Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Kristijan Štuhec

Nadzorni sistem za rastlinjak

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Patricio Bulić

Ljubljana, 2019

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Načrtujte nadzorni sistem za majhen rastlinjak. Sistem naj meri vlažnost tal, osvetlitev ter temperaturo zraka. Omogoča naj avtomatsko zalivanje tal in osvetljevanje. Krmilni del implementirajte na vgrajenem sistemu STM32F4. Poleg krmilnega dela implementirajte še spletni strežnik na vgrajenem sistemu ESP8266 ter aplikacijo za osebni računalnik. Aplikacija naj omogoča spremljanje stanja v rastlinjaku ter nastavitev parametrov za krmilnik.

Zahvala prijatelju Jonu Muhoviču za tehnično in moralno podporo, zahvala za tehnično pomoč asistentoma Anžetu Rezlju in Juretu Demšarju, zahvala očetu Branetu Senegačniku za lektoriranje, zahvala mentorju Patriciu Buliću, zahvala družini za moralno podporo.

Diplomsko nalogo posvečam babici Miri in dedku Marjanu.

"I've failed over and over and over again in my life. And that is why I succeeded."
– Michael Jordan

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
HTML	Hyper text markup language	jezik za označevanje nadbesedila
LED	Light-emitting diode	svetleča dioda
GPIO	General-purpose input/output	splošno-namenski vhod/izhod
USART	Universal synchronous and asynchronous receiver transmitter	univerzalni sinhroni asinhroni sprejemnik oddajnik
NVIC	Nested Vectored Interrupt Controller	krmilnik za gnezdene vektorske prekinitve
PSP	Interrupt service routine	prekinitveno servisni program
PWM	Pulse-width modulation	pulzno-širinska modulacija
ROM	Read-only memory	bralni pomnilnik
RTOS	Real-time operating system	Realno-časovni operacijski sistem
AWT	Abstract Window Toolkit	Ogrodje za abstrakcijo oken
GUI	Graphical User Interface	grafični uporabniški vmesnik
ADC	Analog to Digital Converter	analogno-digitalni pretvornik

Povzetek

Naslov: Nadzorni sistem za rastlinjak

Avtor: Kristijan Štuhec

Cilj diplomskega dela je bil razviti nadzorni sistem za rastlinjak, ki avtomatizira oskrbovanje rastlin. Nadzorni sistem skrbi za optimalen nivo vode, za optimalno osvetlitev, ki je nastavljiva, ter za ustrezno temperaturo v rastlinjaku. Pri načrtovanju smo poskrbeli, da je sistem modularen - z minimalnimi prilagoditvami je primeren tako za manjša korita kot za večje rastlinjake. Nadzorni sistem je sestavljen iz razvojne ploščice STM32F4 discovery, ki je povezana z brezžičnim modulom ESP8266. Nanj smo namestili strežnik, na katerem so podatki o parametrih nadzornega sistema prikazani v obliki HTML strani. Podatke smo s strežnika prebrali z javanskim programskim vmesnikom, jih obdelali in v realnem času prikazali na grafu.

Ključne besede: STM32F4 Discovery, rastlinjak, nadzorni sistem, ESP8266, Java.

Abstract

Title: Greenhouse control systems

Author: Kristijan Štuhec

The aim of the diploma thesis was to develop a control system for a greenhouse, which would automate the care for the plants. The control system ensures an optimal water level, optimal and adjustable lighting and ensures that the temperature in the greenhouse is appropriate. During the design we made sure that the system is modular - with minimal adjustments it is suitable for both a small flowerpot or a larger greenhouse. The control system consists of the STM32F4 discovery development board, which is connected to the wireless module ESP8266. On it we installed a server on which the parameters of the control system are displayed in the form of a HTML page. The data is then read from it with the Java program interface, processed and displayed in a graph in real time.

Keywords: STM32F4 Discovery, greenhouse, control system, ESP8266, Java.

Poglavje 1

Uvod

Z avtomatizacijo se danes srečujemo bolj ali manj na vseh področjih življenja. Kmetijstvo je kljub velikemu tehničnemu napredku še vedno precej odvisno od vremena. Zaradi suše, pozebe in poplav vsako leto propade veliko pridelka, kljub temu pa je zaščita le-tega sorazmerno slabo razvita. Rastline za rast potrebujejo dovolj sončne svetlobe, vodo in hranilne snovi. V diplomski nalogi smo se osredotočili na oskrbo s svetlobo in vodo, saj je ta potrebna vsak dan, medtem je oskrba s hranilnimi snovmi (gnojenje) manj zahtevna in jo lahko opravimo na primer enkrat na leto. V našem sistemu se osvetljevanje izvaja s svetili LED, dolžina le-tega pa je odvisna od vrste rastline. Odločili smo se za osvetljevanje osem ur dnevno, kar je primerno za večino rastlin (osvetljevanje lahko prilagodimo tudi glede na to, ali je rastlinjak na mestu, kjer rastline prejemajo tudi dnevno sončno svetlobo). Zalivanje rastlin je zelo pomembno, saj lahko rastlina v zelo kratkem času celo odmre, če je v zemlji, v kateri raste, vode preveč ali premalo. Zato je zelo pomembno vzdrževati konstantno raven vode v zemlji ter skrbeti, da je zemlja po celi posodi enakomerno vlažna.

Poglavje 2

Uporabljene tehnologije

2.1 STM32F407 discovery

Naš nadzorni sistem poganja cenovno ugodna razvojna ploščica STM32F407 discovery, ki jo je razvilo podjetje STMicroelectronics (Slika 2.1). Glavne značilnosti STM32F407 so:

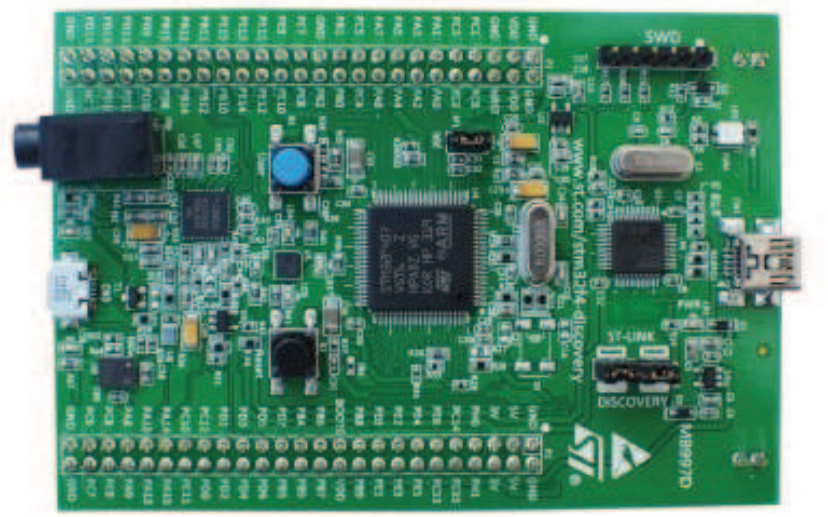
- ARM Cortex-M4 procesor s frekvenco 168 MHz
- 1 MB pomnilnika
- 192 kB predpomnilnika
- 14 časovnikov
- 82 splošno namenskih vhodno-izhodnih pinov
- 15 komunikacijskih vmesnikov

STM32F407 se napaja s 5 V preko USB-ja [20].

2.1.1 Splošnonamenske vhodno-izhodne naprave

Razvojna ploščica STM32F407 discovery nam omogoča kontrolo 82 vhodno-izhodnih pinov [21] (Slika 2.1). Pine upravljamo s pomočjo štirih kontrolnih registrov, dveh podatkovnih registrov in dveh registrov alternativne funkcije. S kontrolnimi registri pinu nastavimo želeno funkcionalnost. Seznam kontrolnih registrov:

- `GPIOx_MODER` - nastavljanje smeri



Slika 2.1: Razvojna ploščica STM32F407 Discovery [21].

