LAMPIRAN

Lampiran A *Listing Code*

```
#include <EEPROM.h>
#include "Sensor_TDS.h"
#include "fis_header.h"
#include <WiFi.h>
#include <FirebaseESP32.h>
#define WIFI_SSID "kepo1"
#define WIFI_PASSWORD "kepoajalu"
#define      FIREBASE_HOST      "https://akuaponik-59715-default-
rtdb.firebaseio.com"
#define FIREBASE_KEY "f9sSCQ5bVxssszmNhYPkFrN9Oo7f8MfFQtmqS7ey"
#define TDS_SENSOR_PIN 32
#define PH_SENSOR_PIN 33              //pH meter Analog output to
Arduino Analog Input 0
#define Offset 2.2            //deviation compensate
#define LED 13
#define ArrayLenth  40    //times of collection
int pHArray[ArrayLenth];   //Store the average value of the sensor
feedback
int pHArrayIndex = 0;
float setPointpH = 6;
float errorpH;
float deltaerrorpH;
float errorpHsebelum;
float setPointtds = 900;
float errortds;
float deltaerrortds;
float errortdssebelum;
float f_pH;
float f_tdss;
float Input_PH;
float Input_TDS;
float temperature = 25, tdsValue = 0;
float read_ph;
float read_tds;


Sensor_TDS tds_sensor;
FirebaseData firebaseData;
FirebaseJson firebaseJson;
int Relay1 = 3;                                //Pompa pH up
int Relay2 = 21;                                //Pompa pH down
int Relay3 = 19;                                //Pompa tds up
int Relay4 = 18;                                //Pompa tds down
unsigned long ulangi;
int waktu = 20000;
// Number of inputs to the Fuzzy inference system
const int fis_gcI = 4;
// Number of outputs to the Fuzzy inference system
const int fis_gcO = 2;
// Number of rules to the Fuzzy inference system
const int fis_gcR = 50;
FIS_TYPE g_fisInput[fis_gcI];
FIS_TYPE g_fisOutput[fis_gcO];
// Setup routine runs once when you press reset:
void setup()
```

```
{
  Serial.begin(9600);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  Serial.print("Connecting to Wi-Fi");
  while (WiFi.status() != WL_CONNECTED)
  {
    Serial.print(".");
    delay(300);
  }
  Serial.println();
  Serial.print("Connected with IP: ");
  Serial.println(WiFi.localIP());
  Serial.println();
  Firebase.begin(FIREBASE_HOST, FIREBASE_KEY);
  Firebase.reconnectWiFi(true);

  //Set database read timeout to 1 minute (max 15 minutes)
  Firebase.setReadTimeout(firebaseData, 1000 * 60);
  //tiny, small, medium, large and unlimited.
  //Size and its write timeout e.g. tiny (1s), small (10s), medium
(30s) and large (60s).
  Firebase.setwriteSizeLimit(firebaseData, "small");
  Serial.println("Firebase Connected.");
  tds_sensor.setPin(TDS_SENSOR_PIN);
  tds_sensor.setAref(3.3);
  tds_sensor.setAdcRange(4096);
  tds_sensor.setKvalueAddress(16);
  tds_sensor.begin();
  pinMode(33, INPUT);
  pinMode(Relay1, OUTPUT);
  pinMode(Relay2, OUTPUT);
  pinMode(Relay3, OUTPUT);
  pinMode(Relay4, OUTPUT);
}
void hitungeror()
{
  //Rumus Fuzzy pH
  Serial.println("=======================");
  Serial.print("pH = ");
  Serial.println(Input_PH);
  errorpH = setPointpH - Input_PH;
  Serial.print("errorpH = ");
  Serial.println(errorpH);
  deltaerrorpH = errorpH - errorpHsebelum;
  Serial.print("deltaerrorpH = ");
  Serial.println(deltaerrorpH);
