**👤 Person C – Sensorik, Steuerlogik & Gerätesteuerung**

**🏁 Ziel:**

Entwicklung und Umsetzung der gesamten **intelligenten Steuerlogik und Sensorik** des Systems auf Mikrocontroller-Ebene, inklusive:

* Sensorintegration
* Steuerungslogik
* Fehlerbehandlung
* Kommunikation
* OTA-Updatefähigkeit
* Lokales Regelwerk zur Sortierentscheidung

**📋 Aufgabenübersicht (in Themenblöcken)**

**📡 1. Sensorik & Hardwareeinbindung**

**Ziel**: Messung von Füllstand, Gewicht und Benutzeridentifikation

**Teilaufgaben:**

* Auswahl und Einbindung folgender Sensoren:
  + **Ultraschallsensoren** zur Füllstandmessung (z. B. HC-SR04 oder VL53L0X)
  + **Wägezelle + HX711** zur Gewichtsmessung in der Einwurfschale
  + **NFC-Modul** (z. B. PN532) zur Authentifizierung von Wartungspersonal
* Bau eines **Sensor-Teststandes** zur Kalibrierung & Messvalidierung
* Konfiguration und Fehlerkorrektur bei Mehrfachmessung (z. B. Messwert-Median)

**🔄 2. Zentrale Steuerlogik & Aktorsteuerung**

**Ziel**: ESP32 übernimmt das Kommando – basierend auf Sensor- und KI-Daten entscheidet er autonom

**Teilaufgaben:**

* Aufbau der **zentralen Sortier-Logik**:
  + Verarbeitung von Kameradaten (z. B. JSON von Person B)
  + Kombination mit Gewicht zur Validierung (z. B. PET darf max. 200 g wiegen)
* Steuerung der Mechanik:
  + **X/Y/Z-Positionierung der Schale** (z. B. über Stepper oder Servos)
  + **Auslösung der Klappenmechanik**
* Fehlerbehandlung:
  + Verstopfungserkennung (z. B. Gewicht bleibt trotz Klappenöffnung hoch)
  + Reaktion auf Füllstand = 100 %

**🧠 3. Konfigurierbares Regelwerk**

**Ziel**: Entscheidungslogik wird nicht hart gecodet, sondern dynamisch aus z. B. JSON-Config gelesen.

**Teilaufgaben:**

* Erstellen eines einfachen **Regelwerks** z. B. als JSON-Datei:
  + max. Gewicht pro Müllart
  + kritische Füllstandswerte
  + Verhalten bei Fehler (Retry, Ignorieren, Alarm)
* Regelparser auf ESP32 schreiben
* Konfiguration kann später über WLAN aktualisiert werden

**📲 4. Kommunikation & API-Anbindung**

**Ziel**: Verbindung zur Cloud und Web-App

**Teilaufgaben:**

* **MQTT-Client oder HTTP REST** auf ESP32:
  + Sendet regelmäßig Füllstand, Gewicht, Mülltyp
  + Empfängt z. B. Kommandos (z. B. „reset“, „update config“)
* Integration mit Person D:
  + Datenstruktur festlegen (JSON-Format, Topics, Endpunkte)
* Lokale Datenspeicherung bei Verbindungsausfall (Failsafe)

**🔁 5. OTA (Over-the-Air) Updatefähigkeit**

**Ziel**: Der Mikrocontroller soll wartbar sein – ohne Öffnen des Gehäuses

**Teilaufgaben:**

* Einrichtung eines OTA-Updatemechanismus (ESP32 OTA)
* Bereitstellung einer einfachen Web-GUI zur Datei-Upload
* Logik: Update nur im Wartungsmodus zulassen (via NFC)

**🔐 6. Wartungs- & Diagnosesystem**

**Ziel**: Hausmeisterfreundliches Interface + automatisierte Fehlerberichte

**Teilaufgaben:**

* Erkennung typischer Fehlerzustände (z. B. Klappenfehler, Verbindungsabbruch)
* Senden eines **automatischen Fehlerreports** (z. B. via HTTP oder IFTTT)
* Integration eines NFC-gestützten „Wartungszugangs“:
  + NFC-Karte → Admin-Modus aktiviert
  + Manueller Klappenöffnungsmodus
  + Sensor-Kalibrierung möglich

**🔗 Schnittstellen (Koordination mit anderen Personen)**

| **Teammitglied** | **Benötigte Daten** | **Gelieferte Schnittstelle** |
| --- | --- | --- |
| **Person A** | Positionen der Behälter, Aktoransteuerung | Steuerung von X/Y/Z + Klappen |
| **Person B** | Klassifizierte Müllart (Label + Confidence) | ESP32 übernimmt Steuerung & Reaktion |
| **Person D** | Visualisierung / API für App | JSON-Datenstrom, Status, Sensorwerte |

**📦 Erwartetes Ergebnis am Ferienende**

* ESP32 läuft stabil mit Sensorik, Steuerung, Kommunikation
* OTA funktioniert
* REST/MQTT ist implementiert & getestet
* Sortierlogik reagiert zuverlässig auf KI + Sensorwerte
* Diagnosesystem aktiv
* Konfigurierbare Regeln funktionieren