- `GPIOx_OTYPER` - nastavljanje vrste izhoda
- `GPIOx_OSPEEDR` - nastavljanje hitrosti
- `GPIOx_PUPDR` - nastavljanje pull up/pull down upora

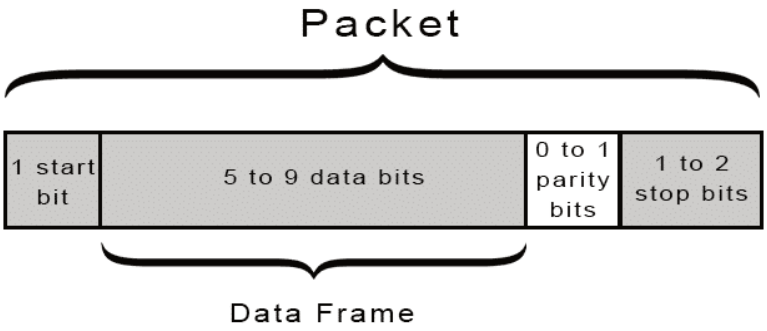
S podatkovnima registroma lahko nastavimo vrednost pina ali pa vrednost preberemo. Seznam podatkovnih registrov:

- `GPIOx_IDR` - preberemo vrednost na pinu
- `GPIOx_ODR` - nastavimo vrednost pina

2.1.2 Serijska komunikacija

Za komunikacijo s strežnikom uporabljamo USART - univerzalni sinhroni-asinhroni sprejemnik-oddajnik (angl. Universal synchronous asynchronous receiver transmitter). Za prenos uporabimo splošno namenski vhodno-izhodni pin, ki mu določimo alternativno funkcijo, kar nam omogoči USART prenos preko njega. Pred prenosom moramo nastaviti več parametrov, med pomembnejšimi so:

- dolžina znaka
- pariteta



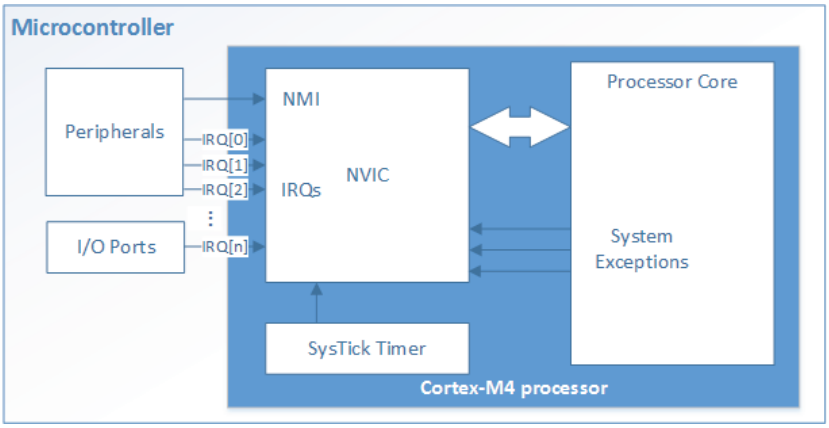
Slika 2.2: Prenos podatkov USART [22].

- število stop bitov
- število pulzov na sekundo

Pred prenosom je potrebno vklopiti uro na vodilu. Prenos podatkov se začne s start bitom. Sledi 5 - 9 podatkovnih bitov ter paritetni bit, v kolikor se odločimo za njega. Prenos se zaključi s stop bitom. Na obeh napravah, udeleženih pri USART prenosu mora biti nastavljeno enako število pulzov na sekundo (angl. Baud rate). Sestava prenosa USART je prikazana na Sliki 2.2.

2.1.3 Prekinitveni krmilnik

Procesor mora dogodke zaznavati, da se lahko nanje odzove. Najbolj enostavno je, da v zanki (angl. polling) preverjamo ali se je dogodek zgodil. Perioda preverjanja mora biti dovolj kratka, da je zakasnitev odziva kar se da majhna, vendar pa tako preverjanje zahteva veliko procesorskega časa. Za razbremenitev procesorja uporabimo prekinitve. Za prekinitve na STM32F407 skrbi NVIC (angl. Nested vectored interrupt controller) in sicer z vektorskimi prekinitvami. Omogoča prekinitev na 82 prekinitvenih kanalih. Na vsakem lahko nastavimo enega od 256 prioritetnih nivojev [21]. Zaradi tesne integracije procesorja in NVIC (Slika 2.3) je zakasnitev ob prekinitvah zelo majhna. Ob nastavitvi smo podali prekinitveni vir in prioriteto. Za

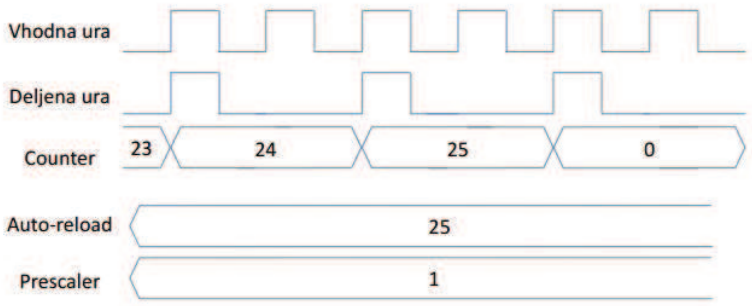


Slika 2.3: STM32F4 NVIC [16].

vsako želeno prekinitev smo napisali PSP (prekinitveno servisni program): funkcijo, ki v primeru prekinitve izvede želeno akcijo. Prekinitveni vir s spremembo stanja na signalu NVIC sporoči prekinitveno zahtevo. Ta nato procesorju sporoči številko prekinitvenega vektorja, procesor pa izvede želeni PSP. Procesor v tem primeru ne preverja, ali je do prekinitve prišlo, ampak samo izvede PSP, ko je do prekinitve že prišlo.

2.1.4 Časovniki

Časovniki se uporabljajo za izvajanje operacij ob natančnih intervalih, merjenje časa in generiranje signala PWM (angl. Pulse-width modulation). STM32 ima dva napredna, šest splošnonamenskih in dva osnovna časovnika. Štetje poteka v 16-bitnem številnem registru, štejemo lahko navzgor, navzdol ali v obe smeri. Frekvenca štetja je odvisna od izbire vhodne ure in nastavitve delilnika. Štetje se ponovno začne, ko preštejemo do vrednosti zapisane v ARR registru (angl. auto reload register). Takrat se sproži prekinitev, ki nam pove, da je vrednost ARR registra dosežena, vrednost številnega registra se nastavi na 0 in štetje se začne od začetka [8, 21] (Slika 2.4).



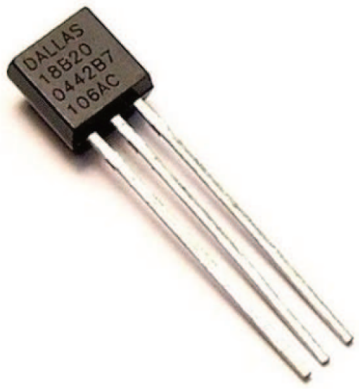
Slika 2.4: Časovni diagram - štetje navzgor [8].

2.1.5 Analogno-digitalni pretvornik

Analogno-digitalni pretvornik se uporablja za merjenje analogne vrednosti ter njeno pretvorbo v digitalno vrednost. STM32 ima tri 12-bitne ADC naprave, vsaka ima 16 vhodov [9]. Vse tri naprave so povezane na vodilo APB2. Rezultat pretvorbe se hrani v levo ali desno poravnanem 16-bitnem registru. ADC nam omogoča enkratno pretvorbo ali stalno pretvorbo [21].

2.2 Razvojno okolje IAR Embedded Workbench

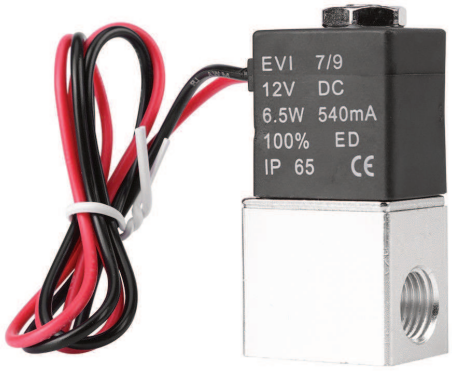
V pomoč pri razvoju nadzornega sistema nam je bil IAR Embedded Workbench, IDE okolje za programiranje mikrokrmilnikov [10]. IAR Embedded Workbench vključuje prevajalnik, razhroščevalnik, povezovalnik ter omogoča pregled stanja registrov. Programirali smo v programskem jeziku C, IAR Embedded Workbench pa je skrbel za prenos kode na STM32F4. Ker je razvojno okolje plačljivo, smo izbrali poskusno verzijo, omejeno z velikostjo kode, kar je zadoščalo za naše potrebe.