  errorpHsebelum = errorpH;
  Serial.print("errorpHsebelum = ");
  Serial.println(errorpHsebelum);
  Serial.println("=======================");
  //Rumus Fuzzy tds
  Serial.print("tds = ");
  Serial.println(Input_TDS);
  errortds = setPointtds - Input_TDS;
  Serial.print("errortds = ");
  Serial.println(errortds);
  deltaerrortds = errortds - errortdssebelum;
  Serial.print("deltaerrortds = ");
  Serial.println(deltaerrortds);
```

```
  errortdssebelum = errortds;
  Serial.print("errortdssebelum = ");
  Serial.println(errortdssebelum);
  Serial.println("=======================");
  Serial.print("g_fisOutput[0]=  ");
  Serial.println(g_fisOutput[0]);
  Serial.print("g_fisOutput[1]=  ");
  Serial.println(g_fisOutput[1]);
}
/*
  void serial_Logger()
  {
  Serial.println(F("=== Input ==="));
    Serial.print(F("PH: "));
    Serial.print(Input_PH);
    Serial.print(F("\t"));
    Serial.print(F("TDS: "));
    Serial.println(Input_TDS);
    Serial.println(F("=== Output ==="));
    Serial.print(F("PH_Motor: "));
    Serial.print(Output_Motor_PH);
    Serial.print(F("\t"));
    Serial.print(F("TDS_Motor: "));
    Serial.println(Output_Motor_TDS);
    Serial.println();
    Serial.println(F("=== Status ==="));
    Serial.print(F("ErrorpH  : "));
    Serial.print(F(errorpH));
    Serial.print(F("\t\t| "));
    Serial.print(F("DeltaErrorpH: "));
    Serial.print(deltaerrorpH);
    Serial.print(F("ErrorTDS : "));
    Serial.print(F(errortds));
    Serial.print(F("\t| "));
    Serial.print(F("DeltaErrorTDS"));
    Serial.print(deltaerrortds);
    Serial.println();
    Serial.println();
  }
*/
// Loop routine runs over and over again forever:
void loop(void)
{
  unsigned long currentMillis = millis();
  if ((currentMillis - ulangi) > waktu) {
    ulangi = millis();
    static float pHValue, voltage;
    {
      pHArray[pHArrayIndex++] = analogRead(PH_SENSOR_PIN);
      if (pHArrayIndex == ArrayLenth)pHArrayIndex = 0;
      voltage = avergearray(pHArray, ArrayLenth) * 3.3 / 4096;
      pHValue = 3.5 * voltage + Offset;
      Input_PH = pHValue;
    }
    {
      tds_sensor.setTemperature(temperature);      //   set   the
temperature and execute temperature compensation
      tds_sensor.update();  //sample and calculate
      Input_TDS = tds_sensor.getTdsValue();
```

```
    }
    hitungeror();
    firebase_logger();
    keputusan();
  }
}
void keputusan ()
{
  // Read Input: ErrorPh
  g_fisInput[0] = errorpH;
  // Read Input: DeltaErrorPh
  g_fisInput[1] = deltaerrorpH;
  // Read Input: ErrorTds
  g_fisInput[2] = errortds;
  // Read Input: DeltaErrorTds
  g_fisInput[3] = deltaerrortds;
  g_fisOutput[0] = 0;
  g_fisOutput[1] = 0;
  fis_evaluate();
  if ( (act(g_fisOutput[0]) == 0) && (act(g_fisOutput[1]) == 0)) {
    off(3);  //Semua RELAY Mati
    off(21);
    off(19);
    off(18);
  }
  if ( (act(g_fisOutput[0]) == 0) && (act(g_fisOutput[1]) == 1)) {
    off(3);
    off(21);
    on(19), delay(5000), off(19);
    off(18);
  }
  if ( (act(g_fisOutput[0]) == 0) && (act(g_fisOutput[1]) == 2)) {
    off(3);
    off(21);
    on(19), delay(7000), off(19);
