Intelligent Waste Sorting System

Class Structure Design Document

---

1. Overview

This intelligent waste sorting system is designed for embedded platforms like Raspberry Pi. It integrates:

- 📷 Image acquisition via camera
- 🧠 Object detection using YOLOv5
- ⏱️ Task scheduling using timers
- ⚙️ Stepper motor control for physical sorting

The architecture follows **SOLID object-oriented design principles**, ensuring high maintainability, testability, and scalability.

---

2. SOLID Principles in Practice

Principle	Implementation in the System
SRP – Single Responsibility	Each class is responsible for one task. CameraCapture handles image input only; YoloDetector only detects; GarbageSorter controls sorting logic.
OCP – Open/Closed	System allows extension (e.g., new waste types) without modifying core logic. Motors, image sources, and models are replaceable.
LSP – Liskov Substitution	StepperMotor can be replaced with a subclass (e.g., servo or virtual motor) without affecting functionality.
ISP – Interface Segregation	Minimal interfaces are exposed (e.g., only captureImage()), reducing unnecessary dependencies.
DIP – Dependency Inversion	High-level logic depends on abstractions (like callbacks), not concrete classes. Detection logic is decoupled from the result processing logic.

---

3. Module Structure & Responsibilities

📷 CameraCapture

- Responsible for capturing images from the camera and saving them.
- Easily replaceable with virtual sources or video feeds.
- **Applies**: SRP, OCP

📝 CameraWorker

- Combines camera capture with periodic scheduling using a Timer.
- Runs in a background thread.
- **Applies**: SRP, ISP

🧠 YoloDetector

- Encapsulates YOLOv5 model loading and inference using ncnn.
- Provides results asynchronously via callback.
- **Applies**: SRP, DIP, OCP

🧺 GarbageSorter

- Coordinates stepper motors based on detection results.
- Manages system state and sorting logic.
- **Applies**: SRP, OCP, DIP

⚙️ StepperMotor

- Controls individual motors via GPIO pins.
- Can be extended or replaced (e.g., with PWM, servo).
- **Applies**: SRP, LSP

⏱️ Timer / scheduleTask

- Provides both periodic and delayed task mechanisms.
- Decouples timing from logical operations.
- **Applies**: SRP, OCP

🧩 ThreadController

- Starts/stops the detection process in a managed thread.
- Supports callback for flexible integration.
- **Applies**: SRP, DIP

📦 **shared_data.h**

- Thread-safe shared queues for images and detection results.
- Modular communication across producers and consumers.
- **Applies**: SRP

---

4. Decoupled Design & Interfaces

- Detection results are passed via callback interfaces → no tight coupling to business logic.
- Camera + Timer modules are independently testable and replaceable.
- Motors expose simple rotate() and release() interfaces for precise control.
- Thread lifecycle is abstracted away from logic via ThreadController.
- Shared data structures are synchronized for safe access.

---

5. Extensibility & Maintainability

- 🔁 **Model Upgrade**: Swap YOLO versions by changing model paths.
- ➕ **More Waste Types**: Extend WasteType enum and motor map.
- 🔄 **Alternate Motor Controllers**: Replace StepperMotor with subclass.
- 🖥️ **UI Integration**: System status callback can feed UI / web dashboards.
- ☑️ **Testing**: Each module is testable independently due to clear responsibility.

---

6. Summary

This system showcases a clean, scalable embedded architecture applying real-world **SOLID principles**.
Its modular and extensible design makes it suitable for:

- 🔬 Education
- 🧪 Research Prototypes
- 🏭 Industrial Sorting Systems

It promotes long-term maintainability and seamless integration of upgrades and hardware changes.