LAMPIRAN

LAMPIRAN

```
// Library
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <SPI.h>
#include <max6675.h>
#include <MPU6050.h>
#include <PZEM004Tv30.h>
#include <SoftwareSerial.h>
// Pin
#define BUTTON_PIN 3     // Tombol Display
#define SET_BUTTON_PIN 2 // Tombol set
#define POTENSIO_PIN A0  // Potensio
#define CS_PIN 6        // MAX6675 chip select pin
#define SCK_PIN 7      // MAX6675 clock pin
#define SO_PIN 5        // MAX6675 data pin
// PZEM-004T
#define PZEM_RX_PIN 14, 16, 18
#define PZEM_TX_PIN 15, 17, 19
// Object
LiquidCrystal_I2C lcd(0x27, 16, 2);
MPU6050 mpu;
MAX6675 thermocouple(SCK_PIN, CS_PIN, SO_PIN);
PZEM004Tv30 pzem1(&Serial3, 0x01);
PZEM004Tv30 pzem2(&Serial2, 0x02);
PZEM004Tv30 pzem3(&Serial1, 0x03);
```

```
// Variabel
enum DisplayMode {
  DISPLAY_FUZZY_STATUS,
  DISPLAY_PHASE_R,
  DISPLAY_PHASE_S,
  DISPLAY_PHASE_T,
  DISPLAY_POWER_FACTOR_ENERGY,
  DISPLAY_TEMPERATURE,
  DISPLAY_VIBRATION,
  DISPLAY_AMPERE_SETTING,
  DISPLAY_MODES_COUNT
};
DisplayMode currentMode = DISPLAY_FUZZY_STATUS;
unsigned long lastButtonPress = 0;
unsigned long lastSetButtonPress = 0;
const unsigned long debounceTime = 200;
unsigned long lastDataLogTime = 0;
const unsigned long dataLogInterval = 1000;
// Ampere set
float referenceAmpere = 5.0;  // Default ampere
float potensioValue = 0;
bool isSettingMode = false;
unsigned long settingModeStartTime = 0;
const unsigned long settingModeTimeout = 15000;

// Sensor data
float voltageR = 0, currentR = 0, powerR = 0, frequencyR = 0;
```

```
float voltageS = 0, currentS = 0, powerS = 0, frequencyS = 0;
float voltageT = 0, currentT = 0, powerT = 0, frequencyT = 0;
float energy = 0, powerFactor = 0;
float temperature = 0;
float vibrationX = 0, vibrationY = 0, vibrationZ = 0, totalVibration = 0;
// Sensor read flag
bool sensorDataReady = false;
unsigned long lastSensorRead = 0;
const unsigned long sensorReadInterval = 500;
// Fuzzy Logic Variabel
struct FuzzyResult {
  float bearingFault;
  float voltageDropFault;
  float suctionBlockFault;
  String primaryFault;
  float faultLevel;
};
FuzzyResult fuzzyResult;
// Fuzzy Logic Konstan
const float NORMAL_VOLTAGE = 220.0;
const float LOW_VOLTAGE_THRESHOLD = 190.0;
const float VERY_LOW_VOLTAGE_THRESHOLD = 160.0;
const float HIGH_TEMP_THRESHOLD = 60.0;
const float VERY_HIGH_TEMP_THRESHOLD = 80.0;
const float MEDIUM_VIBRATION_THRESHOLD = 1.0;
const float HIGH_VIBRATION_THRESHOLD = 1.5;
void setup() {
```

```
  Serial.begin(9600);
  // Setup PLX-DAQ
  Serial.println("CLEARDATA");
Serial.println("LABEL,Time,VoltageR(V),CurrentR(A),VoltageS(V),CurrentS(A),V
oltageT(V),CurrentT(A),Power(W),Energy(kWh),Frequency(Hz),PF,Temperature(C
),VibrationX,VibrationY,VibrationZ,TotalVibration,BearingFault,VoltageDropFault,
SuctionBlockFault,PrimaryFault,FaultLevel,ReferenceAmpere");
  // Inisiasi LCD
  lcd.init();
  lcd.backlight();
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Motor Monitoring");
  lcd.setCursor(0, 1);
  lcd.print("Initializing...");
  // Inisiasi tombol
  pinMode(BUTTON_PIN, INPUT_PULLUP);
  pinMode(SET_BUTTON_PIN, INPUT_PULLUP);
  // Inisiasi mpu lcd
  Wire.begin();
  mpu.initialize();
  // Cek koneksi MPU
  if (mpu.testConnection()) {
    Serial.println("MPU6050 connection successful");
  } else {
    Serial.println("MPU6050 connection failed");
  }
  // Delay PZEM
```

```
  delay(2000);
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("System Ready");
  delay(1000);
  lcd.clear();
}
void loop() {
  handleButton();
  handleSetButton();
  // Handle potensio
  handleAmpereSetting();
  // Interval sensor data
  if (millis() - lastSensorRead >= sensorReadInterval) {
    readAllSensorData();
    lastSensorRead = millis();