    off(18);
  }
  if ( (act(g_fisOutput[0]) == 0) && (act(g_fisOutput[1]) == 3)) {
    off(3);
    off(21);
    off(19);
    on(18), delay(5000), off(18);
  }
  if ( (act(g_fisOutput[0]) == 0) && (act(g_fisOutput[1]) == 4)) {
    off(3);
    off(21);
    off(19);
    on(18), delay(7000), off(18);
  }

  if ( (act(g_fisOutput[0]) == 1) && (act(g_fisOutput[1]) == 0)) {
    on(3), delay(5000), off(3);
    off(21);
    off(19);
    off(18);
  }
  if ( (act(g_fisOutput[0]) == 1) && (act(g_fisOutput[1]) == 1)) {
    on(3), delay(5000), off(3);
    off(21);
```

```
  on(19), delay(5000), off(19);
  off(18);
}
if ( (act(g_fisOutput[0]) == 1) && (act(g_fisOutput[1]) == 2)) {
  on(3), delay(5000), off(3);
  off(21);
  on(19), delay(7000), off(19);
  off(18);
}
if ( (act(g_fisOutput[0]) == 1) && (act(g_fisOutput[1]) == 3)) {
  on(3), delay(5000), off(3);
  off(21);
  off(19);
  on(18), delay(5000), off(18);
}
if ( (act(g_fisOutput[0]) == 1) && (act(g_fisOutput[1]) == 4)) {
  on(3), delay(5000), off(3);
  off(21);
  off(19);
  on(18), delay(7000), off(18);
}
if ( (act(g_fisOutput[0]) == 2) && (act(g_fisOutput[1]) == 0)) {
  on(3), delay(7000), off(3);
  off(21);
  off(19);
  off(18);
}
if ( (act(g_fisOutput[0]) == 2) && (act(g_fisOutput[1]) == 1)) {
  on(3), delay(7000), off(3);
  off(21);
  on(19), delay(5000), off(19);
  off(18);
}
if ( (act(g_fisOutput[0]) == 2) && (act(g_fisOutput[1]) == 2)) {
  on(3), delay(7000), off(3);
  off(21);
  on(19), delay(7000), off(19);
  off(18);
}
if ( (act(g_fisOutput[0]) == 2) && (act(g_fisOutput[1]) == 3)) {
  on(3), delay(7000), off(3);
  off(21);
  off(19);
  on(18), delay(5000), off(18);
}
if ( (act(g_fisOutput[0]) == 2) && (act(g_fisOutput[1]) == 4)) {
  on(3), delay(7000), off(3);
  off(21);
  off(19);
  on(18), delay(7000), off(18);
}
if ( (act(g_fisOutput[0]) == 3) && (act(g_fisOutput[1]) == 0)) {
  off(3);
  on(21), delay(5000), off(21);
  off(19);
  off(18);
}
if ( (act(g_fisOutput[0]) == 3) && (act(g_fisOutput[1]) == 1)) {
  off(3);
```

```
    on(21), delay(5000), off(21);
    on(19), delay(5000), off(19);
    off(18);
  }
  if ( (act(g_fisOutput[0]) == 3) && (act(g_fisOutput[1]) == 2)) {
    off(3);
    on(21), delay(5000), off(21);
    on(19), delay(7000), off(19);
    off(18);
  }
  if ( (act(g_fisOutput[0]) == 3) && (act(g_fisOutput[1]) == 3)) {
    off(3);
    on(21), delay(5000), off(21);
    off(19);
    on(18), delay(5000), off(18);
  }
  if ( (act(g_fisOutput[0]) == 3) && (act(g_fisOutput[1]) == 4)) {
    off(3);
    on(21), delay(5000), off(21);
    off(19);
    on(18), delay(7000), off(18);
  }
  if ( (act(g_fisOutput[0]) == 4) && (act(g_fisOutput[1]) == 0)) {
    off(3);
    on(21), delay(7000), off(21);
    off(19);
    off(18);
  }
  if ( (act(g_fisOutput[0]) == 4) && (act(g_fisOutput[1]) == 1)) {
    off(3);
    on(21), delay(7000), off(21);
    on(19), delay(5000), off(19);
    off(18);
  }
  if ( (act(g_fisOutput[0]) == 4) && (act(g_fisOutput[1]) == 2)) {
    off(3);
    on(21), delay(7000), off(21);
    on(19), delay(7000), off(19);
    off(18);
