TSK24 Projektityö

Buttonpusher

Aku Laurila

akulau@student.uef.fi

Sisällysluettelo

Johdanto

Projektityön tiimoilla syntyi pieni järjestelmä, jonka nimesin luovasti Buttonpusher:iksi, sillä se voisi soveltua vaikkapa nappien paineluun. Käytännössä järjestelmän voi asettaa tekemään mitä vain, mihin kääntyvä servo soveltuisi, oli se sitten napin painaminen, verhojen avaaminen, tuolin kääntäminen tai muu pyörivää, mekaanista liikettä vaativa käyttökohde.

Järjestelmä on pohjimmiltaan yksinkertainen. Se hyväksyy käyttäjältä syötteen siitä, käytetäänkö aktivointiin kosteus- vai lämpötilasensoria, mikä on aktivointiin vaadittava raja-arvo ja halutaanko järjestelmä aktivoida raja-arvon ylittyessä vai alittuessa. Kaikki keskeinen tieto näkyy LCD-näytöllä yksinkertaisessa käyttöliittymässä, jota ohjataan yhdellä napilla painallusten pituutta muuttaen.

Projektityön alkuperäisenä suunnitelmana oli toteuttaa yksinkertainen, lämpötila-aktivoituva servo, jonka aktivoitumisrajaa käyttäjä voi vaihtaa nappia painamalla. Olin suunnitelmaa tehdessäni vielä epävarma siitä, minkälaisen järjestelmän edes osaisin rakentaa, mutta lopputuloksesta tuli mielestäni hyvällä tavalla suunniteltua monipuolisempi ja käyttäjäystävällisempi. Suunniteltujen ominaisuuksien lisäksi, voi käyttäjä antaa enemmän asetuksia aktivoinnille. Järjestelmässä on myös vaihtoehto raja-arvon asettamiselle sarjaliikenteen avulla, jotta arvo voidaan asettaa oletusarvoja tarkemmiksi. Oletusarvot, joita käyttöliittymässä voidaan vaihtaa ovat 0-100, kymmenen arvon välein.

Lähdekoodi on saatavilla suoraan GitHubista:

https://github.com/Ka-Q/Buttonpusher

Demovideo YouTubessa:

https://youtu.be/gR7hWgPGY1E?si=Cu4wei8eV1z5KsnU

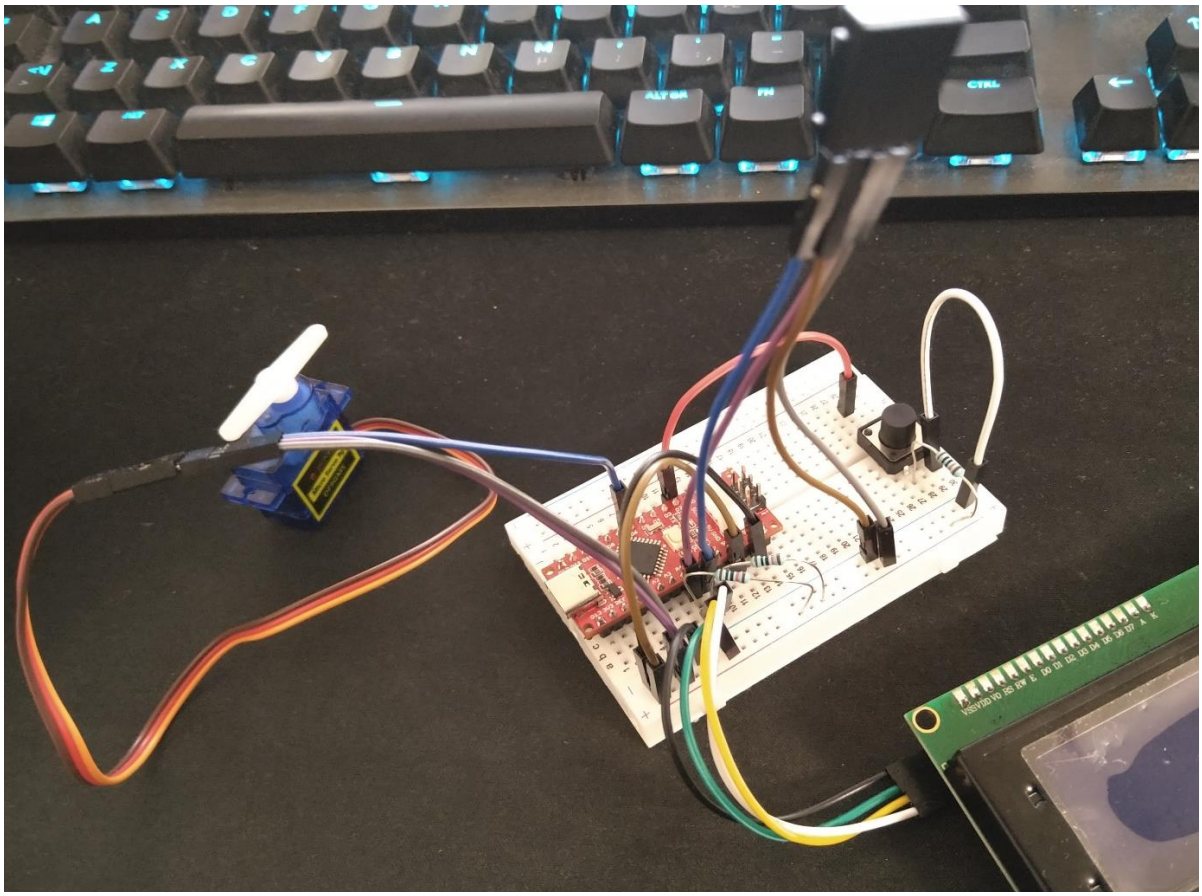
Laitteisto

Komponentit

Käytetyt komponentit:

- Breadboard
- Arduino Nano
- Kosteus- ja lämpötilasensori
- Servo
- LCD-näyttö
- Nappi
- 3 resistoria
- 15 irrallista johtoa

Kytkennät



Valokuva kytkennöistä.

Selitystä:

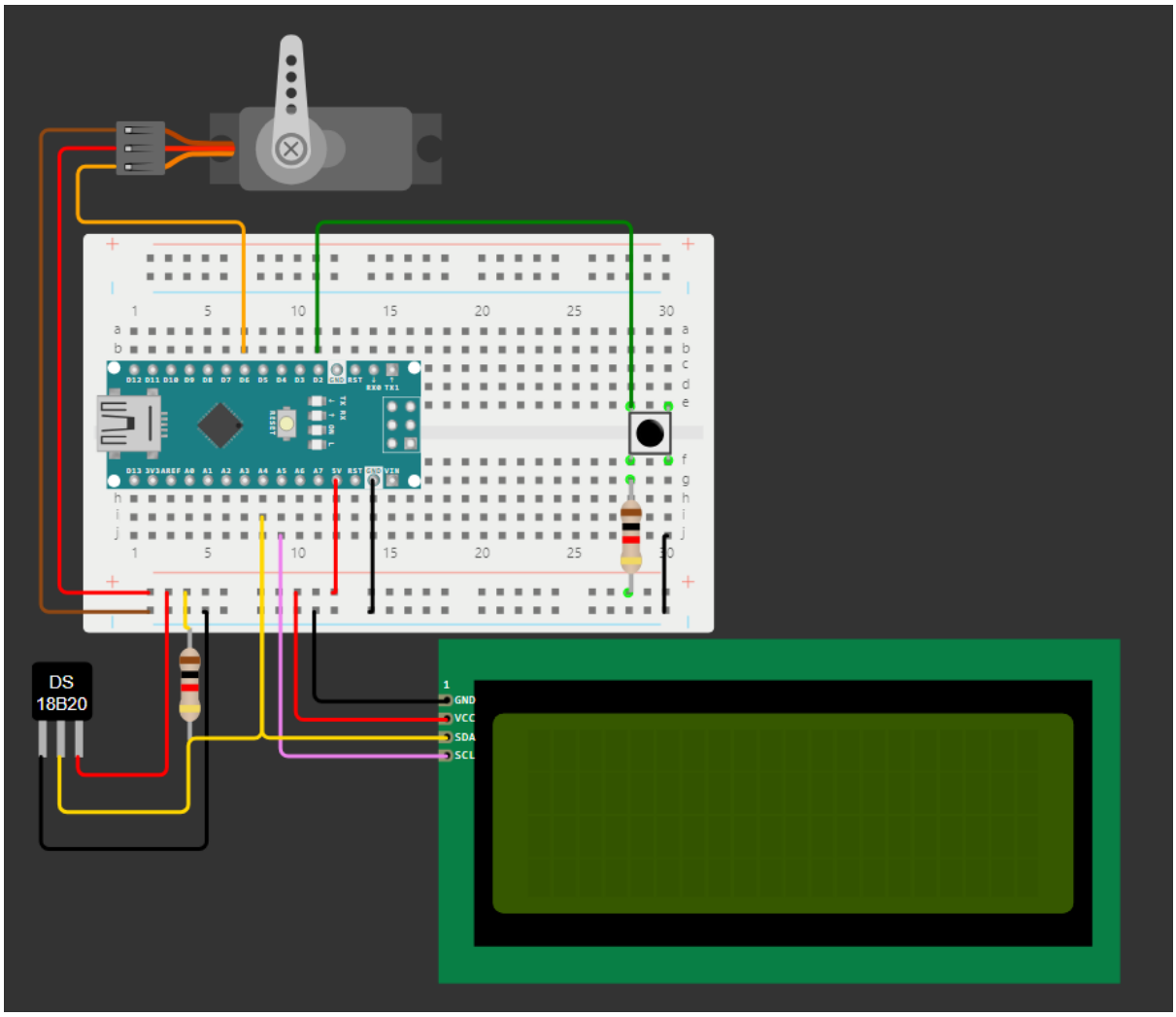
Nappi yhdistettynä kontrollerin pinniin D2. Napilla on myös resistori.

Kosteus- ja lämpötilasensorin SDA kontrollerin pinniiin A4, SCL pinniin A5.

LCD-näytöllä sama kuin sensorilla, eli SDA A4 ja SCL A5.

A4 ja A5 pinneistä vastukset sensoria varten.

Servo liitettynä pinniin D6.



