1. Introduction

Every year, inefficient waste-collection leads to unnecessary pickups, added CO_2 emissions, and overflowing public bins. Our **Smart Trash-Bin Fill-Level Monitoring System** uses a VL53L0X Time-of-Flight sensor mounted inside a 41 × 35 × 60 cm^3 bin to measure the fill height, displays the percentage full on an SH1106 OLED, and—when a user-configurable threshold is exceeded—sends alerts via a Telegram bot.

In future iterations, we plan to support fully outdoor operation by adding:

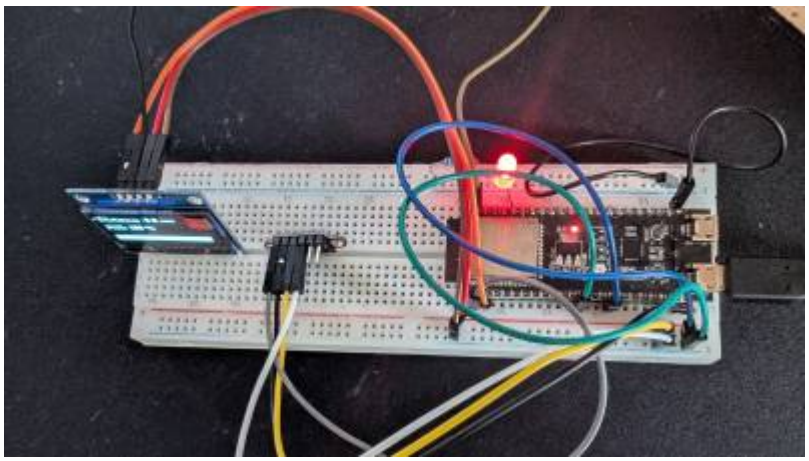
- A GPS module (for geo-tagged alerts)
- A solar-rechargeable Li-Po power supply (with charge controller)
- A weatherproof 3D-printed enclosure

This document walks through each step—hardware assembly (breadboard layout), Arduino IDE firmware, Node-RED flow, Telegram-bot config, and ESP32 simulation—so you can reproduce and extend the system today, then evolve it for outdoor use tomorrow.

2. Materials & System Overview

2.1. Hardware Components

Component	Purpose
ESP32-S3-DevKitC-1	Main MCU, Wi-Fi, GPIOs
VL53L0X ToF sensor	Measures distance from bin top to contents
SH1106 128×64 I^2C OLED (U8g2 lib)	Displays distance (cm) & fill (%)
LED (GPIO 2)	Visual “almost full” warning
Powerbank (5 V USB)	Supplies 5 V to ESP32 for portable operation
Wires, breadboard, connectors	Prototyping and wiring



2.2. Software Components

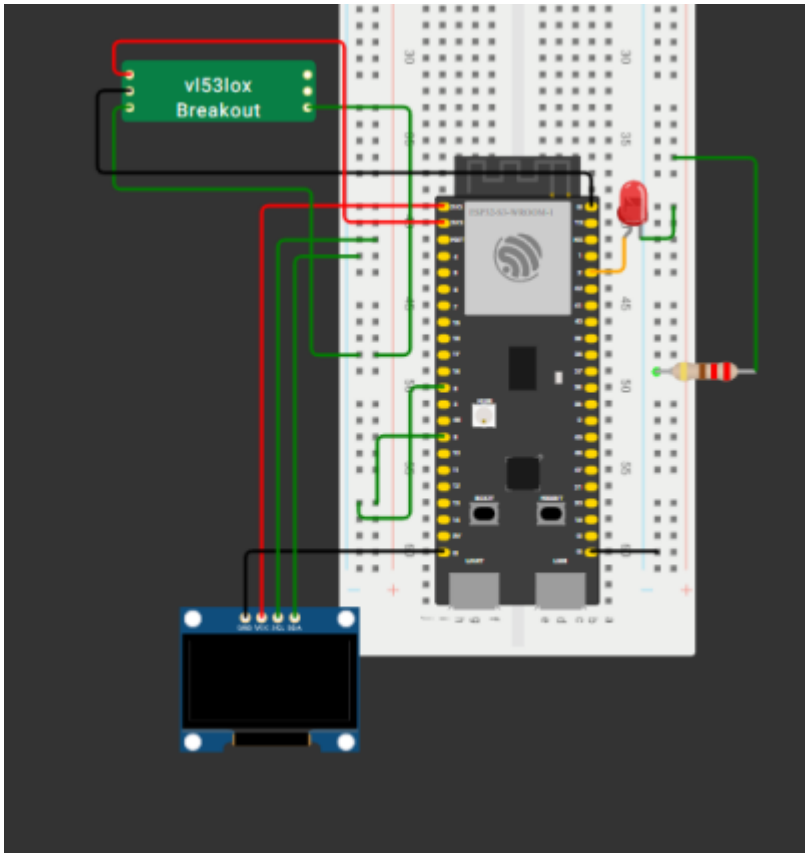
- **Arduino IDE** (v2.x)