  }
  if ( (act(g_fisOutput[0]) == 4) && (act(g_fisOutput[1]) == 3)) {
    off(3);
    on(21), delay(7000), off(21);
    off(19);
    on(18), delay(5000), off(18);
  }
  if ( (act(g_fisOutput[0]) == 4) && (act(g_fisOutput[1]) == 4)) {
    off(3);
    on(21), delay(7000), off(21);
    off(19);
    on(18), delay(7000), off(18);
  }
}
void on(int p)  {
  digitalWrite(p, LOW);
}
void off(int p) {
  digitalWrite(p, HIGH);
}
```

```
bool nol(float n) {
  if (n < 0.001 && n > -0.001) return true;
  else false;
}
int act(float n)
{
  // { 0 = MATI | 1 = Relay 3&4 | 2 = Relay 5&6 }
if(n >= -0.1 && n <= 0.1) return 0;
if(n <= -0.5) return 2;
if(n >= -0.5 && n <= -0.1) return 1;
if(n >= 0.1 && n <= 0.5) return 3;
if(n >= 0.5) return 4;
}
//***************************************************************
*******
// Support functions for Fuzzy Inference System
//***************************************************************
*******
// Trapezoidal Member Function
FIS_TYPE fis_trapmf(FIS_TYPE x, FIS_TYPE* p)
{
    FIS_TYPE a = p[0], b = p[1], c = p[2], d = p[3];
    FIS_TYPE t1 = ((x <= c) ? 1 : ((d < x) ? 0 : ((c != d) ? ((d -
x) / (d - c)) : 0)));
    FIS_TYPE t2 = ((b <= x) ? 1 : ((x < a) ? 0 : ((a != b) ? ((x -
a) / (b - a)) : 0)));
    return (FIS_TYPE) min(t1, t2);
}
FIS_TYPE fis_min(FIS_TYPE a, FIS_TYPE b)
{
    return min(a, b);
}
FIS_TYPE fis_max(FIS_TYPE a, FIS_TYPE b)
{
    return max(a, b);
}
FIS_TYPE    fis_array_operation(FIS_TYPE    *array,    int    size,
_FIS_ARR_OP pfnOp)
{
    int i;
    FIS_TYPE ret = 0;

    if (size == 0) return ret;
    if (size == 1) return array[0];

    ret = array[0];
    for (i = 1; i < size; i++)
    {
        ret = (*pfnOp)(ret, array[i]);
    }
    return ret;
}
//***************************************************************
*******
// Data for Fuzzy Inference System
//***************************************************************
*******
// Pointers to the implementations of member functions
_FIS_MF fis_gMF[] =
```

```
{
    fis_trapmf
};
// Count of member function for each Input
int fis_gIMFCount[] = { 5, 5, 5, 5 };
// Count of member function for each Output
int fis_gOMFCount[] = { 5, 5 };
// Coefficients for the Input Member Functions
FIS_TYPE fis_gMFI0Coeff1[] = { -4, -3, -1, 0 };
FIS_TYPE fis_gMFI0Coeff2[] = { -1, 0, 0, 1 };
FIS_TYPE fis_gMFI0Coeff3[] = { 3, 4, 6, 6 };
FIS_TYPE fis_gMFI0Coeff4[] = { -6, -6, -4, -3 };
FIS_TYPE fis_gMFI0Coeff5[] = { 0, 1, 3, 4 };
FIS_TYPE* fis_gMFI0Coeff[] = { fis_gMFI0Coeff1, fis_gMFI0Coeff2,
fis_gMFI0Coeff3, fis_gMFI0Coeff4, fis_gMFI0Coeff5 };
FIS_TYPE fis_gMFI1Coeff1[] = { -1, -1, -0.6, -0.4 };
FIS_TYPE fis_gMFI1Coeff2[] = { -0.1, 0, 0, 0.1 };
FIS_TYPE fis_gMFI1Coeff3[] = { 0.4, 0.6, 1, 1 };
FIS_TYPE fis_gMFI1Coeff4[] = { -0.5947, -0.3947, -0.1947, 0.005291
};
FIS_TYPE fis_gMFI1Coeff5[] = { 0, 0.2, 0.4, 0.6 };
FIS_TYPE* fis_gMFI1Coeff[] = { fis_gMFI1Coeff1, fis_gMFI1Coeff2,
fis_gMFI1Coeff3, fis_gMFI1Coeff4, fis_gMFI1Coeff5 };
FIS_TYPE fis_gMFI2Coeff1[] = { -1100, -1100, -733.3, -488.9 };
FIS_TYPE fis_gMFI2Coeff2[] = { -122.2, 0, 0, 122.2 };
