LAMPIRAN

Lampiran 1. *Arduino IDE*

```
// ===== KONFIGURASI BLYNK IoT (NEW VERSION) - HARUS DI ATAS =====
#define BLYNK_TEMPLATE_ID "TMPL6xtuYSznF"
#define BLYNK_TEMPLATE_NAME "MAT TOMAT"
#define BLYNK_AUTH_TOKEN "6fAxNcWQq1sz5v8oP2YpAn8T2XTmpTow"
#define BLYNK_PRINT Serial

// ===== INCLUDE LIBRARIES =====
#include <WiFi.h>
#include <WiFiClient.h>
#include <BlynkSimpleEsp32.h>
#include <ESP32Servo.h>

// ===== KONFIGURASI WIFI =====
char ssid[] = "BOBY";
char pass[] = "yowesben";

// ===== PIN DEFINITION =====
// Sensor Warna TCS3200
#define S0 4
#define S1 5
#define S2 18
#define S3 19
#define sensorOut 34

// Actuator
#define SERVO_PIN 13

// ===== VIRTUAL PIN BLYNK =====
#define VPIN_COUNT V0      // Total tomat
#define VPIN_MENTAH V1     // Jumlah mentah
#define VPIN_MATANG V2     // Jumlah matang
#define VPIN_BUSUK V3      // Jumlah busuk
#define VPIN_STATUS V4     // Status terakhir
#define VPIN_RGB V5        // Nilai RGB terakhir
#define VPIN_RESET V6      // Tombol reset (WORKING!)

// ===== OBJECTS =====
Servo sortingServo;
BlynkTimer timer;

// ===== TIMING CONSTANTS =====
unsigned long SERVO_WAIT_TIME = 2000;        // 2 detik default (DYNAMIC - bisa
diubah!)
const unsigned long RETURN_CENTER_TIME = 5000; // 5 detik idle, kembali ke tengah
const int COLOR_READ_INTERVAL = 100;          // Baca warna setiap 100ms
const int MIN_OBJECT_DURATION = 300;          // Minimal 300ms untuk valid object
const int MIN_DETECTION_INTERVAL = 500;      // Minimal 1 detik antar tomat
const int STABLE_READINGS = 3;               // Butuh 3x pembacaan stabil
const int BLYNK_UPDATE_INTERVAL = 2000;       // Update Blynk setiap 2 detik
(REDUCED!)
```

```
// ===== SERVO POSITIONS =====
const int SERVO_KANAN = 45;    // Mentah
const int SERVO_TENGAH = 90;   // Matang
const int SERVO_KIRI = 135;    // Busuk

// ===== GLOBAL VARIABLES =====
// Statistics
int totalCount = 0;
int mentahCount = 0;
int matangCount = 0;
int busukCount = 0;

// Detection and timing state
bool objectDetected = false;
bool lastObjectState = false;
bool objectCounted = false;        // Prevent double counting
String currentClassification = "";
String pendingClassification = "";
unsigned long lastColorRead = 0;
unsigned long lastTomatoTime = 0;
unsigned long servoMoveTime = 0;
unsigned long lastBlynkUpdate = 0;  // Track Blynk updates
unsigned long objectStartTime = 0;  // Track object detection time
unsigned long lastDetectionTime = 0; // Track last valid detection
bool servoCanMove = true;
int currentServoPos = SERVO_TENGAH;
int targetServoPos = SERVO_TENGAH;

// Blynk change detection variables
int lastSentTotal = 0;
int lastSentMentah = 0;
int lastSentMatang = 0;
int lastSentBusuk = 0;
String lastSentStatus = "";

// Color calibration (dari kalibrasi sebelumnya)
int whiteR = 30, whiteG = 32, whiteB = 28;
int blackR = 317, blackG = 331, blackB = 285;

// Color dataset untuk klasifikasi
struct ColorData {
  int r, g, b;
  String type;
};

// Dataset warna tomat (simplified)
ColorData colorDatabase[] = {
  // MENTAH (hijau)
  {55, 50, 65, "MENTAH"},
  {78, 72, 55, "MENTAH"},
  {32, 28, 45, "MENTAH"},
  {153, 160, 85, "MENTAH"},