Slika 2.5: Temperaturi senzor Dallas DS18B20 [3].

2.3 Temperaturni senzor Dallas DS18B20

Za merjenje temperature smo uporabili temperaturi senzor Dallas DS18B20 (Slika 2.5), proizvajalca Maxim Integrated [4]. Senzor omogoča merjenje temperature od $-55\,°C$ do $125\,°C$ z natančnostjo $0.5\,°C$ od $-10\,°C$ do $85\,°C$. Komunikacija s senzorjem poteka po 1-wire protokolu preko zgolj enega pina, kar precej poenostavi njegovo uporabo. Obenem lahko po potrebi na en pin priklopimo poljubno mnogo senzorjev, zaradi česar je nadzorni sistem enostavno razširiti in uporabiti za večji rastlinjak. Vsaka 1-wire naprava ima unikatno 64-bitno kodo, ki jo nastavi proizvajalec in je uporabnik ne more spremeniti. Komunikacija se začne z dolgim nizkim pulzom (inicializacijski pulz) [1], ki ga proži glavna naprava - s tem se resetirajo vse podrejene naprave. Podrejena naprava nato javi svojo prisotnost s kratkim nizkim pulzom (prisotnostni pulz). Nato se lahko začne pošiljanje podatkov med napravama. Glavna naprava pošlje logično 1 z zelo kratkim nizkim pulzom in logično 0 s kratkim nizkim pulzom. Podrejena naprava pošlje logično 1 tako da ne naredi ničesar ter logično 0 s kratkim nizkim pulzom.



Slika 2.6: Elektromagnetni ventil [23].

2.4 Elektromagnetni ventil

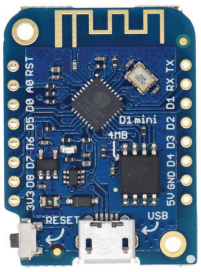
Za zalivanje rastlin skrbi generični 12 V elektromagnetni ventil [23] z močjo 6.5 W, ki omogoča delovanje med −10 °C in 80 °C. Ventil je prikazan na Sliki 2.6. Zaradi varnosti smo izbrali ventil, ki je v stanju brez napetosti zaprt, s tem namreč preprečimo, da bi ob morebitnem izpadu električne energije voda poplavila. Glavni sestavni deli ventila so bat, vzmet in tuljava. V stanju brez napetosti vzmet potisne bat navzdol in zapre ventil. Ko ventil priključimo na napetost, tuljava dvigne bat in ventil se odpre. Ventil dopušča do 1.2 MPa pritiska, kar pomeni da lahko uporabljamo tudi zelo velik zbiralnik vode, če oskrbujemo večje površine. Elektromagnetni ventili se uporabljajo na področjih, kjer je potrebna kontrola pretoka tekočine ali plina [19].

2.5 Svetila LED

Rastline za rast potrebujejo svetlobo. Najpomembnejši sta rdeča (650 nm-730 nm) in modra (400 nm-500 nm) svetloba [17]. Razmerje je odvisno od

Slika 2.7: Svetila LED [18].

rastline in tega ali želimo, da rastlina cveti zgodaj ali pozno. Uporaba samo rdeče svetlobe povzroči, da rastlina zraste više in v nekaterih primerih precej zakasni s cvetenjem. Uporaba večje količine modre svetlobe v kombinaciji z rdečo svetlobo pa povzroči, da so rastline bolj kompaktne in cvetijo predčasno. Tako bi pri rastlini, ki jo uporabljamo samo za pridobitev semen, uporabljali večjo količino modre svetlobe. V kolikor bi rastlino uporabljali za pridelovanje plodov, pa bi uporabili manjšo količino modre svetlobe. Za osvetljevanje rastlin skrbijo 12 V LED svetila z modro in rdečo svetlobo v razmerju 3:1 [18]. Odločili smo se za trakove LED (Slika 2.7) saj to precej olajša osvetljevanje v večjem rastlinjaku - dodatne trakove priključimo na obstoječe in tako osvetljujemo večjo površino. Posamezne žarnice LED so postavljene zaporedno, tako, da so v vrsti 3 rdeče ter 1 modra. Trakove LED lahko kadarkoli zamenjamo s trakovi, pri katerih je razmerje med rdečo in modro svetlobo drugačno, v kolikor je to potrebno.



Slika 2.8: ESP8266 [7].

2.6 ESP8266

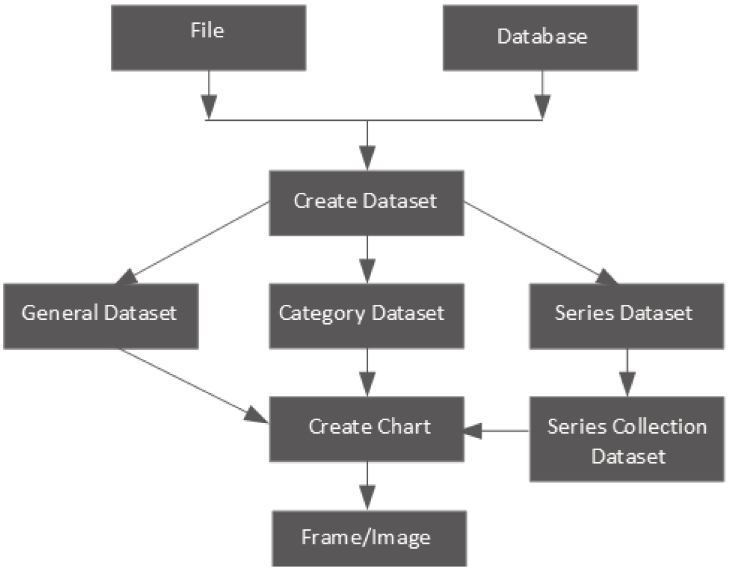
ESP8266 je kompaktna razvojna ploščica proizvajalca Espressif Systems, ki smo jo uporabili za naš strežnik [6] (Slika 2.8). Glavne značilnosti ESP8266 so:

- processor Tensilica L106 32-bit RISC
- do 16 MB zunanjega pomnilnika
- brezžični sprejemnik in oddajnik
- 17 splošno namenskih vhodno-izhodnih pinov

Ploščica deluje v širokem temperaturnem območju od $-40\,°C$ do $125\,°C$. Modul ima na voljo večje število GPIO pinov, podpira USART in 802.11 b/g/n protokole. Napaja se preko USB-ja. RTOS (angl. real time operating system) in wi-fi sklad porabita okoli 20 odstotkov procesorskega časa, tako da ga 80 odstotkov ostane za uporabnikov program. Delo si olajšamo s knjižnicami za Arduino, ki jih ploščica podpira.

2.7 Java

Java je visokonivojski objektno usmerjeni programski jezik, ki smo ga uporabili za izdelavo programskega vmesnika. Za Javo smo se odločili, ker omogoča preprost dostop in branje podatkov s spletne strani. Pri izdelavi grafičnega uporabniškega vmesnika (angl. graphical user interface) smo se odločili za



Slika 2.9: JFreeChart struktura [11].

orodja Swing [2]. V primerjavi z orodji AWT [2] ima nekaj prednosti: ne uporablja elementov operacijskega sistema - izgled je platformno neodvisen. Za izris grafov smo si pomagali z brezplačno knjižnico jFreeChart [12], ki precej poenostavlja prikaz grafov v realnem času. Knjižnica podpira orodja Swing, kar nam je olajšalo implementacijo. jFreeChart je danes najbolj uporabljana knjižnica za grafe v programskem jeziku Java. Osnova je enaka za vse vrste grafov: na osnovi svojih podatkov sestavimo podatkovno zbirko, ki jo kasneje uporabimo v grafu. Kakšno podatkovno zbirko bomo izbrali, je odvisno od želenega grafa. Za ustvarjanje grafov skrbi ChartFactory [12], ki vsebuje metode za ustvarjanje vseh vrst grafov. Tako nastali graf nato enostavno vstavimo v JFrame. Diagram poteka je prikazan na Sliki 2.9.