    sensorDataReady = true;
  }
  // Proses fuzzy ketika data sudah siap
  if (sensorDataReady) {
    processFuzzyLogic();
  }
  // Update display
  updateDisplay();
  // Log data ke Excel PLX-DAQ
  if (millis() - lastDataLogTime >= dataLogInterval && sensorDataReady) {
    logDataToExcel();
```

```
    lastDataLogTime = millis();
  }
  delay(100);
}
void handleButton() {
  if (digitalRead(BUTTON_PIN) == LOW && millis() - lastButtonPress > debounceTime) {
    lastButtonPress = millis();
    if (!isSettingMode) {
      currentMode = static_cast<DisplayMode>((currentMode + 1) % DISPLAY_MODES_COUNT);
      lcd.clear();
    }
  }
}
void handleSetButton() {
  if (digitalRead(SET_BUTTON_PIN) == LOW && millis() - lastSetButtonPress > debounceTime) {
    lastSetButtonPress = millis();
    if (!isSettingMode) {
      // Masuk setting mode
      isSettingMode = true;
      settingModeStartTime = millis();
      currentMode = DISPLAY_AMPERE_SETTING;
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Setting Mode");
      lcd.setCursor(0, 1);
```

```
      lcd.print("Adjust Potensio");
      delay(1000);
    } else {
      // Konfirmasi dan keluar set mode
      referenceAmpere = potensioValue;
      isSettingMode = false;
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Ampere Set:");
      lcd.setCursor(0, 1);
      lcd.print(referenceAmpere, 2);
      lcd.print(" A");
      delay(2000);
      lcd.clear();
    }
  }
}
void handleAmpereSetting() {
  if (isSettingMode) {
    int potValue = analogRead(POTENSIO_PIN);
    potensioValue = map(potValue, 0, 1023, 0, 5000) / 100.0; // 0.00 - 50.00 A
    if (millis() - settingModeStartTime > settingModeTimeout) {
      isSettingMode = false;
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Setting Timeout");
      delay(1000);
```

```
      lcd.clear();
    }
  }
}
void readAllSensorData() {
  readPZEMData();
  readVibrationData();
  readTemperatureData();
}
void readPZEMData() {
  // Read Phase R
  float tempVoltage = pzem1.voltage();
  if (!isnan(tempVoltage) && tempVoltage > 0) {
    voltageR = tempVoltage;
  }
  float tempCurrent = pzem1.current();
  if (!isnan(tempCurrent) && tempCurrent >= 0) {
    currentR = tempCurrent;
  }
  float tempPower = pzem1.power();
  if (!isnan(tempPower) && tempPower >= 0) {
    powerR = tempPower;
  }
  float tempFreq = pzem1.frequency();
  if (!isnan(tempFreq) && tempFreq > 0) {
    frequencyR = tempFreq;
  }
```

```
delay(50);
// Read Phase S
tempVoltage = pzem2.voltage();
if (!isnan(tempVoltage) && tempVoltage > 0) {
  voltageS = tempVoltage;
}
tempCurrent = pzem2.current();
if (!isnan(tempCurrent) && tempCurrent >= 0) {
  currentS = tempCurrent;
}
tempPower = pzem2.power();
if (!isnan(tempPower) && tempPower >= 0) {
  powerS = tempPower;
}
tempFreq = pzem2.frequency();
if (!isnan(tempFreq) && tempFreq > 0) {
  frequencyS = tempFreq;
}
delay(50);
// Read Phase T
tempVoltage = pzem3.voltage();
if (!isnan(tempVoltage) && tempVoltage > 0) {
  voltageT = tempVoltage;
}
tempCurrent = pzem3.current();
if (!isnan(tempCurrent) && tempCurrent >= 0) {
  currentT = tempCurrent;
```

```
  }
  tempPower = pzem3.power();
  if (!isnan(tempPower) && tempPower >= 0) {
    powerT = tempPower;
  }
  tempFreq = pzem3.frequency();
  if (!isnan(tempFreq) && tempFreq > 0) {
    frequencyT = tempFreq;
  }
  delay(50);
  float tempEnergy = pzem1.energy();
  if (!isnan(tempEnergy) && tempEnergy >= 0) {
    energy = tempEnergy;
  }
  float tempPF = pzem1.pf();
  if (!isnan(tempPF) && tempPF >= 0) {
    powerFactor = tempPF;
  }
}
void readVibrationData() {
  if (mpu.testConnection()) {
    int16_t ax, ay, az;
    mpu.getAcceleration(&ax, &ay, &az);
