УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ

Борис Карановић

ПРИМЕНА ФАЗИ ЛОГИКЕ У УГРАДНИМ УРЕЂАЈИМА НА ПРИМЕРУ АУТОМАТИЗАЦИЈЕ ОДРЖАВАЊА ПЛАСТЕНИКА

мастер рад

Београд, 2023.

Ментор:

др Иван Чукић, доцент
Универзитет у Београду, Математички факултет

Чланови комисије:

др Весна Маринковић, доцент
Универзитет у Београду, Математички факултет

др Ивана Томашевић, доцент
Универзитет у Београду, Математички факултет

Датум одбране: _____

Наслов мастер рада: Примена фази логике у уградним уређајима на примеру аутоматизације одржавања пластеника

Резиме: Први део је уводног карактера у фази скупове, фази логику и фази закључивање. У њему су дефинисани фази скупови као и основне скуповне операције дефинисане над овим скуповима. Након тога се дефинишу фази правила закључивања, као и процес дефазификације. Процес дефазификације је превођење фази закључака у бројчану вредност која нам је од користи за даљи рад. Након дефинисања свих појмова приказан је комплетан алгоритам фази закључивања.

Други део рада се састоји од детаљног увида у конкретан проблем аутоматизације пластеника који се разматра, као и описа имплементације решења на уређају *Raspberry Pi*. Наведени су параметри које можемо да контролишемо у пластенику као и које улазне информације су нам за то потребне. У имплементацији ћемо контролисати микро-климу.

У трећем делу рада су приказана и анализирана постојећа решења за оптимизацију пластеника применом фази логике, и упоређена постојећа решења са предложеним. Такође су наведена и нека ограничења и проблеми везани за фази логику као и неки предлози за њихово решавање.

Кључне речи: фази логика, рачунарска интелигенција, рачунарска оптимизација, примена фази логике, Мамданијев фази систем

Садржај

Глава 1

Увод

Да ли је напољу топло или вруће? Да ли је овај човек висок или није? Да ли неко вози брзо или споро? Реални проблеми се често описују непрецизно и некомплетно. Додатно, одговоре на ова питања често дајемо субјективно. Закључивање на основу оваквих изјава је тешко моделовати класичном бинарном логиком која користи две истинитосне вредности - тачно и нетачно. Потребан нам је другачији логички систем, који користи више истинитосних вредности. Фази логика је један облик више-вредносне логике.

Фази логика се базира на чињеници да људи доносе одлуке на основу непрецизних информација које најчешће нису бројчане. Зато су и улазни подаци у фази системе реченице природног језика, као и наша процена или доживљај неког закључка.

Темеље фази логике је поставио азербејџански математичар *Lofti Zadeh* 1965. године са теоријом фази скупова. Фази логика је проучавана и пре 1965. године, у оквирима бесконачно-вредносних логика, од стране пољских математичара *Jana Łukasiewicza* и *Alfreda Tarskog*. Примена фази логике је данас вишеструка у пракси, а такође се често примењује у алгоритмима вештачке интелигенције и теорији управљања динамичким система. Навешћемо неке од подручја примене фази логике која се не налазе у опсегу рачунарства:

- У медицини, фази контролери се користе за дијагностику малигних обољења и дијабетеса, контролу крвног притиска током анестезије као и комплетну контролу процеса давања анестетика пацијентима, проналажењу разлога за појаву Алцхајмерове болести.

- У различитим производним и индустријским процесима, производњи

прехрамбених производа и техничких уређаја.

- У финансијском сектору, фази логика је брз и ефикасан начин за предвиђање вредности берзанских акција, као и за детектовање сумњивих банкарских трансакција.

- У свакодневном животу, фази контролери се често налазе у „паметним" уређајима: веш машинама, клима уређајима, микроталасним рернама, усисивачима, овлаживачима.

- Поред ових подручја примене, још нека интересантна су: контрола свемирских летелица, контрола саобраћаја, препознавање рукописа, гласа, као и анализа људског понашања у криминалистици.

Једна од примена које ће бити анализиране у овом раду је из домена пољопривреде и узгајања биљака, а то је аутоматизација одржавања пластеника.

Глава 2

Фази систем - скупови, логика, закључивање

2.1 Фази скупови, функција припадности

У класичној двовредносној логици, на питање да ли је напољу топло, имамо два одговора: *да* и *не*. За граничну вредност између ове две вредности можемо изабрати, на пример, вредност од 25°C. Можемо изабрати и било коју другу вредност, која би највероватније зависила од климатског подручја особе коју питамо. Ако узмемо за граничну вредност 25°C, онда би 24°C било хладно, док би 26°C било вруће, тј. имамо оштру границу на прелазима из једне истинитосне вредности у другу. Такође, за температуре од 26°C и 39°C ћемо рећи да су топле, иако између њих постоји велика разлика. Коришћење оштрих граница за описивање оваквих појава не налазимо у свакодневном животу [5].

Фази скупови нам омогућавају да се припадност неком скупу дефинише реалним бројем у интервалу [0,1]. Нека је X домен, и $A \subset X$ неки подскуп тог домена. Фази скуп A добијамо тако што за елементе из скупа A дефишемо функцију припадности $\mu_A : A \to [0, 1]$. Фази скупови се могу дефинисати над дискретним или реалним доменима. Могу се представити на два начина:

- Преко скупа уређених парова $A = \{(\mu_A, x_i) | x_i \in X, i = 1, ..., n_x\}$.
- Преко „суме" или „интеграла" у зависности да ли је домен дискретан или реалан. Наравно, ово нису праве суме и интеграли већ искључиво погодне нотације:

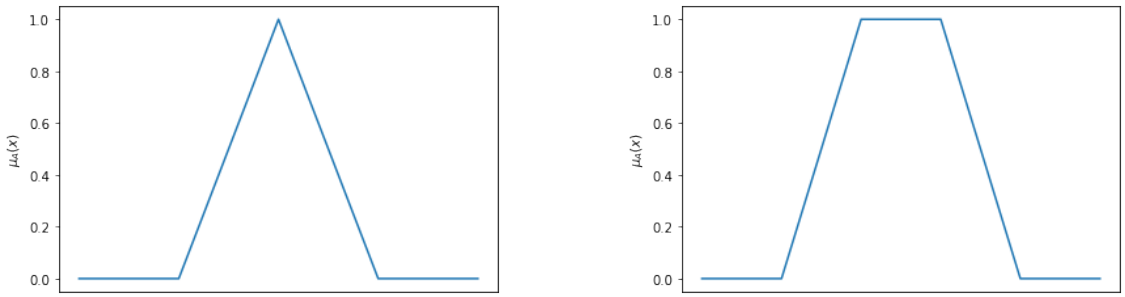
– $A = \sum_{i=1}^{n_x} \mu_A(x_i)/x_i$, за дискретан домен.

– $A = \int_X \mu_A(x)dx$, за реалан домен.

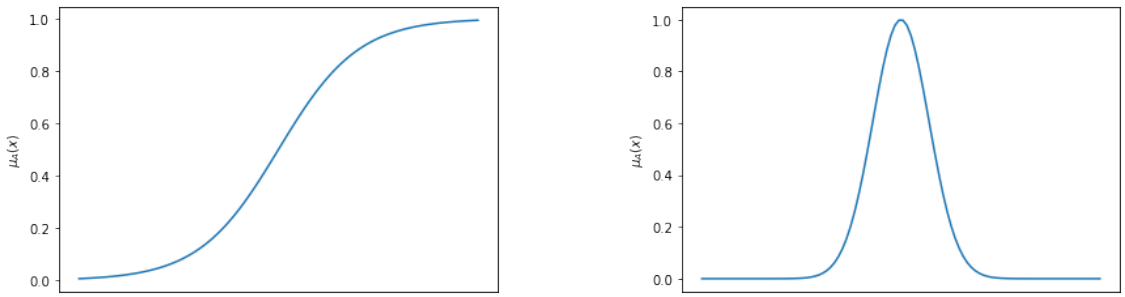
Фази функција припадности скупу испуњава следеће особине:

1. Ограничена је на интервалу [0,1].
2. За сваки елемент домена је једнозначна.

Неке од стандардних фази функција припадности имају следеће облике:



Слика 2.1: Троугаона и трапезоидна функција припадности



Слика 2.2: Сигмоидна и Гаусова функција припадности

Основне скуповне операције се могу дефинисати тако да важе и за фази скупове. Сада ћемо дефинисати и неке од њих:

- **Једнакост скупова** – два фази скупа су једнака ако имају исти домен и ако је за сваки елемент тог домена вредност функције припадности у том елементу једнака: $A = B \Leftrightarrow \mu_A(x) = \mu_B(x), \forall x \in X$.
- **Подскуп** – фази скуп A је подскуп фази скупа B ако и само ако важи да је вредност функције припадности сваког елемента домена за скуп A мања од вредности функције припадности за скуп B, и то за сваки елемент домена: $A \subset B \Leftrightarrow \mu_A(x) < \mu_B(x), \forall x \in X$.

- **Комплемент** – Ако је A фази скуп са функцијом припадности μ_A, тада је A^c комплемент скупа A са функцијом припадности $\mu_{A^c}(x) = 1 - \mu_A(x), \forall x \in X$. Важно је напоменути да следећи идентитети из класичне теорије скупова не важе: $A^C \cap A = \emptyset, A^C \cup A = X$. Закон непротивречности за овако дефинисану операцију није испуњен, а више речи о овоме ће бити у наставку. Све скуповне операције које ћемо дефинисати се могу користити у пракси, а основни разлог за то је ниска рачунарска захтевност. Резултате које производе су најчешће довољно добри.

- **Пресек** – пресек два фази скупа A и B може се дефинисати на више начина, а најчешће се користе следеће дефиниције:

  - минимума: $\mu_{A\cap B}(x) = min\{\mu_A(x), \mu_B(x)\}, \forall x \in X$
  - производа: $\mu_{A\cap B}(x) = \mu_A(x)\mu_B(x), \forall x \in X$, при чему овде треба бити опрезан, већ након неколико узастопних примена функција припадности ће тежити 0.

- **Унија** – унија два фази скупа A и B се може дефинисати на више начина, а најчешће се користе следеће дефиниције:

  - максимума: $\mu_{A\cup B}(x) = max\{\mu_A(x), \mu_B(x)\}, \forall x \in X$
  - суме и пресека: $\mu_{A\cup B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x)\mu_B(x), \forall x \in X$, при чему овде треба бити опрезан, већ након неколико узастопних примена функција припадности ће тежити 0.

Навешћемо и неколико карактеристика фази скупова:

- **Нормалност** – фази скуп је нормалан ако постоји бар један елемент скупа чија је вредност функције припадности 1:

$$\exists x \in A, \mu_A(x) = 1$$

- **Висина** – супремум по функцији припадност:

$$height(A) = sup_x \mu_A(x)$$

- **Подршка** – скуп свих елемената који имају припадност већу од 0:

$$support(A) = \{x \in X | \mu_A(x) > 0\}$$

- **Језгро** – скуп свих елемената који имају припадност 1:

$$core(A) = \{x \in X | \mu_A(x) = 1\}$$

- α**-рез** – скуп свих елемената скупа који имају припадност већу од α:

$$cut(A, \alpha) = \{x \in X | \mu_A(x) > \alpha\}$$

- **Кардиналност** – представља суму свих вредности функције припадности:

$$\begin{aligned} card(A) &= \sum_{x \in X} \mu_A(x) \\ card(A) &= \int_{x \in X} \mu_A(xdx) \end{aligned} \tag{2.1}$$

- **Нормализација** – фази скуп се нормализује тако што се функција припадност подели висином фази скупа:

$$normalized(A) = \mu_A(x)/height(x)$$

2.2 Фази скупови и вероватноћа

Постоји честа забуна између фази скупова и вероватноће. Иако функција припадности може да личи на функцију расподеле вероватноће, ту се свака даља сличност прекида. Вероватноћа се може посматрати као мера вероватноће будућег догађаја на основу нечега што се недавно догодило и што је веома добро познато. Фази функција припадности не говори о сигурности да ће се неки догађај из будућности десити, везује се искључиво за степен истинитости. Степен истинитости се налази, најчешће, у интервалу [0,1]. Приликом пристизања нових информација, сама функција припадности се може мењати тако да боље моделује скуп.

Навешћемо пример да илуструјемо претходно појашњење. *Постоји 85% шансе да ће данас падати киша* – ова реченица нам указује на бројчану вероватноћу да ће данас падати киша. *Данас ће вероватно падати киша* – ова реченица је нејасна, субјективна, и немамо никакав податак о сигурности у њу. Иако се вероватноћа и фази логика баве неизвесношћу, решавају је на различите начине. Фази логика је један начин закључивања над непрецизним и нејасним информацијама.

2.3 Фази логика и резоновање

Дефинишимо спаран дан као дан у коме су температура ваздуха и влажност ваздуха високи. Нека је понедељак (у наставку означен као *пон*) дан за који важи:

$$\mu_{\text{висока температура}}(\text{пон}) = 0.9, \mu_{\text{висока влажност}}(\text{пон}) = 0.8 \tag{2.2}$$

Нека је среда (у наставку означена као *сре*) дан за који важи:

$$\mu_{\text{висока температура}}(\text{сре}) = 0.9, \mu_{\text{висока влажност}}(\text{сре}) = 0.65 \tag{2.3}$$

Дефинисали смо спаран дан, а то је дан у коме је температура висока и влажност висока. Применићемо правило минимума за рачунање пресека и добијамо:

$$\begin{aligned} \mu_{\text{спаран}}(\text{пон}) &= min(0.9, 0.8) = 0.8 \\ \mu_{\text{спаран}}(\text{сре}) &= min(0.9, 0.65) = 0.65 \end{aligned} \tag{2.4}$$

На основу ових вредности, закључујемо да је понедељак спарнији дан од среде. Овај пример је послужио за илустровање начина закључивања на тривијалном нивоу. У реалним околностима, зависности су доста сложеније па имамо скуп правила закључивања.

У фази логици имамо два кључна елемента: ***лингвистичке променљиве*** и ***фази правила закључивања***. Најпре ћемо појаснити појам лингвистичких променљивих.

Лингвистичке променљиве су речи природног језика које описују појам који желимо да моделујемо фази скупом. У примерима: *топао* дан, *влажан* дан, *кишан* дан, речи топао, влажан, кишан су лингвистичке променљиве. Када бисмо моделовали ове лингвистичке променљиве, на пример, за вредности домена бисмо узимали температуру у целзијусима, проценат влажности, број минута колико је киша падала тог дана, редом.

На лингвистичке променљиве можемо да додајемо и квантификаторе. Неки од њих су: *већина, много, ниједан, често, понекад, увек, вероватно, ретко* итд. Такође се користе и квантификатори чија је улога да појачају или ослабе ефекат лингвистичке променљиве: *врло, мало, средње* итд. Модификоване лингвистичке променљиве се могу довести у релацију са основном променљивом путем функције, а обично имају следећу форму: $\mu_{A'} = \mu_A(x)^p, p > 1$.

На примеру температуре, ако имамо лингвистичку променљиву топао дан, можемо увести и променљиву веома топао дан као:

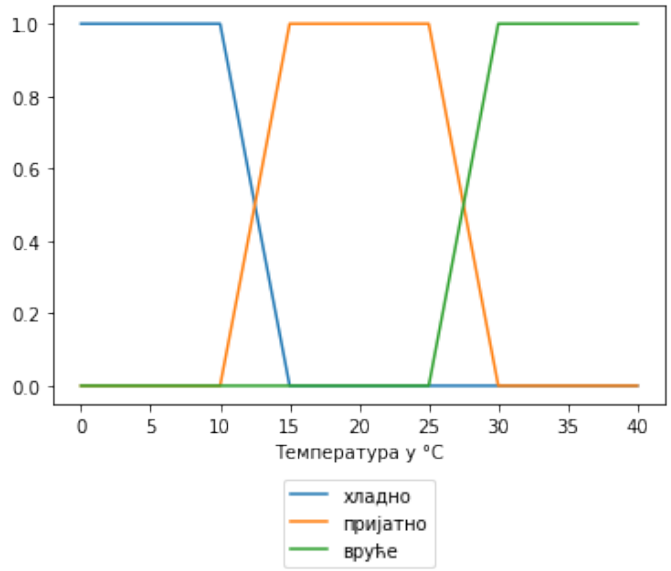
$$\mu_{\text{веома вруће}}(x) = \mu_{\text{вруће}}(x)^2 \tag{2.5}$$

Фази правила закључивања су скуп правила који се прави на основу скупа улазних премиса. Њима моделирамо правила која повезују лингвистичке променљиве, на основу тога како желимо да се наш систем понаша. Ова правила су најчешће у облику ***ако-онда***. На пример: *ако је температура висока и влажност велика, онда је дан спаран.* У *ако* делу правила имамо пресек или унију више улазних фази променљивих, а у *онда* делу се најчешће налази једна излазна фази променљива, али може да се нађе и више њих.

