

BAB IV

PROSES PEMBUATAN ALAT

4.1 Pembuatan Perangkat Keras

Pembuatan perangkat keras merupakan tahap krusial dalam merealisasikan sistem autodosing nutrisi hidroponik agar dapat beroperasi sesuai dengan rencana perancangan. Sistem ini menggunakan mikrokontroler ESP32 sebagai pusat kendali utama yang bertugas mengakuisisi data lingkungan, memproses algoritma kontrol PID adaptif, serta mengatur aktuator untuk menjaga konsentrasi nutrisi secara presisi.

Komponen input utama yang digunakan dalam sistem ini meliputi sensor *Total Dissolved Solids* beserta modul pengondisi sinyalnya, yang berfungsi untuk membaca tingkat kepekatan partikel nutrisi dalam larutan. Selain itu, digunakan sensor suhu waterproof DS18B20 untuk memantau temperatur air secara akurat, guna memastikan kondisi pelarutan nutrisi tetap berada pada rentang yang optimal.

Sebagai komponen output dan aktuator utama, sistem ini menggunakan tiga buah pompa peristaltik DC. Pompa-pompa ini secara spesifik dialokasikan untuk menginjeksikan nutrisi A, nutrisi B, dan air baku. Kinerja ketiga pompa tersebut dikendalikan oleh ESP32 melalui modul driver motor DC L298N, yang memungkinkan pengaturan aliran cairan secara proporsional dan responsif sesuai dengan instruksi dari kontroler.

Untuk antarmuka visual alat digunakan LCD 20x4 dengan modul I2C yang menampilkan parameter esensial seperti nilai TDS, suhu larutan, dan status kerja pompa secara real-time. Di samping itu, ESP32 juga dikonfigurasi untuk terhubung ke jaringan Wi-Fi, memungkinkan sistem mengirimkan data secara kontinu ke platform *Internet of Things*. Hal ini memfasilitasi pengguna dalam melakukan monitoring jarak jauh serta pencatatan data performa sistem.

Pada sektor catu daya, sistem mendapat suplai dari adaptor DC utama. Daya tersebut kemudian diatur melewati modul *Buck Converter* yang berfungsi menurunkan dan menstabilkan tegangan agar sesuai dengan kebutuhan operasional ESP32, sensor, dan modul driver. Seluruh komponen ini dirangkai dan diintegrasikan secara cermat untuk memastikan keandalan sinyal kelistrikan dan memudahkan proses pengujian alat. Pada Tabel 4.1 akan ditampilkan daftar lengkap alat dan bahan yang digunakan dalam pembuatan perangkat keras sistem autodosing hidroponik ini.

4.1.1 Alat dan Bahan

Tahap pertama pembuatan sistem adalah mempersiapkan alat dan bahan. Daftar alat yang akan digunakan dapat dilihat pada Table 4.1 dan Table 4.2 berisi bahan yang digunakan dalam perancangan alat.

Table 4- 1 Bahan yang digunakan

No	Komponen/Bahan	Jumlah
1	ESP32	1 buah
2	Sensor TDS	1 buah
3	Sensor DS18B20	1 buah
4	LCD 20x4	1 buah
5	Buck Converter	1 buah
6	Driver module L298N	2 buah
7	Adaptor 12V	1 buah
8	Motor dc 12V	3 buah
9	connector	3 buah

Table 4- 2 Alat yang digunakan

No	Alat	Jumlah
1	obeng	1 set

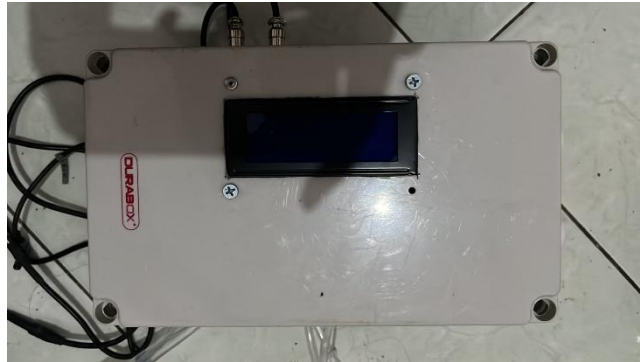
2	Solder	1 buah
3	Bor	1 set
4	Alat Tulis	1 buah
5	Gunting	1 buah
6	Cutter	1 buah
7	Multimeter	1 buah
8	TDS Meter	1 buah
9	gergaji	1 buah
10	Laptop	1 buah

4.1.2 Pembuatan Perangkat Elektronik

Dalam proses pembuatan perangkat elektronik, dilakukan tahapan perancangan dan perakitan seluruh komponen yang berfungsi sebagai pengendali sistem secara keseluruhan. Seluruh komponen tersebut disusun dalam satu sistem terintegrasi yang dikendalikan oleh mikrokontroller berbasis ESP32. Adapun tahapan dalam proses perangkat ini terdiri dari beberapa bagian penting.

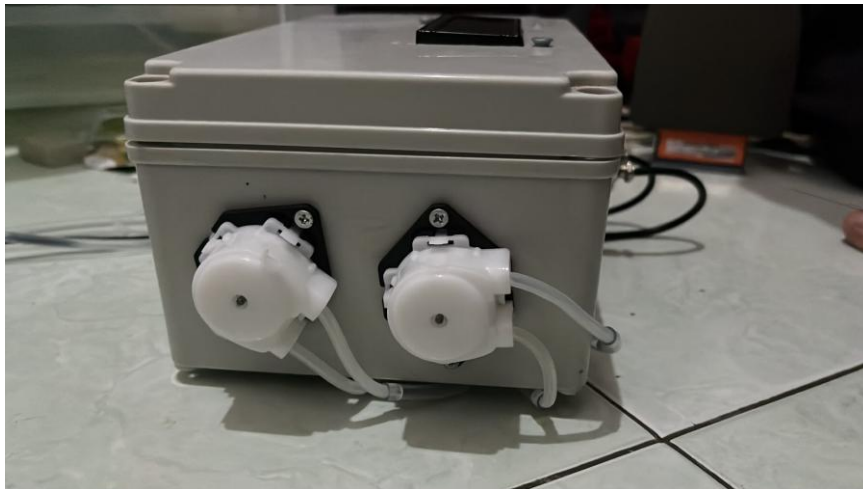
1. Modifikasi Kotak Sistem Kontrol

Langkah pertama dalam perakitan adalah melakukan penyesuaian pada kotak kontrol pelindung. Modifikasi dilakukan dengan melubangi permukaan kotak untuk menempatkan komponen antarmuka yang harus terhubung ke area luar sistem. Pada panel depan dibuat pemotongan persegi yang presisi untuk dudukan layar LCD 20x4 agar parameter sistem dapat dipantau langsung



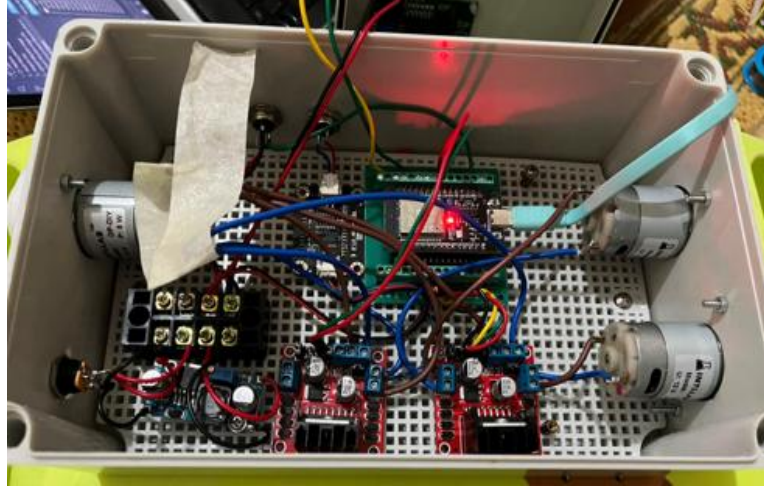
Gambar 4. 1 Pemasangan LCD

Pada panel bagian samping dan bawah dilubangi khusus untuk menempatkan konektor suplai daya utama dan socket konektor untuk jalur kabel masuk dari sensor TDS dan sensor suhu. Selain itu, dibuat tiga lubang sirkular untuk menempatkan bagian kepala pompa peristaltik DC agar selang injeksi cairan mudah diakses tanpa harus membuka kotak.



