Problem procjene i upravljanja temperaturom u kontekstu linearno-kvadratičnog gausovskog problema upravljanja

Šajnić, Oliver

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Oliver Šajnić

PROBLEM PROCJENE I UPRAVLJANJA TEMPERATUROM U KONTEKSTU LINEARNO-KVADRATIČNOG GAUSSOVOG UPRAVLJANJA

Diplomski rad

Voditelj rada:
prof. dr. sc. Luka Grubišić

Zagreb, 2024.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Posvećeno mojoj obitelji i prijateljima

Sadržaj

Uvod

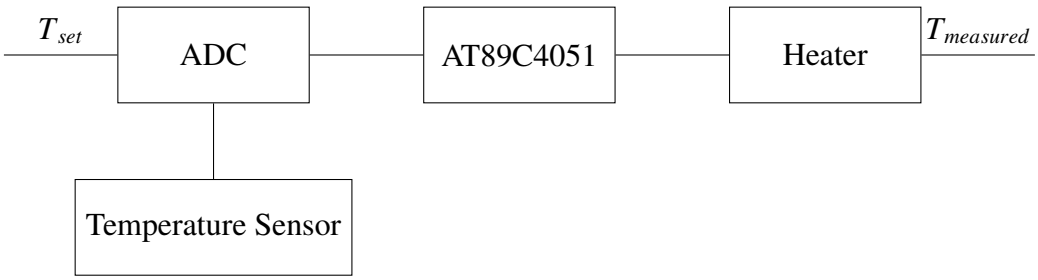
Pojam otvorenog koda u kontekstu razvoja softvera je jedna od vrućih tema koja osim stručne interesiraju i opću javnost. Cilj ovoga rada je razmotriti manje poznat pojam, pojam hardvera otvorenog dizajna (open source hardware). Hardver otvorenog dizajna se koristi kako za razvoj osobnih računala od slobodno dostupnih komponenti tako i za razvoj raznih uređaja unutar koncepta interneta stvari. Jedan od najpoznatijih razvojnih okruženja je sustav Arduino. U ovom radu ćemo na konkretnom primjeru razvoja mikrokontrolera za održavanjem temperature meda 1.1 prikazati razvoj jednog takvog rješenja korištenjem pristupa otvorenog hardvera. U realizaciji ovog rada usporedit ćemo programiranje za ugradbene sustave sa simulacijom njihovog rada korištenjem Python okruženja. Ovo je zapravo tipičan razvojni put jednog takvog cjelokupnog sustava. Prvo se simulacijama testira primjena rješenja u očekivanim režimima rada. Nakon toga se oblikuje hardversko rješenje koje se u zadnjem koraku realizira i naposlijetku validira uspoređujući ga sa rezultatima simulacije. Istaknimo da razvoj agenata, tj. softverskih rješenja koja umjesto klasičnog ulaza odbijaju ulazne podatke sa senzora, a izlazni podaci se realiziraju kroz aktuatore, je temelj umjetne inteligencije kao djela računarstva. Teorija upravljanja je jedna od klasičnih disciplina koja trenutno svoju nadogradnju ima u okvirima šireg pojma umjetne inteligencije. U ovom radu ćemo prezentirati rješenja jednostavnog kontrolera u Arduino hardveru uz programiranja kontrolera u BASCOM-u (osnovnom kompajleru za AVR kontrolere na kojima se temelji Arduino). Nakon toga ćemo prezentirati koncept PID i LQG kontrolera kroz simulaciju rada u Pythonu te pokazati da je takvo rješenje bolje od onoga kojega dobivamo jednostavnim algoritmom. Iako, i jednostavno rješenje daje dovoljno dobre rezultate u praktičnoj primjeni (procesiranje meda). Na kraju ćemo prezentirati plan moguće realizacije PID/LQG kontrolera u Arduino tehnologiji. Konkretnu realizaciju ćemo ostaviti za sljedeći projekt, budući da bi opsegom prelazila plan ovog diplomskog rada.

Motivacija

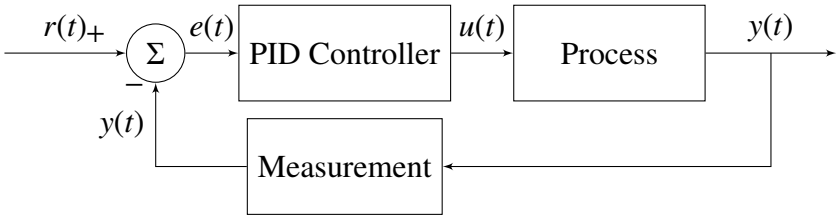
Znanja stečena prilikom obrazovanja, samoinicijativa, ranija zainteresiranost za određena područja programiranja i razni tečajevi doprinjeli su izradi ovog projekta. Plan izrade projekta i svih popratnih stvari poput simulacija i realizacija dao je veoma široku sliku i bogat sadržaj za pisanje rada. Omogućio je sklapanje i povezivanje puno različitih koncepata koji vode istoj svrsi te njihovu usporedbu, što samim time daje puno veći doprinos radu. Na posljetku dobivamo koristan i primjenjiv projekt zbog svoje praktične primjene u rješavanju stvarnih problema. To dodatno motivira i povećava interes za daljnje istraživanje i inovacije. Testiranje i evaluacija rezultata pomoću fizičkog rad i simulacija pruža korisne povratne informacije koje su ključne za unaprijeđenje i optimizaciju sustava. Kompaktnost i fleksibilnost Arduina omogućit će daljnje fizičke realizacije simulacija prikazanih u ovom radu. Nastavna istraživanja i eksperimenti doprinijet će unaprijeđenju funkcionalnosti i pouzdanosti sustava. Iskustvo stečeno tijekom izrade ovog projekta predstavlja temelj za buduće pothvate. Kontinuirano učenje i primjena stečenih znanja i iskustava bit će ključni za daljni razvoj kako projekta, tako i karijere.

Sheme

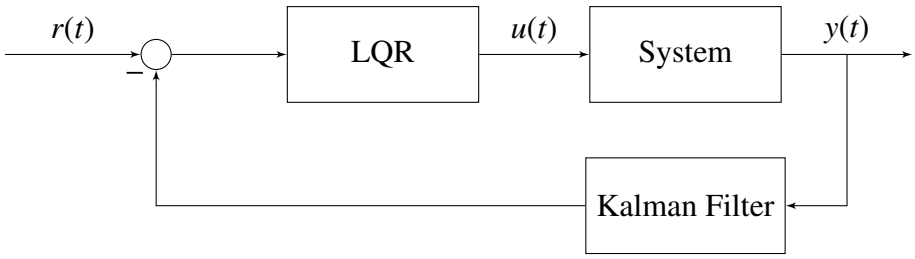
Za početak prikazat ćemo shemu kontrole zagrijavanja fizičkog rada. Ona se sastoji od mikrokontrolera AT89C4051, ADC-a (analogno-digitalnog pretvornika), grijača i senzora temperature. Zadavanjem temperature (oznaka T_{set}) i prosljeđivanjem trenutne temperatre koju mjeri senzor, ADC pretvara analogni u digitalni signal prije nego li ga preda mikrokontroleru. Mikrokontroler generira kontrolni signal koji pali ili gasi grijač po potrebi ovisno o izmjerenoj temperaturi (oznaka $T_{measured}$).



Sljedeća shema je shema PID (proporcionalno-integralno-derivativnog) kontrolera. Sastoji se od ulaznog signala $r(t)$, greške $e(t)$, kontrolnog signala $u(t)$, izlaznog signala $y(t)$, sumatora Σ, PID mikrokontrolera, bloka koji oblikuje izlazni signal i bloka koji vraća izlaznu vrijednost nazad u sumator. Ulazni signal predstavlja željenu vrijednost koju želimo postići, sumator pridodaje zbroj ili razliku ulaznog signala i izlazne vrijednosti što rezultira greškom. PID mikrokontroler provodi kontrolu na temelju greške, a računa proporcionalni, integralni i derivativni član koji generiraju izlazni signal $u(t)$ koji utječe na izlazni signal.



Zadnja shema koju ćemo koristiti je shema LQG (linearno-kvadratičnog-Gaussovog) kontrolera. Sastoji se od ulaznog signala $r(t)$, kontrolnog signala $u(t)$, izlaznog signala $y(t)$, linearno-kvadratičnog regulatora (LQR), sustava i Kalmanovog filtera. Ulazni signal predstavlja željeni izlazni signal. Pritom se kroz rad LQG kontrolera izlazni signal oduzima od ulaznog i računa se greška koja se prosljeđuje LQR-u koji od Kalmanovog filtera dobiva predodžbu o stanju sustava kako bi se optimirala kontrola. Proizvedeni kontrolni signal dovodi se sustavu koji računa izlazni signal.



Poglavlje 1

Izrada fizičkog rada

Promotrimo sada tijek izrade fizičkog dijela rada. U ovom poglavlju opisani su i prikazani svi koraci izrade i upute za korištenje uređaja.

Slika 1.1: Regulator viskoznosti meda

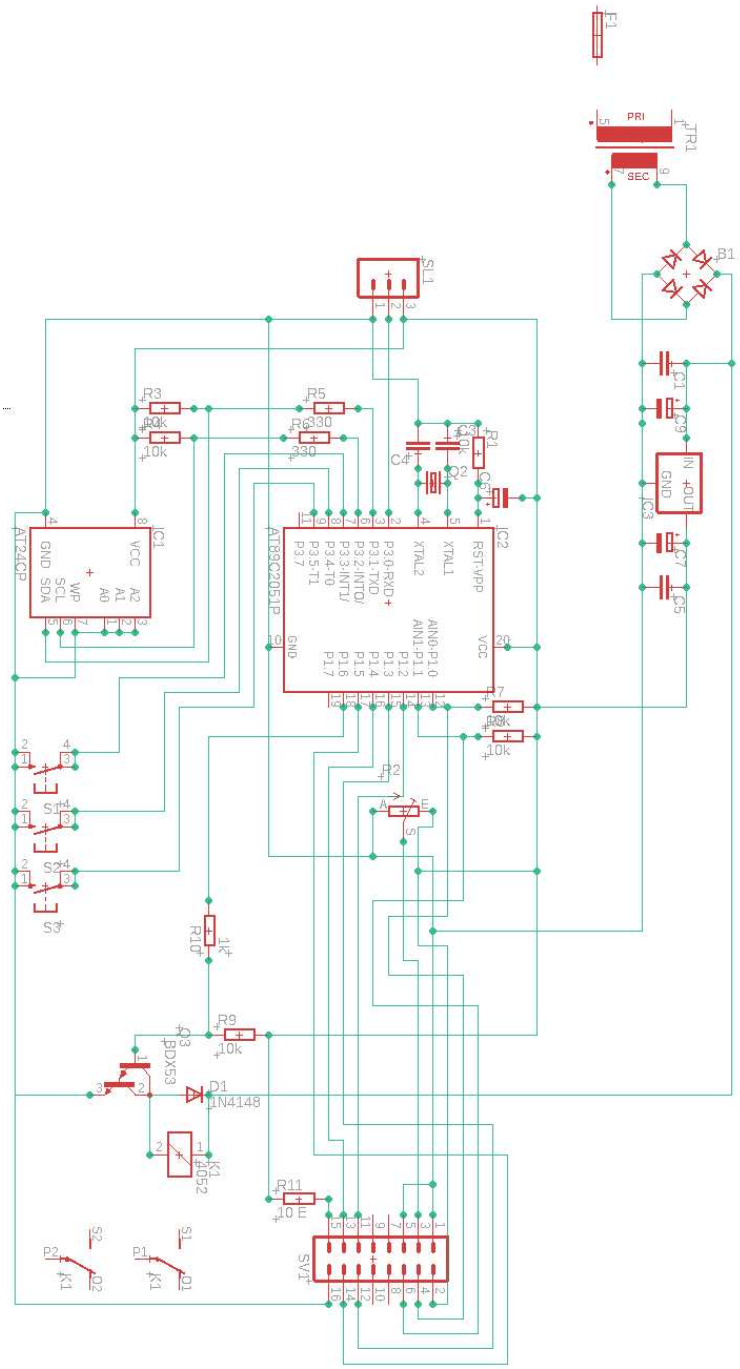
1.1 Sheme

Električna shema

Za izradu električne sheme, kao i za izradu ostalih shema koristimo program Eagle. Za početak potrebno je odabrati komponente koje želimo u shemi. Eagle ima ugrađenu biblioteku dijelova, ali mogu se dodati i vlastite biblioteke, kao i one dostupne na Internetu. Dajemo imena odabranim komponentama. Povezujemo komponente koje odgovaraju istoj takvoj električnoj shemi na papiru. Zatim validiramo shemu kako bi bili sigurni da su sve veze ispravne i da nema grešaka.

Slika 1.2: Program Eagle

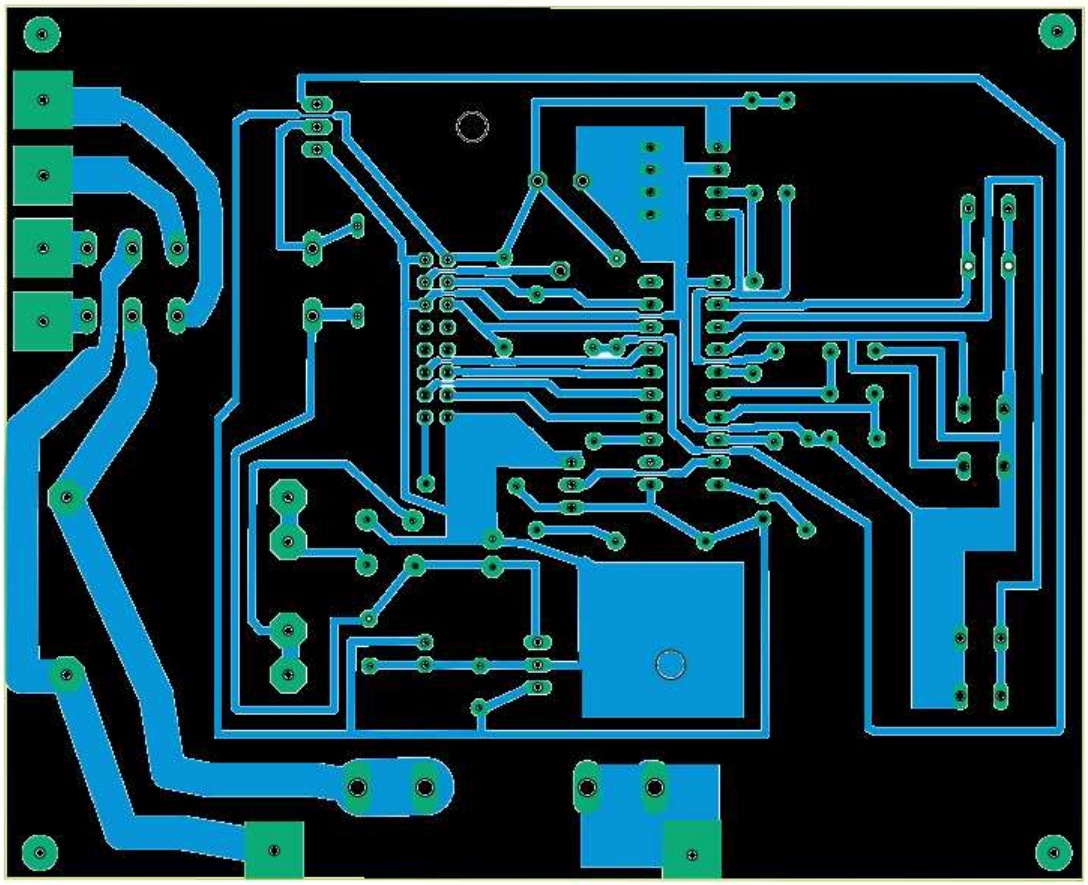
Na slici 1.3 prikazana je električna shema s kojom započinje cijela izrada rada.



