Introduction

Efficient resource management in agriculture and landscaping has become critically important due to mounting environmental pressures. Two of the most pressing issues are the unnecessary overuse of water for irrigation—leading to water scarcity and waste—and the excessive application of fertilizers and chemicals, which infiltrate the soil and contaminate groundwater.

This project addresses these challenges by leveraging real-time, sensor-driven monitoring to optimize irrigation precisely when and where it's needed. By integrating an ESP32 microcontroller with a VL53L8CX Time-of-Flight (ToF) sensor, the system can detect the presence and position of plants or objects in a monitored area. Coupled with instant wireless data transmission and automated control of watering systems, the setup enables the following environmental benefits:

- **Water Conservation:** Irrigation is triggered only when the sensor detects plant presence and proximity, reducing unnecessary watering and helping to preserve scarce water resources.
- **Targeted Fertilizer Application:** By knowing exactly where and when plants are present, the system can help guide precise application of fertilizers and reduce runoff—limiting the amount of chemicals infiltrating natural soil and groundwater.
- **Reduced Environmental Footprint:** Intelligent control systems such as this not only save resources but also help reduce the carbon footprint and ecological impacts associated with traditional, less-efficient agricultural practices.

This project demonstrates how low-cost, network-connected sensors and automation hardware can contribute to sustainable practices in agriculture, urban gardening, or landscape management. The following report details both the hardware and software necessary to build the system, so others can replicate and further adapt it to address environmental needs in their own communities.

Materials and Methods

Materials

- ESP32 Development Board: Primary controller running FreeRTOS.
- VL53L8CX ToF Sensor Module: Delivers 8×8 grid distance measurements for object/plant detection.
- Push-Button Switch: User input, event annotation.
- LED, relay, or actuator (connected to GPIO7): Controls irrigation.
- Wiring/Breadboard or PCB: For sensor, switch, and actuator connections.
- Client computer/device: Receives sensor data via TCP.
- Power Supply: For ESP32 and peripherals.
- Wi-Fi Network: For ESP32 to connect and transmit data.

Pin Assignments

Function	ESP32 GPIO	Notes
I2C SCL	9	ToF sensor
I2C SDA	8	ToF sensor
ToF sensor reset	5	XSHUT line
Output (Actuator)	7	Controls valve/LED/relay
Input (positioning marks reader)	4	With internal pull-up enabled

Methods

The proposed system integrates an ESP32 microcontroller, a VL53L8CX ToF sensor, actuator control, and Wi-Fi-based TCP communication in order to enable intelligent, sustainable irrigation. Below, the implementation approach is detailed in a narrative format, with illustrative code excerpts highlighting key software components.

System Configuration and Setup

- Wi-Fi Networking

The ESP32 is configured to join an existing Wi-Fi network as a station. Wi-Fi credentials are embedded in the code, allowing for easy adjustment depending on deployment site:

```
#define EXAMPLE_ESP_WIFI_SSID "iotlab"
#define EXAMPLE_ESP_WIFI_PASS "iotlab18"
```

Connection status is monitored and maintained using ESP-IDF’s event loop and FreeRTOS event groups. This ensures reliable operation even if the access point is temporarily unavailable:

```
s_wifi_event_group = xEventGroupCreate();
ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_config));
ESP_ERROR_CHECK(esp_wifi_start());
```

- TCP Server for Data Streaming

Once connected to Wi-Fi, the ESP32 runs a TCP server on port 5055. This server streams processed sensor data and event annotations to any client on the local network. The TCP task listens for and accepts new client connections, and handles connection loss gracefully:

```
server_sock = socket(AF_INET, SOCK_STREAM, IPPROTO_IP);
bind(server_sock, (struct sockaddr *)&server_addr, sizeof(server_addr));
listen(server_sock, 1);
// Accept and serve clients
```

- I2C and Sensor Initialization

The VL53L8CX sensor communicates over I²C, using dedicated pins:

```
#define I2C_SCL GPIO_NUM_9
#define I2C_SDA GPIO_NUM_8
```

The code first brings the sensor out of reset via GPIO5 (XSHUT), then initializes it for 8×8 ranging at 10 Hz, ensuring it's ready for environmental monitoring:

```
Dev.platform.reset_gpio = GPIO_NUM_5;
VL53L8CX_Reset_Sensor(&(Dev.platform));
ret = vl53l8cx_init(&Dev);
ret = vl53l8cx_set_resolution(&Dev, VL53L8CX_RESOLUTION_8X8);
ret = vl53l8cx_set_ranging_frequency_hz(&Dev,10);
```

- Data Acquisition and Plant/Object Detection

Periodically (every 100 ms), the ESP32 queries the VL53L8CX for a new distance frame. Object or plant detection is performed by comparing each value to the median of the frame, considering any pixel “close” if it is significantly less than the median (i.e., background distance):

```
// Compute median as background
int bg = median(distance);
// Identify pixels indicating presence (e.g., plant detected)
if (distance[i] < bg - offset) { object_mask[i] = true; }
```

A centroid is computed for detected zones, estimating the plant's position beneath the sensor.