Gambar 4. 2 Pemasangan Pompa Peristaltik

2. Penempatan dan Wiring Komponen



Gambar 4. 3 Penempatan dan Wiring Komponen

Tahap akhir adalah instalasi pengkabelan yang dieksekusi secara sistematis merujuk pada diagram wiring yang telah dirancang sebelumnya. Tegangan dari konektor *supply* dialirkan menuju *Buck Converter* untuk disesuaikan tegangannya, lalu didistribusikan secara paralel melalui terminal blok ke ESP32, driver motor, dan sensor-sensor. Penyolderan dilakukan pada titik-titik sambungan permanen, seperti pada kaki terminal pompa dan konektor sensor. Kabel kontrol dari pin GPIO ESP32 dirutekan menuju driver motor untuk menjalankan aktuasi pompa berdasarkan keluaran algoritma kontrol PID adaptif yang memproses data sensor.

4.1.3 Pembuatan Perangkat Mekanik dan Integrasi Sistem Hidroponik NFT

Perancangan mekanik pada penelitian ini bertujuan untuk merealisasikan media tanam fisik berupa sistem hidroponik *Nutrient Film Technique* (NFT), sekaligus menjadi wadah integrasi bagi perangkat keras autodosing nutrisi yang telah dirakit. Desain mekanik ini harus memastikan aliran sirkulasi air berjalan lancar dan titik-titik injeksi nutrisi terdistribusi dengan baik sebelum dialirkan ke tanaman.

1. Pembuatan Rangka dan Pipa Instalasi

Struktur utama hidroponik NFT dirancang menggunakan pipa PVC yang dilubangi bagian atasnya sebagai tempat peletakan netpot tanaman. Jarak antar lubang disesuaikan dengan jenis tanaman yang akan dibudidayakan agar pertumbuhan daun tidak saling menutupi. Pipa-pipa PVC ini disangga oleh rangka yang dirancang dengan tingkat kemiringan sekitar 1% hingga 5%. Kemiringan ini sangat krusial dalam metode NFT untuk memastikan air nutrisi mengalir membentuk lapisan tipis secara konstan akibat gaya gravitasi, sehingga akar tanaman mendapatkan nutrisi dan oksigen yang maksimal tanpa tergenang.

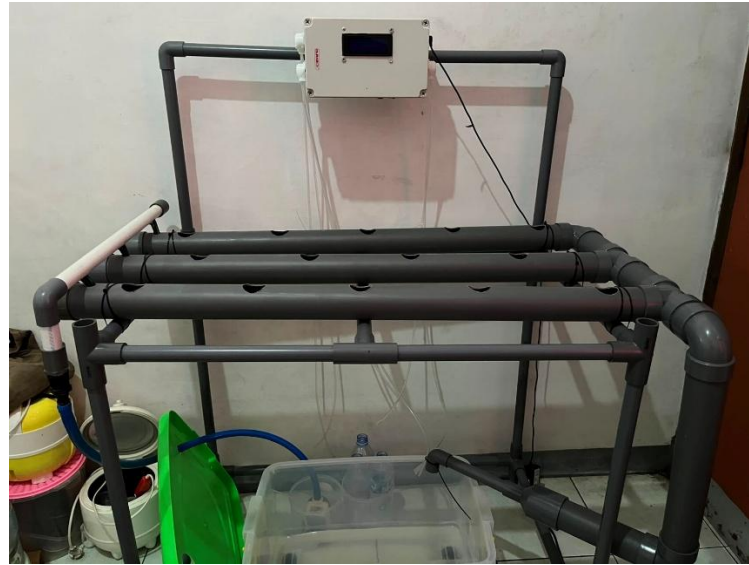
2. Sistem Sirkulasi dan Tandon Air

Di bagian ujung bawah aliran instalasi pipa NFT, dipasang pipa pembuangan/saluran balik yang mengarah langsung ke sebuah bak penampung atau tandon air. Tandon ini berfungsi sebagai tempat penyimpanan utama cairan larutan nutrisi. Sebuah pompa air celup diletakkan di dalam tandon untuk memompa kembali larutan nutrisi ke ujung atas instalasi pipa PVC, menciptakan sebuah siklus sirkulasi air yang tertutup (closed-loop system).

3. Integrasi Sistem Kontrol Autodosing

Bagian terpenting dari perancangan mekanik ini adalah penyatuan kotak sistem kontrol autosoding dengan instalasi hidroponik. Integrasi dilakukan dengan tata letak sebagai berikut:

- a. Kotak panel kontrol dipasang pada tiang rangka penyangga hidroponik pada posisi yang ergonomis. Posisi ini dipastikan aman dari percikan air maupun kebocoran instalasi, namun tetap cukup dekat dengan tandon air.
- b. Probe dari sensor TDS dan sensor suhu DS18B20 dimasukkan ke dalam air di dalam tandon utama. Posisinya diletakkan di dekat area masuknya aliran air balik dari pipa NFT, sehingga sensor dapat membaca kondisi aktual larutan yang baru saja melewati akar tanaman sebelum dikoreksi oleh sistem.
- c. Tiga buah selang silikon yang terhubung dengan output pompa peristaltik pada kotak kontrol ditarik dan diarahkan ke dalam tandon air. Selang-selang ini merupakan jalur injeksi untuk cairan Nutrisi A, Nutrisi B, dan Air Baku.



Gambar 4. 4 Tampilan Keseluruhan Alat

Dengan perancangan mekanik ini, ketika sistem kontrol mendeteksi penurunan nilai TDS, pompa peristaltik akan langsung meneteskan nutrisi baru ke dalam tandon. Aliran air dari pompa celup akan membantu mengaduk nutrisi yang baru diinjeksi tersebut secara alami sebelum disirkulasikan kembali ke seluruh pipa NFT. Hal ini menjamin tingkat kepekatan cairan nutrisi tetap stabil secara otomatis dan presisi.

4.2 Kalibrasi Sensor

4.2.1 Kalibrasi Sensor TDS

Kalibrasi sensor TDS (*Total Dissolved Solids*) bertujuan untuk memastikan tingkat keakuratan pembacaan kepekatan nutrisi larutan, sehingga data yang diproses oleh ESP32 memiliki tingkat presisi yang tinggi. Pada sistem ini, kalibrasi tidak dilakukan dengan menebak nilai ADC secara mentah, melainkan menggunakan metode komparasi. Metode ini dilakukan dengan membandingkan hasil pembacaan dari sensor TDS sistem dengan alat ukur standar yang sudah terkalibrasi pabrik, yakni TDS meter komersial portabel. Alat ukur komersial ini bertindak sebagai referensi utama untuk menentukan faktor kalibrasi yang akan dimasukkan ke dalam program.

Tahap kalibrasi dimulai dengan menyiapkan sampel larutan nutrisi hidroponik (campuran nutrisi AB Mix dan air) di dalam sebuah wadah pengujian. Larutan tersebut pertama-tama diukur secara manual menggunakan TDS meter komersial. Dari hasil pengukuran pada alat standar menunjukkan angka konsentrasi sebesar 408 ppm. Nilai ini dicatat sebagai nilai aktual (nilai sebenarnya) yang harus dicapai oleh sensor TDS yang terpasang pada alat sistem kontrol.



Gambar 4. 5 Pengambilan Sampel menggunakan TDS Meter

Langkah selanjutnya adalah mencelupkan probe sensor TDS yang telah terhubung dengan ESP32 ke dalam sampel larutan yang sama. Pada tahap ini, mikrokontroler dijalankan menggunakan program kalibrasi dasar (bawaan dari library sensor TDS) yang membaca data analog dan mengubahnya menjadi nilai ppm menggunakan faktor kalibrasi default. Hasil pembacaan dari sistem ditampilkan ke layar komputer melalui Serial Monitor. Karena sensor belum dikalibrasi secara spesifik untuk lingkungan tersebut, umumnya akan terdapat selisih pembacaan.

Untuk menghilangkan selisih tersebut, dilakukan penyesuaian pada variabel faktor kalibrasi (seringkali ditulis sebagai `calibrationFactor` atau variabel `k-value`)

yang terdapat di dalam struktur code program. Nilai kalibrasi yang baru didapatkan dengan membandingkan nilai ukur TDS meter komersial dengan nilai baca sensor. Jika nilai sensor lebih rendah dari TDS meter standar, maka nilai variabel kalibrasi di dalam program dinaikkan. Sebaliknya, jika nilai sensor lebih tinggi, maka nilai variabel kalibrasi diturunkan. Nilai yang baru ini kemudian diketik secara manual untuk menggantikan nilai default di dalam program kalibrasi sensor.

