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. 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 I^2C 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:

Adafruit_VL53L0X – Time-of-Flight sensor

U8g2lib – SH1106 OLED driver

WiFi.h / HTTPClient.h – Wi-Fi & HTTP POST

TinyGPSPlus.h – GPS parsing

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 Power rails: +5 V from Li-Po-Solar charger to 5 V rail; 3.3 V regulator feeding 3.3 V rail.

ESP32: VIN ← 5 V, GND ← GND, SDA ← GPIO 8, SCL ← GPIO 9.

VL53L0X: VCC ← 3.3 V, GND ← GND, SDA/SCL as above.

OLED (SH1106): VCC ← 3.3 V, GND ← GND, SDA/SCL as above.

GPS module:

VCC ← 5 V

GND ← GND

TX ← ESP32 RX (GPIO 16?)

RX ← ESP32 TX (GPIO 17?) (**ℹ** Please verify which GPIOs you wired for GPS TX/RX.)

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

<figure> <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 Kopyala Düzenle #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
display.begin();
display.clearBuffer(); display.setFont(u8g2_font_ncenB08_tr);
display.drawStr(0,10,"Init Display"); display.sendBuffer();
// Sensor
if (!lox.begin()) { while(1); }
pinMode(LED_PIN, OUTPUT);
// GPS
```

```
gpsSerial.begin(9600, SERIAL_8N1, /*RX*/16, /*TX*/17);
```

```
// Wi-Fi
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) { delay(500); Serial.print('.'); }
Serial.println("\nWiFi OK");
// Telegram
secured_client.setInsecure();
```

```
}
```

void loop() {

```
// Read GPS
while (gpsSerial.available()) gps.encode(gpsSerial.read());
```

```
// Measure distance
VL53L0X_RangingMeasurementData_t m;
lox.rangingTest(&m, false);
if (m.RangeStatus != 4) {
  float distance_cm = m.RangeMilliMeter/10.0 + DISTANCE_OFFSET_CM;
  float fill_pct = constrain(100.0 - (distance_cm/BIN_HEIGHT_CM)*100.0, 0.0,
100.0);
  lastDistance = distance_cm; lastPercentage = fill_pct;
```

```
  // Update OLED
  display.clearBuffer();
  display.setCursor(0,12);
  display.print("Dist: "); display.print(distance_cm,1);
display.print("cm");
  display.setCursor(0,30);
  display.print("Fill: "); display.print((int)fill_pct); display.print("%");
  // Bar
  int w = map((int)fill_pct,0,100,0,120);
  display.drawFrame(0,45,120,10);
  display.drawBox(0,45,w,10);
  display.sendBuffer();
```

```
  // Alert if ≥80%
  if (fill_pct >= 80.0) {
    digitalWrite(LED_PIN, HIGH);
    if (millis() - lastAlertTime > 15000) {
      sendWarning(distance_cm, fill_pct);
      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);
```

}

void sendWarning(float d, float p) {

```
// HTTP to Node-RED
WiFiClient client; HTTPClient http;
String payload = "{\"distance\":"
  +String(d,1)+",\"fill_percentage\":"+String(p,1)
  +",\"lat\":"+String(gps.location.lat(),6)
  +",\"lng\":"+String(gps.location.lng(),6)+"}";
if (http.begin(client, alert_url)) {
  http.addHeader("Content-Type","application/json");
  http.POST(payload); http.end();
}
// Telegram
String msg = "⚠ *Bin almost full!*\n"
  "📏 "+String(d,1)+"cm\n"
  "📊 "+String(p,1)+"%\n"
  "📍 "+String(gps.location.lat(),6)+","
         +String(gps.location.lng(),6);
bot.sendMessage(chat_id, msg, "Markdown");
```

}

void sendRegularUpdate(float d, float p) {

```
// identical to sendWarning, but different message icon
// ...
```

}

void handleNewMessages(int n) {

```
for (int i=0;i<n;i++){
  String id = String(bot.messages[i].chat_id),
```

```
        txt = bot.messages[i].text,
        name= bot.messages[i].from_name;
  if (txt=="/status") {
    bot.sendMessage(id,
      "🗑 "+String(lastDistance,1)+"cm\n📊 "+String(lastPercentage,1)+"%",
"");
  } else if (txt=="/stop") {
    updatesEnabled=false; bot.sendMessage(id,"🔕 Updates
stopped.","Markdown");
  } else if (txt=="/start") {
    updatesEnabled=true; bot.sendMessage(id,"🔔 Updates
resumed.","Markdown");
  } else {
    bot.sendMessage(id,"❓ Unknown. Use /status,/stop,/start","Markdown");
  }
}
```

} (**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:

Parses JSON (distance, fill_percentage, lat, lng)

Conditions: if fill_percentage ≥ 80 → send email or SMS, else just log

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 Create bot with BotFather → get BOT_TOKEN.

Invite to your group/channel → note the chat_id.

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 VL53L0X Datasheet, STMicroelectronics.

SH1106 OLED driver, U8g2 library: https://github.com/olikraus/u8g2

TinyGPSPlus, Mikal Hart: https://github.com/mikalhart/TinyGPSPlus

UniversalTelegramBot, Brian Lough: https://github.com/witnessmenow/Universal-Arduino-Telegram-Bot

Node-RED Documentation, https://nodered.org/docs/