  {130, 136, 72, "MENTAH"},
  {107, 112, 60, "MENTAH"},

  // MATANG (kuning, oranye, merah)
  {181, 142, 76, "MATANG"}, // Kuning 3 cm
```

```
  {154, 121, 65, "MATANG"}, // Kuning 4 cm
  {127, 99, 53, "MATANG"},  // Kuning 5 cm
  {172, 71, 23, "MATANG"},  // Orange 3 cm
  {146, 60, 20, "MATANG"},  // Orange 4 cm
  {120, 50, 16, "MATANG"},  // Orange 5 cm
  {137, 0, 0, "MATANG"},    // Merah 3 cm
  {116, 0, 0, "MATANG"},    // Merah 4 cm
  {96, 0, 0, "MATANG"},     // Merah 5 cm

  // BUSUK (coklat)
  {127, 29, 0, "BUSUK"}
};

const int DATABASE_SIZE = sizeof(colorDatabase) / sizeof(ColorData);

// ===== FUNCTION DECLARATIONS =====
void setupPins();
void setupServo();
void setupWiFiBlynk();
// void buzzerBeep(int duration = 100);
int readColor(int colorType);
int frequencyToRGB(int freq, int minFreq, int maxFreq);
String classifyColor(int r, int g, int b);
int colorDistance(int r1, int g1, int b1, int r2, int g2, int b2);
void updateServoPosition(String classification);
void detectObjectRealtime();
void handleServoTiming();
void sendToBlynk();
void handleSerialCommand(char cmd);

// ===== SETUP FUNCTION =====
void setup() {
  Serial.begin(115200);
  delay(1000);
  Serial.println("\n===== SISTEM SORTIR TOMAT CONVEYOR =====");
  Serial.println("Dengan timing logic yang benar");

  setupPins();
  setupServo();
  setupWiFiBlynk();

  // Setup timer untuk update Blynk setiap 30 detik (ULTRA CONSERVATIVE!)
  timer.setInterval(30000L, sendToBlynk);

  lastTomatoTime = millis();

  Serial.println("\n=== SISTEM SIAP ===");
  Serial.println("SISTEM SORTIR TOMAT (LOGIC BENAR):");
  Serial.println("1. Tomat kebaca → SERVO LANGSUNG GERAK");
  Serial.println("2. Servo BLOCK 3 detik → biar tomat lewat");
  Serial.println("3. Tomat berikutnya → QUEUE sampai servo ready");
  Serial.println("4. Anti-collision system untuk tomat berurutan");
  Serial.println("5. Data real-time ke Blynk");
  Serial.println("\nPerfect untuk conveyor dengan tomat rapat!");
  Serial.println("Ketik 'h' untuk bantuan\n");

}
```

```
// ===== SETUP FUNCTIONS =====
void setupPins() {
  // TCS3200 setup
  pinMode(S0, OUTPUT);
  pinMode(S1, OUTPUT);
  pinMode(S2, OUTPUT);
  pinMode(S3, OUTPUT);
  pinMode(sensorOut, INPUT);

  // Set frequency scaling to 20%
  digitalWrite(S0, HIGH);
  digitalWrite(S1, LOW);

  // Buzzer removed - not used

  Serial.println("✓ Pins configured");
}

void setupServo() {
  ESP32PWM::allocateTimer(0);
  ESP32PWM::allocateTimer(1);
  ESP32PWM::allocateTimer(2);
  ESP32PWM::allocateTimer(3);

  sortingServo.setPeriodHertz(50);
  sortingServo.attach(SERVO_PIN, 500, 2400);

  delay(500);
  sortingServo.write(SERVO_TENGAH);
  currentServoPos = SERVO_TENGAH;
  targetServoPos = SERVO_TENGAH;

  Serial.println("✓ Servo initialized at 90°");
}

void setupWiFiBlynk() {
  Serial.print("Connecting to WiFi: ");
  Serial.println(ssid);
  WiFi.begin(ssid, pass);

  int attempts = 0;
  while (WiFi.status() != WL_CONNECTED && attempts < 30) {
    delay(500);
    Serial.print(".");
    attempts++;
  }

  if (WiFi.status() == WL_CONNECTED) {
    Serial.println("\n✓ WiFi connected!");
    Serial.print("IP: ");
    Serial.println(WiFi.localIP());