- **Libraries**
  - * * Adafruit_VL53L0X – Time-of-Flight sensor
  - * * U8g2lib – SH1106 OLED driver
  - * * WiFi.h / HTTPClient.h – Wi-Fi & HTTP POST
  - * * UniversalTelegramBot.h – Telegram Bot API
- **Node-RED** (v3.x) on 192.168.10.50:1880 — receives HTTP alerts, dashboards fill percentage, logs events
- **Telegram Bot** (“TrashAlertBot”) configured with token `xxxx:YYYY` and chat ID

3. Hardware Assembly

3.1. Breadboard Layout

- **ESP32**: VIN ← 5 V (from powerbank USB→5 V regulator), GND ← GND, SDA ← GPIO 8, SCL ← GPIO 9.
- **VL53L0X**: VCC ← 3.3 V (ESP32 3V3 pin), GND ← GND, SDA/SCL as below.
- **OLED (SH1106)**: VCC ← 3.3 V, GND ← GND, SDA/SCL as below.
- **LED**: Anode ← GPIO 2 (with 220 Ω resistor), Cathode ← GND.



4. Arduino IDE Firmware

Below is the main sketch. **Please replace** `YOUR_SSID`, `YOUR_PASS`, `NODE_RED_URL`, and

`BOT_TOKEN` with your actual credentials.

```
#include <Wire.h>
#include <Adafruit_VL53L0X.h>
#include <U8g2lib.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include <UniversalTelegramBot.h>

#define I2C_SDA 8
#define I2C_SCL 9
#define LED_PIN 2

#define BIN_HEIGHT_CM 60.0
#define DISTANCE_OFFSET_CM -3.0

// Wi-Fi
const char* ssid       = "YOUR_SSID";
const char* password   = "YOUR_PASS";
// Node-RED endpoint
const char* alert_url  = "http://YOUR_IP_ADDRESS/bin-alert";

// Telegram
const char* telegram_token = "BOT_TOKEN";
String chat_id = "CHAT_ID";

U8G2_SH1106_128X64_NONAME_F_HW_I2C display(U8G2_R0, U8X8_PIN_NONE, I2C_SCL,
I2C_SDA);
Adafruit_VL53L0X lox = Adafruit_VL53L0X();
WiFiClientSecure secured_client;
UniversalTelegramBot bot(telegram_token, secured_client);

unsigned long lastAlertTime = 0, lastSignalTime = 0;
const unsigned long signalInterval = 30000;
float lastDistance = 0, lastPercentage = 0;
bool updatesEnabled = true;

void setup() {
  Serial.begin(115200);
  Wire.begin(I2C_SDA, I2C_SCL);
  // Initialize display
  display.begin();
  display.clearBuffer();
  display.setFont(u8g2_font_ncenB08_tr);
  display.drawStr(0,10,"Init Display");
  display.sendBuffer();
  // Initialize sensor
  if (!lox.begin()) while (1);
  pinMode(LED_PIN, OUTPUT);
  // Connect Wi-Fi
  WiFi.begin(ssid, password);
```

```
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print('.');
  }
  Serial.println("\\nWiFi OK");
  secured_client.setInsecure();
}

void loop() {
  VL53L0X_RangingMeasurementData_t m;
  lox.rangingTest(&m, false);
  if (m.RangeStatus != 4) {
    float d = m.RangeMilliMeter/10.0 + DISTANCE_OFFSET_CM;
    float p = constrain(100.0 - (d/BIN_HEIGHT_CM)*100.0, 0.0, 100.0);
    lastDistance = d; lastPercentage = p;
    // Update OLED
    display.clearBuffer();
    display.setCursor(0,12);
    display.print("Dist: "); display.print(d,1); display.print("cm");
    display.setCursor(0,30);
    display.print("Fill: "); display.print((int)p); display.print("%");
    int w = map((int)p,0,100,0,120);
    display.drawFrame(0,45,120,10);
    display.drawBox(0,45,w,10);
    display.sendBuffer();
    // Alert on ≥80%
    if (p >= 80.0) {
      digitalWrite(LED_PIN, HIGH);
      if (millis() - lastAlertTime > 15000) {
        sendWarning(d, p);
        lastAlertTime = millis();
      }
    } else {
      digitalWrite(LED_PIN, LOW);
    }
  }
  // Periodic update
  if (millis() - lastSignalTime > signalInterval) {
    if (updatesEnabled) sendRegularUpdate(lastDistance, lastPercentage);
    lastSignalTime = millis();
  }
  // Telegram commands
  static unsigned long lastBot = 0;
  if (millis() - lastBot > 1000) {
    int n = bot.getUpdates(bot.last_message_received + 1);
    while (n) {
      handleNewMessages(n);
      n = bot.getUpdates(bot.last_message_received + 1);
    }
    lastBot = millis();
```

