|  |  |
| --- | --- |
| KODINGAN UTAMA SISTEM | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203 | #include <Wire.h> // mengaktifkan komunikasi I2C (Inter-Integrated Circuit)  #include "MAX30105.h" // mendeteksi sinyal PPG, detak jantung, saturasi oksigen, dan aktivitas sensor inframerah  #include <Adafruit\_GFX.h> // pustaka grafis dasar untuk menggambar teks, bentuk, dan grafik  #include <Adafruit\_SSD1306.h> // Pustaka untuk mengontrol layar OLED berbasis driver SSD1306  #include "DecisionTree.h" // implementasi model Decision Tree yang sudah dilatih sebelumnya  Eloquent::ML::Port::DecisionTree clf; // clf adalah objek yang merepresentasikan model Decision Tree. Eloquent berarti model Decision Tree yang dilatih pada platform seperti Python dapat diekspor ke format .h untuk digunakan di Arduino  #define SCREEN\_WIDTH 128 // mendefinisikan konstanta untuk lebar layar OLED resolusi 128 piksel horizontal  #define SCREEN\_HEIGHT 32 // mendefinisikan konstanta untuk tinggi layar OLED resolusi 32 piksel vertikal  #define OLED\_RESET -1 // menunjukkan bahwa layar OLED tidak memiliki pin reset fisik, sehingga fitur reset tidak digunakan  #define SCREEN\_ADDRESS 0x3C // mendefinisikan alamat I2C dari layar OLED. Nilai 0x3C adalah alamat default untuk driver OLED berbasis SSD1306.  Adafruit\_SSD1306 display(SCREEN\_WIDTH, SCREEN\_HEIGHT, &Wire, OLED\_RESET); //membuat objek bernama display dari pustaka Adafruit\_SSD1306 untuk mengontrol layar OLED.  MAX30105 particleSensor; // Membuat objek bernama particleSensor dari pustaka MAX30105 untuk mengontrol sensor MAX30105 (atau MAX30102) untuk membaca data seperti BPM, Sinyal PPG, dan nilai IR.  const int bufferSize = 4; // menunjukkan ukuran buffer untuk menyimpan nilai IR  long buffer[bufferSize]; // array untuk menyimpan nilai IR  int bufferIndex = 0; // indeks saat ini dalam buffer  long previousValue = 0; // menandakan nilai IR sebelumnya untuk mendeteksi kenaikan sinyal  long peakThreshold = 20000; // ambang batas untuk deteksi puncak sinyal  unsigned long lastPeakTime = 0; // waktu puncak terakhir  unsigned long minPeakInterval = 600; // menandakan interval minimum antar puncak untuk deteksi detak jantung  const int nnBufferSize = 100; // mendefinisikan konstanta nnBufferSize yang menentukan ukuran array nnIntervals dengan nilai 100  unsigned long nnIntervals[nnBufferSize]; // mendeklarasikan array nnIntervals dengan ukuran 100  int nnIndex = 0; // mendeklarasikan variabel nnIndex dengan nilai awal 0  // Pin for LEDs and buzzer  const int redLED = 3;  const int greenLED = 8;  const int buzzer = 9;  // Deklarasi fungsi  float calculateAVNN(); // mendeklarasikan fungsi untuk menghitung AVNN  float calculateSDNN(float avnn); // mendeklarasikan fungsi untuk menghitung SDNN yang menerima satu parameter yaitu avnn: sebuah nilai float .  void displayResult(float avnn, float sdnn, unsigned long computationTime, int stressDetected); // mendeklarasikan fungsi bernama displayResult yang menerima empat parameter yaitu avnn: nilai float, sdnn: nilai float, computationTime: waktu komputasi, stressDetected: hasil deteksi stres bertipe int.  void setup() { // bagian dari program Arduino yang hanya dijalankan sekali saat board dinyalakan atau di-reset.    