Wokwi.com:illa tehty suuntaa antava kaaviokuva järjestelmästä. Huom. lämpötilasensorilta lähtevä keltainen johto on todellisuudessa kaksi johtoa pinneihin A4 ja A5 (SDA ja SCL), joilla molemmilla kuuluisi olla omat vastukset.

Ohjelmisto

Käyttöliittymä

Arduinolla pyörivä ohjelma toimii seuraavan kaavan mukaisesti:

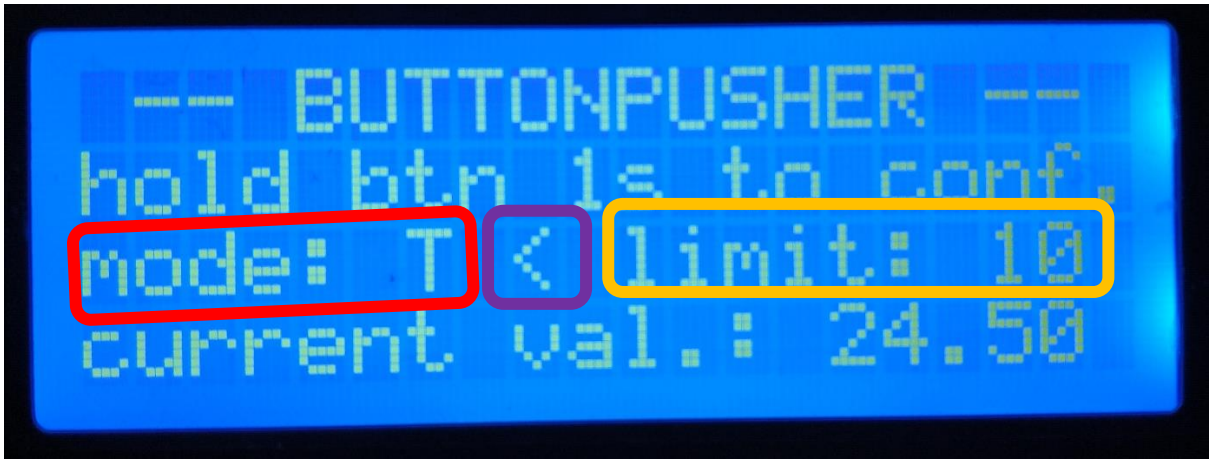
Aloitusnäytölle tulostetaan tietoa järjestelmän reaaliaikaisesta tilasta. Näkymästä näkee käyttäjän valitsemat arvot ja reaaliaikaista sensoridataa joko lämpötilasta tai kosteudesta. Käyttäjä voi halutessaan säätää järjestelmän tilaa painamalla nappia pitkään.

Päästäessään napin pitkän painalluksen jälkeen irti, siirtyy käyttöliittymä toiselle sivulle ”MODE CONFIG”. Täällä käyttäjä voi nappia nopeasti painamalla selata saatavilla olevia vaihtoehtoja sensoreista. Vaihtoehtoja ovat lämpötila (Temperature) tai ilmankosteus (Humidity). Valittuaan mieleisensä vaihtoehdon, voi käyttäjä jälleen painaa nappia pitkään pohjassa ja siirtyä seuraavalle sivulle.

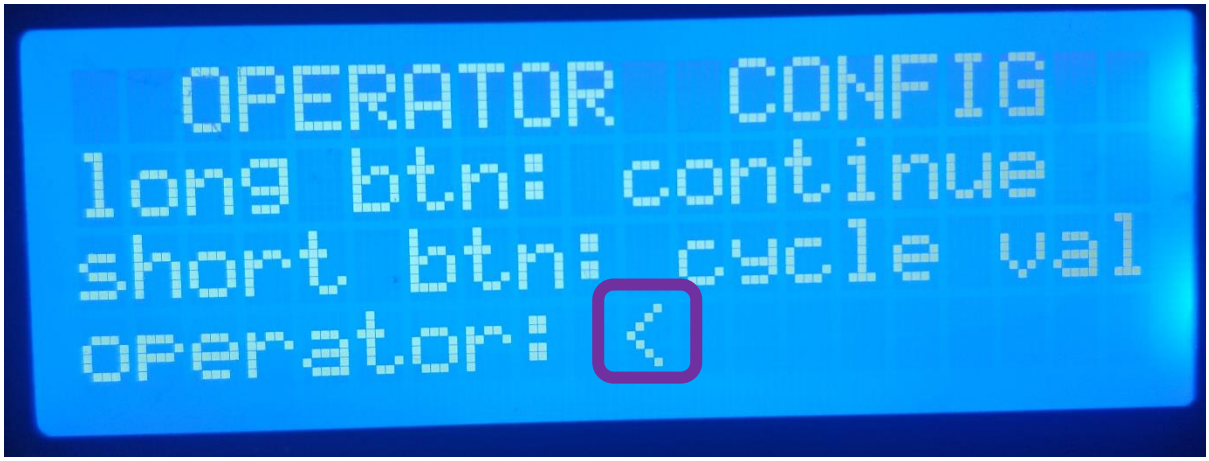
Käyttöliittymän kolmas sivu, ”LIMIT CONFIG” antaa käyttäjän valita ennalta määrättyjen arvojen listalta sensorin raja-arvon, jonka kohdalla järjestelmän tulisi aktivoida servo. Valittuaan raja-arvon nopeilla napin painalluksilla, voi käyttäjä painaa nappia pitkään ja siirtyä viimeiselle asetussivulle.