Slika 1.3: Električna shema

Shema tiskanih vodova

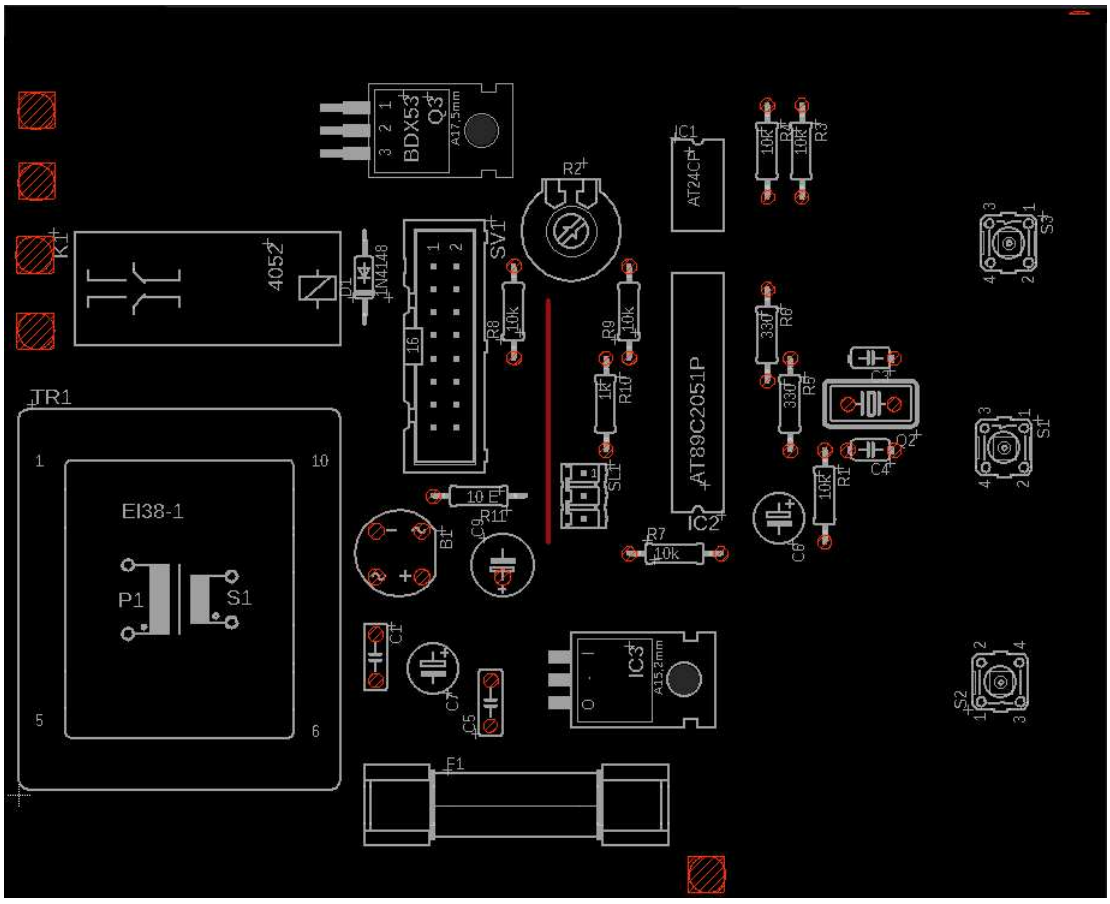
Prije izrade sheme tiskanih vodova (PCB layouta) potrebno je izgenerirati netlistu iz električne sheme. Ona sadrži popis dijelova i njihovih veza koje Eagle koristi za stvaranje sheme tiskanih vodova. Otvaramo novi PCB layout projekt i postavljamo komponente na njega tako da okvirno odgovaraju njihovim pozicijama na električnoj shemi. Pritom pazimo da si olakšamo najlakši način povezivanja komponenti kako ne bi došlo do preklapanja vodova. Eagle pruža i alat za samostalno kreiranje vodova, ali za izradu ove sheme mi nije bio potreban. Potrebno je održavati dovoljnu udaljenost između vodova jer i na taj način bi moglo doći do preklapanja, što u konačnici može rezultirati neispravnim radom uređaja.



Slika 1.4: Shema tiskanih vodova

Montažna shema

Ova shema govori nam kako su komponente složene s druge strane tiskane pločice. Otvaramo postojeći PCB layout pomoću kojeg dolazimo do montažne sheme. To činimo tako da odabiremo potrebne slojeve koje će nam prikazati komponente, njihove nazive, ulaze, izlaze i spojnike.



Slika 1.5: Montažna shema

1.2 Izrada tiskane pločice

Elektronika se oslanja na tiskane pločice kao osnovne komponente u gotovo svim uređajima od onih jednostavnih sve do složenih računalnih sustava. Kako bi se osigurala njihova funkcionalnost i pouzdanost potrebno je puno koraka. Prenošenje uzorka na vodljivi sloj nužan je korak u izradi tiskane pločice. U ovom se koraku oblikuju električne veze (vodovi). Tehnika fotopostupka potrebna je za detaljno i precizno nanošenje uzoraka na tiskanu pločicu.

Nakon što se uzorak prenese, slijedi proces uklanjanja nepotrebnih dijelova vodljivog sloja. Kemijsko jetkanje je zapravo korištenje kemikalija, kiseline i lužine koje omogućavaju ostajanje vodova na tiskanoj pločici. Bušenje je nužno kako bi se kreirale rupe za montažu komponenata. Preciznost osigurava da komponente budu ispravno postavljene, a da se električni signali nesmetano prenose.

Nakon čišćenja, slijedi nanošenje lema ili paste za lemljenje. Ona igra ključnu ulogu u stvaranju pouzdanog električnog i mehaničkog spoja između komponenti i vodova na pločici. Lem ili pasta za lemljenje sadrži metalne čestice koje, kada se zagrijavaju, tvore čvrstu vezu koja omogućuje protok električne struje. Precizno nanošenje lema osigurava da se prava količina materijala nalazi na pravom mjestu, što smanjuje rizik od prekomjernog lema koji može dovesti do nepoželjnih kratkih spojeva, ili nedostatka lema koji može rezultirati slabim spojevima. Pažljiva priprema pločice, od čišćenja do nanošenja lema, osigurava da svaka komponenta ima čvrst i pouzdan spoj koji će trajati kroz životni vijek uređaja.

1.3 Popis materijala

Količina	Tip	Vrijednost	Naziv
3	Tipkalo	/	S1, S2, S3
1	Integrirani krug	7805	IC3
2	Blok kondenzator	33 pF	C3, C4
2	Blok kondenzator	22 pF	C1, C5
2	Elektrolitski kondenzator	10µF / 16V	C6, C7
1	Elektrolitski kondenzator	100µF / 35V	C9
1	Kristal	/	Q2
1	Transformator	/	TR1
1	AMP konektor	/	SL1
1	Konektor za displej	ML16	SV1
1	Ispravljač	RB1A	B1
1	Držač osigurača	SH32	F1
1	Potenciometar	10 k	R2
1	Otpornik	10 E	R11
1	Elektrolitski kondenzator	10 nF / 630V	C11
6	Otpornik	10 k	R1, R3, R4, R7, R8, R9
1	Dioda	1N4148	D1
1	Otpornik	1 k	R10
2	Otpornik	330 E	R5, R6
1	Otpornik	360 E	R14
1	Otpornik	39 E	R15
1	Relej	4052	K2
2	Otpornik	470 E	R12, R13
1	Elektrolitski kondenzator	470 nF / 630V	C10
1	EEPROM	AT24CP	IC1
1	Mikrokontroler	AT89C4051	IC2
1	NPN Darlingtonov tranzistor	BDX53	Q3

1.4 Funkcionalnost tiskane pločice

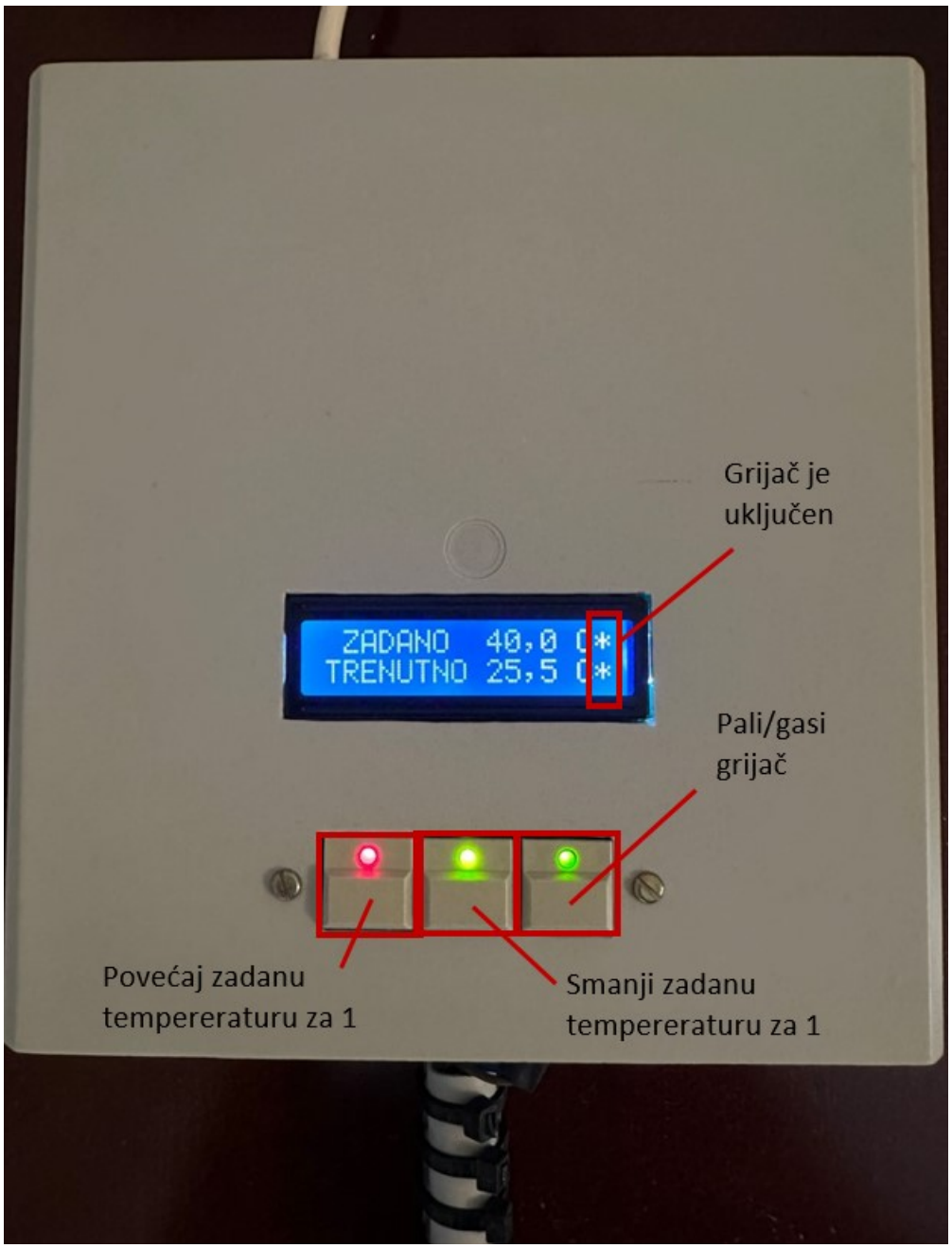
U ovom dijelu opisat ćemo ukratko kako radi tiskana pločica, odnosno koja je uloga električne sheme. Na ulaz dovodimo napon od 220V. Nakon ulaza se odmah nalazi transformator (TR1) koji pretvara napon od 220V u napon od 12V. Taj napon dolazi na Graetzov spoj (B1) koji ispravlja izmjenični napon u istosmjerni napon. To znači da se negativne poluperiode ispravljaju u pozitivne poluperiode. Kondenzatori filtriraju ili "glade" ispravljeni napon. Regulator napon (IC3) stabilizira napon na izlazu pretvarajući ga u napon od 5V koji je potreban za napajanje mikrokontrolera i drugih dijelova.

Mikrokontroler (IC2) dobiva stabilizirani napon. Reset pin omogućuje pravilno resetiranje mikrokontrolera prilikom uključivanja. Pinovi za kristalno osciliranje povezani su s vanjskim kristalom (Q2) i kondenzatorima za generiranje taktnog signala koji određuje radnu frekvenciju mikrokontrolera. EEPROM (IC1) služi za pohranu podataka koje mikrokontroler čita i piše tijekom rada. AMP konektor (SL1) služi za povezivanje grijača. Tipke (S1, S2, S3) povezane su s pinovima mikrokontrolera kako bi se registrirao korisnički odabir, odnosno paljenje/gašenje grijača, povećavanje ili smanjivanje temperature.

Relej (K1) kontroliran je NPN Darlingtonovim tranzistorom (Q1) koji omogućava visoko pojačanje struje. Kada je tranzistor otvoren, struja teče kroz zavojnicu releja i tako zatvara kontakt releja i omogućava protok struje koji relej regulira za pravilan rad grijača. Dioda (D1) povezana je paralelno s relejom kako bi zaštitila tranzistor od povratnog napona prilikom isključivanja releja.

U suštini električna shema predstavlja osnovni sustav koji koristi mikrokontroler za kontrolu raznih vanjskih uređaja. Svaka komponenta ima specifičnu ulogu u održavanju stabilnog rada sustava pri čemu je omogućen korisnički unos i pohrana podataka.