Serial.begin(115200); // inisialisasi komunikasi serial dengan baud rate 115200    Serial.println("Initializing...");    // Inisialisasi OLED    if (!display.begin(SSD1306\_SWITCHCAPVCC, SCREEN\_ADDRESS)) { // memulai OLED display dengan komunikasi I2C menggunakan alamat SCREEN\_ADDRESS. Jika display tidak ditemukan atau gagal diinisialisasi, program akan masuk ke dalam blok if      Serial.println(F("OLED tidak terdeteksi"));      while (1); // program berhenti di sini jika OLED tidak terdeteksi, untuk mencegah program melanjutkan eksekusi    }    display.clearDisplay(); // membersihkan tampilan pada OLED display    display.display(); // menampilkan konten yang telah dipersiapkan pada layar OLED    // Inisialisasi sensor MAX30105    if (!particleSensor.begin(Wire, I2C\_SPEED\_FAST)) { // inisialisasi sensor MAX30105 menggunakan komunikasi I2C dengan kecepatan tinggi. Jika sensor tidak ditemukan, program akan masuk ke dalam blok if      Serial.println("MAX30105 was not found. Please check wiring/power.");      while (1); // program berhenti di sini jika sensor tidak terdeteksi, untuk mencegah program melanjutkan eksekusi    }    byte ledBrightness = 0x1F; // mengatur kecerahan LED pada sensor MAX30102    byte sampleAverage = 8; // menentukan jumlah sampel yang akan diambil dan dihitung rata-ratanya. Nilai 8 berarti sensor akan menghitung rata-rata dari 8 sampel sebelum menghasilkan data.    byte ledMode = 3; // menggunakan LED merah, IR, dan hijau untuk pengukuran lebih luas secara bersamaan    int sampleRate = 100; // menentukan laju pengambilan sampel sensor (sampling rate), yang mengontrol seberapa sering sensor mengukur data. Nilai 100 berarti sensor akan mengambil data sebanyak 100 kali per detik (100 Hz). Semakin tinggi nilai sampling rate, semakin banyak data yang diambil, yang dapat meningkatkan akurasi tetapi membutuhkan lebih banyak pemrosesan dan konsumsi daya.    int pulseWidth = 411; // menentukan lebar pulsa LED yang digunakan dalam pengukuran. Lebar pulsa ini mempengaruhi durasi cahaya yang dipancarkan oleh LED    int adcRange = 4096; // menentukan rentang ADC (Analog-to-Digital Converter) sensor, yang mengontrol resolusi pengukuran. Nilai 4096 berarti sensor menggunakan rentang 4096 level (12-bit ADC). Nilai yang lebih tinggi memberikan resolusi lebih tinggi, yang memungkinkan sensor mengukur dengan akurasi lebih baik, tetapi juga mengonsumsi lebih banyak daya dan memerlukan lebih banyak pemrosesan    particleSensor.setup(ledBrightness, sampleAverage, ledMode, sampleRate, pulseWidth, adcRange);    for (int i = 0; i < bufferSize; i++) buffer[i] = 0; // menginisialisasi buffer dengan nilai 0. Mengisi array buffer dengan nilai 0 pada setiap elemen. Ini dilakukan untuk membersihkan atau menginisialisasi buffer sebelum digunakan untuk menyimpan data sensor, memastikan tidak ada nilai sampah yang tersisa dari pengukuran sebelumnya.    for (int i = 0; i < nnBufferSize; i++) nnIntervals[i] = 0; // menginisialisasi array nnIntervals dengan nilai 0. Mengisi array nnIntervals dengan nilai 0 pada setiap elemen. Array ini mungkin digunakan untuk menyimpan interval antara dua detak jantung berturut-turut, dan inisialisasi dilakukan untuk memastikan tidak ada nilai yang tidak diinginkan sebelum data dimulai.    // Inisialisasi pin LED dan buzzer    pinMode(redLED, OUTPUT);    pinMode(greenLED, OUTPUT);    pinMode(buzzer, OUTPUT);    // Set LED and buzzer off initially    digitalWrite(redLED, LOW); // digunakan untuk mengubah status pin digital ke HIGH atau LOW. LOW berarti tidak ada arus yang mengalir ke pin, yang menyebabkan LED merah mati    digitalWrite(greenLED, LOW); // digunakan untuk mengubah status pin digital ke HIGH atau LOW. LOW berarti tidak ada arus yang mengalir ke pin, yang menyebabkan LED hijau mati    digitalWrite(buzzer, LOW); // digunakan untuk mengubah status pin digital ke HIGH atau LOW. LOW berarti tidak ada arus yang mengalir