Setelah variabel kalibrasi diperbarui di dalam source code, program tersebut diunggah ulang (compile and upload) ke ESP32. Pengujian pembacaan kemudian diulang pada larutan yang sama untuk memverifikasi hasilnya. Serial Monitor kembali dipantau untuk memastikan bahwa output nilai ppm yang dihasilkan oleh sensor sistem kini telah sejajar atau memiliki toleransi galat (*error*) yang sangat minim terhadap angka 408 ppm dari TDS meter komersial.



Gambar 4. 6 Hasil Pembacaan Sensor Setelah Kalibrasi

Tahap kalibrasi komparatif ini merupakan langkah yang sangat esensial sebelum sistem dirakit secara permanen. Dengan memasukkan parameter kalibrasi yang telah teruji ke dalam program utama, sistem autodosing dapat mendeteksi konsentrasi nutrisi secara valid. Data pembacaan yang akurat ini mutlak dibutuhkan oleh mikrokontroler ESP32 sebagai landasan logika perhitungan, sehingga

keputusan untuk mengaktifkan pompa penyiraman nutrisi dapat dieksekusi secara presisi sesuai dengan target setpoint yang dibutuhkan oleh tanaman hidroponik.

4.2.2 Kalibrasi Sensor DS18B20

Kalibrasi sensor suhu DS18B20 dilakukan untuk memastikan bahwa hasil pengukuran temperatur larutan nutrisi yang dibaca oleh sensor sesuai dengan kondisi suhu sebenarnya. Proses kalibrasi ini dilakukan dengan cara membandingkan pembacaan sensor DS18B20 terhadap alat ukur standar komersial, yaitu memanfaatkan fitur pengukur suhu yang terintegrasi pada TDS meter digital portabel. Pengujian dilakukan dengan mencelupkan probe dari kedua alat tersebut ke dalam sampel wadah larutan air yang sama secara bersamaan.

Pada pengujian ini, sensor DS18B20 menampilkan suhu larutan sebesar 26,5°C, sedangkan fitur pengukur suhu pada alat ukur TDS meter menunjukkan angka 26,6°C pada kondisi sampel yang sama. Selisih pembacaan yang sangat kecil tersebut menunjukkan bahwa sensor DS18B20 memiliki tingkat akurasi yang sangat baik dan masih sesuai dengan spesifikasi rentang toleransi pabrik. Dengan demikian, sensor dapat digunakan untuk mendukung sistem kompensasi suhu pada pembacaan kepekatan nutrisi, serta menunjang kinerja sistem autodosing hidroponik berbasis ESP32 secara keseluruhan.

4.3 Pembuatan Perangkat Lunak

Perangkat lunak pada sistem ini dirancang untuk mengendalikan proses otomatisasi serta mengelola komunikasi antara mikrokontroler ESP32, pembacaan sensor, pengendalian aktuator, dan platform pemantauan *Internet of Things* (IoT). Pengembangan program dilakukan menggunakan bahasa pemrograman C++ dalam lingkungan pengembangan terintegrasi (IDE).

Pada tahap implementasinya, alur program diawali dengan proses inisialisasi pin Input/Output (I/O) dan konfigurasi koneksi Wi-Fi agar modul ESP32

dapat terhubung ke jaringan internet secara stabil. Setelah inisialisasi berhasil, perangkat lunak mengatur proses akuisisi data dari sensor Total Dissolved Solids (TDS) untuk memantau konsentrasi nutrisi pada sistem hidroponik secara aktual.

Data yang diperoleh dari sensor tersebut kemudian diproses sebagai nilai umpan balik (*feedback*) ke dalam algoritma kontrol Proportional-Integral-Derivative (PID) adaptif. Perangkat lunak menjalankan fungsi *auto-tuning* untuk menjadwalkan dan menyesuaikan parameter gain PID secara dinamis. Kalkulasi ini menghasilkan sinyal kendali presisi yang memerintahkan aktuator (pompa dosing) untuk menyalurkan larutan nutrisi hingga mencapai nilai setpoint yang telah ditetapkan.

Untuk keperluan pemantauan dan pencatatan, data dari sistem—seperti nilai pembacaan sensor, status aktuator, dan parameter kontrol—dikirimkan secara real-time ke antarmuka aplikasi Blynk. Secara bersamaan, perangkat lunak juga melakukan data logging secara berkala ke Google Spreadsheet. Pengiriman data ini memanfaatkan komunikasi HTTP dengan format bungkusan data JSON yang ditangkap dan dieksekusi oleh endpoint dari Google Apps Script.

Sebagai langkah keandalan sistem, perangkat lunak dilengkapi dengan logika penanganan gangguan (*error handling*). Program akan secara otomatis melakukan upaya rekoneksi (*auto-reconnect*) apabila terjadi kegagalan pada jaringan Wi-Fi atau putusnya komunikasi ke server IoT. Dengan demikian, sistem ini memastikan otomasi penyesuaian dosis nutrisi hidroponik berjalan secara mandiri dan datanya terekam secara berkesinambungan.

Penyusunan program pada proyek ini dilakukan menggunakan lingkungan pengembangan Visual Studio Code (VS Code) dengan ekstensi PlatformIO, yang berfungsi sebagai platform pemrograman utama untuk mikrokontroler ESP32. Program yang dikembangkan memiliki beberapa fungsi utama, antara lain melakukan pembacaan data dari sensor Total Dissolved Solids (TDS) dan sensor suhu, menjalankan perhitungan algoritma kontrol Proportional-Integral-Derivative (PID) adaptif, serta memberikan instruksi kendali pada aktuator pompa dosing

nutrisi. Selain itu, perangkat lunak juga dirancang untuk mengaktifkan fitur web server tertanam pada modul ESP32.

Data yang diperoleh dari sensor-sensor dan parameter sistem tersebut tidak hanya dikirimkan ke platform pemantauan berbasis cloud—yaitu aplikasi Blynk dan Google Spreadsheet—tetapi juga dapat diakses secara langsung melalui antarmuka web lokal. Hal ini bertujuan agar kondisi nutrisi hidroponik dapat dipantau dan dikonfigurasi secara real-time melalui jaringan Wi-Fi lokal (Intranet), sekaligus tetap melakukan perekaman riwayat data (data logging) secara otomatis ke platform cloud menggunakan format bungkus data JSON melalui endpoint Google Apps Script.

Berikut ini pembagian dari modul program yang dikembangkan pada sistem meliputi:

- a. Program untuk koneksi jaringan dan IoT
- b. Program untuk Web Server Lokal
- c. Program untuk pembacaan Modul Sensor
- d. Program untuk algoritma Kontrol PID Adaptif
- e. Program untuk koneksi ke Rangkaian Aktuator
- f. Program untuk koneksi ke Tampilan LCD
- g. Program penanganan gangguan (*Failsafe*)

4.3.1 Program untuk koneksi jaringan dan IoT

Sub-bab ini membahas tahapan pembuatan program yang bertujuan untuk menghubungkan mikrokontroler ESP32 dengan jaringan Wi-Fi, Web Server lokal, aplikasi Blynk, serta platform pemantauan database Google Spreadsheet. Konektivitas ini menjadi tulang punggung sistem karena memungkinkan parameter kontrol nutrisi (seperti nilai sensor, status pompa, dan nilai PID) dipantau secara lokal melalui Intranet maupun dikirim dan direkam secara real-time melalui jaringan internet.

```
#include <Arduino.h>
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <WebServer.h>
#include <ESPmDNS.h>
#include <Preferences.h>
#include <HTTPClient.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <BlynkSimpleEsp32.h>
#include <time.h>
```