    Serial.println("Connecting to Blynk...");
    Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);
```

```
    delay(2000);
    if (Blynk.connected()) {
      Serial.println("✓ Blynk connected!");

      // Initialize dashboard
      Blynk.virtualWrite(VPIN_COUNT, 0);
      Blynk.virtualWrite(VPIN_MENTAH, 0);
      Blynk.virtualWrite(VPIN_MATANG, 0);
      Blynk.virtualWrite(VPIN_BUSUK, 0);
      Blynk.virtualWrite(VPIN_STATUS, "READY");

      Serial.println("✓ Dashboard initialized");
    } else {
      Serial.println("✗ Blynk connection failed");
    }
  } else {
    Serial.println("\n✗ WiFi failed - continuing offline");
  }
}

// ===== COLOR SENSOR FUNCTIONS =====
int readColor(int colorType) {
  switch(colorType) {
    case 0: // Red
      digitalWrite(S2, LOW);
      digitalWrite(S3, LOW);
      break;
    case 1: // Green
      digitalWrite(S2, HIGH);
      digitalWrite(S3, HIGH);
      break;
    case 2: // Blue
      digitalWrite(S2, LOW);
      digitalWrite(S3, HIGH);
      break;
  }

  delayMicroseconds(100);
  return pulseIn(sensorOut, LOW, 10000);
}

int frequencyToRGB(int freq, int minFreq, int maxFreq) {
  if (freq == 0) return 0;
  int rgb = map(freq, minFreq, maxFreq, 255, 0);
  return constrain(rgb, 0, 255);
}

int colorDistance(int r1, int g1, int b1, int r2, int g2, int b2) {
  return abs(r1 - r2) + abs(g1 - g2) + abs(b1 - b2);
}

// ===== CLASSIFICATION FUNCTION =====
String classifyColor(int r, int g, int b) {
  // Check if object present (brightness threshold)
  int brightness = (r + g + b) / 3;
```

```
  // DEBUG: Print brightness
  Serial.print(" [Brightness: ");
  Serial.print(brightness);
  Serial.print("]");

  if (brightness < 30) {
    return "BACKGROUND";
  }

  // Find closest color in database
  int minDistance = 999999;
  String bestMatch = "TIDAK_JELAS";

  for (int i = 0; i < DATABASE_SIZE; i++) {
    int distance = colorDistance(r, g, b, colorDatabase[i].r, colorDatabase[i].g,
colorDatabase[i].b);
    if (distance < minDistance) {
      minDistance = distance;
      bestMatch = colorDatabase[i].type;
    }
  }

  // Use adaptive threshold based on brightness
  int threshold = 100;
  if (brightness < 100) threshold = 120;
  else if (brightness > 200) threshold = 80;

  Serial.print(" [Distance: ");
  Serial.print(minDistance);
  Serial.print(", Threshold: ");
  Serial.print(threshold);
  Serial.print("]");

  if (minDistance <= threshold) {
    return bestMatch;
  }

  // Fallback ratio-based classification
  float total = r + g + b;
  if (total > 30) {
    float rRatio = r / total;
    float gRatio = g / total;

    Serial.print(" [R%: ");
    Serial.print(rRatio*100, 1);
    Serial.print(", G%: ");
    Serial.print(gRatio*100, 1);
    Serial.print("]");

    if (gRatio > 0.4 && gRatio > rRatio) {
      return "MENTAH";
    }
    else if (rRatio > 0.4) {
      return "MATANG";
    }
    else if (rRatio > 0.35 && brightness < 120) {
      return "BUSUK";
```

```
    }
   }

   return "TIDAK_JELAS";
}

// ===== SERVO CONTROL WITH TIMING (SIMPLIFIED) =====
void updateServoPosition(String classification) {
  // QUEUE SERVO COMMAND - DATA SUDAH MASUK!
