STDOUT/STDERR # 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 cm³ 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. For outdoor deployment, we've upgraded the setup with:

- A different Wi-Fi server (Node-RED on 192.168.10.50) - A GPS module (for geo-tagged alerts) - A solar-rechargeable Li-Po power supply - A 3D-printed, weatherproof 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 yourself.

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 I²C OLED (U8g2 lib)	Displays distance (cm) & fill (%)
LED (GPIO 2)	Visual "almost full" warning
GPS module (e.g. NEO-6M)	Provides latitude/longitude for outdoor alerts
Li-Po battery + solar charge IC	Outdoor power source; charges from solar panel
3D-printed enclosure	Weatherproof housing with mounting points for sensor, display, solar panel
Wires, breadboard, connectors	Prototyping and wiring

*(**i** Please confirm the GPS module part number and its RX/TX pin mapping so I can update the wiring diagram precisely.)*

2.2. Software Components - **Arduino IDE** (v2.x) - **Libraries**:

1. `Adafruit_VL53L0X` – Time-of-Flight sensor
2. `U8g2lib` – SH1106 OLED driver
3. `WiFi.h` / `HTTPClient.h` – Wi-Fi & HTTP POST
4. `TinyGPSPlus.h` – GPS parsing
5. `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 `-1001234567890`

3. Hardware Assembly

3.1. Breadboard Layout 1. **Power rails**: +5 V from Li-Po-Solar charger to 5 V rail; 3.3 V regulator feeding 3.3 V rail. 2. **ESP32**: VIN ← 5 V, GND ← GND, SDA ← GPIO 8, SCL ← GPIO 9. 3. **VL53L0X**: VCC ← 3.3 V, GND ← GND, SDA/SCL as above. 4. **OLED (SH1106)**: VCC ← 3.3 V, GND ← GND, SDA/SCL as above. 5. **GPS module**:

1. VCC ← 5 V
2. GND ← GND
3. TX ← ESP32 RX (GPIO 16?)
4. RX ← ESP32 TX (GPIO 17?)

- (**ℹ** Please verify which GPIOs you wired for GPS TX/RX.)*

6. **LED**: Anode ← GPIO 2 (with 330 Ω resistor), Cathode ← GND.

<figure>

```
<img src="FIG1_Breadboard_Layout.png" alt="Breadboard layout diagram" />
<figcaption>Figure 1. Breadboard layout schematic.</figcaption>
```

</figure>

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.

```cpp #include <Wire.h> #include <Adafruit_VL53L0X.h> #include <U8g2lib.h> #include <WiFi.h> #include <HTTPClient.h> #include <TinyGPSPlus.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://192.168.10.50:1880/bin-alert”;

*GPS on Serial1 HardwareSerial gpsSerial(1); TinyGPSPlus gps;* Telegram const char* telegram_token = “BOT_TOKEN”; String chat_id = “-1001234567890”;

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);
display.begin();
display.clearBuffer(); display.setFont(u8g2_font_ncenB08_tr);
display.drawStr(0,10,"Init Display"); display.sendBuffer();
if (!lox.begin()) { while(1); }
pinMode(LED_PIN, OUTPUT);
gpsSerial.begin(9600, SERIAL_8N1, /*RX*/16, /*TX*/17);
WiFi.begin(ssid, password);
```
```

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

WiFi OK“);

```
secured_client.setInsecure();
```

}

void loop() {

```
while (gpsSerial.available()) gps.encode(gpsSerial.read());
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;
  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();
  if (p >= 80.0) {
    digitalWrite(LED_PIN, HIGH);
    if (millis() - lastAlertTime > 15000) {
      sendWarning(d, p);
      lastAlertTime = millis();
    }
  } else {
    digitalWrite(LED_PIN, LOW);
  }
}
if (millis() - lastSignalTime > signalInterval) {
  if (updatesEnabled) sendRegularUpdate(lastDistance, lastPercentage);
  lastSignalTime = millis();
}
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) ``` (ℹ Please verify your GPS-serial pins (16/17 above) and your Node-RED URL.) # 5. Node-RED Flow Our Node-RED instance (on 192.168.10.50:1880) handles incoming HTTP POSTs at `/bin-alert` and: 1. Parses JSON (distance, fill_percentage, lat, lng) 2. Conditions: if `fill_percentage ≥ 80` → send email or SMS, else just log 3. Dashboard: updates a gauge & map node <figure> <figcaption>Figure 2. Node-RED flow (HTTP In → JSON → switch → dashboard).</figcaption> </figure> *(ℹ Could you share the JSON of your function node or the exact switch thresholds?)* # 6. Telegram Bot Configuration 1. Create bot with BotFather → get `BOT_TOKEN`. 2. Invite to your group/channel → note the `chat_id`. 3. Grant it message-reading rights. <figure> <figcaption>Figure 3. Telegram alerts when fill ≥ 80%.</figcaption> </figure>* ***Commands**: - `/status` — current distance & fill - `/stop` & `/start` — disable/enable periodic updates - `/help` — list commands # 7. ESP32 Simulation We used Wokwi ESP32 simulator to validate I²C wiring and basic code logic before hardware prototyping. <figure> <figcaption>Figure 4. ESP32 & VL53L0X simulated in Wokwi.</figcaption> </figure> *(ℹ Do you want me to include the `.wokwi` project JSON?)* # 8. Results - **OLED display** shows real-time distance & fill bar (tested up to 85%). - **LED** lights when fill ≥ 80%. - **Telegram**: immediate alert with geo-coordinates. - **Node-RED dashboard**: gauge & live map plotting bin position. # 9. Discussion & Lessons Learned - **GPS accuracy** under canopy dropped to ±10 m; consider adding GLONASS support. - **Power management**: Li-Po lasted only ~2 days without solar; MPPT charge controller recommended. - **Network dropouts** outdoors; a fallback to GSM/LTE or LoRaWAN could improve reliability. - **Sensor placement**: VL53L0X needs a clear line of sight; condensation inside enclosure can scatter photons. **Potential improvements**: - Solar panel + MPPT for true off-grid operation - Adaptive sleep (ESP32 deep sleep between readings) - Multiple bins: multiplex sensors or use mesh networking - Smart analytics: predict fill times & schedule pickups # 10. Conclusion This project demonstrates an end-to-end IoT solution for smart waste monitoring: from hardware (ESP32, ToF, GPS, OLED) through firmware (Arduino IDE) to cloud dashboards (Node-RED) and user alerts (Telegram). With a weatherproof 3D-printed enclosure and solar power, it can operate outdoors, helping cities optimize waste collection and reduce emissions. # 11. References 1. **STMicroelectronics VL53L0X Datasheet** 2. **U8g2 SH1106 OLED driver** – https://github.com/olikraus/u8g2 3. **TinyGPSPlus Library** – https://github.com/mikalhart/TinyGPSPlus 4. **UniversalTelegramBot Library** – https://github.com/witnessmenow/Universal-Arduino-Telegram-Bot 5. **Node-RED Documentation** – https://nodered.org/docs/*