liquid level in axis y => max
                y_max_arg = np.argmax(st_y)
                left_liq = stats[y_max_arg,0]
```

```
                top_liq = stats[y_max_arg,1]
                right_liq = left_liq + stats[y_max_arg,2]
                bottom_liq = top_liq + stats[y_max_arg,3]
                mask_bin[top_bang:bottom_bang,left_bang:right_bang] = 255
                mask_bin[top_liq:bottom_liq,left_liq:right_liq] = 255
        return number_label, mask_bin
#-----------------------------------------------------------------------------------------------
def get_bottle_mask(bin):
        def clean(cln_bin, larg_num):
                components = cv2.connectedComponentsWithStats(cln_bin, connectivity=8,
ltype=cv2.CV_32S)
                _, labelmap, stats, centers = components
                st = stats[:, 4]
                largests = [np.argmax(st)]
                st[largests[0]] = 0
                largests.append(np.argmax(st))
                cln_bin = np.zeros(cln_bin.shape, dtype=np.uint8)
                cln_bin[labelmap == largests[larg_num]] = 255
                return cln_bin, stats[largests[0]]
        bin, _ = clean(bin, 1)
        bin = cv2.bitwise_not(bin)
        bin, stats = clean(bin, 0)
        left = stats[0]
        top = stats[1]
        right = left + stats[2]
        bottom = top + stats[3]
        mask = bin[top:bottom, left:right]
        mask = cv2.merge((mask, mask, mask))
        return mask, left, top, right, bottom
#-----------------------------------------------------------------------------------------------
# The liquid level segmantation
def liqlevel_segmentation(image):
        image_h, image_w, _ = image.shape
        center_img = image[:, int(image_w * 0.03): int(image_w * 0.97)]
        blur = int(0.015 * image_w)
        center_h, center_w, _ = center_img.shape
```

```
        img_hsv = cv2.cvtColor(center_img, cv2.COLOR_RGB2HSV)
        filtered = cv2.inRange(img_hsv, (LIQUID_LEVEL_H_MIN, LIQUID_LEVEL_S_MIN, LIQUID_LEVEL_V_MIN),(LIQUID_LEVEL_H_MAX, LIQUID_LEVEL_S_MAX, LIQUID_LEVEL_V_MAX))
        binary = cv2.GaussianBlur(filtered, (5,5), 15)
        ret, binary = cv2.threshold(binary,50,255,cv2.THRESH_BINARY | cv2.THRESH_OTSU)
        binary = cv2.GaussianBlur(filtered, (15,15), 15)
        ret, binary = cv2.threshold(binary,50,255,cv2.THRESH_BINARY | cv2.THRESH_OTSU)
        binary = cv2.GaussianBlur(filtered, (15,15), 15)
        ret, binary = cv2.threshold(binary,50,255,cv2.THRESH_BINARY | cv2.THRESH_OTSU)
        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
        binary = cv2.morphologyEx(binary, cv2.MORPH_CLOSE, kernel)
        binary =cv2.bitwise_not (binary)
        return binary
# ------------------------------------------------------------------------------------------
# Label segmentation
def label_segmentation(image):
        image_h, image_w, _ = image.shape
        center_img = image[:, int(image_w * 0.03): int(image_w * 0.97)]
        blur = int(0.015 * image_w)
        center_h, center_w, _ = center_img.shape
        img_hsv = cv2.cvtColor(center_img, cv2.COLOR_RGB2HSV)
        filtered = cv2.inRange(img_hsv, (LABEL_H_MIN, LABEL_S_MIN, LABEL_V_MIN),(LABEL_H_MAX, LABEL_S_MAX, LABEL_V_MAX))
        binary = cv2.GaussianBlur(filtered, (5,5), 15)
        ret, binary = cv2.threshold(binary,50,255,cv2.THRESH_BINARY | cv2.THRESH_OTSU)
        binary = cv2.GaussianBlur(filtered, (15,15), 15)
        ret, binary = cv2.threshold(binary,50,255,cv2.THRESH_BINARY | cv2.THRESH_OTSU)
        binary = cv2.GaussianBlur(filtered, (15,15), 15)
        ret, binary = cv2.threshold(binary,50,255,cv2.THRESH_BINARY | cv2.THRESH_OTSU)
        binary =cv2.bitwise_not (binary)
        kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (15, 15))
```