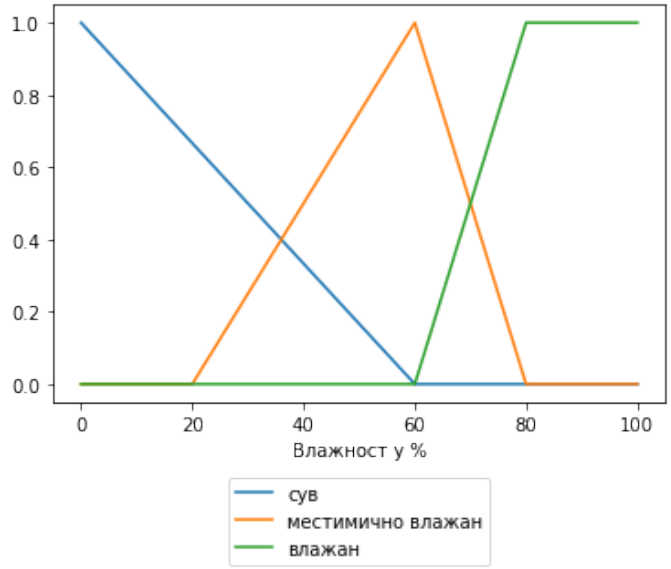
Фази правила заједно са фази скуповима чине базу знања за рад система за одлучивање базираног на фази логици – ***фази контролеру***.

У наставку ће бити речи о комплетном процесу закључивања уз помоћ фази система. До сада смо дефинисали све што је потребно за имплементацију једног типа фази система. Следи и пропратни пример на коме ћемо представити алгоритам, корак по корак.

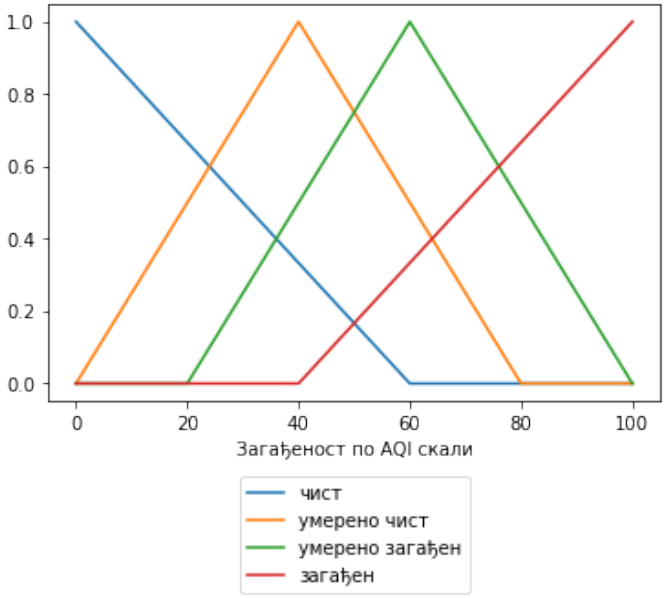
Желимо да моделирамо систем који ће закључивати о томе какав је дан напољу и да ли је погодан за шетњу или спортске активности. Улазни скупови на основу којих ћемо вршити закључивање су температура, влажност и загађење ваздуха. Следи приказ функција припадности за ова три фази скупа (слике 2.3, 2.4 и 2.5). Избор ових функција је слободан избор аутора, пропорционалан нашем животном подручју.



Слика 2.3: Фази скуп креиран на основу информација о температури

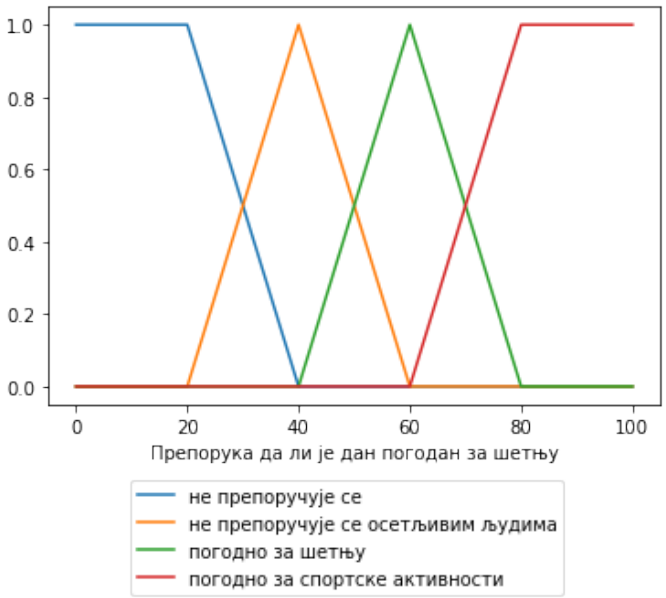


Слика 2.4: Фази скуп креиран на основу информација о влажности ваздуха



Слика 2.5: Фази скуп креиран на основу информација о загађености ваздуха

Сада ћемо навести и излазни скуп (слика 2.6), са предлозима о томе да ли је дан погодан за излазак и које се активности препоручују. Наравно, све функције припадности се могу слободно модификовати у складу са предлозима и препорукама доменских експерата.



Слика 2.6: Излазлни фази скуп - препорука какав је дан

У табели 2.1 се налазе правила закључивања. Сва правила су конјукције

Температура	Влажност	Загађење	Препорука
хладно	сув	чист	погодно за шетњу
хладно	сув	умерено чист	погодно за шетњу
хладно	сув	умерено загађен	не препоручује се осетљивим људима
хладно	сув	загађен	не препоручује се осетљивим људима
хладно	м. влажан	чист	погодно за шетњу
хладно	м. влажан	умерено чист	не препоручује се осетљивим људима
хладно	м. влажан	умерено загађен	не препоручује се осетљивим људима
хладно	м. влажан	загађен	не препоручује се
хладно	влажан	чист	не препоручује се осетљивим људима
хладно	влажан	умерено чист	не препоручује се осетљивим људима
хладно	влажан	умерено загађен	не препоручује се
хладно	влажан	загађен	не препоручује се
пријатно	сув	чист	погодно за спортске активности
пријатно	сув	умерено чист	погодно за шетњу
пријатно	сув	умерено загађен	не препоручује се осетљивим људима
пријатно	сув	загађен	не препоручује се осетљивим људима
пријатно	м. влажан	чист	погодно за спортске активности
пријатно	м. влажан	умерено чист	погодно за шетњу
пријатно	м. влажан	умерено загађен	не препоручује се осетљивим људима
пријатно	м. влажан	загађен	не препоручује се
пријатно	влажан	чист	погодно за шетњу
пријатно	влажан	умерено чист	не препоручује се осетљивим људима
пријатно	влажан	умерено загађен	не препоручује се
пријатно	влажан	загађен	не препоручује се
вруће	сув	чист	погодно за спортске активности
вруће	сув	умерено чист	погодно за спортске активности
вруће	сув	умерено загађен	погодно за шетњу
вруће	сув	загађен	не препоручује се осетљивим људима
вруће	м. влажан	чист	погодно за шетњу
вруће	м. влажан	умерено чист	погодно за шетњу
вруће	м. влажан	умерено загађен	не препоручује се осетљивим људима
вруће	м. влажан	загађен	не препоручује се осетљивим људима
вруће	влажан	чист	погодно за шетњу
вруће	влажан	умерено чист	не препоручује се осетљивим људима
вруће	влажан	умерено загађен	не препоручује се
вруће	влажан	загађен	не препоручује се

Табела 2.1: Табела правила закључивања

услова са леве стране, док су излазне вредности променљиве дате у последњој колони.

Припрема за извршење алгоритма започиње процесом фазификације. Најпре се реченице из улазног простора (природни језик) претварају у фази репрезентацију (фази скуп са одговарајућом функцијом припадности). У нашем

случају су то фази скупови температуре, влажности и загађености као улазни скупови, и дан погодан за излазак и спортске активности као излазни скуп. Затим се дефинишу сва фази правила закључивања, такође превођењем из улазног простора на природном језику у *ако-онда* правила. У примеру је то табела 2.1.

Желимо да пронађемо одговор за следеће улазне податке: Температура је 14°C, влажност ваздуха је 50% и загађеност ваздуха је 30 AQI. Рачунамо функције припадности за сваки фази скуп.

- Температура:

$$\begin{aligned} \mu_{\text{хладно}}(14) &= 0.2 \\ \mu_{\text{пријатно}}(14) &= 0.8 \\ \mu_{\text{вруће}}(14) &= 0 \end{aligned} \tag{2.6}$$

- Влажност:

$$\begin{aligned} \mu_{\text{суво}}(50) &= 0.167 \\ \mu_{\text{местимично влажан}}(50) &= 0.75 \\ \mu_{\text{влажан}}(50) &= 0 \end{aligned} \tag{2.7}$$

- Загађеност:

$$\begin{aligned} \mu_{\text{чист}}(30) &= 0.5 \\ \mu_{\text{умерено чист}}(30) &= 0.75 \\ \mu_{\text{умерено загађен}}(30) &= 0.25 \\ \mu_{\text{загађен}}(30) &= 0 \end{aligned} \tag{2.8}$$

Циљ нам је да применимо правила закључивања над фазификованим улазима. Правила закључивања су у *ако-онда* облику, а ми знамо вредности функције припадности у тачкама за које желимо да пронађемо одговор. Знамо да се у „ако" делу правила налази конјункција, па ћемо користити правило минимума приликом замене вредности у табели правила закључивања. Најпре ћемо у табели да заменимо вредности вредностима функција припадности које смо израчунали (2.2), и израчунати пресек улазних скупова (правило минимума, правила су логичке конјукције).

Следећи корак је рачунање степена припадности сваком од закључака које имамо у табели, а то су заправо вредности излазног фази скупа. Он се рачуна

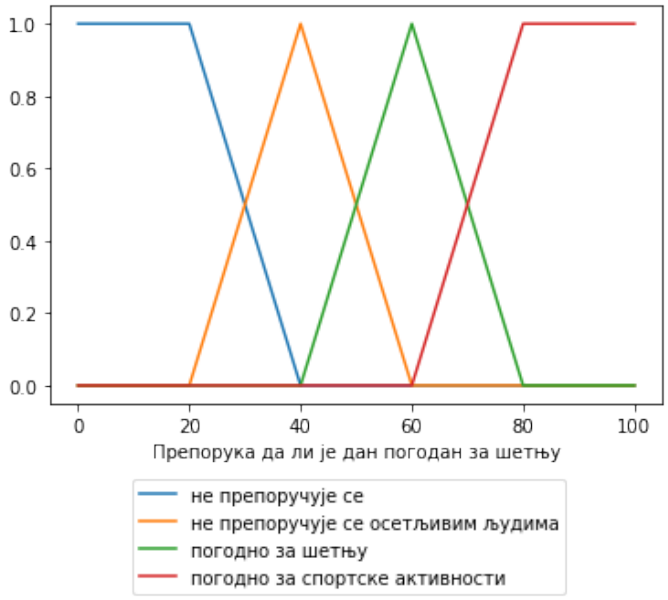
Температура	Влажност	Загађење	Пресек	Препорука
0.2	0.167	0.5	0.167	погодно за шетњу
0.2	0.167	0.75	0.167	погодно за шетњу
0.2	0.167	0.25	0.167	не препоручује се ос људима
0.2	0.167	0	0	не препоручује се ос људима
0.2	0.75	0.5	0.167	погодно за шетњу
0.2	0.75	0.75	0.2	не препоручује се ос људима
0.2	0.75	0.25	0.2	не препоручује се ос људима
0.2	0.75	0	0	не препоручује се
0.2	0	0.5	0	не препоручује се ос људима
0.2	0	0.75	0	не препоручује се ос људима
0.2	0	0.25	0	не препоручује се
0.2	0	0	0	не препоручује се
0.8	0.167	0.5	0.167	погодно за спортске активности
0.8	0.167	0.75	0.167	погодно за шетњу
0.8	0.167	0.25	0.167	не препоручује се ос људима
0.8	0.167	0	0	не препоручује се ос људима
0.8	0.75	0.5	0.5	погодно за спортске активности
0.8	0.75	0.75	0.75	погодно за шетњу
0.8	0.75	0.25	0.25	не препоручује се ос људима
0.8	0.75	0	0	не препоручује се
0.8	0	0.5	0	погодно за шетњу
0.8	0	0.75	0	не препоручује се ос људима
0.8	0	0.25	0	не препоручује се
0.8	0	0	0	не препоручује се
0	0.167	0.5	0	погодно за спортске активности
0	0.167	0.75	0	погодно за спортске активности
0	0.167	0.25	0	погодно за шетњу
0	0.167	0	0	не препоручује се ос људима
0	0.75	0.5	0	погодно за шетњу
0	0.75	0.75	0	погодно за шетњу
0	0.75	0.25	0	не препоручује се ос људима
0	0.75	0	0	не препоручује се ос људима
0	0	0.5	0	погодно за шетњу
0	0	0.75	0	не препоручује се ос људима
0	0	0.25	0	не препоручује се
0	0	0	0	не препоручује се

Табела 2.2: Табела са замењеним вредностима и израчунатим минимумом

по следећој формули: $\beta_i = max(\alpha_{ki})$, за свако правило у којем фигурише закључак са индексом i. То значи да за сваку вредност излазне променљиве, проналазимо сва правила која одговарају у табели 2.2 и од тих вредности узимамо максималну вредност. Добијамо следеће вредности:

$$\begin{aligned} \mu_{\text{не препоручује се}} &= 0 \\ \mu_{\text{не препоручује се осетљивим људима}} &= 0.25 \\ \mu_{\text{погодно за шетњу}} &= 0.75 \\ \mu_{\text{погодно за спортске активности}} &= 0.5 \end{aligned} \tag{2.9}$$

Последњи корак је превођење фази закључака у решење које је на природном језику. Желимо да одредимо лингвистичке променљиве за претходно одређене припадности закључцима. Имамо излазни фази скуп (слика 2.7) и питање је како одредити лингвистичку променљиву за ове вредности. Вредности на x-координанти на овом графику представља оцену дана у интервалу [0,100]. Можемо рећи да што је број већи да је дан лепши и пријатнији. Оваква скала је избор аутора, и додатно осликава широликост примене фази логике на непрецизне и некомплетне информације.



Слика 2.7: Излазни фази скуп - препорука какав је дан

Постоји више метода како можемо доћи до излазне бројчане вредности, већина начина је помоћу коришћења центроида. Центроид је центар масе про-

извољне дводимензионе фигуре која има униформну густину. Неформално, то је тачка у којој би дводимензиона фигура била у тежишту. На примеру троугла, центроид се поклапа са тачком тежишта троугла.

Рачунање центроида ћемо објаснити помоћу методе декомпозиције сложене фигуре на простије. Просте геометријске фигуре које се користе су произвољни троугли и паралелограми. За троугао знамо да израчунамо центроид (тј. тежиште), а за паралелограме ћемо узети пресечну тачку дијагонала. Овде смо обухватили мало више простијих фигура него што се у већини фази функција припадности налази, у применама је то најчешће правоугли троугао, квадрат и правоугаоник. Ако сложену фигуру разложимо на k мањих и простих фигура, имаћемо k центроида, и означавамо их са C_i. Такође, за сваку од k фигура знамо да израчунамо и површину P_i. Формула за центроид C сложене фигуре се рачуна по следећим формулама за x и y:

$$C_x = \frac{\sum_i C_{i_x} P_i}{\sum_i P_i} \quad C_y = \frac{\sum_i C_{i_y} P_i}{\sum_i P_i} \tag{2.10}$$

Следе неки начин рачунања излазне вредности помоћу центроиде:

- **Макс-мин** – Вредност излазног фази скупа са највећом вредношћу из табеле је изабрано, бирамо површину функције припадности те вредности за даље рачунање. Рачунамо центроид те површине и x-координата центроида се узима као резултат рада фази контролера.

- **Упросечавање** – Рачунамо средњу вредност свих фази вредности излазног скупа из табеле. Повлачимо хоризонталну праву кроз израчунату вредност, и површина испод те праве је површина над којом ћемо рачунати центроиду. Центроида се рачуна, и x-координата центроида се узима као резултат рада фази контролера

- **Скалирање** – функције припадности свих вредности излазног фази скупа се скалирају тако да максимална вредност сваке од тих вредности она вредност која је добијена у табели. Тиме ограничавамо моћ утицаја сваког правила да одлучује онолико колико добро описује излазну вредност. За ту површину рачунамо центроид, и настављамо као и у претходним случајевима.

- **Исецање** – функције припадности сваке вредности излазног фази скупа се секу хоризонталном правом тако да им вредности не могу премаши-

ти вредност добијену из табеле. И у овом случају ограничавамо моћ утицаја. Рачунање центроида се врши на површину добијену исецањем почетних функција припадности, и настављамо као у претходним случајевима.

Методе засноване на рачунању центроида су једне од најчешће коришћених. Поред њих постоји још доста различитих метода дефазификације. Неки од њих су: рачунање центара сума, проналажење праве која врши бисекцију површине, прва, средња или последња максимална вредност. Наравно, сваки од ових метода ће дати различит резултат.