Gambar 4. 7 Library Utama

Pada Gambar 4.7 memperlihatkan pendefinisian pustaka (library) utama yang digunakan dalam pengembangan konektivitas sistem ini. Pustaka WiFi.h dan WiFiClientSecure.h digunakan untuk mengaktifkan antarmuka jaringan nirkabel dan koneksi aman pada ESP32. Untuk mengakomodasi pemantauan lokal dan pengaturan konfigurasi, sistem menggunakan pustaka WebServer.h dan ESPmDNS.h guna membangun halaman web interaktif (Local Web Server) yang dapat diakses melalui alamat http://esp32.local. Selain itu, pustaka Preferences.h dimanfaatkan untuk menyimpan kredensial jaringan ke dalam memori permanen perangkat (Non-Volatile Storage). Pustaka BlynkSimpleEsp32.h digunakan untuk komunikasi real-time ke dashboard IoT, sementara HTTPClient.h digunakan untuk melakukan transmisi data sensor ke endpoint Google Apps Script.

```
// ----- KONFIGURASI (disimpan di NVS / Preferences) -----
// autoTuneGains() berdasarkan besar & laju perubahan error) atau manual
// (Kp/Ki/Kd memakai nilai isian user apa adanya). config.kp/ki/kd selalu
// berisi nilai yang SEDANG dipakai PID -> updatePumps() yang mengisinya
// dari salah satu sumber sesuai config.autoTuneMode.
struct Config {
    String ssid;
    String password;
    double setpointTDS; // ppm (satu-satunya target yang diisi user)
    double kp, ki, kd; // nilai yang AKTIF dipakai PID saat ini (hasil auto-tuning ATAU manual)
    double deadband; // ppm, toleransi di sekitar setpoint
    bool autoTuneMode; // true = auto-tuning, false = manual (pakai manualKp/Ki/Kd)
    double manualKp, manualKi, manualKd; // nilai Kp/Ki/Kd isian user untuk mode manual
    String blynkToken;
    String sheetUrl; // URL Google Apps Script Web App (exec)
} config;

void loadConfig() {
    prefs.begin("hidroponik", true);
    // Default WiFi awal (dipakai saat NVS masih kosong / pertama kali flash) ->
    // dari WIFI_SSID/WIFI_PASSWORD yang di-hardcode di atas. Setelah ini bisa
    // diubah kapan saja lewat halaman konfigurasi mDNS (http://esp32.local);
    // kalau NVS pernah kesimpan kosong, tetap jatuh balik ke nilai hardcode
    // (sama seperti pola BLYNK_AUTH_TOKEN di bawah).
    config.ssid = prefs.getString("ssid", "");
    if (config.ssid.length() == 0) config.ssid = WIFI_SSID;
    config.password = prefs.getString("pass", WIFI_PASSWORD);
    config.setpointTDS = prefs.getDouble("sp", 1200.0);
    config.deadband = prefs.getDouble("db", 30.0);
}
```

Gambar 4. 8 Konfigurasi Jaringan dan Otoritas IoT

Gambar 4.8 menunjukkan struktur data dan fungsi `loadConfig()` yang digunakan untuk mengelola konfigurasi jaringan dan otorisasi IoT. Berbeda dengan pendekatan konvensional yang menyimpan data secara tetap di dalam kode program, sistem ini memanfaatkan `struct Config` untuk menyimpan variabel `ssid`, `password`, `blynkToken`, dan `sheetUrl` secara dinamis. Melalui pustaka `Preferences`, mikrokontroler dapat membaca kredensial terakhir yang disimpan di dalam memori saat perangkat baru dinyalakan. Jika memori masih kosong, sistem akan menggunakan nilai default. Pendekatan ini memungkinkan pengguna untuk mengubah jaringan Wi-Fi, mengganti token Blynk, atau memperbarui URL Google Spreadsheet melalui antarmuka web lokal tanpa perlu memprogram ulang (*re-flash*) modul ESP32.

```
// ----- KONEKSI WIFI + FALLBACK AP + mDNS -----
void connectWiFi() {
  if (config.ssid.length() > 0) {
    Serial.printf("[WiFi] Menghubungkan ke SSID '%s'...\n", config.ssid.c_str());
    WiFi.mode(WIFI_STA);
    WiFi.begin(config.ssid.c_str(), config.password.c_str());
    unsigned long start = millis();
    while (WiFi.status() != WL_CONNECTED && millis() - start < 15000) {
      delay(300);
      Serial.print(".");
    }
    Serial.println();
  } else {
    Serial.println("[WiFi] SSID kosong di konfigurasi -> langsung ke mode AP");
  }

  if (WiFi.status() != WL_CONNECTED) {
    Serial.println("[WiFi] Gagal konek -> masuk mode Access Point konfigurasi");
    WiFi.mode(WIFI_AP);
    WiFi.softAP(AP_SSID, AP_PASSWORD);
    Serial.printf("[WiFi] AP aktif: SSID='%s' IP=%s\n", AP_SSID, WiFi.softAPIP().toString().c_str());
    Serial.printf("[WiFi] Hubungkan HP/laptop ke WiFi '%s' lalu buka 192.168.4.1\n", AP_SSID);
  } else {
    Serial.printf("[WiFi] Terhubung! IP=%s RSSI=%dBm\n", WiFi.localIP().toString().c_str(), WiFi.RSSI());
    configTime(GMT_OFFSET_SEC, DST_OFFSET_SEC, NTP_SERVER);
    Serial.printf("[NTP] Sinkronisasi waktu dimulai (server: %s)\n", NTP_SERVER);
  }
}
```

Gambar 4. 9 Inisiasi Konektivitas

Pada Gambar 4.9 menampilkan inisialisasi konektivitas pada fungsi `connectWiFi()`. Proses diawali dengan memanggil fungsi `WiFi.begin()` menggunakan SSID dan password yang telah dimuat dari memori. Program akan melakukan perulangan pengecekan status selama maksimal 15 detik. Apabila koneksi gagal terjalin—misalnya karena jaringan tidak tersedia—sistem dirancang dengan fitur keselamatan (failsafe) di mana modul ESP32 akan beralih secara otomatis menjadi Access Point (AP) mandiri dengan nama "Hidroponik-Setup". Hal ini memastikan bahwa pengguna tetap dapat masuk ke halaman konfigurasi untuk memperbaiki pengaturan jaringan. Jika koneksi berhasil, sistem akan melakukan sinkronisasi waktu (Network Time Protocol/NTP) dan mengaktifkan layanan mDNS.

```
// ----- KIRIM KE GOOGLE SPREADSHEET -----
void sendToSpreadsheet() {
    if (config.sheetUrl.length() == 0) {
        Serial.println("[Sheet] Lewati kirim -> URL belum diisi");
        return;
    }
    if (WiFi.status() != WL_CONNECTED) {
        Serial.println("[Sheet] Lewati kirim -> WiFi tidak terhubung");
        return;
    }

    Serial.println("[Sheet] Mengirim data ke Google Spreadsheet...");
    WiFiClientSecure client;
    client.setInsecure(); // lewati verifikasi sertifikat demi kesederhanaan (aman untuk skala hobi)

    HTTPClient http;
    String url = config.sheetUrl
        + "?tds=" + String(currentTDS, 1)
        + "&suhu=" + String(currentTemp, 1)
        + "&setpoint=" + String(config.setpointTDS, 1)
        + "&pumpA=" + String(pumpAOn)
        + "&pumpB=" + String(pumpBOn)
        + "&pumpC=" + String(pumpCOn)
        + "&speedPct=" + String((int)((pumpSpeed / 255.0) * 100.0 + 0.5))
        + "&kp=" + String(config.kp, 3)
        + "&ki=" + String(config.ki, 3)
        + "&kd=" + String(config.kd, 3)
        + "&mode=" + String(config.autoTuneMode ? "auto" : "manual");
}
```

Gambar 4. 10 Setup Google Spreadsheet

Gambar 4.10 menunjukkan rutinitas `sendToSpreadsheet()` untuk pengiriman data historis sensor dan parameter PID ke Google Spreadsheet secara periodik setiap 10 detik. Proses ini memanfaatkan protokol HTTP Client aman (`WiFiClientSecure`). Pada blok kode ini, nilai parameter—seperti TDS aktual, suhu air, nilai adaptif gain (K_p , K_i , K_d), serta status pompa—dirangkai menjadi sebuah URL query string yang disisipkan ke dalam variabel `sheetUrl`.

URL yang telah memuat parameter tersebut kemudian dikirimkan menggunakan metode `http.GET()` menuju endpoint Google Apps Script. Apabila transmisi berhasil, server Google akan merespons dengan kode HTTP 200, dan script pada cloud akan mengekstrak data tersebut untuk dituliskan ke dalam baris dan kolom Spreadsheet secara otomatis. Status keberhasilan pengiriman ini (`lastSheetHttpCode`) kemudian dicatat untuk ditampilkan pada layar LCD dan Web Server lokal. Mekanisme ini menjamin bahwa seluruh riwayat dinamika kontrol nutrisi hidroponik terdokumentasi dengan baik tanpa membebani memori ESP32 dengan proses parsing JSON yang rumit.