1.5 Legenda za upotrebu uređaja



Slika 1.6: Legenda za upotrebu uređaja

Slika 1.7: Prekidač za paljenje/gašenje uređaja

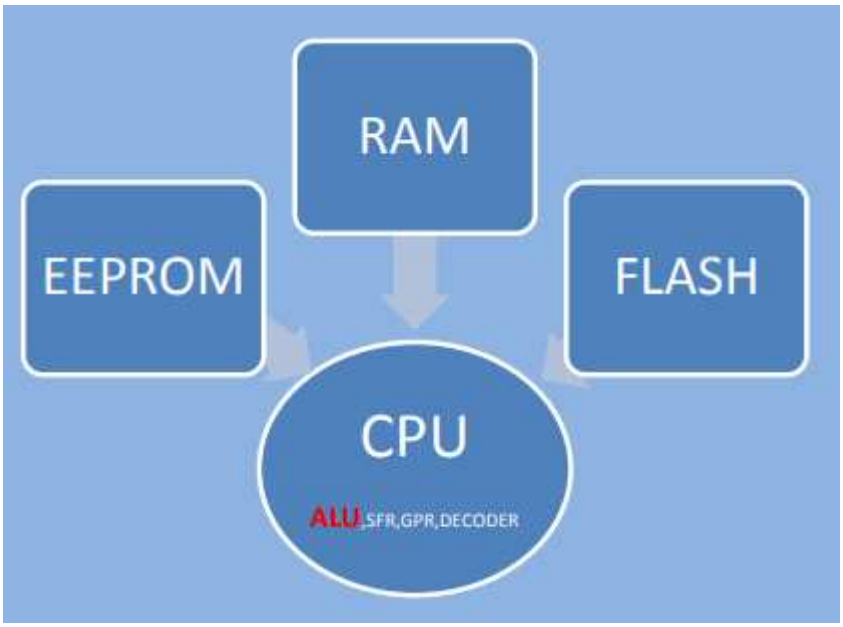
Poglavlje 2

Mikrokontroleri

Mikrokontroleri, kao dijelovi elektroničkih uređaja i sustava doživljavaju veliku upotrebu i širenje te se pojavljuju u skoro svim djelatnostima od dječjih igračaka do industrije i upravljačkih mehanizama aviona. Mikrokontroleri se mogu proučavati sa više stajališta: tehnološkog, računarskog i korisničkog. Prva dva se tiču inženjera elektrotehnike i projektiranje hardvera, dok je za stručnjake koji rade u području upravljanja sustavima, mikrokontroleri su interesantni sa korisničkog stajališta kao komponenta koja treba biti u stanju obaviti odgovarajuće funkcije u nekom sustavu.

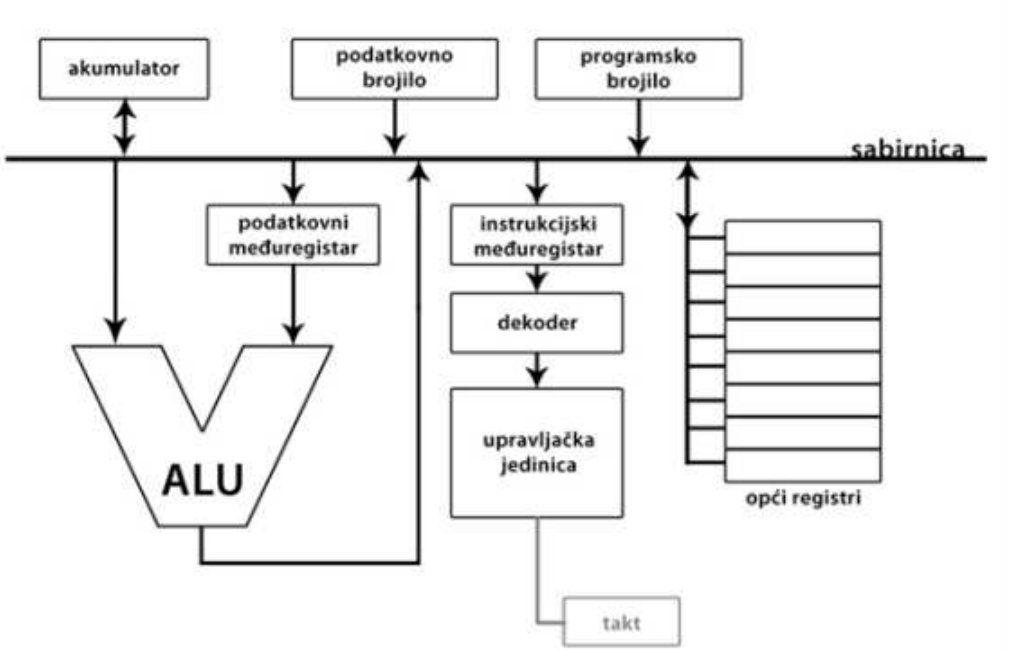
2.1 Arhitektura mikrokontrolera

Procesor mikrokontrolera izvršava naredbe te obavlja aritmetičke i logičke operacije. Sve naredbe dolaze iz FLASH memorije u kojoj se nalaze u binarnom obliku. Unutarnji RAM ima memoriju male veličine. EEPROM služi za pohranu podataka koji nastaju prilikom rada mikroračunala. EEPROM također pamti podatke dok je uređaj isključen, a podatci se mogu brisati i ponovo pisati jer ju je moguće ponovo programirati električnom strujom.



Slika 2.1: Glavni dijelovi mikrokontrolera

Jezgra procesora je ALU (aritmetičko logička jedinica). Naredbe stižu putem sabirnice, ali također i adrese i podatci (operandi) koje će naredba obraditi. Da bi ALU razumjela naredbe one se moraju dekodirati. Akumulator je 8-bitni registar koji služi za privremenu pohranu rezultata. ALU izvršava operacije nad podatcima. Naredbe su napisane u asemblerskom jeziku za pripadajuću arhitekturu mikrokontrolera. Te naredbe dekodira dekoder, a tek ih onda ALU izvršava. Klasična arhitektura podudara se s Von Neumannovim modelom računala.



Slika 2.2: Unutrašnjost procesora

Projektiranje ugradbenih sustava

Da bi mogli usporediti mikrokontrolere, moramo prvo shvatiti njihovu ulogu, odnosno zahtjeve koje se stavlja pred njih.

Opći zahtjevi su:

- **Niska cijena**
  Najveći dio ugradbenih proizvoda se radi za tržišta gdje krajnji korisnici nisu voljni potrošiti više novca za neznatno bolje performanse ili nekoliko dodatnih pogodnosti. Stoga se cilja optimalan odnos cijena/performansa, te da tražene performanse imaju što manju cijenu.

- **Napajanje**
  Potrošnja je još jedan kritičan faktor pri dizajnu ovih sustava. To je uglavnom zbog toga što veliki broj ovakvih sustava radi pomoću baterije, a od njih se očekuje da rade i budu aktivni čak i u dužim periodima kao na primjer mobiteli i slično. Niska potrošnja često je obavezna.

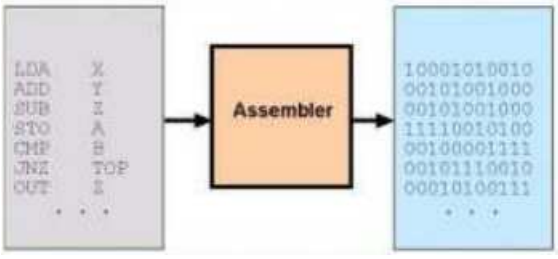
- **Predvidljivost**
  Zbog čestih realno-vremenskih zahtjeva, a dio su uloge ugradbenih sustava, predvidljivost je esencijalna. To bi značilo da se ponašanje sustava mora moći predvidjeti pod svim mogućim uvjetima i događajima. Softver se često projektira za scenarij najgoreg slučaja i to deterministički.

- **Odziv**
  Ugradbeni sustavi poput upravljačkih i regulacijskih sustava su određeni događajima te stoga na njih moraju odgovarati dovoljno brzo. Takvi sustavi se dosta oslanjaju na prekide. Zbog toga moraju biti dizajnirani tako da imaju dovoljno brze odzive na prekide.

- **Vremenska točnost**
  Distribuirani ugradbeni sustavi koji upravljaju udaljenim bazama podataka i mrežni uređaji zahtjevaju preciznu vremensku granulaciju. Rezolucija i točnost koji se očekuju, za većinu ovih primjena, su reda mikrosekundi.

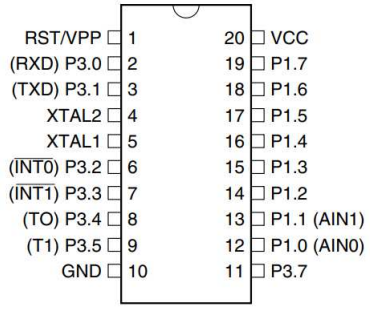
Asemblerski jezik

Asembler je programski jezik niske razine namijenjen za direktnu komunikaciju sa hardverom. Može se pisati binarno, heksadekadski, ali i obliku mnemonika (strojnih naredbi). Čini vezu između softverskih programa i njihovih podložnih hardvera. Ima svoju sintaksu, labele, operatore i direktive da pretvori kod u instrukcije koje koristi računalo. Izvođenje ovakvog programskog jezika je često brže od programskih jezika visoke razine. Koristimo ga kada želimo veću kontrolu u manipulaciji hardvera, direktan pristup specijaliziranim instrukcijama ili evaluaciji performansa. Programi pisani u asembleru su nešto čitljiviji i lakši za razumijevanje od binarnog zapisa, ali ih je još uvijek teško pisati i ispravljati. Mogu se izvršavati samo na procesoru za koji su pisani. Danas se uglavnom koristi na specijaliziranim procesorima (mikrokontrolerima) koji se ugrađuju u različite elektroničke naprave.



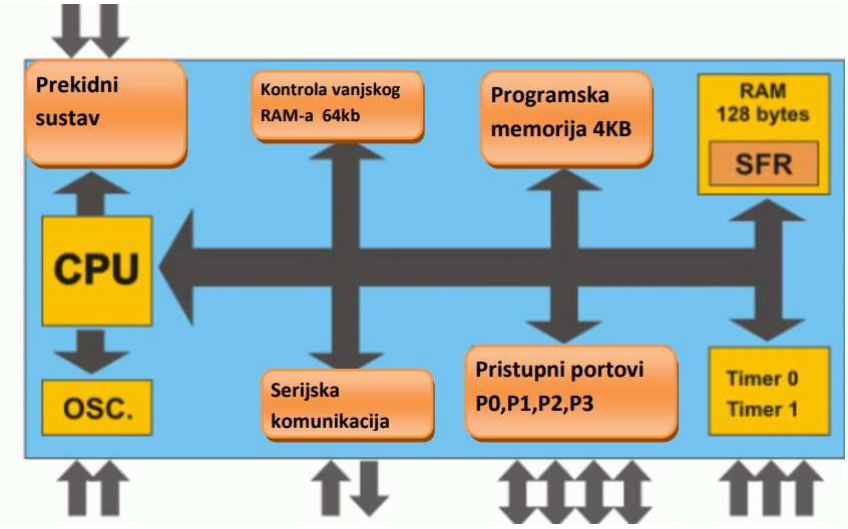
Slika 2.3: Uloga asemblera

2.2 Mikrokontroler AT89C4051

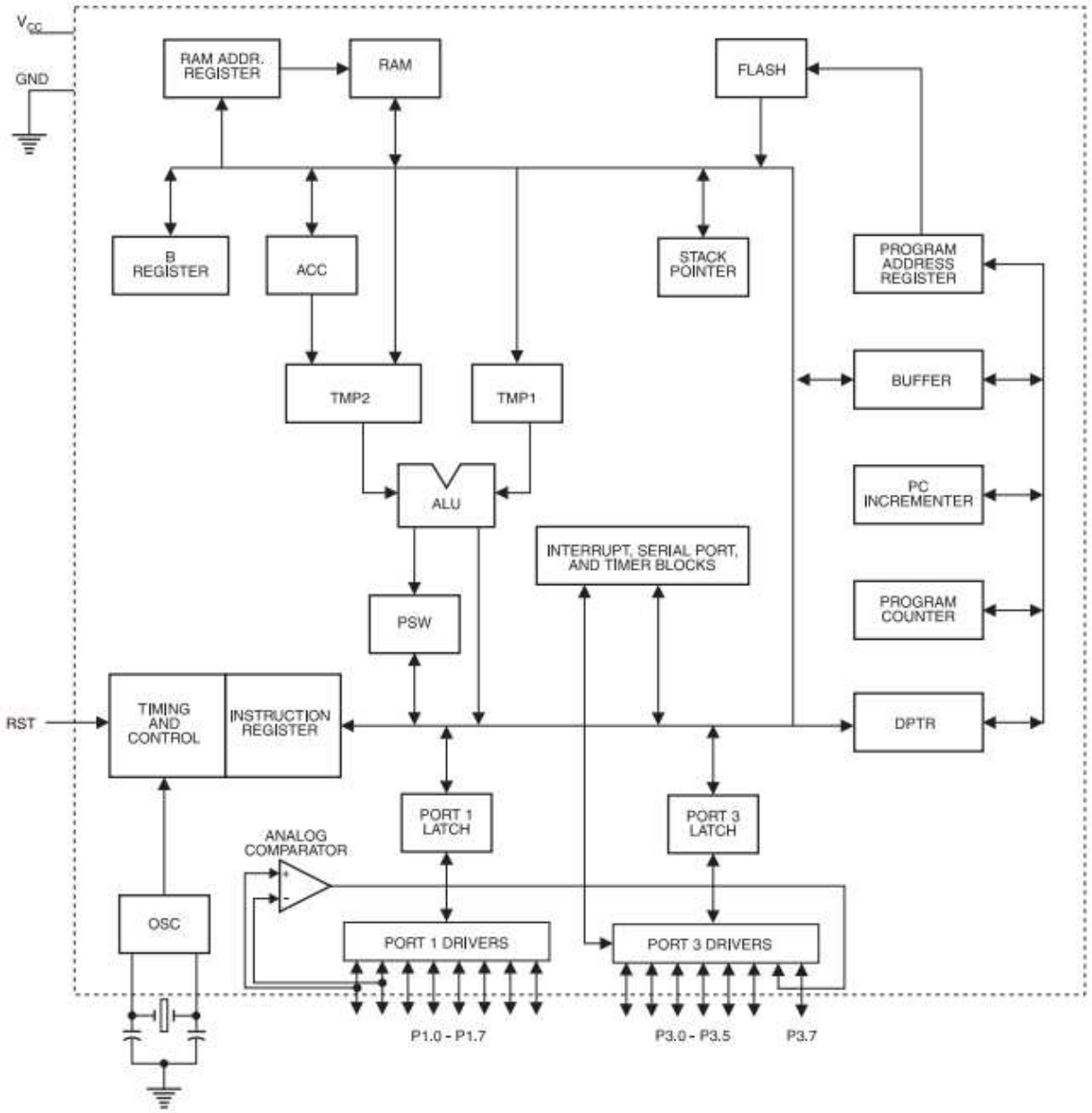


Slika 2.4: Mikrokontroler AT89C4051

Mikrokontroler ima četiri pristupna porta, 4 kB FLASH memorije, 2 tajmera, unutrašnji RAM od 128 B, sustav za serijsku komunikaciju, prekidni sustav i procesor. Pripada CISC arhitekturi mikrokontrolera što znači da koristi kompleksan set instrukcija, ali i ima smanjen broj instrukcija. Dio je 8051 arhitekture chipova koji pripadaju CMOS "obitelji" mikrokontrolera. Ima 20 pinova, a od toga je 15 ulazno-izlaznih portova (P1 i P3). Na slici 2.5 vidimo razliku između procesora i mikrokontrolera, a slika 2.6 prikazuje blok dijagram mikrokontrolera AT89C4051 na kojoj su vidljivi dijelovi mikrokontrolera i njihova povezanost.