ke pin, yang menyebabkan buzzer mati  }  void loop() {    long currentValue = particleSensor.getIR(); // mengambil nilai IR yang diukur oleh sensor MAX30102    unsigned long currentTime = millis(); // menyimpan waktu saat ini sejak program dimulai    buffer[bufferIndex] = currentValue; // bufferIndex menunjukkan posisi di dalam array buffer tempat nilai sensor disimpan. Setelah nilai disimpan, bufferIndex akan diperbarui    bufferIndex = (bufferIndex + 1) % bufferSize; // indeks buffer diperbarui dengan cara menambahkannya 1 dan menggunakan operasi modulo (%) untuk memastikan bahwa indeks tetap berada dalam batas ukuran buffer (bufferSize). Jika indeks melebihi ukuran buffer, ia akan kembali ke 0 (sirkulasi buffer)    long avgValue = 0; // mendeklarasikan variabel untuk menyimpan nilai rata-rata dari buffer    for (int i = 0; i < bufferSize; i++) avgValue += buffer[i]; // untuk setiap elemen dalam buffer, nilai sensor ditambahkan ke avgValue, untuk menghitung total nilai    avgValue /= bufferSize; // setelah semua nilai dalam buffer dijumlahkan, avgValue dibagi dengan bufferSize untuk mendapatkan rata-rata nilai    if (currentValue > peakThreshold && currentValue > previousValue && (currentTime - lastPeakTime) > minPeakInterval) { // memeriksa apakah nilai currentValue lebih besar dari ambang batas peakThreshold, apakah lebih besar dari nilai sebelumnya (previousValue), dan apakah interval waktu antara puncak terakhir (lastPeakTime) lebih besar dari nilai minimum (minPeakInterval). Jika semua kondisi ini benar, maka ada puncak baru yang terdeteksi      if (lastPeakTime > 0) { // memastikan bahwa lastPeakTime sudah diatur sebelumnya, dan detak pertama dapat dilewati karena tidak memiliki puncak sebelumnya untuk dihitung intervalnya        unsigned long peakInterval = currentTime - lastPeakTime; // selisih antara waktu saat ini (currentTime) dan waktu puncak terakhir (lastPeakTime) memberi interval waktu antara dua puncak detak jantung berturut-turut        if (peakInterval > 300 && peakInterval < 2000) { // memeriksa interval antara dua puncak detak jantung harus lebih besar dari 300 ms dan kurang dari 2000 ms untuk dianggap valid          nnIntervals[nnIndex] = peakInterval; // menyimpan interval puncak ke dalam array nnIntervals          nnIndex = (nnIndex + 1) % nnBufferSize; // nnIndex diperbarui untuk menyimpan nilai interval NN berikutnya. Operasi modulo memastikan indeks tetap berada dalam ukuran buffer nnBufferSize          unsigned long startTime = millis(); // menyimpan waktu mulai perhitungan untuk menghitung waktu komputasi          float avnn = calculateAVNN(); // menghitung nilai AVNN          float sdnn = calculateSDNN(avnn); // menghitung nilai SDNN          // Persiapkan fitur untuk Decision Tree          float features[] = {avnn, sdnn}; // array features menyimpan nilai AVNN dan SDNN yang akan digunakan sebagai input untuk model Decision Tree yang akan mendeteksi stres          int stressDetected = clf.predict(features); // menggunakan model Decision Tree untuk mendeteksi stres          unsigned long computationTime = millis() - startTime;          displayResult(avnn, sdnn, computationTime, stressDetected);          // Update LED and buzzer based on stress          if (stressDetected == 1) { // Stress            digitalWrite(redLED, HIGH);   // Red LED on            digitalWrite(greenLED, LOW);  // Green LED off            digitalWrite(buzzer, HIGH);          } else { // No stress            digitalWrite(redLED, LOW);    // Red LED off            digitalWrite(greenLED, HIGH); // Green LED on            