Poglavje 3

Načrtovanje in izdelava nadzornega sistema

3.1 Merjenje nivoja vode

Namenski merilec vlažnosti je sestavljen iz dveh sond, medsebojno oddaljenih 1 cm, ki se ju zapiči v zemljo [14] (Slika 3.2). Senzor nato meri prevodnost med sondama. Ko je zemlja ustrezno vlažna, steče tok med sondama in tako vemo, da je zemlja ustrezno namočena.

Namenski merilec vlažnosti pa ima pomanjkljivost: vlažnost meri na zelo majhni površini. To rešimo tako, da sondi postavimo daleč narazen, natančneje vsako sondo na en konec posode. To pomeni, da bo ustrezen nivo vode zaznan šele, ko bo vsa zemlja v posodi ustrezno vlažna. Eno sondo priključimo na 3 V pin na razvojni ploščici, drugo sondo pa na ADC. V nastavitvah ADC smo izbrali resolucijo 6 bitov. To pomeni, da nam ADC analogno izmerjeno napetost preslika v digitalno vrednost med 0 in 63. Najprej smo morali zemljo ustrezno namočiti [24] in nato z ADC izmeriti vrednost, ki nam jo ta vrača, da bi lahko v sistemu nastavili mejo, pri kateri mora nadzorni sistem zemljo zaliti. Zemljo smo dali v posodo z luknjami, jo zalili ter pustili 30 minut, da je odvečna voda odtekla. Zemljo smo nato prestavili v našo posodo in s pomočjo ADC izmerili vrednost 50, ki smo jo nato postavili za

Slika 3.1: Naša rešitev merjenja nivoja vode.

referenčno vrednost. Zalivalni sistem nato periodično preverja vrednost na pinu. Če je zaznana vrednost nad 50, je nivo vode ustrezen, v nasprotnem primeru je potrebno zalivanje (Slika 3.1).

3.2 Zalivanje

Za zalivanje zemlje smo uporabili 12 V elektromagnetni ventil (opisan v Poglavju 2.4), ki smo ga povezali s posodo z vodo in posodo z zemljo. Posodo z vodo smo dvignili 25 mm nad posodo z zemljo. S tem smo se izognili potrebi po vodni črpalki, saj za pretok poskrbi gravitacija. Cev z luknjami smo zvili v spiralo, tako da je pokrivala dno posode, kot prikazuje Slika 3.1. Rele smo povezali s ventilom in razvojno ploščico (Slika 3.4). Na začetku smo GPIO pin postavili v nizko napetostno stanje in tako poskrbeli, da je bil ventil zaprt. Ko je nadzorni sistem javil neustrezen nivo vode, smo GPIO pin postavili v visoko napetostno stanje. Tako je tok stekel skozi rele in odprl ventil.

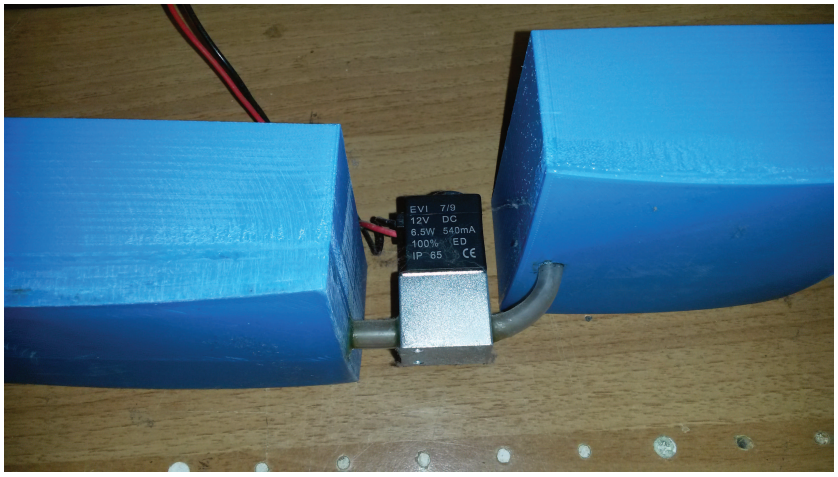
Slika 3.2: Merilec vlažnosti zemlje meri vlažnost na zelo majhni površini [15].

```
void zalivanje()
{
    ADC_SoftwareStartConv(ADC1);
    while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC));

    volatile uint16_t a;
    a = ADC_GetConversionValue(ADC1);

    if(a < 50)  {
        //odpremo ventil
        ventil = 1;
        sprintf(buf1, "%d", ventil);
        GPIO_WriteBit(GPIOD, GPIO_Pin_1, Bit_SET);
    }
    else if (a > 50) {
        //zapremo ventil
        ventil = 0;
        sprintf(buf1, "%d", ventil);
        GPIO_WriteBit(GPIOD, GPIO_Pin_1, Bit_RESET);
    }

}
```

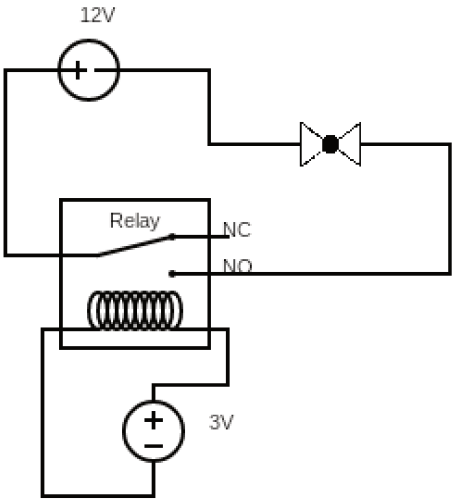


Slika 3.3: Elektromagnetni ventil, umeščen med posodo z zemljo in posodo z vodo.

3.3 Merjenje temperature zraka

Za merjenje temperature zraka smo uporabili senzor Dallas DS18B20 (opisan v Poglavju 2.3), ki smo ga priključili na razvojno ploščico STM32. Senzor za komunikacijo uporablja 1-wire protokol (opisan v Poglavju 3.3). Med 5 V in podatkovni pin smo namestili 4700 ohmski upor (Slika 3.5). S tem smo podatkovni pin dvignili na visoko napetostno stanje, kar je nujno potrebno za komunikacijo med razvojno ploščico in senzorjem. Da lahko vzpostavita medsebojno komunikacijo ga morata razvojna ploščica in senzor namreč spustiti na nizko napetostno stanje. Za poenostavitev merjenja smo si pomagali s knjižnicami [13], namenjenimi našemu senzorju[1]. Temperaturo smo merili enkrat na sekundo, odčitano vrednost pa pošiljali na strežnik. Kot pri vseh 1-wire napravah je bila najprej potrebna inicializacija naprave, šele nato pa se je lahko začelo odčitavanje temperature.

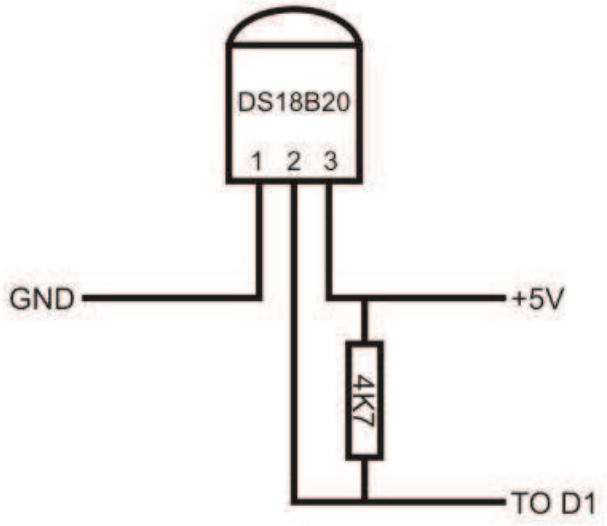
[1]Knjižnice so dostopne na: `https://stm32f4-discovery.net/`