  if (!servoCanMove) {
    pendingClassification = classification;
    Serial.print(">> Servo command QUEUED: ");
    Serial.print(classification);
    Serial.println(" (data sudah masuk, tunggu giliran servo)");
    return;
  }

  int newTarget = SERVO_TENGAH;

  if (classification == "MENTAH") {
    newTarget = SERVO_KANAN;
  }
  else if (classification == "MATANG") {
    newTarget = SERVO_TENGAH;
  }
  else if (classification == "BUSUK") {
    newTarget = SERVO_KIRI;
  }

  if (newTarget != currentServoPos) {
    // DIRECT MOVEMENT - NO SMOOTH STEPPING
    Serial.print(">> SERVO MOVING: ");
    Serial.print(currentServoPos);
    Serial.print("° → ");
    Serial.print(newTarget);
    Serial.print("° (");
    Serial.print(classification);
    Serial.println(")");

    // Move servo directly
    sortingServo.write(newTarget);
    currentServoPos = newTarget;
    targetServoPos = newTarget;

    // BLOCK servo untuk 3 detik berikutnya
    servoCanMove = false;
    servoMoveTime = millis();

  }
}

// ===== MAIN DETECTION FUNCTION (IMMEDIATE & SMOOTH) =====
void detectObjectRealtime() {
  unsigned long currentTime = millis();

  // Read color every COLOR_READ_INTERVAL (CONTINUOUS!)
  if (currentTime - lastColorRead >= COLOR_READ_INTERVAL) {
```

```
    // Read RGB values
    int rFreq = readColor(0);
    int gFreq = readColor(1);
    int bFreq = readColor(2);

    // Convert to RGB
    int r = frequencyToRGB(rFreq, whiteR, blackR);
    int g = frequencyToRGB(gFreq, whiteG, blackG);
    int b = frequencyToRGB(bFreq, whiteB, blackB);

    // Classify
    String classification = classifyColor(r, g, b);

    // Print readings (continuous & smooth)
    Serial.print("RGB: ");
    Serial.print(r); Serial.print(",");
    Serial.print(g); Serial.print(",");
    Serial.print(b); Serial.print(" -> ");
    Serial.print(classification);
    Serial.print(" | WiFi: ");
    Serial.print(WiFi.status() == WL_CONNECTED ? "OK" : "NO");
    Serial.print(" | Blynk: ");
    Serial.println(Blynk.connected() ? "OK" : "NO");

    // Send RGB to Blynk (IMPROVED FREQUENCY)
    if (WiFi.status() == WL_CONNECTED && Blynk.connected()) {
      unsigned long currentTime = millis();
      if (currentTime - lastBlynkUpdate >= 3000) {  // Every 3 seconds (more frequent)
        String rgbData = String(r) + "," + String(g) + "," + String(b);
        Blynk.virtualWrite(VPIN_RGB, rgbData);
        lastBlynkUpdate = currentTime;
        Serial.print("RGB sent to Blynk: ");
        Serial.println(rgbData);
      }
    }

    // IMMEDIATE DETECTION & PROCESSING (WITH DEBOUNCING)
    bool currentObjectState = (classification != "BACKGROUND" && classification !=
"TIDAK_JELAS");

    // NEW OBJECT DETECTED - PROCESS ONLY ONCE!
    if (currentObjectState && !lastObjectState && !objectCounted) {

      // Check minimum interval between detections
      if (currentTime - lastDetectionTime >= MIN_DETECTION_INTERVAL) {
        Serial.println("\n=== TOMAT TERDETEKSI ===");

        objectStartTime = currentTime;
        objectCounted = true;  // Mark as counted to prevent double counting
        lastDetectionTime = currentTime; // Update last detection time

      // DATA LANGSUNG MASUK!
      totalCount++;
      if (classification == "MENTAH") {
        mentahCount++;
      }
      else if (classification == "MATANG") {
```

```
  matangCount++;
 }
 else if (classification == "BUSUK") {
  busukCount++;
 }

 Serial.println("*** DATA PROCESSED IMMEDIATELY ***");
 Serial.print("Klasifikasi: ");
 Serial.println(classification);
 Serial.print("RGB: ");
 Serial.print(r); Serial.print(",");
 Serial.print(g); Serial.print(",");
 Serial.println(b);
 Serial.print("Total: "); Serial.print(totalCount);
 Serial.print(" (M:"); Serial.print(mentahCount);
 Serial.print(" T:"); Serial.print(matangCount);
 Serial.print(" B:"); Serial.print(busukCount); Serial.println(")");

 // SERVO COMMAND (dengan delay 3 detik)
 updateServoPosition(classification);

 // Update Blynk - SMART UPDATE (hanya saat data berubah!)