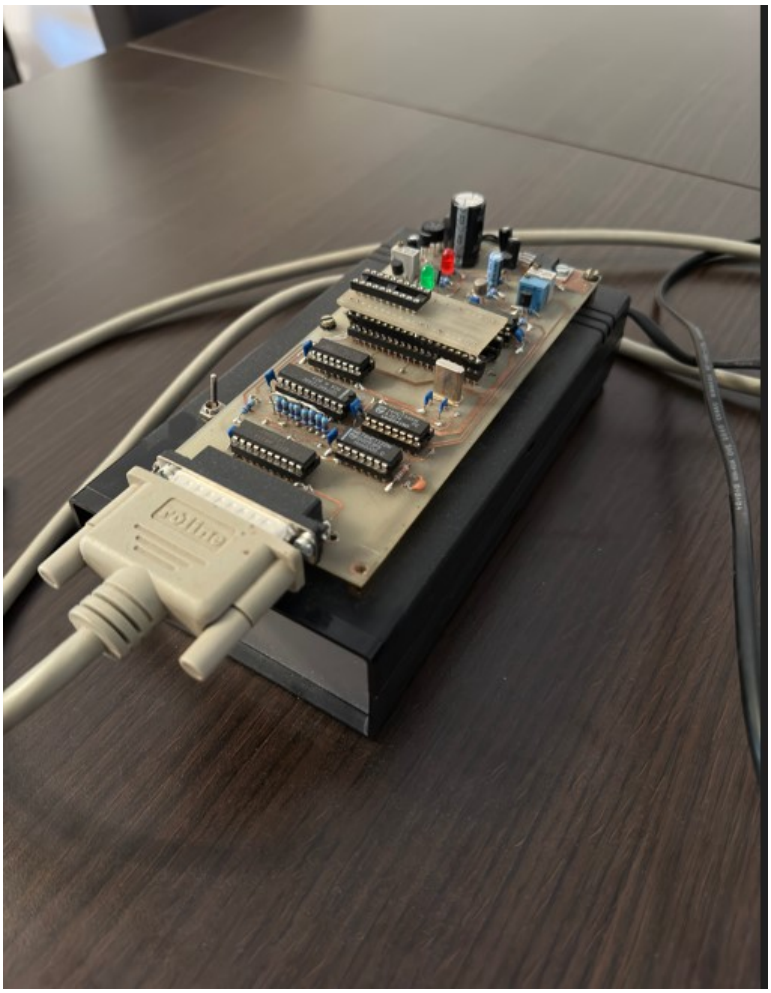
Slika 2.5: Unutrašnjost mikrokontrolera



Slika 2.6: AT89C4051 blok dijagram

Programiranje mikrokontrolera

Za programiranje mikrokontrolera odabran je BASCOM 8051 DEMO. Ova verzija dozvoljava ograničen broj naredbi pošto je demo verzija, ali je dovoljna za izradu ovog rada. Nakon što se napiše program potrebno ga je kompajlirati. Kompajliranjem dobivamo nekoliko datoteka od kojih je jedna binarna koju odabiremo za učitavanje programa u mikrokontroler koji će ga izvršiti. Tada dolazi vrijeme za upotrebu programatora. Spajamo ga na paralelni port na računalu (potrebno je imati nešto starije računalo ili neki od postojećih adaptera). Budući da koristimo programator PG302 (slika 2.7) potrebno ga je u opcijama odabrati jednako kao i chip koji se koristi. Nakon što programiramo mikrokontroler testiramo ga da bi provjerili ponaša li se naš program, to jest rad očekivano.



Slika 2.7: Programator PG302

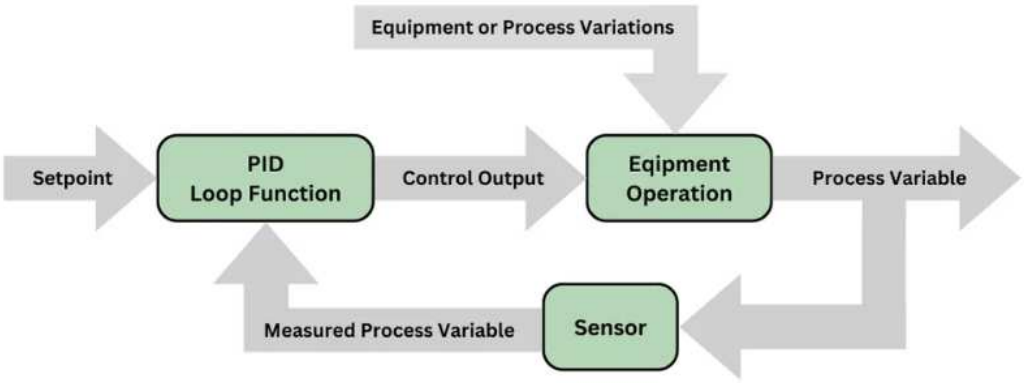
2.3 PID kontroler

Proporcionalno-integralno-derivativni (PID) kontroler je važan za industrijske sustave. On je snažan kontrolni mehanizam koji manipulira ulaznu varijablu baziranu na signalu greške, odnosno razlici između postavljene vrijednosti i promatrane vrijednosti.

PID petlja uključuje tri parametra:

- Proporcionalnu vrijednost
- Integralnu vrijednost
- Derivativnu vrijednost

U petlji se alterira izlazni rezultat za kojeg želimo da postigne željenu vrijednost koju smo postavili i to sa minimalnim brojem grešaka i optimalnom brzinom. Time ga činimo potrebnim alatom u različitim inženjerskim i tehnološkim aplikacijama.

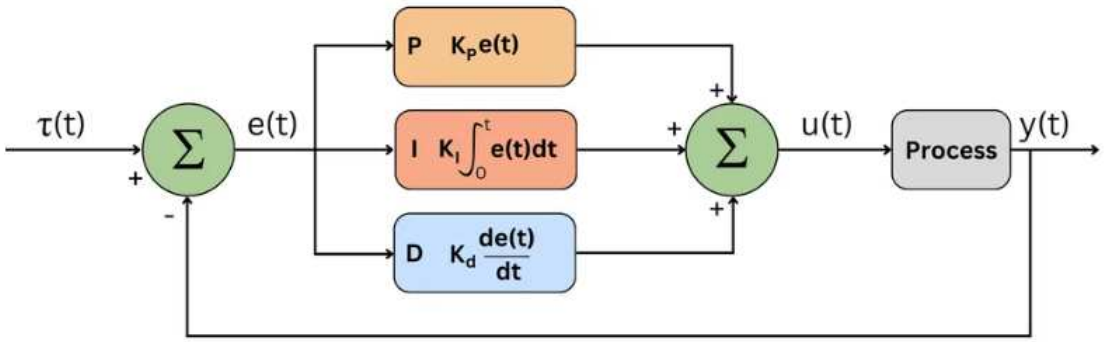


Slika 2.8: Dijagram osnovne PID petlje

Osnovne PID kontrole

Algoritam PID kontrolera bazira se na povratnim informacijama kontrolne petlje mehanizma u kojemu se koristi. Kada je PID petlja aktivna, ona izračunava vrijednost greške kao razliku između mjerene vrijednosti i željene vrijednosti. Kontroler tada nastoji minimizirati tu grešku s vremenom tako da prilagodi kontrolne varijable. Proporcionalni član stvara izlaznu vrijednost proporcionalnu trenutnoj vrijednosti greške. Integralni član je proporcionalan s magnitudom greške i trajanjem greške što ga čini ključnim za dugoročnu

preciznost. Derivativni član se izračunava na temelju brzine promjene greške. On pridonosi predviđanju budućih grešaka koje se temelje na trenutnoj brzini promjene.



Slika 2.9: Dijagram kontrolne PID petlje

Kao snažan kontrolni mehanizam, PID kontroler je primjeren za sustave gdje je matematički model nedostupan ili je prekompleksan da bi se modelirao. U takvim slučajevima, PID kontroler, sa svojim parametrima za ugađanje, može osigurati traženi nivo performansi.

Koncept kontrole uz povratne informacije

Kontrola koja ima povratne informacije je sustav koji koristi informacije iz mjerenja da bi manipulirao varijablu za postizanje željenog rezultata. Izlaz se dovodi ponovo na ulaz i pritom se zatvara petlja. Kontinuiranim provjerama izlaza i pravilnim prilagođavanjem ulaza, sustav se sam regulira kako bi održao željeni izlaz. U PID petlji, varijabla koja se mjeri poput temperature ili brzine je kontinuirano pod kontrolom i usporedbom sa željenom vrijednošću. Razlika između željene i promatrane vrijednosti izračunava se kao greška. PID kontroler radi kako bi minimizirao tu grešku tako da prilagođava kontrolnu varijablu poput električne energije dovedene na grijač ili potrebnu poziciju. Ključna stvar kontrole uz povratne informacije leži u sposobnosti da koristi izlazne podatke kako bi utjecala na ulaz, osiguravajući time da se odaziv sustava poklapa s željenom vrijednošću. Ova značajka daje sustavu karakteristike samoregulacije koja ga čini odbojnim na vanjske smetnje i varijacije u sustavnoj dinamici.

PID algoritam

PID algoritam koristi tri odvojena moda za izračun izlaza koji će biti pridjeljen kontrolnoj varijabli.
Prvi mod je proporcionalni mod ili P mod. Izlaz kontrolera je direktno proporcionalan trenutnoj vrijednosti greške u P modu. Ova greška očituje se kao razlika željene vrijednosti i mjerene vrijednosti. Matematički, proporcionalni član je izražen kao:

$$P_{out} = K_p \cdot e(t),$$

gdje je P_{out} proporcionalni izlaz, K_p proporcionalno pojačanje i $e(t)$ trenutna greška u trenutku t.

Drugi mod je integralni mod ili I mod. Ovaj integralni dio uzima u obzir magnitudu greške i trajanje njezine pojave. Napravljeno je da se odaziva akumuliranim greškama kroz vrijeme, te da pridonosi eliminaciji preostalih čekajućih grešaka koje su prisutne samo u P kontrolerima. Matematički reprezentacija integralnog člana je:

$$I_{out} = K_i \cdot \int e(t)dt,$$

gdje je I_{out} integralni izlaz, K_i integralno pojačanje i integral od $e(t)$ je integral po vremenu od početka do trenutka t.

Treći mod je derivativni mod ili D mod. Derivativni dio PID kontrolera brine o brzini promjene greške. On pridaje predviđanju budućih grešaka koje su bazirane na trenutnoj brzini promjene i pomaže smanjiti prekoračenje i vrijeme smirivanja. Derivativni mod može povećati vrijeme podizanja sustava, pogotovo ako je konstanta derivativnog pojačanja prevelika. Možemo ju reprezentirati kao:

$$D_{out} = K_d \cdot \frac{d}{dt}e(t),$$

gdje je D_{out} derivativni izlaz, K_d je derivativno pojačanje i $\frac{d}{dt}e(t)$ je brzina promjene greške kroz u trenutku t.

Ukupni izlaz PID algoritma je suma ovih triju člana koja se može reprezentirati kao:

$$PID_{out} = P_{out} + I_{out} + D_{out}.$$

Ovaj kombinirani izlaz se tada koristi da bi se prilagodila kontrolna varijabla tako da se minimizira pogreška i da se sustav optimalno odaziva na promjene željene vrijednosti ili smetnje.

2.4 LQG kontroler

Ovim kontrolerom nastoji se postići bolja kontrola temperature. Pojava greške je normalna i očekivana, ali se ona ovdje nastoji što više smanjiti. Da bi to postigli potreban nam je dio koji će voditi brigu o procjenama stanja sustava koje su potrebne za učinkovitu kontrolu. Dodavanje takozvanog procjenitelja stanja sustava logičkoj kontroli komplicira problem robustnosti jer je procjenitelj i sam po sebi dinamički sustav koji je podložan nestabilnosti. Jer je izolirani linearno-Gaussov procjenitelj matematički blizanac linearno-kvadratičnog regulatora, on posjeduje iste margine stabilnosti. Kada je procjenitelj dio povratne veze stohastičkog regulatora, dinamičke interakcije između procjenitelja i kontroliranog sustava povećavaju šanse za nestabilnost. Ako je robustnost stohastičkog regulatora neadekvatna mogu se koristiti razne metode za njezin oporavak. Logično je osigurati odgovarajuću povratnu vezu za početak. Procjenitelj je dizajniran da pridonosi dovoljno dobro predviđanje. Sama robustnost može biti stohastički problem. Potrebno je minimalizirati vjerojatnost nastanka nestabilnosti.

Kalmanov filter

Kalmanov filter dugo je smatran optimalnim rješenjem za mnoge zadatke predviđanja. Njegova primjena u analizi vizualnih kretanja je često dokumentirana. Standardna izvedba opisana je u nastavku za praktičnu upotrebu. Filter je konstruiran kao pomagalo za smanjenje pogreške. Svrha filtera je izvlačenje potrebnih informacija iz signala ignorirajući sve ostalo. Može se promatrati koliko se dobro filter ponaša. Zapravo, Kalmanov filter služi za procesiranje šumova. Postoje dvije vrste šumova. Procesni šum je nesigurnost u modelu sustava koja proizlazi iz nepredviđenih varijacija u samom sustavu, dok je mjerni šum nesigurnost u točnosti mjerenja koja proizlazi iz različitih izvora šumova i smetnji u senzorima ili okolini. Prema tome, želimo da filter minimizira gubitke i spriječi nepotrebne dobitke procesnog i mjernog šuma. Ključni koraci Kalmanovog filtera su predikcija i ažuriranje.