4.3.2 Program untuk Web Server Lokal

Sub-bab ini membahas implementasi antarmuka pengguna berbasis web (Local Web Server) yang ditanamkan langsung ke dalam mikrokontroler ESP32. Fitur ini dirancang agar pengguna dapat memantau status nutrisi, melihat riwayat pergerakan kontrol PID, serta mengubah konfigurasi krusial (seperti batas setpoint TDS, SSID jaringan, dan parameter IoT) secara mandiri tanpa bergantung pada layanan pihak ketiga di internet. Akses ke halaman web ini dipermudah menggunakan protokol mDNS, sehingga pengguna cukup mengetikkan http://esp32.local pada browser perangkat yang terhubung di jaringan Wi-Fi yang sama.

```
void handleStatus() {
    String json = "{";
    json += "\"tds\": " + String(currentTDS, 1) + ",";
    json += "\"suhu\": " + String(currentTemp, 1) + ",";
    json += "\"setpoint\": " + String(config.setpointTDS, 1) + ",";
    json += "\"pumpA\": " + String(pumpAOn) + ",";
    json += "\"pumpB\": " + String(pumpBOn) + ",";
    json += "\"pumpC\": " + String(pumpCOn) + ",";
    json += "\"pumpSpeedPct\": " + String((int)((pumpSpeed / 255.0) * 100.0 + 0.5)) + ",";
    json += "\"kp\": " + String(config.kp, 3) + ",";
    json += "\"ki\": " + String(config.ki, 3) + ",";
    json += "\"kd\": " + String(config.kd, 3) + ",";
    json += "\"autoTuneMode\": " + String(config.autoTuneMode) + ",";
    json += "\"history\":[";
    for (int i = 0; i < historyCount; i++) {
        int idx = (historyHead - 1 - i + HISTORY_SIZE) % HISTORY_SIZE; // terbaru dulu
        SensorLog &h = history[idx];
        unsigned long secAgo = (millis() - h.t) / 1000;
        if (i > 0) json += ",";
        json += "{\"t\": " + String(secAgo) +
            ", \"tds\": " + String(h.tds, 1) +
            ", \"suhu\": " + String(h.temp, 1) +
            ", \"pumpA\": " + String(h.pumpA) +
            ", \"pumpB\": " + String(h.pumpB) +
            ", \"pumpC\": " + String(h.pumpC) +
            ", \"speedPct\": " + String((int)((h.speed / 255.0) * 100.0 + 0.5)) + "}";
    }
    json += "]";
}
```

Gambar 4. 11 Potongan Kode Program untuk Pembentukan Data JSON

Gambar 4.11 menunjukkan blok fungsi `handleStatus()` yang bertugas sebagai endpoint Application Programming Interface (API) internal bagi web server ESP32. Fungsi ini bertugas untuk menyusun (serialize) variabel kontrol aktual—seperti nilai TDS, suhu air, status ketiga pompa, hingga parameter gain PID hasil kalkulasi otomatis—menjadi sebuah struktur data berformat JSON. Data

JSON ini juga merangkum 10 riwayat pembacaan sensor terakhir yang tersimpan di dalam struktur memori internal (history). Penggunaan format JSON memungkinkan halaman HTML untuk meminta pembaruan data secara dinamis (Asynchronous JavaScript and XML / AJAX) tanpa perlu memuat ulang (reload) keseluruhan halaman web.

```
String htmlPage() {
  String html = "<!DOCTYPE html><html><head><meta charset='utf-8'>"
    "<meta name='viewport' content='width=device-width, initial-scale=1'>"
    "<title>Konfigurasi Hidroponik</title>"
    "<style>"
    "  *{box-sizing:border-box}"
    "  body{font-family:-apple-system, Segoe UI, Roboto, Arial, sans-serif; background:#f2f5f4;"
    "    margin:0;padding:24px 16px;color:#1f2933}"
    "  .wrap{max-width:640px;margin:0 auto}"
    "  h2{margin:0 0 16px}"
    "  h3{margin:0 0 10px;font-size:15px;color:#2f855a;text-transform:uppercase;letter-spacing:.04em}"
    "  .card{background:#fff;border-radius:10px;padding:18px 20px;margin-bottom:16px;"
```

Gambar 4. 12 Struktur HTML Halaman Konfigurasi dan Pemantauan Sistem

```
String statusScript() {
  return R"JS(
<script>
function badge(on){ return "<span class='badge ' + (on?'on':'off') + "</span>" + (on?'ON':'OFF') + "</span>"; }
function toggleManualFields(){
  const manual = document.querySelector('input[name="mode"][value="manual"]').checked;
  document.getElementById('manualFields').classList.toggle('disabled', !manual);
}
async function refreshStatus(){
  try {
    const r = await fetch('/status');
    const d = await r.json();
    document.getElementById('tdsVal').textContent = d.tds.toFixed(1);
    document.getElementById('suhuVal').textContent = d.suhu.toFixed(1);
    document.getElementById('speedVal').textContent = d.pumpSpeedPct + '%';
    document.getElementById('pumpBadges').innerHTML =
      'Pompa A ' + badge(d.pumpA) + ' &nbsp; Pompa B ' + badge(d.pumpB) + ' &nbsp; Air Baku ' + badge(d.pumpC);
    document.getElementById('tuneVal').innerHTML =
      (d.autoTuneMode ? 'Auto-tuned' : 'Manual') + ' &nbsp; Kp:' + d.kp.toFixed(3) + ' Ki:' + d.ki.toFixed(3) + ' Kd:' + d.kd.toFixed(3);
    let rows = '';
    if (d.history.length === 0) {
      rows = "<tr><td colspan='7'>Belum ada data</td></tr>";
    } else {
      for (const h of d.history) {
        rows += "<tr><td>" + h.t + " dtk lalu</td><td>" + h.tds.toFixed(1) + "</td><td>" + h.suhu.toFixed(1) +
          "</td><td>" + badge(h.pumpA) + "</td><td>" + badge(h.pumpB) + "</td><td>" + badge(h.pumpC) +
          "</td><td>" + h.speedPct + "%</td></tr>";
      }
    }
  }
}
```

Gambar 4. 13 Kode Program untuk Pembaruan Data Real-Time

Pada Gambar 4.13 memperlihatkan penyusunan struktur HTML dan script JavaScript yang ditanamkan ke dalam bentuk variabel String di C++. Antarmuka web ini tidak menggunakan file eksternal dari MicroSD atau SPIFFS, melainkan langsung dirender dari memori Flash ESP32 untuk meminimalkan latensi dan potensi error pada sistem berkas (file system). Terdapat logika JavaScript

setInterval(refreshStatus, 2000) yang akan secara aktif melakukan fungsi pengambilan (fetch) ke rute /status setiap 2 detik sekali. Hal ini menciptakan tampilan dashboard lokal yang terasa real-time dan interaktif.

```
void handleSave() {
  Serial.printf("[Web] POST /save dari %s -> menyimpan konfigurasi baru\n",
    server.client().remoteIP().toString().c_str());
  if (server.hasArg("ssid")) config.ssid = server.arg("ssid");
  if (server.hasArg("pass")) config.password = server.arg("pass");
  if (server.hasArg("sp")) config.setpointTDS = server.arg("sp").toDouble();
  if (server.hasArg("db")) config.deadband = server.arg("db").toDouble();
  if (server.hasArg("mode")) config.autoTuneMode = (server.arg("mode") == "auto");
  if (server.hasArg("mkp")) config.manualKp = server.arg("mkp").toDouble();
  if (server.hasArg("mki")) config.manualKi = server.arg("mki").toDouble();
  if (server.hasArg("mkd")) config.manualKd = server.arg("mkd").toDouble();
  if (server.hasArg("token")) config.blynkToken = server.arg("token");
  if (server.hasArg("sheet")) config.sheetUrl = server.arg("sheet");

  saveConfig();

  server.send(200, "text/html",
    "<html><body><h3>Konfigurasi disimpan. Perangkat akan restart...</h3></body></html>");
  Serial.println("[System] Restart dalam 1.5 detik untuk menerapkan konfigurasi baru...");
  delay(1500);
  ESP.restart();
}
```

Gambar 4. 14 Potongan Kode untuk Menyimpan Konfigurasi Sistem

Gambar 4.14 menunjukkan fungsi handleSave() yang bertugas untuk memproses data masukan dari pengguna ketika tombol "Simpan & Restart" pada antarmuka web ditekan. Fungsi ini akan menangkap parameter HTTP POST (form submission), mengekstrak nilai-nilainya ke dalam variabel config yang aktif, lalu memanggil fungsi saveConfig() untuk menyimpan preferensi baru tersebut ke dalam ruang penyimpanan permanen (Non-Volatile Storage / NVS). Setelah penyimpanan berhasil, program dikonfigurasi untuk memberikan jeda sementara (delay(1500)) dan kemudian memanggil perintah ESP.restart() agar mikrokontroler segera menjalankan siklus booting ulang dengan parameter pengaturan yang telah diperbarui.