```
        binary = cv2.morphologyEx(binary, cv2.MORPH_OPEN, kernel)
        return binary
# ------------------------------------------------------------------------------------------
# Check the existence of label
def check_label_exist(image, S_good_label):
        binary = label_segmentation(image)
        cv2.imshow('binary-check_label_exist',binary)
        contours, _ = cv2.findContours(binary, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
        print('Length of contours-testbefore: check_lablels_exist()',len(contours))
        try:
                if (len(contours) >= 2):
                        c = max(contours, key = cv2.contourArea)
                        contours.remove(c)
                        while (len(contours) >= 2):
                                c1 = min(contours, key = cv2.contourArea)
                                contours.remove(c1)
        except:
                print('error check label exists')
        print('Length of contours-testafter: check_lablels_exist()',len(contours))
        cnt = contours[0]
        S_label = int(cv2.contourArea(cnt))
        if (S_label < (S_good_label / 2)):
                label_exist = False
        else:
                label_exist = True
        return label_exist
# ------------------------------------------------------------------------------------------
# Find the height of label (y_axis)
def find_labels(image):
        image_h, image_w, _ = image.shape
        center_img = image[:, int(image_w * 0.03): int(image_w * 0.97)]
        binary = label_segmentation(image)
        cv2.imshow('binary-find_labels',binary)
        contours, _ = cv2.findContours(binary, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
```

```
        print('Length of contours-testbefore: find_lablels()',len(contours))
        try:
                if (len(contours) >= 2):
                        c = max(contours, key = cv2.contourArea)
                        contours.remove(c)
                        while (len(contours) >= 2):
                                c1 = min(contours, key = cv2.contourArea)
                                contours.remove(c1)
        except:
                print('error find label')
        print('Length of contours-testafter: find_lablels()',len(contours))
        binary = cv2.drawContours(binary, contours, -1, (100,87,51),3)
        hull = [cv2.convexHull(c) for c in contours]
        binary = cv2.drawContours(binary,hull,-1,(100,87,55),3)
        label_y_arr = []
        for num, cnt in enumerate(contours):
                x, y, w, h = cv2.boundingRect(cnt)
    # Getting ROI
                roi = image[y:y+h, x:x+w]
                # show ROI
                cv2.rectangle(center_img,(x,y),( x + w, y + h ),(255,0,0),2)
                cv2.imshow('marked label areas',center_img)
                if (y > (BANG_HEIGHT * image_h)):
                        label_y_arr.append(y)
                        label_y_arr.append(y + h)
        min_label_y, max_label_y = 0, 0
        if label_y_arr:
                min_label_y = min(label_y_arr)
                max_label_y = max(label_y_arr)
        return min_label_y, max_label_y
#-------------------------------------------------------------------------------------------------------
# Find the exact ratio between label and bottle
def right_labels():
        image = input_image('D:\\Testimg\\Bottle_checking (CC)\\good1.jpg')
        binary = label_segmentation(image)
        image_h, image_w, _ = image.shape
```

```
        center_img = image[:, int(image_w * 0.03): int(image_w * 0.97)]
        contours, _ = cv2.findContours(binary, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
        try:
                if (len(contours) >= 2):
                        c = max(contours, key = cv2.contourArea)
                        contours.remove(c)
                        while (len(contours) >= 2):
                                c1 = min(contours, key = cv2.contourArea)
                                contours.remove(c1)
        except:
                print('error right label')
        cnt = contours[0]
        S_label = cv2.contourArea(cnt)
        binary = cv2.drawContours(binary, contours, -1, (100,87,51),3)
        hull = [cv2.convexHull(c) for c in contours]
        binary = cv2.drawContours(binary,hull,-1,(100,87,55),3)
        label_y_arr = []
        for num, cnt in enumerate(contours):
                x, y, w, h = cv2.boundingRect(cnt)
    # Getting ROI
                roi = image[y:y+h, x:x+w]
                # show ROI
                cv2.rectangle(center_img,(x,y),( x + w, y + h ),(255,0,0),2)
                if (y > (BANG_HEIGHT * image_h)):
                        label_y_arr.append(y)
                        label_y_arr.append(y + h)
        min_label_y, max_label_y = 0, 0
        if label_y_arr:
                min_label_y = min(label_y_arr)
                max_label_y = max(label_y_arr)
        return S_label, min_label_y, max_label_y
# ------------------------------------------------------------------------------------------------
# Find the liquid level ratio
def find_liquid_level(image):
        image_h, image_w, _ = image.shape
```