Slika 3.4: Shema povezave releja in elektromagnetnega ventila.

```
void temperaturniSenzor()
{
    //inicializacija
    uint8_t device[1][8];
    TM_OneWire_t OneWire1;
    SystemInit();
    TM_DELAY_Init();
    TM_OneWire_Init(&OneWire1, GPIOD, GPIO_Pin_0);
    TM_OneWire_First(&OneWire1);
    TM_OneWire_GetFullROM(&OneWire1, device[0]);
    resolucija = TM_DS18B20_Resolution_12bits
    TM_DS18B20_SetResolution(&OneWire1, device[0], resolucija);

    //branje temperature
    while (1) {
        TM_DS18B20_StartAll(&OneWire1);
        while (!TM_DS18B20_AllDone(&OneWire1));
              if (TM_DS18B20_Read(&OneWire1, device[0], &temps[0])) {
                  sprintf(buf, "%3.2f", temps[0]);
              }
```

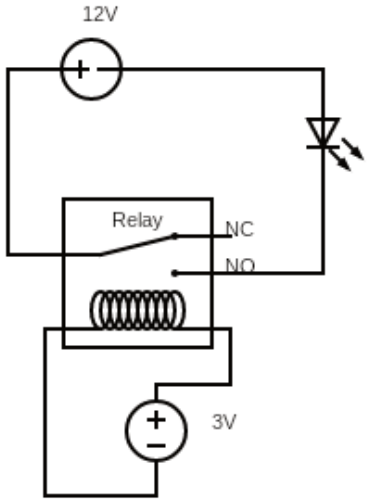


Slika 3.5: Shema priklopa senzorja [5].

```
21      }
22  }
```

3.4 Osvetljevanje

Za osvetljevanje smo uporabili namenska 12 V svetila LED moči 15 W. Po naši osnovni nastavitvi so žarnice 8 ur svetile, nato pa so se za 16 ur izklopile. Omogočili smo tudi spreminjanje te vrednosti preko programskega vmesnika. Ker žarnice oddajajo precej toplote, smo implementirali zaščito, ki jih je izklopila, ko je bila določena temperatura prekoračena. Privzeta mejna vrednost je bila 50 °C, ker pa je prag odvisen od posamezne rastline, smo omogočili spreminjanje te vrednosti preko programskega vmesnika. Podobno kot vodni ventil smo tudi žarnice krmilili z relejem, ki smo ga povezali med vir napetosti in led svetila (Slika 3.6).



Slika 3.6: Shema povezave releja in led svetil.

```
1  void InitializeLedLights()
2  {
3      //vklopimo uro
4      RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
5
6      //nastavitev in inicializacija strukture
7      GPIO_InitTypeDef GPIO_LED;
8      GPIO_LED.GPIO_Pin = GPIO_Pin_2;
9      GPIO_LED.GPIO_Mode = GPIO_Mode_OUT;
10     GPIO_LED.GPIO_OType = GPIO_OType_PP;
11     GPIO_LED.GPIO_Speed = GPIO_Speed_50MHz;
12     GPIO_Init(GPIOD, &GPIO_LED);
13
14     //Vedno zacnemo z obdobjem prižganih led luci
15     GPIO_WriteBit(GPIOD, GPIO_Pin_2, Bit_SET);
16 }
17
18 void osvetljevanje()
19 {
20     sprintf(buf2, "%d", svetlostMode);
```

```
21      if(temps[0] > zeljenaTemperatura) {
22        //prevelika temperatura, ugasamo led luci za najmanj 10 minut.
23        GPIO_WriteBit(GPIOD, GPIO_Pin_2, Bit_RESET);
24        svetlostMode = 0;
25        osvetljevanjeTimeOut = 600;
26      } else if (osvetljevanjeTimeOut > 0) {
27        //ce 10 minutni timeout še ni koncan, ne storimo nic
28      }
29      else if(svetlostMode == 0 && svetlostDolzina >= (24-zeljenoOsv)) {
30        //prizgemo luc
31        svetlostMode = 1;
32        svetlostDolzina = 0;
33        GPIO_WriteBit(GPIOD, GPIO_Pin_2, Bit_SET);
34      } else if(svetlostMode == 1 && svetlostDolzina >= zeljenoOsv) {
35        //ugasnemo luc
36        svetlostMode = 0;
37        svetlostDolzina = 0;
38        GPIO_WriteBit(GPIOD, GPIO_Pin_2, Bit_RESET);
39      }
40  }
41
42  void InitializeTimerHour()
43  {
44      //nastavitev in inicializacija strukture
45      NVIC_InitTypeDef nvicStructure;
46      nvicStructure.NVIC_IRQChannel = TIM4_IRQn;
47      nvicStructure.NVIC_IRQChannelPreemptionPriority = 0;
48      nvicStructure.NVIC_IRQChannelSubPriority = 1;
49      nvicStructure.NVIC_IRQChannelCmd = ENABLE;
50      NVIC_Init(&nvicStructure);
51
52      //vklopimo uro
53      RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);
54
55      //nastavitev in inicializacija strukture
56      TIM_TimeBaseInitTypeDef timerInitStructure;
57      timerInitStructure.TIM_Prescaler = 8400-1;
58      timerInitStructure.TIM_CounterMode = TIM_CounterMode_Up;
59      timerInitStructure.TIM_Period = 10000;
60      timerInitStructure.TIM_ClockDivision = 0;
```

```
61      timerInitStructure.TIM_RepetitionCounter = 0;
62      TIM_TimeBaseInit(TIM4, &timerInitStructure);
63      TIM_Cmd(TIM4, ENABLE);
64      TIM_ITConfig(TIM4, TIM_IT_Update, ENABLE);
65  }
66
67  void TIM4_IRQHandler()
68  {
69      if (TIM_GetITStatus(TIM4, TIM_IT_Update) != RESET) {
70          TIM_ClearITPendingBit(TIM4, TIM_IT_Update);
71          //preverimo ali je 10 minutni timeout minil
72          if(osvetljevanjeTimeOut > 0) {
73            osvetljevanjeTimeOut--;
74            if(osvetljevanjeTimeOut == 0) svetlostMode = 1;
75          } else {
76            stevec++;
77          }
78      }
79      // preverimo ali je že potekla 1 ura
80      if(stevec >= 3600) {
81        svetlostDolzina++;
82        stevec = 0;
83      }
84  }
```

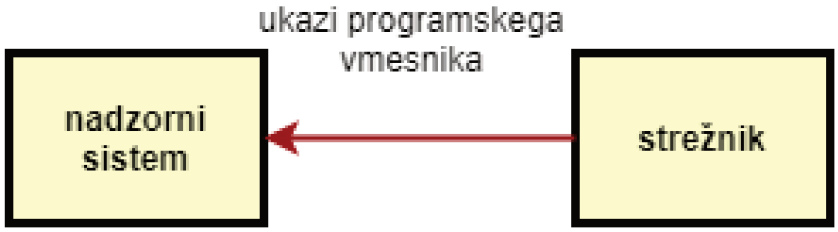
3.5 Pošiljanje podatkov na strežnik

Podatke o temperaturi, stanju ventila in stanju svetil smo pošiljali na naš strežnik (Slika 3.7). Za to smo uporabili serijski vmesnik USART (opisan v poglavju 2.1.2). Podatke smo pošiljali kot posamezne znake (angl. char). Prenos smo začeli z nizom “STM32“, ki programskemu vmesniku pove, da gre za legalen prenos in ne za napako. Nato smo začeli s pošiljanjem vrednosti, ki smo jih ločili s podpičjem, da jih je programski vmesnik laže posamič prepoznal, saj tako težje pride do napak. Podatke smo pošiljali enkrat na sekundo.