4.3.3 Program untuk Pembacaan Modul Sensor

Sub-bab ini menjelaskan rancangan perangkat lunak yang bertugas melakukan akuisisi data dari lingkungan fisik, tepatnya dari sensor *Total Dissolved*

Solids (TDS) untuk mengukur kepekatan nutrisi, dan sensor DS18B20 untuk mengukur suhu air. Kualitas pembacaan dari kedua sensor ini sangat krusial karena data yang dihasilkan menjadi nilai umpan balik (*feedback*) utama bagi algoritma kontrol PID adaptif dalam menentukan kecepatan pompa dosing.

```
// ----- BACA SENSOR TDS (DFRobot) + DS18B20 -----
#define TDS_SAMPLE_COUNT 15 // jumlah sampel ADC untuk median filter (ganjil)

// insertion sort sederhana, cukup cepat untuk array sekecil TDS_SAMPLE_COUNT
void sortSamples(int16_t arr[], int n) {
    for (int i = 1; i < n; i++) {
        int16_t key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}
```

Gambar 4. 15 Potongan Kode untuk Median Filter Data Sensor

Gambar 4.15 memperlihatkan implementasi teknik penyaringan sinyal digital (digital filtering) berupa median filter sederhana. Probe sensor TDS konduktometri sering kali menghasilkan sinyal analog yang bergejolak (noisy) akibat riak air atau efek polarisasi. Untuk mengatasi hal ini, program didefinisikan untuk mengambil 15 sampel data (variabel TDS_SAMPLE_COUNT) dalam satu siklus pembacaan. Array sampel tersebut kemudian diurutkan dari nilai terkecil hingga terbesar menggunakan algoritma insertion sort pada fungsi sortSamples(). Dengan mengambil nilai tengah (median) dari array yang telah diurutkan, sistem dapat mengabaikan pembacaan ekstrem yang terdeteksi (data pencilan / outliers), sehingga nilai dasar yang digunakan untuk perhitungan selanjutnya menjadi jauh lebih stabil.

```

void readSensors() {
    dsSensors.requestTemperatures();
    float t = dsSensors.getTempCByIndex(0);
    if (t != DEVICE_DISCONNECTED_C && t > -50 && t < 100) {
        currentTemp = t;
    }

    // Median filtering ala DFRobot: probe TDS berisik, beberapa sampel
    // diambil lalu diambil nilai tengahnya supaya pembacaan lebih stabil.
    // analogReadMilliVolts() pakai kurva kalibrasi pabrik ESP32 -> lebih akurat
    // daripada konversi manual raw ADC ke tegangan.
    int16_t samples[TDS_SAMPLE_COUNT];
    for (int i = 0; i < TDS_SAMPLE_COUNT; i++) {
        samples[i] = (int16_t)analogReadMilliVolts(TDS_PIN);
    }
    sortSamples(samples, TDS_SAMPLE_COUNT);
    int16_t mv0 = samples[TDS_SAMPLE_COUNT / 2];
}

```

Gambar 4. 16 Potongan Kode untuk Pembacaan Suhu dan Sensor TDS

Pada Gambar 4.16 menunjukkan proses pembacaan suhu dan pengambilan data sampel dari sensor TDS. Pembacaan suhu dieksekusi melalui pustaka DallasTemperature dengan memanggil perintah `getTempCByIndex(0)`. Nilai suhu ini langsung disimpan ke variabel global `currentTemp` apabila pembacaan dinilai valid (tidak sama dengan `DEVICE_DISCONNECTED_C`). Secara paralel, sistem melakukan perulangan (looping) sebanyak jumlah sampel untuk mengambil nilai dari pin analog sensor TDS. Menariknya, sistem tidak menggunakan fungsi `analogRead()` standar, melainkan menggunakan `analogReadMilliVolts(TDS_PIN)`. Perintah spesifik ESP32 ini memanfaatkan kurva kalibrasi Analog-to-Digital Converter (ADC) bawaan pabrik, sehingga nilai tegangan yang terbaca dalam satuan milivolt (mV) menjadi lebih linier dan akurat.

```

// Formula standar DFRobot TDS sensor + kompensasi suhu
float compensationCoefficient = 1.0 + 0.02 * (currentTemp - 25.0);
float compensationVoltage = voltage / compensationCoefficient;
float tds = (133.42 * pow(compensationVoltage, 3)
            - 255.86 * pow(compensationVoltage, 2)
            + 857.39 * compensationVoltage) * 0.5;

currentTDS = (tds < 0) ? 0 : tds;

Serial.printf("[Sensor] Suhu=%.1fC TDS_mV=%d TDS=%.1fppm\n", currentTemp, mv0, currentTDS);
}

```

Gambar 4. 17 Implementasi Perhitungan Nilai TDS Berdasarkan Sensor DFRobot

Gambar 4.17 menampilkan proses konversi matematis nilai tegangan sensor TDS menjadi nilai konduktivitas dalam satuan Parts Per Million (ppm). Karena konduktivitas air sangat dipengaruhi oleh perubahan suhu, program menerapkan faktor kompensasi suhu (temperature compensation) menggunakan variabel `currentTemp` yang baru saja dibaca dari sensor DS18B20. Koefisien kompensasi dihitung berdasarkan standar referensi suhu air 25°C dengan rasio perubahan sebesar 2% per derajat Celsius. Tegangan aktual kemudian dibagi dengan koefisien kompensasi tersebut. Nilai akhir TDS didapatkan dengan memasukkan tegangan yang telah dikompensasi ke dalam rumusan polinomial orde tiga standar pabrikan (seperti DFRobot), sehingga sistem mampu menjamin pembacaan konsentrasi nutrisi tetap akurat terlepas dari fluktuasi suhu air di tangki.

4.3.4 Program untuk Algoritma Kontrol PID Adaptif

Sub-bab ini membahas otak utama dari sistem otomasi, yaitu perancangan algoritma kontrol Proportional-Integral-Derivative (PID) yang mengkalkulasi kecepatan dan arah pompa dosing nutrisi maupun air baku. Sistem ini tidak menggunakan nilai gain (K_p , K_i , K_d) yang diatur secara manual (hardcoded), melainkan menerapkan metode penjadwalan gain heuristik (heuristic gain-scheduling). Pendekatan adaptif ini memungkinkan parameter kendali untuk terus menyesuaikan diri (auto-tuning) secara otomatis setiap saat berdasarkan besaran simpangan (error) antara nilai TDS aktual dengan target yang ditetapkan pengguna.

```

// ----- PID DENGAN AUTO-TUNING GAIN (Kp/Ki/Kd) -----
// makin cepat berubah -> makin agresif, Ki mengikui terbalik (hanya kuat
// saat error sudah kecil & stabil, untuk trim halus tanpa windup).
const double KP_MIN = 0.5, KP_MAX = 4.0;
const double KI_MIN = 0.01, KI_MAX = 0.15;
const double KD_MIN = 0.1, KD_MAX = 1.5;
const double ERROR_RATE_REF = 50.0; // ppm/detik yang dianggap "laju error cepat"

double pidIntegral = 0;
double pidLastError = 0;
unsigned long pidLastTime = 0;

double mapf(double x, double inMin, double inMax, double outMin, double outMax) {
    double t = (inMax == inMin) ? 0.0 : (x - inMin) / (inMax - inMin);
    t = constrain(t, 0.0, 1.0);
    return outMin + t * (outMax - outMin);
}

void autoTuneGains(double error, double errorRate) {
    double setpointRef = (config.setpointTDS > 1.0) ? config.setpointTDS : 1.0;
    double errNorm = constrain(fabs(error) / setpointRef, 0.0, 1.0);
    double rateNorm = constrain(fabs(errorRate) / ERROR_RATE_REF, 0.0, 1.0);

    config.kp = mapf(errNorm, 0.0, 1.0, KP_MIN, KP_MAX);
    config.ki = mapf(errNorm, 0.0, 1.0, KI_MAX, KI_MIN); // terbalik: error kecil -> Ki besar (trim halus)
    config.kd = mapf(rateNorm, 0.0, 1.0, KD_MIN, KD_MAX);
}