FIS_TYPE fis_gMFI2Coeff3[] = { 488.9, 733.3, 1100, 1100 };
FIS_TYPE fis_gMFI2Coeff4[] = { -733.3, -488.9, -244.4, 0 };
FIS_TYPE fis_gMFI2Coeff5[] = { 0, 244.4, 488.9, 733.3 };
FIS_TYPE* fis_gMFI2Coeff[] = { fis_gMFI2Coeff1, fis_gMFI2Coeff2,
fis_gMFI2Coeff3, fis_gMFI2Coeff4, fis_gMFI2Coeff5 };
FIS_TYPE fis_gMFI3Coeff1[] = { -100, -100, -60, -40 };
FIS_TYPE fis_gMFI3Coeff2[] = { -10, 0, 0, 10 };
FIS_TYPE fis_gMFI3Coeff3[] = { 40, 60, 100, 100 };
FIS_TYPE fis_gMFI3Coeff4[] = { -60, -40, -20, 0 };
FIS_TYPE fis_gMFI3Coeff5[] = { 0, 20, 40, 60 };
FIS_TYPE* fis_gMFI3Coeff[] = { fis_gMFI3Coeff1, fis_gMFI3Coeff2,
fis_gMFI3Coeff3, fis_gMFI3Coeff4, fis_gMFI3Coeff5 };
FIS_TYPE** fis_gMFICoeff[] = { fis_gMFI0Coeff, fis_gMFI1Coeff,
fis_gMFI2Coeff, fis_gMFI3Coeff };
// Coefficients for the Output Member Functions
FIS_TYPE fis_gMFO0Coeff1[] = { -1, -1, -0.6, -0.5 };
FIS_TYPE fis_gMFO0Coeff2[] = { -0.1, 0, 0, 0.1 };
FIS_TYPE fis_gMFO0Coeff3[] = { 0.5, 0.6, 1, 1 };
FIS_TYPE fis_gMFO0Coeff4[] = { -0.5, -0.4, -0.2, -0.1 };
FIS_TYPE fis_gMFO0Coeff5[] = { 0.1, 0.2, 0.4, 0.5 };
FIS_TYPE* fis_gMFO0Coeff[] = { fis_gMFO0Coeff1, fis_gMFO0Coeff2,
fis_gMFO0Coeff3, fis_gMFO0Coeff4, fis_gMFO0Coeff5 };
FIS_TYPE fis_gMFO1Coeff1[] = { -1, -1, -0.6, -0.5 };
FIS_TYPE fis_gMFO1Coeff2[] = { -0.1, 0, 0, 0.1 };
FIS_TYPE fis_gMFO1Coeff3[] = { 0.5, 0.6, 1, 1 };
FIS_TYPE fis_gMFO1Coeff4[] = { -0.5, -0.4, -0.2, -0.1 };
FIS_TYPE fis_gMFO1Coeff5[] = { 0.1, 0.2, 0.4, 0.5 };
FIS_TYPE* fis_gMFO1Coeff[] = { fis_gMFO1Coeff1, fis_gMFO1Coeff2,
fis_gMFO1Coeff3, fis_gMFO1Coeff4, fis_gMFO1Coeff5 };
FIS_TYPE** fis_gMFOCoeff[] = { fis_gMFO0Coeff, fis_gMFO1Coeff };
// Input membership function set
int fis_gMFI0[] = { 0, 0, 0, 0, 0 };
int fis_gMFI1[] = { 0, 0, 0, 0, 0 };
int fis_gMFI2[] = { 0, 0, 0, 0, 0 };
```

```
int fis_gMFI3[] = { 0, 0, 0, 0, 0 };
int* fis_gMFI[] = { fis_gMFI0, fis_gMFI1, fis_gMFI2, fis_gMFI3};
// Output membership function set
int fis_gMFO0[] = { 0, 0, 0, 0, 0 };
int fis_gMFO1[] = { 0, 0, 0, 0, 0 };
int* fis_gMFO[] = { fis_gMFO0, fis_gMFO1};
// Rule Weights
FIS_TYPE fis_gRWeight[] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };
// Rule Type
int fis_gRType[] = { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };
// Rule Inputs
int fis_gRI0[] = { 4, 1, 0, 0 };
int fis_gRI1[] = { 4, 4, 0, 0 };
int fis_gRI2[] = { 4, 2, 0, 0 };
int fis_gRI3[] = { 4, 3, 0, 0 };
int fis_gRI4[] = { 4, 5, 0, 0 };
int fis_gRI5[] = { 1, 1, 0, 0 };
int fis_gRI6[] = { 1, 4, 0, 0 };
int fis_gRI7[] = { 1, 2, 0, 0 };
int fis_gRI8[] = { 1, 3, 0, 0 };
int fis_gRI9[] = { 1, 5, 0, 0 };
int fis_gRI10[] = { 2, 1, 0, 0 };
int fis_gRI11[] = { 2, 4, 0, 0 };
int fis_gRI12[] = { 2, 2, 0, 0 };
int fis_gRI13[] = { 2, 3, 0, 0 };
int fis_gRI14[] = { 2, 5, 0, 0 };
int fis_gRI15[] = { 5, 1, 0, 0 };
int fis_gRI16[] = { 5, 4, 0, 0 };
int fis_gRI17[] = { 5, 2, 0, 0 };
int fis_gRI18[] = { 5, 3, 0, 0 };
int fis_gRI19[] = { 5, 5, 0, 0 };
int fis_gRI20[] = { 3, 1, 0, 0 };
int fis_gRI21[] = { 3, 4, 0, 0 };
int fis_gRI22[] = { 3, 2, 0, 0 };
int fis_gRI23[] = { 3, 3, 0, 0 };
int fis_gRI24[] = { 3, 5, 0, 0 };