Tako u predikciji predviđamo stanje varijable $\hat{x}$:

$$\hat{x}_{k+1} = \hat{x}_k + Bu_k$$

gdje je B matrica kontrolnog ulaza, a u_k kontrolni ulaz u k-toj iteraciji.

Predikcija procjene greške računa se uz pomoć matrice P kao

$$P_{k+1} = AP_kA^T + Q$$

gdje je A matrica stanja, a Q matrica procesnog šuma.

Sljedeći korak je ažuriranje u kojemu dobivamo Kalmanov dobitak K_k kao

$$K_k = P_k H^T (HP_k H^T + R)^{-1}$$

gdje je P_k matrica procjene greške, H matrica mjerenja i R matrica mjernog šuma.

Ažuriramo procjenu stanja $\hat{x}$ na sljedeći način:

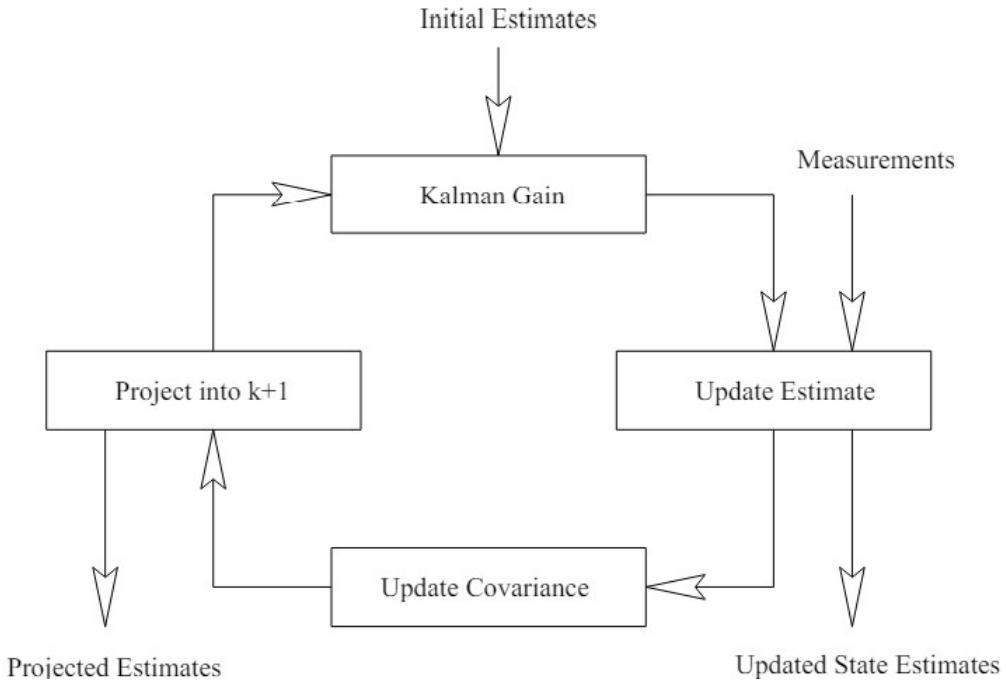
$$\hat{x}_k = \hat{x}_k + K(z - H\hat{x}_k)$$

gdje je z stvarno mjerenje.

Ažuriramo i matricu procjene greške P:

$$P_k = P_k - K_k H P_k.$$

Slika 2.10 prikazuje rekurzivni algoritam Kalmanovog filtera.



Slika 2.10: Rekurzivni algoritam Kalmanovog filtera

LQG kontrola

LQG metodologija daje sistematički pristup za izradu stabiliziranih kontrolera za bilo koji linearni model, uključujući i MIMO (multiple-input and multiple-output) sustave poput komunikacijskih, industrijskih i radarskih sustava te medicinskih uređaja gdje su prisutni složeni procesi i precizno upravljanje. Teorija je dobro razrađena, a postoje i analitička rješenja isto kao i uspješni softver za produciranje takvih kontrolera. Ono što se promatra povezuje se lijepo sa stvarnošću ako je stanje modela korespondentno u fizičkom sustavu. Kako bi dobili bolju kontrolu od LQR kontrole, koristimo kombinaciju Kalmanovog filtera i LQR kontrolera.

Zakon optimalne kontrole glasi:

$$u = -K\hat{x}$$

gdje je

$$K = -R^{-1}B^T S$$

pri čemu je S zadan kao

$$S = A^T S + SA + Q - SBR^{-1}B^T S$$

i $\hat{x}$ definiran kao

$$\hat{x} = A\hat{x} + Bu + K_F(y - H\hat{x})$$

uz Kalmanov dobitak K_F koji se računa kao

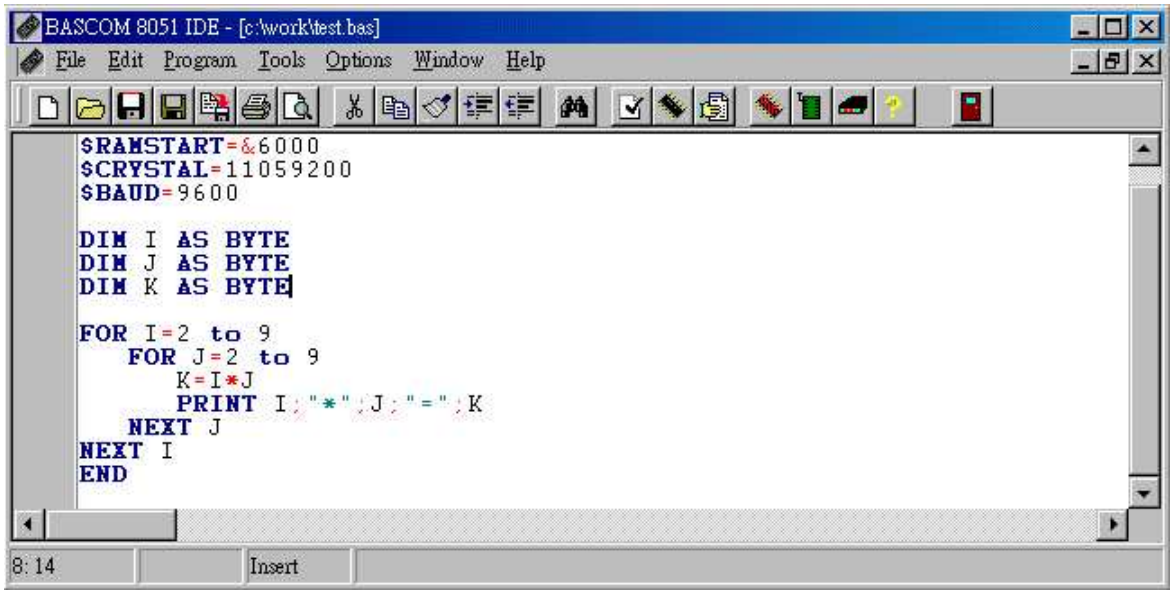
$$K_F = PH^T(HPH^T + R)^{-1}.$$

Poglavlje 3

Programski jezik BASCOM

BASCOM je visokokvalitetni programski jezik koji se koristi za razvoj ugradbenih sustava, posebno za mikrokontrolere poput AVR-a, PIC-a i drugih. Razvila ga je tvrtka MCS Electronics. Prvi put je predstavljen 1995. godine i od tada je doživio niz nadogradnji. U početku je bio namijenjen za AVR mikrokontrolere, a kasnije i za PIC mikrokontrolere. Jedna od stvari koje ga krase je jednostavnost pisanja koda, čak i bez velikog poznavanja hardvera mikrokontrolera. Omogućava visok nivo apstrakcije što pomaže programerima u pisanju programa. Kao i ostali programski jezici dolazi sa mnoštvom biblioteka za korištenje LCD ekrana, senzora i slično. Usput rečeno, BASCOM je kompajler, a uz to ima i jednostavno otklanjanje grešaka. Ima opciju prolaženja kroz kod korak po korak, praćenja promjenjivih vrijednosti i simulacija rada mikrokontrolera. Koristi jednostavnu sintaksu sličnu BASIC-u što znatno olakšava pisanje koda. Podržava uvjetno grananje, petlje i skakanje na potprograme pod nekim nazivom. Omogućava i pisanje funkcija i procedura. BASCOM pruža slabu podršku za objektno orijentirano programiranje. U usporedbi s drugim programskim jezicima ima manjak naprednih alata i manjak funkcionalnosti što ga čini manje prikladnim za općenitu upotrebu. Ali, i dan danas je veoma koristan alat programerima ugradbenih sustava.

Slika 3.1: BASCOM-8051 logo



Slika 3.2: BASCOM-8051 sučelje

Poglavlje 4

Progamski kod u BASCOM-u

Na početku se deklariraju varijable za daljnu upotrebu. **Byte** označava 8-bitni broj bez predznaka, a **Integer** je 16-bitni broj s predznakom (od -32768 do 32767). **Word** se odnosi na 16-bitni broj s predznakom. Oznaka **Dim** inicijalizira te varijable tako da se mogu koristiti za pohranu podataka respektivnog tipa. LCD displej spajamo na nožice (pinove) mikrokontrolera u softveru prije kompajliranja i pokretanja programa.

Pinovi LCD displeja:

- Enable (E) - održava LCD uključenim kako bi dohvaćao podatke
- Register Select (RS) - odabire podatkovne ili kontrolne registre
- DB4-DB7 - dohvaćaju podatke u 4-bit modu
- Serial Clock Line (SCL) - prenosi signal okidanja da bi sinkronizirao komunikaciju
- Serial Data Line (SDA) - omogućava prijenos podataka u oba smjera

Zatim slijedi konfiguracija LCD displeja koja koristi 1-wire komunikacijski protokol koji se koristi za uređaje sa senzorima temperature. Grijač je u početku ugašen. Čitamo zadnju temperaturu iz EEPROM-a koja je bila zadana. Ispisuje prvo moje ime i prezime, a zatim fakultet i godinu izrade rada. Rad koristi I2C (Inter-Integrated Circuit) komunikacijski protokol za sve dijelove koji se spajaju na glavnu sabirnicu. U tom slučaju je mikrokontroler ”master”, a ostali dijelovi imaju naziv ”slave”. Pri početku rada potrebno je resetirati senzor, zanemariti ono što od zadnjeg korištenja uređaja piše u EEPROM-u vezano uz trenutnu temperaturu. Dohvaćamo novu trenutnu temperaturu iz memorije i to redom LSB (least significant bit) i MSB (most significant bit). Koristimo operacije nad binarnim operandima kako bi izdvojili cijeli i decimalni dio temperature. Na LCD displeju tada može ispisati zadanu i trenutnu temperaturu.

Funkcionalnosti tipki na uređaju:

- Ako je pritisnuta tipka za plus zadana temperatura se poveća za 1 i tada se nova temperatura zapiše na memorijsku lokaciju u EEPROM-u,
- Ako je pritisnuta tipka za minus zadana temperatura se smanji za 1 i tada se nova temperatura zapiše na memorijsku lokaciju u EEPROM-u,
- Ukoliko pritisnemo tipku za paljenje grijača, u uvjetu u petlji se konačno pali grijač ($P1.6 = 1$). Ako je pritisnuta ponovno, grijač se ponovo gasi ($P1.6 = 0$).

4.1 Programski kod

```
1           $large  'kompajliraj kao veliki memorijski model'
2  Dim Pp As Byte , T1 As Byte , T2 As Byte , T3 As Byte ,
3  Temp As Integer
4  Dim Tem As Byte , A As Word
5
6  '   DISPLEJ E = P1.0 , RS = P1.1 , DB4 = P1.2 , DB5 = P1.3
7              , DB6 = P1.4 , DB7 = P1.5
8              , SCL = P3.2 , SDA = P3.1
9  '
10
11 Config 1wire = P3.0      'komunikacijski protokol'
12
13 Config Lcd = 16 * 2      '16 kolona i 2 reda na LCD displeju'
14 Config Lcdbus = 4        '4-bitni mod rada'
15 Cursor Off               'ne prikazuj kursor (ne blinkaj)'
16 Cls                      'obri i ekran'
17
18 '***** Deflcdchar ?,234,228,238,241,236,230,241,238 *****
19  replace ? with number (0-7) *** 89C4051 ***'
20
21 'Chr(1) prvo slovo prezimena'
22 Deflcdchar 1 , 234 , 228 , 238 , 241 , 236 , 230 , 241 , 238
23 'Chr(2) drugo slovo prezimena'
24 Deflcdchar 2 , 224 , 226 , 228 , 238 , 241 , 240 , 241 , 238
25
```

```
26
27 '**********OSNOVNA PETLJA**********'
28
29 P1.6 = 0    'grijac ugasen'
30 Pp = 1
31
32 Gosub Citaeeprom    'procitaj zadnje zadanu temperaturu'
33
34 Cls
35
36 For A = 1 To 20
37
38 Locate 1, A
39 Lcd "Oliver"
40 Locate 2 , A
41 Lcd Chr(1) ; "ajni" ; Chr(2)
42 If A < 20 Then
43 Waitms 500
44 Waitms 200
45 End If
46 Cls
47
48 Next
49
50 For A = 1 To 20
51
52 Locate 1 , A
53 Lcd "PMF" ; Chr(45) ; "MATEMATIKA"  'Chr(45) je -'
54 Locate 2 , A
55 Lcd "2023"
56 If A < 20 Then
57 Waitms 500
58 Waitms 200
59 End If
60 Cls
61
62 Next
63
64
```

```
65
66
67  Do
68
69  Gosub Temperatura
70
71  If Pp = 1 Then P1.6 = 0   'grijac ugasen, ne pali ga'
72
73  If Pp = 2 Then
74     If Tem > T1 Then
75     P1.6 = 1              'pali grijac'
76     Else
77        P1.6 = 0
78     End If
79     End If
80
81  Cls
82  Lcd " ZADANO  " ; Tem ; ",0 C"
83  Lowerline
84  Lcd "TRENUTNO " ; T1 ; "," ; T3 ; " C"
85
86  If Pp = 2 Then      'ako je upaljen grijac'
87  Locate 1 , 16
88  Lcd "*"             'prikazi na LCD *'
89  Locate 2 , 16
90  Lcd "*"
91  End If
92
93  Waitms 250
94
95  If P3.4 = 0 Then Gosub Plus     'tipka plus'
96  If P3.3 = 0 Then Gosub Minus    'tipka minus'
97  If P3.5 = 0 Then Gosub Pali     'tipka pali'
98
99  Loop
100
101 '*****************************************************'
102
103
```

```
104
105
106 Plus:
107
108 Tem = Tem + 1
109 Waitms 250
110 Gosub Upiseeprom
111 Return
112
113 Minus:
114
115 Tem = Tem - 1
116 Waitms 250
117 Gosub Upiseeprom
118 Return
119
120 '**************  UKLJUCI ILI ISKLJUCI RELEJ  *************'
121
122 Pali:
123
124 Pp = Pp + 1
125 If Pp > 2 Then Pp = 1
126 Waitms 250
127 Return
128
129 '****************  MJERI TEMPERATURU  ********************'
130
131 Temperatura:
132
133 1wreset              'resetiraj senzor'
134 1wwrite &HCC         'zanemari ROM'
135 1wwrite &H44         'startaj senzor'
136 Waitms 250
137 1wreset
138 1wwrite &HCC
139 1wwrite &HBE         'dohvati temperaturu iz memorije'
140 T1 = 1wread()        'ocitaj nizi bajt temperature (LSB)'
141 T2 = 1wread()        'ocitaj visi bajt temperature (MSB)'
142 1wreset
```

```
143
144
145 '
146 Temp = Makeint(T1 , T2)
147 T1 = Temp / 2
148
149 T1 = T1 - 200
150 T2 = Temp Mod 2
151
152 T3 = T2 And 1
153 If T3 = 1 Then T3 = 5
154 '
155                                 'spoji T1 i T2'
156 Temp = Makeint(T1 , T2)         'Temp = (T2 * 256) + T1'
157                                 'T1 je cijeli dio temperature'
158 T1 = Temp * 10                  'makni predznak'
159 T1 = Temp / 16                  'pomak udesno za 4 bita'
160
161 T2 = Temp Mod 2                 'LSB'
162
163 T3 = T2 And 1
164 If T3 = 1 Then T3 = 5
165
166 Return
167
168 '**********************  EEPROM  ************************'
169
170 Citaeeprom:
171
172 I2cstart                    'zapocni I2C komunikaciju (start)'
173 I2cwbyte &B10100000         'posalji adresu EEPROM-a za pisanje'
174 I2cwbyte 10                 'posalji adresu memorijskog mjesta
175                             u EEPROM-u'
176 I2cstart                    'ponovi start'
177 I2cwbyte &B10100001         'posalji adresu EEPROM-a za citanje'
178 I2crbyte Tem , 9            'spremi u Tem, zavrsi citanje'
179
180 Return
181
```

```
182
183 Upiseeprom:
184
185 I2cstart
186 I2cwbyte &B10100000
187 I2cwbyte 10            'na adresu u EEPROM-u upisi Tem'
188 I2cwbyte Tem
189
190 I2cstop               'prekini I2C komunikaciju (stop)'
191 Waitms 50             'sacekaj 50ms da EEPROM upise podatak'
192
193 Return
```