```

Gambar 4. 18 Potongan Kode untuk Penalaan PID Adaptif

Gambar 4.18 memperlihatkan mekanisme penyesuaian parameter kontrol PID secara dinamis (auto-tuning). Proses ini diawali dengan menormalisasi besar error (simpangan dari setpoint) dan laju perubahan error tersebut ke dalam skala 0,0 hingga 1,0. Nilai normalisasi ini kemudian dipetakan ke batas minimum dan maksimum gain yang aman melalui fungsi perantara `mapf()`. Logika heuristik yang diterapkan adalah: konstanta Proporsional (K_p) dan Derivatif (K_d) akan semakin agresif membesar ketika nilai error menjauh atau berubah sangat cepat. Sebaliknya, konstanta Integral (K_i) dipetakan secara terbalik; nilainya hanya akan menguat pada saat error sudah kecil dan stabil guna melakukan koreksi halus (fine-trimming) tanpa memicu terjadinya penumpukan kesalahan (integral windup).

```
double computePID(double error, double errorRate, double dt) {
    pidIntegral += error * dt;
    pidIntegral = constrain(pidIntegral, -500.0, 500.0); // anti-windup

    double output = config.kp * error + config.ki * pidIntegral + config.kd * errorRate;
    output = constrain(output, -100.0, 100.0);

    pidLastError = error;
    pidLastTime = millis();
    return output;
}
```

Gambar 4. 19 Potongan Kode untuk Perhitungan Sinyal Kendali

Pada Gambar 4.19 ditunjukkan penerapan matematis algoritma PID itu sendiri di dalam fungsi `computePID()`. Pertama, dilakukan kalkulasi nilai Integral dengan menjumlahkan error seiring berjalannya waktu (dt), disertai dengan mekanisme pembatasan nilai (anti-windup) untuk mencegah aktuator bekerja berlebihan. Selanjutnya, dilakukan penjumlahan dari komponen Proporsional, Integral, dan Derivatif menggunakan parameter gain (K_p , K_i , K_d) yang telah disesuaikan sebelumnya. Keluaran fungsi ini menghasilkan nilai persentase sinyal kendali dari rentang -100% hingga 100%, yang selanjutnya akan memandu sistem untuk memutuskan apakah tangki membutuhkan tambahan nutrisi atau tambahan air baku.

4.3.5 Program untuk Koneksi ke Rangkaian Aktuator

Sub-bab ini menjelaskan logika pemrograman yang bertugas mengeksekusi hasil perhitungan algoritma PID menjadi tindakan fisik nyata. Sistem ini menggunakan driver motor ganda L298N sebagai perantara (interface) antara mikrokontroler dengan aktuator berupa tiga buah pompa dosing DC (Pompa A untuk Nutrisi makro, Pompa B untuk Nutrisi mikro, dan Pompa C untuk Air Baku). Alih-alih hanya menghidupkan dan mematikan pompa secara kasar (kontrol On/Off), program ini memanfaatkan fitur Pulse Width Modulation (PWM) perangkat keras ESP32 untuk mengatur laju aliran (flow rate) masing-masing pompa secara proporsional.

```
// ----- KONTROL POMPA (MOTOR DRIVER L298N, KECEPATAN PROPORSIONAL) -----
// EN di board di jumper permanen ke 5V, jadi PWM kecepatan dikirim ke pin
// IN1/IN3 (pwmPins) -> lihat catatan di bagian PIN & HARDWARE di atas.
void pumpInit() {
    int pwmPins[3] = {PUMP_A_IN1, PUMP_B_IN3, PUMP_C_IN1};
    int dirPins[3] = {PUMP_A_IN2, PUMP_B_IN4, PUMP_C_IN2};
    int pwmChannels[3] = {PUMP_A_PWM_CHANNEL, PUMP_B_PWM_CHANNEL, PUMP_C_PWM_CHANNEL};

    for (int i = 0; i < 3; i++) {
        pinMode(dirPins[i], OUTPUT);
        digitalWrite(dirPins[i], LOW); // arah tetap maju, pompa dosing cuma butuh 1 arah
        ledcSetup(pwmChannels[i], PWM_FREQ_HZ, PWM_RES_BITS); // API core v2.0.x: channel, freq, resolusi
        ledcAttachPin(pwmPins[i], pwmChannels[i]); // hubungkan channel ke pin fisik
        ledcWrite(pwmChannels[i], 0);
    }
}
```

Gambar 4. 20 Inisialisasi Pin Kendali Pompa Peristaltik dan Modul L298N

Gambar 4.20 menampilkan proses inisialisasi pin aktuator pada awal program. Fungsi `pumpInit()` mengonfigurasi dua buah pin arah rotasi (IN1 dan IN2) serta satu pin pengaktif (EN) untuk setiap pompa yang terhubung ke driver motor L298N. Karena sistem dosing nutrisi hanya memerlukan aliran cairan satu arah, pin arah rotasi dikunci secara permanen (HIGH dan LOW) pada tahap inisialisasi ini. Selanjutnya, program mengaitkan pin pengaktif (EN) dengan modul periferif LEDC bawaan ESP32 menggunakan fungsi `ledcAttach()`. Modul ini diatur untuk membangkitkan sinyal PWM dengan frekuensi 1000 Hz dan resolusi 8-bit, yang memungkinkan kecepatan motor diatur ke dalam 256 tingkatan (0 hingga 255).

```

void updatePumps() {
    unsigned long now = millis();
    double dt = (pidLastTime == 0) ? (SENSOR_INTERVAL / 1000.0) : (now - pidLastTime) / 1000.0;
    if (dt <= 0) dt = 0.001;

    double error = config.setpointTDS - currentTDS;
    double errorRate = (error - pidLastError) / dt;

    if (config.autoTuneMode) {
        autoTuneGains(error, errorRate);
    } else {
        config.kp = config.manualKp;
        config.ki = config.manualKi;
        config.kd = config.manualKd;
    }
    pidOutput = computePID(error, errorRate, dt);

    // Deadband dibandingkan langsung terhadap error ppm (bukan pidOutput %),
    // supaya nilainya benar-benar berarti "toleransi ppm di sekitar setpoint"
    // sesuai label di halaman konfigurasi.
    bool butuhNutrisi = (error > config.deadband);
    bool butuhAirBaku = (error < -config.deadband);

    pumpSpeed = (uint8_t)constrain((fabs(pidOutput) / 100.0) * 255.0, 0.0, 255.0);
    pumpAOn = butuhNutrisi;
    pumpBOn = butuhNutrisi;
    pumpCOn = butuhAirBaku;
}

```

Gambar 4. 21 Implementasi Logika Kendali Aktuasi Pompa dengan Deadband

Pada Gambar 4.21 ditunjukkan eksekusi logika penggerak pompa di dalam fungsi `updatePumps()`. Setelah mendapatkan nilai keluaran dari kalkulasi PID (`pidOutput`), program akan membandingkan nilai error (simpangan aktual) terhadap batas toleransi yang ditetapkan (`deadband`). Jika nilai error melebihi batas positif, artinya kepekatan nutrisi kurang, sehingga Pompa A dan B (nutrisi) akan diaktifkan secara bersamaan. Sebaliknya, jika error melewati batas negatif, artinya larutan terlalu pekat, sehingga Pompa C (air baku) yang akan menyala. Kecepatan aktual pompa (`pumpSpeed`) ditentukan secara langsung oleh besaran absolut dari keluaran persentase PID, yang diubah ke dalam skala 8-bit, lalu dikirimkan ke driver motor menggunakan instruksi `ledcWrite()`. Mekanisme ini memastikan pemberian dosis berjalan mulus dan presisi.