int fis_gRI25[] = { 0, 0, 1, 1 };
int fis_gRI26[] = { 0, 0, 1, 4 };
int fis_gRI27[] = { 0, 0, 1, 2 };
int fis_gRI28[] = { 0, 0, 1, 3 };
int fis_gRI29[] = { 0, 0, 1, 5 };
int fis_gRI30[] = { 0, 0, 4, 1 };
int fis_gRI31[] = { 0, 0, 4, 4 };
int fis_gRI32[] = { 0, 0, 4, 2 };
int fis_gRI33[] = { 0, 0, 4, 3 };
int fis_gRI34[] = { 0, 0, 4, 5 };
int fis_gRI35[] = { 0, 0, 2, 1 };
int fis_gRI36[] = { 0, 0, 2, 4 };
int fis_gRI37[] = { 0, 0, 2, 2 };
int fis_gRI38[] = { 0, 0, 2, 3 };
int fis_gRI39[] = { 0, 0, 2, 5 };
int fis_gRI40[] = { 0, 0, 5, 1 };
int fis_gRI41[] = { 0, 0, 5, 4 };
int fis_gRI42[] = { 0, 0, 5, 2 };
int fis_gRI43[] = { 0, 0, 5, 3 };
```

```
int fis_gRI44[] = { 0, 0, 5, 5 };
int fis_gRI45[] = { 0, 0, 3, 1 };
int fis_gRI46[] = { 0, 0, 3, 4 };
int fis_gRI47[] = { 0, 0, 3, 2 };
int fis_gRI48[] = { 0, 0, 3, 5 };
int fis_gRI49[] = { 0, 0, 3, 3 };
int*  fis_gRI[]  =  {  fis_gRI0,  fis_gRI1,  fis_gRI2,  fis_gRI3,
fis_gRI4,  fis_gRI5,  fis_gRI6,  fis_gRI7,  fis_gRI8,  fis_gRI9,
fis_gRI10, fis_gRI11, fis_gRI12, fis_gRI13, fis_gRI14, fis_gRI15,
fis_gRI16, fis_gRI17, fis_gRI18, fis_gRI19, fis_gRI20, fis_gRI21,
fis_gRI22, fis_gRI23, fis_gRI24, fis_gRI25, fis_gRI26, fis_gRI27,
fis_gRI28, fis_gRI29, fis_gRI30, fis_gRI31, fis_gRI32, fis_gRI33,
fis_gRI34, fis_gRI35, fis_gRI36, fis_gRI37, fis_gRI38, fis_gRI39,
fis_gRI40, fis_gRI41, fis_gRI42, fis_gRI43, fis_gRI44, fis_gRI45,
fis_gRI46, fis_gRI47, fis_gRI48, fis_gRI49 };
// Rule Outputs
int fis_gRO0[] = { 1, 0 };
int fis_gRO1[] = { 1, 0 };
int fis_gRO2[] = { 1, 0 };
int fis_gRO3[] = { 1, 0 };
int fis_gRO4[] = { 1, 0 };
int fis_gRO5[] = { 4, 0 };
int fis_gRO6[] = { 4, 0 };
int fis_gRO7[] = { 4, 0 };
int fis_gRO8[] = { 4, 0 };
int fis_gRO9[] = { 4, 0 };
int fis_gRO10[] = { 2, 0 };
int fis_gRO11[] = { 2, 0 };
int fis_gRO12[] = { 2, 0 };
int fis_gRO13[] = { 2, 0 };
int fis_gRO14[] = { 2, 0 };
int fis_gRO15[] = { 5, 0 };
int fis_gRO16[] = { 5, 0 };
int fis_gRO17[] = { 5, 0 };
int fis_gRO18[] = { 5, 0 };
int fis_gRO19[] = { 5, 0 };
int fis_gRO20[] = { 3, 0 };
int fis_gRO21[] = { 3, 0 };
int fis_gRO22[] = { 3, 0 };
int fis_gRO23[] = { 3, 0 };
int fis_gRO24[] = { 3, 0 };
int fis_gRO25[] = { 0, 1 };
int fis_gRO26[] = { 0, 1 };
int fis_gRO27[] = { 0, 1 };
int fis_gRO28[] = { 0, 1 };
int fis_gRO29[] = { 0, 1 };
int fis_gRO30[] = { 0, 4 };
int fis_gRO31[] = { 0, 4 };
int fis_gRO32[] = { 0, 4 };
int fis_gRO33[] = { 0, 4 };
int fis_gRO34[] = { 0, 4 };
int fis_gRO35[] = { 0, 2 };
int fis_gRO36[] = { 0, 2 };
int fis_gRO37[] = { 0, 2 };
int fis_gRO38[] = { 0, 2 };
int fis_gRO39[] = { 0, 2 };
int fis_gRO40[] = { 0, 5 };
int fis_gRO41[] = { 0, 5 };
int fis_gRO42[] = { 0, 5 };
```

```
int fis_gRO43[] = { 0, 5 };
int fis_gRO44[] = { 0, 5 };
int fis_gRO45[] = { 0, 3 };
int fis_gRO46[] = { 0, 3 };
int fis_gRO47[] = { 0, 3 };
int fis_gRO48[] = { 0, 3 };
int fis_gRO49[] = { 0, 3 };
int*  fis_gRO[]  =  {  fis_gRO0,  fis_gRO1,  fis_gRO2,  fis_gRO3,
fis_gRO4,  fis_gRO5,  fis_gRO6,  fis_gRO7,  fis_gRO8,  fis_gRO9,
fis_gRO10, fis_gRO11, fis_gRO12, fis_gRO13, fis_gRO14, fis_gRO15,