Ми ћемо овде приказати метод пондерисаног просека, из разлога лакшег рачунања вредности. Користе се на симетричним функцијама припадности. У просеку, даје јако сличне резултате као методе са центроидом, осим што су доста мање рачунарски захтевне. Вредности x_i су вредности добијене из табеле за сваку вредност излазног фази скупа, наведене у 2.9. Вредности $\mu_C(x_i)$ x-координате оса симетрије датих функција припадности. Следи формула за рачунање вредности:

$$\begin{aligned} output &= \frac{\sum_{i=1}^{n_x} x_i \mu_C(x_i)}{\sum_{i=1}^{n_x} \mu_C(x_i)} \\ &= \frac{0 * 20 + 0.25 * 40 + 0.75 * 60 + 0.5 * 80}{0 + 0.25 + 0.75 * 0.5} \\ &= \frac{91.7}{1.5} = 63.3 \end{aligned} \tag{2.11}$$

2.4 Фази контролери

Дизајн фази контролера представља један од највећих изазова фази логике. Желимо да покушамо да управљамо реченицама, јер су реченице ближе људима од математичких једначина и формула. Може се рећи да је фази логика спрега између природног језика и рачунара који треба да управља неким процесом. Први фази контролер су направили *Mamdani* и *Assilian* 1975. године [3], а сврха му је била управљање парним мотором. Циљ је био да се одржи константна брзина помоћу контроле притиска на цилиндре, а то се регулисало количином топлоте која је довођена до бојлера. Данас је примена фази контролора вишеструка.

Фази контролер се може посматрати као нелинеарна функција која пресликава неке улазе у неке излазе. Циљ контролера је да за сваку улазну вредност произведе излазну вредност која најбоље апроксимира жељено понашање. Жељено понашање дефинише доменски експерт дефинисањем фази правила и фази функција припадности. Фази контролер се састоји од четири главне компоненте:

1. **Скуп фази правила**: представља базу знања фази контролера. Уноси га доменски експерт. Ова правила уносе нелинеарност у доношење одлука шта ће бити излаз из контролера.

2. **Фазификатор**: задужен је за трансформацију улаза у функције припадности. Такође трансформише и излаз у функцију припадности. Коришћење нелинеарних функција уноси нелинеарност у контролер.

3. **Механизам закључивања**: врши закључивања по унапред одређеним правилима и уноси податке из фазификатора у базу знања – скуп фази правила. Након одређивања припадности сваке променљиве излазног фази скупа, долазимо до дефазификације. Нелинеарност овде уводи коришћење минимума и максимума за пресек и унију у правилима.

4. **Дефазификатор**: задужен за конвертовање фази закључка у бројчану вредност излазног фази скупа која ће се даље користити. У практичним примена те вредности се углавном директно преносе уређајима на извршавање. На пример ако је излазна вредност температура од 35°C, та вредност ће бити прослеђена грејачу да постави термостат на ту вредност. Постоје разни механизми за дефазификацију.

Постоји више типова фази контролера, већина имају врло сличан дизајн, а најчешће се разликују у неким компонентама попут механизма закључивања и дефазификатора. Као што смо навели, експерт из домена примене мора да дефинише следеће: домен вредности на којима ће се вршити фази закључивање, дефинисање фази скупова, функција припадности, фази правила, коју методу дефазификације користити, који механизам закључивања користити.

Поред ових проблема, транспорт и обрада података који дођу до контролера са разних сензора могу да представљају изазов. Обрада података најчешће

подразумева отклањања шума, дискретизацију реалних вредности, скалирање, обраду екстремних вредности итд. Наведени проблеми представљају изазов приликом конструисања фази контролера, као и потенцијални простор за увођење грешке која може да доведе до контрапродуктивног резултата.

Навешћемо три најчешћа типа контролера:

1. Контролер заснован на табели
2. Мамданијев фази контролер (*Mamdani*)
3. Такаги-Сугено фази контролер (*Takagi-Sugeno*)

Контролер заснован на табелама се користи у дискретном домену где имамо коначан скуп улаза и где је изводљиво направити табелу која ће за дате вредности моћи врло брзо да креира излаз. То се дешава простим прегледом табеле за дате улазе. Овај тип контролера је веома неефикасан за већи број улаза и излаза.

Мамданијев контролер [3] је контролер чији смо пример дали у прошлом делу. Састоји се од следећих корака:

1. Препознати и идентификовати улазне лингвистичке променљиве и њихове домене вредности.
2. Препознати и идентификовати излазне лингвистичке променљиве и њихове домене вредности.
3. Дефинисати скуп функција припадности за све улазне и излазне променљиве.
4. Конструисати скуп правила.
5. Извршити фазификацију улазних вредности.
6. Извршити механизам закључивања ради одређивања правила које учествују у дефазификацији као и јачину њиховог утицаја.
7. Извршити дефазификацију ради одређивања акције која ће бити предузета.

Код Мамданијевог контролера и контролера заснованог на табелама излазни скупови су једнозначно одређени. Излазни скупови могу бити и линеарне

комбинације улаза. *Takagi* и *Sugeno* [17] су представили овакав тип контролера. У овом типу контролера, правила су следећег облика:

$$if\ (A_1\ is\ a_1, A_2\ is\ a_2, ..., A_n\ is\ a_n)\ then\ C\ is\ f(a_1, a_2, ..., a_n) \tag{2.12}$$

Увиђамо да у овим правилима са леве стране може да се нађе произвољан број улазних фази променљивих, за разлику од два која смо користили до сад. Такође, са десне стране правила немамо само једну излазну фази променљиву већ неку функцију која зависи од улазних фази променљивих. f је функција која уноси додатну нелинеарност у правила одлучивања. Скуп C је скуп излазних променљивих, скупови A_i су улазни скупови са функцијом припадности μ_{A_i}. База знања је формирана од n_k правила.

Ради поређења, навешћемо поново и облик правила код Мамданијевог контролера:

$$if\ (A\ is\ a,\ B\ is\ b)\ then\ (C\ is\ c) \tag{2.13}$$

Излазне вредности се такође рачунају преко пресека, помоћу мин-оператора:

$$\alpha_k = \min_{\forall i | a_i \in A_k} \{\mu_{A_i}(a_i)\} \tag{2.14}$$

Код Мамандијевог контролера, целокупан систем је описан статичким правилима. За Такаги-Сугенов контролер важи да је последица правила математичка функција, која може да уведе више динамичности приликом одлучивања, и потенцијално може дати боље резултате. Овај прекид статичности код Мамандијевог контролера је уједно и главна предност Такаги-Сугеновог контролера. Два проблема која се отварају су потреба за креирањем функција, као и већа потреба за рачунарским израчунавањем.

Глава 3

Фази системи у агрикултури

Пластеници су структуре за контролисани узгој биљака. Углавном се користе за производњу биљака за људску употребу на индустријском нивоу, али могу бити и мањих димензија у случају производње за домаћинство. Последњих деценија, услед недостатка хране у свету, пластеници постају све важнији. Према истраживањима, пластеници могу да имају 15 пута веће приносе у односу на узгој на њивама. Такође, приликом узгајања воћа, у пластеницима је око 90% воћа спремно за тржишну продају, док ван пластеника то износи од 40% до 60% [6].

Пластеници имају своју микро-климу: температура, влажност ваздуха, влажност земљишта су неки од параметара микро-климе који се могу контролисати. Поред микро-климе, постоји још параметара који се могу контролисати: наводњавање, ђубрење, додатно осветљење или засена. Као предности узгоја на овакав начин, можемо навести следеће:

- Услед контролисане температуре и влажности, можемо узгајати биљке са различитих подручја.
- Због контроле наводњавања и ђубрења, уштеда воде може бити вишеструка. Такође се минимизује шанса за исушивање или труљења биљака услед премало или превише воде, као и топљења биљака услед превелике концентрације ђубрива.
- Контрола корова, штеточина и болести је доста лакша него у случају класичног узгоја на њивама.

- Опционо, биљке могу бити заштићене од града, кише, снега или неких других временских непогода.

3.1 Историјат

Идеја о узгоју биљака у контролисаним условима датира још од периода античког Рима. Велика Британија и Холандија су прве земље у Европи које су почеле да експериментишу са контролисаним узгојем биљака у XVII веку. Потреба за пластеницима у Европи је порасла са увозом биљака из Северне Америке и Азије у том периоду. Први грејани пластеник је направљен у Челсију 1673. године. У њему су се узгајале биљке које су биле коришћене у медицинске сврхе. Француски ботаничар, *Charles Lucien Bonaparte* је заслужан за прављење првог модерног пластеника у Лајдену, Холандија.

Пластеници доживају процват популарности у XIX веку у Великој Британији, када је скоро свака богатија кућа, парк или јавна површина имала пластенике у коме су чували декоративне биљке. Највећи пластеници тог времена су се налазили у Великој Британији, и у многима су се гајиле палме. Напретци у процесу производње стакла и употреба челика приликом изградње пластеника су допринели повећању броја пластеника и убртаном развоју. Из овог периода, навешћемо неке од најпознатијих пластеника међу којима неки постоје и данас: кућа палми у Лондону, ботаничка башта у Белфасту, Краљевски пластеници у Лаекену, Гласпласт у Минхену.

Са почетком XX века пластеници добијају и нове облике. До тада су прављени правоугаони пластеници, а новији су сферног и пирамидалног облика. Климатрон у ботаничкој башти у Мисурију је рани пример сферног пластеника. Неколико пластеника у Мутарту конзерваторијуму у Канади су пирамидалног облика. Све чешће у изградњи се користе новији материјали: алуминијум као носећи конструктивни материјал и полиетиленске фолије за покривање пластеника.

Тренутно највећи пластеник на свету се налази у Сингапуру, и покрива површину од $12.800m^2$. Саграђен је 2012. године. [1]

[1] https://www.guinnessworldrecords.com/world-records/113569-largest-glass-greenhouse

3.2 Принцип рада

Стакленик је структура где су носиви елементи најчешће направљени од алуминијума или челика, а зидови и кров пластеника су од стакла, полиетилена или неког другог провидног материјала. Сунчеви зраци пролазе кроз провидне површине, а због затворености пластеника не долази до веће размене гасова и температура унутар пласеника расте. Показује се да је *температура* један од најбитнијих фактора приликом узгоја биљака.

Ако бисмо имали идеално затворен пластеник, у њему би температура расла на нивое који не одговарају већини биљака, тј. дошло би до исушивања неких биљака. Пластеници нису идеално задихтовани, и долази до протока ваздуха, односно вентилације. *Вентилација* је такође једна од неопходних компонената за вођење пластеника. Сврха вентилације је у регулисању температуре и влажности ваздуха, као и у протоку ваздуха ради спречавања развоја патогена који се развијају у устајалом ваздуху. Такође, вентилацијом се допрема свеж ваздух у коме се налази и угљен-диоксид који је неопходан за фотосинтезу. Вентилација се врши помоћу отварања прозора на пластеницима или уз помоћ вентилатора.

Топлота која се ствара сунчевим грејањем пластеника често није довољна. У том случају се користи додатно грејање на различите погоне. Поред фосилних горива, користе се обновљиви извори енергије: соларни панели, геотермално грејање. Данас је популарно грејање настало као продукт рада неке друге индустрије. Пластеници се постављају поред фабрика или постројења које производе вишак топлоте која им није потребна, па се она користи за загревање пластеника.

У земљама где током одређених делова године дан траје кратко, уводи се и *додатно осветљење*. Насупрот томе, у случају потребе за смањењем светла, користе се полу-прозирне фолије које се могу употребити по потреби.

Према *Либиговом закону минимума* (*Liebig's law of minimum*) [4], раст није одређен укупном количином ресурса које имамо, него је условљен најоскуднијим ресурсом. У пластеницима то је најчешће *угљен-диоксид*. Постоји више начина за додавање угљен-диоксида у пластеник, један од најчешћих је коришћење гаса као нус-продукта неког другог индустријског процеса, на пример у производњи шећера.

3.3 Фази логика и агрикултура

У последњим годинама паметни системи се све више користе у разним процесима производње и контроле. Агрикултура је пример у коме се врло често користе подаци из природе који долазе са степеном несигурности и недоречитости. Фази логика се примењује у пољопривредним и биолошким системима да бисмо добили прецизније и поузданије резултате. Поред фази логике, и остале области вештачке интелигенције се често користе у агрикултури.

Marmin, Mushthofa (2013) [13] су анализирали постојеће радове који су се бавили проналажењем адекватне пољопривредне површине, предвиђањем временских прилика, коришћењем пестицида и процене квалитета усева. У раду су приказали да је коришћење компјутерског вида (поље вештачке интелигенције) заједно са фази логиком донело добре резултате у решавању постојећих проблема у агрикултури.

T. P. Roseline, N. Ganesan, C. Tauro (2015) [18] су се бавили применама фази логике у унапређивању постојећих пољопривредних процеса. Контролисање корова, штеточина и болести, процене земљишта су неке од тема које су анализирали и примењивали заједно са традиционалним методама.

L. Sakharova, M. Strukov, I. Akrepov, T. Alekseychik, A. Chuvenkov (2017) [11] су применили фази логику на процену количине усева користећи агро-метеоролошке моделе. Овај приступ је направио алат који може да рангира површине за пољопривреду на основу временских услова за ту територију као и могућности за узгој одређене врсте.

L. Li, K.W.E. Cheng, J.F. Pan (2017) [10] су дизајнирали паметни систем за контролу пластеника користећи PLC контролере. Параметри које су користили су температура, влажност ваздуха и количина светла.

D. K. Lim, B. H. Ahn, J.H. Jeong (2018) [2] су креирали фази систем за контролисање влажности ваздуха и температуре ради оптимизовања енергетске ефикасности система за вентилацију.

I. Mat, M. R. M Kassim, A. N. Harun, I.M. Yusoff (2016) [8] су користили бежичну мрежу сензора влажности да направе ефикасан систем за наводњавање. Направили су посебну методу за примену у пољопривреди чији су тестови показали уштеду од 1500ml воде по дрвету дневно.

Поред наведених радова постоји још доста радова са темом примене фази

логике у агрикултури и пластеницима. Већина примена које се односе на гајење биљака у било каквим условима се такође могу прилагодити и користити условима у пластеницима.

Глава 4

Систем за аутоматизацију пластеника

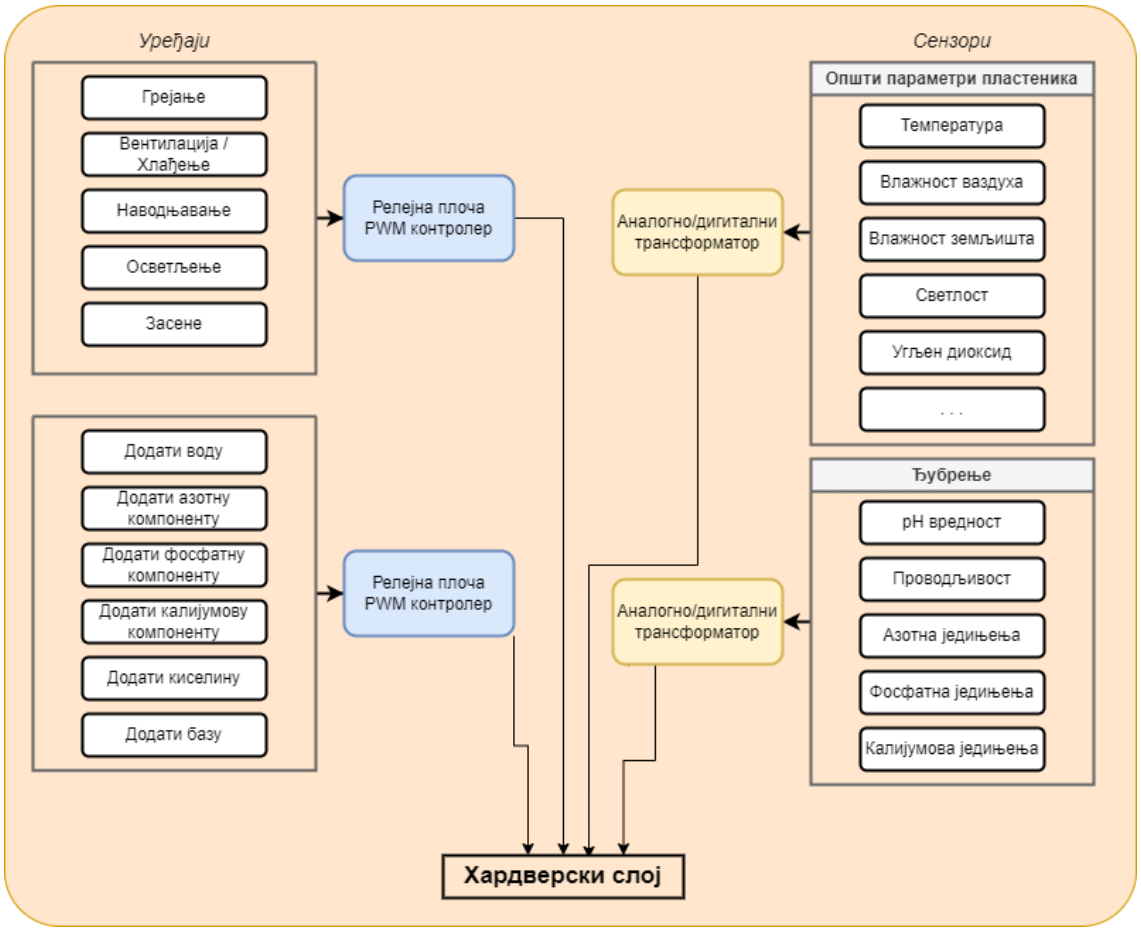
4.1 Архитектура

Архитектуру система који ће бити приказан у овом раду ћемо започети са хардверским слојем (слика 4.1).