```
        center_img = image[:, int(image_w * 0.03): int(image_w * 0.97)]
        binary = liqlevel_segmentation(image)
        binary_copy = binary.copy()
        binary_copy = cv2.bitwise_not(binary_copy)
        components = cv2.connectedComponentsWithStats(binary_copy, connectivity=8, ltype=cv2.CV_32S)
        _, labelmap, stats, centers = components
        # S of label
        st_s = stats[:, 4]
        s_largest = np.argmax(st_s)
        stats1 = np.delete(stats,s_largest,0)
        # The width of label
        st_w = stats1[:, 2]
        # y0
        st_y = stats1[:, 1]
        w_largests = [np.argmax(st_w)]
        W = np.where(st_w > (image_w / 2))
        st_w_arg = W[0]
        number_label = len(st_w_arg)
        if (len(st_w_arg) > 1):
                y_min_arg = np.argmin(st_y)
                left_liq = stats1[y_min_arg,0]
                top_liq = stats1[y_min_arg,1]
                right_liq = left_liq + stats1[y_min_arg,2]
                bottom_liq = top_liq + stats1[y_min_arg,3]
        y_min_arg = np.argmin(st_y)
        left_liq = stats1[y_min_arg,0]
        top_liq = stats1[y_min_arg,1]
        right_liq = left_liq + stats1[y_min_arg,2]
        bottom_liq = top_liq + stats1[y_min_arg,3]
        return top_liq
#-----------------------------------------------------------------------------------------------
# Find the exact ratio between liquid and bottle
def right_liquid_level():
        image = input_image('D:\\Testimg\\Bottle_checking (CC)\\good1.jpg')
        image_gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
```

```
        image_gray = cv2.GaussianBlur(image_gray, (5,5), 0)
        ret, image_binary = cv2.threshold(image_gray,50,255,cv2.THRESH_BINARY | cv2.THRESH_OTSU)
        image_binary_Gauss = cv2.GaussianBlur(image_binary, (15,15), 15)
        ret2, image_binary2 = cv2.threshold(image_binary_Gauss, 50, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
        image_binary2_Gauss = cv2.GaussianBlur(image_binary2, (15,15), 15)
        ret3, image_binary3 = cv2.threshold(image_binary2_Gauss, 50, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
        # ----------------------------------------------------------------
        # Find and Draw contours
        contours, hierarchy = cv2.findContours(image_binary3,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
        image_binary3c = cv2.drawContours(image_binary3, contours, -1, (100,87,51),3)
        hull = [cv2.convexHull(c) for c in contours]
        image_binary3c_final = cv2.drawContours(image_binary3,hull,-1,(100,87,55),3)
        #----------------------------------------------------------------
        # Remove the largest contours
        if (len(contours) >= 3):
            c = max(contours, key = cv2.contourArea)
            contours.remove(c)
            while (len(contours) >= 3):
                c1 = min(contours, key = cv2.contourArea)
                contours.remove(c1)
        #-----------------------------------------------
        #Draw ROI with the bottle
        #sort contours
        orig1 =image.copy()
        sorted_ctrs = sorted(contours, key=lambda ctr: cv2.boundingRect(ctr)[0])
        # print('Number of contour',len(contours))
        for i, ctr in enumerate(sorted_ctrs):
            # Get bounding box
            x, y, w, h = cv2.boundingRect(ctr)
        # Getting ROI
            roi = image[y:y+h, x:x+w]
        #-------------------------------------------------------------
        #Test draw contour and calculate distance
```

```
colors = ((0, 0, 255), (240, 0, 159), (0, 165, 255), (255, 255, 0),(255, 0, 255))
refObj = None
# loop over the contours individually
for c in sorted_ctrs:
        # if the contour is not sufficiently large, ignore it
        if cv2.contourArea(c) < 100:
                continue
        # compute the rotated bounding box of the contour
        box = cv2.minAreaRect(c)
        box = cv2.cv.BoxPoints(box) if imutils.is_cv2() else cv2.boxPoints(box)
        box = np.array(box, dtype="int")
        # order the points in the contour such that they appear
        # in top-left, top-right, bottom-right, and bottom-left
        # order, then draw the outline of the rotated bounding
        # box
        box = perspective.order_points(box)
        # compute the center of the bounding box
        cX = np.average(box[:, 0])
        cY = np.average(box[:, 1])
        # if this is the first contour we are examining (i.e.,
        # the left-most contour), we presume this is the
        # reference object
        if refObj is None:
                # unpack the ordered bounding box, then compute the
                # midpoint between the top-left and top-right points,
                # followed by the midpoint between the top-right and
                # bottom-right
                (tl, tr, br, bl) = box
                (tlblX, tlblY) = midpoint(tl, bl)
                (trbrX, trbrY) = midpoint(tr, br)
                (tltrX, tltrY) = midpoint(tl, tr)
                (blbrX, blbrY) = midpoint(bl, br)
                bottle_y2 = int(bl[1])
                bottle_x1 = int(tl[0])
                bottle_x2 = int(tr[1])
                # compute the Euclidean distance between the midpoints,
```