 if (WiFi.status() == WL_CONNECTED && Blynk.connected()) {
  static unsigned long lastCounterUpdate = 0;
  unsigned long currentTime = millis();

  // Check if any data changed
  bool dataChanged = (totalCount != lastSentTotal ||
                      mentahCount != lastSentMentah ||
                      matangCount != lastSentMatang ||
                      busukCount != lastSentBusuk ||
                      classification != lastSentStatus);

  // Update conditions: data changed AND (5 seconds passed OR significant change)
  bool shouldUpdate = false;
  if (dataChanged) {
   if (currentTime - lastCounterUpdate >= 5000) shouldUpdate = true; // 5 detik max
   if (totalCount - lastSentTotal >= 3) shouldUpdate = true;         // 3 tomat baru
  }

  if (shouldUpdate) {
   // Send only CHANGED values to save messages
   int messagesSent = 0;

   if (totalCount != lastSentTotal) {
    Blynk.virtualWrite(VPIN_COUNT, totalCount);
    lastSentTotal = totalCount;
    messagesSent++;
    delay(50);
   }

   if (mentahCount != lastSentMentah) {
    Blynk.virtualWrite(VPIN_MENTAH, mentahCount);
    lastSentMentah = mentahCount;
    messagesSent++;
    delay(50);
   }
```

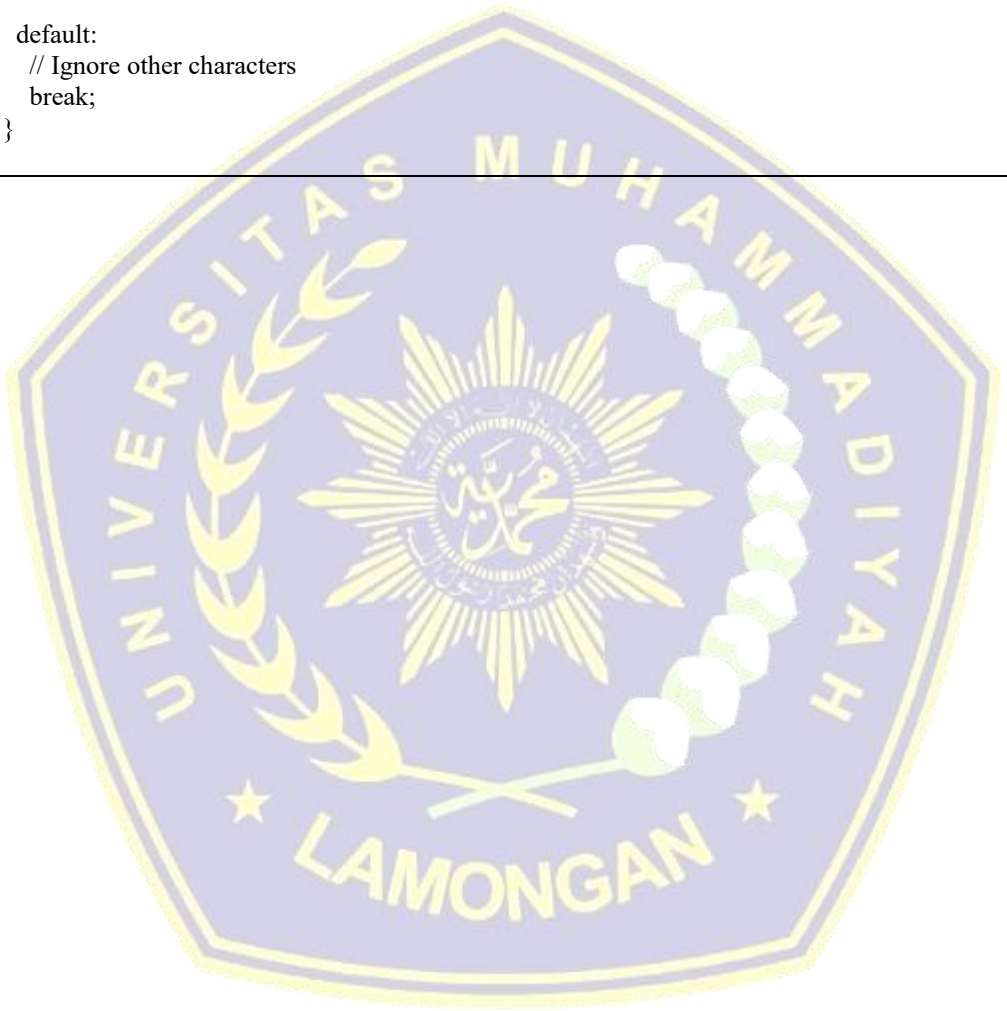
```
    if (matangCount != lastSentMatang) {
      Blynk.virtualWrite(VPIN_MATANG, matangCount);
      lastSentMatang = matangCount;
      messagesSent++;
      delay(50);
    }

    if (busukCount != lastSentBusuk) {
      Blynk.virtualWrite(VPIN_BUSUK, busukCount);
      lastSentBusuk = busukCount;
      messagesSent++;
      delay(50);
    }

    if (classification != lastSentStatus) {
      Blynk.virtualWrite(VPIN_STATUS, classification);
      lastSentStatus = classification;
      messagesSent++;
      delay(50);
    }

    // Force RGB update when tomato detected
    String currentRGB = String(r) + "," + String(g) + "," + String(b);
    Blynk.virtualWrite(VPIN_RGB, currentRGB);
    messagesSent++;
    delay(50);

    lastCounterUpdate = currentTime;

    Serial.print("✓ SMART UPDATE to Blynk (");
    Serial.print(messagesSent);
    Serial.print(" msgs) - Total: ");
    Serial.print(totalCount);
    Serial.print(" (M:");
    Serial.print(mentahCount);
    Serial.print(" T:");
    Serial.print(matangCount);
    Serial.print(" B:");
    Serial.print(busukCount);
    Serial.println(")");
  } else if (dataChanged) {
    Serial.print("✓ Data changed but waiting (next update in ");