Параметре које желимо да аутоматизујемо можемо да групишемо у скупове, које ћемо називати ***подсистеми***. Број тих подсистема може бити произвољан, у примеру ћемо аутоматизовати *опште параметре пластеника*. Сви подсистеми су одвојене целине, и могу се независно дефинисати и посматрати. Поред општих параметара пластеника, можемо аутоматизовати количинске односе различитих ђубрива и коришћење пестицида у пластеницима.

У општим параметрима пластеника се као улазни подаци налазе сензори који мере температуру, влажност ваздуха, влажност земљишта, количину светлости и количину угљен-диоксида. Сензори за ове параметре могу бити аналогни или дигитални. Аналогни су најчешће имплементирани као варијабилни отпорници - отпор на сензору ће се линеарно мењати у зависности од вредности коју је сензор измерио. Вредности се фабрички калибришу у неколико тачака, а остале вредности се добијају одговарајућом апроксимацијом. Аналогни сигнал се мора пре даље употребе претворити у дигитални, за шта се користе аналогно-дигитални конвертери. Дигитални сензори на себи већ имају електронику која је задужена за трансформисање аналогног напона у дигитални сигнал, односно низ нула и јединица које представљају вредност.

У реалној примени се јавља потреба за преносом сигнала од сензора до



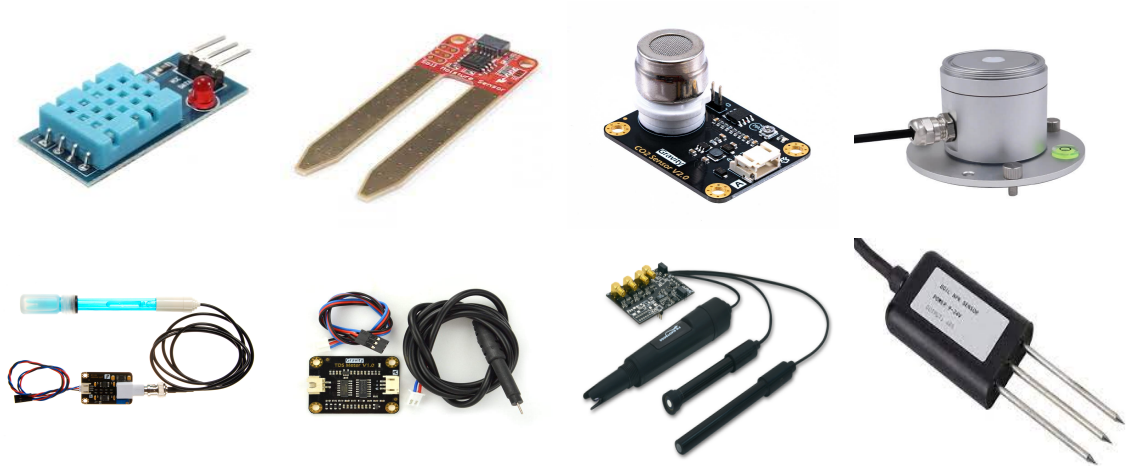
Слика 4.1: Хардверски слој

уређаја који ће обрађивати те информације. За пренос података се користе стандардизовани протоколи за трансфер података, неки од најчешће коришћених су I^2C, SPI или $UART$. Прва два су заснована на главни-подређени (*master-slave*) архитектури са часовником, док је $UART$ протокол за серијску комуникацију два уређаја и не користи часовнике за синхронизацију.

Сваком скупу улазних параметара одговара и један скуп излазних уређаја. Излазни уређаји у случају општих улазних параметара су: грејање, вентилација, наводнавање, осветљење и засена. Наравно овај скуп као и скуп улазних параметара се могу произвољно мењати. Сви ови уређаји могу бити коришћени на два начина: коришћени преко прекидача (*On/Off*) или регулисани. Уређаји који раде преко прекидача се у електроници најчешће користе преко релеја који испуњава улогу прекидача. Регулисани уређаји се регулишу пулсном модулацијом или одсецањем фреквенце. Први приступ се често користи

у електроници и *IoT* применама, а своди се на то да уређај укључимо па искључимо у веома кратком временском интервалу. Што је тај интервал краћи, добијамо ефекат непрекидности у опсегу регулисања уређаја. Оваква регулација се користи у једносмерним колима. Одсецање фреквенце (одсецање синусоиде) се примењује ако контролишемо уређаје који раде на наизменичном напону. Доста су компликованији за имплементацију од регулатора који раде помоћу пулсне модулације.

Оно што је важно напоменути је да сваки скуп улазних параметара треба да има тачно један скуп излазних уређаја. Овај пар можемо назвати подсистем. Један или више подсистема чине хардверски слој. На слици 4.1 први подсистем је задужен за микроклиму, а други за ђубриво.



Слика 4.2: Примери сензора који се могу користити у агрикултури

На слици 4.2 су приказани неки примери сензора који се могу користити. Са лева на десно, први па други ред, налазе се:

- DHT11 сензор температуре и влажности ваздуха.
- Сензор влажности земљишта.
- Сензор количине угљен-диоксида.
- Сензор количине фотосинтентичког зрачења.
- pH сензор.
- Мерач проводљивости раствора за ђубрење.

- Сензор азотних, фосфатних и калијумових једињења у течности.
- Сензор азотних, фосфатних и калијумових једињења у земљишту.

Сада ћемо размотрити централни део система. Он се састоји из два дела: рачунара за прихватање сигнала, извршавање алгоритма и издавање команди излазним уређајима, и другог рачунара који је задужен за интеракцију корисника са пластеником: задавање фази скупова, фази правила и добијање општих информација о систему.

Први рачунар је у облику микрорачунара базираног на једној плочи (*single−board*). На једној плочи се налази процесорска јединица, радна меморија, меморија за складиштење, хардверски пинови који се користе за повезивање са другим уређајима и сензорима. Неки од њих имају могућности за инсталацију оперативних система. Ови рачунари се још називају и **уградни рачунари**. Уградни рачунари на које је могућа инсталација оперативних система се по могућностима и способностима могу упоредити са персоналним рачунарима, међутим њихов хардвер је често слабији, тј. процесори имају мање језгара, радна меморија је такође врло често вишеструко мања. Ми ћемо у примеру користити *Raspberry Pi 3 B+*. Овај рачунар има четворојезгарни 64-битни процесор на *ARM* архитектури и 1GB радне меморије. На њему се налази 40-пински конектор преко кога можемо да повежемо додатне уређаје и сензоре. Ови пинови се називају и *улазно-излазни пинови опште намене*. Могу се софтверски програмирати да ли ће бити улазни или излазни, неки од њих чак имају и хардверску подршку за пулсно управљање, а неки имају аналогно-дигиталне конвертере који се могу активирати по потреби. За повезивање на интернет, овај рачунар може да користи жични *ethernet* или бежични *Wi-Fi* протокол. Уз додатни модул, могуће је повезати се на интернет и преко базних станица мобилне телефоније или сателита. Напајање рачунара је 5V једносмерне струје, а неопходна јачина струје да бисмо имали стабилан систем износи барем 2.5A. Због ових једноставних услова, *Raspberry Pi* се може повезати и на акумулаторске изворе струје, или чак на соларне панеле. У пластеницима је ово врло лако имплементирати, и тиме се ослобађа потреба за жичним напајањима.

Други рачунар је потребан за интеракцију корисника са пластеником. Унос фази скупова, правила као и провера општег стања у пластенику се

врши преко овог система. На овом рачунару се налази база података *MySQL*. База може бити и било ког другог произвођача. Такође, не мора да буде ни релациона, може да буде и база заснована на документима (*NoSQL*). Осим базе, налази се и веб апликација коју корисници користе за интеракцију. Веб апликација је урађена уз помоћ радног оквира *ASP.net Core*.

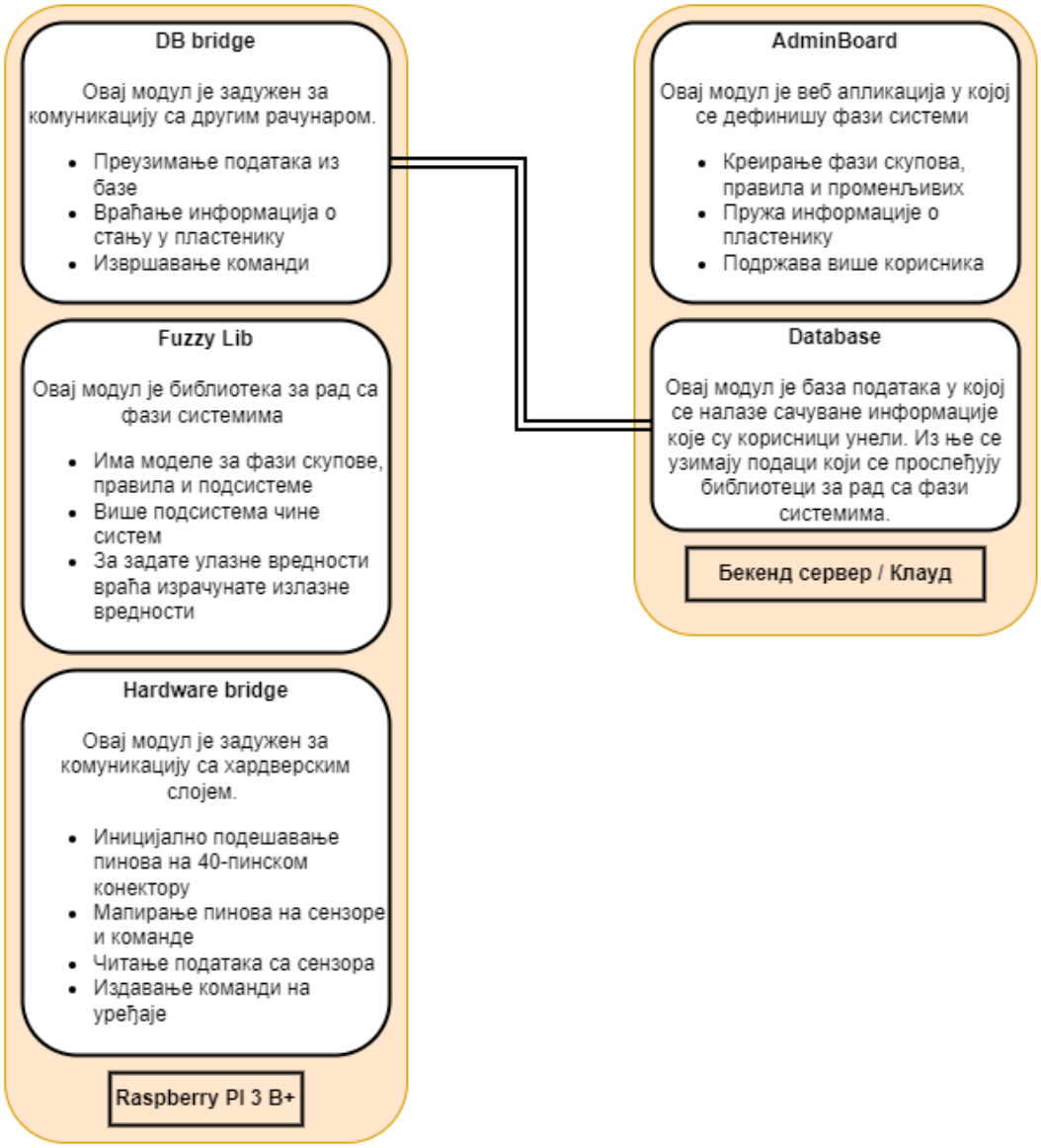
Тренутно је у свету све популарније рачунарство у облаку. Платформа, инфраструктура и софтвер се могу користити као услуге, у мерама које обим корисника захтева. Лака скалабилност је још једна од предности рачунарства у облаку. Веб апликација се може држати на облаку, али и на било којем другом типу рачунара. У нашем примеру ће се извршавати на кућном рачунару са *Intel64* архитектуром.

Ова два рачунара се могу и „објединити" у један, тј. да се све потребне апликације извршавају на једном рачунару - *Raspberry Pi*. Међутим овај приступ има више мана. Перфомансе хардвера уређаја *Raspberry Pi* се могу мерити са данашњим мобилним телефонима. Иако се *.Net* и *ASP.net Core* могу користити на оперативном систему *Linux*, покретање обе апликације доводи до драстичног успорења рада. Основна намена *Raspberry Pi* уређаја је едукација деце о рачунарским наукама у школама. Међутим приступачност уређаја, као и његове способности за *IoT* пројекте су га учиниле најпопуларнијим микро рачунаром на свету. У реалним применама, користили би се робуснији уређаји са потенцијално већим перфомансама. Међутим сама архитектура тих рачунара је иста као *Raspberry Pi*, тако да се развој и тестирање целокупног система може врло успешно извршити на овом уређају. Схематски приказ задатака оба рачунара и место размене података између њих се налази на слици 4.3.

Целокупна архитектура је замишљена као модуларна, тако да свака апликација представља независни модул. Разлози који доприносе модуларности су наведени у претходним деловима, па ћемо сад само поновити:

- Рачунар задужен за управљање може бити било који уређај који подржава радни оквир *.Net Core*. Треба да постоји начин за повезивање уређаја на њега, било непосредно путем универзалних улазно-излазних пинова или посредно путем трећег уређаја.

- Сензори и уређаји могу бити било како распоређени и повезани на различите пинове. Такође немамо ограничења у броју сензора и уређаја



Слика 4.3: Веза између два рачунара: *Raspberry Pi* и рачунара на коме се налази веб апликација и база

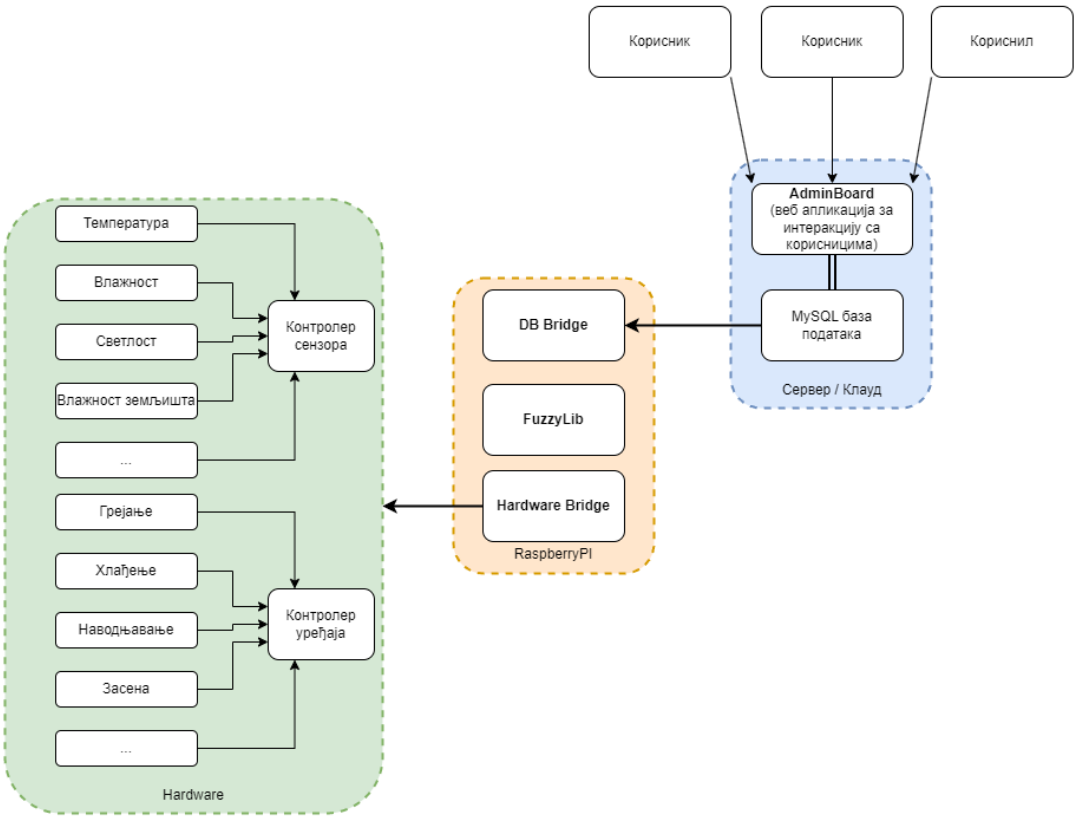
које повезујемо. Са софтверске стране гледишта, можемо дефинисати правила и скупове за произвољан број сензора и уређаја. Са хардверске стране, не можемо повезати произвољан број уређаја и сензора зато што немамо произвољно много пинова. Потребно је да постоји и софтверско пресликавање који ће се пин користити за коју сврху.

- За избор рачунара на коме ће се извршавати веб апликација и који ће

бити домаћин за базу података, може се изабрати кућни рачунар, серверски рачунар или рачунарство у облаку. Неопходно је да овај рачунар подржава *.Net Core* радни оквир.