```
                    # then construct the reference object
                    D = dist.euclidean((tlblX, tlblY), (trbrX, trbrY))
                    refObj = (box, (cX, cY), D / 5.78)
                    continue
            # draw the contours on the image
            orig = image.copy()
            cv2.drawContours(orig, [box.astype("int")], -1, (0, 255, 0), 2)
            cv2.drawContours(orig, [refObj[0].astype("int")], -1, (0, 255, 0), 2)
            cv2.circle(orig, (int(blbrX), int(blbrY)), 5, (50,115,255), -1)
            D1_body = dist.euclidean((tltrX, tltrY), (blbrX, blbrY)) / refObj[2]
            # print('the length of bottle body', D1_body)
            # stack the reference coordinates and the object coordinates
            # to include the object center
            refCoords = np.vstack([refObj[0], refObj[1]])
            objCoords = np.vstack([box, (cX, cY)])
            box_copy = box
            (tl2, tr2, br2, bl2) = box_copy
            (tltrX2, tltrY2) = midpoint(tl2, tr2)
            (blbrX2, blbrY2) = midpoint(bl2, br2)
            bottle_y1 = int(tl2[1])
            cv2.circle(orig, (int(tltrX2), int(tltrY2)), 5, (50,115,255) , -1)
            cv2.line(orig, (int(blbrX), int(blbrY)), (int(tltrX2), int(tltrY2)),(50,115,255),
2)
            Bottle_height = dist.euclidean((blbrX, blbrY), (tltrX2, tltrY2)) / refObj[2]
            Bottle_position = np.array([bottle_x1,bottle_x2,bottle_y1, bottle_y2])
            (bottleX, bottleY) = midpoint((blbrX, blbrY), (tltrX2, tltrY2))
            cv2.putText(orig, "{:.1f}cm".format(Bottle_height), (int(bottleX),
int(bottleY - 10)),cv2.FONT_HERSHEY_SIMPLEX, 0.55, (50,115,255), 2)
            for ((xA, yA), (xB, yB), color) in zip(refCoords, objCoords, colors):
                    # draw circles corresponding to the current points and
                    # connect them with a line
                    cv2.circle(orig, (int(xA), int(yA)), 5, color, -1)
                    cv2.circle(orig, (int(xB), int(yB)), 5, color, -1)
                    cv2.line(orig, (int(xA), int(yA)), (int(xB), int(yB)),color, 2)
                    # compute the Euclidean distance between the coordinates,
                    # and then convert the distance in pixels to distance in
```

```
                    # units
                    D = dist.euclidean((xA, yA), (xB, yB)) / refObj[2]
                    (mX, mY) = midpoint((xA, yA), (xB, yB))
                    cv2.putText(orig, "{:.1f}cm".format(D), (int(mX), int(mY - 10)),cv2.FONT_HERSHEY_SIMPLEX, 0.55, color, 2)
                cv2.imshow('Ratio distance ',orig)
        Ratio_liquidvsbottle = D1_body / Bottle_height
        return Ratio_liquidvsbottle, Bottle_position
#-------------------------------------------------------------------------------------------------------
def find_bottle(image):
        Bottle_height = 0
        Bottle_position = 0
        # Gaussian filter and threshold segmentation (Binary and Otsu)
        image_gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
        image_gray = cv2.GaussianBlur(image_gray, (5,5), 0)
        ret, image_binary = cv2.threshold(image_gray,50,255,cv2.THRESH_BINARY | cv2.THRESH_OTSU)
        image_binary_Gauss = cv2.GaussianBlur(image_binary, (15,15), 15)
        ret2, image_binary2 = cv2.threshold(image_binary_Gauss,50,255,cv2.THRESH_BINARY | cv2.THRESH_OTSU)
        image_binary2_Gauss = cv2.GaussianBlur(image_binary2, (15,15), 15)
        ret3, image_binary3 = cv2.threshold(image_binary2_Gauss,50,255,cv2.THRESH_BINARY | cv2.THRESH_OTSU)
        binary = image_binary3.copy()
        kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (15, 15))
        binary = cv2.morphologyEx(binary, cv2.MORPH_OPEN, kernel)
        cv2.imshow('before using connect',binary)
        binary =cv2.bitwise_not (binary)
        number_label, mask_bin = ConnectedComponents(binary)
        binary = cv2.bitwise_and(binary,mask_bin)
        binary = cv2.bitwise_not(binary)
        image_binary4 = binary.copy()
        # Find and Draw contours
        contours, hierarchy = cv2.findContours(binary,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
        # Remove the largest contours
```