Viimeinen sivu, ”OPERATOR CONFIG” on vastaavanlainen kuin aiemmatkin asetussivut, mutta täällä käyttäjä valitsee operaattorin, jonka avulla määritellään, tuleeko sensorin lukeman arvon olla suurempi- (”>”) vai pienempi kuin (”<”) asetettu raja-arvo. Operaattorin päätettyään, käyttäjä painaa jälleen pitkään nappia ja siirtyy takaisin aloitusnäytölle, minne hänen tekemänsä muutokset ovat päivittyneet näkymään.

Graafisen käyttöliittymän lisäksi, voi käyttäjä syöttää halutun sensorin raja-arvon myös sarjaliikenteen avulla. Arvoksi hyväksytään negatiivinen tai positiivinen kokonaisluku.



Yllä kuva aloitusnäytöstä. Punainen: Luettava sensori, Violetti: käytössä oleva operaattori, Oranssi: Sensorin raja-arvo. Alimmalla rivillä on reaaliaikainen sensorin lukema.



Yllä kuvat asetusnäkymistä. Jokaisessa näkymässä alimmalla rivillä olevaa arvoa voidaan muuttaa lyhyellä napinpainalluksella.

Käytetyt kirjastot

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <Servo.h>
#include "Adafruit_Sensor.h"
#include "Adafruit_AM2320.h"
```

Käytettyjä kirjastoja ovat sisäänrakennetut Wire- ja Servo kirjastot, John Rickmanin LiquidCrystal_I2C-kirjasto (https://github.com/johnrickman/LiquidCrystal_I2C) ja Adafruitin Adafruit_Sensor ja Adafruit_AM2320 kirjastot.

Wire-kirjastoa käytetään I2C -laitteiden hallinnointiin. Servo-kirjastolla ohjataan servoa.

LiquidCrystal_I2C-kirjastolla hallinnoidaan LCD-näyttöä ja Adafruitin kirjastoilla mm. luetaan lämpötila- ja ilmankosteusdataa sensorista.

Projektissa ei käytetä muita ulkopuolisia kirjastoja.

Rakenne

Erittelen tähän mielenkiintoisimpia osia ohjelmakoodista. Koko ohjelmakoodi löytyy liitteenä tämän projektiraportin viimeisiltä sivuilta.

Datan talletus

Suurin osa käyttäjän valitsemasta datasta on talletettu taulukoihin ja käyttäjän valinta on oikeasti vain taulukon indeksi:

```
const unsigned long LONG_PRESS_TIME = 1000;
const char* MODES[] = {"Temperature", "Humidity"};
const int MODE_COUNT = 2;
const int LIMITS[] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
const int LIMIT_COUNT = 10;
const char* OPERATORS[] = {"<", ">"};
const int OPERATOR_COUNT = 2;
```

Tämä sen takia, että käyttöliittymässä on kiva näyttää vaikkapa teksti tai merkki, mutta ohjelmakoodissa kokonaislukujen käsittely on helpompaa. Käyttäjän vaihtaessa asetusnäkymässä arvoa, kutsutaan seuraavanlaista funktiota halutulle muuttujalle:

```
// Function for cycling array indexes on short press
int cycleOnShortPress(int val, int count) {
  if (is_short_pressed) {
    val++;
    if (val > count) {
      val = 1;
    }
  }
  return val;
}
```

Funktio tarkistaa ensin, onko kyseessä lyhyt painallus, minkä jälkeen se ”kierrättää” taulukon indeksejä alusta loppuun ja indeksien loputtua takaisin alkuun. Esimerkki funktion käytöstä asetussivulla:

```
// Config screen for changing selected mode
void modeConfigScreen() {
  mode = cycleOnShortPress(mode, MODE_COUNT);
  lcd.setCursor(0,0);
  printConfigInfo("    MODE  CONFIG    ");
  lcd.setCursor(0,3);
  lcd.print("mode:               ");
  lcd.setCursor(6,3);
  lcd.print(MODES[mode - 1]);
}
```

Napin tila (short/long)

Ohjelman pääsilmukassa tarkistetaan joka kierroksella napin painalluksen tilaa funktiolla checkButtonPress():

```
// Calls either buttonDown or buttonUp based on btn_value.
void checkButtonPress(int btn_value) {
  if (btn_value == LOW) {
    buttonDown();
  } else {
    if (press_started) {
      buttonUp();
    }
  }
}

// Acts like an event for button down.
// Starts or continues tracking the time button has been pressed
void buttonDown() {
  if (press_started == false) {
    elapsed_time = 0;
    press_started = true;
    elapsed_time = 0;
    previous_elapsed_time = millis();
  } else {
    elapsed_time = millis() - previous_elapsed_time;
  }
}

// Acts like an event for button up.
// Decides if press was either short or long based on time tracked
void buttonUp() {
  press_started = false;
  is_short_pressed = false;
  is_long_pressed = false;
  if (elapsed_time < LONG_PRESS_TIME) {
    is_short_pressed = true;
  } else {
    is_long_pressed = true;
  }
}
```

Näissä funktioissa päivitetään globaaleja muuttujia is_short_pressed ja is_long_pressed niin, että niitä voidaan tiedustella muualla ohjelmassa.

Raja-arvon lukeminen sarjaliikenteestä

Pääsilmukassa luetaan myös mahdollista kustomoitua arvoa raja-arvolle:

```
// Read custom value from Serial
  if (Serial.available() > 0) {
    String input_line = Serial.readStringUntil('\n');
    input_line.trim(); // Remove leading/trailing spaces

    // Convert the input to an integer
    int temp_value = input_line.toInt();

    if (temp_value >= -999 && temp_value <= 999) {
      custom_limit = temp_value;
      use_custom_limit = true;
      Serial.print("Accepted integer: ");
      Serial.println(custom_limit);
    } else {
      Serial.println("Value out of range. Must be between -999 and 999.");
    }
  }
```

Mikäli use_custom limit on asetettu tässä todeksi, niin järjestelmä käyttää sitä limit-muuttuja sijasta.

Sensorin lukeminen ja servon ohjaaminen

Sensoria luetaan vain, kun käyttäjä on aloitusnäkymässä:

```
// Read value based on selected mode
  value = ERROR;
  if (mode == 1) {
    value = am2320.readTemperature();
  } else if (mode == 2) {
    value = am2320.readHumidity();
  }
```

Pääsilmukassa servoa sitten ohjataan käyttäjän valitsemien asetusten mukaan:

```
  // Move servo if in default view (screen_number == 0)
  // if evaluate() true -> move to 270, if evaluate() false -> reset to 0
  if (screen_number == 0) {
    bool move_servo = false;
    if (use_custom_limit) {
      move_servo = compare(value, custom_limit);
    } else {
      move_servo = compare(value, LIMITS[limit - 1]);
    }
    if (move_servo) {
      myservo.write(270);
      delay(100);
    } else {
      myservo.write(0);
      delay(100);
    }
  }
```

Sensorin lukemaa dataa verrataan asetettuun rajaan valitulla operaattorilla (op) compare() -funktion avulla:

```
// Function for compares given values using globally set operator
bool compare(float value, int value2) {
  if (value == ERROR) {
    return false;
  }

  if (op == 1) {
    return (value < value2);
  } else if (op == 2) {
    return (value > value2);
  } else {
    return false;
  }
}
```

LCD-näytölle tulostaminen

Näytölle tulostaminen on hyvin suoraviivaista LiquidCrystal_I2C-kirjastoa käyttäen. Tulostaminen tapahtuu pääosin käteviä funktioita setCursor() ja print() käyttäen. Esimerkiksi, asetetaan kursori ensimmäisen rivin (0) alkuun (0) ja tulostetaan TITLE_TEXT. Sen jälkeen siirretään kursori toisen rivin (1) alkuun (0) ja tulostetaan TOOLTIP_TEXT:

```
lcd.setCursor(0,0);
lcd.print(TITLE_TEXT);
lcd.setCursor(0,1);
lcd.print(TOOLTIP_TEXT);
```

Tällä samalla tavalla tulostetaan kaikki näytöllä näkyvä informaatio. Alla hieman monimutkaisempi esimerkki, jossa tulostetaan aloitusnäytöllä näkyvää infoa valituista asetuksista.