Slika 3.7: Posiljanje podatkov na strežnik.

```
1  void InitializeUsart()
2  {
3
4     //nastavitev in inicializacija strukture
5      NVIC_InitTypeDef nvicStructure;
6      nvicStructure.NVIC_IRQChannel = USART1_IRQn;
7      nvicStructure.NVIC_IRQChannelPreemptionPriority = 0;
8      nvicStructure.NVIC_IRQChannelSubPriority = 1;
9      nvicStructure.NVIC_IRQChannelCmd = ENABLE;
10     NVIC_Init(&nvicStructure);
11
12
13
14     //vklopimo uro
15     RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
16     RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
17
18     //nastavitev in inicializacija strukture
19     GPIO_InitTypeDef GPIO_InitStruct;
20     GPIO_InitStruct.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
21     GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
22     GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
23     GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
24     GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
25     GPIO_Init(GPIOB, &GPIO_InitStruct);
26
27     GPIO_PinAFConfig(GPIOB, GPIO_PinSource6, GPIO_AF_USART1);
28     GPIO_PinAFConfig(GPIOB, GPIO_PinSource7, GPIO_AF_USART1);
29
```

```
30      //nastavitev in inicializacija strukture
31      USART_InitTypeDef USART_InitStruct;
32      USART_InitStruct.USART_BaudRate = 9600;
33      USART_InitStruct.USART_WordLength = USART_WordLength_8b;
34      USART_InitStruct.USART_StopBits = USART_StopBits_1;
35      USART_InitStruct.USART_Parity = USART_Parity_No;
36      USART_InitStruct.USART_HardwareFlowControl
37          =USART_HardwareFlowControl_None;
38      USART_InitStruct.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;
39
40      USART_Init(USART1, &USART_InitStruct);
41      USART_Cmd(USART1, ENABLE);
42
43      USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
44
45  }
46
47  void posiljanje()
48  {
49      while(!USART_GetFlagStatus(USART1, USART_FLAG_TXE));
50
51      char* header = "STM32";
52      char* dodatek = buf;
53      while(*header)   {
54          while(USART_GetFlagStatus(USART1,USART_FLAG_TXE) == RESET);
55          USART_SendData(USART1,*header);
56          header++;
57      }
58      while(USART_GetFlagStatus(USART1,USART_FLAG_TXE) == RESET);
59          USART_SendData(USART1, ';');
60      while(*dodatek)   {
61          while(USART_GetFlagStatus(USART1,USART_FLAG_TXE) == RESET);
62          USART_SendData(USART1,*dodatek);
63          dodatek++;
64      }
65      while(USART_GetFlagStatus(USART1,USART_FLAG_TXE) == RESET);
66          USART_SendData(USART1, ';');
67      while(USART_GetFlagStatus(USART1,USART_FLAG_TXE) == RESET);
68         USART_SendData(USART1,*buf1);
69       while(USART_GetFlagStatus(USART1,USART_FLAG_TXE) == RESET);
```



Slika 3.8: Sprejem ukazov programskega vmesnika preko strežnika.

```
70          USART_SendData(USART1, ';');
71        while(USART_GetFlagStatus(USART1,USART_FLAG_TXE) == RESET);
72          USART_SendData(USART1, *buf2);
73        while(USART_GetFlagStatus(USART1,USART_FLAG_TXE) == RESET);
74          USART_SendData(USART1, ';');
75  }
```

3.6 Sprejem ukazov s programskega vmesnika

Ukaze programskega vmesnika smo prejemali s strežnika preko serijskega vmesnika USART (Slika 3.7). Periodično preverjanje, ali smo prejeli podatke, bi po nepotrebnem obremenjevalo procesor. Zaradi tega smo izkoristili prekinitve (opisane v Poglavju 2.1.3), ki nam jih omogoča STM32. Ko se je sprožila USART prekinitev, smo preverili, ali so v registru USART novi podatki. Če so, jih zapišemo v tabelo. Ker je dolžina podatkov, ki naj bi jih dobili, fiksna, smo natanko vedeli, kdaj smo prejeli vse potrebne podatke.

```
1  void USART1_IRQHandler(void) {
2      if(USART_GetFlagStatus(USART1, USART_FLAG_RXNE)) {
3          data[cnt] = USART_ReceiveData(USART1);
4          cnt++;
5      }
6      if(cnt > 1) {
7        cnt = 0;
8        zeljenaTemperatura = data[0];
```

```
9         zeljenoOsv = data[1];
10      }
11  }
```

Poglavje 4

Načrtovanje in izdelava strežnika

4.1 Vzpostavitev strežnika

Najprej smo se z ESP8266 povezali na Wi-Fi omrežje, uporabili ukaz `WiFi.begin()`, kot parametra smo uporabili uporabniško ime in geslo omrežja. Strežnik smo pognali z ukazom `server.begin()`, z ukazom `server.on()` smo določili funkcijo, ki bo skrbela za procesiranje vhodnih zahtev. Če hočemo strežnik prenašati med različnimi omrežji, moramo vedeti IP naslov strežnika. Preprosta rešitev je, da uporabimo statični IP naslov. Ker je običajno mogoče, da je na omrežje priključenih več naprav, smo izbrali IP naslov 192.168.1.200 - visoka zadnja številka zmanjša možnost, da je v omrežju naprava s prav takim IP naslovom. Tukaj smo predpostavili, da uporabnik sam nastavi IP naslove in s tem poskrbi, da je 192.168.1.200 dosegljiv na lokalnem omrežju.

```
1 void setup() {
2
3   Serial.begin(9600);
4   WiFi.begin("ime", "geslo");
5
```

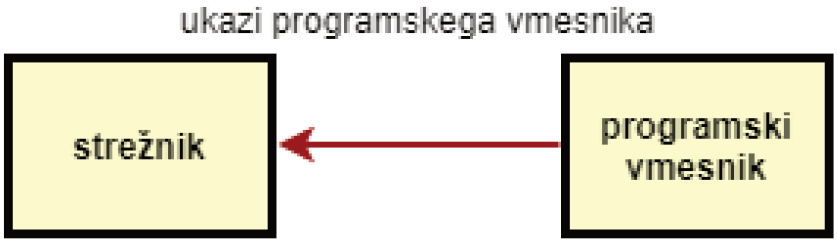
Slika 4.1: Sprejem podatkov nadzornega sistema.

```
6   while (WiFi.status() != WL_CONNECTED) {
7     delay(1000);
8     Serial.println("Connecting..");
9   }
10
11  server.on("/", handleRootPath);
12  server.begin();
13  IPAddress ip(192,168,1,200);
14  IPAddress gateway(192,168,1,1);
15  IPAddress subnet(255,255,255,0);
16  WiFi.config(ip, gateway, subnet);
17 }
```

4.2 Sprejem podatkov nadzornega sistema

Ukaze nadzornega sistema smo prejemali s strežnika preko serijskega vmesnika USART. Na strežniku smo periodično preverjali, ali nam je nadzorni sistem poslal podatke (Slika 3.7). Prejetih podatkov nismo obdelali, ampak smo jih na našem strežniku objavili v prvotni obliki, torej kot spletno stran, obdelavo pa smo prepustili programskemu vmesniku.

```
1  void readSTM() {
2    if (Serial.available() > 0) {
3          receivedStringSTM = Serial.readString();
4    }
5  }
```



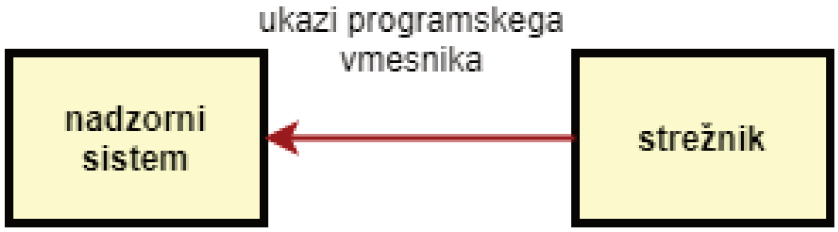
Slika 4.2: Sprejem ukazov programskega vmesnika.

```
6
7  void handleRootPath() {
8     server.send(200, "text/plain", receivedStringSTM);
9  }
```

4.3 Sprejem ukazov programskega vmesnika

Podatke programskega vmesnika je strežnik sprejemal preko zahteve POST. Strežnik je vsakokrat prejel vse parametre, tudi tiste, ki jih v programskem vmesniku nismo spreminjali. To je olajšalo nadaljnje procesiranje podatkov. Strežnik prejetih podatkov ni obdelal sam, temveč jih je poslal programskemu vmesniku v prvotni obliki (Slika 4.2).

```
1  void handleRootPath() {
2      if (server.hasArg("plain")== false){
3              return;
4      }
5      receivedStringJava = server.arg("plain");
6      sendSTM();
7  }
```



Slika 4.3: Pošiljanje ukazov nadzornemu sistemu.