```
try:
        if (len(contours) >= 3):
                c = max(contours, key = cv2.contourArea)
                contours.remove(c)
                while (len(contours) >= 3):
                        c1 = min(contours, key = cv2.contourArea)
                        contours.remove(c1)
except:
        print('error find bottle')
image_binary3c = cv2.drawContours(binary, contours, -1, (100,87,51),3)
hull = [cv2.convexHull(c) for c in contours]
image_binary3c_final = cv2.drawContours(binary,hull,-1,(100,87,55),3)
#Draw ROI with the bottle
#sort contours
org1 =image.copy()
sorted_ctrs = sorted(contours, key=lambda ctr: cv2.boundingRect(ctr)[0])
for i, ctr in enumerate(sorted_ctrs):
        # Get bounding box
        x, y, w, h = cv2.boundingRect(ctr)
# Getting ROI
        roi = image[y:y+h, x:x+w]
        cv2.rectangle(org1,(x,y),( x + w, y + h ),(255,0,0),2)
        cv2.imshow('marked areas - test',org1)
#Test draw contour and calculate distance
colors = ((0, 0, 255), (240, 0, 159), (0, 165, 255), (255, 255, 0),(255, 0, 255))
refObj = None
# loop over the contours individually
# print('length sorted_ctrs',len(sorted_ctrs))
for c in sorted_ctrs:
        # if the contour is not sufficiently large, ignore it
        if cv2.contourArea(c) < 100:
                continue
        # compute the rotated bounding box of the contour
        box = cv2.minAreaRect(c)
        box = cv2.cv.BoxPoints(box) if imutils.is_cv2() else cv2.boxPoints(box)
        box = np.array(box, dtype="int")
```

```
# order the points in the contour such that they appear
# in top-left, top-right, bottom-right, and bottom-left
# order, then draw the outline of the rotated bounding
# box
box = perspective.order_points(box)
# compute the center of the bounding box
cX = np.average(box[:, 0])
cY = np.average(box[:, 1])
# if this is the first contour we are examining (i.e.,
# the left-most contour), we presume this is the
# reference object
if refObj is None:
        # unpack the ordered bounding box, then compute the
        # midpoint between the top-left and top-right points,
        # followed by the midpoint between the top-right and
        # bottom-right
        (tl, tr, br, bl) = box
        (tlblX, tlblY) = midpoint(tl, bl)
        (trbrX, trbrY) = midpoint(tr, br)
        (tltrX, tltrY) = midpoint(tl, tr)
        (blbrX, blbrY) = midpoint(bl, br)
        bottle_y2 = int(bl[1])
        bottle_x1 = int(tl[0])
        bottle_x2 = int(tr[1])
        # compute the Euclidean distance between the midpoints,
        # then construct the reference object
        D = dist.euclidean((tlblX, tlblY), (trbrX, trbrY))
        D1_body = dist.euclidean((tltrX, tltrY), (blbrX, blbrY))
        refObj = (box, (cX, cY), D / 5.72)
        continue
# draw the contours on the image
orig = image.copy()
cv2.drawContours(orig, [box.astype("int")], -1, (0, 255, 0), 2)
cv2.drawContours(orig, [refObj[0].astype("int")], -1, (0, 255, 0), 2)
cv2.circle(orig, (int(blbrX), int(blbrY)), 5, (50,115,255), -1)
# stack the reference coordinates and the object coordinates
```

```
                # to include the object center
                refCoords = np.vstack([refObj[0], refObj[1]])
                objCoords = np.vstack([box, (cX, cY)])
                box_copy = box
                (tl2, tr2, br2, bl2) = box_copy
                (tltrX2, tltrY2) = midpoint(tl2, tr2)
                (blbrX2, blbrY2) = midpoint(bl2, br2)
                bottle_y1 = int(tl2[1])
                cv2.circle(orig, (int(tltrX2), int(tltrY2)), 5, (50,115,255) , -1)
                cv2.line(orig, (int(blbrX), int(blbrY)), (int(tltrX2), int(tltrY2)),(50,115,255),
2)
                Bottle_height = dist.euclidean((blbrX, blbrY), (tltrX2, tltrY2)) / refObj[2]
                Bottle_position = np.array([bottle_x1,bottle_x2,bottle_y1, bottle_y2])
                (bottleX, bottleY) = midpoint((blbrX, blbrY), (tltrX2, tltrY2))
                cv2.putText(orig, "{:.1f}cm".format(Bottle_height), (int(bottleX),
int(bottleY - 10)),cv2.FONT_HERSHEY_SIMPLEX, 0.55, (50,115,255), 2)
                for ((xA, yA), (xB, yB), color) in zip(refCoords, objCoords, colors):
                        # draw circles corresponding to the current points and
                        # connect them with a line
                        cv2.circle(orig, (int(xA), int(yA)), 5, color, -1)
                        cv2.circle(orig, (int(xB), int(yB)), 5, color, -1)
                        cv2.line(orig, (int(xA), int(yA)), (int(xB), int(yB)),color, 2)
                        # compute the Euclidean distance between the coordinates,
                        # and then convert the distance in pixels to distance in
                        # units
                        D = dist.euclidean((xA, yA), (xB, yB)) / refObj[2]
                        (mX, mY) = midpoint((xA, yA), (xB, yB))
                        cv2.putText(orig, "{:.1f}cm".format(D), (int(mX), int(mY -
10)),cv2.FONT_HERSHEY_SIMPLEX, 0.55, color, 2)
                cv2.imshow('Bottle mask ',orig)
        return image_binary4, Bottle_height, D1_body, Bottle_position
# ------------------------------------------------------------------------------------------------------
# Draw text result
def draw_textresult(image, label_exist, label_ok, liquid_ok):
        labelstatus = ""
        if (label_exist == False):
```