fis_gRO16, fis_gRO17, fis_gRO18, fis_gRO19, fis_gRO20, fis_gRO21,
fis_gRO22, fis_gRO23, fis_gRO24, fis_gRO25, fis_gRO26, fis_gRO27,
fis_gRO28, fis_gRO29, fis_gRO30, fis_gRO31, fis_gRO32, fis_gRO33,
fis_gRO34, fis_gRO35, fis_gRO36, fis_gRO37, fis_gRO38, fis_gRO39,
fis_gRO40, fis_gRO41, fis_gRO42, fis_gRO43, fis_gRO44, fis_gRO45,
fis_gRO46, fis_gRO47, fis_gRO48, fis_gRO49 };
// Input range Min
FIS_TYPE fis_gIMin[] = { -6, -1, -1100, -100 };
// Input range Max
FIS_TYPE fis_gIMax[] = { 6, 1, 1100, 100 };
// Output range Min
FIS_TYPE fis_gOMin[] = { -1, -1 };
// Output range Max
FIS_TYPE fis_gOMax[] = { 1, 1 };
//************************************************************
*******
// Data dependent support functions for Fuzzy Inference System
//************************************************************
*******
FIS_TYPE fis_MF_out(FIS_TYPE** FuzzyRuleSet, FIS_TYPE x, int o)
{
    FIS_TYPE mfOut;
    int r;
    for (r = 0; r < fis_gcR; ++r)
    {
        int index = fis_gRO[r][o];
        if (index > 0)
        {
            index = index - 1;
            mfOut        =        (fis_gMF[fis_gMFO[o][index]])(x,
fis_gMFOCoeff[o][index]);
        }
        else if (index < 0)
        {
            index = -index - 1;
            mfOut    =    1    -    (fis_gMF[fis_gMFO[o][index]])(x,
fis_gMFOCoeff[o][index]);
        }
        else
        {
            mfOut = 0;
        }
        FuzzyRuleSet[0][r] = fis_min(mfOut, FuzzyRuleSet[1][r]);
    }
    return fis_array_operation(FuzzyRuleSet[0], fis_gcR, fis_max);
}
FIS_TYPE fis_defuzz_centroid(FIS_TYPE** FuzzyRuleSet, int o)
{
```

```
    FIS_TYPE    step    =    (fis_gOMax[o]    -    fis_gOMin[o])    /
(FIS_RESOLUSION - 1);
    FIS_TYPE area = 0;
    FIS_TYPE momentum = 0;
    FIS_TYPE dist, slice;
    int i;
    // calculate the area under the curve formed by the MF outputs
    for (i = 0; i < FIS_RESOLUSION; ++i){
        dist = fis_gOMin[o] + (step * i);
        slice = step * fis_MF_out(FuzzyRuleSet, dist, o);
        area += slice;
        momentum += slice*dist;
    }
    return ((area == 0) ? ((fis_gOMax[o] + fis_gOMin[o]) / 2) :
(momentum / area));
}
//************************************************************
*******
// Fuzzy Inference System
//************************************************************
*******
void fis_evaluate()
{
    FIS_TYPE FuzzyInput0[] = { 0, 0, 0, 0, 0 };
    FIS_TYPE FuzzyInput1[] = { 0, 0, 0, 0, 0 };
    FIS_TYPE FuzzyInput2[] = { 0, 0, 0, 0, 0 };
    FIS_TYPE FuzzyInput3[] = { 0, 0, 0, 0, 0 };
    FIS_TYPE* FuzzyInput[fis_gcI] = { FuzzyInput0, FuzzyInput1,
FuzzyInput2, FuzzyInput3, };
    FIS_TYPE FuzzyOutput0[] = { 0, 0, 0, 0, 0 };
    FIS_TYPE FuzzyOutput1[] = { 0, 0, 0, 0, 0 };
    FIS_TYPE* FuzzyOutput[fis_gcO] = { FuzzyOutput0, FuzzyOutput1,
};
    FIS_TYPE FuzzyRules[fis_gcR] = { 0 };
    FIS_TYPE FuzzyFires[fis_gcR] = { 0 };
    FIS_TYPE* FuzzyRuleSet[] = { FuzzyRules, FuzzyFires };
    FIS_TYPE sW = 0;

    // Transforming input to Fuzzy Input
    int i, j, r, o;
    for (i = 0; i < fis_gcI; ++i)
    {
        for (j = 0; j < fis_gIMFCount[i]; ++j)