4.3.6 Program untuk Koneksi ke Tampilan LCD

Sub-bab ini menguraikan tahapan pemrograman untuk menampilkan informasi operasional sistem secara lokal (on-site) melalui layar Liquid Crystal

Display (LCD) berukuran 20x4 karakter. Mengingat keterbatasan jumlah karakter yang dapat ditampilkan pada satu waktu, program dirancang menggunakan mekanisme paging (pemisahan halaman) bergantian. Selain itu, rutinitas khusus ditambahkan untuk mendeteksi dan memulihkan layar dari gangguan sinyal (noise) frekuensi tinggi yang sering ditimbulkan oleh kerja aktuator pompa DC.

```
// Hard Reset + Re-Init kalau memang error -> lcd
bool i2cBusHealthy() {
    Wire.beginTransmission(LCD_I2C_ADDR);
    return Wire.endTransmission() == 0;
}

void recoverI2CAndLCD() {
    Wire.end();
    delay(5);
    Wire.begin(21, 22);
    Wire.setClock(100000);
    lcd.init();
    lcd.backlight();
}
```

Gambar 4. 22 Fungsi Pengecekan Status dan Pemulihan Jalur Komunikasi I2C

Gambar 4.22 menunjukkan implementasi logika failsafe spesifik untuk jalur komunikasi perangkat I2C. Pengoperasian motor DC (pompa dosing) via PWM (driver L298N) sering kali memicu lonjakan arus balikan (noise/spikes) yang dapat mengganggu sinyal SDA/SCL. Gangguan ini umumnya menyebabkan tampilan LCD menjadi acak (gibberish) atau blank. Untuk menanggulangnya, program menggunakan fungsi `i2cBusHealthy()` guna menguji (ping) koneksi ke alamat I2C LCD (0x27) sebelum setiap pembaruan layar. Apabila bus terdeteksi error (`Wire.endTransmission() != 0`), fungsi `recoverI2CAndLCD()` akan dieksekusi secara otomatis untuk memutuskan koneksi I2C, memberikan jeda sesaat, dan melakukan inisialisasi ulang (`lcd.init()`) agar sinkronisasi chip HD44780 kembali normal tanpa perlu merestart seluruh sistem ESP32.

```

void updatePumps() {
    double dt = (pidLastTime == 0) ? (SENSOR_INTERVAL / 1000.0) : (now - pidLastTime) / 1000.0;
    if (dt <= 0) dt = 0.001;

    double error = config.setpointTDS - currentTDS;
    double errorRate = (error - pidLastError) / dt;

    if (config.autoTuneMode) {
        autoTuneGains(error, errorRate);
    } else {
        config.kp = config.manualKp;
        config.ki = config.manualKi;
        config.kd = config.manualKd;
    }
    pidOutput = computePID(error, errorRate, dt);

    // Deadband dibandingkan langsung terhadap error ppm (bukan pidOutput %),
    // supaya nilainya benar-benar berarti "toleransi ppm di sekitar setpoint"
    // sesuai label di halaman konfigurasi.
    bool butuhNutrisi = (error > config.deadband);
    bool butuhAirBaku = (error < -config.deadband);

    pumpSpeed = (uint8_t)constrain(fabs(pidOutput) / 100.0 * 255.0, 0.0, 255.0);
    pumpAOn = butuhNutrisi;
    pumpBOn = butuhNutrisi;
    pumpCOn = butuhAirBaku;

    ledcWrite(PUMP_A_PWM_CHANNEL, pumpAOn ? pumpSpeed : 0);
    ledcWrite(PUMP_B_PWM_CHANNEL, pumpBOn ? pumpSpeed : 0);
}

```

Gambar 4. 23 Potongan Kode Fungsi untuk Eksekusi PWM ke Motor Pompa

Pada Gambar 4.23 diperlihatkan logika pemrograman mekanisme paging untuk pembaruan tampilan LCD yang dijalankan setiap 1 detik. Karena ruang tampilan terbatas, informasi dibagi menjadi dua "halaman" yang bergantian secara otomatis setiap 4 detik (diatur oleh variabel `PAGE_INTERVAL`). Halaman pertama berfokus pada dinamika sistem kontrol, menampilkan nilai TDS aktual, nilai setpoint, pembacaan suhu air, serta status dan persentase kecepatan ketiga pompa secara langsung. Halaman kedua difokuskan pada diagnostik jaringan, menampilkan status keterhubungan Wi-Fi, status server Blynk, hasil transmisi terakhir ke Google Spreadsheet, serta pencatatan waktu aktual yang disinkronisasi langsung dari server *Network Time Protocol* (NTP).

4.3.7 Program Penanganan Gangguan (*Failsafe*)

Sub-bab ini menjelaskan rancangan ketahanan (*robustness*) perangkat lunak dalam menghadapi kondisi kegagalan sistem keras maupun jaringan. Pada sistem

berbasis *Internet of Things* (IoT) yang diletakkan di lingkungan *greenhouse*, putusnya koneksi Wi-Fi secara tiba-tiba akibat gangguan router atau pemadaman listrik sesaat adalah hal yang lumrah. Oleh karena itu, mikrokontroler ESP32 dibekali dengan mekanisme peralihan mode otomatis (*fallback mode*) agar kontrol nutrisi tetap berjalan dan tidak merugikan tanaman, sekaligus memberikan akses bagi pengguna untuk melakukan intervensi darurat.

```

if (WiFi.status() != WL_CONNECTED) {
  Serial.println("[WiFi] Gagal konek -> masuk mode Access Point konfigurasi");
  WiFi.mode(WIFI_AP);
  WiFi.softAP(AP_SSID, AP_PASSWORD);
  Serial.printf("[WiFi] AP aktif: SSID='%s' IP=%s\n", AP_SSID, WiFi.softAPIP().toString().c_str());
  Serial.printf("[WiFi] Hubungkan HP/laptop ke WiFi '%s' lalu buka 192.168.4.1\n", AP_SSID);
} else {
  Serial.printf("[WiFi] Terhubung! IP=%s RSSI=%dBm\n", WiFi.localIP().toString().c_str(), WiFi.RSSI());
  configTime(GMT_OFFSET_SEC, DST_OFFSET_SEC, NTP_SERVER); (const char [47])"[NTP] Sinkronisasi waktu dimulai (server: %s)\n"
  Serial.printf("[NTP] Sinkronisasi waktu dimulai (server: %s)\n", NTP_SERVER);
}

if (MDNS.begin(MDNS_HOSTNAME)) {
  Serial.printf("[mDNS] Aktif -> buka http://%s.local untuk konfigurasi\n", MDNS_HOSTNAME);
} else {
  Serial.println("[mDNS] Gagal memulai mDNS");
}

```

Gambar 4. 24 Inisialisasi Jaringan WiFi dan Mode Access Point

Gambar 4.24 memperlihatkan logika pertahanan utama jaringan pada fungsi `connectWiFi()`. Setelah gagal terhubung ke konfigurasi Wi-Fi utama (WIFI_STA) dalam batas waktu 15 detik, program tidak akan berhenti atau membekukan sistem (hang). Sebaliknya, program memerintahkan ESP32 untuk berpindah ke mode WIFI_AP (Access Point). Dalam mode ini, mikrokontroler akan memancarkan sinyal Wi-Fi-nya sendiri dengan Service Set Identifier (SSID) "Hidroponik-Setup" dan IP statis 192.168.4.1. Mekanisme ini berfungsi ganda: pertama, sebagai peringatan visual bahwa sistem sedang offline; kedua, membuka jalur lokal bagi pengguna (melalui smartphone atau laptop) untuk mengakses Web Server lokal secara langsung guna memperbaiki password Wi-Fi atau mengubah target TDS darurat meskipun tidak ada akses internet sama sekali.

```

Serial.printf("[Blynk] token panjang=%d, WiFi=%s\n", config.blynkToken.length(),
              WiFi.status() == WL_CONNECTED ? "connected" : "TIDAK connected");
if (config.blynkToken.length() > 0 && WiFi.status() == WL_CONNECTED) {
  Blynk.config(config.blynkToken.c_str());
  bool ok = Blynk.connect(5000);
  Serial.printf("[Blynk] connect() hasil = %s\n", ok ? "SUKSES" : "GAGAL (akan retry via Blynk.run())");
  blynkEnabled = true;
} else {
  Serial.println("[Blynk] dilewati -> token kosong atau WiFi belum konek");
}

```

Gambar 4. 25 Konfigurasi dan Koneksi ke Server Blynk IoT

Pada Gambar 4.25 ditunjukkan penanganan toleransi kegagalan (*fault tolerance*) pada saat menghubungkan modul ke server pemantauan awan (Blynk). Sistem diprogram untuk tidak bersikap menghalangi (*non-blocking*). Pemanggilan fungsi `Blynk.connect(5000)` diberikan batas tunggu (*timeout*) spesifik selama 5 detik. Jika dalam 5 detik server Blynk gagal merespons, fungsi akan mengembalikan nilai kegagalan, namun program utama (setup) tetap berlanjut menyelesaikan inisialisasinya hingga masuk ke kalang utama (`loop()`). Fungsi `Blynk.run()` di dalam kalang utama selanjutnya akan menangani upaya penyambungan ulang (*reconnection*) di latar belakang secara otomatis ketika akses internet kembali stabil, sehingga proses pemantauan sensor dan algoritma penggerak pompa tidak pernah terhenti.