Poglavlje 5

Simulacija rada u Pythonu

5.1 PID kontroler

Za izvedbu simulacije rada PID kontrolera u Pythonu potrebno je nekoliko biblioteka: time, random, tkinter i matplotlib. U nastavku su objašnjeni i povezani ključni dijelovi programskog koda.

Main

Postavljamo zadanu temperaturu na 40 stupnjeva. Zatim isto tako definiramo vrijednosti proporcionalnog, integralnog i derivativnog pojačanja (njih možemo proizvoljno mijenjati). Tada stvaramo naš (PID) kontroler, sustav i aplikaciju koristeći ranije definirane klase.

Klasa PIDKontroler

Pozivanjem ove klase kontroler poprima zadanu temperaturu i pojačanja. Unutar klase još se dodjeljuju i inicijaliziraju varijable za zadnju korekciju i integral čije vrijednosti su na početku 0. U klasi se nalazi i funkcija za kontrolu koja računa korekciju kao razliku zadane i trenutne temperature, provjerava je li grijač upaljen (ako nije postavlja integral na 0 jer tada ne treba računati grešku), računa se derivacija kao razlika zadnjih dviju grešaka te se stvara izlazna vrijednost.

Klasa Sustav

Pozivanjem ove klase postavljamo početnu (sobnu) temperaturu na 25 stupnjeva. U početku rada grijač je upaljen i inicijaliziran je pad temperature. Unutar te klase definirana je funkcija za ažuriranje temperature kojoj prosljeđujemo promjenu koju izračunava ranija

funkcija kontrole i zadanu temperaturu. Ovisno o tome je li grijač upaljen ili ugašen stvaraju se realne promjene temperature. Ako se grijač ugasi, tada se događa nelinearan pad temperature koji simulira realan pad temperature. Definiramo i funkciju kojoj je zadaća da vrati temperaturu.

Klasa App

U ovoj klasi je razrađena gotovo sva logika programa. Pozivanjem ove klase kreira se grafičko sučelje za aplikaciju koja prati rad sustava i u isto vrijeme je interaktivna jer dopušta upis zadane temperature i paljenje/gašenje grijača. Kao parametre prosljeđujemo joj kontroler i sustav. Kreiramo dvije prazne liste u koje spremamo podatke o trenutnoj i zadanoj temperaturi u trenutku t. Poziva dvije osnovne funkcije ključne za rad:

```
napravi_sucelje()   i   azuriraj_temperaturu()
```

koje će kasnije imati svaka svoje objašnjenje. Osim njih koristimo funkciju koja unosi promjenu zadane temperature iz aplikacije, prosljeđuje ju kontroleru i ažurira promjenu na sučelju aplikacije. Dodatno postoji funkcija koja služi za paljenje/gašenje grijača i pritom samo mijenja varijablu, koja sadrži trenutno stanje grijača, na True ili Flase.

Funkcija napravi_sucelje ()

Pozivanjem ove funkcije kreiraju se nazivi za trenutnu i zadanu temperaturu, mjesto za upis zadane temperature, gumbi za postavljanje nove zadane temperature i uključivanje ili isključivanje grijača. Formira se graf 6x4 incha s rezolucijom 100 dpi-a koji prikazuje graf zadane (plave linije) i graf trenutne temperature (narančaste linije). Graf ima x i y os te legendu koja se automatski pomiče kako ne bi ometala prikaz grafa.

Funkcija azuriraj_temperaturu()

Funkcija poziva sebe svakih 1250 milisekundi. Dohvaća trenutnu temperaturu, računa promjenu i ažurira trenutnu temperaturu. Sprema podatke o trenutnoj i zadanoj temperaturi u svakoj iteraciji. Također mora svaki put promijeniti u aplikaciji trenutnu temperaturu. Predaje grafu podatke o trenutnoj i zadanoj temperaturi (cijele liste) kako bi u svakom trenutku graf znao prikazati potpune grafove ovih temperatura od početka do trenutne iteracije.

5.2 PID kontroler - programski kod

```
1  import time
2  import random
3  import tkinter as tk
4  from tkinter import ttk
5  from matplotlib.figure import Figure
6  from matplotlib.backends.backend_tkagg import
7                                          FigureCanvasTkAgg
8
9  class PIDKontroler:
10
11     def __init__(self, kp, ki, kd, zadana_temp):
12         self.kp = kp      # Proporcionalno pojacanje
13         self.ki = ki      # Integralno pojacanje
14         self.kd = kd      # Derivativno pojacanje
15         self.zadana_temp = zadana_temp
16         self.zadnja_korekcija = 0
17         self.integral = 0
18
19         # Kontrola temperature
20     def control(self, trenutna_temp, grijac_upaljen):
21
22         korekcija = self.zadana_temp - trenutna_temp
23         if grijac_upaljen:
24             self.integral += korekcija
25         else:
26             self.integral = 0
27         derivative = korekcija - self.zadnja_korekcija
28         izlaz = self.kp * korekcija + self.ki * self.integral
29                 + self.kd * derivative
30         self.zadnja_korekcija = korekcija
31
32         return izlaz
33
34
35
36
37
```

```
38
39
40
41 class Sustav:
42
43     def __init__(self, pocetna_temp = 25): # Pocetak rada
44         self.temp = pocetna_temp
45         self.grijac_upaljen = True
46         self.temp_sobe = pocetna_temp
47         self.pad_temp = 0.1
48
49     # Simulacija promjene temperature
50     def update(self, promjena, zadana_temp):
51
52         if self.grijac_upaljen:
53
54             self.temp += promjena
55             if self.temp < zadana_temp:
56                 # Dodaje realne promjene temperature
57                 self.temp += random.uniform(0.1, 0.2)
58             else:
59                 self.temp += random.uniform(-0.2, -0.1)
60
61         else:
62             if self.temp > self.temp_sobe:
63                 # Nelinearan pad temperature
64                 self.temp -= self.pad_temp *
65                             (self.temp - self.temp_sobe)
66
67
68     def dohvat_temp(self):
69         return self.temp
70
71
72
73
74
75
76
```

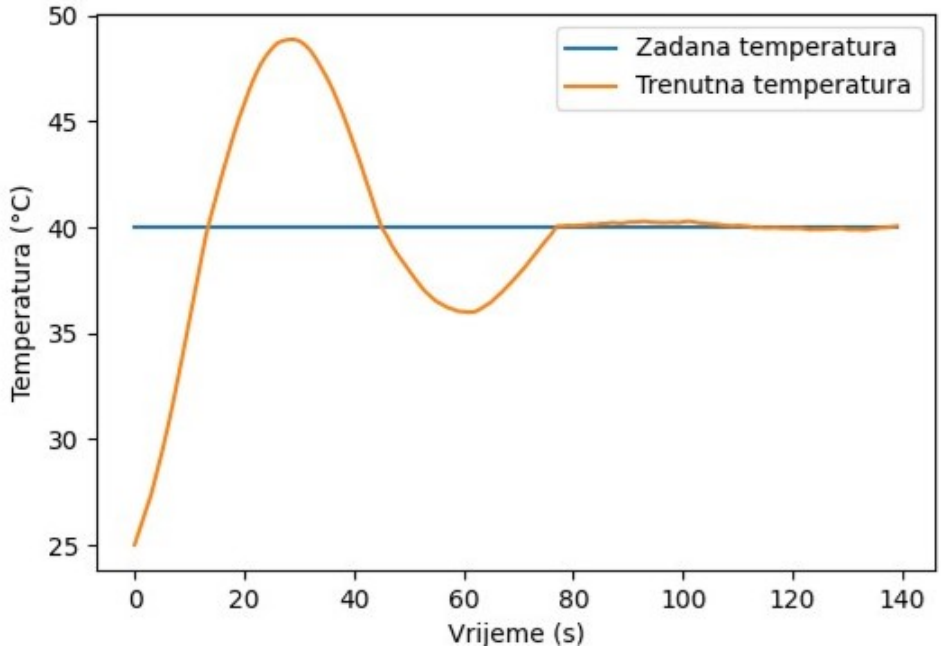
```
77
78
79
80
81  class App(tk.Tk):
82
83      def __init__(self, kontroler, sustav):
84          super().__init__() # nasljedi sve iz 'tk' klase
85          self.title("PID Kontroler GUI")
86          self.kontroler = kontroler
87          self.sustav = sustav
88          self.zadana_temp_list = []
89          self.trenutna_temp_list = []
90
91          self.napravi_sucelje()
92          self.azuriraj_temperaturu()
93
94      def napravi_sucelje(self):
95          # nazivi
96          self.trenutna_temp_naziv = ttk.Label(self,
97                      text = "Trenutna temperatura: 0 C ",
98                      font = ("Arial", 16))
99          # vertikalni razmak 10 piksela
100         self.trenutna_temp_naziv.pack(pady = 10)
101
102         self.zadana_temp_naziv = ttk.Label(self,
103                 text = "Zadana temperatura: 40.0 C ",
104                 font = ("Arial", 16))
105         self.zadana_temp_naziv.pack(pady = 10)
106
107         # mjesto za unos zadane temperature
108         self.zadana_temp_upis = ttk.Entry(self)
109         self.zadana_temp_upis.insert(0, "40.0")
110         self.zadana_temp_upis.pack(pady = 10)
111
112
113
114
115
```

```
116         # gumbi
117         self.gumb_postavi_temp = ttk.Button(self,
118                         text = "Postavi temperaturu",
119                         command = self.postavi_zadanu_temp)
120         self.gumb_postavi_temp.pack(pady = 10)
121 
122         self.gumb_pali_gasi_grijac = ttk.Button(self,
123                         text = "Ukljuci/Iskljuci grijac",
124                         command = self.pali_gasi_grijac)
125         self.gumb_pali_gasi_grijac.pack(pady = 10)
126 
127         # graf 6x4 incha s rezolucijom 100 dpi
128         self.graf = Figure(figsize = (6, 4), dpi = 100)
129         # 1 red, 1 stupac, 1 graf
130         self.ax = self.graf.add_subplot(111)
131         # graf zadane temperature
132         self.graf1, = self.ax.plot(self.zadana_temp_list,
133                               label = 'Zadana temperatura')
134         # graf trenutne temperature
135         self.graf2, = self.ax.plot(self.trenutna_temp_list,
136                               label = 'Trenutna temperatura')
137         self.ax.set_xlabel('Vrijeme (s)')
138         self.ax.set_ylabel('Temperatura ( C )')
139         self.ax.legend() # legenda za graf
140 
141         # podloga za graf unutar prozora
142         self.podloga = FigureCanvasTkAgg(self.graf,
143                                          master = self)
144         self.podloga.get_tk_widget().pack()
145 
146     def postavi_zadanu_temp(self):
147         try:
148             # procitaj unesenu temperaturu
149             zadana_temp = float(self.zadana_temp_upis.get())
150             # postavi zadanu temperaturu u kontroleru
151             self.kontroler.zadana_temp = zadana_temp
152             # azuriraj promjenu u sucelju
153             self.zadana_temp_naziv.config(
154             text=f"Zadana temperatura: {zadana_temp:.2f} C ")
```

```
155
156         except ValueError:
157             pass  # greska
158
159     def pali_gasi_grijac(self):
160         self.sustav.grijac_upaljen =
161                         not self.sustav.grijac_upaljen
162
163     def azuriraj_temperaturu(self):
164         trenutna_temp = self.sustav.dohvat_temp()
165         promjena = self.kontroler.control(trenutna_temp,
166                                 self.sustav.grijac_upaljen)
167         self.sustav.update(promjena,
168                            self.kontroler.zadana_temp)
169
170         self.zadana_temp_list.append(
171             self.kontroler.zadana_temp)
172         self.trenutna_temp_list.append(trenutna_temp)
173
174         # azuriraj promjenu u sucelju
175         self.trenutna_temp_naziv.config(
176         text=f"Trenutna temperatura: {trenutna_temp:.2f} C ")
177
178         self.graf1.set_data(
179                 range(len(self.zadana_temp_list)),
180                 self.zadana_temp_list)
181         self.graf2.set_data(
182                 range(len(self.trenutna_temp_list)),
183                 self.trenutna_temp_list)
184         self.ax.relim() # brine o granicama grafa
185         self.ax.autoscale_view() # auto skaliranje grafa
186
187         self.podloga.draw()
188
189         # funkcija poziva sebe svakih 1250 ms
190         self.after(1250, self.azuriraj_temperaturu)
191
192
193
```