4.4 Pošiljanje ukazov nadzornemu sistemu

Ukaze smo pošiljali preko serijskega vmesnika USART. Da bi zmanjšali obremenitev procesorja, smo jih pošiljali le takrat, kadar smo jih prejeli od programskega vmesnika (Slika 4.3).

```
1 void sendSTM() {
2   Serial.print(receivedStringJava);
3 }
```

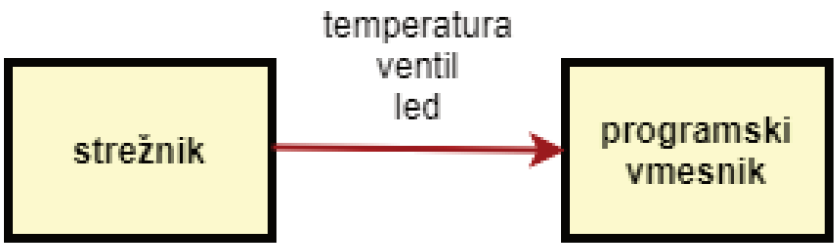
Poglavje 5

Načrtovanje in izdelava programskega vmesnika

5.1 Dostop do strežnika

Do strežnika smo dostopali tako, da smo odprli spletno stran na naslovu 192.168.1.200 (gre seveda za lokalni IP naslov). Če se je spletna stran odprla, smo prebrali prvo vrstico, v kateri so bili podatki nadzornega sistema.

```
1 public static void izpis(String naslov) {
2         URL oracle;
3         oracle = new URL(naslov);
4         BufferedReader in = new BufferedReader(
```



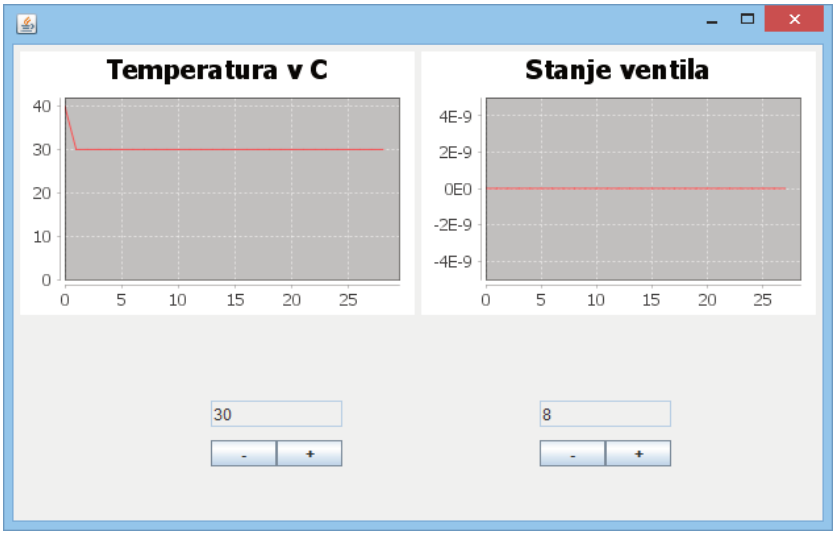
Slika 5.1: Pošiljanje parametrov nadzornega sistema s strežnika do programskega vmesnika.

```
5           new InputStreamReader(oracle.openStream()));
6
7           String inputLine;
8           if((inputLine = in.readLine()) != null) {
9
10                  System.out.println(inputLine);
11                  in.close();
12
13                  // obdelamo podatke
14                  String[] podatki = inputLine.split(";");
15                  gui.trenutnaTemperatura(Double.valueOf(podatki[1]));
16                  gui.trenutnoStanjeVentila(Integer.valueOf(podatki[2]));
17                  gui.zeljtrenutnoStanjeOsv(Integer.valueOf(podatki[3]));
18          }
```

5.2 Prikaz parametrov nadzornega sistema v programskem vmesniku

Preden smo podatke prikazali v programskem vmesniku, smo morali preveriti, ali gre za prave podatke ali pa je nekje na poti med nadzornim sistemom, strežnikom in programskim vmesnikom prišlo do napake. Najprej preverimo, če je prvi del niza enak "STM32". Nadalje preverimo, ali je temperatura v območju, ki ga temperaturni senzor podpira ter če je stanje ventila in stanje žarnic bodisi 0 ali 1. Ko smo potrdili, da so podatki pravilni, smo jih prikazali v programskem vmesniku. Uporabili smo jFreeChart, Java knjižnico, ki nam je olajšala prikaz podatkov v grafih. Grafe smo izpisovali v realnem času, in sicer enkrat na sekundo.

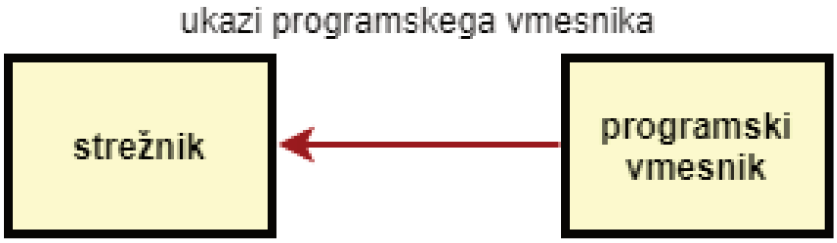
```
1 static TabChart tabChart = new TabChart();
2
3 public void trenutnaTemperatura(double t) {
4     tabChart.trenutnaTemperatura(t);
5 }
6 public void trenutnoStanjeVentila(int v) {
```



Slika 5.2: Prikaz podatkov v vmesniku.

```
7       tabChart.trenutnoStanjeVentila(v);
8   }
9   public void zeljtrenutnoStanjeOsv(int o) {
10      tabChart.zeljtrenutnoStanjeOsv(o);
11  }
12  public static void main(String[] args) {
13      JFrame f=new JFrame();
14      f.setDefaultCloseOperation(f.EXIT_ON_CLOSE);
15      JTextField tf = new JTextField("30");
16      tf.setEditable(false);
17      tf.setBounds(150,270,100,20);
18      JButton b1=new JButton("-");
19      JButton b2=new JButton("+");
20      b1.setBounds(150,300,50, 20);
21      b2.setBounds(200,300,50, 20);
22      JTextField tf1 = new JTextField("8");
23      tf1.setEditable(false);
24      tf1.setBounds(400,270,100,20);
25      JButton b3=new JButton("-");
26      JButton b4=new JButton("+");
27      b3.setBounds(400,300,50, 20);
28      b4.setBounds(450,300,50, 20);
```

```
29      b1.addActionListener(new ActionListener() {
30          public void actionPerformed(ActionEvent event) {
31              Rastlinjak rastlinjak = new Rastlinjak();
32              rastlinjak.spremeniTemperaturo(-1);
33              tf.setText(String.valueOf(rastlinjak.zeljenaTemperatura));
34              }
35      });
36      b2.addActionListener(new ActionListener() {
37          public void actionPerformed(ActionEvent event) {
38              Rastlinjak rastlinjak = new Rastlinjak();
39              rastlinjak.spremeniTemperaturo(1);
40              tf.setText(String.valueOf(rastlinjak.zeljenaTemperatura));
41              }
42      });
43      b3.addActionListener(new ActionListener() {
44          public void actionPerformed(ActionEvent event) {
45              Rastlinjak rastlinjak = new Rastlinjak();
46              rastlinjak.spremeniOsvetljevanje(-1);
47              tf1.setText(String.valueOf(rastlinjak.zeljenoOsv));
48              }
49      });
50      b4.addActionListener(new ActionListener() {
51              public void actionPerformed(ActionEvent event) {
52                  Rastlinjak rastlinjak = new Rastlinjak();
53                  rastlinjak.spremeniOsvetljevanje(1);
54                  tf1.setText(String.valueOf(rastlinjak.zeljenoOsv));
55              }
56      });
57      //dodamo vse elemente
58      f.add(b1);
59      f.add(b2);
60      f.add(tf);
61      f.add(b3);
62      f.add(b4);
63      f.add(tf1);
64      JPanel jPanel = tabChart.display();
65      f.add(jPanel);
66      f.pack();
67      f.setSize(631,400);
68      f.setLayout(null);
```



Slika 5.3: Pošiljanje ukazov strežniku.

```
69      f.setVisible(true);
70  }
```

