Here's a concrete **YOLO + homography** pipeline that (1) estimates **trash height** from the far wall, (2) estimates **bag distance** and **bag real-world size**, then (3) returns a **binary decision**: **≤ 13-gallon bag** vs **> 13-gallon bag**.

I'm going to make one practical assumption so this can be implemented end-to-end:

- You'll classify "13-gallon" by comparing the bag's **estimated real-world height** (or width) to a **threshold** you set from a few examples (because "13-gallon" bags vary a lot in shape when filled).
  You *can* make this much stronger by switching to **instance segmentation** (YOLO-seg) and using "area/outline" too.

A. What you train YOLO to detect

You'll get best results with **two models** (or one multi-head model, but simplest is two):

Model 1: Far-wall geometry model (detection)

Classes:

- `wall_corner` (or 4 separate classes: `corner_tl` , `corner_tr` , `corner_br` , `corner_bl`)
- `trashline_point` (optional; I usually do trashline with segmentation instead)

This model's job is: **find the far wall corners** reliably.

Model 2: Trash + bag model (segmentation recommended)

Classes:

- `trash_bag` (instance segmentation preferred)
- `trash_mass` (optional segmentation mask to find the trashline)

If you can only do detection (boxes), it still works, but segmentation makes "trashline" and "bag silhouette size" much more reliable.

B. One-time setup: define wall coordinates (real world)

You said the inside rectangle of the truck has width/height:

- Width = 15 ft
- Height = 10 ft

Define far-wall coordinate system:

- origin at bottom-left corner of far wall (floor contact)
- x rightward along the wall (0 → 15)
- y upward along the wall (0 → 10)

So the real coordinates of the far wall corners:

- $P_{BL} = (0, 0)$

- $P_{BR} = (15, 0)$
- $P_{TR} = (15, 10)$
- $P_{TL} = (0, 10)$

C. Pipeline overview (high level)

For each frame:

1. **Detect far-wall corners** (Model 1)
2. Compute **homography** H mapping **image pixels → wall feet**
3. Detect **trashline** (boundary where trash meets the wall)
4. Convert the trashline pixel row(s) into **trash height at the wall** H_t (feet)
5. Detect **trash bag** (Model 2) and pick a bag candidate
6. Estimate bag distance D / R from geometry
7. Estimate bag real size (height/width in feet) via homography + scaling
8. Compare to your "13-gallon" threshold ⇒ output **binary**

D. The core math

D1) Homography (image → wall)

Given 4 pixel corner points (u_i, v_i) and 4 real points (x_i, y_i),
compute:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim H \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

OpenCV does this with `cv2.findHomography(img_pts, wall_pts)`.

Once you have H, any pixel point maps to wall coordinates (feet):

$$(x, y) = \text{project}(H \cdot [u, v, 1]^T)$$

D2) Trash height at far wall

Find the "trashline" pixel locations on the far wall. Pick a representative point (u_{line}, v_{line}), map it through the homography to (x_{line}, y_{line}).

Then:

$$H_t \approx y_{line}$$

That's the trash height (feet) at the far wall.

D3) Bag distance using known camera height + tilt

Let:

- Camera height above floor: $H_c = 10$ ft
- Camera down-tilt: $\alpha = 20°$
- Trash surface height: use your estimate H_s

What should H_s be?

- The wall trash height H_t is the height *at the far wall*. The trash surface may slope.
- A practical assumption: **use** $H_s = H_t$ unless you model the slope.

Then:

$$\Delta H = H_c - H_s$$

Distance along surface (center ray):

$$D = \frac{\Delta H}{\tan(\alpha)}$$

Line-of-sight distance:

$$R = \frac{\Delta H}{\sin(\alpha)}$$

> If the bag is not centered, you can adjust using pixel row → ray angle, but homography-based sizing is usually enough for your binary classification.

D4) Bag real-world size (feet) from homography

This is the "magic" part: you can estimate the bag's **real-world size** without needing camera FOV.

If you have segmentation mask for the bag:

- Take the bag's bottom center pixel $p_b = (u_b, v_b)$
- Take the bag's top center pixel $p_t = (u_t, v_t)$
- Map both via homography to wall-plane coordinates:
  - $P_b = (x_b, y_b)$

- $P_t = (x_t, y_t)$

But note: the bag is not on the wall plane. So mapping directly is not physically exact.

Better method (recommended): local scale at bag depth
Use homography to estimate **a scale factor in ft/pixel** around the bag's location by mapping two nearby pixel points on the floor direction.

Example:

- Let $p_1 = (u_b, v_b)$
- Let $p_2 = (u_b + \delta, v_b)$ (small horizontal pixel shift, e.g. 20 px)
- Map both with homography to wall coords: P_1, P_2
- Compute:

$$s_x \approx \frac{\|P_2 - P_1\|}{\delta} \text{ ft/pixel}$$

Similarly for vertical pixel shift:

- $p_3 = (u_b, v_b - \delta)$ map to P_3

$$s_y \approx \frac{\|P_3 - P_1\|}{\delta} \text{ ft/pixel}$$

Then bag pixel height h_{px} (from segmentation bbox or mask) becomes:

$$H_{bag} \approx h_{px} \cdot s_y$$

and bag pixel width w_{px} becomes:

$$W_{bag} \approx w_{px} \cdot s_x$$

This gives you a depth-consistent "local scaling" using the known wall geometry.

E. Binary decision: "≤13-gal" vs ">13-gal"

Why volume is hard from one camera

A bag's *volume* depends on how it bulges. From a single view, you won't get true volume reliably.