```
                labelstatus = "NO LABEL"
        else:
                if (label_ok == True):
                        labelstatus = "OK"
                elif (label_ok == False):
                        labelstatus = "BAD"
        cv2.putText(image,"Label position: {:.10s}".format(labelstatus),(10,20),cv2.FONT_HERSHEY_SIMPLEX,0.5,(0,146,152),2)
        # ------------------------------------------------------------
        liquidstatus = ""
        if (liquid_ok == True):
                liquidstatus = "OK"
        elif (liquid_ok == False):
                liquidstatus = "BAD"
        cv2.putText(image,"Liquid volume: {:.10s}".format(liquidstatus),(10,40), cv2.FONT_HERSHEY_SIMPLEX,0.5,(0,146,152),2)
        return image
# ------------------------------------------------------------------------------------------------------------
# Handle image to process image
def handle_image(img,rightlabel,rightliquid):
        right_liquid_max = rightliquid[0]
        right_liquid_min = rightliquid[1]
        S_good_label = rightlabel[0]
        label_rightbottom = rightlabel[1]
        label_rightheight = rightlabel[2]
        label_rightbottom_max = rightlabel[3]
        label_rightbottom_min = rightlabel[4]
        label_rightheight_max = rightlabel[5]
        label_rightheight_min = rightlabel[6]
        img_h, img_w,  _ = img.shape
        img_e = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        border = int(img_h * CUT_BORDER)
        image = img[:, border:(img_w -border)]
        image_h, image_w, _ = image.shape
        image_original = np.copy(image)
        pre_roi = get_roi(image, int(image_w * 0.2), int(image_w * 0.8))
```

```
        pre_mask = cv2.bitwise_not(find_background(image, pre_roi))


        kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
        pre_mask = cv2.morphologyEx(pre_mask, cv2.MORPH_OPEN, kernel)
        bin1, Bottle_height,Bottle_body,Bottle_position = find_bottle(image)
        left_border, right_border, roi_width = get_pre_borders (bin1)
        cut_img1 = image[:, (left_border - roi_width):(right_border + roi_width)]
        cut_img1_h, cut_img1_w, _ = cut_img1.shape
        bottle_test1 = cut_img1.copy()
        bottle_test2 = cut_img1.copy()
        bottle_test3 = cut_img1.copy()
        roi = get_roi(cut_img1, roi_width, cut_img1_w - roi_width)
        ROI_EXT = int(cut_img1_w * 0.05)
        cut_img2 = cut_img1[:, (roi_width - ROI_EXT):(cut_img1_w - roi_width +
ROI_EXT)]
        cut_img2_h, cut_img2_w, _ = cut_img2.shape
        ROI_EXT21 = int(cut_img2_w * 0.05)
        ROI_EXT22 = int(cut_img2_h * 0.05)
        cut_bin,Bottle_height1, _,Bottle_position1 = find_bottle(cut_img2)
        cut_img = cut_img2[(Bottle_position1[2] -
ROI_EXT22):(Bottle_position1[3]+ROI_EXT22) , (Bottle_position1[0]-
ROI_EXT21):(Bottle_position1[1]+ROI_EXT21)]
        bottle = cut_img
        bottle_h, bottle_w, _ = bottle.shape
        BottleHeight = (Bottle_position1[3] - Bottle_position1[2])
        # ----------------------------------------------------------------------
        # Check the quality of label
        # ----------------------------------------------------------------------
        labelexist = check_label_exist(bottle_test1,S_good_label)
        # Check the existence of label
        if (labelexist == True):
                print('Bottle has label')
        else:
                print('Bottle has no label')
        label_ok = False
        label_bottom = 0
```