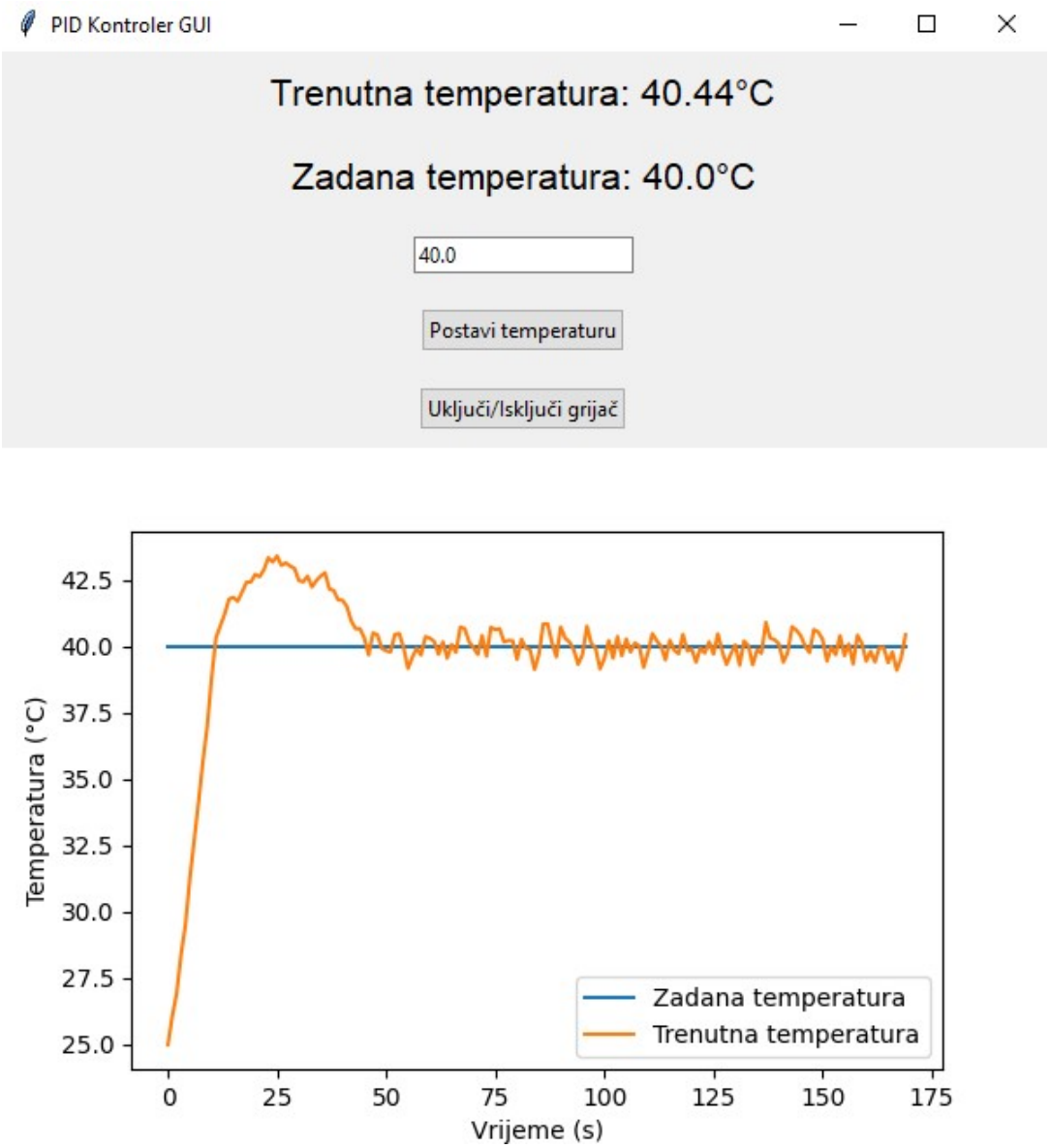
```
194
195 def main():
196
197     zadana_temp = 40.0
198     kp, ki, kd = 0.02, 0.01, 0.01  # PID kontroler konstante
199
200     kontroler = PIDKontroler(kp, ki, kd, zadana_temp)
201     sustav = Sustav()
202
203     app = App(kontroler, sustav)
204     app.mainloop()
205
206 main()
```

5.3 PID kontroler - rezultat



Slika 5.1: Graf PID kontrole

5.4 Aplikacija



Slika 5.2: Aplikacija za simulacija rada

5.5 LQG kontroler

Programski kod koji simulira rad LQG kontrolera koristi LQR kontrolu i Kalmanov filter. Raniji teorijski dio obuhvaća samu implementaciju. Dobar dio programskog koda nalik je na programski kod PID kontrole, pa njega nećemo ponovo objašnjavati. Sada ćemo samo ukratko objasniti dijelove vezane uz samu LQG kontrolu i Kalmanov filter.

Klasa LQGKontroler

Ovoj klasi prosljeđuju se matrice i zadana temperatura koje su unaprijed definirane u glavnom programu. Računa kontrolnu matricu K pozivom funkcije za optimalnu kontrolu. Zatim klasi KalmanFilter prosljeđuje potrebne parametre. Sadrži i funkciju za kontrolu koju koristi kako bi ažurirao procjenu stanja $\hat{x}$ i kontrolni ulaz u te krenuo sa predikcijom stanja.

5.6 LQG kontroler - programski kod

```
1
2
3  import time
4  import random
5  import numpy as np
6  import tkinter as tk
7  from tkinter import ttk
8  from matplotlib.figure import Figure
9  from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
10
11 class KalmanFilter:
12     def __init__(self, A, B, H, Q, R):
13         self.A = A      # matrica stanja
14         self.B = B      # matrica kontrolnog ulaza
15         self.H = H      # matrica mjerenja
16         self.Q = Q      # matrica procesnog suma
17         self.R = R      # matrica mjernog suma
18         self.x = np.zeros((A.shape[0], 1)) # procjena stanja
19         self.P = np.eye(A.shape[0]) # matrica procjene greske
20
21
22
```

```
23
24
25
26      # predikcija Kalmanovog filtera
27      def predvidi(self, u):
28
29          self.x = self.A @ self.x + self.B @ u
30          self.P = self.A @ self.P @ self.A.T + self.Q
31          return self.x
32
33      # azuriraj Kalmanov filter
34      def update(self, z):
35
36          # razlika izmedu stvarnog mjerenja i predikcije
37          y = z - self.H @ self.x
38          # procjena nesigurnosti
39          S = self.H @ self.P @ self.H.T + self.R
40          # Kalmanov dobitak
41          K_f = self.P @ self.H.T @ np.linalg.inv(S)
42          self.x = self.x + K_f @ y
43          self.P = self.P - K_f @ self.H @ self.P
44          return self.x
45
46  class LQGKontroler:
47      def __init__(self, A, B, Q, R, zadana_temp):
48          self.A = A       # matrica stanja sustava
49          self.B = B       # matrica kontrolnog ulaza
50          self.Q = Q       # matrica tezine stanja
51          self.R = R       # matrica tezine kontrolnog ulaza
52          self.zadana_temp = zadana_temp
53          self.K = self.optimalna_kontrola()
54          self.kalman_filter = KalmanFilter(A, B,
55              np.eye(A.shape[0]), 0.1 * np.eye(A.shape[0]),
56              0.1 * np.eye(A.shape[0]))
57
58
59
60
61
```

```
62
63
64
65
66      def optimalna_kontrola(self):  # racuna matricu K
67
68          # inicijalizacija P matrice
69          S = np.matrix(np.zeros(self.Q.shape))
70          broj_iteracija = 150
71          eps = 0.01  # tolerancija za konvergenciju
72          # inicijalizacija kontrolne matrice K
73          K = np.zeros((self.B.shape[1], self.A.shape[0]))
74          for i in range(broj_iteracija):
75              K_nova = -np.linalg.inv(self.R) @ self.B.T @ S
76              S_nova = self.A.T @ S + S @ self.A + self.Q -
77                      S @ self.B @ np.linalg.inv(self.R) @
78                      self.B.T @ S
79              if np.linalg.norm(K_nova - K) < eps:
80                  break
81              K = K_nova
82              S = S_nova
83          return K
84
85      def control(self, trenutna_temp):
86
87          x_potez = self.kalman_filter.update(
88                      np.array([[trenutna_temp]]))
89          u = -self.K @ (x_potez -
90                          np.array([[self.zadana_temp]]))
91          self.kalman_filter.predvidi(u)
92          return u
93
94
95
96
97
98
99
100
```

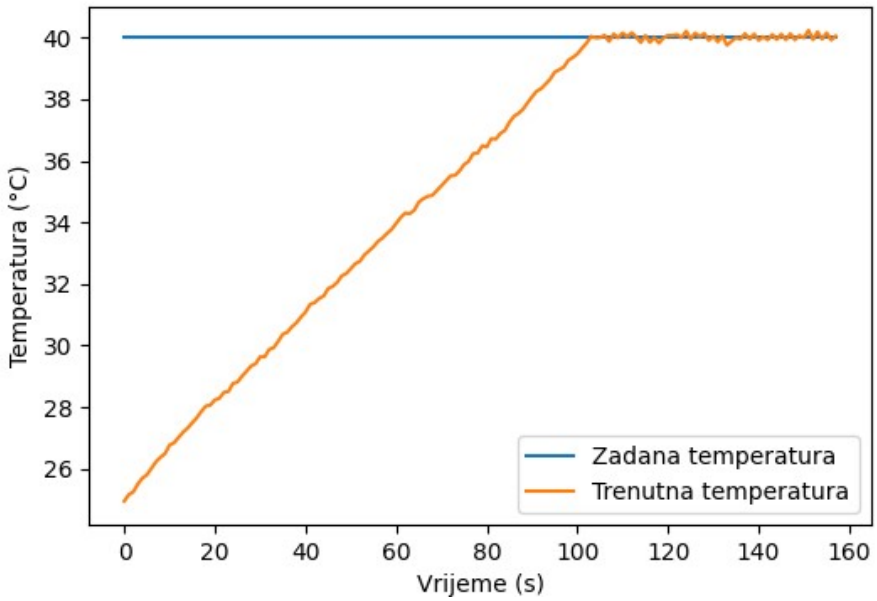
```
101 class Sustav:
102     def __init__(self, pocetna_temp = 25):
103         self.temp = pocetna_temp
104         self.grijac_upaljen = True
105         self.temp_sobe = pocetna_temp
106         self.pad_temp = 0.1
107 
108     def update(self, promjena, zadana_temp):
109 
110         vrijednost = promjena[0, 0]  # pozicija (0,0)
111         if self.grijac_upaljen:
112             self.temp += vrijednost
113             if self.temp < zadana_temp:
114                 self.temp += random.uniform(0.1, 0.15)
115             else:
116                 self.temp += random.uniform(-0.15, -0.1)
117         else:
118             if self.temp > self.temp_sobe:
119                 self.temp -= self.pad_temp * (
120                             self.temp - self.temp_sobe)
121 
122     def dohvat_temp(self):
123         # simuliraj buku
124         return self.temp + random.gauss(0, 0.05)
125 
126 class App(tk.Tk):
127     def __init__(self, kontroler, sustav):
128         # za pozive konstruktora nadklase tk.Tk
129         super().__init__()
130         self.title("LQG Kontroler GUI")
131         self.kontroler = kontroler
132         self.sustav = sustav
133         self.zadana_temp_list = []
134         self.trenutna_temp_list = []
135 
136         self.napravi_sucelje()
137         self.azuriraj_temperaturu()
138 
139 
```

```
140     def napravi_sucelje(self):
141
142         self.trenutna_temp_naziv = ttk.Label(self,
143                     text = "Trenutna temperatura: 0 C ",
144                     font = ("Arial", 16))
145         self.trenutna_temp_naziv.pack(pady = 10)
146
147         self.zadana_temp_naziv = ttk.Label(self,
148                 text = "Zadana temperatura: 40.0 C ",
149                 font = ("Arial", 16))
150         self.zadana_temp_naziv.pack(pady = 10)
151
152         self.zadana_temp_upis = ttk.Entry(self)
153         self.zadana_temp_upis.insert(0, "40.0")
154         self.zadana_temp_upis.pack(pady = 10)
155
156         self.gumb_postavi_temp = ttk.Button(self,
157                         text = "Postavi temperaturu",
158                         command = self.postavi_zadanu_temp)
159         self.gumb_postavi_temp.pack(pady = 10)
160
161         self.gumb_pali_gasi_grijac = ttk.Button(self,
162                         text = "Uklju i/Isklju i grija ",
163                         command = self.pali_gasi_grijac)
164         self.gumb_pali_gasi_grijac.pack(pady = 10)
165
166         self.graf = Figure(figsize = (6, 4), dpi = 100)
167         self.ax = self.graf.add_subplot(111)
168         self.graf1, = self.ax.plot(self.zadana_temp_list,
169                             label = 'Zadana temperatura')
170         self.graf2, = self.ax.plot(self.trenutna_temp_list,
171                             label = 'Trenutna temperatura')
172         self.ax.set_xlabel('Vrijeme (s)')
173         self.ax.set_ylabel('Temperatura ( C )')
174         self.ax.legend()
175
176         self.canvas = FigureCanvasTkAgg(self.graf,
177                                         master = self)
178         self.canvas.get_tk_widget().pack()
```

```
179     def postavi_zadanu_temp(self):
180
181         try:
182             zadana_temp = float(self.zadana_temp_upis.get())
183             self.kontroler.zadana_temp = zadana_temp
184             self.zadana_temp_naziv.config(
185             text=f"Zadana temperatura: {zadana_temp:.2f} C ")
186         except ValueError:
187             pass
188
189     def pali_gasi_grijac(self):
190         self.sustav.grijac_upaljen =
191                             not self.sustav.grijac_upaljen
192
193     def azuriraj_temperaturu(self):
194
195         trenutna_temp = self.sustav.dohvat_temp()
196         promjena = self.kontroler.control(trenutna_temp)
197         self.sustav.update(promjena,
198                             self.kontroler.zadana_temp)
199
200         self.zadana_temp_list.append(
201                             self.kontroler.zadana_temp)
202         self.trenutna_temp_list.append(trenutna_temp)
203
204         self.trenutna_temp_naziv.config(
205         text=f"Trenutna temperatura: {trenutna_temp:.2f} C ")
206
207         self.graf1.set_data(
208                     range(len(self.zadana_temp_list)),
209                     self.zadana_temp_list)
210         self.graf2.set_data(
211                     range(len(self.trenutna_temp_list)),
212                     self.trenutna_temp_list)
213         self.ax.relim()
214         self.ax.autoscale_view()
215         self.canvas.draw()
216
217         self.after(1250, self.azuriraj_temperaturu)
```

```
218 def main():
219 
220     zadana_temp = 40
221     pocetna_temp = 25
222     A = np.array([[1, 0], [0, 1]])
223     B = np.array([[1, 0], [0, 1]])
224     Q = np.array([[1, 0], [0, 1]])
225     R = np.array([[1, 0], [0, 1]])
226 
227     kontroler = LQGKontroler(A, B, Q, R, zadana_temp)
228     sustav = Sustav(pocetna_temp)
229     app = App(kontroler, sustav)
230     app.mainloop()
231 
232 main()
```