```
// Print current config info. Examples:
// "mode: H < limit: 30" (humidity less than 30)
// "mode: T > limit: 10" (temperature more than 10)
lcd.setCursor(0,2);
lcd.print("mode: ");
lcd.print(MODES[mode - 1][0]);
lcd.print(" ");
lcd.print(OPERATORS[op - 1][0]);
lcd.print(" limit:");
if (use_custom_limit) {
  printLimit(custom_limit);
} else {
  printLimit(LIMITS[limit-1]);
}
```

Testaus ja tuloksia/havaintoja

Järjestelmällä on tiettyjä vaatimuksia, jotka sen tulee täyttää ollakseen toimiva ja käyttäjäystävällinen. Prioriteetit korkeimmasta matalimpaan: 1, 2, 3

VAATIMUS	TESTAUSASKELEET	P	TULOS
Servo käyttäytyy odotetulla tavalla, kun vahditaan lämpötilaa	Asetetaan mode: Temperature, limit: 30, operator: >. Nostetaan ympäristön lämpötilaa ja havaitaan servon käyttäytymistä. Lasketaan lämpötilaa ja servo palautuu.	1	Toimii suunnitellusti
Servo käyttäytyy odotetulla tavalla, kun vahditaan kosteutta	Asetetaan mode: Humidity, limit: 70, operator: >. Nostetaan ympäristön ilmankosteutta ja havaitaan servon käyttäytymistä. Lasketaan kosteutta ja servo palautuu.	1	Toimii suunnitellusti
Järjestelmä tunnistaa lyhyen ja pitkän painalluksen oikein.	Painetaan nappia pitkään. Käyttöliittymän tulisi siirtyä seuraavalle sivulle. Jos sivu on asetussivu, nopealla painalluksella arvon tulee muuttua.	1	Toimii suunnitellusti. Aloitusnäytössä lyhyt painallus ei tee mitään.
Servo liikkuu ainoastaan, kun järjestelmä on aloitustilassaan. Servo ei saa reagoida vielä asetuksia säädettäessä.	Asetetaan mikä vain mode. Valitaan mikä vain arvo, jonka tiedetään olevan korkeampi, kuin reaaliaikainen sensorin lukema arvo. Valitaan operaattori ”<”. Havaitaan servon käyttäytymistä.	2	Toimii suunnitellusti. Servo liikkuu oikeaan asentoonsa vasta, kun asetuksista on jatkettu takaisin aloitusnäyttöön.
Graafinen käyttöliittymä pyyhkii aiempien näkymien tiedot oikein, eikä väärää tietoa jää näytölle.	Vaihdetaan näytöltä toiselle. Havainnoidaan etenkin, ettei tyhjiin paikkoihin ole jäänyt kirjaimia aiemmalta näytöltä	3	Toimii suunnitellusti. Näytölle ei jää vanhaa dataa kummittelemaan.
Sarjaliikenteestä hyväksytään ainoastaan positiivisia tai negatiivisia kokonaislukuja	Syötetään sarjaliikenteeseen arvoja ja varmistetaan, että ainoastaan kokonaisluvut hyväksytään. Muulloin arvoksi 0.	2	Toimii suunnitellusti. Mikäli syöte alkaa kokonaisluvulla, niin luetaan siihen asti, kunnes ei enää onnistu -> 12asd = 12, mutta asd12 = 0

Edellä taulukoidut testit ovat sellaisia, jota tein jatkuvasti ohjelmaa kirjoittaessani. Testit on toistettu useaan kertaan järjestelmän valmistumisen jälkeen ja niissä on käytetty myös paljon variaatiota testattavissa arvoissa. Näiden vaatimuskriteerien ansiosta järjestelmän vaadittu toiminta on hyvin määritelty ja testien avulla järjestelmästä tuli toimiva ja vakaa.

Vaikka suuri osa ominaisuuksista toimi niitä toteutettaessa, tuli ohjelmoidessa uudelleenkirjoitettua monia osia sovelluksesta. Joskus näitä uudelleenkirjoituksia / refactorointeja tehdessä jokin järjestelmän osa-alue hajosi, eikä toiminut enää suunnitellulla tavalla. Rutiininomaiset testit auttoivat järjestelmän pitämisessä vakaana kehitystyön edetessä.

Alkuperäinen projektisuunnitelma määritteli joitain projektin vaatimuksia, mutta olin tarkoituksella jättänyt suunnitelman hieman avoimeksi. Kehitystyön lomassa syntyneet testit auttoivat myös määrittelemään tarkemmin projektin vaatimuksia. Testien avulla löytyi monia bugeja, mutta ilmeisin, mikä mieleeni tulee, on väärän taulukkoindeksin käyttö sensoridataa verratessa. Tämä ilmeni testitaulukon kahta ensimmäistä testiä tehdessä, kun sensori käyttäytyi täysin odottamattomalla tavalla. Eräs toinen testauksessa löytynyt bugi oli LCD-näytölle jäävät ”haamukirjaimet” näkymää vaihdettaessa. Sovellus ei pyyhkinyt riviä kunnolla ennen uuden näkymän piirtämistä, joten tyhjiin paikkoihin jäi aiemman näkymän dataa.

Lämpötila/kosteussensori antaa satunnaisen epäsatunnaisesti nan-arvoja, mutta tiedonhaun jälkeen vaikutti siltä, että ongelma on yleinen käyttämässämme sensorissa. Ongelma korjaantuu useimmiten kytkemällä virta pois ja takaisin päälle, joten jätin sensorilukemien virheet huomioimatta bugina. Korjautuisi lienee laadukkaammalla sensorilla?

Sarjaliikennettä lukiessa kokonaisluku hyväksytään myös, vaikka sen perässä olisi sopimattomia merkkejä. Esim. ”12asd” lukee arvoksi 12, mutta ”asd12” ei ole 12, sillä merkkien lukeminen keskeytyy jo ennen numeroita. Tämä toiminnallisuus on toistaiseksi ok, eikä varsinainen bugi.

Johtopäätökset ja yhteenveto