digitalWrite(buzzer, LOW);          }        } else {          Serial.println("Artefak terdeteksi, interval diabaikan.");        }      }      lastPeakTime = currentTime; // mengatur waktu puncak terakhir (lastPeakTime) menjadi waktu saat ini    }    previousValue = avgValue; // menyimpan nilai rata-rata yang dihitung (avgValue) ke dalam variabel previousValue    delay(10);  }  // Fungsi untuk menghitung AVNN  float calculateAVNN() { // mengembalikan nilai AVNN dalam bentuk float, yg berarti hasilnya berupa angka desimal    unsigned long total = 0; // inisialisasi variabel total dengan nilai 0. Variabel ini akan digunakan untuk menjumlahkan semua nilai interval NN yang valid    int count = 0; // inisialisasi variabel count dengan nilai 0. Variabel ini akan menghitung berapa banyak interval NN yang valid    for (int i = 0; i < nnBufferSize; i++) { // menelusuri array nnIntervals[] sebanyak nnBufferSize untuk menghitung total dan jumlah interval NN yang valid      if (nnIntervals[i] > 0) { // mengecek apakah interval NN pada posisi i lebih besar dari 0 (berarti valid). Jika valid, interval akan dihitung        total += nnIntervals[i]; // menambahkan nilai interval NN yang valid ke dalam total        count++; // meningkatkan penghitung count setiap kali ada interval NN yang valid      }    }    if (count == 0) return 0; // jika tidak ada interval NN yang valid (yaitu count == 0), maka fungsi mengembalikan nilai 0 untuk menghindari pembagian dengan nol    return (float)total / count; // menghitung AVNN dengan membagi total interval yang valid dengan jumlah interval yang valid  }  // Fungsi untuk menghitung SDNN  float calculateSDNN(float avnn) { // menghitung SDNN, yang merupakan akar kuadrat dari varians dari interval NN dengan menggunakan parameter avnn (yang dihitung dari fungsi calculateAVNN())    float variance = 0; // inisialisasi variabel variance dengan nilai 0. Variabel ini akan digunakan untuk menghitung varians dari interval NN    int count = 0;    for (int i = 0; i < nnBufferSize; i++) {      if (nnIntervals[i] > 0) {        variance += pow(nnIntervals[i] - avnn, 2); // menghitung selisih kuadrat antara interval NN yang valid dan nilai rata-rata AVNN, kemudian menambahkannya ke dalam variance        count++;      }    }    if (count == 0) return 0;    return sqrt(variance / (count – 1)); // menghitung **SDNN** dengan mengambil akar kuadrat dari varians yang dibagi dengan jumlah interval NN yang valid dikurangi dengan 1  }  // Fungsi untuk menampilkan hasil pada Serial dan OLED  void displayResult(float avnn, float sdnn, unsigned long computationTime, int stressDetected) {    display.clearDisplay();    Serial.print("AVNN: ");    Serial.print(avnn);    Serial.println(" ms");    Serial.print("SDNN: ");    Serial.print(sdnn);    Serial.println(" ms");    Serial.print("Computation Time: ");    Serial.print(computationTime);    Serial.println(" s");    Serial.print("Status: ");    Serial.println(stressDetected == 1 ? "Stres" : "Tidak Stress");    // Tampilkan data pada OLED    display.setTextSize(1);    display.setTextColor(SSD1306\_WHITE);    display.setCursor(0, 0);    display.print("AVNN: ");    display.print(avnn);    display.println(" ms");    display.setCursor(0, 8);    display.print("SDNN: ");    display.print(sdnn);    display.println(" ms");    display.setCursor(0, 16);    display.print("Status: ");    display.println(stressDetected == 1 ? "Stres" : "Tidak Stres");    display.setCursor(0, 24);    display.print("Komputasi: ");    display.print(computationTime);    display.println(" s");    display.display();  } |