    Serial.print((5000 - (currentTime - lastCounterUpdate))/1000);
    Serial.println("s)");
  } else {
    Serial.println("✓ Data processed (no changes for Blynk)");
  }
} else {
  Serial.println("✓ Data processed (Blynk not connected)");
}

      lastTomatoTime = currentTime;
} else {
  Serial.print("Detection ignored - too soon (");
```

```
        Serial.print((MIN_DETECTION_INTERVAL - (currentTime -
lastDetectionTime))/1000.0, 1);
        Serial.println("s remaining)");
      }
    }

    // OBJECT LOST - RESET COUNTING FLAG
    if (!currentObjectState && lastObjectState) {
      Serial.println("Objek hilang dari sensor");

      // Reset counting flag after object completely passes
      unsigned long objectDuration = currentTime - objectStartTime;
      if (objectDuration >= 500) {  // Minimum 500ms object duration
        objectCounted = false;  // Allow counting next object
        Serial.println("Ready for next object");
      }
    }

    lastObjectState = currentObjectState;
    lastColorRead = currentTime;
  }
}

// ===== SERVO TIMING HANDLER =====
void handleServoTiming() {
  unsigned long currentTime = millis();

  // Handle servo wait time (3 detik biar tomat lewat)
  if (!servoCanMove) {
    unsigned long elapsedWait = currentTime - servoMoveTime;

    if (elapsedWait >= SERVO_WAIT_TIME) {
      servoCanMove = true;
      Serial.print(">> SERVO READY! (");
      Serial.print(SERVO_WAIT_TIME / 1000.0, 1);
      Serial.println(" detik selesai)");

      // Process pending command (kalau ada tomat berikutnya)
      if (pendingClassification != "" && pendingClassification != "BACKGROUND") {
        Serial.print(">> Processing QUEUED tomat: ");
        Serial.println(pendingClassification);
        updateServoPosition(pendingClassification);
        pendingClassification = "";
      }
    }
  }

  // Handle auto return to center (5 detik idle)
  unsigned long timeSinceLastTomato = currentTime - lastTomatoTime;

  if (timeSinceLastTomato >= RETURN_CENTER_TIME &&
      currentServoPos != SERVO_TENGAH &&
      servoCanMove) {

    Serial.println("\n>> 5 detik idle - servo kembali ke tengah");
    sortingServo.write(SERVO_TENGAH);
    currentServoPos = SERVO_TENGAH;
```

```
    targetServoPos = SERVO_TENGAH;
    lastTomatoTime = currentTime;
  }
}

// ===== BLYNK FUNCTIONS (CHANGE DETECTION) =====
void sendToBlynk() {
  // Periodic sync - hanya kirim yang berubah!