        {
            FuzzyInput[i][j] =
                (fis_gMF[fis_gMFI[i][j]])(g_fisInput[i],
fis_gMFICoeff[i][j]);
        }
    }
    int index = 0;
    for (r = 0; r < fis_gcR; ++r)
    {
        if (fis_gRType[r] == 1)
        {
            FuzzyFires[r] = FIS_MAX;
            for (i = 0; i < fis_gcI; ++i)
            {
                index = fis_gRI[r][i];
                if (index > 0)
```

```
                    FuzzyFires[r]     =     fis_min(FuzzyFires[r],
FuzzyInput[i][index - 1]);
                else if (index < 0)
                    FuzzyFires[r]  =  fis_min(FuzzyFires[r],  1  -
FuzzyInput[i][-index - 1]);
                else
                    FuzzyFires[r] = fis_min(FuzzyFires[r], 1);
            }
        }
        else
        {
            FuzzyFires[r] = FIS_MIN;
            for (i = 0; i < fis_gcI; ++i)
            {
                index = fis_gRI[r][i];
                if (index > 0)
                    FuzzyFires[r]     =     fis_max(FuzzyFires[r],
FuzzyInput[i][index - 1]);
                else if (index < 0)
                    FuzzyFires[r]  =  fis_max(FuzzyFires[r],  1  -
FuzzyInput[i][-index - 1]);
                else
                    FuzzyFires[r] = fis_max(FuzzyFires[r], 0);
            }
        }
        FuzzyFires[r] = fis_gRWeight[r] * FuzzyFires[r];
        sW += FuzzyFires[r];
    }
    if (sW == 0)
    {
        for (o = 0; o < fis_gcO; ++o)
        {
            g_fisOutput[o] = ((fis_gOMax[o] + fis_gOMin[o]) / 2);
        }
    }
    else
    {
        for (o = 0; o < fis_gcO; ++o)
        {
            g_fisOutput[o] = fis_defuzz_centroid(FuzzyRuleSet, o);
        }
    }
}
double avergearray(int* arr, int number) {
  int i;
  int max, min;
  double avg;
  long amount = 0;
  if (number <= 0) {
    Serial.println("Error number for the array to avraging!/n");
    return 0;
  }
  if (number < 5) { //less than 5, calculated directly statistics
    for (i = 0; i < number; i++) {
      amount += arr[i];
    }
    avg = amount / number;
    return avg;
  } else {
```

```
    if (arr[0] < arr[1]) {
      min = arr[0]; max = arr[1];
    }
    else {
      min = arr[1]; max = arr[0];
    }
    for (i = 2; i < number; i++) {
      if (arr[i] < min) {
        amount += min;      //arr<min
        min = arr[i];
      } else {
        if (arr[i] > max) {
          amount += max;  //arr>max
          max = arr[i];
        } else {
          amount += arr[i]; //min<=arr<=max
        }
      }//if
    }//for
    avg = (double)amount / (number - 2);
  }//if
  return avg;
}
// Firebase Logger
void firebase_logger()
{
  firebaseJson.set("/Input_TDS", Input_TDS);
  firebaseJson.set("/deltaerrortds", deltaerrortds);
  firebaseJson.set("/errortds", errortds);
  firebaseJson.set("/Input_PH", Input_PH);
  firebaseJson.set("/deltaerrorph", deltaerrorpH);
  firebaseJson.set("/errorph", errorpH);
  firebaseJson.set("/aktuatortds", g_fisOutput[1]);
  firebaseJson.set("/aktuatorph", g_fisOutput[0]);
  Firebase.updateNode(firebaseData, "/akuaponik", firebaseJson);
}
```

Lampiran B Hasil Pengujian