```
        label_height = 0
        # Check the position and the height of label
        if (labelexist == True):
                minlabely,maxlabely = find_labels(bottle_test2)
                # print('min label y', minlabely)
                # print('max label y', maxlabely)
                label_bottom = 1 - ((maxlabely - Bottle_position1[2]) / (Bottle_position1[3] - Bottle_position1[2]))
                label_height = (maxlabely - minlabely) / (Bottle_position1[3] - Bottle_position1[2])
                print('label_bottom',label_bottom)
                print('label_height',label_height)
                if (label_bottom > label_rightbottom_min) and (label_bottom < label_rightbottom_max):
                        if (label_height > label_rightheight_min) and (label_height < label_rightheight_max):
                                label_ok = True
                print('Check if label is ok: ', label_ok)
        # ------------------------------------------------------------------
        # Check the liquid level
        # ------------------------------------------------------------------
        liquid_level_y = find_liquid_level(bottle_test3)
        liquid_level = 1 - ((liquid_level_y - Bottle_position1[2]) / (Bottle_position1[3] - Bottle_position1[2]))
        print('liquid level', liquid_level)
        liquid_ok = False
        if (liquid_level > right_liquid_min) and (liquid_level < right_liquid_max):
                liquid_ok = True
        print('Check if liquid level is ok: ', liquid_ok)
        recognize_info = label_height, label_bottom, liquid_level, labelexist, label_ok, liquid_ok
        # ------------------------------------------------------------------
        # Draw text result to image
        # ------------------------------------------------------------------
        result_image = draw_textresult(image,labelexist,label_ok,liquid_ok)
        cv2.imshow('the result of bottle checking', image)
        return result_image, recognize_info
```

```
# ------------------------------------------------------------------------------------------------
# Save image
# def save_img(result_image, result_path, file_name):
#------------------------------------------------------------------------------------------------
# Capture camera from webcam
def capture_cam(captures_path, result_path,right_label_matrix, right_liquid_matrix):
        cap = cv2.VideoCapture(0)
        cap.set(3, 1280)
        cap.set(4, 960)
        recognize_info = None
        while (True):
                ret, frame = cap.read()
                if not ret:
                        print("No capture")
                        break
                if recognize_info:
                        frame = draw_textresult(frame, recognize_info[3],recognize_info[4],recognize_info[5])
                cv2.imshow('Bottle check', frame)
                k = cv2.waitKey(100)
                if k == 27:  # Esc key to stop
                        break
                elif k == 32:  # Space key to recognize
                        # Upon the initial click of <Space> the current frame is
                        # recognized, the rendering of the results in the display
                        # window received from the webcam and saving of the images
                        # are performed.
                        # The second click on <Space> will clear the rendering of the
                        # results of recognition and make the process ready to repeat itself.
                        if recognize_info:
                                recognize_info = None
                        else:
                                file_name = datetime.now().strftime("%Y%m%d-%H%M%S-%f.png")
                                cv2.imwrite(os.path.join(captures_path, file_name), frame)
```

```
                result_image, recognize_info = handle_image(frame,right_label_matrix, right_liquid_matrix)
                # save_img(result_image, result_path, file_name)
                cv2.imwrite(os.path.join(result_path, file_name),result_image)
        cap.release()
        cv2.destroyAllWindows()
# ------------------------------------------------------------------------------------------------
# Take video from path and process
def capture_video(video_path, captures_path, result_path,right_label_matrix, right_liquid_matrix):
        cap = cv2.VideoCapture(video_path)
        cap.set(3, 1280)
        cap.set(4, 960)
        recognize_info = None
        while (True):
                ret, frame = cap.read()
                if not ret:
                        print("No capture")
                        break
                if recognize_info:
                        frame = draw_textresult(frame, recognize_info[3],recognize_info[4],recognize_info[5])
                cv2.imshow('Bottle check', frame)
                k = cv2.waitKey(100)
                if k == 27:  # Esc key to stop
                        break
                elif k == 32:  # Space key to recognize
                        # Upon the initial click of <Space> the current frame is
                        # recognized, the rendering of the results in the display
                        # window received from the webcam and saving of the images
                        # are performed.
                        # The second click on <Space> will clear the rendering of the
                        # results of recognition and make the process ready to repeat itself.
                        if recognize_info:
                                recognize_info = None
                        else:
```