- База података може бити и релациона и база заснована на табелама. У имплементацији је коришћен радни оквир за објектно релационо пресликавање - *Entity Framework Core*. Ми смо изабрали базу података *MySQL*, а овај радни оквир подржава пресликавање у релационе базе *Microsoft SQL Server, SQLite, PostgreSQL* као и нерелациону базу *CosmosDb*. Неке друге мање коришћене базе података су написале и своје адаптере за коришћење са овим ОРМ-ом (објектно-релациони мапер тј. радни оквир за објектно релационо пресликавање).

На слици 4.4 је приказана схема комплетне архитектуре система.



Слика 4.4: Комплетан приказ архитектуре

4.2 Дефиниција скупова и правила на нашем примеру

У овом одељку ћемо навести подешавања која ћемо користити на нашем систему. Модел и вредности за све функције припадности су креиране на основу постојеће литературе, а оне су базиране на информацијама добијеним од доменских експерата. Креираћемо један модул за одржавање микроклиме у пластенику.

Модул за микроклиму користи пет улазних информација, а контролише шест уређаја. Улазни сензори су: температура ваздуха, влажност ваздуха, влажност земљишта, количина светлости, количина угљен-диоксида. Излазни уређаји су: грејач, вентилација, наводњавање, додатно осветљење, ролетне за засену, додавање угљен-диоксида. За излазне уређаје су креиране функције припадности приказане на слици 4.5.

Свака функција припадности за излазне уређаје је зависна од једне или две улазне функције припадности. Те везе су следеће:

- Грејање: зависи од температуре и влажности ваздуха.
- Вентилација: зависи од температуре и влажности ваздуха.
- Наводњавање: зависи од влажности земљишта и влажности ваздуха.
- Ролетне за засену: зависи од температуре и количине светлости.
- Додатно осветљење: зависи од количине светлости.
- Додавање угљен-диоксида: зависи од количине угљен-диоксида и количине светлости.

На слици 4.6 се налазе графикони функција припадности улазних параметара. Навешћемо и детаљан опис сваког од њих:

- Температура ваздуха: температура која се налази унутар пластеника, и означена је у степенима целзијуса. Улазни сензор за овај параметар може да буде дигитални *DHT11* чију смо слику приказали на почетку главе. Овај сензор поред температуре може да мери и влажност ваздуха.

- Влажност ваздуха: мерена у пластенику, а означена је у процентима. Проценат је однос грама водене паре у јединици запремине ваздуха и максималне количине водене паре у јединици запремине ваздуха на тој температури. На свакој температури постоји различита вредност максималне количине водене паре у ваздуху, и та максимална количина се назива тачка потпуне засићености. Што је већа температура, већа је и тачка потпуне засићености. Већина сензора за мерење влажности мере овај тип влажности ваздуха - релативну влажност.

- Влажност земљишта: представља однос количине воде и количине чврсте материје у земљишту. Такође се означава у процентима. Различити типови земљишта су покривени истим односом. Један пример сензора је приказан у прошлој глави, а већина њих су дигитални сензори који раде са десетобитним подацима, чиме имамо 1024 различите вредности за апроксимацију интервала [0,100].

- Количина светлости: мерена у пластенику. Додатни фактор који треба имати у виду је да се део светлости одбија од стаклене или пластичне зидове и кров пластеника, и тиме се подиже температура у пластенику. Мери се у луксима, односно луменима по метру квадратном.

- CO_2: мерена у пластенику, мери се дигиталним сензорима, а мера која се најчешће користи је *ppm (parts per million)*. Иако ниједна бездимензиона величина није призната од стране Међународног система јединица, приликом мерења гасова ова јединица се врло често у пракси користи. Адекватна мера у нашем случају би била и количина CO_2 у милиграмима по литри ваздуха.

Навели смо улазне и излазне функције припадности, и описали смо какве зависности између њих постоје. На основу њих, направљена је следећа табела правила закључивања, приказана у табели 4.1.

Преостало је још да дефинишемо дефазификатор, односно да преведемо фази излаз у вредност коју ћемо послати контролеру уређаја. На основу те вредности, он ће управљати излазним уређајима. Користићемо методу пондерисаног просека, задату формулом 2.11. Навели смо да даје резултате сличне методима рачунања центроиде, али је доста једноставнија за израчунавање.

4.3 FuzzyLib - библиотека за рад са фази системима

За потребе овог мастер рада, најпре је направљена *C#* библиотека за рад са фази системима. Она се састоји од следећих класа:

- *FuzzyInput* - класа која представља једну функцију припадности. Она је описана именом, идентификатором, листама X и Y координата тачака које описују ту функцију припадности. Најважнији метод ове класе је *CalculateFunctionValue* који прима параметар X - вредност за коју желимо да израчунамо вредност функције припадности. Сложеност ове операције је *O(n)*, где је *n* број тачака које описују функцију припадности. За рачунање је потребно да итерирамо кроз низ тих тачака и пронађемо подинтервал у коме се параметар X налази. Затим формулом за рачунање праве кроз две тачке израчунавамо вредност. Подржане су троугаона, трапезоидна и линеарна функција припадности. Следи кôд ове методе:

```
1  public float CalculateFunctionValue(float x)
2  {
3      if (x < Points[0].X)
4          return Points[0].Y;
5      if (x > Points[^1].X)
6          return Points[^1].Y;
7      for(int i = 0; i < Points.Count-1; ++i)
8      {
9          float x1 = Points[i].X;
10         float x2 = Points[i+1].X;
11
12         if ((x1 <= x) && (x <= x2))
13         {
14             float y1 = Points[i].Y;
15             float y2 = Points[i + 1].Y;
16
17             if (y1 == y2)
18                 return y2;
19             if (y1 < y2)
20                 return (x - x1) / (x2 - x1);
21             return (x2 - x) / (x2 - x1);
22         }
23     }
24
25     throw new ArgumentOutOfRangeException(''Given number X is not valid'');
```

```
26 }
```

- *FuzzyInputSet* - класа која представља фази скуп. Она је описана именом, идентификатором, листом функција припадности које описују скуп. Метода *RecalculateFunctionsValues* служи да иницира поновно израчунавање вредности свих функција припадности тог скупа. У пракси, приликом сваког добијања нових информација са сензора, потребно је да поново израчунамо вредност сваке функције припадности.

```
1 public void RecalculateFunctionsValues(float x)
2 {
3     Values.ForEach(e => e.CalculateFunctionValue(x));
4 }
```

- *FuzzyOutput* - класа која представља једну функцију припадности излазног скупа. Она је описана именом, идентификатором, листама X и Y координата тачака које описују ту функцију припадности, вредношћу пондерисаног просека. Вредности функције припадности су из интервала [0,1], тако да се пондерисани просек може рачунати као збир x-координата свих тачака чија је вредност y-координате једнака 1, подељен са бројем тачака које су учествовале у збиру. Следи код ове методе:

```
1  public float CalculateWeightedAverage()
2  {
3      float average = 0;
4      float n = 0;
5      foreach (var point in Points)
6      {
7          if(point.Y == 1)
8          {
9              average += point.X;
10             ++n;
11         }
12     }
13
14     if (n == 0)
15         throw new DivideByZeroException(''List of points is empty'');
16
17     return average / n;
18 }
```

- *FuzzyOutputSet* - класа која представља излазни фази скуп. Она је описана именом, идентификатором, листом функција припадности које описују излазни скуп. Ова класа ће бити касније коришћена за приступ функцијама припадности, њиховим пондерисаним просецима и вредностима.

- *FuzzyRules* - класа која представља фази правила. Она је описана са два улазна фази функције припадности (*FuzzyInput*), једном излазном фази функцијом припадности (*FuzzyOutput*) и логичким оператором. Логички оператор може бити логичко И и ИЛИ, а у кôду су представљене енумерацијом. Најважнији метод ове класе је *RecalculateOutputValue* који ће израчунати вредност излазној фази функцији припадности на основу тренутне вредности и вредности улазних фази функција. Следи кôд ове методе:

```
1  public void RecalculateOutputValue()
2  {
3      if (Operator == LogicOperator.AND)
4          Output.Value = Math.Max(Output.Value, Math.Min(Input1.Value,
               Input2.Value));
5      else
6          Output.Value = Math.Max(Output.Value, Math.Max(Input1.Value,
               Input2.Value));
7  }
```

- *FuzzySystem* - класа која представља један фази систем. Састоји се од листе улазних фази скупова (*List<FuzzyInputSet>*), једног излазног фази скупа (*FuzzyOutputSet*) и листе фази правила (*List<FuzzyOutputSet>*). Навешћемо методе ове класе:

  - *ChangeInputSetValue* - мења вредност једном улазном скупу на основу његовог идентификатора или имена.
  - *UpdateOutputValue* - за свако правило које се налази у систему, поново рачуна вредност излазне фази функције припадности.
  - *ResetOutputvalue* - за свако правило у систему, ресетује вредност излазне фази функције припадности.
  - *CalculateOutput* - ова метода представља дефазификатор у фази систему и најважнија је метода ове класе. У нашем систему, за дефа-

зификацију користимо методу пондерисаног просека за рачунање излазне вредности.

Следе кôдови наведених метода:

```
1  public void ChangeInputSetValue(float value, int? id, string name = null)
2  {
3      InputSets.ForEach(set =>
4      {
5          if ((id.HasValue && set.Id == id) || (name != null && set.Name
               == name))
6          {
7              set.RecalculateFunctionsValues(value);
8              return;
9          }
10     });
11 }
12
13 private void UpdateOutputValue()
14 {
15     Rules.ForEach(rule => {
16         rule.RecalculateOutputValue();
17     });
18 }
19
20 private void ResetOutputValue()
21 {
22     Rules.ForEach(rule => {
23         rule.ResetOutputValue();
24     });
25 }
26
27 public float CalculateOutput()
28 {
29     UpdateOutputValue();
30
31     float up = 0, down = 0;
32     foreach(var outputValue in OutputSet.Values)
33     {
34         up += outputValue.Value * outputValue.WeightedAverage;
35         down += outputValue.Value;
36     }
37
38     ResetOutputValue();
39     return (up / down);
40 }
```

Ова библиотека је пропраћена и јединичним тестовима као и докумената-

цијом у *XML* формату. Имплементација је урађена на основу алгоритма који смо навели у првој глави рада. Иницијализација система је замишљена тако да се прво наведу сви улазни фази скупови (*FuzzyInputSet*), а сваки од њих је листа фази променљивих, описаних одговарајућим функцијама припадности (*FuzzyInput*). Затим дефинишемо један излазни фази скуп и листу фази променљивих за тај излазни скуп. Од почетних подешавања, недостају нам још само фази правила (*FuzzyRules*), и коначно имамо дефинисан цео фази систем (*FuzzySystem*).

Излазна вредност се рачуна сваки пут када имамо нове улазне параметре. Њих у систему мењамо позивом методе *ChangeInputSetValue* са задатом новом вредношћу и именом или идентификатором фази скупа. Излазну вредност добијамо позивом методе *CalculateOutput*. Ове две методе су једине које ћемо често позивати, тако да је коришћење саме библиотеке веома једноставно.

4.4 AdminBoard - апликација за управљање системом за аутоматизацију

AdminBoard је апликација за управљање фази системом. У њој можемо да дефинишемо фази системе, скупове, одговарајуће фази функције припадности и правила. Поред креирања, сваки од наведених елемената фази система се може обрисати или изменити. Такође, ова апликација се користи и за комуникацију са уградним уређајем *Raspberry Pi*, тако да она представља језгро целокупне апликације за решавање проблема аутоматизације рада пластеника. Апликација је развијена уз помоћ радног оквира *ASP.Net Core*. Имплементирана је по *MVC (Model-View-Controller)* [1] архитектуралном шаблону за развој софтвера. Сама имплементација *MVC* шаблона у радном оквиру је таква да бекенд и фронтенд могу бити спрегнути, што ћемо искористити у имплементацији. Развој апликације је бржи, пресликавање долазних објеката на обе стране није потребно јер све то радни оквир ради за нас. Писање интеграционих тестова који проверавају да ли су објекти које обе стране очекују приликом слања *HTTP* захтева није потребно, декларисање модела нам даје гаранције. Сама логичка комплексност апликације није толика да нам коришћење *MVC* шаблона представља проблем.

4.5 Фронтенд апликације

Фронтенд користи технологију *Razor Pages* за креирање динамичких веб страница. Основа *Razor Pages* страница је базирана на *HTML5*, у који се додаје *C#* кôд који уводи динамичност. Наравно, као и у свакој *HTML5* страници, можемо користити и *JavaScript* кôд. За страницу можемо дефинисати ***модел*** са којим желимо да радимо на тој страници - било да га прегледамо, креирамо или мењамо. Ти модели су заправо објекти за пренос информација из једног дела апликације у други (*DTO - Data transfer object*). Важно је навести да ови објекти смеју само да преносе податке, у њима не треба да буде имплементирана било каква манипулација над тим подацима, осим евентуално метода које трансформишу један тип објеката у други. Ово је главни начин преноса информација са нивоа контролера на ниво погледа, односно *VC* веза у *MVC* шаблону. Дизајнерски шаблон је направљен од стране компаније *CreativeTim*, и преузет је са следећег линка: `https://www.creative-tim.com/product/argon-dashboard-asp-net`. Сам шаблон пружа већину потребних компонената за приказивање садржаја. За приказивање графикона је коришћена библиотека *ChartJs*. Навешћемо скраћен кôд фајла *Values/Edit.cshmtl* и показати неке специфичности *Razor Pages* технологије:

```
1  @model AdminBoard.Models.ValueViewModel
2
3  <script
       src=''//ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js''></script>
4  <script src=''https://cdn.jsdelivr.net/npm/chart.js''></script>
5
6  <div class=''row'' id=''edit''>
7      <div class=''col''>
8          <div class=''card w-75 mx-auto''>
9              <div>
10                 <h3 class=''pt-4 text-center''> Edit value  </h3>
11                 <form method=''post'' asp-controller=''Values'' asp-action=''Edit''
                       class=''p-4''>
12
13                     @*Canvas*@
14                     <canvas id=''chart''></canvas>
15                     <input type=''hidden'' asp-for=''ValueID''/>
16
17                     @*Name*@
18                     <div class=''form-group''>
19                         <label for=''nameInput''>Name</label>
20                         <input type=''text'' class=''form-control''
```

```
                            id=''nameInput'' placeholder=''Enter fuzzy value
                            name'' asp-for=''Name''>
21                      </div>
22
23                      ...
24
25                      @*Save*@
26                      <div class=''form-group w-100 mx-auto''>
27                          <div class=''input-group-addon text-center''
                                id=''buttons''>
28                              <button class=''btn btn-outline-success''
                                    type=''submit''>Save</button>
29                              <a class=''btn btn-outline-warning''
                                    href=''@Url.Action(''Index'', ''Values'')''>Back</a>
30                          </div>
31                      </div>
32                  </form>
33              </div>
34          </div>
35      </div>
36  </div>
37
38  <script>
39      const config = {
40         ...
41      };
42      const myChart = new Chart(document.getElementById(''chart''), config);
43
44      const XCoordFunc = () => {
45          let array = JSON.parse(''['' + $(''#XCoords'').val() + '']'');
46          myChart.data.labels = array.map((e) => { return String(e) });
47          myChart.update();
48      }
49
50      const YCoordFunc = () => {
51         ...
52      }
53
54      $(''#XCoords'').change(() => { XCoordFunc() });
55      $(''#YCoords'').change(() => { YCoordFunc() });
56
57      $(''#edit'').ready(() => {
58          $(''SelectedSet'').val(@Model.SelectedSet);
59          XCoordFunc();
60          YCoordFunc();
61      });
62  </script>
```

- У првој линији кôда, декларишемо који тип ће бити коришћен за модел ове странице. Сваки *C#* кôд у *Razor* страницама почиње симболом @.

- У линији 11, наводимо форму која ће бити послата *POST* методом. Дефинишемо и два додатна параметра: *asp-controller* и *asp-action*, који показују којој акцији и контролеру ће бити прослеђени подаци. У позадини, свака акција у контролеру је заправо један *endpoint* са кога се шаљу или примају подаци. Објекти се шаљу у *JSON* формату и аутоматски се пресликавају на жељене *C#* моделе.

- У линијама 15 и 20 имамо параметар *asp-for* који означава које поље модела је пресликано.

- У линији 29, као везу ка другој страници не морамо да уносимо линк ка њој. Довољно је да дефинишемо контролер и методу, а *Url.Action* метода ће урадити тај посао за нас.

- Од линије 38 па до краја, налази се *JavaScript* кôд који је потребан за генерисање графикона који представља фази функцију припадности.

Следе снимци екрана на којима је приказана измена фази функције припадности и измена корисничког профила, редом (слике 4.7 и 4.8)

4.6 Бекенд апликације

Имплементација бекенд дела је започета имплементацијом ***контролера***. Контролери су класе чије методе представљају *endpoint*-е, односно *URL-ове* са којих можемо да добијамо или шаљемо податке на сервер. Методе у контролерима називамо и ***акцијама***. Свака акција мора бити декорисана атрибутом који означава тип *HTTP* захтева. Неки од тих атрибута су: *[HttpGet], [HttpPost], [HttpDelete].* Такође, атрибутом можемо навести и руту која ће активирати акцију декорисану тим атрибутом. У случају да не наведемо, користиће се следећи шаблон: *baseUrl/controller/action*, у коме *baseUrl* означава основну путању саме апликације, а *controller* и *action* одговарајући контролер и акцију, по називу како смо их назвали у кôду. За повратни тип акција је изабран предефинисани интерфејс *IActionResult* који енкапсулира већину повратних типова који нама требају. Неке имплементације интерфејса су:

- *RedirectToAction* - Повратни тип који позива неку другу акцију у контролеру, најчешће ону која ће вратити неки поглед, односно *Razor* страницу.