So the practical approach is:

- choose a proxy measurement: **height** or **projected area**
- calibrate thresholds from real examples

Recommended rule

Use estimated real-world bag height H_{bag}:

$$\text{if } H_{bag} \le T_{13} \Rightarrow \text{“}\le\text{13-gallon” else “}>\text{13-gallon”}$$

Where T_{13} is a threshold you determine empirically.
A decent starting range for a "typical 13-gallon kitchen bag" when tied and filled might be ~1.8–2.3 ft tall, but you should set T_{13} using your actual operational definition.

You can also do a 2D rule:

- if H_{bag} and W_{bag} both exceed thresholds ⇒ ">13"

F. Concrete step-by-step pipeline (what you implement)

Step 0 — Load models

- `yolo_wall.pt` (corner detection)
- `yolo_bag_seg.pt` (bag segmentation)
- (optional) `yolo_trash_seg.pt` (trash mask)

Step 1 — Detect far wall corners

- Run `yolo_wall` on frame
- Extract 4 corner points
- If using a single `wall_corner` class, cluster and assign TL/TR/BR/BL by sorting (y then x)

Step 2 — Compute homography

- `img_pts = [TL, TR, BR, BL]` (pixel coords)
- `wall_pts = [(0,10), (15,10), (15,0), (0,0)]` (feet coords)
- Compute H

Step 3 — Estimate trash height at wall

Two options:

Option 3A: segmentation of trash mass

- Run trash segmentation
- Restrict mask to the far wall ROI
- Find the highest "trash pixels" that touch the wall (trashline)
- Pick median x along wall, get v_{line}

Option 3B: edge-based (if no trash seg)

- Warp (rectify) the wall using the homography into a 15×10 ft "wall image"
- In this rectified view, find the y coordinate where the wall stops being visible (trash begins)

- Convert to feet directly (because rectified wall is now in wall coordinates)

Then set:

$$H_t = y_{line}$$

Step 4 — Detect bag and choose candidate

- Run bag segmentation
- Choose the bag with highest confidence (or the closest one / largest pixel area)

Get:

- bag bbox pixel height h_{px}
- bag bbox pixel width w_{px}
- bottom-center pixel (u_b, v_b)

Step 5 — Compute local ft/pixel scale near the bag using homography

Pick $\delta = 20$ px.

Map:

- $p_1 = (u_b, v_b)$
- $p_2 = (u_b + \delta, v_b)$
- $p_3 = (u_b, v_b - \delta)$

Convert using H to P_1, P_2, P_3 in feet.

Compute:

$$s_x = \frac{\|P_2 - P_1\|}{\delta}, \quad s_y = \frac{\|P_3 - P_1\|}{\delta}$$

Estimate:

$$H_{bag} = h_{px} \cdot s_y, \quad W_{bag} = w_{px} \cdot s_x$$

Step 6 — Binary classify

$$\text{result} = (H_{bag} > T_{13})$$

Return:

- `True` = “>13 gallon”
- `False` = “≤13 gallon”

Also return confidence score:

- based on bag detection confidence
- wall corner confidence
- stability of H_t over last N frames

G. Pseudocode (Python + OpenCV + YOLO)

```
import cv2 import numpy as np WALL_W, WALL_H = 15.0, 10.0 # feet T13 = 2.2 # ft example threshold; calibrate with your data! DELTA = 20 # pixels for local scale wall_pts = np.array([ [0.0, WALL_H], # TL [WALL_W, WALL_H], # TR [WALL_W, 0.0], # BR [0.0, 0.0] # BL ], dtype=np.float32) def project_point(H, u, v): p = np.array([u, v, 1.0], dtype=np.float32) q = H @ p return (q[0]/q[2], q[1]/q[2]) # feet def get_homography_from_corners(corners_px): # corners_px must be TL, TR, BR, BL in pixels img_pts = np.array(corners_px, dtype=np.float32) H, _ = cv2.findHomography(img_pts, wall_pts, method=0) return H def estimate_local_scale(H, u, v, delta=DELTA): x1,y1 = project_point(H, u, v) x2,y2 = project_point(H, u+delta, v) x3,y3 = project_point(H, u, v-delta) sx = np.hypot(x2-x1, y2-y1) / delta sy = np.hypot(x3-x1, y3-y1) / delta return sx, sy def classify_bag(frame, wall_corners_px, bag_bbox, bag_conf): """ wall_corners_px: [(uTL,vTL),(uTR,vTR),(uBR,vBR),(uBL,vBL)] bag_bbox: (x1,y1,x2,y2) pixels """ H = get_homography_from_corners(wall_corners_px) x1,y1,x2,y2 = bag_bbox w_px = max(1, x2-x1) h_px = max(1, y2-y1) u_b = (x1 + x2) / 2.0 v_b = y2 # bottom of box sx, sy = estimate_local_scale(H, u_b, v_b) H_bag = h_px * sy # feet W_bag = w_px * sx # feet is_over_13 = (H_bag > T13) return { "over_13": bool(is_over_13), "H_bag_ft": float(H_bag), "W_bag_ft": float(W_bag), "bag_conf": float(bag_conf), "sx_ft_per_px": float(sx), "sy_ft_per_px": float(sy), }
```

H. Notes that matter in the real hopper

1. **Trashline height isn't the same everywhere** (trash slopes).
   You'll get best results if you estimate a **trash surface plane** by sampling trashline at multiple x positions.
2. **Corner detection must be stable.**
   Use temporal smoothing: average corners over last 10–30 frames.
3. **Segmentation beats boxes.**
   If you can, use YOLO-seg for bags and trash mass.
4. **Calibrate the 13-gal threshold** from your own data.
   - Collect 50–100 examples of "13-gal or smaller" and "bigger than 13-gal"
   - Compute estimated H_{bag} and pick a threshold that minimizes false positives/negatives