    // Convert ke m/s atau g
    vibrationX = ax / 16384.0;
    vibrationY = ay / 16384.0;
    vibrationZ = az / 16384.0;
```

```
    // Kalkulasi
    totalVibration = sqrt(vibrationX*vibrationX + vibrationY*vibrationY + vibrationZ*vibrationZ);
  }
}
void readTemperatureData() {
  float tempReading = thermocouple.readCelsius();
  if (!isnan(tempReading) && tempReading > -100 && tempReading < 1000) {
    temperature = tempReading;
  }
}
// Implementasi fuzzy
float triangularMF(float x, float a, float b, float c) {
  if (x <= a || x >= c) return 0.0;
  if (x == b) return 1.0;
  if (x < b) return (x - a) / (b - a);
  return (c - x) / (c - b);
}
float trapezoidalMF(float x, float a, float b, float c, float d) {
  if (x <= a || x >= d) return 0.0;
  if (x >= b && x <= c) return 1.0;
  if (x < b) return (x - a) / (b - a);
  return (d - x) / (d - c);
}
float gradedMF(float x, float low, float high) {
  if (x <= low) return 0.0;
  if (x >= high) return 1.0;
```

```
  return (x - low) / (high - low);
}
void processFuzzyLogic() {
  float avgVoltage = (voltageR + voltageS + voltageT) / 3.0;
  float avgCurrent = (currentR + currentS + currentT) / 3.0;
  float currentRatio = avgCurrent / referenceAmpere;
  // VOLTAGE FUZZY
  float voltageNormal = trapezoidalMF(avgVoltage, 200, 210, 240, 250);
  float voltageLow = triangularMF(avgVoltage, 150, 190, 210);
  float voltageVeryLow = trapezoidalMF(avgVoltage, 0, 0, 150, 190);
  // CURRENT FUZZY
  float currentNormal = trapezoidalMF(currentRatio, 0, 0, 0.6, 1.0);
  float currentHigh = triangularMF(currentRatio, 0.8, 1.1, 1.3);
  float currentVeryHigh = gradedMF(currentRatio, 1.2, 2.5);
  // TEMPERATURE FUZZY
  float tempNormal = trapezoidalMF(temperature, 0, 0, 30, 38);
  float tempHigh = triangularMF(temperature, 35, 45, 60);
  float tempVeryHigh = gradedMF(temperature, 55, 90);
  // VIBRATION FUZZY
  float vibLow = trapezoidalMF(totalVibration, 0, 0, 0.5, 1.1);
  float vibMedium = triangularMF(totalVibration, 0.3, 1.0, 1.0);
  float vibHigh = gradedMF(totalVibration, 0.8, 1.6);
  // FUZZY RULES VOLTAGE DROP
  float voltageRule1 = voltageVeryLow;
  float voltageRule2 = min(voltageLow, currentVeryHigh);
  float voltageRule3 = min(voltageLow, currentHigh);
```

```
  fuzzyResult.voltageDropFault = max(max(voltageRule1, voltageRule2), voltageRule3);
  // FUZZY RULES BEARING FAULT
  float bearingRule1 = min(vibHigh, tempHigh);
  float bearingRule2 = min(vibMedium, tempHigh);
  float bearingRule3 = min(vibHigh, tempVeryHigh);
  float bearingRule4 = vibHigh;
  fuzzyResult.bearingFault = max(max(max(bearingRule1, bearingRule2), bearingRule3), bearingRule4);
  // FUZZY RULES SUCTION BLOCK
  float suctionRule1 = min(currentVeryHigh, tempVeryHigh);
  float suctionRule2 = min(currentHigh, tempHigh);
  float suctionRule3 = min(currentHigh, voltageNormal);
  float suctionRule4 = min(currentVeryHigh, voltageLow);
  fuzzyResult.suctionBlockFault = max(max(max(suctionRule1, suctionRule2), suctionRule3), suctionRule4);
  // DEFUZZIFICATION
  float faultThreshold = 0.15;
  // Tentukan kerusakan
  float maxFault = max(max(fuzzyResult.bearingFault, fuzzyResult.voltageDropFault), fuzzyResult.suctionBlockFault);

  if (maxFault < faultThreshold) {
    fuzzyResult.primaryFault = "NORMAL";
    fuzzyResult.faultLevel = 0.0;
  }
  else if (fuzzyResult.suctionBlockFault == maxFault && fuzzyResult.suctionBlockFault >= faultThreshold) {
    fuzzyResult.primaryFault = "SUCTION_BLOCK";
```

```
    fuzzyResult.faultLevel = fuzzyResult.suctionBlockFault;
  }
  else if (fuzzyResult.bearingFault == maxFault && fuzzyResult.bearingFault >=
faultThreshold) {
    fuzzyResult.primaryFault = "BEARING";