- *StatusCode* - У зависности од додатних аргумената, кôдне грешке и додатне поруке, враћа *HTTP* одговор.
- *JsonResult* - Враћа објекат пресликан у *JSON* формату, и статусни кôд 200.

Над контролерима можемо да дефинишемо и ауторизационе елементе. Важе на нивоу свих акција у контролеру. Ауторизација се може увести или укинути и на акцијама појединачно. У апликацији за ауторизацију је коришћена библиотека *Microsoft Identity*.

У конструктору контролера дефинишемо сервисе који су нам потребни. Обично се ту налазе сервиси који дохватају податке из базе података, односно репозиторијуми, сервиси који дохватају податке помоћу *HTTP* протокола, разни механизми за логовање стања и рада апликације. У нашем кôду се налазе сервиси који дохватају податке из базе података. Поред њих, имамо и менаџере за аутентикацију и ауторизацију, као и сервис који је задужен за приказивање искакајућих нотификација на фронт-енду.

Следи кôд као илустрација једног контролера. У овом случају, приказан је контролер фази скупова, неке акције су изостављене ради компактнијег приказа. Поља класе су сервиси које ће контролер користити, конструктор иницијализује те сервисе. Све остале методе представљају акције.

```
1  [Authorize]
2     public class SetController : Controller
3     {
4         private readonly ILogger<SetController> _logger;
5         private readonly UserManager<User> _userManager;
6         private readonly SignInManager<User> _signInManager;
7         private readonly SetService _setService;
8         private readonly SubsystemService _subsystemService;
9         private readonly INotyfService _notificationService;
10
11        public SetController(
12            ILogger<SetController> logger,
13            UserManager<User> userManager,
14            SignInManager<User> signInManager,
15            SubsystemService subsystemService,
16            SetService setService,
17            INotyfService notificationService)
18        {
19            _logger = logger;
20            _userManager = userManager;
```

```
21              _signInManager = signInManager;
22              _setService = setService;
23              _subsystemService = subsystemService;
24              _notificationService = notificationService;
25          }
26
27          [HttpGet]
28          public IActionResult Index()
29          {
30              var sets = _setService.GetAll();
31              ViewBag.Names = _subsystemService.GetNames();
32              return View(''Index'', sets);
33          }
34
35          [HttpGet]
36          public IActionResult Create()
37          {
38              SetViewModel setViewModel = new(_subsystemService.GetAll());
39              return View(''Create'', setViewModel);
40          }
41
42          [HttpPost]
43          public IActionResult Create(SetViewModel model)
44          {
45              try
46              {
47                  _setService.Insert(model.ConvertToSet());
48                  _notificationService.Success(''Set inserted successfully.'');
49                  return RedirectToAction(''Index'');
50              }
51              catch (Exception e)
52              {
53                  _notificationService.Error(''Failed to insert set'');
54                  return StatusCode(500, $''Failed to create set: {e.Message}'');
55              }
56          }
57
58          [HttpDelete]
59          public IActionResult Delete(int id)
60          {
61              try
62              {
63                  _setService.Delete(id);
64                  return Json(true);
65              }
66              catch (Exception e)
67              {
68                  _notificationService.Error(''Failed to delete set'');
69                  return StatusCode(500, $''Failed to delete set: {e.Message}'');
70              }
71          }
```

```
72      }
```

У инфраструктурном слоју апликације се налазе сервиси. Имплементација је урађена тако да се сервис може посматрати да је имплементиран по шаблону за дизајнирање репозиторијума. У њима се налазе методе за дохватање свих објеката, одређеног објекта, креирање нових, измену и брисање постојећих. Иако се овај слој у нашој имплементацији може шаблонизирати тако да се једна шаблонска класа инстанцира за различите типове података, то није урађено ради могућности за додавање специфичности за сваки тип података који може бити шаблон. У већини сложенијих имплементација се не ради шаблонизирање управо из наведеног разлога.

Следи кôд као илустрација једног сервиса. У овом случају, приказан је сервис за рад са фази правилима:

```
1  public class RuleService
2      {
3          private readonly ILogger<RuleService> _logger;
4          private readonly FuzzyGreenhouseDbContext _context;
5
6          public RuleService(ILogger<RuleService> logger,
               FuzzyGreenhouseDbContext context)
7          {
8              _logger = logger;
9              _context = context;
10         }
11
12         public List<Rule> GetAll()
13         {
14             try
15             {
16                 return _context.Rule.ToList();
17             }
18             catch (Exception e)
19             {
20                 _logger.LogError($''Failed to get all rules. Message:
                       {e.Message}'');
21                 throw new Exception(e.Message);
22             }
23         }
24
25         public Rule Get(int id)
26         {
27             try
28             {
29                 return _context.Rule.Where(e => e.RuleID == id).First();
```

```
30              }
31              catch (Exception e)
32              {
33                  _logger.LogError($"Failed to get rule with ID={id}. Message:
                        {e.Message}");
34                  throw new Exception(e.Message);
35              }
36          }
37
38          public void Insert(Rule rule)
39          {
40              try
41              {
42                  _context.Rule.Add(rule);
43                  _context.SaveChanges();
44              }
45              catch (Exception e)
46              {
47                  _logger.LogError($"Failed to add new rule. Message:
                        {e.Message}");
48                  throw new Exception(e.Message);
49              }
50          }
51
52          public void Delete(int id)
53          {
54              ...
55          }
56
57          public void Update(Rule rule)
58          {
59              ...
60          }
61      }
```

У слоју података се налазе контексти база података. То су класе које дефинишу ентитете и релације које ћемо користити у бази података. *Entity Framework Core* је радни оквир за објектно релационо пресликавање који је коришћен за пресликавање ентитета у бази података и класних објеката које користимо у контролерима. Коришћен је ***предност-кôду*** (*code-first*) приступ, у коме се комплетна база података дефинише кроз кôд. Ентитети се дефинишу класама, а колоне у ентитетима се дефинишу пољима класа. Свако поље може да има подразумевану вредност, што *C#* као језик дозвољава, такође можемо да имамо и недостајуће вредности (коришћењем вредности *null*).

Најпре ћемо приказати класу која представља базу података, односно контекстну класу:

```
1  public class FuzzyGreenhouseDbContext : DbContext
2      {
3          public FuzzyGreenhouseDbContext() { }
4
5          public
               FuzzyGreenhouseDbContext(DbContextOptions<FuzzyGreenhouseDbContext>
               options) : base(options) { }
6
7          public DbSet<Rule> Rule { get; set; }
8          public DbSet<Set> Set { get; set; }
9          public DbSet<Value> Value { get; set; }
10         public DbSet<Version> Version { get; set; }
11         public DbSet<Subsystem> Subsystem { get; set; }
12
13         protected override void OnModelCreating(ModelBuilder modelBuilder)
14         {
15             modelBuilder.ApplyConfiguration(new RuleConfiguration());
16             modelBuilder.ApplyConfiguration(new SetConfiguration());
17             modelBuilder.ApplyConfiguration(new ValueConfiguration());
18             modelBuilder.ApplyConfiguration(new VersionConfiguration());
19             modelBuilder.ApplyConfiguration(new SubsystemConfiguration());
20         }
21     }
```

У свакој класи која представља ентитет се налазе поља која представљају колоне те класе. У њима се могу налазити и објекти других ентитета, што ће се у бази превести као постојање страног кључа једног ентитета на други. За сваки ентитет можемо додатно дефинисати и неку додатну конфигурацију: рад са индексима, додатним страним кључевима, конверзијама типова, итд. Радни оквири за објектно релационо пресликавање могу да сами донесу одлуке о томе које индексе и кључеве треба дефинисати, али предложено решење не мора да буде оптимално. Креирање одговарајућих индекса у бази представља један од изазова приликом пројектовања самих база. Самим тим и креирање индекса је потенцијалан проблем и за радне окивире за пресликавања у случајевима када имамо јако велике ентитете над којима се извршавају различити упити. Да би се база креирала први пут, као и да би се применила било каква измена, користе се миграције. У бази се увек, поред самих ентитета које желимо да чувамо, чувају и слике архитектуре саме базе, тј. ентитети и релације без самих података. Свака миграција као резултат има нову слику архитектуре, тако да са тог аспекта увек можемо да се вратимо на произвољно стање базе, а подаци између те две слике би били изгубљени. Миграције су класе које имају две методе: прва је метода у којој се дефини-

шу нове промене које требају бити примењене да бисмо применили промене у односу на последњу слику архитектуре базе, а у другој методи су наредбе које поништавају ефекте прве методе. Са дефинисаним сликама архитектуре базе, и списком миграција, увек можемо да имамо било које стање базе. Ово је врло корисно у реалним апликацијама, међутим треба бити врло опрезан са променама - неке могу довести до неповратног губитка података.

Следи кôд који приказује ентитет који моделира фази функцију припадности и његова конфигурација. У овој конфигурацији правимо конверзију између *C#* листе бројева и њене стринг интерпретације. У бази не постоји тип листа података, тако да је ово решење које се природно наметнуло.

```
1  public class Value
2      {
3          public Value() { }
4
5          public Value(string valueName, List<float> xs, List<float> ys)
6          {
7              Name = valueName;
8              XCoords = xs;
9              YCoords = ys;
10         }
11
12         public int ValueID { get; set; }
13         public string Name { get; set; }
14         public List<float> XCoords { get; set; }
15         public List<float> YCoords { get; set; }
16
17         public Set Set { get; set; }
18         public int SetID { get; set; }
19     }
20
21     public class ValueConfiguration : IEntityTypeConfiguration<Value>
22     {
23         public void Configure(EntityTypeBuilder<Value> builder)
24         {
25             var listConverter = new ValueConverter<List<float>, string>(
26                 v => string.Join(",", v),
27                 w => w.Split(',', System.StringSplitOptions.None).Select(q =>
                       float.Parse(q)).ToList()
28             );
29
30             builder.Property("XCoords").HasConversion(listConverter);
31             builder.Property("YCoords").HasConversion(listConverter);
32         }
33     }
```

Најбитније аспекте веб апликације за контролисање фази система смо појаснили, као и неке основне дизајнерске и архитектуралне шаблоне који су коришћени у њима. Имплементација саме апликације је углавном пратила добре праксе модерног програмирања базирано на домену и проблему који желимо да моделирамо и решимо. Наравно, имплементација није искључиво пратила таква упутства и савете, већ је имплементирана тако да буде лако читљива и да се може лако тестирати.

4.7 GreenhouseCore - апликација за уградни уређај

У овом одељку су приказани детаљи апликације која се извршава на уградном уређају. *GreenhouseCore* је *.Net Core* апликација која је задужена за комуникацију са базом података на којој се налазе подешавања и вредности фази система, читање вредности са улазних сензора и пренос израчунатих вредности на излазне уређаје. Подсетимо се да је архитектура целокупног система илустрована на слици 4.4.

Сваки модул апликације има свој одговарајући директоријум. Библиотека за рад са фази системима (одељак 4.3) је коришћена у оквиру ове апликације. Сама апликација је предвиђена да се константно извршава на уградном уређају, попут системског задатка.

DB Bridge је модул који је задужен за комуникацију са базом података. Повезивање на базу података се динамички остварује путем одговарајућег повезујућег стринга, током самог извршавања апликације. Две основне операције које се могу извршити на овом модулу су:

- *FetchLatestVersion* – саме вредности фази система се могу невезано од ове апликације мењати на *AdminBoard*-у. Уградни уређај мора бити свестан са којом верзијом података ради, и да их освежи по потреби.
- *FetchData* – дохвата вредности из базе података и преводи на типове са којима *FuzzyLib* ради.

Hardware Bridge је модул који је задужен за комуникацију са излазним уређајима. У њему се налазе следеће класе:

- *RPi3Middleware* – класа задужена за комуникацију са *MCP3008* аналогно-дигиталним конвертером. За потребе овог рада и тестирања рада апликације, овај аналогно-дигитални конвертер ће нам представљати апстракцију контролера уређаја и сензора.

  – У конструктору класе иницијализујемо *SPI* магистралу, дефинишемо фреквенцију часовника ради усклађивања и идентификатор магистрале ради одабира одговарајућег уређаја. По изласку из конструктора имамо успешно повезан аналогно-дигитални конвертер на уградни уређај. Само повезивање на уређај је остварено помоћу постојећег радног оквира *.Net Core* за уградне уређаје.

  – *ReadValueFromADConverter* је метода која служи за дохватање вредности са одговарајућег пина. Пин се уноси као параметар методе. Вредност коју добијамо је реалан број из интервала [0,100] који представља проценат максималне вредности коју дати сензор може да прочита. Без знања о самом сензору, овај податак нема употребну вредност. У наставку ћемо објаснити како добијамо податке који нам служе за улаз у фази систем.

- *Sensor* – Ова класа представља један сензор. Имаћемо више сензора који су повезани на уградни уређај и поља ове класе пружају додатне информације. Класа садржи следећа поља:

  – *Name*: име сензора, ради лакшег приказа приликом рада самог система и потреба дебаговања.

  – *DatabaseId*: референца на који фази скуп овај сензор реферише

  – *MinValue, MaxValue*: граничне вредности које сензор може да детектује. Вредност која је прочитана ће бити израчуната уз помоћ процента који смо дефинисали у прошлој класи и овог поља.

  – *Description*: додатне информације о самом сензору, ово поље је опционо .

- *PinoutConfigurations* – класа која представља везу између наведене две, и у којој се врши конфигурација пинова на уградном уређају. Од поља имамо два речника: један који пресликава улазне сензоре са идентификаторима улазних фази скупова и други који пресликава излазне.

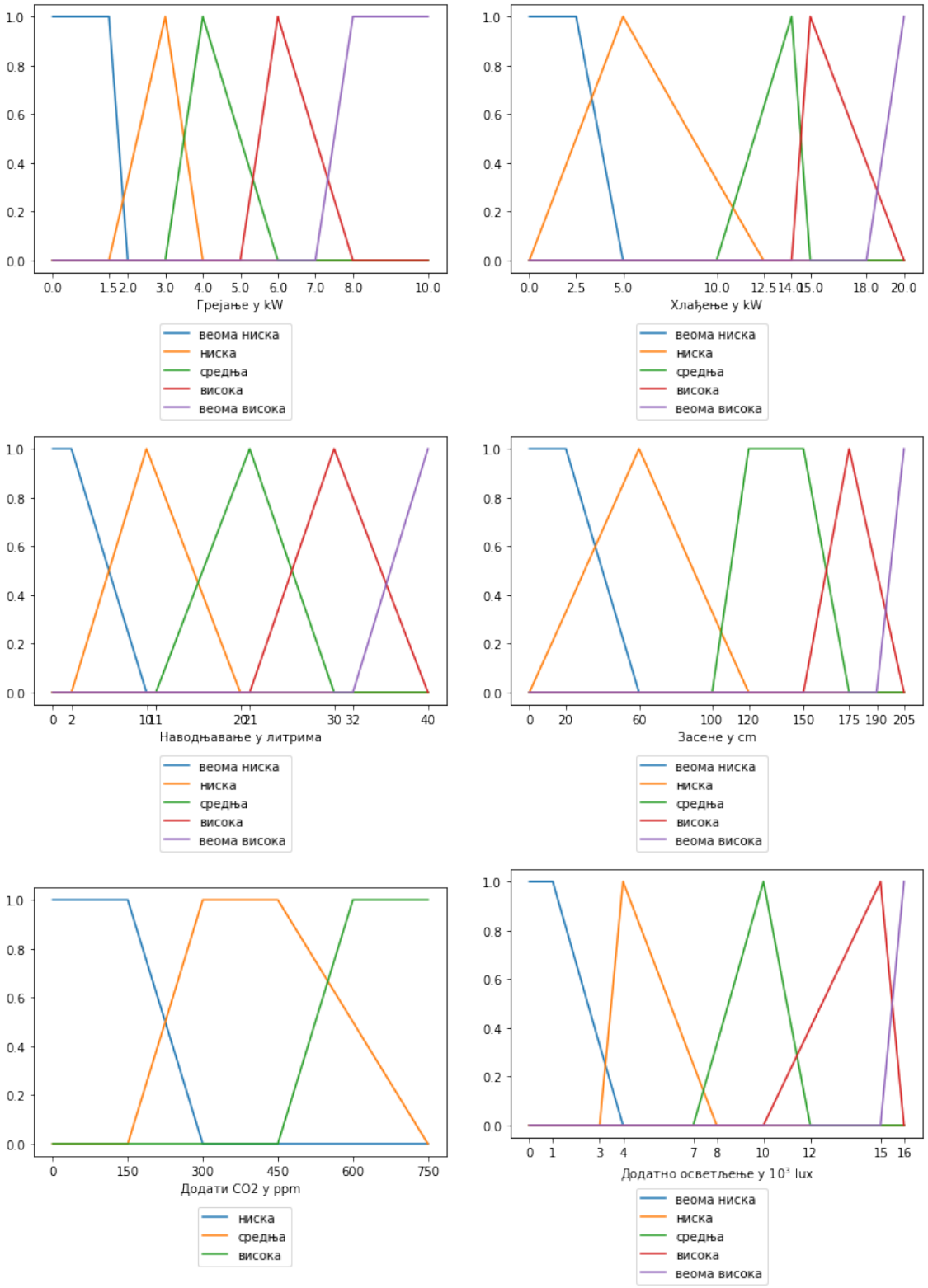
Следи приказ кôда наведених класа:

```
1  public static class RPi3Middleware
2      {
3          public static SpiConnectionSettings spiConnectionSettings { get;
               private set; }
4          public static SpiDevice spiDevice { get; private set; }
5          public static Mcp3008 adConverter { get; private set; }
6
7          private const int clkFrequency = 1000000;
8
9          static RPi3Middleware()
10         {
11             spiConnectionSettings = new SpiConnectionSettings(0, 0)
12             {
13                 ClockFrequency = clkFrequency
14             };
15
16             spiDevice = SpiDevice.Create(spiConnectionSettings);
17
18             adConverter = new Mcp3008(spiDevice);
19         }
20
21         public static double ReadValueFromADConverter(int pinNumber)
22         {
23             // Method will return number from range [0,100] which represent
24             // percentage of potentiometer
25
26             if(pinNumber < 0 || pinNumber > 5)
27             {
28                 throw new ArgumentException(''Invalid pin number. Channels
                       available: [0,1,2,3,4,5]'');
29             }
30
31             double value = adConverter.Read(pinNumber);
32             value /= 10.24;
33             value = Math.Round(value);
34
35             return value;
36         }
37     }
38
39     public class PinoutConfigurations
40     {
41         public Dictionary<int, Sensor> InputsPinoutConfigurations { get; set;
               }
42         public KeyValuePair<int, Sensor> OutputPinoutConfiguration { get;
               set; }
43
44         public PinoutConfigurations()
```

