Computer Vision for the Project

Functional Requirements:

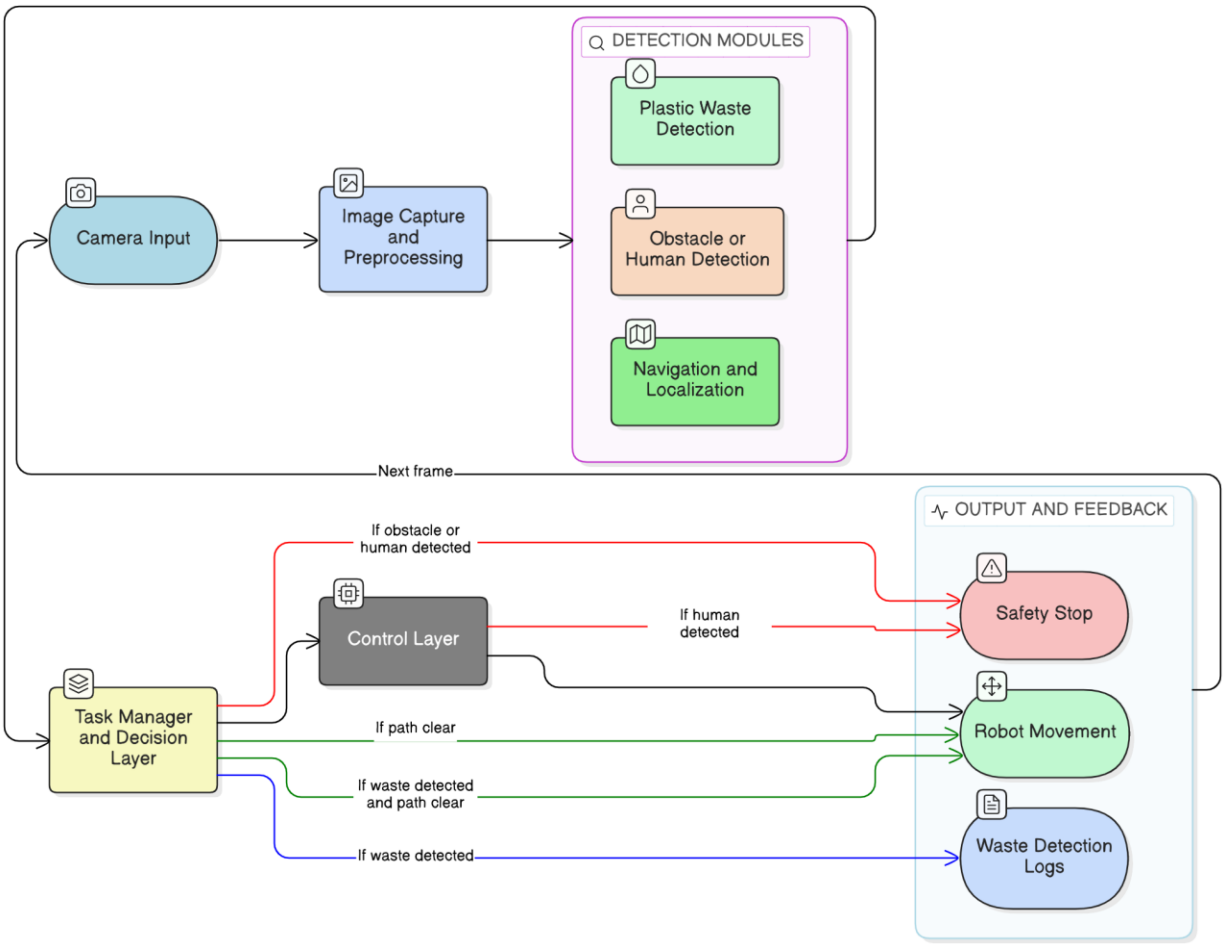
a) Detecting plastic/waste (Core)
b) Navigating - Localizing the waste (Supporting-must)
c) Obstacle Avoidance (If possible!)

Solutions Requirements:

a) Three different solutions; with detailed specifications.
b) Decide some parameters in order to evaluate the solutions.

Parameters:
Accuracy
Response time
Cost

Solution Pipeline:



1.Traditional Computer Vision (Color + Shape + Contour Detection) - OpenCV only

Process:
The captured image is converted to HSV color space. Thresholds are applied for common plastic colors (blue, green, white, transparent). Contours are extracted from the binary mask, then filtered by geometric properties such as area, aspect ratio (for bottles), circularity (for caps), and rectangularity (for wrappers/bags).

Advantages:

- Very fast and lightweight, suitable for Raspberry Pi without accelerators.
- No dataset or training required.
- Easy to debug and explain.

Limitations:

- Sensitive to lighting changes and shadows.

- Confuses similar-colored natural objects (e.g., blue leaves).
- Accuracy typically limited to ~60–75% outdoors.

Resources: Raspberry Pi + camera, OpenCV, picamera2.
Best Use Case: Quick baseline prototype, early testing, or when power budget is tight.

2. Lightweight Object Detection Model (YOLOv8n or MobileNet SSD)

How it works:
Use a pre-trained lightweight detector (YOLOv8n or MobileNet SSD) → fine-tune on a small dataset (300–600 images, e.g., TACO dataset) → run inference → bounding boxes + class labels → filter plastics only.

Pros:
Higher accuracy (80–90% after fine-tuning).
Provides location (bounding box) for robotic pickup.
More robust to lighting changes.

Cons:
Requires dataset collection and training.
Higher CPU/GPU usage (but still feasible on Pi 4/5).
Slightly larger model size (~5–10 MB).

Resources needed: Raspberry Pi 4/5, TensorFlow Lite or Ultralytics YOLOv8, small labeled dataset.
Best for: Utilized solution if accuracy and localization are top priorities.

3. Hybrid Approach: Color Segmentation + Edge AI Classification

How it works:
Use OpenCV color thresholding + contour detection to find candidate regions → crop patches → run a small CNN classifier (MobileNetV2/EfficientNet-Lite) → confirm "plastic" vs "non-plastic."

Pros:
Combines speed of CV with accuracy of ML.
Reduces false positives compared to pure CV.
More efficient than running ML on full frames.

Cons:
More complex pipeline.
Requires careful threshold tuning.

Resources needed: Raspberry Pi + camera, OpenCV, TensorFlow Lite (tiny classifier ~2–4 MB).

Best for: Balanced performance when compute is limited but accuracy matters.

Comparison Table

Method	How it Works	Pros	Cons	Resources Needed	Best For
1. Traditional CV (OpenCV)	HSV color thresholding + contour + shape filters	Very fast, no training, low CPU, easy to debug	Sensitive to lighting, lower accuracy (~60–75%), false positives	Raspberry Pi + camera, OpenCV	Baseline prototype, early testing
2. Lightweight ML (YOLOv8n / MobileNet SSD)	Pre-trained detector fine-tuned on small dataset → bounding boxes	High accuracy, robust to lighting, gives location	Needs dataset & training, higher CPU load	Raspberry Pi 4/5, TensorFlow Lite, dataset	Utilized solution, accuracy + localization
3. Hybrid (CV + Tiny CNN)	CV pre-filter → cropped patches → small CNN classifier	Balanced speed + accuracy, fewer false positives	More complex pipeline, threshold tuning needed	Raspberry Pi + camera, OpenCV + TFLite	Balanced performance, outdoor robustness