- Actuator (Irrigation) Control

If the detected object is centered (i.e., the plant is beneath the sensor), the output GPIO (GPIO7) is asserted to trigger irrigation; otherwise it remains low, ensuring only occupied zones are watered:

```
if (/* central pixels detect presence */) {
    gpio_set_level(GPIO_NUM_7, 1); // Open valve/activate relay
} else {
    gpio_set_level(GPIO_NUM_7, 0); // Close valve
}
```

- Button Handling and Event Annotation

A push-button (GPIO4) enables manual event annotation. Button interrupts are debounced using a hardware timer for reliability:

```
gpio_isr_handler_add(BUTTON_PIN, interr_handler, (void*)BUTTON_PIN);
// In ISR task:
if (current_state == 0 && last_state == 1 && (timestamp - last_change) >=
DEBOUNCE_uS) {
    // Send a "mark" frame to client
}
```

- Data Transmission and Logging

Each sensor frame (including timestamps and any event marks) is immediately transmitted to any connected client via TCP. The raw data (typically a timestamp and 64 distance readings) can be

received, visualized, and logged using a Python client.

Example packet preparation:

```
// Prepare buffer for [timestamp][frame]
uint8_t sendbuf[8 + FRAME_SIZE * 2];
memcpy(sendbuf, &timestamp, 8);
memcpy(sendbuf + 8, frame, FRAME_SIZE * 2);
// Send to client
send(client_sock, sendbuf, sizeof(sendbuf), 0);
```

Software Flow

Written in C using ESP-IDF framework.

- Uses FreeRTOS for multitasking: independent tasks for TCP communication, sensor polling, and button handling.
- Hardware timer (GPTimer) ensures accurate event timestamps and debouncing.
- All configuration parameters (SSID, pins, thresholds) are user-adjustable.

Assembly

- Wire the ESP32 to the VL53L8CX using I2C (SCL: 9, SDA: 8) and sensor XSHUT to GPIO5.
- Connect button to GPIO4 (with internal or external pull-up to 3.3V).
- Connect actuator (e.g., relay valve) control input to GPIO7.
- Flash the ESP32 with the provided code, making any adjustments for your setup.
- Start ESP32; ensure it connects to Wi-Fi.
- Connect a client to ESP32's IP on TCP port 5055 to view or log streaming data.

Results

Functional Testing

When the ESP32 is powered and connected to Wi-Fi:

- The VL53L8CX sensor continuously scans its field, providing an 8×8 distance map.
- The ESP32 detects objects (e.g., plant) based on pixels closer than the background by a given threshold.
- When an object is close to the central region of the sensor frame (representing a plant directly under the sensor), GPIO7 is activated—demonstrating selective and efficient irrigation.

- Actuator remains off when no plant is detected or it is not near the center, preventing watering of bare soil and reducing excessive chemical/fertilizer application.
- All sensor frames and button events are timestamped and streamed live to TCP clients for monitoring or data analysis.

Environmental Impact

- **Water Use Reduction:** System only waters when plant presence is confirmed and in the precise location, minimizing waste.
- **Reduced Chemical Runoff:** As irrigation is limited to when and where needed, less fertilizer is washed into groundwater.
- **Data Gathering:** Collected distance/time data supports further optimization, trend analysis, and integration with weather/fertilization schedules.

Reliability

- Button interrupts reliably send annotated “mark” frames for event logging or manual input.
- The system is resilient to network disconnects, with reconnection and data buffering as programmed.

Discussion

- This system showcases the potential of integrating low-cost sensor networks and automation for sustainable environmental stewardship:
- Precision Irrigation: Only waters when plant is actually present, avoiding traditional timer-based schemes that can waste water and leach chemicals.
- Scalability: Multiple ESP32/sensor nodes can be deployed across large fields or greenhouses, each acting independently but monitored from a central server.
- Customization: Sensor thresholds, actuator logic, and even fertilization scheduling can be tailored using the streamed data, allowing fine-grained environmental control.

Limitations & Improvements:

- Current system is distance-based; integrating soil moisture or plant health sensors could further refine watering decisions.
- Wireless reliability is dependent on network strength; alternative protocols (e.g., LoRa) could be used in rural deployments.
- Data encryption/authentication could be added for more secure remote management.

In summary, this project presents a practical, adaptable example of how sensor-driven automation can help address water and chemical conservation challenges in environmental and agricultural settings. The design and methods are fully replicable, providing a baseline for further innovation and

environmental impact.

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

Permanent link:
https://student-wiki.eolab.de/doku.php?id=amc:ss2025:group-a:start&rev=1753774511

Last update: **2025/07/29 09:35**