    fuzzyResult.faultLevel = fuzzyResult.bearingFault;
  }
  else if (fuzzyResult.voltageDropFault == maxFault &&
fuzzyResult.voltageDropFault >= faultThreshold) {
    fuzzyResult.primaryFault = "VOLTAGE_DROP";
    fuzzyResult.faultLevel = fuzzyResult.voltageDropFault;
  }
  else {
    fuzzyResult.primaryFault = "NORMAL";
    fuzzyResult.faultLevel = 0.0;
  }
}
void updateDisplay() {
  switch(currentMode) {
    case DISPLAY_PHASE_R:
      displayPhaseData("R", voltageR, currentR, powerR, frequencyR);
      break;
    case DISPLAY_PHASE_S:
      displayPhaseData("S", voltageS, currentS, powerS, frequencyS);
      break;
    case DISPLAY_PHASE_T:
      displayPhaseData("T", voltageT, currentT, powerT, frequencyT);
      break;
```

```
case DISPLAY_POWER_FACTOR_ENERGY:
  lcd.setCursor(0, 0);
  lcd.print("PF:");
  lcd.print(powerFactor, 2);
  lcd.setCursor(0, 1);
  lcd.print("E:");
  lcd.print(energy, 1);
  lcd.print("kWh");
  break;
case DISPLAY_TEMPERATURE:
  lcd.setCursor(0, 0);
  lcd.print("Temperature     ");
  lcd.setCursor(0, 1);
  lcd.print(temperature, 1);
  lcd.print(" C            ");
  break;
case DISPLAY_VIBRATION:
  lcd.setCursor(0, 0);
  lcd.print("Vibration       ");
  lcd.setCursor(0, 1);
  lcd.print("Tot: ");
  lcd.print(totalVibration, 2);
  lcd.print("g     ");
  break;
case DISPLAY_FUZZY_STATUS:
  lcd.setCursor(0, 0);
  lcd.print(fuzzyResult.primaryFault);
```

```
      // Clear remaining characters in first row
      for(int i = fuzzyResult.primaryFault.length(); i < 16; i++) {
        lcd.print(" ");
      }
      lcd.setCursor(0, 1);
      lcd.print("Level: ");
      lcd.print(fuzzyResult.faultLevel, 2);
      lcd.print("      ");
      break;
    case DISPLAY_AMPERE_SETTING:
      if (isSettingMode) {
        lcd.setCursor(0, 0);
        lcd.print("Set Ref Ampere: ");
        lcd.setCursor(0, 1);
        lcd.print(potensioValue, 2);
        lcd.print(" A          ");
      } else {
        lcd.setCursor(0, 0);
        lcd.print("Ref Ampere:     ");
        lcd.setCursor(0, 1);
        lcd.print(referenceAmpere, 2);
        lcd.print(" A          ");
      }
      break;
  }
}
```

```
void displayPhaseData(String phase, float voltage, float current, float power, float frequency) {
  lcd.setCursor(0, 0);
  lcd.print("V" + phase + ":");
  lcd.print(voltage, 1);
  lcd.print("V I:");
  lcd.print(current, 2);
  lcd.print("A");
  lcd.setCursor(0, 1);
  lcd.print("P:");
  lcd.print(power, 1);
  lcd.print("W F:");
  lcd.print(frequency, 1);
  lcd.print("Hz");
}
void logDataToExcel() {
  Serial.print("DATA,TIME,");
  Serial.print(voltageR);
  Serial.print(",");
  Serial.print(currentR);
  Serial.print(",");
  Serial.print(voltageS);
  Serial.print(",");
  Serial.print(currentS);
  Serial.print(",");
  Serial.print(voltageT);
  Serial.print(",");
```

```
  Serial.print(currentT);
  Serial.print(",");
  Serial.print(powerR + powerS + powerT);
  Serial.print(",");
  Serial.print(energy);
  Serial.print(",");
  Serial.print(frequencyR);
  Serial.print(",");
  Serial.print(powerFactor);
  Serial.print(",");
  Serial.print(temperature);
  Serial.print(",");
  Serial.print(vibrationX);
  Serial.print(",");
  Serial.print(vibrationY);
  Serial.print(",");
  Serial.print(vibrationZ);
  Serial.print(",");
  Serial.print(totalVibration);
  Serial.print(",");
  Serial.print(fuzzyResult.bearingFault);
  Serial.print(",");
  Serial.print(fuzzyResult.voltageDropFault);
  Serial.print(",");
  Serial.print(fuzzyResult.suctionBlockFault);
  Serial.print(",");
  Serial.print(fuzzyResult.primaryFault);
```

```
  Serial.print(",");
  Serial.print(fuzzyResult.faultLevel);
  Serial.print(",");
  Serial.println(referenceAmpere);
}
```