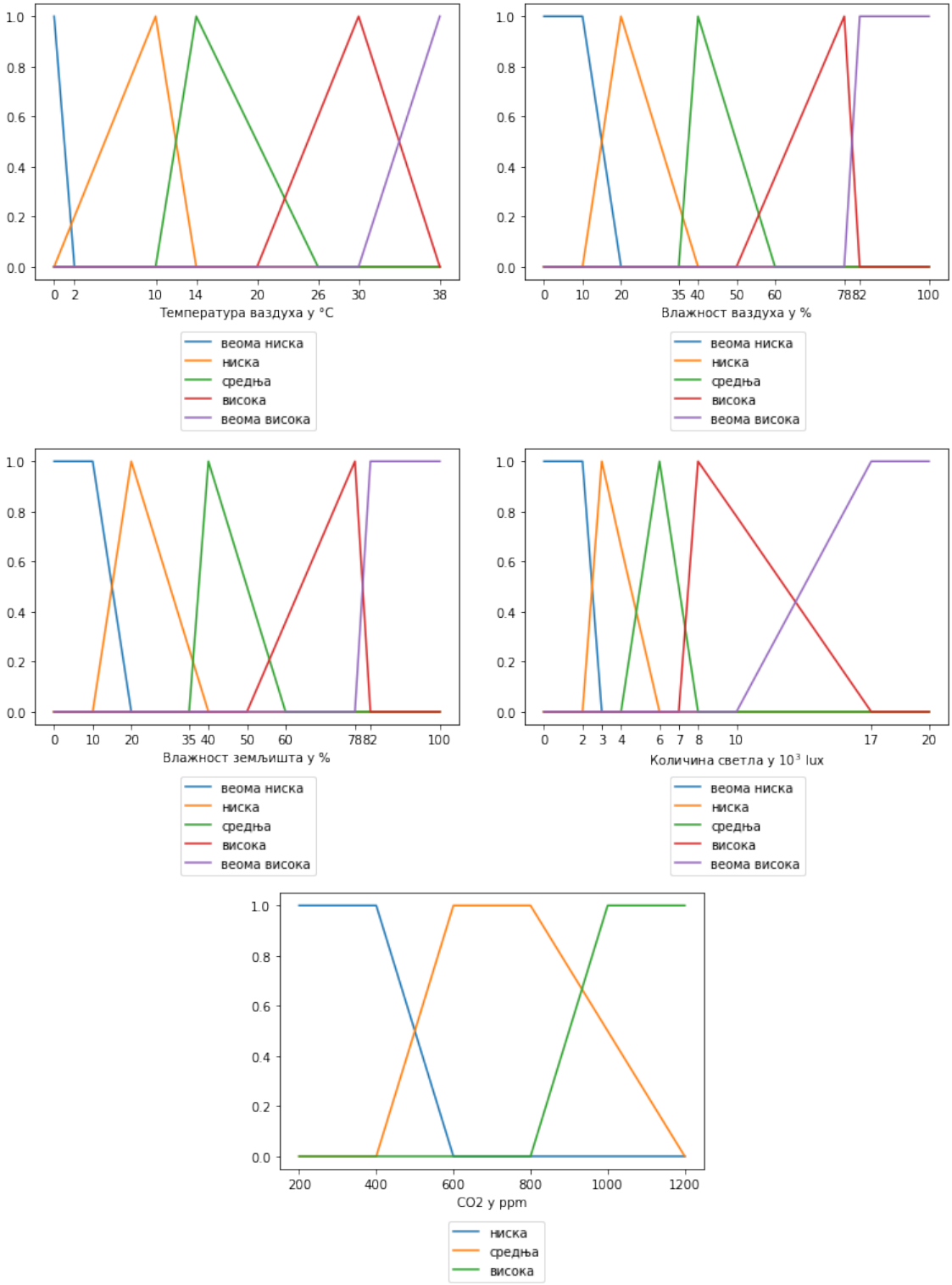
```
45          {
46              InputsPinoutConfigurations = new Dictionary<int, Sensor>();
47          }
48
49          public void AssignInputPin(Sensor sensorInput, int pinNumber)
50          {
51              if(InputsPinoutConfigurations.ContainsKey(pinNumber))
52              {
53                  throw new ArgumentException($"Pin number {pinNumber} is
                        occupied by sensor:
                        {InputsPinoutConfigurations[pinNumber]}");
54              }
55
56              InputsPinoutConfigurations.Add(pinNumber, sensorInput);
57          }
58
59          public void AssingOutputPin(Sensor sensorOutput, int pinNumber)
60          {
61              OutputPinoutConfiguration = new KeyValuePair<int,
                    Sensor>(pinNumber, sensorOutput);
62          }
63
64          public double ReadPinValuePercentage(int pinNumber)
65          {
66              return RPi3Middleware.ReadValueFromADConverter(pinNumber);
67          }
68
69          public double ReadPinValueInRange(int pinNumber)
70          {
71              var sensorInput = InputsPinoutConfigurations[pinNumber];
72              var value = sensorInput.MinValue +
73                          (sensorInput.MaxValue - sensorInput.MinValue) *
                                RPi3Middleware.ReadValueFromADConverter(pinNumber)
                                / 100;
74              return Math.Round(value, 2);
75          }
76
77          public void DisplayPinoutAndValues()
78          {
79              Console.WriteLine("{0,15}{1,15}{2,15}{3,15}{4,15}{5,15}",
80                  "Pin",
81                  "Name",
82                  "MinValue",
83                  "MaxValue",
84                  "Value %",
85                  "Value"
86              );
87              Console.WriteLine("----------------------------------------------------------------
88
89              foreach (var pair in InputsPinoutConfigurations)
90              {
```

```
91                  Console.WriteLine("{0,15}{1,15}{2,15}{3,15}{4,15}{5,15}",
92                      pair.Key,
93                      pair.Value.Name,
94                      pair.Value.MinValue,
95                      pair.Value.MaxValue,
96                      ReadPinValuePercentage(pair.Key),
97                      ReadPinValueInRange(pair.Key)
98                  );
99              }
100         }
```

Навели смо преглед имплемнтација самих апликација за решење проблема аутоматизације пластеника. Репозиторијум на ком се налази комплетан кôд апликација се може пронаћи на следећем линку: `https://github.com/bozzano101/FuzzyGreenhouse`



Слика 4.5: Графикони фази променљивих излазних уређаја



Слика 4.6: Графикони фази променљивих улазних сензора

Грејање

Влажност ваз / Темп. ваз	Веома ниска	Ниска	Средња	Висока	Веома висока
Веома ниска	Веома јако	Веома јако	Јако	Јако	Јако
Ниска	Веома јако	Јако	Јако	Средње	Средње
Средња	Средње	Средње	Средње	Средње	Слабо
Висока	Средње	Слабо	Слабо	Слабо	Слабо
Веома висока	Веома слабо	Веома слабо	Веома слабо	Веома слабо	Веома слабо

Вентилација

Влажност ваз / Темп. ваз	Веома ниска	Ниска	Средња	Висока	Веома висока
Веома ниска	Веома слаба	Веома слаба	Веома слаба	Веома слаба	Веома слаба
Ниска	Слаба	Слаба	Слаба	Слаба	Слаба
Средња	Слаба	Средња	Средња	Средња	Средња
Висока	Средња	Средња	Средња	Јака	Јака
Веома висока	Јака	Јака	Јака	Веома јака	Веома јака

Наводњавање

Влажност ваз Влажност земљ.	Веома ниска	Ниска	Средња	Висока	Веома висока
Веома ниска	Веома јако	Јако	Јако	Средње	Слабо
Ниска	Веома јако	Јако	Средње	Средње	Слабо
Средња	Јако	Јако	Средње	Средње	Слабо
Висока	Јако	Средње	Средње	Слабо	Веома слабо
Веома висока	Средње	Средње	Слабо	Слабо	Веома слабо

Ролетне за засену

Кол светлост / Темп ваз	Веома мала	Мала	Средња	Велика	Веома велика
Веома ниска	Веома мала	Веома мала	Веома мала	Мала	Мала
Ниска	Веома мала	Мала	Мала	Мала	Средња
Средња	Веома мала	Мала	Средња	Средња	Велика
Висока	Мала	Мала	Средња	Велика	Велика
Веома висока	Мала	Средња	Средња	Велика	Веома велика

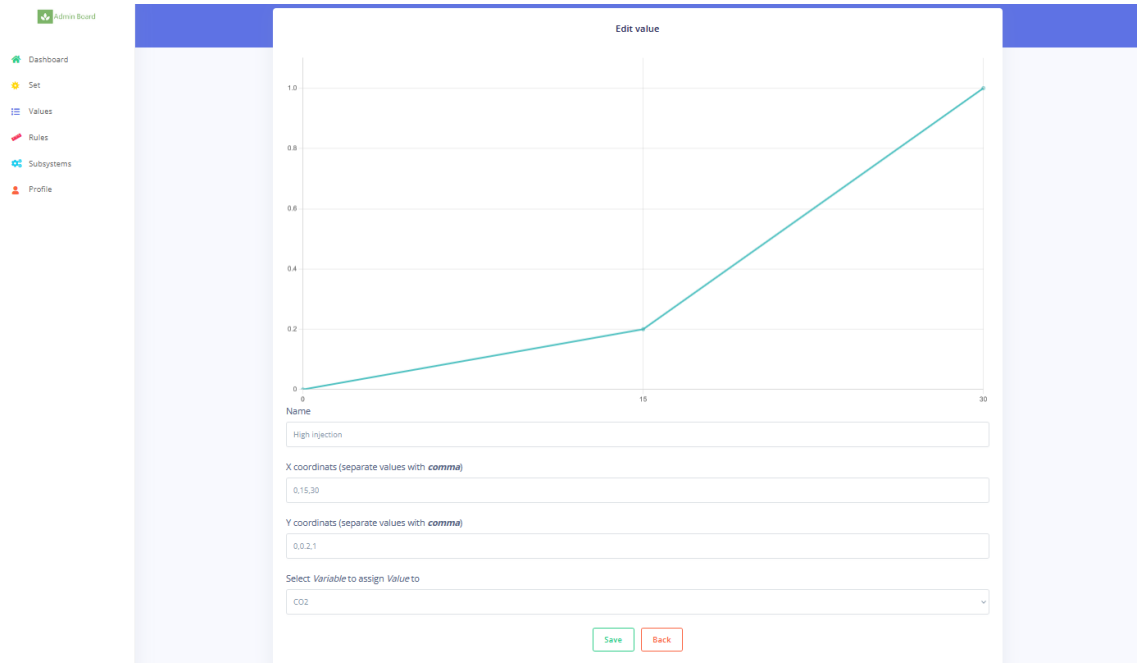
Додатно осветљење

Кол светлост	Веома мала	Мала	Средња	Велика	Веома велика
	Веома јако	Јако	Средње	Слабо	Веома слабо

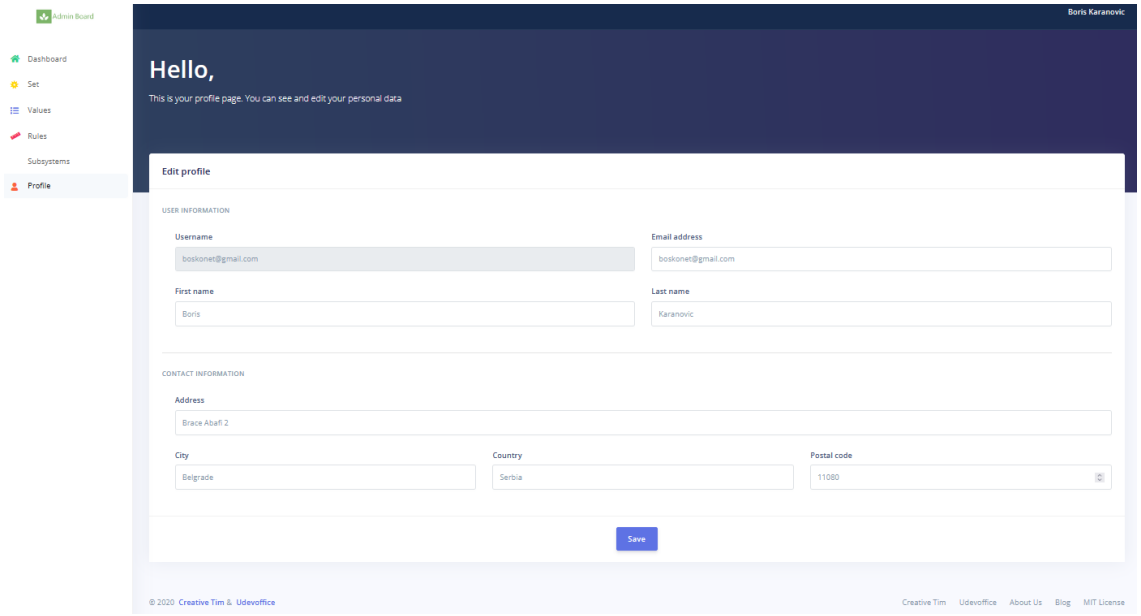
Додатно осветљење

Кол ЦО2	Мала	Средња	Велика
	Велика	Средња	Мала

Табела 4.1: Табела правила закључивања



Слика 4.7: Снимак екрана: Измена фази функције припадности



Слика 4.8: Снимак екрана: Измена корисничког профила

Глава 5

Анализа решења и потенцијалних проблема

5.1 Анализа резултата и упоређивање са постојећим системима

Предложени фази систем за одржавање пластеника оптимизује употребу електричне енергије, воде, ђубрива и осталих ресурса ради што успешнијег гајења биљака. На дугорочном периоду, уштеда енергије на грејању, осветљењу и заливању може бити огромна. Грејање пластеника се углавном изводи великим грејачима који раде на фосилна горива, ређе електричном енергијом. Количина воде која се користи у пластеницима такође није занемарљива, поготово у пределима у којима нема много природних вода, поред глобалног загревања које је све присутније. У пластенику је већина улаза међусобно зависна на неки начин. Навешћемо неколико примера:

- Ниска температура и висока влажност доводе до појаве плесни, а за сузбијање њих се користе разни хемијски препарати
- Заливање смањује температуру и повећава влажност ваздуха. Премало воде доводи до исушивања биљака, а превише воде може да доведе неке врсте до трулења.
- Коришћење засена доводи до смањивање топлоте, коришћење додатне расвете може да подигне температуру, превелика влажност може да

утиче и на додатне рефлекторе, да смање пролазност светла у случају маглења.

Фази контролер решава и проблеме узроковане овим зависностима.

Примена фази контролера на примеру пластеника, као и примена фази логике у пољопривреди су релативно често обрађиване теме. Сада ћемо навести неке примере, као и резултате које су аутори добили.

У [10] се користи влажност ваздуха, температура и количина светлости да би се контролисало грејање, засена и додатно осветљење. Коришћени су *PLC* (*programmable logic controller*) контролер и *KingView* софтвер. *PLC* контролери већину своје примене проналазе у индустријским постројењима. Доста су гломазни, отпорни су на хемикалије, температуру, влажност ваздуха и могу да раде у различитим условима. Програмирање је доста комплексно, и за најситнију промену је потребно ангажовати стручњака за програмирање искључиво ових уређаја. Флексибилност практично и да не постоји, рачунарска моћ такође није велика. Њихова основна сврха је аутоматизација неких процеса у индустрији који су константни и за које није потребно често прилагођавање. Иако са технолошког аспекта ово решење није најефикасније могуће, резултати који су наведени у овом раду представљају успех предложеног решења.

