#include <lmic.h>

#include <hal/hal.h>

#include <SPI.h>

#include <NewPing.h> // Library for the ultrasonic sensor

// Pin definitions for the ultrasonic sensor

const int trig = 5;

const int echo = 3;

int distance;

int duration;

// Create a NewPing instance

NewPing sonar(trig, echo, 400); // Parameters: TRIG\_PIN, ECHO\_PIN, MAX\_DISTANCE

// LMIC pin mapping

const lmic\_pinmap lmic\_pins = {

.nss = 10,

.rxtx = LMIC\_UNUSED\_PIN,

.rst = 9,

.dio = {2, 6, 7},

};

// Device EUI, Application EUI, and App Key

static const u1\_t PROGMEM DEVEUI[8] = { 0xAC, 0x1F, 0x09, 0xFF, 0xFE, 0x0C, 0xAD, 0xE5 };

static const u1\_t PROGMEM APPEUI[8] = { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 };

static const u1\_t PROGMEM APPKEY[16] = { 0xC8, 0xA2, 0x62, 0x7F, 0x25, 0x1B, 0xA9, 0xAF, 0x2F, 0xFF, 0x55, 0x02, 0xC0, 0xA5, 0xB2, 0xA7 };

void os\_getDevEui(u1\_t\* buf) { memcpy\_P(buf, DEVEUI, 8); }

void os\_getArtEui(u1\_t\* buf) { memcpy\_P(buf, APPEUI, 8); }

void os\_getDevKey(u1\_t\* buf) { memcpy\_P(buf, APPKEY, 16); }

// Transmission interval

const unsigned TX\_INTERVAL = 60; // seconds

static void do\_send(osjob\_t\* j) {

// Get distance from the ultrasonic sensor

distance = sonar.ping\_cm(); // Use NewPing library to get distance

// Check for invalid readings

if (distance == 0) {

Serial.println(F("Error: Distance measurement failed."));

distance = 50; // Set a default value in case of error

}

// Convert distance to percentage

float maxDistance = 50.0; // Maximum distance in cm

float minDistance = 0.0; // Minimum distance in cm

float percentage = ((maxDistance - distance) / (maxDistance - minDistance)) \* 100;

// Ensure percentage is within bounds

if (percentage > 100) percentage = 100;

if (percentage < 0) percentage = 0;

// Prepare data for transmission

uint8\_t payload[1];

payload[0] = (uint8\_t)percentage; // Convert percentage to uint8\_t

LMIC\_setTxData2(1, payload, sizeof(payload), 0);

Serial.println(F("Packet queued"));

}

static osjob\_t sendjob;

void onEvent(ev\_t ev) {

unsigned long currentTime = millis(); // Get the current time in milliseconds

Serial.print(currentTime);

Serial.print(F(" ms - "));

switch (ev) {

case EV\_SCAN\_TIMEOUT:

Serial.println(F("EV\_SCAN\_TIMEOUT - Scan operation timed out"));

break;

case EV\_BEACON\_FOUND:

Serial.println(F("EV\_BEACON\_FOUND - Beacon detected"));

break;

case EV\_BEACON\_MISSED:

Serial.println(F("EV\_BEACON\_MISSED - Beacon missed"));

break;

case EV\_BEACON\_TRACKED:

Serial.println(F("EV\_BEACON\_TRACKED - Beacon is being tracked"));

break;

case EV\_JOINING:

Serial.println(F("EV\_JOINING: The Device is attempting to join the network"));

break;

case EV\_JOINED:

Serial.println(F("EV\_JOINED - Device has joined the network"));

LMIC\_setLinkCheckMode(0); // Disable link check

break;

case EV\_RFU1:

Serial.println(F("EV\_RFU1 - Reserved for future use"));

break;

case EV\_JOIN\_FAILED:

Serial.println(F("EV\_JOIN\_FAILED - Device failed to join the network"));

break;

case EV\_REJOIN\_FAILED:

Serial.println(F("EV\_REJOIN\_FAILED: Device failed to rejoin the network"));

break;

case EV\_TXCOMPLETE:

Serial.println(F("EV\_TXCOMPLETE - Transmission complete"));

os\_setTimedCallback(&sendjob, os\_getTime() + sec2osticks(TX\_INTERVAL), do\_send);

break;

case EV\_LOST\_TSYNC:

Serial.println(F("EV\_LOST\_TSYNC - Lost time synchronization"));

break;

case EV\_RESET:

Serial.println(F("EV\_RESET - Device has been reset"));

break;

case EV\_RXCOMPLETE:

Serial.println(F("EV\_RXCOMPLETE - Reception complete"));

break;

case EV\_LINK\_DEAD:

Serial.println(F("EV\_LINK\_DEAD - Communication link is dead"));

break;

case EV\_LINK\_ALIVE:

Serial.println(F("EV\_LINK\_ALIVE - Communication link is alive"));

break;

default:

Serial.print(F("Unknown event: "));

Serial.println(ev);

break;

}

}

void setup() {

Serial.begin(9600);

// Set pin modes for the ultrasonic sensor

pinMode(trig, OUTPUT);

pinMode(echo, INPUT);

Serial.println(F("Starting"));

// Initialize LMIC and settings

os\_init();

LMIC\_reset();

LMIC\_setClockError(MAX\_CLOCK\_ERROR \* 1 / 100);

LMIC\_setLinkCheckMode(0);

// Set the frequency to 433 MHz

LMIC.freq = 433175000; // Use a frequency in the 433 MHz band

// Set data rate and transmit power

LMIC\_setDrTxpow(DR\_SF7, 14);

// Start the first transmission

do\_send(&sendjob);

}

void loop() {

// Measure distance

digitalWrite(trig, LOW);

delayMicroseconds(2);

digitalWrite(trig, HIGH);

delayMicroseconds(10);

digitalWrite(trig, LOW);

duration = pulseIn(echo, HIGH);

distance = (duration \* 0.034) / 2;

// Convert distance to percentage

float maxDistance = 50.0; // Maximum distance in cm

float minDistance = 0.0; // Minimum distance in cm

float percentage = ((maxDistance - distance) / (maxDistance - minDistance)) \* 100;

// Check if percentage is greater than 100% or if distance is less than or equal to 0 cm

if (distance <= 0) {

Serial.println("Bin Overflowing");

} else if (percentage > 100) {

Serial.println("Bin Overflowing (percentage > 100%)");

}

// Ensure percentage is within bounds

if (percentage > 100) percentage = 100;

if (percentage < 0) percentage = 0;

// Print distance and percentage for debugging

Serial.print("Distance: ");

Serial.print(distance);

Serial.print(" cm, Percentage: ");

Serial.print(percentage);

Serial.println(" %");

// Run LMIC event loop

os\_runloop\_once();

// Delay between measurements

delay(3000);

}