/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* GravityEc.cpp

\*

\* Copyright (C) 2017 [DFRobot](http://www.dfrobot.com),

\* GitHub Link :<https://github.com/DFRobot/watermonitor>

\* This Library is free software: you can redistribute it and/or modify

\* it under the terms of the GNU General Public License as published by

\* the Free Software Foundation, either version 3 of the License, or

\* (at your option) any later version.

\*

\* Description:Monitoring water quality parameters Conductivity

\*

\* Product Links：<http://www.dfrobot.com.cn/goods-882.html>

\*

\* Sensor driver pin：A1 (ecSensorPin(A1))

\*

\* author : Jason(jason.ling@dfrobot.com)

\* version : V1.0

\* date : 2017-04-17

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "GravityEc.h"

#include "Arduino.h"

GravityEc::GravityEc(ISensor\* temp) :ecSensorPin(A1), ECcurrent(0), index(0), AnalogAverage(0),

AnalogValueTotal(0), averageVoltage(0), AnalogSampleTime(0), printTime(0),sum(0),

tempSampleTime(0), AnalogSampleInterval(25),printInterval(700)

{

this->ecTemperature = temp;

}

GravityEc::~GravityEc()

{

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// function name: setup ()

// Function Description: Initializes the sensor

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void GravityEc::setup()

{

pinMode(ecSensorPin, INPUT);

for (byte thisReading = 0; thisReading < numReadings; thisReading++)

readings[thisReading] = 0;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// function name: update ()

// Function Description: Update the sensor value

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void GravityEc::update()

{

calculateAnalogAverage();

calculateEc();

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// function name: getValue ()

// Function Description: Returns the sensor data

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

double GravityEc::getValue()

{

return ECcurrent;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// function name: calculateAnalogAverage ()

// Function Description: Calculates the average voltage

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void GravityEc::calculateAnalogAverage()

{

if (millis() - AnalogSampleTime >= AnalogSampleInterval)

{

AnalogSampleTime = millis();

readings[index++] = analogRead(ecSensorPin);

if (index == numReadings)

{

index = 0;

for (int i = 0; i < numReadings; i++)

this->sum += readings[i];

AnalogAverage = this->sum / numReadings;

this->sum = 0;

}

}

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// function name: calculateAnalogAverage ()

// Function Description: Calculate the conductivity

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void GravityEc::calculateEc()

{

if (millis() - printTime >= printInterval)

{

printTime = millis();

averageVoltage = AnalogAverage\*5000.0 / 1024.0;

double TempCoefficient = 1.0 + 0.0185\*(this->ecTemperature->getValue() - 25.0); //temperature compensation formula: fFinalResult(25^C) = fFinalResult(current)/(1.0+0.0185\*(fTP-25.0));

double CoefficientVolatge = (double)averageVoltage / TempCoefficient;

if (CoefficientVolatge < 150) {

ECcurrent = 0;

return;

}

else if (CoefficientVolatge > 3300)

{

ECcurrent = 20;

return;

}

else

{

if (CoefficientVolatge <= 448)

ECcurrent = 6.84\*CoefficientVolatge - 64.32; //1ms/cm<EC<=3ms/cm

else if (CoefficientVolatge <= 1457)

ECcurrent = 6.98\*CoefficientVolatge - 127; //3ms/cm<EC<=10ms/cm

else

ECcurrent = 5.3\*CoefficientVolatge + 2278; //10ms/cm<EC<20ms/cm

ECcurrent /= 1000; //convert us/cm to ms/cm

}

}

}