Projektityön tuotoksena syntyi toimiva järjestelmä, joka vastasi jotakuinkin alkuperäistä projektisuunnitelmaa. Järjestelmään tuli suunnitelmassa määrittelemättömiä lisäominaisuuksia, mutta ne tukevat järjestelmän toimintaa ja tekevät siitä käyttäjäystävällisemmän ja antavat sille hiotumman olemuksen. Projekti oli ensikosketukseni fyysisiin komponentteihin breadboardilla, sillä en ole päässyt paikalle kurssin luennoille tai harjoituksiin. Olen positiivisesti yllättynyt siitä, miten hyvin projekti eteni, vaikka aiempaa kokemusta aiheesta ei ollut.

Kurssimateriaali ja Internet-resurssit auttoivat tarpeeksi ja en rikkonut (tietääkseni) yhtäkään komponenttia. Lämpötila/kosteussensori antaa tosiaan joskus nan-arvoja, mutta teki niin jo alusta asti. Aihetta käsittelevien keskustelufoorumien perusteella, arvelisin tämän olevan vika sensorissa itsessään. Arvelen ongelman korjaantuvan laadukkaammalla sensorilla.

Komponenteilla räpeltäminen oli hauskaa, mutta itse nautin eniten järjestelmän ohjelmoinnista. C++ ei ole itselleni hirveän tuttua perusteita enempää, joten ohjelmakoodin rakenteessa olisi parannettavaa. Käytän ohjelmakoodissa paljon globaaleja muuttujia ja uskon, että niitä saisi karsittua pois ohjelmakoodin selkeyttämiseksi. Sain koodia kuitenkin siivoiltua jonkin verran, pilkottua tuplakoodia funktioiksi ja kommentoitua hyvin, joten olen loppujen lopuksi erittäin tyytyväinen ohjelman rakenteeseen.

Projektista tuli onnistunut ja opin paljon uutta komponenteista ja niiden ohjelmoinnista. Projekti vastaa pääpiirteittäin suunniteltua, ollen kuitenkin ominaisuuksiltaan kehittyneempi versio.

Lähdekoodi on saatavilla suoraan GitHubista:

https://github.com/Ka-Q/Buttonpusher

Demovideo YouTubessa:

https://youtu.be/gR7hWgPGY1E?si=Cu4wei8eV1z5KsnU

Projektityössä hyödylliseksi koettuja oppaita/resursseja:

https://docs.arduino.cc/tutorials/generic/basic-servo-control/

https://github.com/harshkzz/Arduino-LCD20x4-i2c

https://learn.adafruit.com/adafruit-am2320-temperature-humidity-i2c-sensor/arduino-usage

Liitteet

Ohjelmakoodi kokonaisuudessaan

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <Servo.h>
#include "Adafruit_Sensor.h"
#include "Adafruit_AM2320.h"

// CONSTANTS FOR PINS & MISC
const int BUTTON_PIN = 2;
const int SERVO_PIN = 6;

const char LCD_ADDRESS = 0x27;
const float ERROR = -999999;

// CONSTANTS FOR CONFIG
const unsigned long LONG_PRESS_TIME = 1000;
const char* MODES[] = {"Temperature", "Humidity"};
const int MODE_COUNT = 2;
const int LIMITS[] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
const int LIMIT_COUNT = 10;
const char* OPERATORS[] = {"<", ">"};
const int OPERATOR_COUNT = 2;

// CONSTANTS FOR LCD TEXTS
char TITLE_TEXT[] = " -- BUTTONPUSHER -- ";
String TOOLTIP_TEXT = "hold btn " + String(LONG_PRESS_TIME / 1000) + "s to conf.";
char SELECTED_LIMIT_TEXT[] = "selected lim: ";
char CURRENT_VALUE_TEXT[] = "current val.: ";
char LONG_TOOLTIP_TEXT[] = "long btn: continue  ";
char SHORT_TOOLTIP_TEXT[] = "short btn: cycle val";

// GLOBALS FOR BUTTON HANDLING
unsigned long elapsed_time = 0;
unsigned long previous_elapsed_time = 0;

bool press_started = false;
bool is_long_pressed = false;
bool is_short_pressed = false;

// GLOBALS FOR STATE
int screen_number = 0;
int mode = 1;
int limit = 1;
int op = 1;
float value = ERROR;
```

```
bool use_custom_limit = false;
int custom_limit_serial = 0;
int custom_limit = 0;

// DEVICES
Adafruit_AM2320 am2320 = Adafruit_AM2320(); // temp/humidity sensor
Servo myservo; // servo
LiquidCrystal_I2C lcd(LCD_ADDRESS, 20, 4); // lcd

void setup() {
  Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT);
  pinMode(SERVO_PIN, OUTPUT);

  // Curernt ms as elapsed time
  previous_elapsed_time = millis();

  // Init temp/humidity sensor
  am2320.begin();

  // Init servo
  myservo.attach(SERVO_PIN);

  // Init lcd
  lcd.init();
  lcd.backlight();
}

void loop() {
  // uncomment for debug printing
  // debugPrint();

  // Read custom value from Serial
  if (Serial.available() > 0) {
    String input_line = Serial.readStringUntil('\n');
    input_line.trim(); // Remove leading/trailing spaces

    // Convert the input to an integer
    int temp_value = input_line.toInt();

    if (temp_value >= -999 && temp_value <= 999) {
      custom_limit = temp_value;
      use_custom_limit = true;
      Serial.print("Accepted int: ");
      Serial.println(custom_limit);
    } else {
      Serial.println("Value out of range. Must be between -999 and 999.");
    }
  }
```

```
  // Grab the button value and check press dration
  int btn_value = digitalRead(BUTTON_PIN);
  checkButtonPress(btn_value);

  // Change screen number on long press
  if (is_long_pressed) {
    screen_number += 1;
  }

  // Show screen based on screen_number
  if (screen_number == 0) {
    defaultScreen();
  } else if (screen_number == 1) {
    modeConfigScreen();
  } else if (screen_number == 2) {
    limitConfigScreen();
  } else if (screen_number == 3) {
    operatorConfigScreen();
  } else {