У [2] је предложен нови метод за контролу количине и притиска испарења воде на основу темпертуре и тренутне влажности ваздуха. Није предложена техничка имплементација решења, али је теоретски значај дефинисаног метода коришћен и у другим радовима и примењиван у пракси, где је дао добре резултате.

У [8] је приказан метод за контролисање наводњавања. Дефинисана је бежична мрежа сензора (*WSN*) која је мерила температуру и влажности ваздуха и земљишта, а помоћу фази логике дефинисан систем за одређивање количине воде којом треба да се пластеник залива. Резултат је смањење потрошње воде приликом заливања.

У [14] се наводи систем за контролу грејања, хлађења, наводњавања и осветљења базиран на фази логици. Са техничке стране је дефинисана бежична мрежа сензора а за централни део је коришћен *Android* уређај.

Веб апликацији за интеракцију корисника са системом, развијеној за потребе овог рада, је могуће приступити са било које тачке на Земљи, као и са било ког рачунара или мобилног телефона. Улазни сензори се врло лако, без потребе за знањем програмирања, могу унети у апликацију. Приликом уноса,

приказан је график функција припадности ради илустрације функције припадности у дводимензионој равни. Сам систем је модуларан, немамо никакве услове у смислу броја сензора, броја излазних уређаја (осим техничких способности *Raspberry Pi*). База знања која је неопходна за рад, фази променљиве и правила, се дефинишу преко корисничког интерфејса. Такође можемо дефинисати и више различитих фази контролера: један за микро-климу, а други за прихрану. Избор рачунара који ће прикупљати информације са сензора и издавати команде на излазне уређаје је разнолик, уз ограничења која смо навели. Број уређаја који испуњавају те услове је велик, и може се пронаћи уређај који ће задовољити услове и потребе пластеника.

У наведеној литератури су анализирана решења и предлози за унапређивање. Такође, фази правила која су наведена у раду су правила која су дефинисали доменски експерти која се налазе у литератури. Сам систем је имплементиран на модерној и ефикасној *.Net Core* платформи која се данас извршава на оперативним системима *Linux, macOS, Windows, Android, iOS* и архитектурама *IA-32, x86-64, s390x, ARM* чиме је покривено преко 95% [1] свих уређаја који су тренутно у употреби. Сви радови који су коришћени у литератури су навели успехе у циљевима који су били предложени.

5.2 Проблем неиспуњености закона непротивречности у фази логици

Током 70-тих година прошлог века, фази логику је одбацио значајан број лингвиста као теорију за решавање проблема недореченост у природном језику. Приликом навођења примера „да ли је дан погодан за излазак напоље и које се активности препоручују" увидели смо да фази променљиве и функције припадности подсећају на вероватноћу. Та сличност није случајна, и у литератури постоје радови у којима је креирана спрега између вероватноће и фази логике. *Bart Kosko* [7] је у свом раду почео од претпоставке да је вероватноћа подобласт фази логике. Вероватноћа не треба да буде подобласт фази логике, и аутор је показао да је фази логика базирана на вероватноћи. Показао је и да можемо помоћу вероватноће моделирати систем за управљање и да ће успех тог фази система зависити од вероватноће. Подсетимо се формула за

[1] https://gs.statcounter.com/os-market-share

рачунање пресека и уније два фази скупа које смо навели:

$$\begin{aligned} \mu_{A\cap B}(x) &= min\{\mu_A(x), \mu_B(x)\}, \forall x \in X \\ \mu_{A\cup B}(x) &= max\{\mu_A(x), \mu_B(x)\}, \forall x \in X \end{aligned} \tag{5.1}$$

Нека имамо неку вредност $\mu_A(x) = 0.5$, тада је вредност комплемента $\mu_A(x^C) = 1-\mu_A(x) = 0.5$. Ако применимо правило минимума за пресек ова два скупа, добијамо да је вредност пресека такође 0.5, а пресек два комплементна скупа је празан скуп, где би функција припадности требала да буде 0. Тиме закон непротивречности није задовољен. Доводи се у питање да ли су формуле наведене за рачунање функције припадности комплемента, формуле за пресек и унију ваљане. У практичним применама, најчешће се користе баш такве формуле за рачунање излаза фази система, из разлога врло простог израчунавања и ниске рачунарске захтевности.

Формула за рачунање коју смо навели се може извести из формуле класичне дефиниције вероватноће, у општем случају када догађаји A и B нису дисјунктни.

$$P(A \cap B) = P(A) + P(B) - P(A \cup B) \tag{5.2}$$

Формула коју смо навели за пресек је тачна и у фази логици и у вероватноћи у случају када A и B нису дисјунктни. У вероватноћи, ако су догађаји A и B независни, немају пресек, тада је и вероватноћа њиховог пресека 0, па би формула за унију била: $P(A \cup B) = P(A) + P(B)$.

Формуле за пресек у случају независних и условних догађаја би биле, редом:

$$\begin{aligned} P(A \cap B) &= P(A)P(B) \\ P(A \cap B) &= P(A|B)P(B) \end{aligned} \tag{5.3}$$

У случају дисјункције, логичко ИЛИ се може дефинисати на следеће начине:

$$\begin{aligned} P(A \cup B) &= max(P(A), P(B)) \\ P(A \cup B) &= P(A) + P(B) \\ P(A \cup B) &= P(A) + P(B) - P(A \cap B) \end{aligned} \tag{5.4}$$

Први начин је коректан ако имамо два догађаја која су зависна, али су међусобно дисјунктни. Други начин је коректан ако су догађаји независни и

узајамно дисјунктни. Трећи начин је коректан ако су догађаји независни, али нису узајамно дисјунктни. У фази логици, као формулу за унију користимо само први случај. У примеру који смо навели, влажност ваздуха није узајамно искључива са температуром, па формула за пресек важи. Ово не мора да важи у општем случају, када би одговор био нетачан.

Наведени проблеми су решени у систему који је назван *„Компензациони фази систем"* [9], у коме су редефинисани фази скупови и одговарајуће операције. Фази правила су остала идентична. У компензационом фази систему се користи геометријска средина, дефинисана као n-ти корен производа n бројева. Навешћемо формуле за пресек и унију:

$$\begin{aligned} P(A_1 \cap A_2 ... \cap A_n) &= \sqrt[n]{A_1 A_2 ... A_n} \\ P(A_1 \cup A_2 ... \cup A_n) &= 1 - \sqrt[n]{A_1 A_2 ... A_n} \end{aligned} \tag{5.5}$$

5.3 Потенцијални проблеми и предлози решења

Поставља се и питање да ли фази правила могу довољно добро да опишу понашање неког система да би могли да га успешно контролишемо. Као што смо навели, њих треба да дефинише доменски експерт за дато подручје примене. Најчешће се користе правила која моделирају начин употребе који ми желимо. Међутим, систем треба да ради и у абнормалним ситуацијама, када може да произведе излазе који могу довести до катастрофалних грешака (отказивање комплетног система). Решење за овај проблем се може наћи у повећаном броју правила, као и дефинисању понашања у неким специјалним случајевима. Теорија хаоса показује да све примене из реалног живота могу имати бесконачно могућности за понашање, а немогуће је покрити све могућности. Ово је проблем за све системе за управљање, не само за фази контролере.

За одлучивање коју излазну променљиву ћемо користити смо користили метод центроиде који рачуна упросечене вредности. Овај метод се користи зато што и ако дође до неке грешке, она ће бити упросечена и смањена је шанса да она доведе систем до потпуног отказивања.

Вредности функције припадност за фази променљиве су дефинисане на интервалу [0,1]. Ово није обавезно али је пожељно. Вредности за вероватноћу

се поклапају са овим доменом, што нам олакшава дефинисање формуле за пресек, унију, комплемент. Поред тога, могу се појавити и неки скривени проблеми. Препорука је да се произвољан домен преведе у интервал [0,1]. Најчешће се ради линеарно скалирање, али може и одсецање свих вредности изнад 1 и испод 0.

Такође, вредности улаза је потребно на неки начин ограничити. Откази сензора су догађаји који се дешавају и на које одговори морају бити предвидиви. Такође треба обратити пажњу на шум који сензор може да произведе. Начин преноса сигнала од сензора до самих контролера сензора је данас најчешће бежичан, па и ту имамо простор за појаву проблема.

5.4 Даља унапређења апликације

Дефинисали смо модул који аутоматизује опште параметре у пластенику. Уз одреднице доменских експерата, можемо дефинисати и неке друге модуле, попут контроле коришћења пестицида и контроле односа различитих хемијских једињења у ђубриву.

Библиотека за рад са фази системима има дефинисану методу пондерисаних просека за рачунање излазних вредности. Може проширити и неким другим методама. Већина тих метода доводи до подизања временске сложености израчунавања. Такође, користи се Мамданијев тип контролера, који је најзаступљенији у практичним применама. Поред њега постоји и Такаги-Сугенов тип контролера који такође може бити уврштен. Његова примена би захтевала и измене на корисничком порталу.

Од осталих грана вештачке интелигенције и алгоритама базираних на процесима у природи, оптимизацијаројем честица се данас удружује са фази системима ради производње бољих резултата. У примеру аутоматизације рада пластеника, оптимизацијаројем честица се може применити у процесу дефазификације, где нам је потребно да одредимо која ће излазна фази функција припадности имати највећи ефекат на резултат.

Сама апликација за рад са уградним уређајем је креирана за рад са аналогним сензорима, који су доста чешћи од дигиталних за потребе оптимизације параметара пластеника. Коришћење само ових сензора олакшава процес тестирања и откривања грешка на самом уградном уређају. Приликом тестирања, повезивање са уградни уређај се врши преко *SSH* протокола, и апликација

се покреће. Додатно ограничење уградног уређаја је да се процес дебаговања на даљину (*remote debugging*) [2], које представља стандард у свету програмирања уградних уређаја, са кућном конфигурацијом и *Raspberry Pi* уградним уређајем не може квалитетно извршити. Сам процес повезивања је јако спор, а радни оквир *.Net Core* који пружа подршку за рад са *MCP3008* конвертером не подржава опцију дебаговања на даљину у радном окружењу. У тренутку писања рада, као најбоље решење се наметнуо нови дебагер, *RaspberryDebugger* [3]. Унапређење процеса дебаговања ове апликације би свакако представљао вредно побољшање.

[2]https://learn.microsoft.com/en-us/visualstudio/debugger/remote-debugging?view=vs-2022
[3]https://github.com/StanDeMan/RaspberryDebugger

Глава 6

Закључак

Анализом и упоређивањем постојећих решења за аутоматизацију и контролу процеса у пластеницима и агрикултури се увидело да није развијен модуларан систем који би могао да аутоматизује различите процесе. Такође, технологије које су коришћене у некима од њих нису најпогоднije за ту примену. Предложено решење је модуларно, може аутоматизовати произвољан број параметара у границама колико уградни уређај дозвољава. Тај број параметара се додатно може проширити одговарајућим експандерским микрочиповима. Сам вишекориснички интерфејс је направљен са идејом да буде лако доступан доменским експертима који ће их користити и нема потребе за куцањем било каквог кôда ради подешавања. Сам одабир технологија које су коришћене у апликацијама дозвољава одабир различитих рачунара на којима ће радити. Не постоје нека већа рачунарска ограничења, и апликације успешно раде на скоро свим модерним рачунарима и уградним уређајима. Проблеми које смо описали у претходној глави се и даље налазе у већини примена фази логике. Ретко се дешавају у регуларним условима рада, а још ређе у абнормалним условима рада. На примеру пластеника, робустност и поузданост целог система није угрожена овим проблемима. Као највећи проблем се поставља безбедност и ауторизација целог система.

Фази контролери се данас користе у различитим доменима примене. Користе се и паралелно са неким другим алгоритмима вештачке интелигенције и еволутивним алгоритмима (генетски алгоритам, оптимизација роjем честица, оптимизација мрављом колонијом). Лакоћа примене фази логике и ниска рачунарска захтевност је такође један важан аспект примене на уградним уређајима. На питање шта се тренутно дешава са фази логиком, њеном при-

меном, као и каква је њена будућности, као и завршну реченицу, навешћемо цитат *Peter Clarke*-а из чланка објављеног у часопису *EETimes* [1]:

'But perhaps fuzzy logic's time has come.'

[1] https://www.eetimes.com/whatever-happened-to-fuzzy-logic/

Литература

[1] Pop D. и Altar A. „Designing an MVC Model for Rapid Web Application Development". У: *24th DAAAM International Symposium on Intelligent Manufacturing and Automation* (2013), стр. 1172–1179.

[2] Lim D.K, Ahn B.H. и Jeong J.H. „Method to control an air conditioner by directly measuring the relative humidity of indoor air to improve the comfort and energy efficiency". У: *Applied Energy* 215 (2018), стр. 290–299.

[3] Mamdani E. и Assilian S. „An experiment in linguistic synthesis with a fuzzy logic controller". У: *Elsevier International Journal of Man-Machine Studies* 1.7 (1975), стр. 1–13.

[4] Reference Module in Earth Systems и 2022 Environmental Sciences. *Liebig's Law of the Minimum*. hhttps://www.sciencedirect.com/topics/agricultural-and-biological-sciences/liebigs-law-of-the-minimum, pristupljeno 17.7.2023. 2022.

[5] Andries P. Engelbrecht. *Computational Intelligence: An Introduction, 2nd Edition*. John Wiley Sons, 2007.

[6] Encyclopedia of Food и 2016 Health. *Greenhouse Production*. https://www.sciencedirect.com/topics/agricultural-and-biological-sciences/greenhouse-production, pristupljeno 13.7.2023. 2016.

[7] Graeme Heald. „Issues with Reliability of Fuzzy Logic". У: *International Journal of Computational Intelligence Systems* (2018). DOI: 10.13140/RG.2.2.33328.40968.

[8] Mat I. и др. „IoT in Precision Agriculture applications using Wireless Moisture Sensor Network". У: *2016 IEEE Conference on Open Systems (ICOS)* (2016). DOI: 10.1109/ICOS.2016.7881983.

[9] Montero J. „Compensatory Fuzzy Logic". У: 32 (2011).

[10] Li L., Cheng K.W.E. и Pan J.F. „7th International Conference on Power Electronics Systems and Applications - Smart Mobility, Power Transfer Security (PESA)". У: (2017), стр. 1–5. DOI: 10.1109/PESA.2017.8277762.

[11] Sakharova L. и др. „Application of fuzzy set theory in agro-meteorological models for yield estimation based on statistics". У: *Procedia Computer Science* 120 (2017), стр. 820–829.

[12] Jomaa M. и др. „Greenhouse Modeling, Validation and Climate Control based on Fuzzy logic". У: *Engineering, Technology and Applied Science Research* 9.4 (2019), стр. 4405–4410.

[13] Marimin M. и Mushthofa M. „Fuzzy Logic Systems and Applications in Agro-industrial Engineering and Technology". У: *International Journal of Computer Aplications* (2013).

[14] Alpay Ö. и Erdem E. „The Control of Greenhouses Based on Fuzzy Logic Using Wireless Sensor Networks". У: *International Journal of Computational Intelligence Systems* 12.1 (2019), стр. 190–203.

[15] Shamshiri R. и др. „Greenhouse Automation Using Wireless Sensors and IoT Instruments Integrated with Artificial Intelligence". У: (2021). DOI: 10.5772/intechopen.97714.

[16] Sowmiyaa S. и др. „An Insight into Fuzzy Logic Computation Technology and Its Applications in Agriculture and Meteorology". У: *Oriental Journal of Computer Science and Technology* 13.2-3 (2020), стр. 97–101.

[17] Takagi T. и Sugeno M. „Fuzzy identification of systems and its applications to modeling and control". У: *IEEE transactions on systems, man, and cybernetics* (1985), стр. 116–132.

[18] Roseline T.P., Ganesan N. и Tauro C. „A Study of Applications of Fuzzy Logic in Various Domains of Agricultural Sciences". У: *International Journal of Computer Aplications* ICTAC 2015.1 (2015), стр. 15–18.

[19] др. Александар Картељ. *Предавања и материјали са курса Рачунарска интелигенција*. http://poincare.matf.bg.ac.rs/~aleksandar.kartelj/?content=RI. 2020.

Биографија аутора

Борис Карановић је рођен у Београду, 13. априла 1995. године. Основну школу и гимназију природно-математичког смера је завршио у Београду.

Математички факултет (Универзитет у Београду) уписује 2014. године на смеру Рачунарство и математика, а 2017. године се пребацује на смер Информатика. Основне студије је завршио 2020. године, а исте године уписује и мастер студије на смеру Информатика на истом факултету.

Запослен је био од децембра 2020. до децембра 2021. у *A3 Allmänna IT AB* као фулстек програмер (*C#/React*). Од децембра 2021 до данас је запослен у *IVC Evidensia Djursjukvård* као клауд бекенд програмер (*C#/Azure*)