```
                file_name = datetime.now().strftime("%Y%m%d-%H%M%S-%f.png")
                cv2.imwrite(os.path.join(captures_path, file_name), frame)
                frame = cv2.resize(frame, None, fx=0.7, fy=0.7, interpolation=cv2.INTER_CUBIC)
                result_image, recognize_info = handle_image(frame,right_label_matrix, right_liquid_matrix)
                # save_img(result_image, result_path, file_name)
                cv2.imwrite(os.path.join(result_path, file_name),result_image)
        cap.release()
        cv2.destroyAllWindows()
# ------------------------------------------------------------------------------------------------------------
def main():
        # ------------------------------------------------------------
        # Find the exact ratio between liquid level and the height of bottle
        # Find the exact ratio between the height of label and the height of bottle
        #------------------------------------------------------------
        right_ratio, Bottle_rightposition = right_liquid_level()
        right_liquid_max = right_ratio + 0.02
        right_liquid_min = right_ratio - 0.02
        right_liquid_matrix = right_liquid_max, right_liquid_min
        print('Ratio between the height of bottle and the liquid level', right_ratio)
        S_good_label,min_rightlabel_y, max_rightlabel_y = right_labels()
        S_good_label = int(S_good_label)
        # print('The area of good label', S_good_label)
        label_rightbottom = 1 - ((max_rightlabel_y - Bottle_rightposition[2]) / (Bottle_rightposition[3] - Bottle_rightposition[2]))
        label_rightheight = (max_rightlabel_y - min_rightlabel_y) / (Bottle_rightposition[3] - Bottle_rightposition[2])
        label_rightbottom_max = label_rightbottom + 0.03
        label_rightbottom_min = label_rightbottom - 0.03
        label_rightheight_max = label_rightheight + 0.01
        label_rightheight_min = label_rightheight - 0.01
        right_label_matrix = S_good_label, label_rightbottom, label_rightheight, label_rightbottom_max, label_rightbottom_min, label_rightheight_max, label_rightheight_min
        print('label_rightbottom_max ',label_rightbottom_max )
```

```
                print('label_rightbottom_min',label_rightbottom_min)
                print('label_rightheight_max',label_rightheight_max)
                print('label_rightheight_min',label_rightheight_min)
                capturepath = "D:\\Testimg\\Capture"
                resultpath = "D:\\Testimg\\Result"
                videopath = "D:\\Testimg\\Bottle_video(CC)\\video_test8.mp4"
                capture_video(videopath,capturepath,resultpath,right_label_matrix,
right_liquid_matrix)


if __name__ == "__main__":
    main()
```

## 3.4 Kết quả đạt được

Sau khi test với các trường hợp, mô hình của nhóm thực hiện đã đạt kết quả tốt, với tỉ lệ chính xác 100%, với loại chai Coca-Cola


Nhãn TỐT, mực nước TỐT


Nhãn TỐT, mực nước TỐT

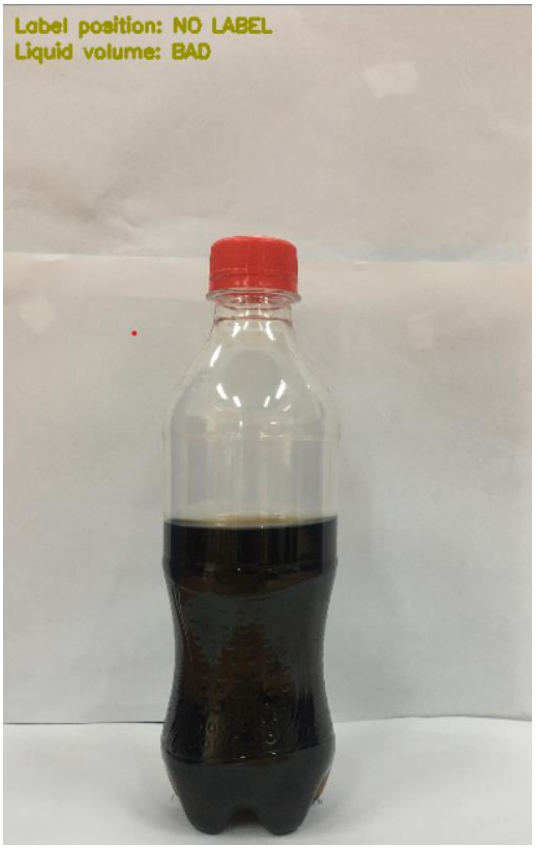

Nhãn TỐT, mực nước TỐT


Nhãn TỐT, mực nước KÉM


Nhãn TỐT, mực nước KÉM


Nhãn TỐT, mực nước KÉM


Nhãn KÉM, mực nước KÉM




Nhãn KHÔNG CÓ, mực nước KÉM


Nhãn KÉM, mực nước KÉM


Nhãn KÉM, mực nước KÉM