    screen_number = 0;
    defaultScreen();
  }

  // Move servo if in default view (screen_number == 0)
  // if evaluate() true -> move to 270, if evaluate() false -> reset to 0
  if (screen_number == 0) {
    bool move_servo = false;
    if (use_custom_limit) {
      move_servo = compare(value, custom_limit);
    } else {
      move_servo = compare(value, LIMITS[limit - 1]);
    }
    if (move_servo) {
      myservo.write(270);
      delay(100);
    } else {
      myservo.write(0);
      delay(100);
    }
  }

  // Reset button press and add small delay
  is_short_pressed = false;
  is_long_pressed = false;
  delay(10);
}

// The default screen showing:
// Title, selected mode with operator, selected limit and live sensor reading for
selected mode
```

```
void defaultScreen() {

  lcd.setCursor(0,0);
  lcd.print(TITLE_TEXT);
  lcd.setCursor(0,1);
  lcd.print(TOOLTIP_TEXT);

  // Print current config info. Examples:
  // "mode: H < limit: 30" (humidity less than 30)
  // "mode: T > limit: 10" (temperature more than 10)
  lcd.setCursor(0,2);
  lcd.print("mode: ");
  lcd.print(MODES[mode - 1][0]);
  lcd.print(" ");
  lcd.print(OPERATORS[op - 1][0]);
  lcd.print(" limit:");
  if (use_custom_limit) {
    printLimit(custom_limit);
  } else {
    printLimit(LIMITS[limit-1]);
  }

  // Read value based on selected mode
  value = ERROR;
  if (mode == 1) {
    value = am2320.readTemperature();
  } else if (mode == 2) {
    value = am2320.readHumidity();
  }

  // Print current sensor reading for selected mode
  lcd.setCursor(0,3);
  lcd.print(CURRENT_VALUE_TEXT);
  lcd.print(value);
}

// Config screen for changing selected mode
void modeConfigScreen() {
  mode = cycleOnShortPress(mode, MODE_COUNT);
  lcd.setCursor(0,0);
  printConfigInfo("    MODE  CONFIG    ");
  lcd.setCursor(0,3);
  lcd.print("mode:               ");
  lcd.setCursor(6,3);
  lcd.print(MODES[mode - 1]);
}

// Config screen for changing selected limit
void limitConfigScreen() {

  if (is_short_pressed) {
```

```
    use_custom_limit = false;
  }

  limit = cycleOnShortPress(limit, LIMIT_COUNT);
  lcd.setCursor(0,0);
  printConfigInfo("    LIMIT CONFIG    ");
  lcd.setCursor(0,3);
  lcd.print("limit:              ");
  lcd.setCursor(7, 3);
  lcd.print(LIMITS[limit - 1]);
}

// Config screen for changing selected operator
void operatorConfigScreen() {
  op = cycleOnShortPress(op, OPERATOR_COUNT);
  printConfigInfo("  OPERATOR  CONFIG  ");
  lcd.setCursor(0,3);
  lcd.print("operator:           ");
  lcd.setCursor(10, 3);
  lcd.print(OPERATORS[op - 1]);
}

// Function for cycling array indexes on short press
int cycleOnShortPress(int val, int count) {
  if (is_short_pressed) {
    val++;
    if (val > count) {
      val = 1;
    }
  }
  return val;
}

// Function for compares given values using globally set operator
bool compare(float value, int value2) {
  if (value == ERROR) {
    return false;
  }

  if (op == 1) {
    return (value < value2);
  } else if (op == 2) {
    return (value > value2);
  } else {
    return false;
  }
}

// Calls either buttonDown or buttonUp based on btn_value.
void checkButtonPress(int btn_value) {
  if (btn_value == LOW) {
```

```
    buttonDown();
  } else {
    if (press_started) {
      buttonUp();
    }
  }
}

// Acts like an event for button down.
// Starts or continues tracking the time button has been pressed
void buttonDown() {
  if (press_started == false) {
    elapsed_time = 0;
    press_started = true;
    elapsed_time = 0;
    previous_elapsed_time = millis();
  } else {
    elapsed_time = millis() - previous_elapsed_time;
  }
}

// Acts like an event for button up.
// Decides if press was either short or long based on time tracked
void buttonUp() {
  press_started = false;
  is_short_pressed = false;
  is_long_pressed = false;
  if (elapsed_time < LONG_PRESS_TIME) {
    is_short_pressed = true;
  } else {
    is_long_pressed = true;
  }
}

// Prints the limit with needed whitespace around it.
void printLimit(int val) {
  if (val >= 0) {
    lcd.print(" ");
  }
  lcd.print(val);
  if (val < 100 && val > -100) {
    lcd.print(" ");
  }
  if (val < 10 && val > -10) {
    lcd.print(" ");
  }
}

// Prints given title + tooltips for a config screen
void printConfigInfo(String title) {
  lcd.setCursor(0,0);
```

```
  lcd.print(title);
  lcd.setCursor(0,1);
  lcd.print(LONG_TOOLTIP_TEXT);
  lcd.setCursor(0,2);
  lcd.print(SHORT_TOOLTIP_TEXT);
}

// Debug printing of important values
void debugPrint() {
  Serial.println("");
  Serial.print("Value ");
  Serial.print(value);
  Serial.print(", Limit ");
  Serial.print(LIMITS[limit - 1]);
  Serial.print(", Op ");
  Serial.print(OPERATORS[op - 1]);
  Serial.print(" ::: ");
  Serial.print(compare(value, LIMITS[limit - 1]));
}
```