5.7 LQG kontroler - rezultat



Slika 5.3: Graf LQG kontrole

Poglavlje 6

Usporedba fizičkog rada i simulacije

6.1 Programski kod u BASCOM-u i Pythonu

Odmah vidimo na prvi pogled da je programski kod u BASCOM-u kraći u odnosu na programski kod u Pythonu. Potrebno je naglasiti kako ova dva programska koda nisu u potpunosti ekvivalentni. Ne zato što se razlikuju, već time što se u fizičkom radu koristi jeftini mikrokontroler koji se koristi u sustavima koji ne zahtijevaju preveliku preciznost, dok je PID mikrokontroler prilagodljiv složenijim sustavima i pruža visoku preciznost. Programski kod u BASCOM-u je jednostavan i izravan. Nekada može biti malo teži za razumjeti jer koristi binarne podatke i binarne adrese, pa je potrebno dobro poznavati operacije i logiku pohranjivanja, dohvaćanja i čitanja podataka.

Za razliku od BASCOM-a, Python pruža veću fleksibilnost i jednostavniju implementaciju, makar je programski kod u njemu dulji. Programski kod je poprilično samogovoreći. Ima mnoštvo modernih biblioteka, kao i mogućnost izrade grafičkog sučelja, što nije tako lagano implementirati u BASCOM-u. Pruža mnoštvo opcija i precizno navođenje i specificiranje svega što se u njemu koristi.

Valja naglasiti da svaki programski jezik ima svoje prednosti i mane, ali isto tako da su osmišljeni za specifične projekte i okruženja u kojima se koriste.

6.2 Funkcionalnost fizičkog rada i simulacije

Počnimo od fizičkog rada. Jasno je vidljivo kako postoje razlike između fizičkog rada i simulacije. Dobrim dijelom za to su odgovorni mikrokontroleri. U fizičkom radu koristimo stariji, jeftin i manje precizan Atlemlov mikrokontoler, a u simulaciji simuliramo rad skupljeg, preciznog i pouzdanog PID mikrokontrolera ili još boljeg LQG mikrokontrolera. Postoje i još neke razlike, na primjer položaj senzora u odnosu na grijač koji sam odabrao. Očito i radi toga dobivam sporije hlađenje kutije u kojoj se nalazi grijač i toliko odstupanje od zadane temperature (otprilike 5°C) kada se grijač automatski upali (u slučaju kada trenutna temperatura padne ispod zadane temperature).

Valjalo bi sada dati neke argumente zašto je ipak pogodna ovakva izvedba. Zamislimo da grijemo vodu ili med u posudi. Želimo da voda ili med poprime tu zadanu temperaturu. Trebamo imati na umu također da će i oni sami preuzeti toplinu. Tu nailazimo na problem da vrlo vjerojatno temperature vode ili meda i grijača neće biti ujednačene, pogotovo njihovi krajnji dijelovi u posudi. Također, pošto će voda ili med preuzimati dio topline, to bi nam omogućilo brži pad temperature, pa smo na neki način ipak obrazložili da mala greška u izradi neće biti toliko izražena. Ali, nije ovo zaključak do kojega smo htjeli doći.

Prisjetimo se principa rada termoregulatora i centralnog grijanja. Neka se termoregulator nalazi na jednoj strani prostorije na zidu, a neka je radijator na drugoj strani prostorije nasuprot njemu. Recimo da se želi postići temperatura 23°C u prostoriji. U termoregulatoru se nalazi senzor. Kreće grijanje koje traje sve dok prostorija nasuprot radijatoru ne postigne 23°C i onda prestaje grijanje. Znači nije cilj postići temperaturu od 23°C na mjestu gdje je radijator, već zagrijati cijelu prostoriju na zadanu temperaturu. Istina je da će temperatura biti nešto veća što smo bliži izvoru grijanja.

Isto tako i u korištenju ovog rada želi se postići da cijela posuda postigne tu temperaturu (ili da joj se dosta približi). Dio vode ili meda oko kutije u kojoj se nalazi grijač će imati nešto veću temperaturu u odnosu na ostatak posude, ali će se toplina bolje raspodijeliti po ostatku posude u odnosu na to da koristimo PID ili LQG kontrolu u stvarnosti. U ovom slučaju ovakva izvedba fizičkog rada zapravo puno bolje odgovara grijanju vode ili meda što je čini pogodnom i efikasnom realizacijom za njegovu specifičnu upotrebu.

Što se tiče simulacija u Pythonu, jasno je vidljiva i razlika samih simulacija. Simulacija koja simulira rad PID mikrokontrolera ima vidljivo odstupanje od zadane temperature u nekom vremenskom intervalu nakon što postigne tu zadanu temperaturu, ali se kasnije stabilizra na toj zadanoj temperaturi. Kod simulacije LQG mikrokontrolera u trenutku kada se postigne zadana temperatura, ona odmah ostaje na toj zadanoj temperaturi. Prema

tome, ova simulacija je najbliža realizacija perfektnog uređaja za kontrolu temperature u smislu preciznosti, pouzdanosti, funkcionalnosti i održavanja temperature.

6.3 Poteškoće pri izradi fizičkog rada i simulacije

Znao sam od početka da mogu izvesti što sam si zadao napraviti, ali išao sam s spoznajom da će se tu i tamo javiti neiščekivani problemi koliko god ja to ne htio. Početak izrade išao je relativno glatko. Prvi veći izazov bio je osmisliti kako izvesti i napraviti PCB layout pomoću učitane električne sheme. Raspored nekih vodova morao je biti malo zgusnut. Najvjerojatnije iz tog razloga mi prvi put pločica nije radila prilikom sastavljanja uređaja. Morao sam ponovo započeti cijeli dio vezan uz izradu pločice. Nije mi bilo druge, nego se nadati najboljemu. Na svu sreću druga identična pločica je radila. Ali, to sam saznao tek kada sam prvi put spojio LCD displej i nakon što sam programirao chip, a drugi put je sve radilo bez problema. Tu se također javila mala poteškoća sa razumijevanjem koje nožice chipa spajam sa pinovima LCD displeja u smislu položaja i povezivanja E, RS i ostalih pinova kroz BASCOM-8051 softver.

Što se tiče pisanja koda u BASCOM-u imao sam sitnih problema u snalaženju u radu s adresama EEPROM-a, ali uz podršku koju BASCOM-AVR pruža na Internetu sam se lagano snašao. U usporedbi s Pythonom rekao bi da je Python bio lakši jer je svakako logičniji i s njime sam više u doticaju, a i prolazio sam tečaj za Python što mi je dobro došlo za izradu aplikacijskog sučelja. Jedine poteškoće u Pythonu zadavao mi je LQG kontroler zbog različitih literatura na Internetu. Bio sam u dilemi koju literaturu odabrati jer je riječ o kompleksnijoj implementaciji. No, na kraju sam se uspio odlučiti za kombinaciju dijelova iz pojedinih literatura ([5], [8], [9]) i nakon dosta prepravljanja programskog koda i njegovog testiranja postigao sam željeni cilj, a to je precizna LQG kontrola.

Zaključak

Nagovor mentora da napravim simulacije u Pythonu bila je super ideja i poticaj da obogatim ovaj diplomski rad. Najpogodniji način njihove stvarne izvedbe bio bi uz Arduinov hardver. Pretpostavljam da bi neke dodatne dijelove poput senzora i grijača trebalo posebno nabaviti, ali ostatak izvedbe bi bio nalik simulaciji. Već smo istaknuli prednosti i mane ovih izvedbi kontrola grijanja. Sada možemo sagledati što bi se još dalo poboljšati oko fizičke izvedbe rada. Vrlo vjerojatno bi se rad poboljšao samim time da se senzor približi grijaču ili ugradnjom senzora visoke točnosti, no to izgleda ne bi u potpunosti riješilo problem. Jedno rješenje je kada bi se ispočetka radio fizički dio da se implementira PID regulator jer u BASCOM-u postoji podrška za PID kontrolu. Nažalost, izvedba LQG regulatora nije moguća u BASCOM-u jer ne podržava implementaciju složenih kontrolnih tehnika poput LQG kontrole. Drugo rješenje bilo bi ugradnja dodatnih senzora ili sučelja koja bi prilagodila rad sustava nalik onima koji se koriste u simulacijama. Možda bi i neke sigurnosne funkcije u programu pridonijele spriječavanju prekomjernog zagrijavanja. Sve to zahtjeva još dosta testiranja i optimizacija u realnom okruženju u kojemu se fizički rad koristi. Uz takve pothvate bi opseg diplomskog rada bio znatno veći, pa to ostavljamo za sljedeći projekt.

Bibliografija

[1] Marjan Anic, *Mikroračunala na lak način*, http://www.ss-strukovna-vvlatkovica-zd.skole.hr/images/pages/Nastavni_materijali/Spahic/mikroracunala/Miikroraunala_na_lak_nain.pdf.

[2] Senad Huseinbegović dipl.ing.el. Doc.dr. Sead Kreso, dipl.ing.el., *Projektovanje mikroprocesorskih sistema*, https://ssvv.hr/images/pages/Nastavni_materijali/Spahic/mikroracunala/Predavanje_9[1]_Sarajevo.PDF.

[3] dipl.inž.el. dr.sc. Jurica Kundrata, Vikica Lukić, *Izrada tiskanih pločica*, Og grafika d.o.o., Jastrebarsko, Sisak, 2023.

[4] Stefan Hoffmann, *A simple introduction to electronics with AVR microcontrollers and BASCOM*, 3rd edition., 2010.

[5] Mario Innocenti, *LQG Control*, pogl. 2, str. 1–18, 2021.

[6] Burkhard Kainka, *Basiskurs BASCOM-AVR*, 1st edition., 2011.

[7] Claus Kühnel, *Programming the AVR RISC microcontrollers with BASCOM-AVR*, 3rd edition., 2010.

[8] Tony Lacey, *Tutorial: The Kalman Filter*, pogl. 11, str. 1–8.

[9] Robert F. Stengel, *Optimal control and estimation*, 1st edition., Dover publications, Inc., New York, 1994.

[10] Roland Walter, *AVR microcontrollers textbook*, 3rd updated edition., 2009.

[11] Umar Waseem, *PID Controller Loops: A Comprehensive Guide to Understanding and Implementation*, (2023).

Sažetak

Fokus ovog rada je najviše na izvedbi programskog dijela rada. Želi se postići i objasniti glavnu izvedbu i primjenu rada. Manji dio rada odnosi se i na izradu fizičkog rada, a veći na njegovu izvedbu i funkcionalnost, kako fizičkog rada tako i simulacije u Pythonu. Prikazani su i objašnjeni svi koraci izrade, programski kod je detaljno obrazložen, samogovoreć i povezan s ostatkom koda. Izriče se jasna razlika između fizičkog rada i simulacije. Navedene su prednosti i nedostatci oba pristupa te njihova područja primjene. Rad daje dobar uvid i argumentaciju za njegove specifične primjene i razlaže zašto su neke metode pogodnije od drugih. Glavna stvar koja se želi iz svega postići je zainteresirati čitatelja, predočiti mu ideju, plan, sliku i funkciju ovog naizgled jednostavnog, ali itekako interesantnog, korisnog i primjenjivog rada.

Summary

The focus of this work is mainly on the implementation of the program part of the work. It is intended to achieve and explain the main performance and application of the work. A smaller part of the work also refers to the creation of the physical work, and a larger part to its performance and functionality, both of the physical work and the simulation in Python. All production steps are shown and explained, the programming code is explained in detail, self-explanatory and linked to the rest of the code. A clear distinction is made between physical work and simulation. The advantages and disadvantages of both approaches and their areas of application are listed. The paper provides good insight and arguments for its specific applications and explains why some methods are more suitable than others. What we want to achieve above all is to interest the reader, to present to him the idea, plan, image and function of this seemingly simple, but very interesting, useful and applicable work.

Životopis

Rođen sam 4. lipnja 1998. u Zagrebu. Dolazim iz malog mjesta zvanog Gradići u okolici Velike Gorice. U Velikoj Gorici sam pohađao Osnovnu školu Nikole Hribara, a obrazovanje sam nastavio u Tehničkoj školi Ruđera Boškovića u Zagrebu. Upisao sam ju jer sam htio imati struku i diplomu po završetku srednje škole. Pružala je jaku matematiku, elektrotehniku i programiranje koje me je nakon završetka navelo na upisivanje Prirodoslovno - matematičkog fakulteta na odsjeku Matematike 2017. godine. Obrazovanje na fakultetu bilo je usmjereno većinom na matematiku, ali zbog znanja stečenih u srednjoj školi upisivao sam što više informatičkih izbornih predmeta. Nakon završenog preddiplomskog studija, 2021. godine upisujem diplomski studij Matematika i Informatika; smjer: nastavnički. Tijekom ljetnog semestra na zadnjoj godini studija pojavio mi se interes za rad s bazama podataka i to na predmetu Napredne baze podataka. Sad skoro godinu dana paralelno studiram i radim kao BI (Business Intelligence) specijalist u firmi APIS IT. Ovim radom htio sam pokazati svoju privrženost području programiranja u koje dalje usmjeravam svoje obrazovanje, ali isto tako sam htio ukomponirati što više različitih vještina stečenih tijekom srednjoškolskog obrazovanja i obrazovanja na fakultetu. Nakon završetka studija, namjeravam nastaviti s usavršavanjem i istovremenom izgradnjom karijere u ovim područjima programiranja.