5.3 Pošiljanje ukazov strežniku

Programski vmesnik je ukaze na strežnik pošiljal preko zahtevka POST. Zahtevek smo poslali le, kadar smo spreminjali želene vrednosti v programskem vmesniku.

```
1   public static void request() {
2       System.out.println("temperatura: " + zeljenaTemperatura);
3       System.out.println("osvetljevanje: " + zeljenoOsv);
4       URL url = new URL("http://192.168.1.200");
5       HttpURLConnection httpCon = (HttpURLConnection) url.openConnection();
6       httpCon.setDoOutput(true);
7       httpCon.setRequestMethod("POST");
8       httpCon.setRequestProperty("Content-Type","text/plain");
9       OutputStream os = httpCon.getOutputStream();
10      OutputStreamWriter osw = new OutputStreamWriter(os, "UTF-8");
11      osw.write(zeljenaTemperatura);
12      osw.flush();
13      osw.close();
14      os.close();
15      httpCon.connect();
16      System.out.println(httpCon.getResponseCode());
17      System.out.println(httpCon.getResponseMessage());
```

```
18      httpCon = (HttpURLConnection) url.openConnection();
19      httpCon.setDoOutput(true);
20      httpCon.setRequestMethod("POST");
21      httpCon.setRequestProperty("Content-Type","text/plain");
22      os = httpCon.getOutputStream();
23      osw = new OutputStreamWriter(os, "UTF-8");
24      osw.write(zeljenoOsv);
25      osw.flush();
26      osw.close();
27      os.close();
28  httpCon.connect();
29  }
```

Poglavje 6

Sklepne ugotovitve

V diplomskem delu smo razvili in implementirali avtonomen sistem za gojenje rastlin. Osredotočili smo se na osvetljevanje in zalivanje, ki sta za rast rastlin najpomembnejša. Poskrbeli smo za enakomerno zalivanje po celotni površini brez uporabe vodne črpalke. Z izbiro ustreznega vodnega ventila smo poskrbeli za varnost v primeru izpada sistema iz električnega omrežja. Omogočili smo prilagodljivo nastavitev trajanja osvetljevanja s svetili LED, glede na potrebe rastline in glede na količino sončne svetlobe. Podatke smo pošiljali v programski vmesnik, kjer smo jih predstavili v grafu.

Nadzorni sistem bi bilo mogoče še dodatno izboljšati. Da bi bil sistem bolj energetsko učinkovit, bi lahko namesto vodnega ventila uporabili pasiven namakalni sistem. Pri osvetljevanju bi lahko dodali senzor za svetilnost, ki bi svetila LED izklopil, kadar je dovolj sončne svetlobe. Vlažnost zraka je manj pomembna kot vlažnost zemlje, vendar bi lahko dodali tudi senzor za vlažnost zraka in zrak vlažili, kadar bi bila njegova vlažnost prenizka. Ker bi to vplivalo tudi na temperaturo zraka in vlažnost zemlje, nam tega v časovnem okviru eksperimenta ni uspelo implementirati. V okviru diplomske naloge se nismo ukvarjali s fizično zaščito pred pozebo in točo. Zaščito pred točo se doseže tako, da se rastlinjak pokrije z ustreznim zaščitnim materialom, ki pa vendarle prepušča sončno svetlobo. Pred pozebo pa se rastline lahko zaščiti z grelci, ki se vklopijo, kadar je temperatura pod določeno mejo. V diplomski

nalogi smo predpostavljali, da imamo na voljo neomejeno količino vode. V praksi to ni vedno res, vendar se ta problem enostavno reši z zbiralniki vode, ki v deževnih obdobjih kopičijo vodo za čas, ko je suša.

Literatura

[1] 1-wire protocol. Dosegljivo: https://en.wikipedia.org/wiki/1-Wire#Communication_protocol. [Dostopano: 22. 1. 2019].

[2] Awt and swing. Dosegljivo: http://edn.embarcadero.com/article/26970. [Dostopano: 22. 1. 2019].

[3] Dallas ds18b20. Dosegljivo: https://potentiallabs.com/cart/ds18b20-temperature-sensor. [Dostopano: 22. 1. 2019].

[4] Dallas ds18b20 manual. Dosegljivo: https://cdn-shop.adafruit.com/datasheets/DS18B20.pdf. [Dostopano: 22. 1. 2019].

[5] Ds18b20 temperature sensor. Dosegljivo: http://jakemakes.eu/esp32-arduino-ds18b20-temperature-sensor/. [Dostopano: 22. 1. 2019].

[6] Esp8266 datasheet. Dosegljivo: https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf. [Dostopano: 22. 1. 2019].

[7] Esp8266 d1 mini. Dosegljivo: https://wiki.wemos.cc/products:d1:d1_mini. [Dostopano: 22. 1. 2019].

[8] Organizacija računalniških sistemov, prosojnice. Dosegljivo: https://ucilnica.fri.uni-lj.si/pluginfile.php/96254/mod_resource/content/2/07-TIM.pdf. [Dostopano: 22. 1. 2019].

[9] Vgrajeni sistemi, prosojnice. Dosegljivo: https://ucilnica.fri.uni-lj.si/pluginfile.php/24399/mod_resource/content/0/05_ADC.pdf. [Dostopano: 22. 1. 2019].

[10] Iar embedded workbench. Dosegljivo: https://www.iar.com/iar-embedded-workbench/. [Dostopano: 22. 1. 2019].

[11] jfreechart architecture. Dosegljivo: https://www.tutorialspoint.com/jfreechart/jfreechart_architecture.htm. [Dostopano: 22. 1. 2019].

[12] jfreechart tutorial. Dosegljivo: https://www.tutorialspoint.com/jfreechart/. [Dostopano: 22. 1. 2019].

[13] T Majerle. Stm32f4 discovery tutorial. *Libraries and tutorials for STM32F4 series MCUs by Tilen Majerle–Library 08–ILI9341 LCD for STM32F4*, 2015.

[14] Soil moisture sensor. Dosegljivo: https://en.wikipedia.org/wiki/Soil_moisture_sensor. [Dostopano: 22. 1. 2019].

[15] Soil moisture sensor test plan. Dosegljivo: https://www.instructables.com/id/Soil-Moisture-Sensor-Test-Plan/. [Dostopano: 22. 1. 2019].

[16] Nvic. Dosegljivo: https://filderbaer.wordpress.com/2015/03/09/5-interrupt-programming/. [Dostopano: 22. 1. 2019].

[17] Red and blue lights. Dosegljivo: https://www.maximumyield.com/red-and-blue-lights/2/1387. [Dostopano: 22. 1. 2019].

[18] Smd 5050 led. Dosegljivo: https://www.ebay.com/itm/SMD-5050-LED-Strip-Grow-Light-Lamp-Full-Spectrum-For-Plant-Veg-DC-12V-Power-/222520948465. [Dostopano: 22. 1. 2019].

[19] Solenoid valve. Dosegljivo: https://www.solenoidsolutionsinc.com/infographics/how-a-2-way-normally-closed-solenoid-valve-works/. [Dostopano: 22. 1. 2019].

[20] Stm32f405xx/stm32f407xx datasheet. Dosegljivo: https://www.st.com/resource/en/datasheet/dm00037051.pdf. [Dostopano: 22. 1. 2019].

[21] Stm32f405xx/stm32f407xx reference manual. Dosegljivo: https://www.st.com/content/ccc/resource/technical/document/reference_manual/3d/6d/5a/66/b4/99/40/d4/DM00031020.pdf/files/DM00031020.pdf/jcr:content/translations/en.DM00031020.pdf. [Dostopano: 22. 1. 2019].

[22] Usart. Dosegljivo: http://www.circuitbasics.com/basics-uart-communication/. [Dostopano: 22. 1. 2019].

[23] Solenoid valve. Dosegljivo: https://www.ebay.com/itm/Normally-Closed-Fast-Response-Electric-Air-Water-Solenoid-Valve-1-4-Inch-12V-DC-/121090156302. [Dostopano: 22. 1. 2019].

[24] How much water do your plants really need? Dosegljivo: https://www.greenhousemag.com/article/gmpro-0310-water-plants-automating-irrigation//. [Dostopano: 22. 1. 2019].