  if (WiFi.status() == WL_CONNECTED && Blynk.connected()) {
    static unsigned long lastPeriodicUpdate = 0;
    unsigned long currentTime = millis();

    if (currentTime - lastPeriodicUpdate >= 30000) {  // Every 30 seconds
      int messagesSent = 0;

      // Only send data that changed since last update
      if (totalCount != lastSentTotal) {
        Blynk.virtualWrite(VPIN_COUNT, totalCount);
        lastSentTotal = totalCount;
        messagesSent++;
        delay(100);
      }

      if (mentahCount != lastSentMentah) {
        Blynk.virtualWrite(VPIN_MENTAH, mentahCount);
        lastSentMentah = mentahCount;
        messagesSent++;
        delay(100);
      }

      if (matangCount != lastSentMatang) {
        Blynk.virtualWrite(VPIN_MATANG, matangCount);
        lastSentMatang = matangCount;
        messagesSent++;
        delay(100);
      }

      if (busukCount != lastSentBusuk) {
        Blynk.virtualWrite(VPIN_BUSUK, busukCount);
        lastSentBusuk = busukCount;
        messagesSent++;
        delay(100);
      }

      lastPeriodicUpdate = currentTime;

      if (messagesSent > 0) {
        Serial.print("Periodic sync (");
        Serial.print(messagesSent);
        Serial.print(" changes) - Total: ");
        Serial.println(totalCount);
      } else {
        Serial.println("Periodic sync - no changes");
      }
    }
  }
}
```

```
// Reset dari Blynk (WORKING!)
BLYNK_WRITE(VPIN_RESET) {
  int buttonState = param.asInt();

  if (buttonState == 1) {
    // Reset all counters immediately
    totalCount = 0;
    mentahCount = 0;
    matangCount = 0;
    busukCount = 0;

    // Reset tracking variables
    lastSentTotal = 0;
    lastSentMentah = 0;
    lastSentMatang = 0;
    lastSentBusuk = 0;
    lastSentStatus = "";

    Serial.println("✓ Counters reset from Blynk!");

    // Force update dashboard immediately
    if (WiFi.status() == WL_CONNECTED && Blynk.connected()) {
      Blynk.virtualWrite(VPIN_COUNT, 0);
      delay(100);
      Blynk.virtualWrite(VPIN_MENTAH, 0);
      delay(100);
      Blynk.virtualWrite(VPIN_MATANG, 0);
      delay(100);
      Blynk.virtualWrite(VPIN_BUSUK, 0);
      delay(100);
      Blynk.virtualWrite(VPIN_STATUS, "RESET");

      Serial.println("✓ Dashboard reset completed!");
    }
  }
}

// ===== MAIN LOOP =====
void loop() {
  // Cek dan reconnect WiFi/Blynk jika perlu
  static unsigned long lastConnectionCheck = 0;
  if (millis() - lastConnectionCheck > 10000) {  // Cek setiap 10 detik
    if (WiFi.status() != WL_CONNECTED) {
      Serial.println("WiFi disconnected! Reconnecting...");
      WiFi.begin(ssid, pass);
      delay(5000);
    }

    if (WiFi.status() == WL_CONNECTED && !Blynk.connected()) {
      Serial.println("Blynk disconnected! Reconnecting...");
      Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);
      delay(2000);
    }

    lastConnectionCheck = millis();
```

```
  }

  // Blynk enabled with ultra conservative rate limiting
  if (WiFi.status() == WL_CONNECTED) {
    Blynk.run();
    timer.run();
  }

  detectObjectRealtime();
  handleServoTiming();
  }
  delay(50);  // Small delay for stability
}
    default:
      // Ignore other characters
      break;
  }
}
```



Lampiran 2 Dokumentasi