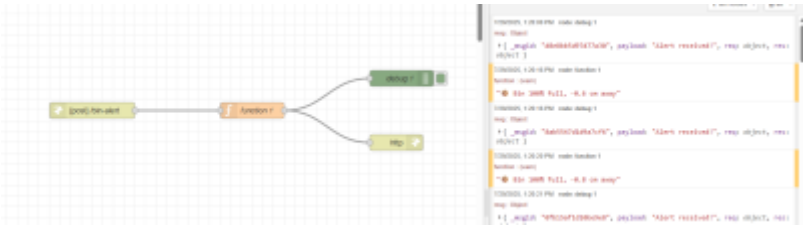
```
  }
  delay(500);
}

// ... (sendWarning, sendRegularUpdate, handleNewMessages functions)
```

5. Node-RED Flow

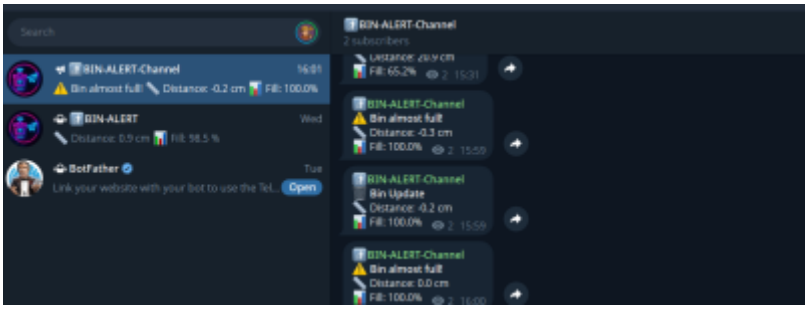
Our Node-RED instance (192.168.10.50:1880) handles incoming HTTP POSTs at `/bin-alert` and:

- Parses JSON (distance, fill_percentage)
- Switch: if `fill_percentage ≥ 80` → trigger email/SMS, else log
- Dashboard: updates a gauge node



6. Telegram Bot Configuration

- Create bot with BotFather → get `BOT_TOKEN`.
- Invite to your group/channel → note the `chat_id`.
- Grant it message-reading rights.



Commands:

- /start - Start bot
- /status - Get current bin fill
- /stop - Stop regular update messages
- /startupdates - Resume regular updates
- /help - Show this message

7. Results

- **OLED display**: real-time distance & fill bar (tested up to 85%).
- **LED**: lights when fill ≥ 80%.
- **Telegram**: immediate alert with bin status.
- **Node-RED dashboard**: gauge plotting fill percentage.





8. Discussion & Lessons Learned

- **Portable power**: using a USB powerbank delivers ~8 hrs runtime; for longer operation, a solar-powered Li-Po pack is recommended.
- **Connectivity**: outdoors Wi-Fi drops; consider fallback via GSM/LTE or LoRaWAN.
- **Enclosure**: no weatherproof housing yet—future 3D-printed case should protect electronics from dust and moisture.
- **Sleep modes**: ESP32 deep-sleep between measurements can drastically reduce power draw.
- **Multi-sensor**: adding ambient temperature/humidity could enable smarter waste-decomposition predictions.
- **Predictive analytics**: log historical fill data (via MQTT or cloud DB) to forecast optimal pickup schedules.
- **Firmware OTA**: integrate over-the-air updates for remote code maintenance.
- **Scalability**: mesh-network multiple bins to central server for fleet management.

9. Conclusion

This project provides a low-cost IoT solution for bin monitoring today, and a clear roadmap—GPS,

solar power, enclosure, analytics—for robust outdoor deployment tomorrow.

10. References

- **STMicroelectronics VL53L0X Datasheet**
- **U8g2 SH1106 OLED driver** – https://github.com/olikraus/u8g2
- **UniversalTelegramBot Library** – https://github.com/witnessmenow/Universal-Arduino-Telegram-Bot
- **Node-RED Documentation** – https://nodered.org/docs/

From:
https://student-wiki.eolab.de/ - **HSRW EOLab Students Wiki**

Permanent link:
https://student-wiki.eolab.de/doku.php?id=amc:ss2025:group-t:start&rev=1753790876

Last update: **2025/07/29 14:07**